Université de Montréal

Learning and Time: On using memory and curricula for language understanding

par

Caglar Gulcehre

Département de mathématiques et de statistique
Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures
en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en informatique

May 2018

SOMMAIRE

Cette thèse présente quelques-unes des étapes entreprises pour pouvoir un jour résoudre le problème de la compréhension du langage naturel et d'apprentissage de dépendances à long terme, dans le but de développer de meilleurs algorithmes d'intelligence artificielle. Cette thèse est écrite comme une thèse par articles, et contient cinq publications scientifiques. Chacun de ces articles propose un nouveau modèle ou algorithme et démontre leur efficacité sur des problèmes qui impliquent des dépendances à long terme ou la compréhension du langage naturel. Malgré le fait que quelque uns de ces modèles n'ont été testés que sur une seule tâche (comme la traduction automatique neuronale), les méthodes proposées sont généralement applicables dans d'autres domaines et sur d'autres tâches.

Dans l'introduction de la thèse, nous expliquons quelques concepts fondamentaux de l'entraînement de réseaux de neurones appliqués sur des données séquentielles. Tout d'abord, nous présentons succinctement les réseaux de neurones, puis, de façon plus détaillé, certains algorithmes et méthodes utilisés à travers cette thèse.

Dans notre premier article, nous proposons une nouvelle méthode permettant d'utiliser la grande quantité de données monolingue disponible afin d'entraîner des modèles de traduction. Nous avons accompli cela en entraînant d'abord un modèle Long short-term memory (LSTM) sur un large corpus monolingue. Nous lions ensuite la sortie de la couche cachée du modèle avec celle d'un décodeur d'un modèle de traduction automatique. Ce dernier utilise un mécanisme d'attention et est entièrement entraîné par descente de gradient. Nous avons montré que la méthode proposée peut augmenter la performance des modèles de traduction automatique neuronale de façon significative sur les tâches où peu de données multilingues sont disponibles. Notre approche augmente également l'efficacité de l'utilisation des données dans les systèmes de traduction automatique. Nous montrons aussi des améliorations sur les paires de langues suivantes: turc-anglais, allemand-anglais, chinois-anglais et tchèque-anglais.

Dans notre deuxième article, nous proposons une approche pour aborder le problème des mots rares dans plusieurs tâches du traitement des langages. Notre approche modifie l'architecture habituelle des modèles encodeur-décodeur avec attention, en remplaçant la couche softmax du décodeur par notre couche pointer-softmax. Celle-ci permet au décodeur de pointer à différents endroits dans la phrase d'origine. Notre modèle apprend à alterner

entre copier un mot de la phrase d'origine et prédire un mot provenant d'une courte liste de mots prédéfinie, de manière probabiliste. L'approche que nous avons proposée est entièrement entraînable par descente de gradient et n'utilise qu'un objectif de maximum de vraisemblance sur les tâches de traduction. Nous avons aussi montré que le pointer-softmax aide de manière significative aux tâches de traduction et de synthèse de documents.

Dans notre article "Plan, Attend, Generate: Planning for Sequence-to-Sequence Models", nous proposons deux approches pour apprendre l'alignement dans les modèles entraînés sur des séquences. Lorsque la longueur de l'entrée et celle de la sortie sont trop grandes, apprendre les alignements peut être très difficile. La raison est que lorsque le décodeur est trop puissant, il a tendance à ignorer l'alignement des mots pour ne se concentrer que sur le dernier mot de la séquence d'entrée. Nous avons proposé une nouvelle approche, inspirée d'un algorithme d'apprentissage par renforcement, en ajoutant explicitement un mécanisme de planification au décodeur. Ce nouveau mécanisme planifie à l'avance l'alignement pour les k prochaines prédictions. Notre modèle apprend également un plan de correction pour déterminer lorsqu'il est nécessaire de recalculer les alignements. Notre approche peut apprendre de haut niveaux d'abstraction au point de vue temporel et nous montrons que les alignements sont généralement de meilleure qualité. Nous obtenons également des gains de performance significatifs comparativement à notre modèle de référence, malgré le fait que nos modèles ont moins de paramètres et qu'ils aient été entraînés moins longtemps.

Dans notre article "Dynamic Neural Turing Machine with Soft and Hard Addressing Schemes", nous proposons une nouvelle approche pour ajouter de manière explicite un mécanisme de mémoire aux réseaux de neurones. Contrairement aux RNNs conventionnels, la mémoire n'est pas seulement représentée au niveau des activations du réseau, mais également dans une mémoire externe. Notre modèle, D-NTM, utilise un mécanisme d'adressage plus simple que les Neural Turing Machine (NTM) en utilisant des paires clé-valeur. Nous montrons que les modèles disposant de ce nouveau mécanisme peuvent plus efficacement apprendre les dépendances à long terme, en plus de mieux généraliser. Nous obtenons des améliorations sur plusieurs tâches incluant entre autres la réponse aux questions sur bAbI, le raisonnement avec implication, MNIST permuté, ainsi que des tâches synthétiques.

Dans notre article "Noisy Activation Functions", nous proposons une nouvelle fonction d'activation, qui rend les activations stochastiques en leur ajoutant du bruit. Notre motivation dans cet article est d'aborder les problèmes d'optimisation qui surviennent lorsque nous utilisons des fonctions d'activation qui saturent, comme celles généralement utilisées dans les RNNs. Notre approche permet d'utiliser des fonctions d'activation linéaires par morceaux sur les RNNs à porte. Nous montrons des améliorations pour un grand nombre de tâches sans

effectuer de recherche d'hyper paramètres intensive. Nous montrons également que supprimer le bruit dans les fonctions d'activation a un profond impact sur l'optimisation.

SUMMARY

The goal of this thesis is to present some of the small steps taken on the path towards solving natural language understanding and learning long-term dependencies to develop artificial intelligence algorithms that can reason with language. This thesis is written as a thesis by articles and contains five articles. Each article in this thesis proposes a new model or algorithm and demonstrates the efficiency of the proposed approach to solve problems that involve long-term dependencies or require natural language understanding. Although some of the models are tested on a particular task (such as neural machine translation), the proposed methods in this thesis are generally applicable to other domains and tasks (and have been used in the literature).

In the introduction of this thesis, we introduce some of the fundamental concepts behind training sequence models using neural networks. We first provide a brief introduction to neural networks and then dive into details of the some of approaches and algorithms that are used throughout this thesis.

In our first article, we propose a novel method to utilize the abundant amount of available monolingual data for training neural machine translation models. We have accomplished this goal by first training a long short-term memory (LSTM) language model on a large monolingual corpus and then fusing the outputs or the hidden states of the LSTM language model with the decoder of the neural machine translation model. Our neural machine translation model is trained end to end with an attention mechanism. We have shown that our proposed approaches can improve the performance of the neural machine translation models significantly on the rare resource translation tasks and our approach improved the data-efficiency of the end to end neural machine translation systems. We report improvements on Turkish-English (Tr-En), German-English (De-En), Chinese-English (Zh-En) and Czech-English (Cz-En) translation tasks.

In our second paper, we propose an approach to address the problem of rare words for natural language processing tasks. Our approach augments the encoder-decoder architecture with attention model by replacing the final softmax layer with our proposed pointer-softmax layer that creates pointers to the source sentences as the decoder translates. In the case of pointer-softmax, our model learns to switch between copying a word from the source

and predicting a word from a shortlist vocabulary in a probabilistic manner. Our proposed approach is end-to-end trainable with a single maximum likelihood objective of the NMT model. We have also shown that it improves the performance of summarization and the neural machine translation model. We report significant improvements in machine translation and summarization tasks.

In our "Plan, Attend, Generate: Planning for Sequence-to-Sequence Models" paper, we propose two new approaches to learn alignments in a sequence to sequence model. If the input and the source context is very long, learning the alignments for a sequence to sequence model can be difficult. In particular, because when the decoder is a large network, it can learn to ignore the alignments and attend more on the last token of the input sequence. We propose a new approach which is inspired by a hierarchical reinforcement learning algorithm and extend our model with an explicit planning mechanism. The proposed alignment mechanism plans and computes the alignments for the next k tokens in the decoder. Our model also learns a commitment plan to decide when to recompute the alignment matrix. Our proposed approach can learn high-level temporal abstractions, and we show that it qualitatively learns better alignments. We also achieve significant improvements over our baseline despite using smaller models and with less training.

In "Dynamic Neural Turing Machine with Soft and Hard Addressing Schemes," we propose a new approach for augmenting neural networks with an explicit memory mechanism. As opposed to conventional RNNs, the memory is not only represented in the activations of the neural network but in an external memory that can be accessed via the neural network controller. Our model, D-NTM uses a more straightforward memory addressing mechanism than NTM which is achieved by using key-value pairs for each memory cell. We find out that the models augmented with an external memory mechanism can learn tasks that involve long-term dependencies more efficiently and achieve better generalization. We achieve improvements on many tasks including but not limited to episodic question answering on bAbI, reasoning with entailment, permuted MNIST task and synthetic tasks.

In our "Noisy Activation Functions" paper, we propose a novel activation function that makes the activations stochastic by injecting a particular form of noise to them. Our motivation in this paper is to address the optimization problem of commonly used saturating activation functions that are used with the recurrent neural networks. Our approach enables us to use piece-wise linear activation functions on the gated recurrent neural network models. We show improvements in a wide range of tasks without doing any extensive hyperparameter search by a drop-in replacement. We also show that annealing the noise of the activation function can have a profound continuation-like effect on the optimization of the network.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Acknowledgments

Ph.D. had been one of the toughest journey of my life which started in 2012 during one of the fiercest winters of Montreal following the student strikes against the tuition increase which Quebec government was planning to do. In the first few years of my Ph.D., I had hard time getting used to the environment in Montreal and I was feeling lonely. This caused me to have difficult time in connecting with others and at the time I was diagnosed with depression. After my first three years, I have started to love the city and established a very deep connection with both the lab and Montreal. Even now I fondly remember walking at the Parc Mont-Royal in the evenings during the summer and discussing research with my friends at McCarrold's. Definitely, the last five years that I have spent in Montreal will be the most memorable times of my life.

I would like to thank many people who helped me along my path to writing this thesis and helped me on my journey to finish my Ph.D.

I would especially like to thank my thesis advisor, Yoshua Bengio, for taking me under his wing, when I did not know much about neural networks and for running a lab where so many researchers are free to explore creative ideas. I also feel deeply debted to him for supporting and helping me, both financially and emotionally during the difficult periods of my Ph.D. and life.

I would like to deeply thank Kyunghyun Cho whom I learned a lot from in particular about writing papers. I had a very fruitful collaboration with him and always enjoyed getting his opinions on the ideas that I wanted to explore more. He helped me in the beginning of my Ph.D. We would still be training traditional statistical machine translation systems without Cho's enthusiasm on solving Neural Machine Translation. After working with him my attitude towards the way I used to do research has completely changed. I became more practical and pragmatic about research than before, and that made me feel more productive than before.

I would like to present my special thanks to my dear friend Orhan Firat. He helped me get through very difficult times during my Ph.D. and got me beer many times when I really needed. His thoughtful comments on life and research put me on the right track when I felt like I got lost during my Ph.D.

I really appreciate my collaborator Marcin Moczulski for always bringing very interesting and bright insights about the research problems we were dealing with. Marcin's endless enthusiasm on delving deeper into the research ideas and/or papers we were discussing helped me to understand the underlying concepts better.

I would like to thank to all of my co-authors, in particular –Yoshua Bengio, Kyungyhun Cho, Razvan Pascanu, Marcin Moczulski, Sarath Chandar, Sungjin Ahn, Orhan Firat, Julian Serban, Junyoung Chung, Ramesh Nallapati, Dzmitry Bahdanau, Bowen Zhou, Bing Xiang, Bart van Merriënboer, Kelvin Xu, Dzmitry Bahdanau, Yann Dauphin, Francesco Visin, Misha Denil and many others whom helped me to advance during my Ph.D.

I would like to thank Maluuba for awarding me the Maluuba PhD Fellowship in Deep Learning. This fellowship has given me the freedom to spend time on projects focusing on language understanding.

I would like to thank Frédéric Bastien, Pascal Lamblin, Matthieu Germain, Simon LeFrancois for keeping all of the computing and software infrastructure at MILA running smoothly, and helping to get recurrent neural networks running fast in Theano.

I would like to thank Thomas Paine, Brendan Shillingford, Jiwoon Im, Francesco Visin, Sungjin Ahn, Faruk Ahmed and Devon Hjelm for proofreading and providing me feedback on my thesis. I really appreciate Francis Dutil, Edward Greffensttete and many others at Deepmind for helping me writing the French translation of the summary for this thesis.

Finally I would like to thank to my manager at Deepmind, Nando deFreitas for giving me opportunity to work on writing my P.hD thesis while working with his team.

Chapter 1

INTRODUCTION

The focus of this thesis is to develop neural network models that can learn from and understand the vast amount of available textual data created by civilizations over many generations. Machine understanding of the text can potentially enable automation of mundane and laborious services that would typically require human intervention. Designing systems that can understand or comprehend language is an essential step in that direction. However, this can be a challenging problem to solve by only relying on rule-based and symbolic models (Winograd, 1972) that are defined by a fixed set of rules and heuristics. The number of rules required to design a system that can understand language even at the level of a three-year-old toddler can be great (Lenat, Prakash, and Shepherd, 1985). Designing a domain-specific symbolic system for most language understanding tasks would also require extensive expert knowledge. The design of rule-based systems for reasoning and language generation with good generalization is a challenging problem. One of the reasons for this is due to the underlying mechanisms behind the computational and cognitive nature of language remain a mystery even with the recent advancements in theoretical linguistics. Several traditional computational linguistic models are built based on linguistic studies in different languages (Steedman and Baldridge, 2011; Joshi and Schabes, 1997; Chomsky, 1956). However, those models are either too general in a way they cover grammars and languages that are outside the realm of human languages or very specific such that they are only valid for a small subset of human languages (Bozsahin, 2012). On the other hand, a deep learning approach can offer a different perspective to this problem, letting the model analyze the data and come up with its model of the language based on the examples that it has seen. Although the representations learned by a neural network can be a black box, they would still perform sufficiently well on the NLP task that they are trained for, and sometimes the representations learned from a neural network for a particular task can be used for other tasks as well (Radford, Narasimhan, Salimans, and Sutskever, 2018). In this section, we will give a brief introduction to neural networks and an overview of the rest of the thesis.

Neural networks have been widely adopted on different domains of science and industrial applications with a notable success. In particular, great improvements in different areas of AI research, such as image recognition (Szegedy, Ioffe, Vanhoucke, and Alemi, 2016; Simonyan and Zisserman, 2014), segmentation (Dai, He, and Sun, 2015), speech recognition (Amodei, Anubhai, Battenberg, Case, Casper, Catanzaro, Chen, Chrzanowski, Coates, Diamos, *et al.*, 2015), machine translation (Wu, Schuster, Chen, Le, Norouzi, Macherey, Krikun, Cao, Gao, Macherey, *et al.*, 2016), machine reading comprehension (Hermann, Kocisky, Grefenstette, Espeholt, Kay, Suleyman, and Blunsom, 2015; Rajpurkar, Zhang, Lopyrev, and Liang, 2016), and text summarization (Nallapati, Zhou, Gulcehre, Xiang, *et al.*, 2016a) resulted in it becoming the *de facto* choice for machine learning applications with the both availability of new models and datasets. Today, those results are considered as a breakthrough and have completely altered the way researchers envision the course of different scientific fields and domains. Some of the factors that lie behind recent successes of deep learning are due to the availability of the large datasets, improved and more scalable learning algorithms, new methods that make the optimization of neural networks easier and availability of new hardware architectures for parallel *basic linear algebra subprograms* (BLAS) and matrix operations, examples of this are *graphics processing units* (GPUs) and *field-programmable gate arrays* (FPGAs).

Statistical NLP approaches such as n-gram language models (Damerau, 1971; Manning, Manning, and Schütze, 1999), and IBM translation models (Brown, Pietra, Pietra, and Mercer, 1993) have laid the foundations of the statistical approaches for NLP that are carried forward with the recent deep learning approaches. The heuristic-based and symbolic approaches to dialogue (Winograd, 1972) also inspired developing novel techniques for dialogue understanding with deep learning methods. Moreover, recently we have been seeing more examples of marriages between the symbolic and the connectionist approaches (Johnson, Hariharan, van der Maaten, Hoffman, Fei-Fei, Zitnick, and Girshick, 2017; Andreas, Rohrbach, Darrell, and Klein, 2016).

Learning features and representations have become essential for all machine learning (ML) applications, and here we define some important concepts in this context which we discuss throughout this thesis:

Feature engineering is the process of manually finding features to transform the model's inputs in a way that the new abstract/high-level representation of the inputs would make the learning of the task easier. Those features are not learned, but they are mainly hand-engineered or designed for the task of interest by using prior knowledge about that particular task and data. Some well-known examples of feature engineering algorithms are approaches

such as SIFT (Lowe, 1999), HOG features (Zhu, Yeh, Cheng, and Avidan, 2006) for image recognition, TF-IDF (Sparck Jones, 1972) or bag of n-grams for learning from documents, and MFCC (Mermelstein, 1976) for speech recognition. These features are still being used in industrial applications of large scale information processing pipelines, usually in addition to learned features.

Representation Learning, in the general sense, enables us to build model that can extract useful features for a task by learning transformations on the raw input data. The main motivation for *representation learning* is that the engineering of features for each different task can be difficult and time-consuming. Moreover, hand-engineered features do not necessarily have to be composable unless they are engineered to be so and they do not have to be transferable across different tasks. A leading motivation for the research on representation learning is that it is possible to learn representations that can disentangle the factors of variation in the input data. The ability to disentangle factors of variation in an unsupervised way would achieve more data-efficient algorithms. Potentially, it can also help the model to learn a class of functions that are less variant to changes in the environment and have better generalization capabilities.

Deep Learning is based on the idea of learning layers of abstractions and compositional features. The composition of functions can be obtained by stacking multiple layers and adding non-linearities, which makes the input to output mapping to become highly nonlinear and *deep*. Deep learning techniques usually use neural networks to model non-linear input to output mappings due to the expressivity of neural networks. However, restricting deep learning algorithms just to neural networks would be a conservative view. Ideally, each layer can learn features that could be useful for the task and as one goes deeper in the hierarchy, the representations can provide higher-level and a more abstract summary of the input. Before the recent resurgence of deep learning algorithms, the field of machine learning mainly focused on learning (usually linear) simple local mappings (e.g. by kernels) over the feature-engineered data. One driving force that enabled the deep learning revolution has been representation learning (Bengio *et al.*, 2009a; Bengio, Courville, and Vincent, 2013c). The representations obtained by deep learning algorithms can also be used to learn features that can be composed with other features learned on a different dataset.

The rest of the thesis is organized as follows:

In Chapter 2, we provide a simple overview of deep learning methods for sequence processing. We both discuss the models for sequence modeling and how to train or optimize those models. We give a brief overview of recurrent models and memory models.

In Chapter 3, we give a high-level overview of the deep learning methods for NLP. We provide a brief overview of basic deep learning models for various NLP applications such as machine translation and summarization.

In Chapter 5 we present our work on how to incorporate monolingual language models to the sequence to sequence models with attention. We consider both shallow and deep fusion of the language model. Incorporating the language model improves the performance significantly on low-resource language pairs.

In Chapter 7, we introduce a mechanism to deal with the rare words. Our proposed model probabilistically selects between whether to copy the word from the input context or predict it by using a shortlist softmax. We have shown promising results on text summarization and machine translation by using the sequence to sequence models.

In Chapter 9, we develop a new memory-based sequential model inspired by Neural Turing Machines. We introduce several modifications on the original model to simplify the read and write mechanisms and perform an extensive ablation study.

In Chapter 11, we show that a model that incorporates a planning mechanism in its attention can help to overcome the issues of learning attention over very long sequences. We developed two variations of the planning for the attention mechanism and showed that they both improve over the baselines on neural machine translation and question generation.

In Chapter 13, we introduce noisy activation functions which inject noise into the activation function to avoid the gradients to get stuck in a particular regime of the saturation. We were able to train recurrent neural networks with piecewise linear activation function by using noisy activation functions.

In Chapter 14, we conclude with the open problems in NLP.

Chapter 2

DEEP LEARNING ALGORITHMS FOR MODELING SEQUENCES

2.1. A Primer on Neural Networks

The general notion of artificial neural networks is inspired from its biological counterparts where the neurons wire together via dendrites creating synapses with other neurons to form new memories and learn new tasks. In that sense, the neural networks that are used for machine learning can be considered to be a cartoon view of the biological neural networks and there are crucial differences between the two. For example, the synaptic strengths are fixed at the end of the learning phase in the artificial neural networks and in general, we do not have the concept of firing rate with traditional artificial neural networks. In the rest of this thesis, when we refer to neural networks what we mean is the artificial neural network.

2.1.1. Neural Networks

In this section, we will introduce multi-layer perceptrons (MLP) and discuss how inference happens in neural networks. Neural networks are very general parameterized function approximators which define a highly nonlinear mapping between the inputs and the outputs for a given task. The basic idea of neural networks can be traced back to perceptrons which were proposed by Rosenblatt (1958).

A typical deep neural network can be constructed by applying layers of affine-transformations parameterized by weight matrices $\mathbf{W}^i \in \mathbb{R}^{d^i_{in} \times d^i_{out}}$ and the biases $\mathbf{b}^i \in \mathbb{R}^{d^i_{out}}$ for every i^{th} layer. In this formalism, $\mathbf{h}^0 \in \mathbb{R}^{d^0_x}$ corresponds to the input layer ($\mathbf{x}$). For a deep feedforward neural network, the activations of every $l^{th} > 0$ layer with nonlinearity $f(\cdot)$ can be computed by using the recursive function in Equation (2.1.1) below. In Figure 2.3, we demonstrate how different layers can be stacked together in a deep feedforward neural network.

$$\mathbf{h}^l = f(\mathbf{W}^l \mathbf{h}^{l-1} + \mathbf{b}^l) \tag{2.1.1}$$

Each unit $\mathbf{h}_i$ in a layer can be thought of as a small computational unit that first computes the affine projection over its input to compute the pre-activation $\mathbf{z}_i$ as in Equation (2.1.2) for a layer with m input features. The affine transformation as the pre-activation computes a hyperplane and an element-wise nonlinearity applied on the hyperplane turns it into a nonlinear decision surface. As a result, non-linear units can partition their input space via nonlinear decision boundaries as in Figure 2.1.

$$\mathbf{z}_i = \sum_{j=0}^{m} \mathbf{W}_{ij}\mathbf{x}_j + \mathbf{b}_j \tag{2.1.2}$$

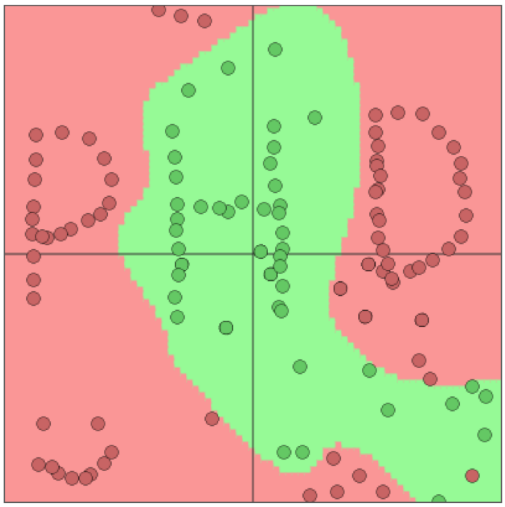


FIG. 2.1. An illustration of a neural network partitioning the input space of data consists of two different categories green and red into nonlinear decision boundaries. These boundaries are obtained with a neural network of two layers and six hidden units at each layer.[1]

Thus each i^{th} unit of $\mathbf{h}^l$ would compute a weighted sum of its own input and the offset via the biases as seen in Figure 2.2. In Equation (2.1.3), we show the computation of the activations of the units in the layer $\mathbf{h}^l$.

$$\begin{pmatrix} \mathbf{h}_0^l \\ \vdots \\ \mathbf{h}_n^l \end{pmatrix} = \begin{pmatrix} f(\sum_{j=0}^{m} \mathbf{W}_{0j}^l \mathbf{h}_j^{l-1} + \mathbf{b}_j^l) \\ \vdots \\ f(\sum_{j=0}^{m} \mathbf{W}_{nj}^i \mathbf{h}_j^{l-1} + \mathbf{b}_j^l) \end{pmatrix} \tag{2.1.3}$$

[1]This figure is obtained with Andrej Karpathy's excellent webpage for the ConvnetJS library `https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html`.

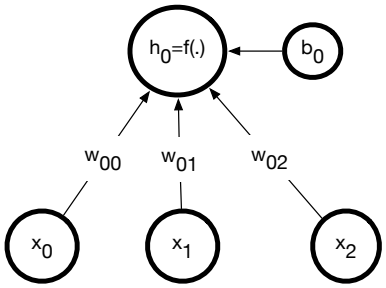


FIG. 2.2. A depiction of how an activation of neural network is computed for the unit $\mathbf{h}_0$ where the inputs are depicted as $\mathbf{x}$. Thus the pre-activations of a unit is basically the sum of its inputs weighted by $\mathbf{W}_0$ weights coming into the unit 0 and the bias $\mathbf{b}_0$ added. The activations of the unit are computed via the application of the activation function $f(\cdot)$ on the pre-activations.

The high-level representation of a particular observation would emerge as a result of the patterns of activations in the layers of a neural network. For every different concept recognized by a neural network, a group of neurons that are specialized for a particular feature of that concept would respond more strongly than the others. However, in a neural representation, the neurons in each group can activate to multiple concepts as opposed to a local representation. In that sense, if two concepts are close to each other, their representations in the ambient space will be close as well. This type of representation is called *distributed representation* (Hinton, Rumelhart, and McClelland, 1984) and recently it has been a focus of great deal of works (Mikolov, Sutskever, Chen, Corrado, and Dean, 2013; Pascanu, Montufar, and Bengio, 2013c; Montufar, Pascanu, Cho, and Bengio, 2014). As opposed to local representations, the distributed representation can be more efficient in terms of parameter use and can learn representations that can generalize better than local representations (Bengio, LeCun, *et al.*, 2007).

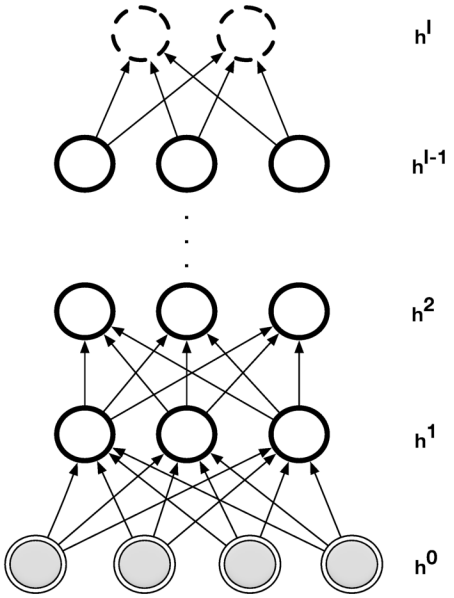


FIG. 2.3. Computational graph of a feed-forward deep neural network. $\mathbf{h}^0$ is the input layer $\mathbf{x}$ and the final layer $\mathbf{h}_l$ denotes the output layer (softmax, sigmoid or linear), for example in the case of a classification task it can be a logistic regression layer.

Cybenko (1989) has shown that a single layered neural network with a finite number of neurons and sigmoid activation function can approximate any continuous function. More formally the universal approximation theorem can be stated as shown by (Cybenko, 1989),

Definition 2.1.1. Universal Function Approximation Theorem: *Let $\sigma(\cdot)$ be a non-constant, monotonically increasing, bounded activation function. For any $f \in C([0,1]^d)$ and $\epsilon > 0$, there exists $n \in \mathbb{N}$ number of units with $v_i, b_i \in R$ real constants and vectors $\mathbf{w}_i \in \mathbb{R}^d$, $\mathbf{x} \in \mathbb{R}^d$ such that:*

$$|f(x) - \sum_{i}^{n} v_i \sigma(\mathbf{w}_i^\top x + b_i)| < \epsilon$$

In parallel to Cybenko et. al.'s work, Hornik, Stinchcombe, and White (1989) have shown that this inequality also holds for multi-layer perceptron with any smooth bounded activation function. Sonoda and Murata (2015) have shown the any multi-layer perceptron with unbounded activation function is also a universal approximator.

Recently there have been several attempts to design different activation functions for neural networks, mainly to make the training of neural networks easier and improve generalization. It

seems like the choice of the activation function highly depends on the task and the architecture that the model is being trained on. In Table 2.1, we have shown the most popular ones along with their gradients.

TAB. 2.1. A zoo of common activations functions and their derivatives used with neural networks.

	function	derivative
sigmoid(x)	$\frac{1}{1+\exp(-x)}$	$\text{sigmoid}(x)(1-\text{sigmoid}(x))$
tanh(x)	$\frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$	$1-\tanh^2(x)$
rectifier(x)	$\max(x,\ 0)$	$\begin{cases}1 & if\ x\ >\ 0\\ 0 & \text{otherwise}\end{cases}$
softmax(x)	$\frac{\exp(x_i)}{\sum_{j=0}^{N}\exp(x_j)}$	$\text{softmax}(x)(1-\text{softmax}(x))$
softplus(x)	$\log(1\ +\ \exp(x))$	sigmoid(x)

2.1.2. Backpropagation

Backpropagation is the most common way to obtain parameter gradients to train feedforward neural networks. The first order gradients for each parameter in the model is obtained by using the chain rule of differentiation. This principle is very intuitive and has already been used in different disciplines of science (Werbos, 1994) to be able to compute the gradients of the output of a multivariate function that is a composition of different functions. Rumelhart, Hinton, and Williams (1986b)'s work is widely considered as the most important work which introduced this algorithm to the neural networks with practical applications and intuitive explanations.

The backpropagation algorithm has two phases. First, the forward-*propagation* of the signal from input to compute the activations of the layers and the loss function. Once the loss function is computed, we perform backward-*propagation* to compute the gradients using the chain rule and update the parameters. Consider a loss function $\mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta})$ which measures the discrepancy between the input $\mathbf{x}$, target $\mathbf{y}$ and a neural network with l layers and the parameters $\boldsymbol{\theta}$. The gradient with respect to the j^{th} layer's weights $\mathbf{W}^j$ can be written as,

$$\frac{\partial \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta})}{\partial \mathbf{W}^j} = \frac{\partial \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta})}{\partial \mathbf{h}_l} \frac{\partial \mathbf{h}^l}{\partial \mathbf{h}^i} \frac{\partial \mathbf{h}^i}{\partial \mathbf{W}^j}, \tag{2.1.4}$$

$\frac{\partial \mathbf{h}^l}{\partial \mathbf{h}^i}$ is also called the Jacobian Matrix and it can be further decomposed:

$$\frac{\partial \mathbf{h}^l}{\partial \mathbf{h}^i} = \prod_{i \leq j < l} \frac{\partial \mathbf{h}^j}{\partial \mathbf{h}^{j-1}}. \tag{2.1.5}$$

It is possible to reuse the Jacobians when we compute the gradients for each parameter. In this way the computational complexity of the backpropagation would be $O(|\boldsymbol{\theta}|)$ where $|\boldsymbol{\theta}|$ stands for the number of the parameters of the neural network. On the other hand, computing the gradients for the backpropagation with finite differences would cost $O(|\boldsymbol{\theta}|^2)$. Backpropagation is cheaper than finite difference, because when the loss function is just a scalar, the gradients for all parameters can be computed simultaneously and that ends up becoming a sequence of matrix vector products.

2.2. Sequential Modeling with Neural Networks

2.2.1. Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are another family of neural networks that can process sequences of input tokens in a sequential manner. The main idea behind recurrent neural networks is to extend the feedforward neural networks to sequences with variable lengths. The predecessors of the modern day recurrent neural networks stems from Jordan Nets (Jordan, 1986) and Elman Networks (Elman, 1990).

RNNs can learn to model the dependencies in the input sequence by recursively applying the same function with shared parameters along the direction where the function is being applied over. This direction can be either temporal (such as in speech, NLP, videos, ...etc) or in the spatial axis of the input sequence. The state of the recurrent neural network can learn to keep dependencies across an arbitrary, unspecified amount of time.

The recurrent function to compute the hidden state $\mathbf{h}_t$ of the recurrent neural network at the timestep t for an input sequence $\mathbf{x} = (\mathbf{x}_0, \cdots, \mathbf{x}_t, \cdots, \mathbf{x}_T)$ with a sequence of length T can be written as in Equation 2.2.1.

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t) \tag{2.2.1}$$

It is possible to stack the recurrent layers, as with feedforward neural networks. The "last layer" of the RNN can also be called as the output or prediction layer predicting the target $\mathbf{y}_t$ at every timestep t. A full parameterization of the deep stacked recurrent neural network can be represented with the parameters $\mathbf{W}_h \in \mathbb{R}^{d_h \times d_h}, \mathbf{W}_x \in \mathbb{R}^{d_h \times d_x}, \mathbf{W}_o \in \mathbb{R}^{d_o \times d_h}, \mathbf{b}_h \in \mathbb{R}^{d_h}, \mathbf{b}^o \in \mathbb{R}^{d_o}$, the activations of the hidden state being $\phi(\cdot)$ and the output function being $g(\cdot)$ for a network with $l+1$ layers (l recurrent and the input layers, $+1$ for the output layer)

which can be a softmax, linear or sigmoid activation function depending on the application. Let $\mathbf{h}_0$ be the input sequence $\mathbf{x}$. The parametrization of the model is hence,

$$\begin{aligned} \mathbf{h}_t^l &= \phi(\mathbf{W}_x \mathbf{h}_t^{l-1} + \mathbf{W}_h \mathbf{h}_{t-1}^l + \mathbf{b}_h^l) \\ \mathbf{o}_t &= g(\mathbf{W}_o \mathbf{h}_t^l + \mathbf{b}_o) \end{aligned} \tag{2.2.2}$$

We illustrate the flow of information across the sequence and through depth in Figure 2.4.

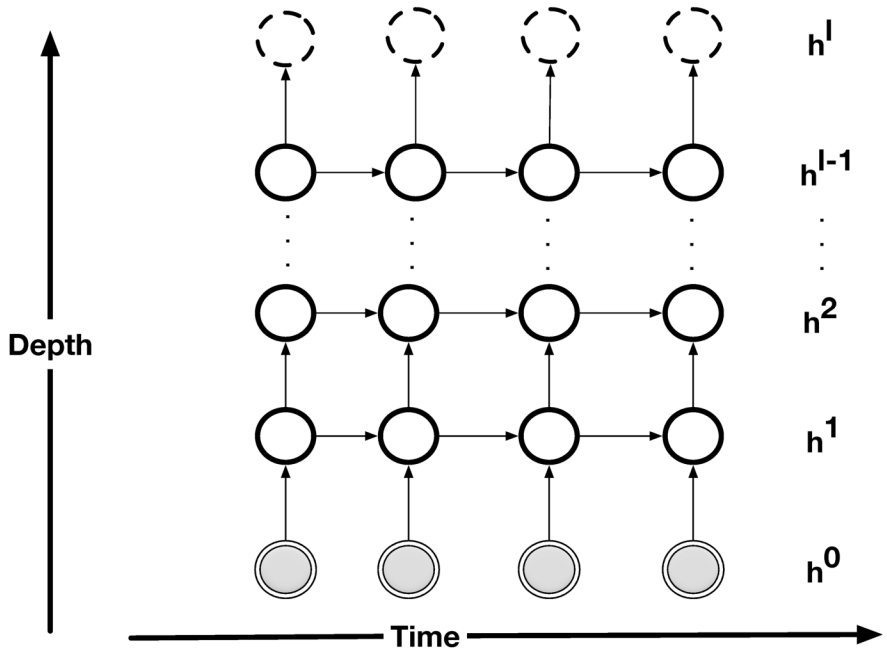


FIG. 2.4. An illustration of a deep recurrent neural network. We show both the visualization of the flow information through time and from input to output as well.

The hidden state of the recurrent neural network $\mathbf{h}_t$ represents the compressed history that has been seen in the input sequence. The loss function $\mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta})$ which measures the discrepancy between the predictions of the model and desired outputs for a model that predicts at every timestep can be written as,

$$\mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}) = \sum_{t=1}^{T} \mathcal{L}(\mathbf{x}_t, \mathbf{y}_t; \boldsymbol{\theta}) \tag{2.2.3}$$

2.2.2. Bidirectional RNNs

As opposed to regular RNNs that process the inputs in the sequence uni-directionally (typically left to right), bidirectional recurrent neural networks (Schuster and Paliwal, 1997) compute the representation of the sequence by processing it both from left to right and right to left.

In a bidirectional recurrent neural network, both the future and the past inputs are accessible from every state. For example, in NLP applications a word can have linguistic dependencies both to the tokens that appear before and the tokens that appear after. In that sense, bidirectional RNNs are more powerful in terms of capturing complicated syntactic dependencies.

As depicted in Figure 2.5, a bidirectional RNN has two states that scan the input in opposite directions. $\overleftarrow{\mathbf{h}_t}$ is the backward state that reads the input from right to left and the $\overrightarrow{\mathbf{h}_t}$ that reads the input from left to right. The hidden state of the bidirectional RNN $\mathbf{h}_t$ is the concatenation of the forward and the backward states, $\mathbf{h}_t = [\overleftarrow{\mathbf{h}_t}; \overrightarrow{\mathbf{h}_t}]$.

Bidirectional RNNs have found a great number of applications in neural machine translation (Bahdanau *et al.*, 2014), abstractive summarization (Nallapati *et al.*, 2016a), speech recognition (Graves, Jaitly, and Mohamed, 2013) and offline hand-writing recognition (Graves and Schmidhuber, 2009).

2.3. Backpropagation Through Time (BPTT)

The backpropagation through time algorithm was first discussed in (Rumelhart, Hinton, and Williams, 1986a) and later on further elaborated by Werbos (1990). BPTT is an extension of the backpropagation algorithm for the RNNs unrolled over the input sequences of variable length. BPTT is the most popular way to train RNNs.

Similar to the backpropagation algorithm, BPTT also has two phases, forward propagation, and backward propagation. During forward propagation, the model is unrolled over the input sequence and during backward propagation, the gradients of the model is computed over the unrolled graph. The gradient of the output layers of an RNN can be written easily as,

$$\frac{\partial \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta})}{\partial \mathbf{W}_o} = \sum_{1 \leq t \leq T} \frac{\partial \mathcal{L}(\mathbf{x}_t, \mathbf{y}_t; \boldsymbol{\theta})}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{W}_o}, \tag{2.3.1}$$

The challenging aspect of the BPTT algorithm is to compute the gradients with respect to the recurrent weights, mainly because at each time-step the graph needs to be unrolled until the beginning of the sequence and the gradients need to be computed for every time-step. Let

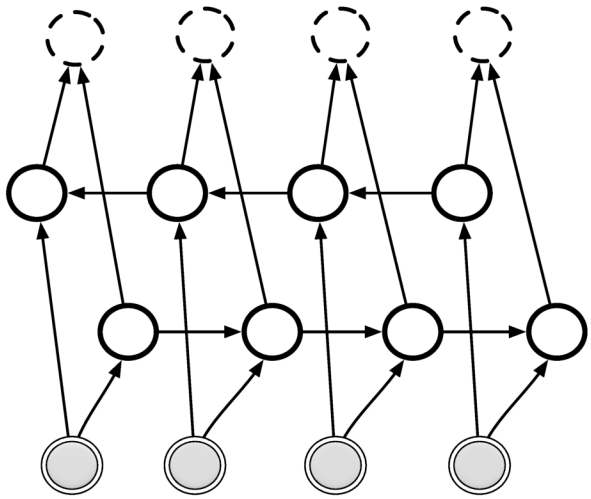


FIG. 2.5. A depiction of a bidirectional RNN. There are two different RNNs running over the input. The first RNN scans the input from left to right(forward) and the second RNN scans the input tokens from right to left (backward). Before the prediction layer, we concatenate the hidden states of the both the forward and the backward RNNs.

us first define $\frac{\partial \mathbf{h}_t^*}{\partial \mathbf{W}_h}$ to be the *immediate gradient* as in (Pascanu, Gulcehre, Cho, and Bengio, 2013a), such that in the case of immediate gradients for all t^- less than t, we consider the gradients of $\mathbf{h}_{t^-}$ with respect to $\mathbf{W}_h$ to be 0. We can write the gradients of the loss function $\mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta})$ with respect to the recurrent weights as in Equation 2.3.2.

$$\frac{\partial \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta})}{\partial \mathbf{W}_h} = \sum_{1 \leq t \leq T} \frac{\partial \mathcal{L}(\mathbf{x}_t, \mathbf{y}_t; \boldsymbol{\theta})}{\partial \mathbf{h}_t} \sum_{1 \leq k \leq t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k^*}{\partial \mathbf{W}_h}. \tag{2.3.2}$$

Further we can rewrite the Jacobian matrix $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k}$ as the product of the jacobians at each timestep,

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \prod_{i=k}^{t-1} \frac{\partial \mathbf{h}_{i+1}}{\partial \mathbf{h}_i} \tag{2.3.3}$$

We provide the pseudocode for the BPTT algorithm of an RNN in Algorithm 1. It is possible to compute the gradients for this model with an arbitrary computational graph by using reverse automatic differentiation (Baydin, Pearlmutter, Radul, and Siskind, 2015). This

would eliminate the risk of introducing bugs while computing the gradients manually. The computational complexity of that algorithm would be $O(T)$ for a sequence of length T and the memory space complexity would be $O(d_h T)$ for a model with d_h units.

Algorithm 1 The general back-propagation through time algorithm for a single-layer RNN. $\mathcal{L}_t$ refers to the loss at time-step t. The $(^*)$ in gradients $(\partial \mathbf{h}_t^* / \partial \mathbf{W}_h)$ refers to the immediate derivative.

procedure BPTT
  for t in $1 \dots T$ **do** ▷ First do Forward propagation.
    $\mathbf{h}_t \leftarrow \phi(\mathbf{W}_x \mathbf{h}_t^{l-1} + \mathbf{W}_h \mathbf{h}_{t-1}^l + \mathbf{b}_h^l)$
    $\mathbf{o}_t \leftarrow g(\mathbf{W}_o \mathbf{h}_t^l + \mathbf{b}_o)$
  end for
  $\mathbf{db}_o \leftarrow 0$ ▷ Initialize accumulated gradients.
  $\mathbf{dW}_o \leftarrow 0$
  $\mathbf{dW}_h \leftarrow 0$
  $\mathbf{dW}_x \leftarrow 0$
  $\mathbf{db}_h \leftarrow 0$
  $\mathbf{dh}_{T+1} \leftarrow 1$
  for t in $T \dots 1$ **do** ▷ Do backward propagation.
    $\mathbf{dW}_o \leftarrow \partial \mathcal{L}_t / \partial \mathbf{W}_o$
    $\mathbf{db}_o \leftarrow \partial \mathcal{L}_t / \partial \mathbf{b}_o$
    $\mathbf{dh}_t \leftarrow \mathbf{dh}_{t+1} (\partial \mathbf{h}_t / \partial \mathbf{h}_{t-1}) + (\partial \mathcal{L}_t / \partial \mathbf{h}_t)$
    $\mathbf{dW}_h \leftarrow \mathbf{dW}_h + \mathbf{dh}_t (\partial \mathbf{h}_t^* / \partial \mathbf{W}_h)$
    $\mathbf{dW}_x \leftarrow \mathbf{dW}_x + \mathbf{dh}_t (\partial \mathbf{h}_t^* / \partial \mathbf{W}_x)$
    $\mathbf{db}_h \leftarrow \mathbf{db}_h + \mathbf{dh}_t (\partial \mathbf{h}_t^* / \partial \mathbf{b}_h)$
  end for
  $\boldsymbol{\theta} \leftarrow [\mathbf{db}_o, \mathbf{dW}_o, \mathbf{dW}_h, \mathbf{dW}_x, \mathbf{db}_h]$
  return $\boldsymbol{\theta}$
end procedure

2.3.1. Scaling BPTT

It can be difficult to scale RNNs on very long sequences when training with BPTT due to the large memory consumption because storing activations of all intermediate nodes would grow linearly in the length of the sequences. Furthermore, as the length of the sequence grows the number of timesteps that the model would need to backpropagate through time would grow as well. Thus the computational complexity of such an approach would grow linearly with respect to the length of the sequence. These issues can become more pronounced when training models over very long documents or sequences of genomes.

A practical way to deal with the memory complexity is to tradeoff memory consumption with computation. **Checkpointing** (Griewank, 1992) deals with this issue by dividing the sequence into subsequences of length k and it is possible to just store the activations of every k timesteps as a checkpoint in the forward propagation. During the backpropagation through time, it is possible to reconstruct the activations in a subsequence (instead of storing activations at every timestep) by storing only the activations of every k time-steps and the activations in the subsequence of interest. One can recompute the activations in a subsequence by just doing a forward propagation from the closest checkpoint and update the parameters of the model in this subsequence. The memory complexity of BPTT with checkpointing would be $O(T/k + k)$, and if k is chosen to be $\sqrt{T}$ as proposed by (Chen, Xu, Zhang, and Guestrin, 2016b), it can be $O(\sqrt{T})$.

A well-known approach to deal with the issue regarding the computational complexity of BPTT is to approximate the gradients by only backpropagating errors at every k_1 timesteps and for k_2 timesteps through time. This approach is usually called **truncated backpropagation through time** (TBTT) (Williams and Peng, 1990). If k_1 is large enough and k_2 is small, this method can be more efficient than BPTT. For $\mathcal{A} = (k_1,\ 2k_1,\ 3k_1,\ 4k_1,\ \ldots,\ T)$, or in other words for $\mathcal{A}$ is the sequence of indices that corresponds to every k_1^{th} in the input sequence. TBPTT will use an approximation to the BPTT gradients as shown in Equation 2.3.4 and the residual error of this approximation ϵ is shown in Equation 2.3.5. As k_1 increases and k_2 the number of timesteps that will be included in TBTT become smaller, the approximation error will increase as well. However, if k_1 is small and k_2 is large, TBPTT would not be faster than BPTT but it would consume less memory $O(k_2)$ rather than $O(T)$. Let us note that, even the asymptotic complexity of the TBPTT is the same as BPTT algorithm that we have provided in Algorithm 1, $O(T)$, but a linear speedup can be observed by choosing $k_1 > k_2$. For example $k_1 = 2$ and $k_2 = 1$, one would only need propagate the errors for only a single timestep.

$$\frac{\partial \mathcal{L}(\mathbf{x}_t, \mathbf{y}_t; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \approx \sum_{t \in \mathcal{A}} \frac{\partial \mathcal{L}(\mathbf{x}_t, \mathbf{y}_t; \boldsymbol{\theta})}{\partial \mathbf{h}_t} \sum_{t-k_2 \leq j \leq t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_j} \frac{\partial \mathbf{h}_j^*}{\partial \mathbf{W}_h}, \tag{2.3.4}$$

$$\epsilon = \left| \sum_{t \in A} \frac{\partial \mathcal{L}(\mathbf{x}_t, \mathbf{y}_t; \boldsymbol{\theta})}{\partial \mathbf{h}_t} \sum_{1 \leq j \leq t-k_2} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_j} \frac{\partial \mathbf{h}_j^*}{\partial \mathbf{W}_h} + \sum_{t \notin A} \frac{\partial \mathcal{L}(\mathbf{x}_t, \mathbf{y}_t; \boldsymbol{\theta})}{\partial \mathbf{h}_t} \sum_{1 \leq j \leq t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_j} \frac{\partial \mathbf{h}_j^*}{\partial \mathbf{W}_h} \right|. \tag{2.3.5}$$

2.3.2. The Problems of BPTT

There are some very well-known problems that can make the training of the RNNs with BPTT to be difficult. The difficulty of the training arises from the repeated application of the same function with tied weights through time. Because the iterative application of the same function results into a very highly nonlinear dynamical system and a small change in the inputs can put the system into a chaotic state.

Vanishing Gradients

Bengio, Simard, and Frasconi (1994) have first shown that the RNNs can suffer from a difficulty in learning long-term dependencies due to the gradients vanishing through time.

The reason for the gradients vanishing through time is due to repeated dot products between the Jacobians $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$ at every timestep as in Equation 2.3.6. For example, If the activation function is sigmoid, the derivative of the sigmoid will be upper-bounded by 1/4. Thus the Jacobian $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k}$ would be upper bounded by $(\mathbf{W}_h)^{t-k}(1/4)^{t-k}$. As a result and as noted in (Pascanu *et al.*, 2013a), if the largest singular value of the recurrent weights $\mathbf{W}_h$ is smaller than 4 for sigmoids, the gradients will vanish through time.

Bengio *et al.* (1994) demonstrated that in order for the network to store information in a robust and stable way over a long duration, the eigenvalues of the Jacobians through time have to be less than 1, and as a consequence we are in the regime where gradients vanish over time, meaning that long-term effects get an exponentially smaller weight in the total gradient than the short-term effect. We have also detailed this issue and discussed in the context of memory networks in (Gulcehre, Chandar, and Bengio, 2017a).

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = \mathbf{W}_h \operatorname{diag}\left(\phi'(\mathbf{W}_x \mathbf{x}_t + \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{b}_h)\right) \tag{2.3.6}$$

Exploding Gradients

Another potential problem that can happen due to the repeated multiplication of the Jacobians is that the gradients can explode. Exploding gradients can happen if the largest singular value of the recurrent weights is very large. A typical method to overcome this issue is to use gradient clipping (Pascanu *et al.*, 2013a). If the norm/magnitude of the gradients exceeds a certain threshold, the gradients are renormalized such that the L_2 norm of the gradients would be less than the threshold.

2.3.3. Long Short Term Memory (LSTM)

As we discussed in earlier sections, vanilla RNNs can be dramatically affected by the training problems arising due to vanishing gradients and exploding gradients when trained with BPTT. LSTM (Hochreiter and Schmidhuber, 1997) designed to address those issues regarding learning long-range dependencies. LSTMs have been shown to be very successful in different applications such as NLP (Collobert, Weston, Bottou, Karlen, Kavukcuoglu, and Kuksa, 2011; Sutskever, Vinyals, and Le, 2014; Wu *et al.*, 2016), language modeling (Graves *et al.*, 2013), computer vision (He, Zhang, Ren, and Sun, 2016; Szegedy *et al.*, 2016) and speech recognition (Hannun, Case, Casper, Catanzaro, Diamos, Elsen, Prenger, Satheesh, Sengupta, Coates, *et al.*, 2014).

LSTMs can learn to carry information from the past for arbitrarily long timesteps and they can learn to ignore particular input tokens or reset the state of the hidden state by using gates that interact with the memory of the LSTM if it is no longer relevant to the context. This behavior is learned end to end in a differentiable manner by using parameterized functions. An LSTM unit has three gates, $\mathbf{i}_t$, $\mathbf{o}_t$ and $\mathbf{f}_t$, respectively for the input, output and the forget gates. We consider the parameterization of the gates as proposed in (Zaremba *et al.*, 2014): the activation of every gate is affected by the inputs coming from the current time-step and the hidden state ($\mathbf{h}_t$) of the LSTM. As shown in Equation 2.3.7, the activation function of the gates is sigmoid just to ensure that the gates can learn a binary switching behavior to block or pass the information flowing through the cell. $\mathbf{W}_h \in \mathbb{R}^{3d_h \times d_h}$ is the parameter matrix applied on the recurrent states, $\mathbf{W}_x \in \mathbb{R}^{3d_h \times d_x}$ is the parameter matrix applied on the input for the $\mathbf{h}_t \in \mathbb{R}^{d_h}$ and $\mathbf{x}_t \in \mathbb{R}^{d_x}$.

$$\begin{pmatrix} \mathrm{f}_t \\ \mathrm{i}_t \\ \mathrm{o}_t \end{pmatrix} = \texttt{sigm}\left(\mathbf{W}_h \mathbf{h}_{t-1} \ + \ \mathbf{W}_x \mathbf{x}_t \ + \ \mathbf{b}_h\right), \tag{2.3.7}$$

In Equation 2.3.8, we have also shown that the activations of the candidate cell are also a function of the input and the hidden state of the LSTM, with `tanh` activation function. The forget gate can introduce identity connections through time across the previous states of the cell. This can mitigate the problems related to vanishing gradients. The input gate can learn to ignore the current input, if it is not relevant for example if it is noise. Output gate is mainly for dealing with exploding gradients. We have provided the computational graph of an LSTM unit in Figure 2.6.

$$\mathbf{ç}_t = \texttt{tanh}(\mathbf{W}_h^c \mathbf{h}_{t-1} \ + \ \mathbf{W}_x^c \mathbf{x}_t \ + \ \mathbf{b}_c) \tag{2.3.8}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{ç}_t \tag{2.3.9}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \texttt{tanh}(\mathbf{c}_t) \tag{2.3.10}$$

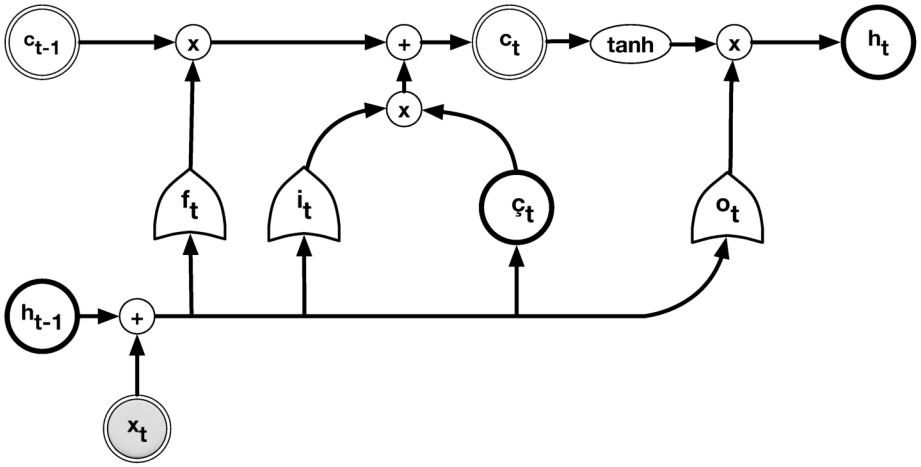


FIG. 2.6. This illustration is a simple overview of an LSTM cell. The input, output and the forget gates modulate the cell of the LSTM and help the gradients propagate and the information flow through time more easily.

2.3.4. Gated Recurrent Unit (GRU)

GRUs (Cho, van Merrienboer, Gulcehre, Bougares, Schwenk, and Bengio, 2014a) are a simplified version of the LSTM with fewer parameters and with two gates instead of three. Models using GRUs have been proven to be successful in different applications such as NLP (Cho *et al.*, 2014a; Trischler, Wang, Yuan, Harris, Sordoni, Bachman, and Suleman, 2016), speech (Bahdanau, Chorowski, Serdyuk, Brakel, and Bengio, 2016b) and vision applications (Visin, Kastner, Courville, Bengio, Matteucci, and Cho, 2015). The GRU has only update and reset gates: the update gate decides whether to just carry the information from the past or use the new candidate state based on the current input and the hidden state of the

previous timestep, see Equation 2.3.12. Reset ($\mathbf{r}_t$) and update ($\mathbf{g}_t$) gates both use the sigmoid activation function and smooth differentiable gating mechanisms similar to the LSTM as shown in Equation 2.3.11. This enables the model to be easily trained with gradient-based optimization techniques.

The update gate of the GRU acts similarly to leaky integrator units (Bengio, Boulanger-Lewandowski, and Pascanu, 2013a), but with smooth and learned per-unit leakage rate. The reset gate can learn to reset the information coming from the memory and just use the input for the current timestep. The details of the model and its computational graph are shown in Figure 2.7. The GRU can be seen as a special case of the LSTM, where both the input and the forget gates are tied together (Greff, Srivastava, Koutník, Steunebrink, and Schmidhuber, 2016).

$$\begin{pmatrix} \mathbf{g}_t \\ \mathbf{r}_t \end{pmatrix} = \texttt{sigm}\left(\mathbf{W}_h(\mathbf{r}_t \odot \mathbf{h}_{t-1}) \;+\; \mathbf{W}_x\mathbf{x}_t \;+\; \mathbf{b}_h\right), \tag{2.3.11}$$

$$\begin{aligned} \mathbf{c}_t &= \texttt{tanh}(\mathbf{W}_h^c(\mathbf{r}_t \odot \mathbf{h}_{t-1}) \;+\; \mathbf{W}_x^c\mathbf{x}_t \;+\; \mathbf{b}_c) \\ \mathbf{h}_t &= \mathbf{g}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{g}_t) \odot \mathbf{c}_t \end{aligned} \tag{2.3.12}$$

2.4. Models with Explicit Memory Structure

It is possible to represent the memory of a neural network in a matrix whose rows corresponds to the contents of the memory. We call this memory matrix $\mathbf{M} \in \mathbb{R}^{d_k \times d_m}$ with d_m features for each k memory cell. This is because it comes with an explicit memory structure and the capacity of the model can be increased by just adding new rows into $\mathbf{M}$. This can be done without introducing additional parameters into the model. When we increase the memory capacity of an ordinary RNN, we would need to increase the size of the hidden state which would result quadratic increase in the number of parameters.

2.4.1. Neural Turing Machines

The neural Turing machine (NTM) is an approach to neural networks with explicit memory (Graves, Wayne, and Danihelka, 2014). The model learns to read from and write into the memory by using read and write heads accessed via the controller. Read and the write heads are neural networks that are conditioned on the hidden state of the controller. The controller $f_{control}(\cdot)$ can either be an LSTM or plain RNN. The size of the memory ($\mathbf{M}$) is fixed at the beginning of training and $\mathbf{M}_t$ is updated at every time-step via the write head.

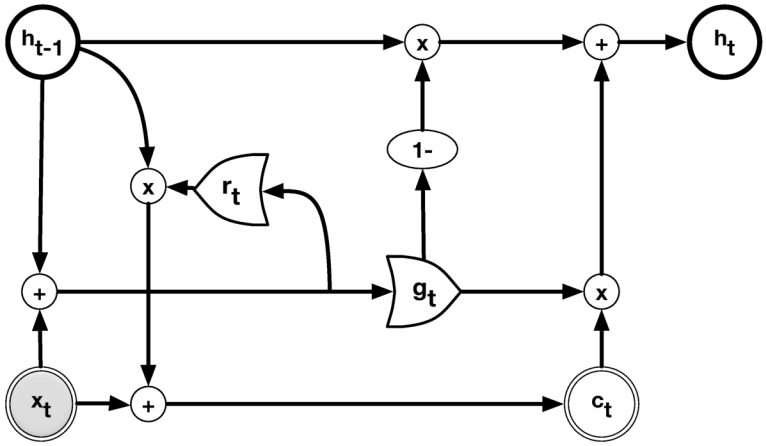


FIG. 2.7. As opposed to the LSTM cells, GRU has only two gates and fewer parameters. Thus implementation of GRU is easier and it is more efficient both in terms of memory and the runtime than an LSTM network.

The hidden state of the controller ($f_{controller}(\cdot)$) is a deep-LSTM network, written as,

$$\mathbf{h}_t = f_{LSTM}(\mathbf{z}_t, \mathbf{m}_t, \mathbf{h}_{t-1}) \tag{2.4.1}$$

Here we will focus on the description of content-based addressing. In the case of content-based addressing, the read weights are computed based on a score with respect to a distance metric $d(\cdot, \cdot)$ and computed between each i^{th} row of the memory $\mathbf{M}_t(i)$ and the the query vectory $\mathbf{q}_t$:

$$\alpha_t^{(i)} = f_{score}(d(\mathbf{M}_t(i), \mathbf{q}_t)) \tag{2.4.2}$$

The query vector ($\mathbf{q}_t$) is obtained by applying a simple linear projection over the hidden-state of the controller.

A common distance metric that is used for $d(\mathbf{M}_t(i), \mathbf{q}_t)$ is the cosine distance:

$$d(\mathbf{M}_t(i), \mathbf{q}_t) = \frac{\mathbf{M}_t(i).\mathbf{q}_t}{||\mathbf{M}_t(i)||||\mathbf{q}_t||} \tag{2.4.3}$$

And the memory content read ($\mathbf{m}_t$) would be,

$$\mathbf{m}_t = \sum_i \alpha_t^{(i)} \mathbf{M}_t(i) \tag{2.4.4}$$

The writing operation involves erasing the old content and updating the memory by using the new content. Both the erase $\mathbf{w}_t^e \in \mathbb{R}^{d_m}$ and the write weights $\mathbf{w}_t^w \in \mathbb{R}^k$ are computed in a similar way to the read weights. The writing happens by first erasing some of the contents from the memory,

$$\mathbf{M}'_t(i) = \mathbf{M}_{t-1}[\mathbf{1} - \mathbf{w}_t^w(i)\mathbf{w}_t^e] \tag{2.4.5}$$

After erasing some of the contents from the memory the model adds the new content ($\mathbf{a}_t \in \mathbb{R}^{d_m}$),

$$\mathbf{M}_t(i) = \mathbf{M}'_t(i) + \mathbf{w}_t^w(i)\mathbf{a}_t \tag{2.4.6}$$

Alternatively, (Graves *et al.*, 2014) also proposed to use location-based addressing which can facilitate both simple iteration across the locations of the memory and random access jumps. We will not go into the details of that, but rotation by one would shift the focus to the next location and negative one would shift the focus in opposite direction. The shifting and the rotation are performed by circular convolution operations. The model also applies sharpening on the attention to make the distribution over different memory cells sharper. Let us note that all the operations that we have described above are fully differentiable and end-to-end trainable.

NTMs are shown to be able to solve several challenging algorithmic tasks that LSTMs fail. Also we have shown that this type of MANNs can deal better with long term dependencies by creating shortcuts through time in the memory (Gulcehre *et al.*, 2017a).

2.4.2. Memory Networks

Memory networks (Weston, Chopra, and Bordes, 2015a) are an approach to integrate an explicit memory into the neural networks. The contents of the memory is usually considered an external resource and the model learns to which part of the memory to read from. The memory can be the continuous representation of the input document or a knowledge base. The memory networks have been proven to be successful in various NLP applications (Weston *et al.*, 2015a; Sukhbaatar, Szlam, Weston, and Fergus, 2015; Kumar, Irsoy, Su, Bradbury, English, Pierce, Ondruska, Gulrajani, and Socher, 2015; Miller, Fisch, Dodge, Karimi, Bordes, and Weston, 2016a). The main distinction of the memory networks compared to the NTMs is that the writing operation in memory networks is not being learned but rather fixed via heuristics. The main advantage of this approach is that, we overcome avoid the difficulty of learning to write —as a result the training becomes easier and more stable.

2.5. On the Loss Surfaces of Neural Networks

The loss surfaces of neural networks are known to be highly non-convex. However, some characteristics of the loss surfaces of neural networks were not identified until recently. Due to the nonconvexity of the neural network loss surfaces, most techniques and analysis that are studied in the convex analysis and convex optimization literature do not apply.

Firstly let us mention that **critical points** of a loss function are the points on the domain of the function where the norm of the gradients with respect to its parameters at that point would be 0. A critical point can either correspond to a local minimum or maximum.

A saddle point can be identified by the eigenvalues of the Hessian. A local minimum corresponds to the Hessian being positive semi-definite, such that for all $\mathbf{v}$ and the Hessian matrix $\mathbf{H}$ i.e. $\mathbf{v}^\top \mathbf{H}\mathbf{v} \geq 0$— in other words, all the eigenvalues of the Hessian matrix should be positive. A critical point can correspond to a local maximum if all the eigenvalues of the Hessian matrix are negative. If the Hessian matrix has both positive and negative eigenvalues, the critical point would be a saddle-point. Intuitively, as the number of parameters (N) of the neural network grows with N parameters, it is unlikely that all the eigenvalues of the Hessian matrix would be positive. Thus, it is much more likely that a neural network would converge to a saddle point rather than a local-minimum unless the critical point stays at a loss value close to the global minimum.

Dauphin, Pascanu, Gulcehre, Cho, Ganguli, and Bengio (2014) and Choromanska, Henaff, Mathieu, Arous, and LeCun (2015) have both shown that the local minima are not a big problem for SGD because they will tend to lie at a very low value of the cost, almost as low as that of the global minima. Dauphin *et al.* (2014) focuses on the analysis of critical points, showing empirically —and by reference to existing math results— that on the optimization path the nearby critical points will be saddle points, and only when approaching to the global minimum cost, there will be local minima. Choromanska *et al.* (2015) further shows empirically that the effect of network size on this effect and uses a different set of mathematical tools (in some sense more restrictive in the assumptions made on the parametric form) to come to the same conclusion about the innocuousness of local minima. Both of those papers have revolutionized our view of the optimization of neural networks.

Kawaguchi (2016) has shown that every local minimum is also a global minimum for a neural network (yielding to 0 training error) with additional simplifying assumptions. For such networks, every critical point that is not a global minimum can be considered as a saddle-point. Sagun, Bottou, and LeCun (2016) empirically investigated the eigenvalues of the Hessian for a neural network that is trained until convergence (achieving very small

training error and norm of the gradient). They have verified that still some of the eigenvalues of the Hessian are negative, though the majority of the eigenvalues concentrate around 0. Thus the authors suggest that the flatness of the error landscape might be beyond the notion of wide basins.

Baldi and Hornik (1989); Saxe, McClelland, and Ganguli (2013) have shown that even the loss surfaces of linear networks can be quite complicated and may involve saddle points.

2.6. Gradient Descent

Optimization algorithms tend to be an iterative process, i.e starting from $\boldsymbol{\theta}^0$ the algorithm will eventually reach to a value $\boldsymbol{\theta}^k$ after k iterations. $\boldsymbol{\theta}^k$ is said to converge to a solution $\boldsymbol{\theta}^*$ iff $||\boldsymbol{\theta}^k - \boldsymbol{\theta}^*||$ becomes 0 as k goes to infinity.

In this section and the subsequent ones ∇ operator refers to the gradient of the function —unless stated otherwise— with respect to its own parameters.

In optimization literature the speed which an algorithm reaches to its optimum value $\boldsymbol{\theta}^*$ at the limit is called as *rate of convergence*. In general the convergence of an algorithm for $\boldsymbol{\theta}_k$ can be written as,

$$\lim_{k\to\infty} \frac{||\boldsymbol{\theta}^{k+1} - \boldsymbol{\theta}^*||}{||\boldsymbol{\theta}^k - \boldsymbol{\theta}^*||^p} = \mu \tag{2.6.1}$$

p and μ are asymptotic rate constants. If $p = 1$ and μ is constant from step to step then the algorithm is said to converge *linearly*. If μ_k changes step to step with $\mu_k \to 0$ and $p = 1$ for $k \to \infty$, then the sequence is said to converge *superlinearly*. If μ_k changes step to step with $\mu_k \to 1$ and $p = 1$ for $k \to \infty$, then the sequence is said to converge *sublinearly*. If $\mu_k > 0$ and $p = 2$ for $k \to \infty$, then the sequence is said to converge *quadratically*.

Gradient Descent (GD) is the most famous family of optimization algorithms used for neural networks mainly due to its simplicity and efficiency. A basic form of the gradient descent algorithm would only require easy to obtain first order gradients. GD iteratively updates the parameters of the model, according to the local gradient information and the step-size (or learning rate) (α_k) at every k^{th} update,

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \alpha_k \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^k) \tag{2.6.2}$$

GD can be treated as a specific form of the hill-climbing algorithm in a continuous space (Russell and Norvig, 1995) or fixed point iteration methods where the iteration continues until the norm of the Jacobians become zero, $||\nabla\mathcal{L}||_2^2 = 0$.

Definition 2.6.1. *A function $f : \mathbb{R}^n \to \mathbb{R}$ is called L-Lipschitz if and only if*

$$||\nabla f(x) - \nabla f(y)||_2 \leq L||x - y||_2 \ \forall x, y \in \mathbb{R}^n$$

Theorem 2.6.1. *If $f(\cdot)$ is a L-Lipschitz function and $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$ and $f(\boldsymbol{\theta}^*) > -\infty$, then the gradient descent algorithm with fixed step-size satisfying $\alpha < \frac{2}{L}$ will converge to a stationary point.*

Proof. Let $\boldsymbol{\theta}^+ = \boldsymbol{\theta}^k - \alpha \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^k)$, since it is L-Lipschitz it will satisfy,

$$f(\boldsymbol{\theta}^+) \leq f(\boldsymbol{\theta}^k) + \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^k)^\top (\boldsymbol{\theta}^+ - \boldsymbol{\theta}^k) + \frac{L}{2}||\boldsymbol{\theta}^+ - \boldsymbol{\theta}^k||^2 \tag{2.6.3}$$

We can replace $\boldsymbol{\theta}^+ = \boldsymbol{\theta}^k - \alpha \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^k)$ in the equation above to get,

$$f(\boldsymbol{\theta}^+) \leq f(\boldsymbol{\theta}^k) - \alpha||\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^k)||^2 + \frac{\alpha^2 L}{2}||\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^k)||^2 \tag{2.6.4}$$

$$= f(\boldsymbol{\theta}^k) - \alpha(1 - \frac{\alpha L}{2})||\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})|| \tag{2.6.5}$$

$$||\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^k)||^2 \leq \frac{1}{\alpha(1 - \frac{\alpha L}{2})}(f(\boldsymbol{\theta}^k) - f(\boldsymbol{\theta}^+)) \tag{2.6.6}$$

If we sum over the iterations, due to the telescoping sums we will obtain,

$$\sum_{k=1}^{\infty} ||\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^k)||^2 \leq \frac{1}{\alpha(1 - \frac{\alpha L}{2})}(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^\infty)) \tag{2.6.7}$$

$$\leq \frac{1}{\alpha(1 - \frac{\alpha L}{2})}(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*)) \tag{2.6.8}$$

This implies that $\lim_{k\to\infty} ||\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^k)||^2 = 0$.

□

Gradient descent can be treated by assuming that the original objective function that we would like to optimize to be a locally quadratic (when you zoom) which can be obtained from the second order Taylor approximation. For the sake of simplicity of our analysis, assuming that $\mathbf{H} \approx \alpha \mathbf{I}$, we can obtain,

$$\mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^*) \approx \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta})^\top (\boldsymbol{\theta}^* - \boldsymbol{\theta}) + \frac{||\boldsymbol{\theta}^* - \boldsymbol{\theta}||_2^2}{2\alpha^{(k)}} \tag{2.6.9}$$

The term $\mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta})^\top (\boldsymbol{\theta}^* - \boldsymbol{\theta})$ is the first order Taylor approximation of $\mathcal{L}(x, y; \boldsymbol{\theta}^*)$ around $\boldsymbol{\theta}$ and it is also the global under-estimator of $\mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^*)$ when it is convex

(also called first-order condition for the gradient descent),

$$\mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^*) \geq \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta})^\top (\boldsymbol{\theta}^* - \boldsymbol{\theta})$$

The term $\frac{1}{2\alpha}||\boldsymbol{\theta} - \boldsymbol{\theta}^*||_2^2$ just penalizes the discrepancy between $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^*$. The full derivation of this equation is simple and given in (Bottou, Curtis, and Nocedal, 2016). α is the learning rate and in Figure 2.8, we have shown how α changes the local quadratic approximation of the univariate convex function $f(x)$ around x'. GD will minimize this quadratic approximation to find the parameters $\boldsymbol{\theta}^k$ at update k,

$$\begin{aligned}
\boldsymbol{\theta}^k &= \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^{k-1}) + \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^{k-1})^\top (\boldsymbol{\theta} - \boldsymbol{\theta}^{k-1}) && (2.6.10)\\
&\quad + \frac{||\boldsymbol{\theta} - \boldsymbol{\theta}^{k-1}||_2^2}{2\alpha^{(k)}}, && (2.6.11)\\
&= \arg\min_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^{k-1})^\top \boldsymbol{\theta} + \frac{||\boldsymbol{\theta} - \boldsymbol{\theta}^{k-1}||_2^2}{2\alpha^{(k)}}, && (2.6.12)\\
&= \arg\min_{\boldsymbol{\theta}} 2\alpha^{(k)} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^{k-1})^\top \boldsymbol{\theta} + ||\boldsymbol{\theta} - \boldsymbol{\theta}^{k-1}||_2^2, && (2.6.13)\\
&= \arg\min_{\boldsymbol{\theta}} ||\boldsymbol{\theta} - (\boldsymbol{\theta}^{k-1} - \alpha^{(k)} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^{k-1})||_2^2. && (2.6.14)
\end{aligned}$$

To ensure the convergence as we discussed earlier, the gradients of the objective should be Lipschitz continuous,

$$||\nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^k) - \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^{k-1})|| \leq \beta ||\boldsymbol{\theta}^k - \boldsymbol{\theta}^{k-1}||, \ for\ 0 \leq \beta \leq 1$$

The Lipschitz continuity would imply that the difference $||\nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^k) - \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^{k-1})||$ should be contractive and if $\mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^k)$ is twice differentiable then the loss function would be convex too. Assuming that we satisfy those conditions, GD would have a rate of convergence of $O(1/k)$ for k being the number of iterations. In order to achieve a bound of $\mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^k) - \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^*) \leq \epsilon$, we must run the algorithm for $O(1/\epsilon)$ iterations. This also implies a sublinear convergence.

If $\mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^k)$ is strongly convex, the gradient descent converges with rate $O(c^k)$ for $0 < c < 1$. This means that a bound of $\mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^k) - \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^*) \leq \epsilon$ can be achieved in only $O(log(1/\epsilon))$ iterations. This is also called linear convergence.

There are two popular forms of gradient descent to train machine learning algorithms, *"Stochastic"* and *"Batch"* learning. Batch learning algorithms require all the training examples to be loaded into the memory and the gradients will be computed over the whole dataset

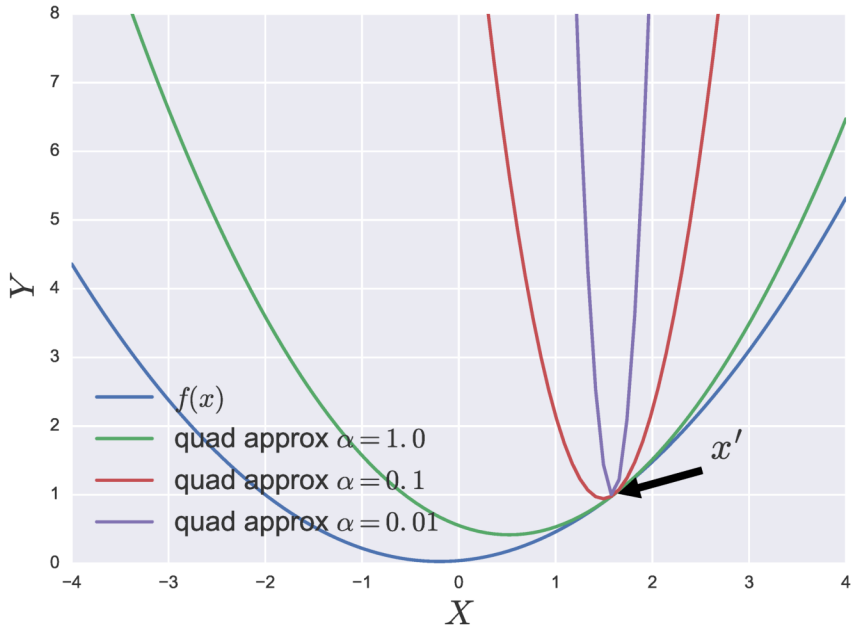


FIG. 2.8. We show the quadratic approximation of a convex function $f(x)$ around x' with respect to different learning rates (α). At each update GD will try to find the minimum of the quadratic approximation.

whereas stochastic algorithms can only do the updates based on the smaller portions of the training set.

2.6.1. Stochastic Gradient Descent

Stochastic gradient descent is just a stochastic approximation for the gradient descent based on sampling a minibatches of examples and computing a Monte-Carlo approximation of the truly expected loss $\mathrm{E}_{(\mathbf{x},\mathbf{y})\sim P(\mathbf{x},\mathbf{y})}[\mathcal{L}(\mathbf{x},\mathbf{y};\boldsymbol{\theta})]$ [2],

$$\mathrm{E}_{(\mathbf{x},\mathbf{y})\sim P(\mathbf{x},\mathbf{y})}[\mathcal{L}(\mathbf{x},\mathbf{y};\boldsymbol{\theta})] = \int \mathcal{L}(\mathbf{x},\mathbf{y};\boldsymbol{\theta}) dP(\mathbf{x},\mathbf{y}) \tag{2.6.15}$$

$$\approx \frac{1}{N}\sum_{i=0}^{N} \mathcal{L}(\mathbf{x}_i,\mathbf{y}_i;\boldsymbol{\theta}) \tag{2.6.16}$$

Depending on the application, minibatches can either be uniformly random sampled before each update from the whole dataset or they can be shuffled at the beginning of the training (Bottou, 2012). The convergence of SGD relies on a decaying learning rate, such that

[2]The Monte-Carlo estimate of the expected loss is also called as empirical loss in the ML literature.

$\sum_i \alpha^{(i)} = \infty$ and $\sum_i (\alpha^{(i)})^2 < \infty$ (Robbins and Monro, 1951). SGD usually has worse rate of convergence (sublinear) compared to GD which is $O(1/\sqrt{k})$. For minibatch of size N, the convergence of SGD can be shown to be $O(1/\sqrt{Nk} + 1/k)$ (Dekel, Gilad-Bachrach, Shamir, and Xiao, 2011). Since N examples will be processed at each iteration the improvement will get worse as the size of the minibatches increase. However, SGD is more immune to the redundancy in the dataset and less vulnerable to the issues regarding the non-convex optimization when compared to the GD (Choromanska *et al.*, 2015). Another important reason for using SGD is that it is empirically well-known that SGD generalizes better and Hardt, Recht, and Singer (2015) have shown that SGD can generalize better due to its stability promoting behavior using some basic results in the convex optimization literature.

Momentum

Momentum (Polyak, 1964) is a very popular way to speed up the convergence of gradient descent based methods. Momentum basically adds a memory to the updates and improves the speed of GD by adding an inertia to it. The inertia comes from accumulated velocity vectors $\mathbf{v}_t$ which is added to the gradients with scaled velocity for every update. For the momentum of rate β, the momentum updates can be obtained by modifying the standard gradient descent rule as,

$$\mathbf{v}^t = \beta \mathbf{v}^{t-1} - \alpha^{(t)} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}_t, \mathbf{y}_t; \boldsymbol{\theta}^{t-1}) \tag{2.6.17}$$

$$\boldsymbol{\theta}^t = \boldsymbol{\theta}^{t-1} + \mathbf{v}^t \tag{2.6.18}$$

An important issue with optimization is pathological curvature. In particular, pathological curvature can appear when the regions of the loss surface are not scaled properly. This usually happens when the Hessian matrix is *ill-conditioned* [3], such that GD will stall in the areas of low curvature. However, momentum would address this issue by amplifying the steps in the low-curvature directions of the loss surface. We illustrate this phenomenon in Figure 2.9 for gradient descent with fixed size. [4]

A drawback of momentum is that in the regimes of high-curvature it can amplify the size of the steps and cause oscillations. Those oscillations can cause unstable training and slower convergence. Sutskever, Martens, Dahl, and Hinton (2013) have proposed to address this issue by using the *Nestorov momentum* for deep neural networks. Nesterov momentum first

[3] Ill conditioning would happen when the condition number of the Hessian, the ratio between the largest and the smallest singular value, is very large.

[4] This figure is obtained by using the excellent online simulation for momentum provided in Goh (2017).

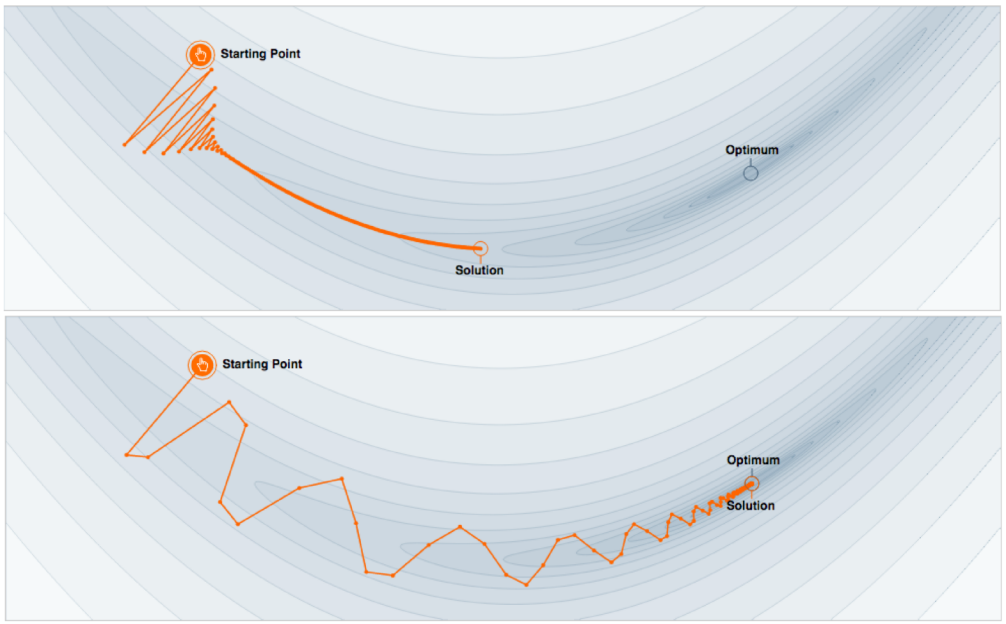


FIG. 2.9. We show the trajectories followed by a GD and GD with momentum with a fixed learning rate of $\alpha = 0.02$. In Figure (A), the steps become very small around the pathological curvature and GD just stalls. In Figure (B), we show GD with momentum rate of 0.86, as seen in this figure the momentum helps GD to escape from the pathological curvature.

performs a partial update on the parameters of the model and then accumulates the velocity by iteratively adding the steps to the velocity,

$$\mathbf{v}^t = \beta \mathbf{v}^{t-1} - \epsilon \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}_t, \mathbf{y}_t; \boldsymbol{\theta}^{t-1} + \beta \mathbf{v}_{t-1}) \tag{2.6.19}$$

$$\boldsymbol{\theta}^t = \boldsymbol{\theta}^{t-1} + \mathbf{v}^t \tag{2.6.20}$$

2.6.2. Adaptive Learning Rate Algorithms

The use of one single global learning rate to update all the parameters of a neural network may not be a very effective approach for the optimization of neural networks. In particular, in the vicinity of a saddle-point the first order gradient information may not be very feasible. On the other hand, second-order or Natural Gradient methods would require one to compute a very large matrix and to invert that matrix which can be quite expensive and infeasible for large neural networks. As a result, diagonal rescaling of the gradients has been quite popular for optimizing neural networks. These algorithms can be viewed as diagonal preconditioner

that can reshape or modify the local geometry of the loss surface in order to improve first-order optimization. In the following two sections we will be focusing on two very popular adaptive learning rate algorithms, RMSProp and Adam. An important advantage of using adaptive algorithms is that those approaches make learning more robust to the choice of learning rate. There have been approaches that propose to completely eliminate the burden of finetuning a learning rate (Gulcehre, Sotelo, Moczulski, and Bengio, 2017c; Schaul, Zhang, and LeCun, 2013), however, those approaches have not gained popularity at this point. In this section, with an abuse of notation, the division operation "/" refers to an element-wise division between two vectors.

RMSProp

RMSProp (Tieleman and Hinton, 2012) has been a very popular method for training neural networks, in particular RNNs (Graves *et al.*, 2013). RMSProp is basically inspired by the RPROP algorithm which increases or decreases the learning rate multiplicatively based on whether if the signs of the gradients for the last two updates agree or not. However, the RPROP algorithm does not work in the stochastic setting. RMSProp accumulates the root-mean-square (RMS) statistics about the gradients using running averages. In that sense, RMSProp estimates the per parameter variance of the gradients via moving averages and uses this estimate in order to normalize the gradients. This normalization process can augment the magnitude of the gradients adaptively when they shrink around a saddle point. RMSProp is also related to the equilibration methods that can speed up the training of the neural networks for escaping from saddle points faster (Dauphin, de Vries, and Bengio, 2015).

$$\mathbf{g}^{(k)} = \beta \mathbf{g}^{(k-1)} + (1-\beta)(\nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^{k-1}))^2 \tag{2.6.21}$$

$$\boldsymbol{\theta}^{k} = \boldsymbol{\theta}^{k-1} - \alpha_k \frac{\nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^{k-1})}{\sqrt{\mathbf{g}^{(k)} + \epsilon}} \tag{2.6.22}$$

Adam

The Adam (Kingma and Ba, 2014) algorithm is similar to RMSProp and uses the statistics obtained from first order gradients via moving averages. As opposed to just normalizing by the second order moment of the gradients, Adam uses the ratio between first order moment and the second order moment of the gradients as well. Kingma and Ba (2014) claims that this approach behaves like an approximate trust region method around the current values of the parameters, and it adopts not only the learning rates but also the momentum as well.

$$\mathbf{g}^{(k)} = \beta_1 \mathbf{g}^{(k-1)} + (1-\beta_1)(\nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^{k-1}))^2 \tag{2.6.23}$$

$$\mathbf{a}^{(k)} = \beta_2 \mathbf{a}^{(k-1)} + (1-\beta_2) \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^{k-1}) \tag{2.6.24}$$

$$\bar{\mathbf{a}}^{(k)} = \frac{\mathbf{a}^{(k)}}{(1-\beta_1)} \tag{2.6.25}$$

$$\bar{\mathbf{g}}^{(k)} = \frac{\mathbf{g}^{(k)}}{(1-\beta_2)} \tag{2.6.26}$$

$$\boldsymbol{\theta}^{k} = \boldsymbol{\theta}^{k-1} - \alpha_k \frac{\bar{\mathbf{a}}^{(k)}}{\sqrt{\bar{\mathbf{g}}^{(k)} + \epsilon}} \odot \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}^{k-1}) \tag{2.6.27}$$

2.7. Curriculum Learning

The main idea of curriculum learning is to solve problems that are difficult to optimize by ordering examples seen by the model from easier to more difficult (Bengio *et al.*, 2009a). The notion of curriculum learning lies back to the incremental learning idea proposed in (Elman, 1993), in which they have shown that it is possible to learn language tasks with a curriculum that is otherwise not solvable by the network. (Bengio *et al.*, 2009a) have shown that it is possible to solve highly non-convex and difficult to optimize problems with a curriculum and they have established its relationship to continuation methods.

2.8. Training Neural Networks with Discrete Decisions

Training neural networks with discrete decisions has been an important research topic that can be useful for various applications of neural networks, e.g.: discrete attention (Mnih, Heess, Graves, *et al.*, 2014), accessing to memory (Zaremba and Sutskever, 2015), conditional computation (Bengio, Léonard, and Courville, 2013b; McGill and Perona, 2017), deep reinforcement learning (Wierstra, Foerster, Peters, and Schmidhuber, 2007). The ability to learn discrete random variables in the architecture may also enable our models to both scale and generalize better. However, learning discrete variables with gradient based optimization techniques is well-known to be challenging due to tradeoffs that arise from bias and variance of the approximations.

One of the simplest approaches for the dealing with gradients of the discrete random variables is to use the **Straight-through** estimator (Bengio *et al.*, 2013b). The idea behind the straight-through estimator is to treat the discrete activation function as if it is a smooth function during the backpropagation, and for the forward computation just to use the discrete function. For example, if a model has a sign function in the forward propagation, to be able

to backpropagate through this function, we treat it as if it is the $\tanh(\cdot)$ function. This is clearly a biased estimator of the gradient, however, in many applications, this approach seems to work relatively well.

Another approach is to approximate the gradients with a particular form of reinforcement learning (RL) by using **REINFORCE** algorithm (Williams, 1992). REINFORCE is a special case of likelihood ratio method and in RL literature it is also known as *Monte-Carlo Policy Gradients*. REINFORCE and policy gradients are more specialized on the stochastic policies and estimating the gradients of the stochastic random variables.[5]

Suppose that x is a random variable with probability density of $p(x; \boldsymbol{\theta})$, and $\psi(\cdot)$ is a scalar-valued function, e.g. the reward, we are interested in maximizing,

$$x^* = \arg\max \mathrm{E}_x[\psi(x)],$$

The function $\psi(x)$ may not be differentiable. However, we can approximate the derivative of $\nabla_{\boldsymbol{\theta}} \mathrm{E}_x[\psi(x)]$,

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \mathrm{E}_x[\psi(x)] &= \nabla_{\boldsymbol{\theta}} \int_x p(x; \boldsymbol{\theta}) \psi(x) dx = \int_x \nabla_{\boldsymbol{\theta}} p(x; \boldsymbol{\theta}) \psi(x) dx, \\ &= \int_x p(x; \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log(p(x; \boldsymbol{\theta})) \psi(x) dx = E_x[\psi(x) \nabla_{\boldsymbol{\theta}} \log(p(x; \boldsymbol{\theta}))],\end{aligned}$$

We can also approximate the integral via the Monte-Carlo approximation,

$$\nabla_{\boldsymbol{\theta}} \mathrm{E}_x[f(x)] \approx \frac{1}{N} \sum_{i=0}^{N} \psi(x_i) \nabla_{\boldsymbol{\theta}} \log(p(x_i; \boldsymbol{\theta})), \tag{2.8.1}$$

However Equation (2.8.1) is well known to be a high variance approximation. It is possible to reduce the variance of this approximation by subtracting a baseline (b) from the rewards (Greensmith, Bartlett, and Baxter, 2004). Baseline can be any function as long as it does not depend on x_i's, in order to not to introduce any bias i.e., the gradient of the subtracted quantity would be 0,

$$\sum_{i=0}^{N} b \nabla_{\boldsymbol{\theta}} \log(p(x_i; \boldsymbol{\theta})) = b \nabla_{\boldsymbol{\theta}} \sum_{i=0}^{N} \log(p(x_i; \boldsymbol{\theta})) = b \nabla_{\boldsymbol{\theta}} 1 = 0 \tag{2.8.2}$$

[5]Although it is possible to train deterministic policy gradients, so far in the literature it is mainly used for reinforcement learning with continuous outputs (Silver, Lever, Heess, Degris, Wierstra, and Riedmiller, 2014).

Chapter 3

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars

– Generated by a character-level LSTM language model on the Shakespeare corpus (Karpathy, Johnson, and Fei-Fei, 2015).

Natural languages evolved to represent an information in a human-interpretable fashion and communicate it with other agents in a verbal manner. Language is often considered as an interface to the human mind and sometimes is referred as a mirror of the mind. Human knowledge is recorded in the form of text for the means of communication and abundant amount of data available in the form of text. As a consequence, understanding and studying language based on the largely available data can also give better insights into the human mind as well.

The inception of using neural networks (Bengio, Ducharme, and Vincent, 2000) for NLP has improved the state of results on several NLP benchmark tasks (Mikolov *et al.*, 2013; Collobert *et al.*, 2011). Immediately, it has become an essential tool for many different applications on different domains. However, compared to the influence that the neural networks have had on computer vision, the achievements obtained in NLP is still in its infancy. The essential NLP problems are still far from being solved. In this section, we investigate different NLP applications and the deep learning approaches that have been proposed to solve those tasks.

Natural Language Understanding (NLU) is the ultimate goal of NLP. NLU requires one to be able to comprehend and understand a text or a document in the semantic-level besides the low-level syntactic aspects. NLU would enable to answer questions related to a text. For example, the question can be fixed for all the texts in the corpora. It can be as simple as "what should be the next word given the preceding (or surrounding) text?" as in language

modeling or "what is the translation of this sentence in German?". We call this type of questions as *"implicit questions"*, mainly because it is not provided in the corpus explicitly, but the model should only be able to learn a mapping from input to output that would answer this question. The model also does not need to learn a representation for the "implicit question" since providing a constant for each input would not provide any useful information to solve the task. On the other hand, the tasks that involve "explicit questions" such as general question-answering or dialogue generation [1] are still quite far from being solved and they are known to be general AI-hard tasks.

An advantage of the deep NLP methods over traditional NLP is the fact that a neural network can learn a continuous vector representation of the words which is oftenwise called as an embedding (Bengio, Ducharme, Vincent, and Janvin, 2003). The ability of learning continuous representations for discrete tokens of words enables the model to be able to learn semantic associations between different words. For example, continuous-space embedding of the words can learn to cluster the synonyms close to each other and antonyms further apart in the Euclidean space. Mikolov *et al.* (2013) have shown those embeddings can be semantically rich enough to be able to perform simple reasoning and analogy making.

3.1. Language Modeling

The goal of the language modeling is to model the joint distribution of the sequence of words $\mathbf{w} = (w_0, w_1, \cdots, w_T)$ of length $T = |\mathbf{w}|$ in a sentence. It is possible to factorize the probabilities with the chain rule, to model the distribution of the words:

$$p(w_0, w_1, \cdots, w_T) = p(w_0) \prod_{i=1}^{T} p(w_i | w_{i-1}, \cdots, w_0) \tag{3.1.1}$$

Basically what the chain rule of probabilities suggests is to predict the next word at every timestep depending on the tokens that have been seen in the past. The probabilistic modeling of language has been very popular in the context of generative text such as in statistical machine translation (Koehn, 2009) for generating translations in the target language, spelling correction to fix the typos and missing words/characters (Kukich, 1992), re-ranking candidates for speech recognition to improve the fluency of the generated transcriptions (Collins and Koo, 2005).

The methods based on the n-gram language modeling (Jurafsky, 2000) have been popular in NLP just for their simplicity. The fundamental principle behind n-gram based language modeling techniques relies on the assumption that each word in a sentence/sequence only

[1] Let us note that those tasks still involve implicit questions as well.

depends on n words (grams) that appear before itself, namely $(n-1)^{th}$ order Markov assumption,

$$p(w_0, \cdots, w_T) \approx p(w_0) \prod_{i=1}^{T} p(w_i | w_{i-1}, \cdots, w_{i-n}) \tag{3.1.2}$$

An n-gram refers to the contiguous sequence of n tokens in the sequence. Estimating the n-gram probabilities with **maximum-likelihood estimation** is simple, just counting the frequency of the n-grams in the corpus is sufficient. However, data-sparsity can be a big issue. For example, for a word/gram that has never appeared in the training set it will assign 0 probability. Overfitting can be an important issue for the n-gram based techniques because the n-gram model just memorizes the training dataset. However, there are different smoothing techniques that try to address this issue, but none of these techniques completely eliminates this problem.

Neural language modeling (Bengio *et al.*, 2003) models the probability of the next word given the previous history ($p(w_i|w_{i-1}, \cdots, w_0)$) with a neural network, typically by using an RNN. A neural language model (NLM) would also learn a distributed representation for the words. This would enable the model to cluster the words that are similar or that co-occur very often together closer in the Euclidean space. Visualizing and analyzing the continuous representation of the word embeddings of the NLM can give us a good intuition about the biases in the dataset (Bolukbasi, Chang, Zou, Saligrama, and Kalai, 2016), help to debug the model and make analogies between different words (Mikolov *et al.*, 2013). For example given an analogy-making riddle, "man is to king as woman is to x" (denoted as $man : king :: woman : x$), simple arithmetic of the embedding vectors finds that $x := queen$ is the best answer because:

$$\mathbf{man} - \mathbf{woman} \approx \mathbf{king} - \mathbf{queen}$$

An RNN based LM (RNNLM) (Mikolov *et al.*, 2013) can potentially keep an unbounded history of the words that appears in the input sequence in its recurrent state. Typically, RNNLM would learn to model the conditional distribution over words,

$$p(w_t | w_{<t}) \propto \mathbf{y}_t^\top \exp(\mathbf{W}_o f(\mathbf{h}^{t-1}, w_{t-1}) + \mathbf{b}_o)), \tag{3.1.3}$$

where $\mathbf{y}_t$ is a one-hot encoded vector indicating one of the words in the target vocabulary. $\mathbf{W}_o$ is a learned weight matrix and $\mathbf{b}_o$ is a bias and they are the parameters of the logit layer. $f(\cdot)$ is the function that computes the hidden state of the RNN.

The whole model is jointly trained to maximize the (conditional) log-likelihood of the training corpus:

$$\max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=0}^{T} \sum_{n=1}^{N} \log p_{\theta}(w_i^{(n)} | w_{<i}^{(n)}),$$

where $\boldsymbol{\theta}$ denotes a set of all the tunable parameters.

The most common types of language modeling tasks are character-level and word-level language modeling. The sequences of characters can be quite long and the RNN that is being trained can be more prone to the difficulties of training RNNs over very long sequences, i.e. vanishing and exploding gradients, than the word-level language modelling. On the other hand, the word-level NLP tasks would be more prone to the issues related to the data-sparsity which happens as a result of rare-words and misspellings/typos in the corpus.

Word-level language modeling would have to deal with the scalability issues coming from the computation of the large-softmax at the output layer. Approaches like the "large vocabulary trick" (Jean, Cho, Memisevic, and Bengio, 2015), "hierarchical softmax" (Morin and Bengio, 2005) and "noise contrastive estimation" (Mnih and Kavukcuoglu, 2013) are trying to solve this issue by approximating the softmax. A character-level NLM would not suffer from those issues. However it can have memory and speed issues due to the length of the sequences.

Language modeling is the most successful method of doing unsupervised learning in NLP. It is relatively easy to train a language model on a monolingual corpus that can capture syntactic dependencies relatively well, but it is difficult to obtain semantically coherent text from an LM trained unconditionally on a monolingual corpus.

Evaluating Language Models: One of the best technique to evaluate a language model is to have humans to evaluate the generated samples. **Human evaluation** can be expensive and time-consuming. Thus, the general research direction has focused on methods that use **intrinsic evaluation** techniques, such as evaluating based on the **perplexity** ($PP(\cdot)$ for short). Perplexity would be computed in particular on a *development* set and it as the average inverse probabilities of the words on a development set,

$$PP(\mathbf{w}) = (p(\mathbf{w}))^{-\frac{1}{|\mathbf{w}|}} \tag{3.1.4}$$

$$= \left(\frac{1}{p(w_0) \prod_{i=1}^{T} p(w_i | w_{i-1}, \cdots, w_0)} \right)^{\frac{1}{|\mathbf{w}|}} \tag{3.1.5}$$

Intuitively perplexity gives us the weighted-average of branching factor for a language model. Namely, if the perplexity is high, the model would need to decide among a large number of words to predict the correct word.

3.1.1. Teacher Forcing

At every timestep, the language model will get an input which can be a word, subword, character, and etc. In order to train an RNN with teacher forcing, during the training, the ground-truth tokens are fed into the language model as input instead of the model's its own output. Nevertheless, during the evaluation and test time the model's own samples —from its predicted next-word distributions— are fed back into the neural network as shown in Figure 3.1.

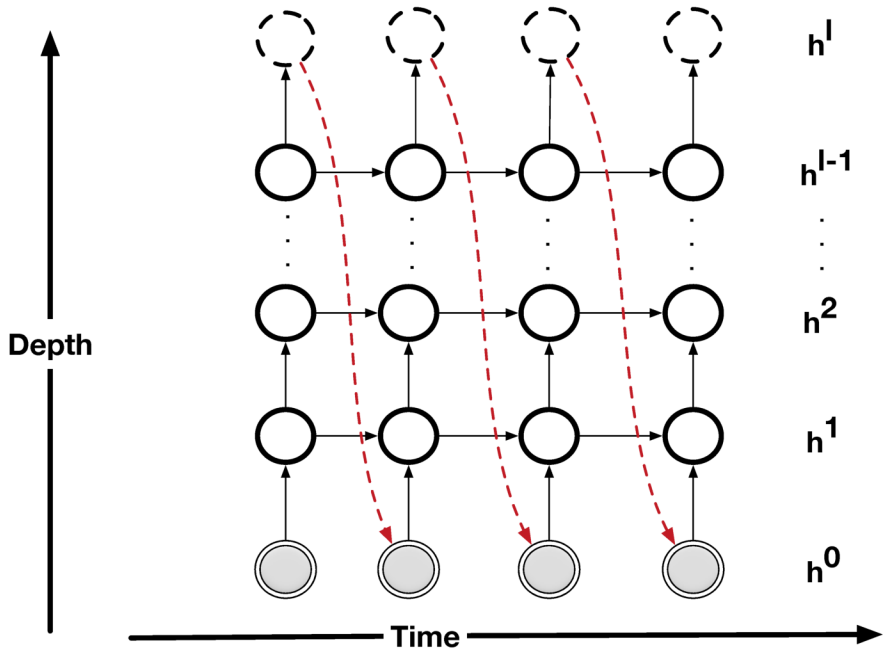


FIG. 3.1. The red arrows in this figure refers to the teacher forcing. Teacher forcing requires the model to take true labels as input during the training and during the evaluation the model's own predictions are fed-back to it.

This is the correct way to do maximum-likelihood when training RNNs on language modeling tasks. However, teacher forcing has well-known issues since during the training the model is not able to learn from its own mistakes (Goodfellow, Bengio, and Courville, 2016).

3.2. Encoder-Decoder Approaches

The main premise of the encoder-decoder approaches for the NLP is to learn a mapping from a source sequence to the target sequence by first encoding the source sequence and then decoding the target based on the summary of the source sequence (Cho *et al.*, 2014a; Sutskever *et al.*, 2014; Kalchbrenner and Blunsom, 2013). In this sense the context can be represented using different methods.

3.2.1. Representing the Context

It is common to use neural networks to obtain a representation of the source context. A very common choice for that is to either use RNNs to get the summary or convolutional neural networks (Kalchbrenner, Grefenstette, and Blunsom, 2014).

One of the simplest way to obtain the summary of the source sequence is to summarize it via a recurrent neural network (Sutskever *et al.*, 2014; Cho *et al.*, 2014a), and the last state of this encoder RNN can be used as a context. The context will be fed back into the decoder RNN in order generate the target sequence. The general architecture is illustrated in Figure 3.2.

Instead of obtaining a single $\mathbf{c}$ for every time-step, it is possible to use a different $\mathbf{c}_t$ for every time-step t by using attention (Bahdanau *et al.*, 2014). We will further discuss the attention mechanism for the neural networks in the following sections of this thesis.

3.3. Machine Translation

Machine translation is one of the most challenging natural language processing task. It has been one of the hallmark application of AI and a topic of philosophical debates for a long time, for example the "Chinese Room Argument" discusses the possibility of true AI based on "Machine Translation" (Cole, 2009).

One of the difficulties of machine translation originates from the difficulty in keeping the semantics of the source sentence (X) fixed while translating it into a sentence in the target language (Y). Mapping from source to target sentences is a many-to-many relationship. Each sentence in the source language can have multiple translations and those sentences do not have to be mutually exclusive in terms of semantics. The differences in terms of word-orderings and the syntactic differences between different languages can also introduce difficulties between language pairs.

Traditional natural language processing (NLP) approaches for Machine Translation involve an alignment problem (can be related to a word or phrase alignment) which can require the

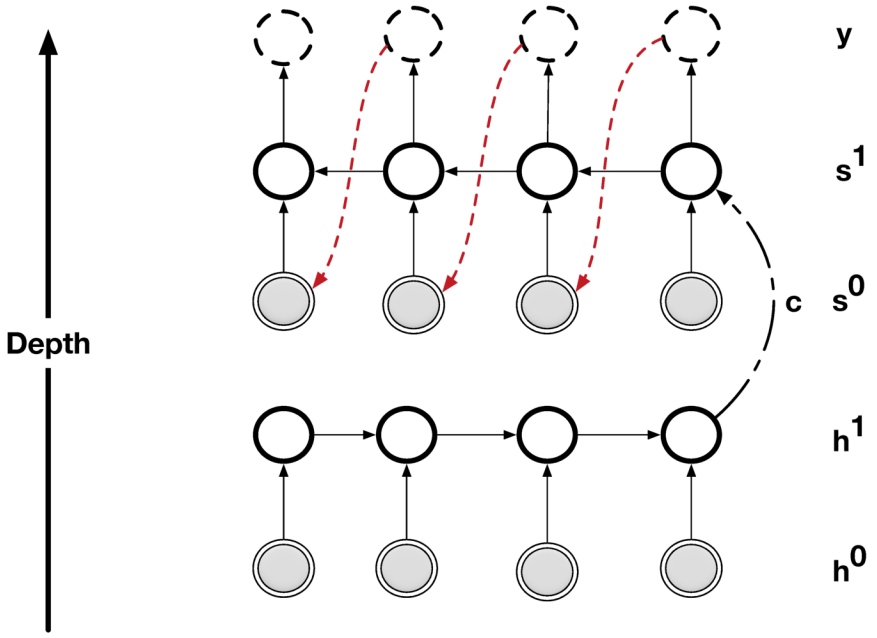


FIG. 3.2. The red arrows in this figure refers to the teacher forcing. In this figure, we have depicted an encoder-decoder architecture for a sequence to sequence model. The encoder ($\mathbf{h}^1$) maps the source sequence $\mathbf{x}$ into the summary $\mathbf{c}$ and by using $\mathbf{c}$, the decoder RNN ($\mathbf{s}$) generates the target sequence $\mathbf{y}$.

utilization of costly operations (Och, Tillmann, Ney, *et al.*, 1999). Basically just doing an exhaustive search in the space of words will obviously be intractable, because the search for MT systems is known to be NP-complete (Och and Ney, 2004).

The task can be summarized as,

$$\mathbf{y}^* = \arg\max_{\mathbf{y}\in \mathrm{Y}} p(\mathbf{y}|\mathbf{x}) \tag{3.3.1}$$

where $\mathbf{s}$ and $\mathbf{t}$ are source and target sequences that needs to be translated. t^* is the most probable sequence that corresponds to the correct translated sequence. We can further decompose this equation,

$$\mathbf{y}^* = \arg\max_{\mathbf{y}\in \mathrm{Y}} p(\mathbf{x}|\mathbf{y})p(\mathbf{y}) \tag{3.3.2}$$

$p(\mathbf{x}|\mathbf{y})$ is usually called inverse translation probability and $p(\mathbf{y})$ is the language model Koehn (2009).

Word based translation techniques would require word reordering and phrase-based models will require phrase reordering. In this case *Inference* corresponds to the actual task of *translating* a given sentence $\mathbf{x}$ into a foreign sentence $\mathbf{y}^*$. *Learning* is to estimate $p(\mathbf{y}|\mathbf{x})$ from data.

Classical machine translation systems use n-gram based language models with expensive search and reordering heuristics. Neural machine translation (NMT) proposes to use fully neural network based algorithms to perform efficient machine translation without any burden of running expensive alignment heuristics.

Neural machine translation (NMT) can be thought of as a language modeling in which the RNNLM is conditioned on a sentence that is in a different language than the one RNNLM is trying to generate in. Neural network based models (Cho *et al.*, 2014a; Sutskever *et al.*, 2013) accomplish this by encoding the sequence of words in the source language $\mathbf{x} = (x_0, \cdots, x_{T_x})$ for $T_x = |\mathbf{x}|$ by using a neural network (encoder) and generating the translation in the target language, $\mathbf{y} = (y_0, \cdots, y_{T_y})$ for $T_y = |\mathbf{y}|$ with another neural network (decoder). The whole model can be trained end to end to be able to maximize the conditional probability of generating the translations in target language given the source,

$$\log p(\mathbf{y}|\mathbf{x}) = \sum_{t=1}^{|\mathbf{y}|} \log p(y_t|y< t, \mathbf{x}) \tag{3.3.3}$$

In order to learn the alignments, between the sequences, neural networks based methods can exploit an attention mechanism to align the source and the target sequences. As suggested by (Bahdanau *et al.*, 2014), the model can learn the alignments and translation at the same time in a completely end to end manner.

3.4. Text Summarization

Similar to the neural machine translation, text-summarization can be cast as a sequence to sequence learning problem where the model tries to learn a mapping from a document $\mathbf{x} = \left(x_0, \cdots, x_{|\mathbf{x}|}\right)$ to the summary $\mathbf{s} = \left(s_0, \cdots, s_{|\mathbf{s}|}\right)$. In general there are two types of summarization tasks that are widely studied,

(1) **Extractive Summarization:** approaches select parts of the source text, then arrange them to form a summary.

(2) **Abstractive Summarization:** approaches use natural language generation to come up with novel summaries which may include passages that are not in the source text.

Abstractive summarization, in general, is a fairly difficult problem and still remains as largely unsolved (Nallapati *et al.*, 2016a) in particular considering different ways of

reformulating the summary and sometimes the model might need to have generalized from additional side information that does not exist in the input document. There have been several deep learning based approaches recently that are proposed that achieves fairly good performance on extractive summarization (Narayan, Papasarantopoulos, Lapata, and Cohen, 2017; Nallapati, Zhai, and Zhou, 2016b).

3.5. Question Answering

Human's quest for knowledge has been an important journey that has contributed advancements of the societies and civilizations. Question answering have been one of the oldest application of NLP (Simmons, Klein, and McConlogue, 1964).

There are two common approaches for question-answering, **text-based question answering** and **knowledge-based question answering**. The text-based QA systems use text as the input data and use information retrieval (IR) techniques to find the relevant documents. On the other hand, knowledge-based question answering techniques leverage the information stored inside a knowledge-base where the information stored in a particular structured form. For example, in Freebase (Bollacker, Evans, Paritosh, Sturge, and Taylor, 2008) each fact is represented in the form of triplets where each triplet has a *subject*, a *relationship* and an *object*.

Below we will review some of the most popular techniques for text-based QA systems,

(1) **Factoid Question Answering:** Most of the current QA systems are focused on solving the Factoid QA problems. Factoid QA basically aims to answer the questions of the form "What is the capital of Brazil?", and the answer would be a single fact. Iyyer, Boyd-Graber, Claudino, Socher, and Daumé III (2014) has proposed an RNN based model to answer factual questions based on a question and a document. In factoid QA systems usually, answer refers to a single word or a short phrase.

(2) **Long-tail Question Answering:** In the long-tail question answering more often usually the answer to the question is a long sentence or a text. The answer to a question such as "How do the magnets work?" would be both long and descriptive. The questions or the answers can be abstract and based on common-sense knowledge (folksonomy) rather than just depending on facts.

Machine Reading Comprehension has been an important research topic especially among the researchers working on question answering. The ability to read a text and accurately comprehend it is essential for both QA and NLU. There have been several works and approaches recently that propose new datasets/tasks (Hermann *et al.*, 2015; Rajpurkar *et al.*, 2016; Nguyen, Rosenberg, Song, Gao, Tiwary, Majumder, and Deng, 2016) and models

(Xiong, Zhong, and Socher, 2016a; Nam, Ha, and Kim, 2016) to address this issue. However most of those tasks are overly simplistic for testing NLU since usually the answer already exists in the document, the task boils down to become a pattern-matching or searching rather than doing more abstract reasoning and over some of those tasks neural networks can already achieve human-level performance (Chen, Bolton, and Manning, 2016a).

Chapter 4

PROLOGUE TO THE FIRST ARTICLE

4.1. Article Details

On integrating a language model into neural machine translation., Gulcehre, Caglar, Orhan Firat, Kelvin Xu, Kyunghyun Cho, and Yoshua Bengio. Computer Speech & Language 45 (2017): 137-148.

Personal Contributions:

The idea of using an external language model to improve the neural machine translation models has been brought up first by Yoshua Bengio in a joint meeting we had after submitting our first NMT paper (Cho *et al.*, 2014a). Kyunghyun Cho suggested Orhan Firat and me to try the idea of "Shallow Fusion" and "Deep Fusion" for IWSLT 2014 and OpenMT 2015 machine translation competitions. The idea of using control gates in the deep fusion was mine. Orhan Firat and I sat down together and regularly performed pair-programming to implement the several parts of the model. We have trained the models together for the submission, but in the final submission, we used the language models that I have trained. We have trained the translation models together with Orhan Firat using different hyperparameter configurations and chose the best ones based on the BLEU score on the dev-set to put in the paper. I wrote the initial draft of the paper for ACL 2015 and Orhan Firat, Kelvin Xu and Kyunghyun Cho did corrections and additions on that draft. Our paper was rejected from ACL 2015. Then we have decided to submit our paper to Elsevier CSL Journal on Special Issue for Neural Machine Translation. For that submission, I had to rerun most of the deep-fusion and shallow-fusion experiments with different hyperparameters and rewrote large part of the paper with the help of Orhan Firat and Kelvin Xu.

4.2. Context

We first started trying the ideas presented in this paper for IWSLT 2014 and OpenMT 2015 machine translation competitions. At the time, "Neural Machine Translation" was just an emerging topic and neural machine translation for the languages besides English,

French and German was still an explored territory. The neural machine translation systems was still behind the statistical machine translation systems with hand-coded features. One reason for that was the statistical machine translation systems use the largely available mono-lingual corpora. We have proposed approaches that extends the attention model proposed in (Bahdanau *et al.*, 2014) with a shallow and deep fusion techniques.

We have first submitted our paper to ACL 2015 and then EMNLP 2015. However, our initial submissions to those conferences were rejected, mainly due to the lack of clarity in the writing. In late 2015, we have updated and largely rewritten the paper and submitted to Elsevier (Computer Speech and Language) journal with additional results and overview of all recent approaches explored after our paper first published on arXiv.

4.3. Contributions

The initial arXiv version of our paper was one of the first application of Neural Machine Translation system on a wide variety of languages with highly inflectional structures such as Turkish.

We have proposed two different ways to integrate unilingual resources into the language model of the neural machine translation models. In shallow fusion, we have trained an LSTM language model on the unilingual data and use its predictions to combine at the output level. In the deep fusion, we have used the LSTM language model trained on the unilingual data in the representation level. Both approaches seemed improve our results both for the low-resource, e.g., Turkish to English IWSLT 2014 and high resource tasks such as WMT German to English.

4.4. Recent Developments

After our initial manuscript has been published on arXiv, there has been several related works that came up on integrating monolingual data for machine translation and low-resource translation. Sennrich et al. (Sennrich, Haddow, and Birch, 2015) proposed two strategies for low resource neural machine translation by making use of monolingual data. The first approach, called "dummy source sentences", is to train the decoder of neural machine translation with a sentence from a monolingual corpus while setting all the context vectors $\mathbf{c}_t$ (see Eq. (5.3.2)) to all-zero vectors. The second approach uses "synthetic source sentences", where each sentence from a target-side monolingual corpus is translated to a synthetic source sentence by a reverse translation model.

Luong et al. (Luong, Le, Sutskever, Vinyals, and Kaiser, 2015c) proposed a multitask neural machine translation model. In this multi-task model, it is possible to include multiple source languages as well as target languages. They experimented with a setting where monolingual translation paths (sequence autoencoders) were added to a neural translation model. The experiment revealed that the translation quality improves by jointly training the translation and autoencoding paths.

There have been two recent approaches to low-resource translation using neural machine translation. First, Firat et al. (Firat, Cho, and Bengio, 2016) introduced a multi-way, multilingual neural machine translation model with a shared attention mechanism. They empirically showed that the proposed multilingual model was able to translate between up to ten language pair–directions with the translation quality comparable to ten separate single-pair translation models.

Chapter 5

ON INTEGRATING A LANGUAGE MODEL INTO NEURAL MACHINE TRANSLATION

5.1. Introduction

Neural machine translation (NMT) is an end-to-end neural network based approach to statistical machine translation (Kalchbrenner and Blunsom, 2013; Sutskever *et al.*, 2014; Cho *et al.*, 2014a; Bahdanau *et al.*, 2014) which has recently been able to achieve state-of-the-art translation quality on many language pairs (Sutskever *et al.*, 2014; Jean, Cho, Memisevic, and Bengio, 2014; Chung *et al.*, 2016; Luong and Manning, 2016). For the recent advances made since the publication of the initial technical report version of this manuscript (Gulcehre, Firat, Xu, Cho, Barrault, Lin, Bougares, Schwenk, and Bengio, 2015), we refer readers to Sec. 5.2.

A large part of the recent successes of NMT has been due to the availability of large amounts of high quality, sentence aligned corpora. In the case of low resource language pairs however, or in a task with severe domain restrictions, large parallel corpora may not be available. In contrast, monolingual corpora is almost always abundant and universally available. Despite being "unlabeled", monolingual corpora still exhibit rich linguistic structure that can be useful for machine translation.

In this work, we explore the question of how to leverage monolingual corpora. Specifically, we explore two ways of integrating a language model (LM) trained only on monolingual data (target language) into an NMT system. First, by combining the LM and NMT system at the output level (which we term *shallow fusion*). Second, by fusing the LM and NMT at the level of their hidden states, enabling us to combine them nonlinearly (which we term *deep fusion*).

We evaluate the proposed fusion strategies under two distinct settings. The first setting involves low-resource translation tasks (Turkish → English and Chinese → English SMS Chat), where training an NMT system with a small parallel corpus alone often leads to severe overfitting. Second, we further evaluate our methods on higher-resource tasks (German→English and Czech→English). In addition to observing an improvement in translation quality on

the small scale tasks, we observe an improvement in the large scale setting. In addition, our experiments show that our *deep fusion* method of fusing the LM and NMT systems more significantly improves translation quality than the *shallow fusion* method.

In the following section (Sec. 5.2), we review recent work in neural machine translation. We present our basic model architecture in Sec. 7.3 and describe our shallow and deep fusion approaches in Sec. 5.4. Next, we describe our datasets and experiments in Sec. 5.5–5.6.

5.2. Background: Neural Machine Translation

Statistical machine translation (SMT) systems maximize the conditional probability $p(\mathbf{y} \mid \mathbf{x})$ of a correct target translation $\mathbf{y}$ given a source sentence $\mathbf{x}$. This is done by maximizing separately a language model $p(\mathbf{y})$ and the (inverse) translation model $p(\mathbf{x} \mid \mathbf{y})$ component by using Bayes' rule:

$$p(\mathbf{y} \mid \mathbf{x}) \propto p(\mathbf{x} \mid \mathbf{y})p(\mathbf{y}).$$

This decomposition into a language model and translation model are meant to make full advantage of available corpora, i.e. monolingual corpora for fitting the language model and parallel corpora for the translation model. In reality, SMT systems tend to model $\log p(\mathbf{y} \mid \mathbf{x})$ directly by linearly combining multiple features by using a so-called log-linear model:

$$\log p(\mathbf{y} \mid \mathbf{x}) = \sum_j f_j(\mathbf{x}, \mathbf{y}) + C, \tag{5.2.1}$$

where f_j is the j-th feature based on both or either of the source and target sentences, and C is a normalization constant which is often ignored. These features include, for instance, pair-wise statistics between two sentences/phrases. The log-linear model is fitted to data, in most cases, by maximizing an automatic evaluation metric other than an actual conditional probability, such as BLEU.

Neural machine translation, on the other hand, aims at directly optimizing $\log p(\mathbf{y} \mid \mathbf{x})$ jointly with the feature extraction as well as the normalization constant. This is typically done under the encoder-decoder framework (Kalchbrenner and Blunsom, 2013; Cho *et al.*, 2014a; Sutskever *et al.*, 2014) consisting of neural networks. The first network encodes the source sentence $\mathbf{x}$ into a continuous-space representation from which the decoder produces a target translation sentence. By using RNN architectures equipped to learn long-term dependencies such as gated recurrent units (GRU, (Cho *et al.*, 2014a)) or long short-term memory (LSTM, (Hochreiter and Schmidhuber, 1997)), the whole system can be trained end-to-end (Cho *et al.*, 2014a; Sutskever *et al.*, 2014).

Once the model learns the conditional distribution or translation model, given a source sentence we can find a translation that approximately maximizes the conditional probability using, for instance, a beam search algorithm.

5.3. Model Description

We use the RNNSearch (Bahdanau *et al.*, 2014) model that learns to jointly (soft-)align and translate as the baseline neural machine translation system in this paper, we will refer it as "NMT".

The encoder of the NMT is a bidirectional RNN that consists of a forward and a backward RNN (Schuster and Paliwal, 1997). The forward RNN reads the input sentence $\mathbf{x} = (x_1, \ldots, x_T)$ from left to right, resulting in a sequence of hidden states $(\overrightarrow{\mathbf{h}}_1, \ldots, \overrightarrow{\mathbf{h}}_T)$. The backward RNN reads $\mathbf{x}$ in the opposite direction and outputs $(\overleftarrow{\mathbf{h}}_1, \ldots, \overleftarrow{\mathbf{h}}_T)$. We concatenate forward and the backward hidden states at each time step to build a sequence of "*annotation*" vectors $(\mathbf{h}_1, \ldots, \mathbf{h}_T)$, where

$$\mathbf{h}_j^\top = \left[\overleftarrow{\mathbf{h}}_j^\top ; \overrightarrow{\mathbf{h}}_j^\top\right].$$

Each annotation vector $\mathbf{h}_j$ encodes the information about the j-th word with respect to all the other surrounding words in the sentence.

In the decoder, at each time-step t, a soft-alignment mechanism first decides on which annotation vectors are most relevant. To do so, a relevance weight α_{tj} for each of the j-th annotation vector are computed with a feedforward neural network f that takes as input $\mathbf{h}_j$, the previous decoder's hidden state $\mathbf{s}_{t-1}$ and the previous output $\mathbf{y}_{t-1}$:

$$e_{tj} = f(\mathbf{s}_{t-1}, \mathbf{h}_j, \mathbf{y}_{t-1}).$$

The outputs e_{tj} are normalized over the sequence of the annotation vectors:

$$\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{T} \exp(e_{tk})}, \tag{5.3.1}$$

and α_{tj} is a relevance score, or an alignment weight, of the j-th annotation vector.

The relevance scores are used to obtain the *context vector* $\mathbf{c}_t$ of the t-th word at the target:

$$\mathbf{c}_t = \sum_{j=1}^{T} \alpha_{tj} \mathbf{h}_j \ , \tag{5.3.2}$$

Then, the decoder's hidden state $\mathbf{s}_t^{\mathrm{TM}}$ at time t is computed from the previous hidden state $\mathbf{s}_{t-1}^{\mathrm{TM}}$, the context vector $\mathbf{c}_t$ and the previously translated word $\mathbf{y}_{t-1}$:

$$\mathbf{s}_t^{\mathrm{TM}} = f_r(\mathbf{s}_{t-1}^{\mathrm{TM}}, \mathbf{y}_{t-1}, \mathbf{c}_t), \tag{5.3.3}$$

where f_r is the gated recurrent unit (Cho *et al.*, 2014a).

We use a deep output layer (Pascanu, Gulcehre, Cho, and Bengio, 2014) to compute the conditional distribution over words:

$$p(\mathbf{y}_t|\mathbf{y}_{<t}, \mathbf{x}) \propto \mathbf{y}_t^{\top} \exp((\mathbf{W}_o f_o(\mathbf{s}_t^{\mathrm{TM}}, \mathbf{y}_{t-1}, \mathbf{c}_t) + \mathbf{b}_o)), \tag{5.3.4}$$

where $\mathbf{y}_t$ is a one-hot encoded vector indicating one of the words in the target vocabulary. $\mathbf{W}_o$ is a learned weight matrix and $\mathbf{b}_o$ is a bias. f_o is a single-layer feedforward neural network with a two-way maxout non-linearity (Goodfellow, Warde-Farley, Mirza, Courville, and Bengio, 2013a).

The whole model, including both the encoder and decoder, is jointly trained to maximize the (conditional) log-likelihood of the bilingual training corpus:

$$\max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{n=1}^{N} \log p_\theta(\mathbf{y}^{(n)}|\mathbf{x}^{(n)}),$$

where the training corpus is a set of $(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})$'s, and $\boldsymbol{\theta}$ denotes a set of all the tunable parameters.

5.4. Integrating Language Model into the Decoder

We propose two approaches for integrating a neural language model (NLM) into a neural machine translation system. Without loss of generality, we use language models implemented with recurrent neural networks (RNNLM, (Mikolov, Kombrink, Deoras, Burget, and Cernocky, 2011)) which are equivalent to the decoder described in the previous section except that it is not conditioned on a context vector (i.e., $\mathbf{c}_t = 0$ in Eqs. (7.3.2)–(11.3.6)).

In the following sections, we assume that both an NMT model (on parallel corpora) as well as a recurrent neural network language model (RNNLM, on larger monolingual corpora) have been trained separately before integration. We denote the hidden state at time t of the RNNLM with $\mathbf{s}_t^{\mathrm{LM}}$.

5.4.1. Shallow Fusion

Shallow fusion is inspired by the way that monolingual language models are incorporated in the decoders of conventional SMT systems (Koehn, 2010; Mohit, Hwa, and Lavie, 2010).

At each time step, a translation model proposes a set of candidate words. The candidates are then scored according to the weighted sum of the scores given by the translation model and the language model.

More specifically, at each time step t, the translation model (in this case, the NMT) computes the score of every possible next word for all the hypotheses $\left\{\mathbf{y}_{\leq t-1}^{(i)}\right\}$. Score of each prediction is a summation of the score of the hypothesis and the score given by the NMT for the next word. All these new hypotheses (a hypothesis from the previous timestep with a next word appended at the end) are then sorted according to their respective scores, and the top K ones are selected as candidates $\left\{\hat{\mathbf{y}}_{\leq t}^{(i)}\right\}_{i=1,\dots,K}$.

We then rescore these hypotheses with the weighted sum of the scores by the NMT and RNNLM, where we only need to recompute the score of the "new word" at the end of each candidate hypothesis. The score of the new word is computed according to the Eq. 5.4.1,

$$\log p(\mathbf{y}_t = k) = \log p_{\text{TM}}(\mathbf{y}_t = k) + \beta \log p_{\text{LM}}(\mathbf{y}_t = k), \tag{5.4.1}$$

where β is a scalar hyper-parameter that needs to be tuned to maximize the translation performance on a development set.

See Fig. 5.1 (a) for illustration.

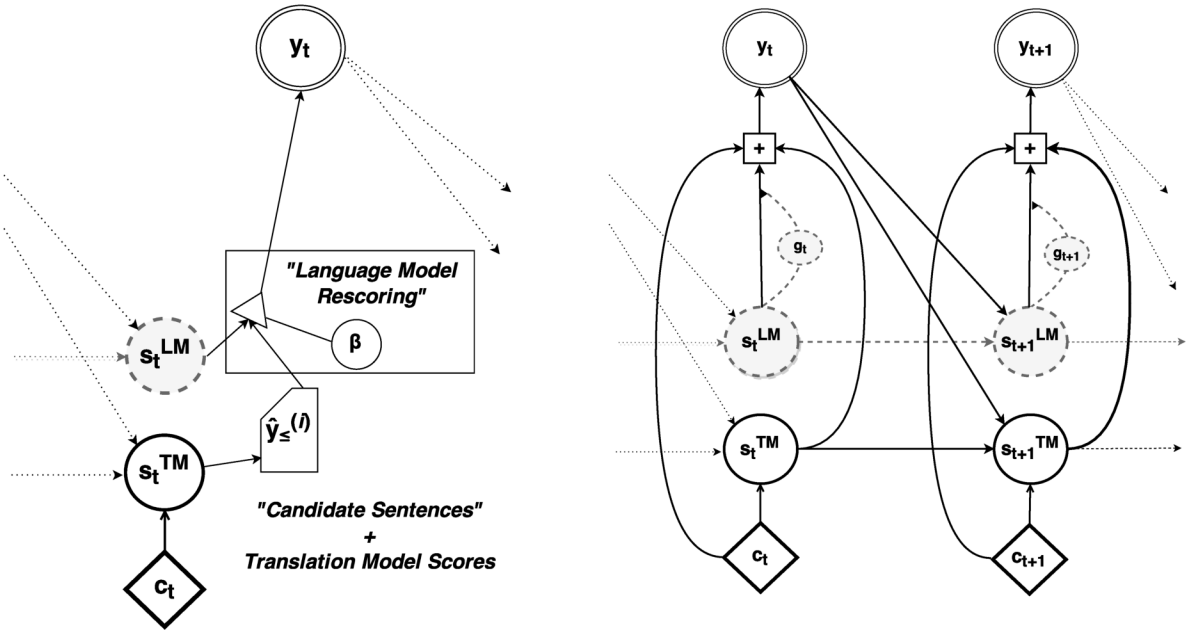
5.4.2. Deep Fusion

In deep fusion, we integrate the RNNLM to the decoder of the NMT by concatenating their hidden states (see Fig. 5.1 (b)). The model is then finetuned to use the hidden states from both of these models when computing the output probability of the next word (see Eq. (11.3.6)). Unlike a vanilla NMT system (without any language model component), the hidden layer of the deep output takes as input the hidden state of the RNNLM in addition to the NMT's decoder, the previous word, and the context such that

$$p(\mathbf{y}_t|\mathbf{y}_{<t}, \mathbf{x}) \propto \exp(\mathbf{y}_t^\top(\mathbf{W}_o f_o(\mathbf{s}_t^{\text{LM}}, \mathbf{s}_t^{\text{TM}}, \mathbf{y}_{t-1}, \mathbf{c}_t) + \mathbf{b}_o)), \tag{5.4.2}$$

where we use the superscripts $^{\text{LM}}$ and $^{\text{TM}}$ to denote the hidden states of the RNNLM and NMT respectively.

During the finetuning, we only update the parameters that are used to parameterize the output from Eq. (5.4.2). This is to ensure that the features learned by the LM from monolingual corpora are not overwritten. It is possible to use monolingual corpora as well when we finetune both models together, but in this paper, we only finetune the output weights using only the parallel corpora.



(a) Shallow Fusion (Sec. 5.4.1) (b) Deep Fusion (Sec. 5.4.2)

FIG. 5.1. Graphical illustrations of the proposed fusion methods. In shallow fusion (a), candidates are scored according to the weighted sum of the scores given by the translation model and the language model during the beam-search. For deep fusion (b), we merge the hidden representation of the NMT's decoder $\mathbf{s}_t^{\text{TM}}$ and the neural language model's $\mathbf{s}_t^{\text{LM}}$ before predicting each word at every time-step and the controller g_t rescales the contribution coming from the LM.

5.4.2.1. *Balancing the LM and TM*

In order to flexibly balance the contributions of hidden representations from both the RNNLM and NMT at the decoder, we use a "controller" mechanism. The need to flexibly balance the signals arises depending on the sentence being translated. For instance, in the case of Zh→En, there are no Chinese words that correspond to articles in English, in which case RNNLM may be more informative. On the other hand, if a noun needs to be translated, it may be better to ignore any signal from the LM, as it may prevent the decoder from choosing the correct translation. Intuitively, controller mechanism helps the model to dynamically weigh the input from each model depending on the word being translated.

The controller mechanism is implemented as a function taking the hidden state of the LM as input and computing

$$g_t = \sigma\left(\mathbf{v}_g^\top \mathbf{s}_t^{\text{LM}} + b_g\right), \tag{5.4.3}$$

	Chinese	English
# of Sentences	436K	
# of Unique Words	21K	150K
# of Total Words	8.4M	5.9M
Avg. Length	19.3	13.5

(a) Zh→En

	Turkish	English
# of Sentences	361K	
# of Unique Words	96K*	95K
# of Total Words	11.4M*	8.1M
Avg. Length	31.6	22.6

(b) Tr→En

	Czech	English
# of Sentences	12.1M	
# of Unique Words	1.5M	911K
# of Total Words	151M	172M
Avg. Length	12.5	14.2

(c) Cs→En

	German	English
# of Sentences	4.1M	
# of Unique Words	1.16M†	742K
# of Total Words	11.4M†	8.1M
Avg. Length	24.2	25.1

(d) De→En

TAB. 5.1. Statistics of the Parallel Corpora. $\star$: After segmentation, †: After compound splitting. The segmentation on Turkish sentences and the compound splitting on German result in longer sentences.

where σ is a logistic sigmoid function. $\mathbf{v}_g$ and b_g are learned parameters.

The output of the controller is then multiplied with the hidden state of the LM. This lets the decoder use the signal from the NMT fully while the controller adjusts the magnitude of the signal coming from LM.

In our experiments, we empirically found that it was better to initialize the bias $\mathbf{b}_g$ to a negative value. This allows the decoder to decide the importance of the LM only when it is deemed necessary.[1]

5.5. Datasets

We evaluate the proposed approaches for integrating monolingual LM into NMT on four different language pairs: Chinese to English (Zh→En), Turkish to English (Tr→En), German to English (De→En) and Czech to English (Cs→En). We consider Tr→En (IWSLT'14) and Zh→En (OpenMT'15) tasks as low-resource translation tasks. By including De→En and Cs→En WMT'15 translation tasks in our experiments, we evaluate the performance of our proposed model on high-resource translation tasks as well. In the following sections, we describe each one of these datasets in more details.

[1] In all our experiments, we set $\mathbf{b}_g = -1$ to ensure that $\mathbf{g}_t$ is initially 0.26 on average.

5.5.1. Parallel Corpora

Zh→En: OpenMT'15

In our experiments, we use the parallel corpora that are provided as part of the NIST OpenMT'15 Challenge. Sentence-aligned pairs from three domains are combined to form a training set: (1) SMS/CHAT and (2) conversational telephone speech (CTS) from DARPA BOLT Project, and (3) newsgroups/weblogs from DARPA GALE Project. In total, the training set contains 430K sentence pairs (see Table 5.1 for the detailed statistics). We train models with this training set and the development set (the concatenation of the provided development and tune sets from the challenge) and evaluate them on the test set. The domain of the development and test sets is restricted to CTS.

Preprocessing

Importantly, we did *not segment* the Chinese sentences and considered each character as a symbol, unlike other approaches which use a separate segmentation tool to segment Chinese characters into words. Any consecutive non-Chinese characters such as Latin alphabets were, however, considered as an individual word. The only preprocessing we did on the English side of the corpus was simple tokenization using the tokenizer from Moses.[2]

Tr→En: IWSLT'14

We used the WIT parallel corpus (Cettolo, Girardi, and Federico, 2012) and SETimes parallel corpus made available as a part of IWSLT'14 (machine translation track). The corpus consists of the sentence-aligned subtitles of TED and TEDx talks, and we concatenated dev2010 and tst2010 to form a development set, and tst2011, tst2012, tst2013 and tst2014 to form a test set.

Preprocessing

As done in the case of Zh→En, initially we removed all special symbols from the corpora and tokenized the Turkish side with the tokenizer provided by Moses. We segmented each Turkish sentence into a sequence of sub-word units using Zemberek[3] followed by morphological disambiguation on the morphological analysis (Sak, Güngör, and Saraçlar, 2007) to overcome the exploding vocabulary due to the rich inflections and derivations in Turkish. We removed any non-surface morphemes corresponding to, for instance, part-of-speech tags.

[2] `https://github.com/moses-smt/mosesdecoder/blob/master/scripts/tokenizer/tokenizer.perl`
[3] `https://github.com/ahmetaa/zemberek-nlp`

5.5.2. Cs→En and De→En: WMT'15

For the training of our models, we used all the available training data provided for Cs→En and De→En in the WMT'15 competition. We used newstest2013 as a development set and newstest2014 for a test set.

Preprocessing

We tokenized the datasets with the tokenizer from Moses decoder first. Sentences longer than eighty words and those that have large mismatch between the lengths of source and target sentences were removed from the training set. Then, we filtered the training data by removing sentence pairs in which one sentence (or both) was written in a wrong language by using a language detection toolkit from (Shuyo, 2010), unless the sentence had 5 words or less. For De→En, we also split compounds in the German side by using Moses.

5.5.3. Monolingual Corpora

The English Gigaword corpus by the Linguistic Data Consortium (LDC), which mainly consists of newswire documents was allowed in both OpenMT'15 and IWSLT'14 challenges for language modeling. We used the tokenized Gigaword corpus without any other preprocessing step to train three different RNNLM's to fuse into NMT for Zh→En, Tr→En and the WMT'15 translation tasks (De→En and Cs→En).

5.6. Experimental Settings

We compare results of each model after finetuning their hyperparameters for each language pair separately on the development set. We treat the type of optimization algorithm as a hyperparameter of the model, and as a result, use a different optimization algorithm for each model.

5.6.1. Training Procedure

5.6.1.1. *Neural Machine Translation*

The input and output of the network are sequences of one-hot vectors whose dimensionality correspond to the sizes of the source and target vocabularies, respectively. We constructed the vocabularies with the most common words in the parallel corpora.Each word was projected into a 620-dimensional word embedding for both the encoder and decoder. We chose the size of the recurrent units for Zh→En and Tr→En to be $1,200$ and $1,000$ respectively. The sizes

of the vocabularies for Chinese, Turkish and English were 10K, 30K, and 40K, respectively, for the Tr→En and Zh→En tasks. For our Cs→En and De→En dataset, we employed the importance sampling based technique introduced in Jean *et al.* (2014) in order to train a model with a large vocabulary (size $200k$).

Each model was optimized using Adadelta (Zeiler, 2012) with minibatches of 80 sentence pairs. At each update, we normalized the gradient such that if the L_2 norm of the gradient exceeds 5, the gradients are renormalized to have a magnitude of 5 (Pascanu *et al.*, 2013a). For the non-recurrent layers (see Eq. (11.3.6)), we used dropout (Hinton, Srivastava, Krizhevsky, Sutskever, and Salakhutdinov, 2012) and additive Gaussian noise (mean 0 and std. dev. 0.001) on each parameter (Graves, 2011) to prevent overfitting. The training set was shuffled at the end of every epoch. Training was early-stopped to maximize the performance of the development set measured by BLEU.[4] We initialized all recurrent weight matrices as random orthonormal matrices.

5.6.1.2. *Language Model*

We trained three RNNLM's with $2,400$ long short-term memory (LSTM, (Hochreiter and Schmidhuber, 1997)) units on English Gigaword Corpus using the vocabularies constructed separately from the English sides of Zh→En and Tr→En corpora respectively. The third language model was trained using $2,000$ LSTM units on the English Gigaword Corpus again but with a vocabulary constructed from the intersection the English sides of Cs→En and De→En. The parameters of the Zh→En and Tr→En language models were optimized using RMSProp (Tieleman and Hinton, 2012), and the Adam optimizer (Kingma and Welling, 2014) was used for the shared Cs→En and De→En model. Any sentence with more than ten percent of its words are out of vocabulary was discarded from the training set. We early-stopped any training run based upon the perplexity of development set.

5.6.2. Shallow and Deep Fusion

5.6.2.1. *Shallow Fusion*

The hyperparameter β in Eq. (5.4.1) was selected to maximize the translation performance on the development set, from the range between 0.001 and 0.1 via a grid search (larger β's did not help.) In our preliminary experiments, we found that it is important to re-normalize the softmax of the LM without the end-of-sequence and out-of-vocabulary symbols ($\beta = 0$

[4] We compute the BLEU score using the multi-blue.perl script from Moses on tokenized sentence pairs for early stopping.

in Eq. (5.4.1).) We speculate that this may be due to the difference in sentence lengths in addition to domain differences.

5.6.2.2. *Deep Fusion*

We experimented with different learning algorithms in order to obtain the optimal performance for our model. The parameters of the deep output layer (Eq. (5.4.2)) as well as the controller (see Eq. (5.4.3)) were finetuned for Zh→En using Adam adaptive learning rate algorithm, and RMSProp with momentum for Tr→En. During the finetuning, the dropout probability and the standard deviation of the weight noise were set to 0.56 and 0.005, respectively. Based on our preliminary experiments, we reduced the level of regularization after the first $10K$ updates. In Cs→En and De→En tasks with large vocabularies, the model parameters were finetuned using Adadelta while scaling down the magnitude of the update steps by 0.01.

5.6.2.3. *Handling Rare Words*

On the De→En and Cs→En translation tasks, we replaced the unknown words generated by the NMT with the words that the NMT assigned highest relevance weight to in the source sentence (Eq. (5.3.1).) We copied the selected source word in the place of the corresponding unknown token in the target sentence. This method is similar to the technique from Luong et. al. (Luong, Sutskever, Le, Vinyals, and Zaremba, 2014) on addressing rare words problem. Instead of relying on an external alignment tool, we used the attention mechanism of the NMT model to extract alignments.

5.7. Results and Analysis

In our experiments, we first provide results on tasks that have low-resources, such as Tr→En IWSLT '14 and Zh→En OpenMT'15 (SMS and call center transcripts.) We also provide results on high-resource tasks, such as De→En and Cs→En, showing that incorporating language models to the decoder of an NMT system can improve the results of these models as well.

5.7.1. Zh→En: OpenMT'15

As a baseline, we trained phrase-based and hierarchical SMT systems (Koehn, Och, and Marcu, 2003; Chiang, 2005) with/without re-scoring by an external neural language model (CSLM, (Schwenk, 2007)), in addition to the NMT based systems. We present the results in Table 5.2.

	SMS/CHAT		CTS	
	Dev	Test	Dev	Test
PB	15.5	14.73	21.94	21.68
+ CSLM	16.02	15.25	23.05	22.79
HPB	15.33	14.71	21.45	21.43
+ CSLM	15.93	15.8	22.61	22.17
NMT	17.32	17.36	23.4	**23.59**
Shallow	16.51	16.69	23.09	23.5
Deep	**17.58**	**17.64**	**23.78**	23.5

TAB. 5.2. Results on the task of Zh→En PB and HPB stand for the phrase-based and hierarchical phrase-based SMT systems, respectively. The results reported in this table are obtained with multi-bleu.perl script.

We observed that integrating a monolingual LM with deep fusion (see Sec. 5.4.2) improved performance in general, except for the CTS task. We noticed that the NMT-based models, regardless of whether the LM was integrated or not, outperformed the phrase-based SMT systems.

5.7.2. Tr→En: IWSLT'14

In Table 5.3, we present our results on Tr→En. Compared to Zh→En, we saw a greater performance improvement (+1.19 BLEU) from the basic NMT to the NMT integrated with the LM under the proposed method of deep fusion. Furthermore, by incorporating the LM using the deep fusion, the NMT systems were able to outperform the best previously reported result (Yılmaz, El-Kahlout, Aydın, Ozil, and Mermer, 2013) by up to +2.0 BLEU points.

	Development Set		Test Set			
	dev2010	**tst2010**	**tst2011**	**tst2012**	**tst2013**	**Test 2014**
Previous Best (Single)	**15.33**	17.14	18.77	18.62	18.88	-
Previous Best (Combination)	-	17.34	18.83	18.93	18.70	-
NMT	14.05 (14.50)	17.64 (18.01)	17.84 (18.40)	18.20 (18.77)	19.50 (19.86)	18.09 (18.64)
NMT+LM (Shallow)	14.08 (14.44)	17.65 (17.99)	17.82 (18.48)	18.20 (18.80)	19.53 (19.87)	18.22 (18.66)
NMT+LM (Deep)	15.30 (15.69)	**18.92** (19.34)	**19.50** (20.17)	**19.57** (20.23)	**20.92** (21.34)	**20.04** (20.56)

TAB. 5.3. We provide the results obtained with both multi-bleu.perl (in paranthesis) and mteval-v13a.pl scripts. On all the test sets, our deep fusion approach outperforms the other methods.

	De→En		Cs→En	
	Dev	Test	Dev	Test
NMT Baseline	25.51	23.61	21.47	21.89
Shallow Fusion	25.53	23.69	21.95	22.18
Deep Fusion	**25.88**	**24.00**	**22.49**	**22.36**

TAB. 5.4. Results for De→En and Cs→En translation tasks on WMT'15 dataset. Although Cs→En and De→En models are trained on high-resource corpora, we observe substantial improvements on the test-sets of both corpora. The results in this table are obtained with multi-blue.perl script.

5.7.3. Cs→En and De→En: WMT-15

We provide the results for Cs→En and De→En in Table 5.4. Shallow fusion achieved +0.09 and +0.29 BLEU score improvements respectively on De→En and Cs→En over the baseline NMT model. We observed 0.39 and 0.47 BLEU score improvements over the NMT baseline by using deep fusion.

5.7.4. Analysis: Effect of Language Model

Our results that we report in this paper suggest a heavy dependency on the degree of similarity between the domain of monolingual corpora and the target domain of translation for performance improvement.

In the case of Zh→En, intuitively, we can tell that the style of writing in both SMS/CHAT, as well as the conversational speech will be different from that of news articles (which constitutes the majority of the English Gigaword corpus.) Empirically, this is reflected in the high development set perplexity under our LM. For instance, see the column Zh→En of Table 5.5. This explains the marginal improvement we observed in Sec. 5.7.1.

On the other hand, in the case of Tr→En, the similarity between the domains of the monolingual corpus and parallel corpora is higher (see the column Tr→En of Table 5.5.) This led to a significantly larger improvement in translation performance by integrating the monolingual language model than the case of Zh→En. Similarly, we observed the improvement by both shallow and deep fusion in the case of De→En and Cs→En, where the perplexity on the development set was much lower.

Unlike shallow fusion, deep fusion allows a model to selectively incorporate the information from the additional LM by the controller mechanism from Sec. 5.4.2.1. Although this controller mechanism works on per-word basis, it can be expected that if the LM is a better model of the target domain, the controller mechanism will be more frequently active on average,

i.e., $\mathrm{E}[g_t] \gg 0$. From Table 5.5, we can see that, on average, the controller mechanism is most active with De→En and Cs→En, where the additional LM was able to model the target sentences best. This effectively means that the deep fusion allows the model to be more robust to the domain mismatch between the TM and LM, thus explaining why deep fusion was more successful than shallow fusion our the experiments.

	Zh→En	Tr→En	De→En	Cs→En
Perplexity	223.68	163.73	78.20	78.20
Average g	0.23	0.12	0.28	0.31
Std Dev g	0.0009	0.02	0.003	0.008

TAB. 5.5. Perplexity of RNNLM's on the development sets and the statistics of the controller gating mechanism g. The higher perplexities observed in Zh-En and Tr-En are due to the domain mismatch in Zh-En and Tr-En.

5.8. A Review of Recent Advances in NMT

Since we proposed the fusion methods in the earlier technical report Gulcehre *et al.* (2015), many advances have been made in the field of neural machine translation, more specifically in the context of integrating monolingual corpora and low-resource translation. In this section, we review some of the related works and their relation to our proposal.

Incorporating Monolingual Corpora

Sennrich et al. (Sennrich *et al.*, 2015) proposed two strategies for low resource neural machine translation by making use of monolingual data. The first approach, called "dummy source sentences", is to train the decoder of neural machine translation with a sentence from a monolingual corpus while setting all the context vectors $\mathbf{c}_t$ (see Eq. (5.3.2)) to all-zero vectors. The second approach uses "synthetic source sentences", where each sentence from a target-side monolingual corpus is translated to a synthetic source sentence by a reverse translation model. These pseudo-parallel sentence pairs are mixed into the existing parallel corpus and used to train an neural translation model. Both of these approaches, and especially the latter approach based on synthetic source sentences, were shown to improve the translation quality significantly on Tr-En, En-De and De-En. Their approaches are orthogonal to the ones proposed in this paper, and it is left as a future work to investigate the combination of these approaches.

Luong et al. (Luong *et al.*, 2015c) proposed a multitask neural machine translation model. In this multi-task model, it is possible to include multiple source languages as well as target languages. They experimented with a setting where monolingual translation paths (sequence

autoencoders) were added to a neural translation model. The experiment revealed that the translation quality improves by jointly training the translation and autoencoding paths. Their work is however limited to a simple encoder-decoder model without the attention mechanism which has proven to be crucial in getting a good neural translation model. This approach based on multiple tasks is orthogonal to the proposed deep and shallow fusion techniques as well.

Low-Resource Translation

There have been two recent approaches to low-resource translation using neural machine translation. First, Firat et al. (Firat *et al.*, 2016) introduced a multi-way, multilingual neural machine translation model with a shared attention mechanism. They empirically showed that the proposed multilingual model was able to translate between up to ten language pair–directions with the translation quality comparable to ten separate single-pair translation models. In their paper, they observed that the quality of low-resource translation was improved when trained as a part of the multilingual model. It will be an interesting future research direction in which the fusion techniques proposed here and the multi-way, multilingual model are combined.

More recently, Zoph et al. (Zoph, Deniz, Jonathan, and Knight, 2016) demonstrated a transfer learning approach to low-resource translation. They first train a neural machine translation model with a large parallel corpus whose target-side matches that of the target low-resource language pair. Then, this model, or its part, is further finetuned on a small parallel corpus of the target task. They observed significant improvements with a variety of languages, including Hausa, Turkish, Uzbek and Urdu, when the model pretrained with Fr-En was used as an initial point. Again, it is certainly possible to incorporate the proposed fusion techniques with this transfer learning approach, and we leave it as a future work.

5.9. Conclusion and Future Work

In this paper, we propose and compare two methods–shallow and deep fusion– for incorporating monolingual corpora into an existing neural translation system. We empirically evaluate these approaches on low-resource translation (Tr→En), focused-domain translation (Zh→En, SMS/Chat and conversational speech) and high-resource translation (Cs→En and De→En) tasks. On the first two tasks, we observed that the proposed deep fusion improves the translation quality by up to +2.0 BLEU against the existing phrase-based statistical machine translation systems. On the latter two, we observed up to +0.47 BLEU improvement over the neural translation baseline without using any monolingual corpus.

Our analysis revealed that the performance improvement by incorporating an external language model highly depends on the similarity between the domains of the monolingual corpus and the target language. When the domains match well, as in the cases of De-En and Cs-En, both the shallow and deep fusion methods worked well. On the other hand, the deep fusion significantly outperformed the shallow fusion when the domains did not match well (Zh-En). We conjecture that this is due to the adaptive mechanism built into the deep fusion that allows the neural translation model to selectively incorporate information from the language model.

Chapter 6

PROLOGUE TO THE SECOND ARTICLE

6.1. Article Details

Pointing the unknown words, Gulcehre, Caglar, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio, ACL 2016.

Personal Contributions:

I came up with the idea of using a pointing or copying mechanism after reading the paper on learning algorithmic tasks by using sequence to sequence models with a pointing mechanism as proposed by Vinyals, Fortunato, and Jaitly (2015) in the summer of 2015 and shared this idea with my supervisor Yoshua Bengio in the form of a draft. Yoshua gave me useful feedback on how to train the model easily. In October 2015, I have started an internship at IBM and decided to work on this idea during my internship. However, throughout my internship in the first few months, I had to focus on a different project. However, in the last few weeks of my internship, I have implemented the algorithm in the paper and started experimenting with that on some toy tasks, including on the bAbI (Weston, Bordes, Chopra, and Mikolov, 2015b) dataset for episodic QA. Ramesh Nallapati implemented the pointer Softmax for summarization and has run the experiments in the paper. I had a separate implementation from Ramesh in Montreal and run the experiments on NMT and the toy task in the paper. Sungjin Ahn and I have written the large part of the paper. Ramesh Nallapati wrote the large part of the section on Summarization experiments.

6.2. Context

Luong, Sutskever, Le, Vinyals, and Zaremba (2015a) have proposed an effective method to address the rare-world problem by using an external alignment tool and Vinyals *et al.* (2015) proposed a method to predict the location of an input used to predict the next word. We have decided to combine both of these ideas to create a generic algorithm that can learn to point at the particular words in the context to predict the output, considering that the rare words to be predicted usually exist in the context sequence in most NLP tasks. Our

approach uses a learned gating mechanism that gates the decision of whether to predict the word from a short list of words or just copying from the source context.

In parallel to our work, Gu, Lu, Li, and Li (2016) and Cheng and Lapata (2016) proposed different approaches to learn to copy the rare words from source to target and both papers analyzed their models on summarization tasks.

Our paper is published at ACL 2016 which is one of the most known conference in the field of natural language processing. The paper received a wide range of interest from NLP community and the method is widely adopted and it has more than 60 citations right now.

6.3. Contributions

We proposed a novel way to deal with the rare-words or unseen words during the training of the NMT model. Our model basically uses two softmax layers and depending on the context it decides to use either to predict the location of the word from the source side or directly predict the word itself from the shortlist.We observed improvements on both NMT and Summarization tasks.

6.4. Recent Developments

The pointing mechanism for the NLP introduced in our paper has been successful and it got adopted into many different applications of NLP. Text summarization has been a very popular application of this (See, Liu, and Manning, 2017; Nallapati *et al.*, 2016b).

Recently (Merity, Xiong, Bradbury, and Socher, 2016) has extended our approach to unconditional language modelling tasks. Grave, Joulin, and Usunier (2016) proposed a method using a neural cache to deal with the problem of rare words.

Extending generative models with an external memory has also been shown to improve the quality of the generated samples of those models (Gemici, Hung, Santoro, Wayne, Mohamed, Rezende, Amos, and Lillicrap, 2017).

Chapter 7

POINTING THE UNKNOWN WORDS

7.1. Introduction

Words are the basic input/output units in most of the NLP systems, and thus the ability to cover a large number of words is a key to building a robust NLP system. However, considering that (i) the number of all words in a language including named entities is very large and that (ii) language itself is an evolving system (people create new words), this can be a challenging problem.

A common approach followed by the recent neural network based NLP systems is to use a softmax output layer where each of the output dimension corresponds to a word in a predefined word-shortlist. This is because computing high dimensional softmax is computationally expensive, in practice the shortlist is limited to have only top-K most frequent words in the training corpus. All other words are then replaced by a special word, called the *unknown word (UNK)*.

The shortlist approach has two fundamental problems. The first problem, which is known as the *rare word* problem, is that some of the words in the shortlist occur less frequently in the training set and thus are difficult to learn a good representation, resulting in poor performance. Second, it is obvious that we can lose some important information by mapping different words to a single dummy token *UNK*. Even if we have a very large shortlist including all unique words in the training set, it does not necessarily improve the test performance, because there still exists a chance to see an unknown word at test time. This is known as the *unknown word* problem. In addition, increasing the shortlist size mostly leads to increasing rare words due to Zipf's Law.

These two problems are particularly critical in language understanding tasks such as factoid question answering Bordes, Usunier, Chopra, and Weston (2015) where the words that we are interested in are often named entities which are usually unknown or rare words.

In a similar situation, where we have a limited information on how to call an object of interest, it seems that humans (and also some primates) have an efficient behavioral

mechanism of drawing attention to the object: *pointing* Matthews, Behne, Lieven, and Tomasello (2012). Pointing makes it possible to deliver information and to associate context to a particular object without knowing how to call it. In particular, human infants use pointing as a fundamental communication tool Tomasello, Carpenter, and Liszkowski (2007).

In this paper, inspired by the pointing behavior of humans and recent advances in the attention mechanism Bahdanau, Cho, and Bengio (2015) and the pointer networks Vinyals *et al.* (2015), we propose a novel method to deal with the rare or unknown word problem. The basic idea is that we can see in many NLP problems as a task of predicting target text given context text, where some of the target words appear in the context as well. We observe that in this case we can make the model *learn to point* a word in the context and copy it to the target text, as well as *when to point*. For example, in machine translation, we can see the source sentence as the context, and the target sentence as what we need to predict. In Figure 7.1, we show an example depiction of how words can be copied from source to target in machine translation. Although the source and target languages are different, many of the words such as named entities are usually represented by the same characters in both languages, making it possible to copy. Similarly, in text summarization, it is natural to use some words in the original text in the summarized text as well.

Specifically, to predict a target word at each timestep, our model first determines the source of the word generation, that is, whether to take one from a predefined shortlist or to copy one from the context. For the former, we apply the typical softmax operation, and for the latter, we use the attention mechanism to obtain the pointing softmax probability over the context words and pick the one of high probability. The model learns this decision so as to use the pointing only when the context includes a word that can be copied to the target. This way, our model can predict even the words which are not in the shortlist, as long as it appears in the context. Although some of the words still need to be labeled as *UNK*, i.e., if it is neither in the shortlist nor in the context, in experiments we show that this *learning when and where to point* improves the performance in machine translation and text summarization.

The rest of the paper is organized as follows. In the next section, we review the related works including pointer networks and previous approaches to the rare/unknown problem. In Section 3, we review the neural machine translation with attention mechanism which is the baseline in our experiments. Then, in Section 4, we propose our method dealing with the rare/unknown word problem, called the **Pointer Softmax (PS)**. The experimental results are provided in the Section 5 and we conclude our work in Section 6.

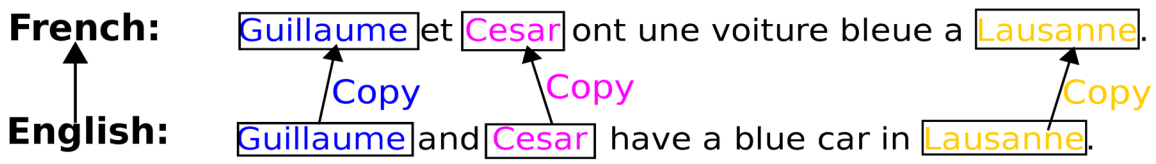


FIG. 7.1. An example of how copying can happen for machine translation. Common words that appear both in source and the target can directly be copied from input to source. The rest of the unknown in the target can be copied from the input after being translated with a dictionary.

7.2. Related Work

The attention-based pointing mechanism is introduced first in the pointer networks Vinyals *et al.* (2015). In the pointer networks, the output space of the target sequence is constrained to be the observations in the input sequence (not the input space). Instead of having a fixed dimension softmax output layer, softmax outputs of varying dimension is dynamically computed for each input sequence in such a way to maximize the attention probability of the target input. However, its applicability is rather limited because, unlike our model, there is no option to choose whether to point or not; it always points. In this sense, we can see the pointer networks as a special case of our model where we always choose to point a context word.

Several approaches have been proposed towards solving the rare words/unknown words problem, which can be broadly divided into three categories. The first category of the approaches focuses on improving the computation speed of the softmax output so that it can maintain a very large vocabulary. Because this only increases the shortlist size, it helps to mitigate the unknown word problem, but still suffers from the rare word problem. The hierarchical softmax Morin and Bengio (2005), importance sampling Bengio and Senécal (2008); Jean *et al.* (2014), and the noise contrastive estimation Gutmann and Hyvärinen (2012); Mnih and Kavukcuoglu (2013) methods are in the class.

The second category, where our proposed method also belongs to, uses information from the context. Notable works are Luong *et al.* (2015a) and Hermann *et al.* (2015). In particular, applying to machine translation task, Luong *et al.* (2015a) learns to point some words in source sentence and copy it to the target sentence, similarly to our method. However, it does not use attention mechanism, and by having fixed sized softmax output over the relative pointing range (e.g., -7, . . . , -1, 0, 1, . . . , 7), their model (the Positional All model) has a limitation in applying to more general problems such as summarization and question

answering, where, unlike machine translation, the length of the context and the pointing locations in the context can vary dramatically. In question answering setting, Hermann *et al.* (2015) have used placeholders on named entities in the context. However, the placeholder id is directly predicted in the softmax output rather than predicting its location in the context.

The third category of the approaches changes the unit of input/output itself from words to a smaller resolution such as characters Graves *et al.* (2013) or bytecodes Sennrich *et al.* (2015); Gillick, Brunk, Vinyals, and Subramanya (2015). Although this approach has the main advantage that it could suffer less from the rare/unknown word problem, the training usually becomes much harder because the length of sequences significantly increases.

Simultaneously to our work, Gu *et al.* (2016) and Cheng and Lapata (2016) proposed models that learn to copy from source to target and both papers analyzed their models on summarization tasks.

7.3. Neural Machine Translation Model with Attention

As the baseline neural machine translation system, we use the model proposed by Bahdanau *et al.* (2015) that learns to (soft-)align and translate jointly. We refer this model as NMT.

The encoder of the NMT is a bidirectional RNN Schuster and Paliwal (1997). The forward RNN reads input sequence $\mathbf{x} = (x_1, \ldots, x_T)$ in left-to-right direction, resulting in a sequence of hidden states $(\overrightarrow{\mathbf{h}}_1, \ldots, \overrightarrow{\mathbf{h}}_T)$. The backward RNN reads $\mathbf{x}$ in the reversed direction and outputs $(\overleftarrow{\mathbf{h}}_1, \ldots, \overleftarrow{\mathbf{h}}_T)$. We then concatenate the hidden states of forward and backward RNNs at each time step and obtain a sequence of *annotation* vectors $(\mathbf{h}_1, \ldots, \mathbf{h}_T)$ where $\mathbf{h}_j = \left[\overrightarrow{\mathbf{h}}_j || \overleftarrow{\mathbf{h}}_j\right]$. Here, $||$ denotes the concatenation operator. Thus, each annotation vector $\mathbf{h}_j$ encodes information about the j-th word with respect to all the other surrounding words in both directions.

In the decoder, we usually use gated recurrent unit (GRU) Cho *et al.* (2014a); Chung, Gülçehre, Cho, and Bengio (2014). Specifically, at each time-step t, the soft-alignment mechanism first computes the relevance weight e_{tj} which determines the contribution of annotation vector $\mathbf{h}_j$ to the t-th target word. We use a non-linear mapping f (e.g., MLP) which takes $\mathbf{h}_j$, the previous decoder's hidden state $\mathbf{s}_{t-1}$ and the previous output y_{t-1} as input:

$$e_{tj} = f(\mathbf{s}_{t-1}, \mathbf{h}_j, y_{t-1}).$$

The outputs e_{tj} are then normalized as follows:

$$l_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{T} \exp(e_{tk})}. \tag{7.3.1}$$

We call l_{tj} as the relevance score, or the alignment weight, of the j-th annotation vector.

The relevance scores are used to get the *context vector* $\mathbf{c}_t$ of the t-th target word in the translation:

$$\mathbf{c}_t = \sum_{j=1}^{T} l_{tj}\mathbf{h}_j \ ,$$

The hidden state of the decoder $\mathbf{s}_t$ is computed based on the previous hidden state $\mathbf{s}_{t-1}$, the context vector $\mathbf{c}_t$ and the output word of the previous time-step y_{t-1}:

$$\mathbf{s}_t = f_r(\mathbf{s}_{t-1}, y_{t-1}, \mathbf{c}_t), \tag{7.3.2}$$

where f_r is GRU.

We use a deep output layer Pascanu *et al.* (2013a) to compute the conditional distribution over words:

$$\begin{aligned} p(y_t = a|y_{<t}, \mathbf{x}) \propto \\ \exp\left(\psi^a_{(\mathbf{W}_o,\mathbf{b}_o)} f_o(\mathbf{s}_t, y_{t-1}, \mathbf{c}_t)\right), \end{aligned} \tag{7.3.3}$$

where $\mathbf{W}$ is a learned weight matrix and $\mathbf{b}$ is a bias of the output layer. f_o is a single-layer feed-forward neural network. $\psi_{(\mathbf{W}_o,\mathbf{b}_o)}(\cdot)$ is a function that performs an affine transformation on its input. And the superscript a in ψ^a indicates the a-th column vector of ψ.

The whole model, including both the encoder and the decoder, is jointly trained to maximize the (conditional) log-likelihood of target sequences given input sequences, where the training corpus is a set of $(\mathbf{x}_n, \mathbf{y}_n)$'s. Figure 7.2 illustrates the architecture of the NMT.

7.4. The Pointer Softmax

In this section, we introduce our method, called as the pointer softmax (PS), to deal with the rare and unknown words. The pointer softmax can be an applicable approach to many NLP tasks, because it resolves the limitations about unknown words for neural networks. It can be used in parallel with other existing techniques such as the large vocabulary trick Jean *et al.* (2014). Our model learns two key abilities jointly to make the pointing mechanism applicable in more general settings: (i) to predict whether it is required to use the pointing or not at each time step and (ii) to point any location of the context sequence whose length can vary widely over examples. Note that the pointer networks Vinyals *et al.* (2015) are in lack of the ability (i), and the ability (ii) is not achieved in the models by Luong *et al.* (2015a).

To achieve this, our model uses two softmax output layers, the *shortlist* softmax and the *location* softmax. The shortlist softmax is the same as the typical softmax output layer where each dimension corresponds a word in the predefined word shortlist. The location softmax is

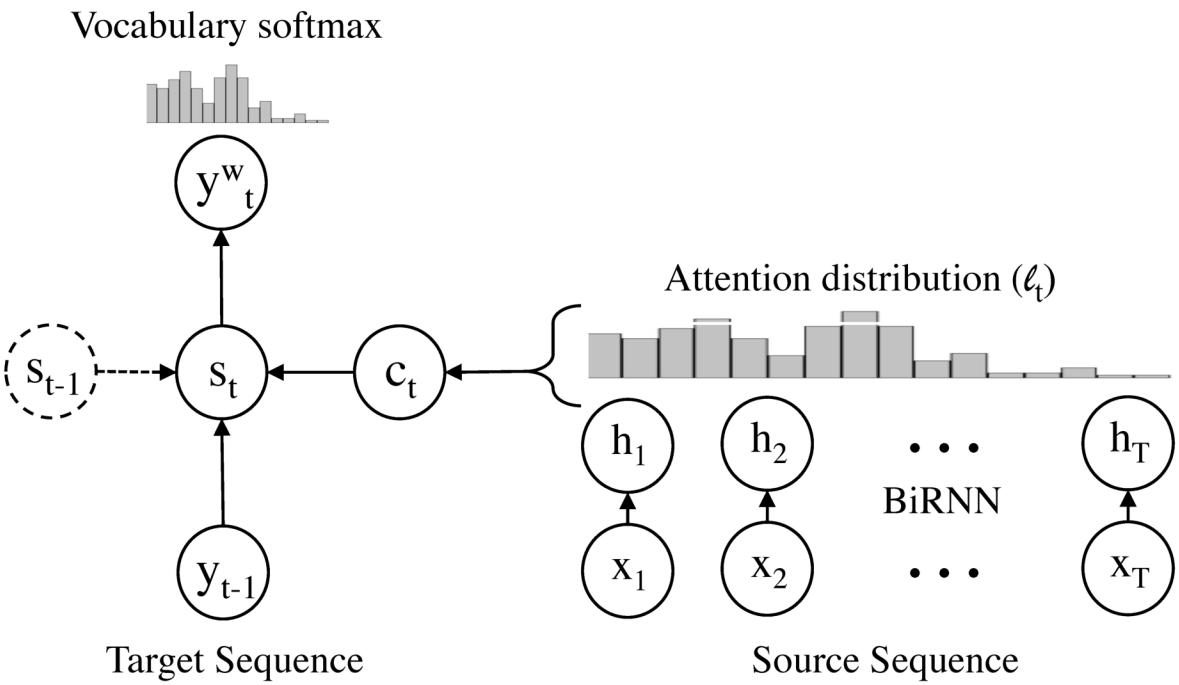


FIG. 7.2. A depiction of neural machine translation architecture with attention. At each timestep, the model generates the attention distribution $\mathbf{l}_t$. We use $\mathbf{l}_t$ and the encoder's hidden states to obtain the context $\mathbf{c}_t$. The decoder uses $\mathbf{c}_t$ to predict a vector of probabilities for the words $\mathbf{w}_t$ by using vocabulary softmax.

a pointer network where each of the output dimension corresponds to the location of a word in the context sequence. Thus, the output dimension of the location softmax varies according to the length of the given context sequence.

At each time-step, if the model decides to use the shortlist softmax, we generate a word w_t from the shortlist. Otherwise, if it is expected that the context sequence contains a word which needs to be generated at the time step, we obtain the location of the context word l_t from the location softmax. The key to making this possible is deciding when to use the shortlist softmax or the location softmax at each time step. In order to accomplish this, we introduce a switching network to the model. The switching network, which is a multilayer perceptron in our experiments, takes the representation of the context sequence (similar to the input annotation in NMT) and the previous hidden state of the output (or the decoder) RNN as its input. It outputs a binary variable z_t which indicates whether to use the shortlist softmax (when $z_t = 1$) or the location softmax (when $z_t = 0$). Note that if the word that is expected to be generated at each time-step is neither in the shortlist nor in the context

sequence, the switching network selects the shortlist softmax, and then the shortlist softmax predicts *UNK*. The details of the pointer softmax model can be seen in Figure 7.3 as well.

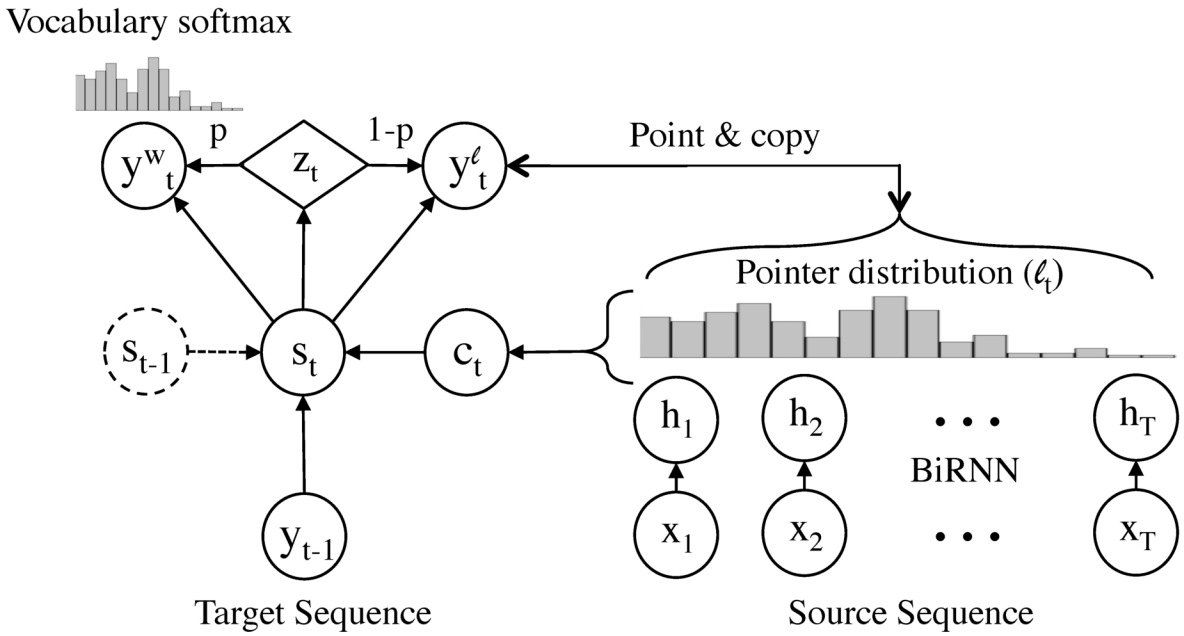


FIG. 7.3. A depiction of the Pointer Softmax (PS) architecture. At each timestep, $\mathbf{l}_t$, $\mathbf{c}_t$ and $\mathbf{w}_t$ for the words over the limited vocabulary (shortlist) is generated. We have an additional switching variable z_t that decides whether to use vocabulary word or to copy a word from the source sequence.

More specifically, our goal is to maximize the probability of observing the target word sequence $\mathbf{y} = (y_1, y_2, \ldots, y_{T_y})$ and the word generation source $\mathbf{z} = (z_1, z_2, \ldots, z_{T_y})$, given the context sequence $\mathbf{x} = (x_1, x_2, \ldots, x_{T_x})$:

$$p_\theta(\mathbf{y}, \mathbf{z}|\mathbf{x}) = \prod_{t=1}^{T_y} p_\theta(y_t, z_t|y_{<t}, z_{<t}, \mathbf{x}). \tag{7.4.1}$$

Note that the word observation y_t can be either a word w_t from the shortlist softmax or a location l_t from the location softmax, depending on the switching variable z_t.

Considering this, we can factorize the above equation further

$$\begin{aligned} p(\mathbf{y}, \mathbf{z}|\mathbf{x}) \quad = \quad & \prod_{t \in \mathcal{T}_w} p(w_t, z_t|(y, z)_{<t}, \mathbf{x}) \times \\ & \prod_{t' \in \mathcal{T}_l} p(l_{t'}, z_{t'}|(y, z)_{<t'}, \mathbf{x}). \end{aligned} \tag{7.4.2}$$

Here, $\mathcal{T}_w$ is a set of time steps where $z_t = 1$, and $\mathcal{T}_l$ is a set of time-steps where $z_t = 0$. And, $\mathcal{T}_w \cup \mathcal{T}_l = \{1, 2, \ldots, T_y\}$ and $\mathcal{T}_w \cap \mathcal{T}_l = \emptyset$. We denote all previous observations at step t by $(y, z)_{<t}$. Note also that $\mathbf{h}_t = f((y, z)_{<t})$.

Then, the joint probabilities inside each product can be further factorized as follows:

$$\begin{aligned} p(w_t, z_t|(y, z)_{<t}) &= p(w_t|z_t = 1, (y, z)_{<t}) \times \\ & \quad p(z_t = 1|(y, z)_{<t}) \end{aligned} \tag{7.4.3}$$

$$\begin{aligned} p(l_t, z_t|(y, z)_{<t}) &= p(l_t|z_t = 0, (y, z)_{<t}) \times \\ & \quad p(z_t = 0|(y, z)_{<t}) \end{aligned} \tag{7.4.4}$$

here, we omitted $\mathbf{x}$ which is conditioned on all probabilities in the above.

The switch probability is modeled as a multilayer perceptron with binary output:

$$p(z_t = 1|(y, z)_{<t}, \mathbf{x}) = \sigma(f(\mathbf{x},\ \mathbf{h}_t;\ \theta)) \tag{7.4.5}$$

$$p(z_t = 0|(y, z)_{<t}, \mathbf{x}) = 1 - \sigma(f(\mathbf{x},\ \mathbf{h}_t;\ \theta)). \tag{7.4.6}$$

And $p(w_t|z_t = 1, (y, z)_{<t}, \mathbf{x})$ is the shortlist softmax and $p(l_t|z_t = 0, (y, z)_{<t}, \mathbf{x})$ is the location softmax which can be a pointer network. $\sigma(\cdot)$ stands for the sigmoid function, $\sigma(x) = \frac{1}{\exp(-x)+1}$.

Given N such context and target sequence pairs, our training objective is to maximize the following log likelihood w.r.t. the model parameter θ

$$\arg\max_{\theta} \frac{1}{N} \sum_{n=1}^{N} \log p_\theta(y_n, z_n|x_n). \tag{7.4.7}$$

7.4.1. Basic Components of the Pointer Softmax

In this section, we discuss practical details of the three fundamental components of the pointer softmax. The interactions between these components and the model is depicted in Figure 7.3.

Location Softmax $\mathbf{l}_t$: The location of the word to copy from source text to the target is predicted by the location softmax $\mathbf{l}_t$. The location softmax outputs the conditional probability distribution $p(l_t|z_t = 0, (y, z)_{<t}, \mathbf{x})$. For models using the attention mechanism such as NMT, we can reuse the probability distributions over the source words in order to predict the location of the word to point. Otherwise we can simply use a pointer network of the model to predict the location.

Shortlist Softmax $\mathbf{w}_t$: The subset of the words in the vocabulary V is being predicted by the shortlist softmax $\mathbf{w}_t$.

Switching network d_t: The switching network d_t is an MLP with sigmoid output function that outputs a scalar probability of switching between $\mathbf{l}_t$ and $\mathbf{w}_t$, and represents the conditional probability distribution $p(z_t|(y,z)_{<t}, \mathbf{x})$. For NMT model, we condition the MLP that outputs the switching probability on the representation of the context of the source text $\mathbf{c}_t$ and the hidden state of the decoder $\mathbf{h}_t$. Note that, during the training, d_t is observed, and thus we do not have to sample.

The output of the pointer softmax, $\mathbf{f}_t$ will be the concatenation of the the two vectors, $d_t \times \mathbf{w}_t$ and $(1 - d_t) \times \mathbf{l}_t$.

At test time, we compute Eqn. (7.4.3) and (7.4.4) for all shortlist word w_t and all location l_t, and pick the word or location of the highest probability.

7.5. Experiments

In this section, we provide our main experimental results with the pointer softmax on machine translation and summarization tasks. In our experiments, we have used the same baseline model and just replaced the softmax layer with pointer softmax layer at the language model. We use the Adadelta Zeiler (2012) learning rule for the training of NMT models. The code for pointer softmax model is available at `https://github.com/caglar/pointer_softmax`.

7.5.1. The Rarest Word Detection

We construct a synthetic task and run some preliminary experiments in order to compare the results with the pointer softmax and the regular softmax's performance for the rare-words. The vocabulary size of our synthetic task is $|V|= 600$ using sequences of length 7. The words in the sequences are sampled according to their unigram distribution which has the form of a geometric distribution. The task is to predict the least frequent word in the sequence according to unigram distribution of the words. During the training, the sequences are generated randomly. Before the training, validation and test sets are constructed with a fixed seed.

We use a GRU layer over the input sequence and take the last-hidden state, in order to get the summary $\mathbf{c}_t$ of the input sequence. The $\mathbf{w}_t$, $\mathbf{l}_t$ are only conditioned on $\mathbf{c}_t$, and the MLP predicting the d_t is conditioned on the latent representations of $\mathbf{w}_t$ and $\mathbf{l}_t$. We use minibatches of size 250 using adam adaptive learning rate algorithm Kingma and Adam (2015) using the learning rate of 8×10^{-4} and hidden layers with 1000 units.

We train a model with pointer softmax where we assign pointers for the rarest 60 words and the rest of the words are predicted from the shortlist softmax of size 540. We observe that increasing the inverse temperature of the sigmoid output of d_t to 2, in other words making the decisions of d_t to become sharper, works better, i.e. $d_t = \sigma(2x)$.

At the end of training with pointer softmax we obtain the error rate of 17.4% and by using softmax over all 600 tokens, we obtain the error-rate of 48.2%.

7.5.2. Summarization

In these series of experiments, we use the annotated Gigaword corpus as described in Rush, Chopra, and Weston (2015a). Moreover, we use the scripts that are made available by the authors of Rush *et al.* (2015a) [1] to preprocess the data, which results to approximately 3.8M training examples. This script generates about 400K validation and an equal number of test examples, but we use a randomly sampled subset of 2000 examples each for validation and testing. We also have made small modifications to the script to extract not only the tokenized words, but also system-generated named-entity tags. We have created two different versions of training data for pointers, which we call *UNK*-pointers data and entity-pointers data respectively.

For the *UNK*-pointers data, we trim the vocabulary of the source and target data in the training set and replace a word by the UNK token whenever a word occurs less than 5 times in either source or target data separately. Then, we create pointers from each UNK token in the target data to the position in the corresponding source document where the same word occurs in the source, as seen in the data before UNK's were created. It is possible that the source can have an UNK in the matching position, but we still created a pointer in this scenario as well. The resulting data has 2.7 pointers per 100 examples in the training set and 9.1 pointers rate in the validation set.

In the entity-pointers data, we exploit the named-entity tags in the annotated corpus and first anonymize the entities by replacing them with an integer-id that always starts from 1 for each document and increments from left to right. Entities that occur more than once in a single document share the same id. We create the anonymization at token-level, so as to allow partial entity matches between the source and target for multi-token entities. Next, we create a pointer from the target to source on similar lines as before, but only for exact matches of the anonymized entities. The resulting data has 161 pointers per 100 examples in the training set and 139 pointers per 100 examples in the validation set.

[1] https://github.com/facebook/NAMAS

If there are multiple matches in the source, either in the UNK-pointers data or the entity-pointers data, we resolve the conflict in favor of the first occurrence of the matching word in the source document. In the UNK data, we model the UNK tokens on the source side using a single placeholder embedding that is shared across all documents, and in the entity-pointers data, we model each entity-id in the source by a distinct placeholder, each of which is shared across all documents.

In all our experiments, we use a bidirectional GRU-RNN Chung *et al.* (2014) for the encoder and a uni-directional RNN for the decoder. To speed-up training, we use the large-vocabulary trick Jean *et al.* (2014) where we limit the vocabulary of the softmax layer of the decoder to 2000 words dynamically chosen from the words in the source documents of each batch and the most common words in the target vocabulary. In both experiments, we fix the embedding size to 100 and the hidden state dimension to 200. We use pre-trained word2vec vectors trained on the same corpus to initialize the embeddings, but we finetune them by backpropagating through the pre-trained embeddings during training. Our vocabulary sizes are fixed to 125K for source and 75K for target for both experiments.

We use the reference data for pointers for the model only at the training time. During the test time, the switch makes a decision at every timestep on which softmax layer to use.

For evaluation, we use full-length Rouge F1 using the official evaluation tool [2]. In their work, the authors of Bahdanau *et al.* (2015) use full-length Rouge Recall on this corpus, since the maximum length of limited-length version of Rouge recall of 75 bytes (intended for DUC data) is already long for Gigaword summaries. However, since full-length Recall can unfairly reward longer summaries, we also use full-length F1 in our experiments for a fair comparison between our models, independent of the summary length.

The experimental results comparing the Pointer Softmax with NMT model are displayed in Table 7.1 for the *UNK* pointers data and in Table 7.2 for the entity pointers data. As our experiments show, pointer softmax improves over the baseline NMT on both UNK data and entities data. Our hope was that the improvement would be larger for the entities data since the incidence of pointers was much greater. However, it turns out this is not the case, and we suspect the main reason is anonymization of entities which removed data-sparsity by converting all entities to integer-ids that are shared across all documents. We believe that on de-anonymized data, our model could help more, since the issue of data-sparsity is more acute in this case.

In Table 7.3, we provide the results for summarization on Gigaword corpus in terms of recall as also similar comparison is done by Rush *et al.* (2015a). We observe improvements

[2] `http://www.berouge.com/Pages/default.aspx`

Tab. 7.1. Results on Gigaword Corpus when pointers are used for UNKs in the training data, using Rouge-F1 as the evaluation metric.

	Rouge-1	Rouge-2	Rouge-L
NMT + lvt	34.87	16.54	32.27
NMT + lvt + PS	**35.19**	**16.66**	**32.51**

Tab. 7.2. Results on anonymized Gigaword Corpus when pointers are used for entities, using Rouge-F1 as the evaluation metric.

	Rouge-1	Rouge-2	Rouge-L
NMT + lvt	34.89	**16.78**	32.37
NMT + lvt + PS	**35.11**	16.76	**32.55**

Tab. 7.3. Results on Gigaword Corpus for modeling UNK's with pointers in terms of recall.

	Rouge-1	Rouge-2	Rouge-L
NMT + lvt	36.45	17.41	33.90
NMT + lvt + PS	**37.29**	**17.75**	**34.70**

on all the scores with the addition of pointer softmax. Let us note that, since the test set of Rush *et al.* (2015a) is not publicly available, we sample 2000 texts with their summaries without replacement from the validation set and used those examples as our test set.

In Table 7.4 we present a few system generated summaries from the Pointer Softmax model trained on the *UNK* pointers data. From those examples, it is apparent that the model has learned to accurately point to the source positions whenever it needs to generate rare words in the summary.

7.5.3. Neural Machine Translation

In our neural machine translation (NMT) experiments, we train NMT models with attention over the Europarl corpus Bahdanau *et al.* (2015) over the sequences of length up to 50 for English to French translation. [3]. All models are trained with early-stopping which is done based on the negative log-likelihood (NLL) on the development set. Our evaluations to report the performance of our models are done on `newstest2011` by using BLUE score[4].

[3]In our experiments, we use an existing code, provided in `https://github.com/kyunghyuncho/dl4mt-material`, and on the original model we only changed the last softmax layer for our experiments
[4]We compute the BLEU score using the multi-blue.perl script from Moses on tokenized sentence pairs.

TAB. 7.4. Generated summaries from NMT with PS. Boldface words are the words copied from the source.

Source #1	china 's tang **gonghong** set a world record with a clean and jerk lift of ### kilograms to win the women 's over-## kilogram weightlifting title at the asian games on tuesday .
Target #1	china 's tang <unk>,sets world weightlifting record
NMT+PS #1	china 's tang **gonghong** wins women 's weightlifting weightlifting title at asian games
Source #2	owing to criticism , nbc said on wednesday that it was ending a **three-month-old** experiment that would have brought the first liquor advertisements onto national broadcast network television .
Target #2	advertising : nbc retreats from liquor commercials
NMT+PS #2	nbc says it is ending a **three-month-old** experiment
Source #3	a senior trade union official here wednesday called on ghana 's government to be " **mindful** of the plight " of the ordinary people in the country in its decisions on tax increases .
Target #3	tuc official,on behalf of ordinary ghanaians
NMT+PS #3	ghana 's government urged to be **mindful** of the plight

We use $30,000$ tokens for both the source and the target language shortlist vocabularies (1 of the token is still reserved for the unknown words). The whole corpus contains $134,831$ unique English words and $153,083$ unique French words. We have created a word-level dictionary from French to English which contains translation of 15,953 words that are neither in shortlist vocabulary nor dictionary of common words for both the source and the target. There are about $49,490$ words shared between English and French parallel corpora of Europarl.

During the training, in order to decide whether to pick a word from the source sentence using attention/pointers or to predict the word from the short-list vocabulary, we use the following simple heuristic. If the word is not in the short-list vocabulary, we first check if the same word $\mathbf{y}_t$ appears in the source sentence. If it is not, we then check if a translated version of the word exists in the source sentence by using a look-up table between the source and the target language. If the word is in the source sentence, we then use the location of the word in the source as the target. Otherwise we check if one of the English senses from the cross-language dictionary of the French word is in the source. If it is in the source sentence, then we use the location of that word as our translation. Otherwise we just use the argmax of $\mathbf{l}_t$ as the target.

For switching network d_t, we observed that using a two-layered MLP with noisy-tanh activation Gulcehre, Moczulski, Denil, and Bengio (2016c) function with residual connection from the lower layer He, Zhang, Ren, and Sun (2015) activation function to the upper hidden layers improves the BLEU score about 1 points over the d_t using ReLU activation function. We initialized the biases of the last sigmoid layer of d_t to -1 such that if d_t becomes more biased toward choosing the shortlist vocabulary at the beginning of the training. We renormalize the gradients if the norm of the gradients exceed 1 Pascanu, Mikolov, and Bengio (2012).

Tab. 7.5. Europarl Dataset (EN-FR)

	BLEU-4
NMT	20.19
NMT + PS	**23.76**

In Table 7.5, we provided the result of NMT with pointer softmax and we observe about 3.6 BLEU score improvement over our baseline.

In Figure 7.4, we show the validation curves of the NMT model with attention and the NMT model with shortlist-softmax layer. Pointer softmax converges faster in terms of number of minibatch updates and achieves a lower validation negative-log-likelihood (NLL) (63.91) after $200k$ updates over the Europarl dataset than the NMT model with shortlist softmax trained for $400k$ minibatch updates (65.26). Pointer softmax converges faster than the model using the shortlist softmax, because the targets provided to the pointer softmax also acts like guiding hints to the attention.

7.6. Conclusion

In this paper, we propose a simple extension to the traditional soft attention-based shortlist softmax by using pointers over the input sequence. We show that the whole model can be trained jointly with single objective function. We observe noticeable improvements over the baselines on machine translation and summarization tasks by using pointer softmax. By doing a very simple modification over the NMT, our model is able to generalize to the unseen words and can deal with rare-words more efficiently. For the summarization task on Gigaword dataset, the pointer softmax was able to improve the results even when it is used together with the large-vocabulary trick. In the case of neural machine translation, we observed that the training with the pointer softmax is also improved the convergence speed of the model as well. For French to English machine translation on Europarl corpora,

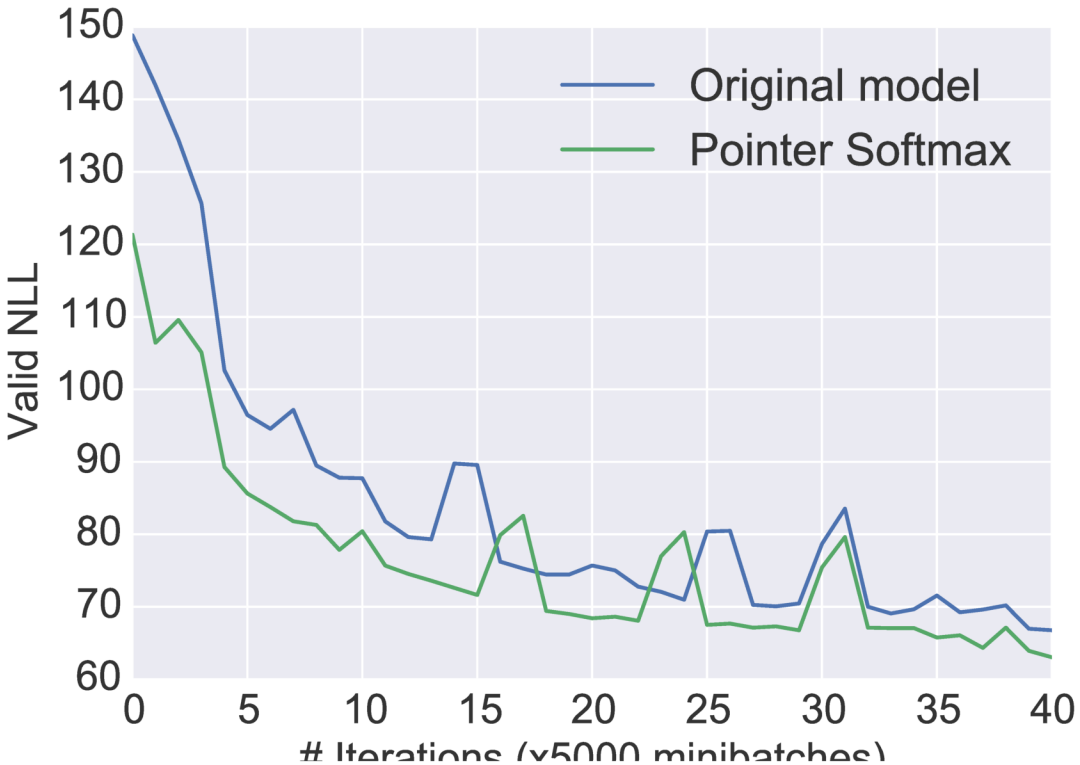


FIG. 7.4. A comparison of the validation learning-curves of the same NMT model trained with pointer softmax and the regular softmax layer. As can be seen from the figures, the model trained with pointer softmax converges faster than the regular softmax layer. Switching network for pointer softmax in this Figure uses ReLU activation function.

we observe that using the pointer softmax can also improve the training convergence of the model.

Chapter 8

PROLOGUE TO THE THIRD ARTICLE

8.1. Article Details

Dynamic Neural Turing Machine with Continuous and Discrete Addressing Schemes, Gulcehre, Caglar, Sarath Chandar, Kyunghyun Cho, and Yoshua Bengio. Neural Computation Journal, 2017.

Personal Contributions:

I got very interested in memory models shortly after the Neural Turing Machines and Memory Networks papers were published. Upon Yoshua Bengio's suggestion, I have tried to reproduce their results and tried those models on various other datasets. Then, I faced the challenge of making NTMs work on NLP tasks. Kyunghyun Cho and I have decided to solve those issues one by one. This lead to the idea of representing the input context via a GRU controller, representing the story with a GRU network instead of BOW (Bag of Words). The different types of regularizations, discrete addressing and trying content based addressing were my idea. Sarath Chandar greatly helped me running the experiments, in particular he ran half of the experiments on bAbI tasks. Kyunghyun Cho proposed to use the addressed vectors. I have written the paper with great helps from Sarath Chandar, Kyunghyun Cho and feedback from Yoshua Bengio.

8.2. Context

We had already most of the results presented in this paper by mid-2015, but we have decided to not put out the paper early on without getting results on more challenging natural language processing tasks. I have kept experimenting with and trying our models on different NLP tasks. However, the results we were getting were not satisfying enough for us and because of that we have not submitted our paper to any venue until NIPS 2016. At NIPS 2016, the reviewers mainly complained about our results and the writing, and it was rejected. The paper was rejected from ICLR 2017 too due to similar reasons and by the time we have submitted there has been several papers proposing similar models as well. Thus the reviewers

were also skeptical about the novelty of the paper. However, our paper is accepted to the Neural Computation Journal after a revision .

8.3. Contributions

We extend the neural Turing machine (NTM) model into a *dynamic neural Turing machine* (D-NTM) by introducing trainable address vectors. This addressing scheme maintains for each memory cell two separate vectors, *content* and *address* vectors. This allows the D-NTM to learn a wide variety of location-based addressing strategies including both linear and nonlinear ones. We implement the D-NTM with both continuous and discrete read and write mechanisms. We investigate the mechanisms and effects of learning to read and write into a memory through experiments on Facebook bAbI tasks using both a **feedforward** and **GRU**-controller. We provide extensive analysis of our model and compare different variations of NTM on this task. We show that our model outperforms LSTM and NTM variants. We provide further experimental results on the sequential pMNIST, Stanford Natural Language Inference, associative recall and copy tasks.

8.4. Recent Developments

After our first revision on arXiv of this paper, there has been new memory models exploring different ways to extend neural networks with memory. The most notable one is the "Differentiable Neural Computers" (Grave *et al.*, 2016) where they have proposed a general memory algorithm that can solve complicated graph and reinforcement learning tasks.

Program synthesis has also been an approach where architectures with explicit memory mechanisms have been useful (Devlin, Uesato, Bhupatiraju, Singh, Mohamed, and Kohli, 2017).

Chapter 9

DYNAMIC NEURAL TURING MACHINE WITH CONTINUOUS AND DISCRETE ADDRESSING SCHEMES

9.1. Introduction

Despite the success of deep learning, (see, e.g., (Goodfellow *et al.*, 2016)) there is still a set of challenging tasks that are not well addressed by conventional general-purpose neural network-based architectures. Those tasks often require a neural network to be equipped with an explicit, external memory in which a larger, potentially unbounded, set of facts need to be stored. They include, but are not limited to, episodic question-answering (Weston *et al.*, 2015a; Hermann *et al.*, 2015; Hill, Bordes, Chopra, and Weston, 2015), learning of compact algorithms (Zaremba, Mikolov, Joulin, and Fergus, 2015), dialogue (Serban, Sordoni, Bengio, Courville, and Pineau, 2016a; Vinyals and Le, 2015) and video caption generation (Yao, Torabi, Cho, Ballas, Pal, Larochelle, and Courville, 2015). These tasks both have long-range dependencies which make learning difficult for conventional RNNs (Bengio *et al.*, 1994; Hochreiter, 1991) and need the models to perform complicated reasoning on the data.

Recently two neural network based architectures are proposed to solve these types of tasks that require an external memory. Memory networks (Weston *et al.*, 2015a) explicitly store all the facts, or information, available at each episode in an external memory (as continuous vectors) and use the attention-based mechanism to index them when computing an output. On the other hand, neural Turing machines (NTM, (Graves *et al.*, 2014)) read each fact in an episode and decides whether to read, write the fact or do both to the external, differentiable memory.

A crucial difference between these two models is that the memory network does not have a mechanism to modify the content of the external memory, while the NTM does. In practice, this leads to easier learning in the memory network, which in turn resulted in that it being used more in real-world tasks (Bordes *et al.*, 2015; Dodge, Gane, Zhang, Bordes, Chopra,

Miller, Szlam, and Weston, 2015). On the contrary, the NTM has mainly been tested on a series of small-scale, carefully-crafted tasks such as copy and associative recall. However, NTM is more expressive, precisely because it can store and modify the internal state of the network as it processes an episode, and we were able to use it without any modifications to the model for different tasks.

The original NTM supports two modes of addressing (which can be used simultaneously): content-based and location-based addressing. The location-based strategy is based on linear addressing, with the distance between each pair of consecutive memory cells fixed to a constant. We address this limitation by introducing a learnable address vector for each memory cell of the NTM with least recently used memory addressing mechanism, and we call this variant a *dynamic neural Turing machine* (D-NTM).

We evaluate the proposed D-NTM on the full set of Facebook bAbI tasks (Weston *et al.*, 2015a) using either **continuous**, differentiable attention or **discrete**, non-differentiable attention (Zaremba and Sutskever, 2015) as an addressing strategy. Our experiments reveal that it is possible to use the discrete, non-differentiable attention mechanism, and in fact, the D-NTM with the discrete attention and GRU controller outperforms the one with the continuous attention. We also provide results on sequential pMNIST, Stanford Natural Language Inference (SNLI) task and algorithmic tasks proposed by (Graves *et al.*, 2014) in order to investigate the ability of our model when dealing with long-term dependencies.

We summarize our contributions in this paper as below,

- We propose a variation of neural Turing machine called a dynamic neural Turing machine (D-NTM) which employs a learnable and location-based addressing.
- We demonstrate the application of neural Turing machines on more natural tasks like episodic question-answering, natural language entailment, digit classification from the pixels besides the synthetic tasks. We provide a detailed analysis of our model on the bAbI task.
- We propose to use the discrete attention mechanism and empirically show that, it can outperform the continuous attention based addressing for episodic QA task.
- We propose a curriculum strategy for our model with the feedforward controller and discrete attention that improves our results significantly.

We focus on comparing our model against similar models such as NTM and LSTM with the same conditions. This helps us to better understand the model's failures.

The remainder of this article is organized as follows. In Section 2, we describe the architecture of Dynamic Neural Turing Machine (D-NTM). In Section 3, we describe the proposed addressing mechanism for D-NTM. Section 4 explains the training procedure. In

Section 5, we briefly discuss some related models. In Section 6, we report results on episodic question answering task. In Section 7, 8, and 9 we discuss the results in sequential MNIST, SNLI, and synthetic algorithm learning tasks respectively. Section 10 concludes the article.

9.2. Dynamic Neural Turing Machine

The proposed dynamic neural Turing machine (D-NTM) extends the neural Turing machine (NTM, (Graves *et al.*, 2014)). The D-NTM consists of two main modules: a controller, and a memory. The controller, which is often implemented as a recurrent neural network, issues a command to the memory so as to read, write to and erase a subset of memory cells.

9.2.1. Memory

D-NTM consists of an external memory $\mathbf{M}_t$, where each memory cell i in $\mathbf{M}_t[i]$ is partitioned into two parts: a trainable address vector $\mathbf{A}_t[i] \in \mathbb{R}^{1\times d_a}$ and a content vector $\mathbf{C}_t[i] \in \mathbb{R}^{1\times d_c}$.

$$\mathbf{M}_t[i] = [\mathbf{A}_t[i]; \mathbf{C}_t[i]] \,.$$

Memory $\mathbf{M}_t$ consists of N such memory cells and hence represented by a rectangular matrix $\mathbf{M}_t \in \mathbb{R}^{N\times(d_c+d_a)}$:

$$\mathbf{M}_t = [\mathbf{A}_t; \mathbf{C}_t] \,.$$

The first part $\mathbf{A}_t \in \mathbb{R}^{N\times d_a}$ is a learnable address matrix, and the second $\mathbf{C}_t \in \mathbb{R}^{N\times d_c}$ a content matrix. The address part $\mathbf{A}_t$ is considered a model parameter that is updated during training. During inference, the address part is not overwritten by the controller and remains constant. On the other hand, the content part $\mathbf{C}_t$ is both read and written by the controller both during training and inference. At the beginning of each episode, the content part of the memory is refreshed to be an all-zero matrix, $\mathbf{C}_0 = \mathbf{0}$. This introduction of the learnable address portion for each memory cell allows the model to learn sophisticated location-based addressing strategies.

9.2.2. Controller

At each timestep t, the controller (1) receives an input value $\mathbf{x}_t$, (2) addresses and reads the memory and creates the content vector $\mathbf{r}_t$, (3) erases/writes a portion of the memory, (4) updates its own hidden state $\mathbf{h}_t$, and (5) outputs a value $\mathbf{y}_t$ (if needed.) In this paper, we use both a gated recurrent unit (GRU, (Cho *et al.*, 2014a)) and a feedforward network to implement the controller such that for a GRU controller

$$\mathbf{h}_t = \text{GRU}(\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{r}_t)$$

and for a feedforward-controller

$$\mathbf{h}_t = \sigma(\mathbf{x}_t, \mathbf{r}_t).$$

9.2.3. Model Operation

At each timestep t, the controller receives an input value $\mathbf{x}_t$. Then it generates the read weights $\mathbf{w}_t^r \in \mathbb{R}^{N\times 1}$. By using the read weights $\mathbf{w}_t^r$, the content vector read from the memory $\mathbf{r}_t \in \mathbb{R}^{(d_a+d_c)\times 1}$ is computed as

$$\mathbf{r}_t = (\mathbf{M}_t)^\top \mathbf{w}_t^r, \tag{9.2.1}$$

The hidden state of the controller ($\mathbf{h}_t$) is conditioned on the memory content vector $\mathbf{r}_t$ and based on the current hidden state of the controller. The model predicts the output label $\mathbf{y}_t$ for the input.

The controller also updates the memory by erasing the old content and writing a new content into the memory. The controller computes three vectors: erase vector $\mathbf{e}_t \in \mathbb{R}^{d_c\times 1}$, write weights $\mathbf{w}_t^w \in \mathbb{R}^{N\times 1}$, and candidate memory content vector $\bar{\mathbf{c}}_t \in \mathbb{R}^{d_c\times 1}$. Both the read ($\mathbf{w}_t^r$) and the write weights ($\mathbf{w}_t^w$) are computed by separate heads (implemented as simple MLPs) and these weight vectors are used to interact with the memory. Erase vector is computed by a simple MLP which is conditioned on the hidden state of the controller $\mathbf{h}_t$. The candidate memory content vector $\bar{\mathbf{c}}_t$ is computed based on the current hidden state of the controller $\mathbf{h}_t \in \mathbb{R}^{d_h\times 1}$ and the input of the controller which is scaled by a scalar gate α_t. The α_t is a function of the hidden state and the input of the controller.

$$\alpha_t = f(\mathbf{h}_t, \mathbf{x}_t), \tag{9.2.2}$$

$$\bar{\mathbf{c}}_t = \text{ReLU}(\mathbf{W}_m \mathbf{h}_t + \alpha_t \mathbf{W}_x \mathbf{x}_t). \tag{9.2.3}$$

where $\mathbf{W}_m$ and $\mathbf{W}_x$ are trainable matrices and ReLU is the rectified linear activation function (Nair and Hinton, 2010). Given the erase, write and candidate memory content vectors ($\mathbf{e}_t$, w_t^w, and $\bar{\mathbf{c}}_t$ respectively), the memory matrix is updated by,

$$\mathbf{C}_t[j] = (1 - \mathbf{e}_t w_t^w[j]) \odot \mathbf{C}_{t-1}[j] + w_t^w[j]\bar{\mathbf{c}}_t. \tag{9.2.4}$$

where the index j in $\mathbf{C}_t[j]$ denotes the j-th row of the content matrix $\mathbf{C}_t$ of the memory matrix $\mathbf{M}_t$.

No Operation (NOP)

As found by Joulin and Mikolov (2015), an additional NOP operation can be useful for the controller *not* to access the memory only once in a while. We model this situation by designating one memory cell as a NOP cell to which the controller should access when it does not need to read or write into the memory. Because reading from or writing into this memory cell is completely ignored.

We illustrate and elaborate more on the read and write operations of the D-NTM in Figure 9.1.

The no operation for the writing in d-NTM is equivalent to learn to set the write gates of the NTM in a way that the original contents of the memory remain unmodified.

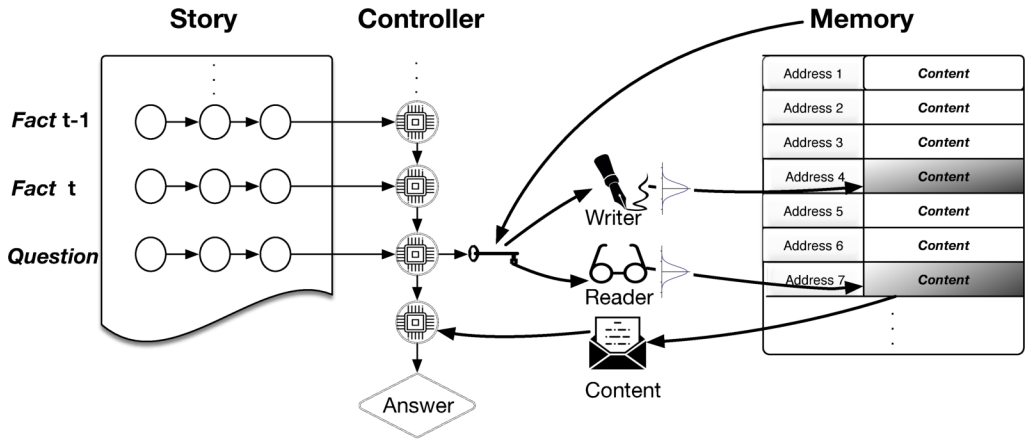


FIG. 9.1. A graphical illustration of the proposed dynamic neural Turing machine with the recurrent-controller. The controller receives the fact as a continuous vector encoded by a recurrent neural network, computes the read and write weights for addressing the memory. If the D-NTM automatically detects that a query has been received, it returns an answer and terminates.

The computation of the read $\mathbf{w}_t^r$ and write vectors $\mathbf{w}_t^w$ are the most crucial parts of the model since the controller decide where to read from and write into the memory by using those. We elaborate this in the next section.

9.3. Addressing Mechanism

Each of the address vectors for both the read and the write heads are computed in similar ways. First, the controller computes a key vector:

$$\mathbf{k}_t = \mathbf{W}_k^\top \mathbf{h}_t + \mathbf{b}_k,$$

For the read and the write operations, $\mathbf{k}_t \in \mathbb{R}^{(d_a+d_c)\times 1}$. $\mathbf{W}_k \in \mathbb{R}^{(d_a+d_c)\times N}$ and $\mathbf{b}_k \in \mathbb{R}^{(d_a+d_c)\times 1}$ are the learnable weight matrix and bias respectively of $\mathbf{k}_t$. Also, the sharpening factor $\beta_t \in \mathbb{R} \geq 1$ is computed as follows:

$$\beta_t = \text{softplus}(\mathbf{u}_\beta^\top \mathbf{h}^t + b_\beta) + 1. \tag{9.3.1}$$

where $\mathbf{u}_\beta$ and b_β are the parameters of the sharpening factor β_t and softplus is defined as follows:

$$\text{softplus}(x) = \log(\exp(x) + 1) \tag{9.3.2}$$

Given the key $\mathbf{k}_t$ and sharpening factor β_t, the logits for the address weights are then computed by,

$$z_t[i] = \beta^t S\left(\mathbf{k}_t, \mathbf{M}_t[i]\right) \tag{9.3.3}$$

where the similarity function is basically the cosine distance where it is defined as $S\left(\mathbf{x}, \mathbf{y}\right) \in \mathbb{R}$ and $1 \geq S\left(\mathbf{x}, \mathbf{y}\right) \geq -1$,

$$S\left(\mathbf{x}, \mathbf{y}\right) = \frac{\mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}||||\mathbf{y}|| + \epsilon}.$$

ϵ is a small positive value to avoid division by zero. We have used $\epsilon = 1e-7$ in all our experiments. The address weight generation which we have described in this section is similar to the content based addressing mechanism proposed in (Graves *et al.*, 2014).

9.3.1. Dynamic Least Recently Used Addressing

We introduce a memory addressing operation that can learn to put more emphasis on the least recently used (LRU) memory (Santoro, Bartunov, Botvinick, Wierstra, and Lillicrap, 2016) locations. As observed by Santoro *et al.* (2016); Rae, Hunt, Harley, Danihelka, Senior, Wayne, Graves, and Lillicrap (2016), we find it easier to learn the write operations with the use of LRU addressing.

To learn a LRU based addressing, first we compute the exponentially moving averages of the logits ($\mathbf{z}_t$) as $\mathbf{v}_t$, where it can be computed as $\mathbf{v}_t = 0.1\mathbf{v}_{t-1} + 0.9\mathbf{z}_t$. We rescale the accumulated $\mathbf{v}_t$ with γ_t, such that the controller adjusts the influence of how much previously written memory locations should effect the attention weights of a particular time-step. Next, we subtract $\mathbf{v}_t$ from $\mathbf{z}_t$ in order to reduce the weights of previously read or written memory locations. γ_t is a shallow MLP with a scalar output and it is conditioned on the hidden state of the controller. γ_t is parametrized with the parameters $\mathbf{u}_\gamma$ and $\mathbf{b}_\gamma$,

$$\gamma_t = \text{sigmoid}(\mathbf{u}_\gamma^\top \mathbf{h}_t + \mathbf{b}_\gamma), \tag{9.3.4}$$

$$\mathbf{w}_t = \text{softmax}(\mathbf{z}_t - \gamma_t \mathbf{v}_{t-1}). \tag{9.3.5}$$

This addressing method increases the weights of the least recently used rows of the memory. The magnitude of the influence of the least-recently used memory locations is learned and adjusted with γ_t. Our LRU addressing is dynamic due to the model's ability to switch between pure content-based addressing and LRU. During training, we do not backpropagate through $\mathbf{v}_t$. Due to the dynamic nature of this addressing mechanism, it can be used for both read and write operations. If needed, the model will automatically learn to disable LRU while reading from the memory.

The address weight vector that are defined in Equation (9.3.5) is a continuous vector. This makes the addressing operation differentiable and we refer to such a D-NTM as continuous D-NTM.

9.3.2. Discrete Addressing

By definition in Eq. (9.3.5), every element in the address vector $\mathbf{w}_t$ is positive and sums up to one. In other words, we can treat this vector as the probabilities of a categorical distribution $\mathcal{C}(\mathbf{w}_t)$ with $\dim(\mathbf{w}_t)$ choices:

$$p[j] = \mathbf{w}_t[j],$$

where $\mathbf{w}_t[j]$ is the j-th element of $\mathbf{w}_t$. We can readily sample from this categorical distribution and form an one-hot vector $\tilde{\mathbf{w}}_t$ such that

$$\tilde{\mathbf{w}}_t[k] = I(k = j),$$

where $j \sim \mathcal{C}(\mathbf{w})$, and I is an indicator function. If we use $\tilde{\mathbf{w}}_t$ instead of $\mathbf{w}_t$, then we will read and write from only one memory cell at a time. This makes the addressing operation non-differentiable and we refer to such a D-NTM as discrete D-NTM. In discrete D-NTM we sample the one-hot vector during training. Once training is over, we switch to a deterministic strategy. We simply choose an element of $\mathbf{w}_t$ with the largest value to be the index of the target memory cell, such that

$$\tilde{\mathbf{w}}_t[k] = \mathbf{I}(k = \text{argmax}(\mathbf{w}_t)).$$

We approximate the gradients of the discrete vectors via a REINFORCE-based method (Williams, 1992) that is described in Section 9.3.2.

9.3.3. Multi-step Addressing

At each time-step, controller may require more than one-step for accessing to the memory. The original NTM addresses this by implementing multiple sets of read, erase and write heads. In this paper, we explore an option of allowing each head to operate more than once at each timestep, similar to the multi-hop mechanism from the end-to-end memory network (Sukhbaatar *et al.*, 2015).

9.4. Training D-NTM

Once the proposed D-NTM is executed, it returns the output distribution $p(\mathbf{y}^{(n)}|\mathbf{x}_1^{(n)}, \ldots, \mathbf{x}_T^{(n)};\ \boldsymbol{\theta})$ for the n^{th} example that is parameterized with $\boldsymbol{\theta}$. We define our cost function as the negative log-likelihood:

$$C(\theta) = -\frac{1}{N}\sum_{n=1}^{N} \log p(\mathbf{y}^{(n)}|\mathbf{x}_1^{(n)}, \ldots, \mathbf{x}_T^{(n)};\ \boldsymbol{\theta}), \tag{9.4.1}$$

where $\boldsymbol{\theta}$ is the set of all the parameters of the model.

Continuous D-NTM, just like the original NTM, is fully end-to-end differentiable and hence we can compute the gradient of this cost function by using backpropagation and learn the parameters of the model with a gradient-based optimization algorithm, such as stochastic gradient descent, to train it end-to-end. However, in discrete D-NTM, we use sampling-based strategy for all the heads during training. This clearly makes the use of backpropagation infeasible to compute the gradient, as the sampling procedure is not differentiable.

9.4.1. Training discrete D-NTM

To train discrete D-NTM, we use REINFORCE (Williams, 1992) together with the three variance reduction techniques–global baseline, input-dependent baseline and variance normalization– as suggested in (Mnih and Gregor, 2014).

Let us define $R(\mathbf{x}) = \log p(\mathbf{y}|\mathbf{x}_1, \ldots, \mathbf{x}_T; \boldsymbol{\theta})$ as a reward. We first center and re-scale the reward by,

$$\tilde{R}(\mathbf{x}) = \frac{R(\mathbf{x}) - b}{\sqrt{\sigma^2 + \epsilon}},$$

where b and σ are respectively the running average and standard deviation of R. ϵ is a very small number added to the standard deviations of the reward to avoid numerical instabilities. We can further center the reward for each input $\mathbf{x}$ separately, i.e.,

$$\bar{R}(\mathbf{x}) = \tilde{R}(\mathbf{x}) - b(\mathbf{x}),$$

where $b(\mathbf{x})$ is computed by a baseline network which takes as input $\mathbf{x}$ and predicts its estimated reward. The baseline network is trained to minimize the Huber loss (Huber, 1964) between the true reward $\tilde{R}(\mathbf{x})$ and the predicted reward $b(\mathbf{x})$. This is also called as input based baseline (IBB) (Mnih and Gregor, 2014).

We use the Huber loss to learn the baseline $b(\mathbf{x})$ which is defined by,

$$H_\delta(z) = \begin{cases} z^2 & \text{for } |z| \leq \delta, \\ \delta(2|z| - \delta), & \text{otherwise,} \end{cases}$$

due to its robustness where z would be $\bar{R}(\mathbf{x})$ in this case. As a further measure to reduce the variance, we regularize the negative entropy of all categorical distributions to facilitate a better exploration during training (Xu *et al.*, 2015b).

Then, the cost function for each training example is approximated as in Equation (9.4.2). In this equation, we write the terms related to computing the REINFORCE gradients that includes terms for the entropy regularization on the action space, the likelihood-ratio term to compute the REINFORCE gradients both for the read and the write heads.

$$\begin{aligned} C^n(\theta) = &- \log p(\mathbf{y}|\mathbf{x}_{1:T}, \tilde{\mathbf{w}}^r_{1:J}, \tilde{\mathbf{w}}^w_{1:J}) \\ &- \sum_{j=1}^{J} \bar{R}(\mathbf{x}^n)(\log p(\tilde{\mathbf{w}}^r_j|\mathbf{x}_{1:T}) + \log p(\tilde{\mathbf{w}}^w_j|\mathbf{x}_{1:T}) \\ &- \lambda_H \sum_{j=1}^{J} (\mathcal{H}(\mathbf{w}^r_j|\mathbf{x}_{1:T}) + \mathcal{H}(\mathbf{w}^w_j|\mathbf{x}_{1:T})). \end{aligned} \tag{9.4.2}$$

where J is the number of addressing steps, λ_H is the entropy regularization coefficient, and $\mathcal{H}$ denotes the entropy.

9.4.2. Curriculum Learning for the Discrete Attention

Training discrete attention with feedforward controller and REINFORCE is challenging. We propose to use a curriculum strategy for training with the discrete attention in order to tackle this problem. For each minibatch, the controller stochastically decides to choose either to use the discrete or continuous weights based on the random variable π_n with probability p_n where n stands for the number of minibatch updates. We only update p_n every k minibatch updates. π_n is a Bernoulli random variable which is sampled with probability of p_n, $\pi_n \sim \text{Bernoulli}(p_n)$. The model will either use the discrete or continuous-attention based on the π_n. We start the training procedure with $p_0 = 1$ and during training p_n is annealed to 0 by setting $p_n = \frac{p_0}{\sqrt{1+n}}$.

We can rewrite the weights $\mathbf{w}_t$ as in Equation (9.4.3), where it is expressed as the combination of continuous attention weights $\bar{\mathbf{w}}_t$ and discrete attention weights $\tilde{\mathbf{w}}_t$ with π_t being a binary variable that chooses to use one of them,

$$\mathbf{w}_t = \pi_n \bar{\mathbf{w}}_t + (1 - \pi_n)\tilde{\mathbf{w}}_t. \tag{9.4.3}$$

By using this curriculum learning strategy, at the beginning of the training, the model learns to use the memory mainly with the continuous attention. As we anneal the p^t, the model will rely more on the discrete attention.

9.4.3. Regularizing D-NTM

If the controller of D-NTM is a recurrent neural network, we find it to be important to regularize the training of the D-NTM so as to avoid suboptimal solutions in which the D-NTM ignores the memory and works as a simple recurrent neural network. We used the regularizers proposed in this section in all our experiments unless it is stated otherwise and we found them very beneficial in particular when an RNN controller is used.

Read-Write Consistency Regularizer

One such suboptimal solution we have observed in our preliminary experiments with the proposed D-NTM is that the D-NTM uses the address part $\mathbf{A}$ of the memory matrix simply as an additional weight matrix, rather than as a means to accessing the content part $\mathbf{C}$. We found that this pathological case can be effectively avoided by encouraging the read head to point to a memory cell which has also been pointed by the write head. This can be implemented as the following regularization term:

$$R_{\text{rw}}(\mathbf{w}^r, \mathbf{w}^w) = \lambda \sum_{t'=1}^{T} ||1 - (\frac{1}{t'} \sum_{t=1}^{t'} \mathbf{w}_t^w)^\top \mathbf{w}_{t'}^r||_2^2 \tag{9.4.4}$$

In the equations above, $\mathbf{w}_t^w$ is the write and $\mathbf{w}_t^r$ is the read weights.

Next Input Prediction as Regularization

Temporal structure is a strong signal that should be exploited by the controller based on a recurrent neural network. We exploit this structure by letting the controller *predict* the input in the future. We maximize the predictability of the next input by the controller during training. This is equivalent to minimizing the following regularizer:

$$R_{\text{pred}}(\mathbf{W}) = -\sum_{t=0}^{T} \log p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{w}_t^r, \mathbf{w}_t^w, \mathbf{e}_t, \mathbf{M}_t; \boldsymbol{\theta})$$

where $\mathbf{x}_t$ is the current input and $\mathbf{x}_{t+1}$ is the input at the next time-step. We find this regularizer to be effective in our preliminary experiments and use it for bAbI tasks. Similarly

(Schmidhuber and Heil, 1995) have proposed a method to compress input sequence into a continuous vector by using predictive coding.

9.5. Related Work

A recurrent neural network (RNN), which is used as a controller in the proposed D-NTM, has an implicit memory in the form of recurring hidden states. Even with this implicit memory, a vanilla RNN is however known to have difficulties in storing information for long time-spans (Bengio *et al.*, 1994; Hochreiter, 1991). Long short-term memory (LSTM, (Hochreiter and Schmidhuber, 1997)) and gated recurrent units (GRU, (Cho *et al.*, 2014a)) have been found to address this issue. However, all these models based solely on RNNs have been found to be limited when they are used to solve, e.g., algorithmic tasks and episodic question-answering.

Our work is inspired by the original NTM work by Graves *et al.* (2014) in which they proposed a model that can modify the contents of its memory to solve complicated algorithmic problems. D-NTM extends their approach from different aspects, first we propose a simpler memory access mechanism that can be trained more easily and outperform the original NTM on some of the real-world tasks. Instead of hard-coding a location-based addressing mechanism as in NTM, D-NTM separates the memory into an address and the content section and address vectors are being learned. D-NTM also integrates an LRU mechanism that helps the model to learn the addressing of the memory more easily. We propose and use different regularizers to eliminate some of the degeneracies that can happen with the memory networks. We also show that our D-NTM architecture can be trained with discrete addressing mechanism as well.

In addition to the finite random access memory of the neural Turing machine, based on which the D-NTM is designed, other data structures have been proposed as external memory for neural networks. In Sun, Giles, and Chen (1997); Grefenstette, Hermann, Suleyman, and Blunsom (2015); Joulin and Mikolov (2015), a continuous, differentiable stack was proposed. Zaremba and Sutskever (2015) used a grid and tape storage mechanism. These approaches differ from the NTM in that their memory is unbounded and can grow indefinitely. On the other hand, they are often not randomly accessible. Zhang, Yu, and Zhou (2015) proposed a variation of NTM that has a structured memory and they have shown experiments on copy and associative recall tasks with this model.

In parallel to our work Yang (2016) and Graves, Wayne, Reynolds, Harley, Danihelka, Grabska-Barwińska, Colmenarejo, Grefenstette, Ramalho, Agapiou, *et al.* (2016) proposed new memory access mechanisms to improve NTM type of models. Graves *et al.* (2016)'s

approach extends NTM by extending the model's memory access mechanism by sparse temporal linking mechanism and their memory is dynamically extensible. However, the implementation of their model is more complicated. Graves *et al.* (2016) also reported superior results on a diverse set of algorithmic learning tasks as well as bAbI task which we compare our model against. Recently, Henaff, Weston, Szlam, Bordes, and LeCun (2016); Seo, Min, Farhadi, and Hajishirzi (2016) proposed new memory based approaches to tackle bAbI task. The Henaff *et al.* (2016)'s approach is quite general, but Seo *et al.* (2016) used a model that is specifically engineered towards solving bAbI task.

Memory networks (Weston *et al.*, 2015a) form another family of neural networks with external memory. In this class of neural networks, information is stored explicitly as it is (in the form of its continuous representation) in the memory, without being erased or modified during an episode. Memory networks and their variants have been applied to various tasks successfully (Sukhbaatar *et al.*, 2015; Bordes *et al.*, 2015; Dodge *et al.*, 2015; Xiong, Merity, and Socher, 2016b; Chandar, Ahn, Larochelle, Vincent, Tesauro, and Bengio, 2016). Miller, Fisch, Dodge, Karimi, Bordes, and Weston (2016b) have also independently proposed the idea of having separate key and value vectors for memory networks. A similar addressing mechanism is also explored in (Reed and de Freitas, 2016) in the context of learning program traces.

Another related family of models is the attention-based neural networks. Neural networks with continuous or discrete attention over an input have shown promising results on a variety of challenging tasks, including machine translation (Bahdanau *et al.*, 2014; Luong, Pham, and Manning, 2015b), speech recognition (Chorowski, Bahdanau, Serdyuk, Cho, and Bengio, 2015), machine reading comprehension (Hermann *et al.*, 2015) and image caption generation (Xu *et al.*, 2015b).

The latter two, the memory network and attention-based networks, are however clearly distinguishable from the D-NTM by the fact that they do not modify the content of the memory.

9.6. Experiments on Episodic Question-Answering

In this section, we evaluate the proposed D-NTM on the synthetic episodic question-answering task called Facebook bAbI (Weston *et al.*, 2015b). We use the version of the dataset that contains 10k training examples per sub-task provided by Facebook.[1] For each episode, the D-NTM reads a sequence of factual sentences followed by a question, all of

[1] `https://research.facebook.com/researchers/1543934539189348`

which are given as natural language sentences. The D-NTM is expected to store and retrieve relevant information in the memory in order to answer the question based on the presented facts.

9.6.1. Model and Training Details

We used the same hyperparameters for all the tasks for a given model. We encode a variable-length fact into a fixed-size representation using a GRU and then feed it to the controller. Unlike a bag-of-words encoding, this allows the D-NTM to exploit the word ordering in each fact. We experiment with both a recurrent and feedforward neural network as the controller that generates the read and write weights. The controller has 180 units. We train our feedforward controller using the noisy-tanh activation function (Gulcehre *et al.*, 2016c) since we were experiencing training difficulties with sigmoid and tanh activation functions. We use both single-step and three-steps addressing with our GRU controller. The memory contains 120 memory cells. Each memory cell consists of a 16-dimensional address part and 28-dimensional content part.

We set aside a random 10% of the training examples as a validation set for each sub-task and use it for early-stopping and hyperparameter search. We train one D-NTM for each sub-task, using Adam (Kingma and Ba, 2014) with its learning rate set to 0.003 and 0.007 respectively for GRU and feedforward controller. The size of each minibatch is 160, and each minibatch is constructed uniform-randomly from the training set.

9.6.2. Goals

The goal of this experiment is three-fold. First, we present for the first time the performance of a memory-based network that can *both* read and write dynamically on the Facebook bAbI tasks[2]. We aim to understand whether a model that has to learn to write an incoming fact to the memory, rather than storing it as it is, is able to work well, and to do so, we compare both the original NTM and proposed D-NTM against an LSTM-RNN.

Second, we investigate the effect of having to learn how to write. The fact that the NTM needs to learn to write likely has adverse effect on the overall performance, when compared to, for instance, end-to-end memory networks (MemN2N, (Sukhbaatar *et al.*, 2015)) and dynamic memory network (DMN+, (Xiong *et al.*, 2016b)) both of which simply store the incoming facts as they are. We quantify this effect in this experiment. Lastly, we show the effect of the proposed learnable addressing scheme.

[2]Similar experiments were done in the recently published (Graves *et al.*, 2016), but D-NTM results for bAbI tasks were already available in arxiv by that time.

We further explore the effect of using a feedforward controller instead of the GRU controller. In addition to the explicit memory, the GRU controller can use its own internal hidden state as the memory. On the other hand, the feedforward controller must solely rely on the explicit memory, as it is the only memory available.

9.6.3. Results and Analysis

In Table 9.1, we first observe that the NTMs are indeed capable of solving this type of episodic question-answering better than the vanilla LSTM-RNN. Although the availability of explicit memory in the NTM has already suggested this result, we note that this is the first time neural Turing machines have been used in this specific task.

Task	LSTM	1-step LBA+CBA NTM	1-step CBA NTM	1-step Continuous D-NTM	1-step Discrete D-NTM	3-steps LBA+CBA NTM	3-steps CBA NTM	3-steps Continuous D-NTM	3-steps Discrete D-NTM
1: 1 supporting fact	0.00	16.30	16.88	5.41	6.66	0.00	0.00	0.00	0.00
2: 2 supporting facts	81.90	57.08	55.70	58.54	56.04	61.67	59.38	46.66	62.29
3: 3 supporting facts	83.10	74.16	55.00	74.58	72.08	83.54	65.21	47.08	41.45
4: 2 argument rels.	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5: 3 argument rels.	1.20	1.46	20.41	1.66	1.04	0.83	1.46	1.25	1.45
6: yes/no questions	51.80	23.33	21.04	40.20	44.79	48.13	54.80	20.62	11.04
7: counting	24.90	21.67	21.67	19.16	19.58	7.92	37.70	7.29	5.62
8: lists/sets	34.10	25.76	21.05	12.58	18.46	25.38	8.82	11.02	0.74
9: simple negation	20.20	24.79	24.17	36.66	34.37	37.80	0.00	39.37	32.50
10: indefinite knowl.	30.10	41.46	33.13	52.29	50.83	56.25	23.75	20.00	20.83
11: basic coreference	10.30	18.96	31.88	31.45	4.16	3.96	0.28	30.62	16.87
12: conjunction	23.40	25.83	30.00	7.70	6.66	28.75	23.75	5.41	4.58
13: compound coref.	6.10	6.67	5.63	5.62	2.29	5.83	83.13	7.91	5.00
14: time reasoning	81.00	58.54	59.17	60.00	63.75	61.88	57.71	58.12	60.20
15: basic deduction	78.70	36.46	42.30	36.87	39.27	35.62	21.88	36.04	40.26
16: basic induction	51.90	71.15	71.15	49.16	51.35	46.15	50.00	46.04	45.41
17: positional reas.	50.10	43.75	43.75	17.91	16.04	43.75	56.25	21.25	9.16
18: size reasoning	6.80	3.96	47.50	3.95	3.54	47.50	47.50	6.87	1.66
19: path finding	90.30	75.89	71.51	73.74	64.63	61.56	63.65	75.88	76.66
20: agent motiv.	2.10	1.25	0.00	2.70	3.12	0.40	0.00	3.33	0.00
Avg.Err.	36.41	31.42	33.60	29.51	**27.93**	32.85	32.76	24.24	**21.79**
Failed (err. > 5%)	16	16	18	16	14	15	14	16	12

TAB. 9.1. Test error rates (%) on the 20 bAbI QA tasks for models using 10k training examples. LBA stands for lcoation based addressing and CBA stands for content based addressing. D-NTM models use GRU controller. In this table, we compare multi-step vs single-step addressing, original NTM with location based+content based addressing vs only content based addressing, and discrete vs continuous addressing D-NTM on bAbI.

All the variants of NTM with the GRU controller outperform the vanilla LSTM-RNN. However, not all of them perform equally well. First, it is clear that the proposed dynamic NTM (D-NTM) using the GRU controller outperforms the original NTM with the GRU controller (NTM, CBA only NTM vs. continuous D-NTM, Discrete D-NTM). As discussed

earlier, the learnable addressing scheme of the D-NTM allows the controller to access the memory slots by location in a potentially nonlinear way. We expect it to help with tasks that have non-trivial access patterns, and as anticipated, we see a large gain with the D-NTM over the original NTM in the tasks of, for instance, 12 - Conjunction and 17 - Positional Reasoning.

Among the recurrent variants of the proposed D-NTM, we notice significant improvements by using discrete addressing over using continuous addressing. We conjecture that this is due to certain types of tasks that require precise/sharp retrieval of a stored fact, in which case continuous addressing is in disadvantage over discrete addressing. This is evident from the observation that the D-NTM with discrete addressing significantly outperforms that with continuous addressing in the tasks of 8 - Lists/Sets and 11 - Basic Coreference. Furthermore, this is in line with an earlier observation in (Xu *et al.*, 2015b), where discrete addressing was found to generalize better in the task of image caption generation.

In Table 9.2, we also observe that the D-NTM with the feedforward controller and discrete attention performs worse than LSTM and D-NTM with continuous-attention. However, when the proposed curriculum strategy from Sec. 9.3.2 is used, the average test error drops from 68.30 to 37.79.

We empirically found training of the feedforward controller more difficult than that of the recurrent controller. We train our feedforward controller based models four times longer (in terms of the number of updates) than the recurrent controller based ones in order to ensure that they are converged for most of the tasks. On the other hand, the models trained with the GRU controller overfit on bAbI tasks very quickly. For example, on tasks 3 and 16 the feedforward controller based model underfits (i.e., high training loss) at the end of the training, whereas with the same number of units the model with the GRU controller can overfit on those tasks after 3,000 updates only.

In Table 9.3, we present the best results obtained for each task after 11 runs with different initialization as also done in (Graves *et al.*, 2016), and we compare out model with NTM, DNC (Graves *et al.*, 2016), and memory network models (Sukhbaatar *et al.*, 2015; Kumar *et al.*, 2015; Xiong *et al.*, 2016b). We show that our model outperforms NTM and performs comparably to other memory models. This approach can be seen as an ensemble learning technique and (Graves *et al.*, 2014; Sukhbaatar *et al.*, 2015) both used similar approaches when reporting their results. In Table 9.4, we report the results as the mean across different tasks. We have not included the models that have very high training errors for each task from the mean results after training for 80k updates on each task. In terms of the mean results D-NTM model with FF-controller performs better than DNC1 and DNC2.

Task	continuous D-NTM	Discrete D-NTM	Discrete* D-NTM	Discrete† D-NTM
1: 1 supporting fact	4.38	81.67	14.79	72.28
2: 2 supporting facts	27.5	76.67	76.67	81.67
3: 3 supporting facts	71.25	79.38	70.83	78.95
4: 2 argument rels.	0.00	78.65	44.06	79.69
5: 3 argument rels.	1.67	83.13	17.71	68.54
6: yes/no questions	1.46	48.76	48.13	31.67
7: counting	6.04	54.79	23.54	49.17
8: lists/sets	1.70	69.75	35.62	79.32
9: simple negation	0.63	39.17	14.38	37.71
10: indefinite knowl.	19.80	56.25	56.25	25.63
11: basic coreference	0.00	78.96	39.58	82.08
12: conjunction	6.25	82.5	32.08	74.38
13: compound coreference	7.5	75.0	18.54	47.08
14: time reasoning	17.5	78.75	24.79	77.08
15: basic deduction	0.0	71.42	39.73	73.96
16: basic induction	49.65	71.46	71.15	53.02
17: positional reas.	1.25	43.75	43.75	30.42
18: size reasoning	0.24	48.13	2.92	11.46
19: path finding	39.47	71.46	71.56	76.05
20: agent motiv.	0.0	76.56	9.79	13.96
Avg. Err.	**12.81**	68.30	37.79	57.21
Failed (err. > 5%)	9	20	19	20

TAB. 9.2. Test error rates (%) on the 20 bAbI QA tasks for models using 10k training examples with the feedforward controller. Discrete* D-NTM model bootstraps the discrete attention with the continuous attention, using the curriculum method that we have introduced in Section 9.3.2. Discrete† D-NTM model is the continuous-attention model which uses discrete-attention at the test time.

We notice a performance gap, when our results are compared to the variants of the memory network (Weston *et al.*, 2015a) (MemN2N and DMN+). We attribute this gap to the difficulty in learning to manipulate and store a complex input.

We would like to highlight that the experimental setup used in DNC (Graves *et al.*, 2016) is different from the setup we use in this paper. This makes the comparisons between the models to be difficult. The main differences broadly are, as the input representations to the controller, they used the embedding representation of each word whereas we have used the representation obtained with a GRU network for each fact. Secondly, they only report joint training results. However, we have trained our models on individual tasks separately. Despite the differences in terms of architecture in DNC paper (see Table 1 in (Graves *et al.*, 2016)), the mean error of their NTM model (28.5% with std of +/- 2.9) is very close to ours (31.4%) with a single run.

We found the feedforward controller with soft addressing to perform better than the GRU controller on bAbI tasks. We believe this particular behavior is due to the construction of the bAbI dataset in which the temporal ordering of the facts do not matter and the facts can appear in an arbitrary order on most tasks. Thus the ability to keep the temporal ordering of the facts is not crucial for most tasks which can be achieved by using an RNN controller. Due to that, we noticed that GRU controller overfits and tends to use the memory in a degenerate manner. Feedforward controller uses an MLP as a controller which on its own do not have any access to the previous context that appears in the story. The only way to access the history for the feedforward controller is to learn to use the memory in a non-degenerate way.

Tasks	Joint NTM	Single D-NTM (ff)	Single D-NTM (GRU)	Joint DNC1	Joint DNC2	Joint MemN2N	Single MemN2N	Single DMN	Single DMN+
1: 1 supporting fact	31.50	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0
2: 2 supporting facts	54.50	27.50	53.13	1.30	0.40	1.00	0.30	1.80	0.3
3: 3 supporting facts	43.90	63.54	41.45	2.40	1.80	6.80	2.10	4.80	1.1
4: 2 argument rels.	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0
5: 3 argument rels	0.80	0.62	1.04	0.50	0.80	6.10	0.80	0.70	0.5
6: yes/no questions	17.10	1.46	11.04	0.00	0.00	0.10	0.10	0.00	0
7: counting	17.80	6.04	2.70	0.20	0.60	6.60	2.00	3.10	2.4
8: lists/sets	13.80	0.00	0.74	0.10	0.30	2.70	0.90	3.50	0
9: simple negation	16.40	0.00	27.63	0.00	0.20	0.00	0.30	0.00	0
10: indefinite knowl.	16.60	1.00	20.83	0.20	0.20	0.50	0.00	0.00	0
11: basic coreference	15.20	0.00	1.25	0.00	0.00	0.00	0.10	0.10	0
12: conjunction	8.90	0.00	1.46	0.10	0.00	0.10	0.00	0.00	0
13: compound coreference	7.40	0.00	1.04	0.00	0.10	0.00	0.00	0.20	0
14: time reasoning	24.20	0.00	55.21	0.30	0.40	0.00	0.10	0.00	0.2
15: basic deduction	47.00	0.00	0.00	0.00	0.00	0.20	0.00	0.00	0
16: basic induction	53.60	49.65	45.41	52.40	55.10	0.20	51.80	0.60	45.3
17: positional reas.	25.50	1.25	9.16	24.10	12.00	41.80	18.60	40.40	4.2
18: size reasoning	2.20	0.00	0.00	4.00	0.80	8.00	5.30	4.70	2.1
19: path finding	4.30	6.35	57.76	0.10	3.90	75.70	2.30	65.50	0
20: agent motiv.	1.50	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0
Avg. err (%)	20.11	7.87	16.49	4.29	3.83	7.49	4.24	6.27	**2.81**
Falied (err. > 5%)	15	5	9	2	2	6	3	2	1

TAB. 9.3. Test error rates (%) on the 20 bAbI QA tasks for models using 10k training examples. This table reports the test error rate of the best model out of several models trained with different random seeds. *Joint* denotes joint training of one model on all tasks and *single* denotes separate training of separate model on each task.

9.6.4. Visualization of Discrete Attention

We visualize the attention of D-NTM with GRU controller with discrete attention in Figure 9.2. From this example, we can see that D-NTM has learned to find the correct supporting fact even without any supervision for the particular story in the visualization.

Tasks	Joint NTM	Joint DNC1	Joint DNC2	Single D-NTM (ff)	Single D-NTM (GRU)
1: 1 supporting fact	40.6 ± 6.7	9.0 ± 12.6	16.2 ± 13.7	2.1 ± 5.3	5.6 ± 8.8
2: 2 supporting facts	56.3 ± 1.5	39.2 ± 20.5	47.5 ± 17.3	43.4 ± 11.1	57.9 ± 3.9
3: 3 supporting facts	47.8 ± 1.7	39.6 ± 16.4	44.3 ± 14.5	66.8 ± 3.2	54.5 ± 12.2
4: 2 argument rels.	0.9 ± 0.7	0.4 ± 0.7	0.4 ± 0.3	2.5 ± 5.6	0.0 ± 0.0
5: 3 argument rels	1.9 ± 0.8	1.5 ± 1.0	1.9 ± 0.6	2.9 ± 5.5	1.5 ± 0.5
6: yes/no questions	18.4 ± 1.6	6.9 ± 7.5	11.1 ± 7.1	35.7 ± 17.0	34.9 ± 17.6
7: counting	19.9 ± 2.5	9.8 ± 7.0	15.4 ± 7.1	8.5 ± 3.3	13.0 ± 6.9
8: lists/sets	18.5 ± 4.9	5.5 ± 5.9	10.0 ± 6.6	4.8 ± 7.3	11.3 ± 9.4
9: simple negation	17.9 ± 2.0	7.7 ± 8.3	11.7 ± 7.4	13.4 ± 11.1	36.1 ± 3.6
10: indefinite knowl.	25.7 ± 7.3	9.6 ± 11.4	14.7 ± 10.8	14.4 ± 9.7	36.4 ± 10.0
11: basic coreference	24.4 ± 7.0	3.3 ± 5.7	7.2 ± 8.1	3.6 ± 5.0	18.3 ± 13.0
12: conjunction	21.9 ± 6.6	5.0 ± 6.3	10.1 ± 8.1	6.2 ± 5.3	11.5 ± 10.6
13: compound coreference	8.2 ± 0.8	3.1 ± 3.6	5.5 ± 3.4	3.5 ± 3.2	8.3 ± 7.9
14: time reasoning	44.9 ± 13.0	11.0 ± 7.5	15.0 ± 7.4	15.0 ± 19.4	58.3 ± 1.5
15: basic deduction	46.5 ± 1.6	27.2 ± 20.1	40.2 ± 11.1	0.0 ± 0.0	30.2 ± 12.6
16: basic induction	53.8 ± 1.4	53.6 ± 1.9	54.7 ± 1.3	52.0 ± 2.2	49.1 ± 2.4
17: positional reas.	29.9 ± 5.2	32.4 ± 8.0	30.9 ± 10.1	14.4 ± 15.2	36.9 ± 11.3
18: size reasoning	4.5 ± 1.3	4.2 ± 1.8	4.3 ± 2.1	0.1 ± 0.1	22.1 ± 22.5
19: path finding	86.5 ± 19.4	64.6 ± 37.4	75.9 ± 30.4	29.5 ± 19.4	64.3 ± 8.7
20: agent motiv.	1.4 ± 0.6	0.0 ± 0.1	0.0 ± 0.0	0.1 ± 0.2	2.6 ± 1.7
Avg. err (%)	28.5 ± 2.9	16.7 ± 7.6	20.8 ± 7.1	**15.9 ± 7.4**	27.6 ± 8.2

TAB. 9.4. Test error rates (%) on the 20 bAbI QA tasks for models using 10k training examples. This table reports the average error rate and standard deviation of several models trained with different random seeds. *Joint* denotes join training of one model on all tasks and *single* denotes separate training of separate model on each task.

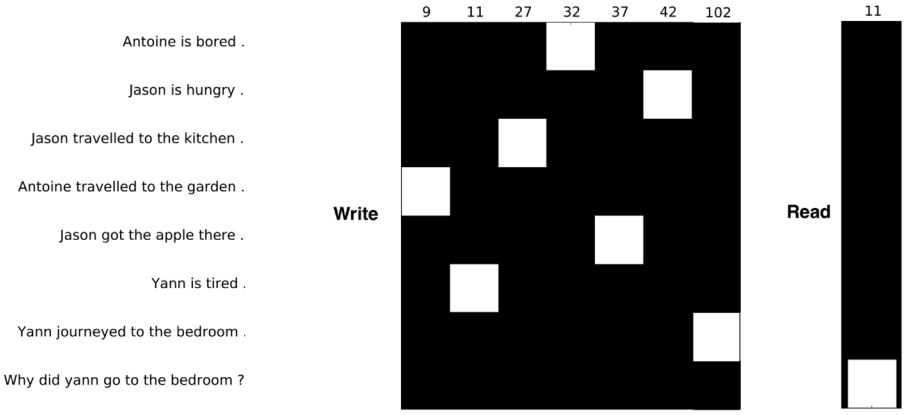


FIG. 9.2. An example view of the discrete attention over the memory slots for both read (left) and write heads(right). x-axis the denotes the memory locations that are being accessed and y-axis corresponds to the content in the particular memory location. In this figure, we visualize the discrete-attention model with 3 reading steps and on task 20. It is easy to see that the NTM with discrete-attention accesses to the relevant part of the memory. We only visualize the last-step of the three steps for writing. Because with discrete attention usually the model just reads the empty slots of the memory.

9.6.5. Learning Curves for the Recurrent Controller

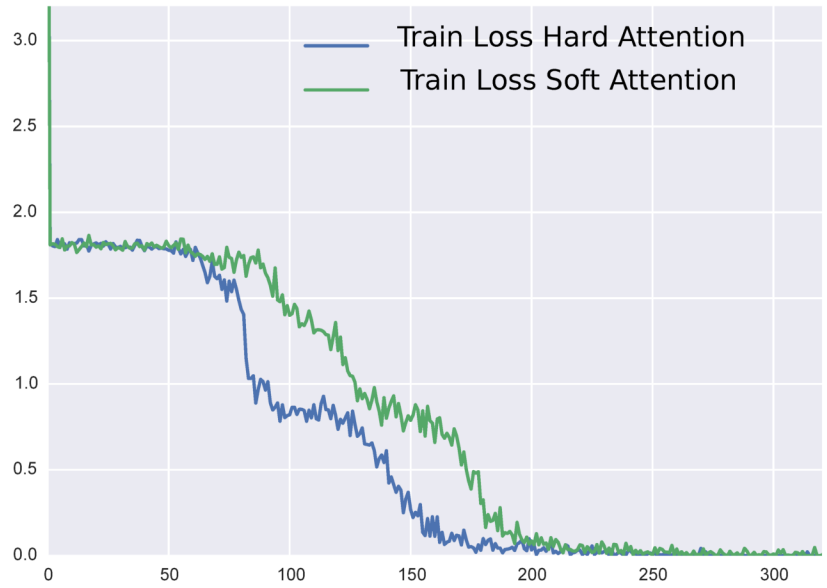


FIG. 9.3. A visualization for the learning curves of continuous and discrete D-NTM models trained on Task 1 using 3 steps. In most tasks, we observe that the discrete attention model with GRU controller does converge faster than the continuous-attention model.

In Figure 9.3, we compare the learning curves of the continuous and discrete attention D-NTM model with recurrent controller on Task 1. Surprisingly, the discrete attention D-NTM converges faster than the continuous-attention model. The main difficulty of learning continuous-attention is due to the fact that learning to write with continuous-attention can be challenging.

9.6.6. Training with Continuous Attention and Testing with Discrete Attention

In Table 9.2, we provide results to investigate the effects of using discrete attention model at the test-time for a model trained with feedforward controller and continuous attention. Discrete* D-NTM model bootstraps the discrete attention with the continuous attention, using the curriculum method that we have introduced in Section 9.4.2. Discrete† D-NTM model is the continuous-attention model which uses discrete-attention at the test time. We observe

that the Discrete† D-NTM model which is trained with continuous-attention outperforms Discrete D-NTM model.

9.6.7. D-NTM with BoW Fact Representation

In Table 9.5, we provide results for D-NTM using BoW with positional encoding (PE) Sukhbaatar *et al.* (2015) as the representation of the input facts. The facts representations are provided as an input to the GRU controller. In agreement to our results with the GRU fact representation, with the BoW fact representation we observe improvements with multi-step of addressing over single-step and discrete addressing over continuous addressing.

Task	Soft D-NTM(1-step)	Discrete D-NTM(1-step)	Soft D-NTM(3-steps)	Discrete D-NTM(3-steps)
1: 1 supporting fact	0.00	0.00	0.00	0.00
2: 2 supporting facts	61.04	59.37	56.87	55.62
3: 3 supporting facts	55.62	57.5	62.5	57.5
4: 2 argument rels	27.29	24.89	26.45	27.08
5: 3 argument rels	13.55	12.08	15.83	14.78
6: yes/no questions	13.54	14.37	21.87	13.33
7: counting	8.54	6.25	8.75	14.58
8: lists/sets	1.69	1.36	3.01	3.02
9: simple negation	17.7	16.66	37.70	17.08
10: indefinite knowl.	26.04	27.08	26.87	23.95
11: basic coreference	20.41	3.95	2.5	2.29
12: conjunction	0.41	0.83	0.20	4.16
13: compound coref.	3.12	1.04	4.79	5.83
14: time reasoning	62.08	58.33	61.25	60.62
15: basic deduction	31.66	26.25	0.62	0.05
16: basic induction	54.47	48.54	48.95	48.95
17: positional reas.	43.75	31.87	43.75	30.62
18: size reasoning	33.75	39.37	36.66	36.04
19: path finding	64.63	69.21	67.23	65.46
20: agent motiv.	1.25	0.00	1.45	0.00
Avg. err (%)	27.02	24.98	26.36	**24.05**
Falied (err. > 5%)	15	14	13	14

TAB. 9.5. Test error rates (%) on the 20 bAbI QA tasks for models using 10k training examples with the GRU controller and representations of facts are obtained with BoW using positional encoding.

9.7. Experiments on Sequential *p*MNIST

In sequential MNIST task, the pixels of the MNIST digits are provided to the model in scan line order, left to right and top to bottom (Le, Jaitly, and Hinton, 2015). At the end of sequence of pixels, the model predicts the label of the digit in the sequence of pixels. We experiment D-NTM on the variation of sequential MNIST where the order of the

pixels is randomly shuffled, we call this task as permuted MNIST (pMNIST). An important contribution of this task to our paper, in particular, is to measure the model's ability to perform well when dealing with long-term dependencies. We report our results in Table 9.6, we observe improvements over other models that we compare against. In Table 9.6, "discrete addressing with MAB" refers to D-NTM model using REINFORCE with baseline computed from moving averages of the reward. Discrete addressing with IB refers to D-NTM using REINFORCE with input-based baseline.

	Test Acc
D-NTM discrete MAB	89.6
D-NTM discrete IB	92.3
Soft D-NTM	**93.4**
NTM	90.9
I-RNN (Le *et al.*, 2015)	82.0
Zoneout (Kruger et. al. 2016)	93.1
LSTM (Kruger et. al. 2016)	89.8
Unitary-RNN (Arjovsky, Shah, and Bengio, 2016)	91.4
Recurrent Dropout (Kruger et. al. 2016)	92.5
Recurrent Batch Normalization (Cooijmans, Ballas, Laurent, and Courville, 2017)	**95.6**

TAB. 9.6. Sequential pMNIST.

In Figure 9.4, we show the learning curves of input-based-baseline (ibb) and regular REINFORCE with moving averages baseline (mab) on the pMNIST task. We observe that input-based-baseline in general is much easier to optimize and converges faster as well. But it can quickly overfit to the task as well. Let us note that, recurrent batch normalization with LSTM (Cooijmans *et al.*, 2017) achieves 95.6% accuracy and it performs much better than other algorithms. However, it is possible to use recurrent batch normalization in our model and potentially improve our results on this task as well.

In all our experiments on sequential MNIST task, we try to keep the capacity of our model to be close to our baselines. We use 100 GRU units in the controller and each content vector of size 8 and with address vectors of size 8. We use a learning rate of $1e-3$ and trained the model with Adam optimizer. We did not use the read and write consistency regularization in any of our models.

9.8. Stanford Natural Language Inference (SNLI) Task

SNLI task (Bowman, Angeli, Potts, and Manning, 2015) is designed to test the abilities of different machine learning algorithms for inferring the entailment between two different

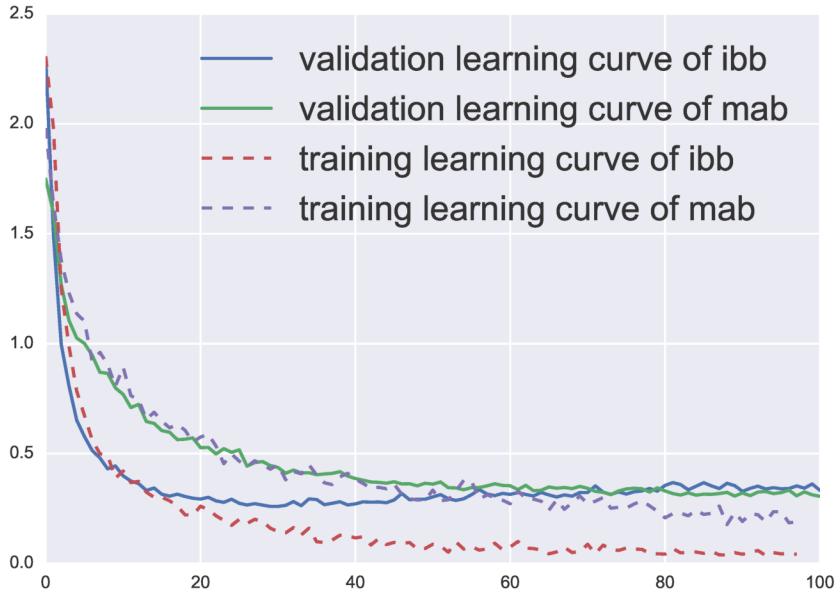


FIG. 9.4. We compare the learning curves of our D-NTM model using discrete attention on *p*MNIST task with input-based baseline and regular REINFORCE baseline. The x-axis is the number of epochs and y-axis is the loss.

statements. Those two statements, can either entail, contradict or be neutral to each other. In this paper, we feed the premise followed by the end of premise (EOP) token and hypothesis in the same sequence as an input to the model. Similarly Rocktäschel, Grefenstette, Hermann, Kočiskỳ, and Blunsom (2015) have trained their model by providing the premise and the hypothesis in a similar way. This ensures that the performance of our model does not rely only on a particular preprocessing or architectural engineering, but rather we rely on the model's ability to represent the sequence and dependencies in the input sequence efficiently. The model proposed by Rocktäschel *et al.* (2015), applies attention over its previous hidden states over premise when it reads the hypothesis.

In Table 9.7, we report results for different models with or without recurrent dropout (Semeniuta, Severyn, and Barth, 2016) and layer normalization (Ba, Kiros, and Hinton, 2016).

The size of the input vocabulary we use in our paper is 41200. We use GLOVE (Pennington, Socher, and Manning, 2014) embeddings to initialize the input embeddings. We use GRU-controller with 300 units and the size of the embeddings are also 300. We optimize our models with Adam. We have done a hyperparameter search to find the optimal learning rate via

random search and sampling the learning rate from log-space between $1e-2$ and $1e-4$ for each model. We use layer-normalization in our controller (Ba *et al.*, 2016).

We have observed significant improvements by using layer normalization and dropout on this task. Mainly because that the overfitting is a severe problem on SNLI. D-NTM achieves better performance compared to both LSTM and NTMs.

	Test Acc
Word by Word Attention (Rocktäschel *et al.*, 2015)	**83.5**
Word by Word Attention two-way (Rocktäschel *et al.*, 2015)	83.2
LSTM + LayerNorm + Dropout	81.7
NTM + LayerNorm + Dropout	81.8
DNTM + LayerNorm + Dropout	**82.3**
LSTM (Bowman *et al.*, 2015)	77.6
D-NTM	80.9
NTM	80.2

Tab. 9.7. Stanford Natural Language Inference Task

9.9. NTM Synthetic Tasks

We explore the possibility of using D-NTM to solve algorithmic tasks such as copy and associative recall tasks. We train our model on the same lengths of sequences that is experimented in (Graves *et al.*, 2014). We compare the results of D-NTM variations and the NTM on Table 9.8. We find out that D-NTM using continuous-attention can successfully learn the “Copy" and “Associative Recall" tasks.

In Table 9.8, we train our model on sequences of the same length as the experiments in (Graves *et al.*, 2014) and test the model on the sequences of the maximum length seen during training. We consider a model to be successful on copy or associative recall if its validation cost (binary cross-entropy) is lower than 0.02 over the sequences of maximum length seen during training. We set the threshold to 0.02 to determine whether a model is successful on a task. Because empirically we observe that the models have higher validation costs perform badly in terms of generalization over the longer sequences. "D-NTM discrete" model in this table is trained with REINFORCE using moving averages to estimate the baseline.

On both copy and associative recall tasks, we try to keep the capacity of our model to be close to our baselines. We use 100 GRU units in the controller and each content vector of has a size of 8 and using address vector of size 8. We use a learning rate of $1e-3$ and trained the model with Adam optimizer. We did not use the read and write consistency regularization

	Copy Tasks	Associative Recall
Soft D-NTM	Success	Success
D-NTM discrete	Success	Failure
NTM	Success	Success

Tab. 9.8. The results of D-NTM architectures on Copy and "Associative Recall" tasks.

in any of our models. For the model with the discrete attention we use REINFORCE with baseline computed using moving averages.

9.10. Conclusion and Future Work

In this paper we extend the neural Turing machines (NTM) by introducing a learnable addressing scheme which allows the NTM to be capable of performing highly nonlinear location-based addressing. This extension, which we refer by dynamic NTM (D-NTM), is extensively tested with various configurations, including different addressing mechanisms (continuous vs. discrete) and different number of addressing steps, on the Facebook bAbI tasks. This is the first time an NTM-type model was tested on this task, and we observe that the NTM, especially the proposed D-NTM, performs better than vanilla LSTM-RNN. Furthermore, the experiments revealed that the discrete addressing works better than the continuous addressing with the GRU controller, and our analysis reveals that this is the case when the task requires precise retrieval of memory content.

Our experiments show that the NTM-based models can be weaker than other variants of memory networks which do not learn but have an explicit mechanism of storing incoming facts as they are. We conjecture that this is due to the difficulty in learning how to write, manipulate and delete the content of memory. Despite this difficulty, we find the NTM-based approach, such as the proposed D-NTM, to be a better, future-proof approach, because it can scale to a much longer horizon (where it becomes impossible to explicitly store all the experiences.)

On pMNIST task, we show that our model can outperform other similar type of approaches proposed to deal with the long-term dependencies. On copy and associative recall tasks, we show that our model can solve the algorithmic problems that are proposed to solve with NTM type of models.

Finally we have shown some results on the SNLI task where our model performed better than the NTM and LSTM on this task. However our results do not involve any task specific

modifications and the results can be improved further by structuring the architecture of our model according to the SNLI task.

The success of both the learnable address and the discrete addressing scheme suggests two future research directions. First, we should try both of these schemes in a wider array of memory-based models, as they are not specific to the neural Turing machines. Second, the proposed D-NTM needs to be evaluated on a diverse set of applications, such as text summarization (Rush, Chopra, and Weston, 2015b), visual question-answering (Antol, Agrawal, Lu, Mitchell, Batra, Zitnick, and Parikh, 2015) and machine translation, in order to make a more concrete conclusion.

Chapter 10

PROLOGUE TO THE FOURTH ARTICLE

10.1. Article Details

Plan, Attend, Generate: Planning for Sequence-to-Sequence Models, Caglar Gulcehre, Francis Dutil, Adam Trischler, Yoshua Bengio, NIPS 2017.
Personal Contributions:
I came up with the ideas of using a planning mechanism inspired by the Strategic Attentive Reader and Writer (STRAW) (Vezhnevets, Mnih, Agapiou, Osindero, Graves, Vinyals, and Kavukcuoglu, 2016) model. I proposed the "Plan, Attend and Generate" and the "repeat Plan Attend Generate" models. I implemented an initial version of those models during my internship at Microsoft Research Maluuba, but later on, Francis Dutil and I implemented a faster version at MILA. The idea of trying our models on algorithmic tasks as well was Adam Trischler's idea. Yoshua Bengio gave us feedback and ideas on training the alignment plan mechanism with discrete outputs. I ran the initial English to German and English to Czech experiments. However, Francis Dutil had to rerun all the machine translation experiments for the camera-ready version of the paper. Adam Trischler and I wrote the paper, and all the authors revised the paper.

10.2. Context

The sequence to sequence models with attention is a compelling architecture, and it has been adapted for many problems successfully. Nevertheless, we observed that a strong decoder could ignore the context coming from the source and generate the sentences according to the probability density determined by the language model on its own. This can cause the model to come up with degenerate samples. This phenomenon was more apparent on the models trained for character-level machine translation: the model tends to ignore the alignments after predicting the first few characters of each word correctly. Planning mechanism helps reinforcement learning models to learn proactive policies, and in this paper, I wanted to see if that would be the case for the attention models as well. The model was mainly inspired

by the STRAW model which got extended to become Feudal Networks (FUN) (Vezhnevets, Osindero, Schaul, Heess, Jaderberg, Silver, and Kavukcuoglu, 2017).

10.3. Contributions

In this paper, we investigate the integration of a planning mechanism into the sequence to sequence models with attention. We develop a model which can plan when it computes alignments between the input and output sequences, constructing a matrix of proposed future alignments and a commitment vector that governs whether to follow or recompute the plan. This mechanism is inspired by the recently proposed strategic attentive reader and writer (STRAW) model for Reinforcement Learning. Our proposed model is end-to-end trainable mainly using differentiable operations. We show that it outperforms a strong baseline on character-level translation tasks from WMT'15, the algorithmic task of finding Eulerian circuits of graphs, and question generation from the text. Our analysis demonstrates that the model computes qualitatively intuitive alignments, converges faster than the baselines, and achieves superior performance with fewer parameters.

Chapter 11

PLAN, ATTEND, GENERATE: PLANNING FOR SEQUENCE-TO-SEQUENCE MODELS

11.1. Introduction

Many important tasks in the machine learning literature can be cast as a sequence-to-sequence problem (Cho *et al.*, 2014a; Sutskever *et al.*, 2014). Machine translation is a prime example of that: a system that takes as input a sequence of words or characters in some source language, then generates a translation – an output sequence of words or characters in the target language.

Neural encoder-decoder models (Cho *et al.*, 2014a; Sutskever *et al.*, 2014) have become a standard approach for sequence-to-sequence tasks like machine translation. Such models generally *encode* the input sequence as a set of vector representations using a recurrent neural network (RNN). A second RNN then *decodes* the output sequence step-by-step, conditioned on the encodings. An important augmentation to this architecture first described by Bahdanau *et al.* (2015), is for models to compute a soft alignment between the encoder representations and the decoder state at each time-step using an *attention* mechanism. The computed alignment conditions the decoder more directly on a relevant subset of the input sequence. The attention mechanism is typically a simple learned function of the decoder's internal state, e.g., an MLP.

In this work, we propose to augment the encoder-decoder model with attention by integrating a planning mechanism. Specifically, we develop a model that uses planning to improve the alignment between input and output sequences. It creates an explicit plan of input-output alignments to use at future time-steps, based on its current observation and a summary of its past actions, which it may follow or modify. This enables the model to plan ahead rather than attending to what is relevant primarily at the current generation step. Concretely, we augment the decoder's internal state with (i) an *alignment plan* matrix and (ii) a *commitment plan* vector. The alignment plan matrix is a template of alignments that the model intends to follow at future time-steps, i.e., a sequence of probability distributions

over input tokens. The commitment plan vector governs whether to follow the alignment plan at the current step or to recompute it and thus models discrete decisions. This is reminiscent to the macro-actions and options in the hierarchical reinforcement learning literature (Dietterich, 2000). This planning mechanism is inspired by the *strategic attentive reader and writer* (STRAW) of Vezhnevets *et al.* (2016) which is originally proposed as a hierarchical reinforcement learning algorithm.

In the parlance of reinforcement learning, existing sequence-to-sequence models with attention can be said to learn reactive policies; however, a model with a planning mechanism could learn more proactive policies. Our work is motivated by the intuition that, although many natural sequences are *output* step-by-step because of constraints on the output process, they are not necessarily *conceived* and *ordered* according to only local, step-by-step interactions. Natural language in the form of speech and writing is again a prime example – sentences are not conceived one word at a time. Planning, that is, choosing some goal along with candidate macro-actions to arrive at it, is one way to induce *coherence* in sequential outputs like language. Learning to generate long coherent sequences, or how to form alignments over long input contexts, is difficult for existing models. In the case of neural machine translation (NMT), the performance of encoder-decoder models with attention deteriorates as sequence length increases (Cho, Van Merriënboer, Bahdanau, and Bengio, 2014b; Sutskever *et al.*, 2014). A planning mechanism could make the decoder's search for alignments more tractable and more scalable.

In this work, we perform planning over the input sequence by searching for alignments; our model does not form an explicit plan of the output tokens to generate. Nevertheless, we find this alignment-based planning to improve performance significantly in several tasks, including character-level NMT. Planning can also be applied explicitly to generation in sequence-to-sequence tasks. For example, recent work by Bahdanau, Brakel, Xu, Goyal, Lowe, Pineau, Courville, and Bengio (2016a) on actor-critic methods for sequence prediction can be seen as this kind of generative planning.

We evaluate our model and report results on character-level translation tasks from WMT'15 for English to German, English to Finnish, and English to Czech language pairs. On almost all pairs we observe improvements over a baseline that represents the state-of-the-art in neural character-level translation. In our NMT experiments, our model outperforms the baseline despite using significantly fewer parameters and converges faster in training. We also show that our model performs better than strong baselines on the algorithmic task of finding Eulerian circuits in random graphs and the task of natural-language question generation from a document and target answer.

11.2. Related Works

Existing sequence-to-sequence models with attention have focused on generating the target sequence by aligning each generated output token to another token in the input sequence. This approach has proven successful in neural machine translation (Bahdanau *et al.*, 2016a) and has recently been adapted to several other applications, including speech recognition (Chan, Jaitly, Le, and Vinyals, 2015) and image caption generation (Xu *et al.*, 2015b). In general these models construct alignments using a simple MLP that conditions on the decoder's internal state. In our work we integrate a planning mechanism into the alignment function.

There have been several earlier proposals for different alignment mechanisms: for instance, Yang, Yang, Dyer, He, Smola, and Hovy (2016) developed a hierarchical attention mechanism to perform document-level classification, while Luo, Chiu, Jaitly, and Sutskever (2016) proposed an algorithm for learning discrete alignments between two sequences using policy gradients (Williams, 1992).

Silver, Huang, Maddison, Guez, Sifre, Van Den Driessche, Schrittwieser, Antonoglou, Panneershelvam, Lanctot, *et al.* (2016) used a planning mechanism based on Monte Carlo tree search with neural networks to train reinforcement learning (RL) agents on the game of Go. Most similar to our work, Vezhnevets *et al.* (2016) developed a neural planning mechanism that can learn high-level temporally abstracted macro-actions, called the strategic attentive reader and writer (STRAW). STRAW uses an action plan matrix, which represents the sequences of actions the model plans to take, and a commitment plan vector, which determines whether to commit an action or recompute the plan. STRAW's action plan and commitment plan are stochastic and the model is trained with RL. Our model computes an alignment plan rather than an action plan, and both its alignment matrix and commitment vector are deterministic and end-to-end trainable with backpropagation.

Our experiments focus on character-level neural machine translation because learning alignments for long sequences is difficult for existing models. This effect can be more pronounced in character-level NMT, since sequences of characters are longer than corresponding sequences of words. Furthermore, to learn a proper alignment between sequences a model often must learn to segment them correctly, a process suited to planning. Previously, Chung *et al.* (2016) and Lee, Cho, and Hofmann (2016) addressed the character-level machine translation problem with architectural modifications to the encoder and the decoder. Our model is the first we are aware of to tackle the problem through planning.

11.3. Planning for Sequence-to-Sequence Learning

We now describe how to integrate a planning mechanism into a sequence-to-sequence architecture with attention (Bahdanau *et al.*, 2015). Our model first creates a *plan*, then computes a soft *alignment* based on the plan, and *generates* at each time-step in the decoder. We refer to our model as PAG (Plan-Attend-Generate).

11.3.1. Notation and Encoder

As input our model receives a sequence of tokens, $X = (x_0, \cdots, x_{|X|})$, where $|X|$ denotes the length of X. It processes these with the encoder, a bidirectional RNN. At each input position i we obtain annotation vector $\mathbf{h}_i$ by concatenating the forward and backward encoder states, $\mathbf{h}_i = [\mathbf{h}_i^{\rightarrow}; \mathbf{h}_i^{\leftarrow}]$, where $\mathbf{h}_i^{\rightarrow}$ denotes the hidden state of the encoder's forward RNN and $\mathbf{h}_i^{\leftarrow}$ denotes the hidden state of the encoder's backward RNN.

Through the decoder the model predicts a sequence of output tokens, $Y = (y_1, \cdots, y_{|Y|})$. We denote by $\mathbf{s}_t$ the hidden state of the decoder RNN generating the target output token at time-step t.

11.3.2. Alignment and Decoder

Our goal is a mechanism that plans which parts of the input sequence to focus on for the next k time-steps of decoding. For this purpose, our model computes an alignment plan matrix $\mathbf{A}_t \in \mathbb{R}^{k \times |X|}$ and commitment plan vector $\mathbf{c}_t \in \mathbb{R}^k$ at each time-step. Matrix $\mathbf{A}_t$ stores the alignments for the current and the next $k-1$ timesteps; it is conditioned on the current input, i.e. the token predicted at the previous time-step $\mathbf{y}_t$, and the current context ψ_t, which is computed from the input annotations $\mathbf{h}_i$. The recurrent decoder function, $f_{\text{dec-rnn}}(\cdot)$, receives $\mathbf{s}_{t-1}$, $\mathbf{y}_t$, ψ_t as inputs and computes the hidden state vector

$$\mathbf{s}_t = f_{\text{dec-rnn}}(\mathbf{s}_{t-1}, \mathbf{y}_t, \psi_t). \tag{11.3.1}$$

Context ψ_t is obtained by a weighted sum of the encoder annotations,

$$\psi_t = \sum_i^{|X|} \alpha_{ti} \mathbf{h}_i, \tag{11.3.2}$$

where the soft-alignment vector $\alpha_t = \texttt{softmax}(\mathbf{A}_t[0]) \in \mathbb{R}^{|X|}$ is a function of the first row of the alignment matrix. At each time-step, we compute a candidate alignment-plan matrix $\bar{\mathbf{A}}_t$ whose entry at the i^{th} row is

$$\bar{\mathbf{A}}_t[i] = f_{\text{align}}(\mathbf{s}_{t-1},\ \mathbf{h}_j,\ \beta_t^i,\ \mathbf{y}_t), \tag{11.3.3}$$

where $f_{\text{align}}(\cdot)$ is an MLP and β_t^i denotes a summary of the alignment matrix's i^{th} row at time $t-1$. The summary is computed using an MLP, $f_r(\cdot)$, operating row-wise on $\mathbf{A}_{t-1}$: $\beta_t^i = f_r(\mathbf{A}_{t-1}[i])$.

The commitment plan vector $\mathbf{c}_t$ governs whether to follow the existing alignment plan, by shifting it forward from $t-1$, or to recompute it. Thus, $\mathbf{c}_t$ represents a discrete decision. For the model to operate discretely, we use the recently proposed Gumbel-Softmax trick (Jang, Gu, and Poole, 2016; Maddison, Mnih, and Teh, 2016) in conjunction with the straight-through estimator (Bengio *et al.*, 2013b) to backpropagate through $\mathbf{c}_t$.[1] The model further learns the temperature for the Gumbel-Softmax as proposed in (Gulcehre *et al.*, 2017a). Both the commitment vector and the action plan matrix are initialized with ones; this initialization is not modified through training.

Alignment-plan update

Our decoder updates its alignment plan as governed by the commitment plan. Denoted by g_t the first element of the discretized commitment plan $\bar{\mathbf{c}}_t$. In more detail, $g_t = \bar{\mathbf{c}}_t[0]$, where the discretized commitment plan is obtained by setting $\mathbf{c}_t$'s largest element to 1 and all other elements to 0. Thus, g_t is a binary indicator variable; we refer to it as the commitment switch. When $g_t = 0$, the decoder simply advances the time index by shifting the action plan matrix $\mathbf{A}_{t-1}$ forward via the shift function $\rho(\cdot)$. When $g_t = 1$, the controller reads the action-plan matrix to produce the summary of the plan, β_t^i. We then compute the updated alignment plan by interpolating the previous alignment plan matrix $\mathbf{A}_{t-1}$ with the candidate alignment plan matrix $\bar{\mathbf{A}}_t$. The mixing ratio is determined by a learned update gate $\mathbf{u}_t \in \mathbb{R}^{k \times |X|}$, whose elements $\mathbf{u}_{ti}$ correspond to tokens in the input sequence and are computed by an MLP with sigmoid activation, $f_{\text{up}}(\cdot)$:

$$\mathbf{u}_{ti} = f_{\text{up}}(\mathbf{h}_i,\ \mathbf{s}_{t-1}),$$
$$\mathbf{A}_t[:, i] = (1 - \mathbf{u}_{ti}) \odot \mathbf{A}_{t-1}[:, i] + \mathbf{u}_{ti} \odot \bar{\mathbf{A}}_t[:, i].$$

To reiterate, the model only updates its alignment plan when the current commitment switch g_t is active. Otherwise it uses the alignments planned and committed at previous time-steps.

Commitment-plan update

The commitment plan also updates when g_t becomes 1. If g_t is 0, the shift function $\rho(\cdot)$ shifts the commitment vector forward and appends a 0-element. If g_t is 1, the model recomputes $\mathbf{c}_t$ using a single layer MLP ($f_c(\cdot)$) followed by a Gumbel-Softmax, and $\bar{\mathbf{c}}_t$ is recomputed by

[1] We also experimented with training $\mathbf{c}_t$ using REINFORCE (Williams, 1992) but found that Gumbel-Softmax led to better performance.

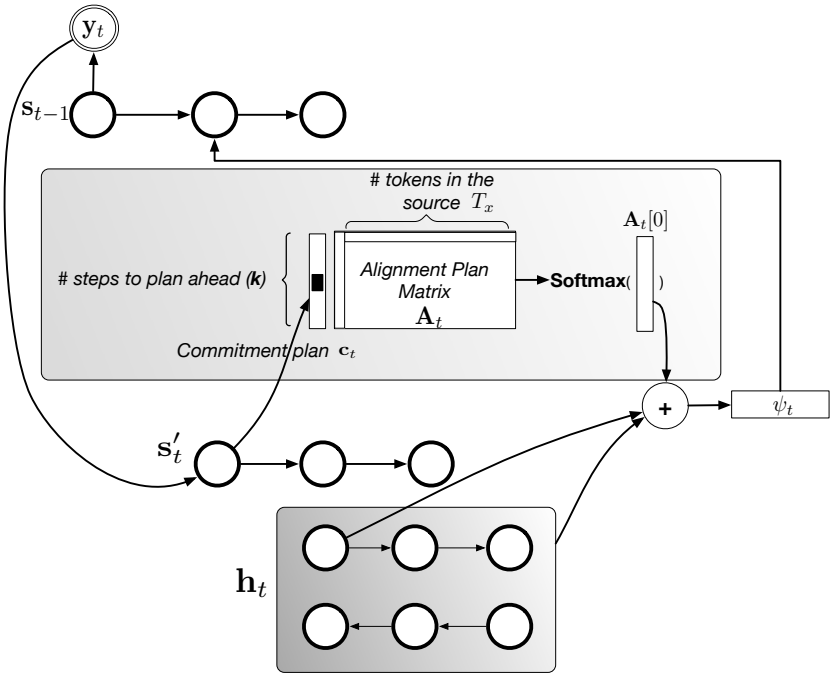


FIG. 11.1. Our planning mechanism in a sequence-to-sequence model that learns to plan and execute alignments. Distinct from a standard sequence-to-sequence model with attention, rather than using a simple MLP to predict alignments our model makes a plan of future alignments using its alignment-plan matrix and decides when to follow the plan by learning a separate commitment vector. We illustrate the model for a decoder with two layers $\mathbf{s}'_t$ for the first layer and the $\mathbf{s}_t$ for the second layer of the decoder. The planning mechanism is conditioned on the first layer of the decoder ($\mathbf{s}'_t$).

discretizing $\mathbf{c}_t$ as a one-hot vector:

$$\mathbf{c}_t = \texttt{gumbel\_softmax}(f_c(\mathbf{s}_{t-1})), \tag{11.3.4}$$

$$\bar{\mathbf{c}}_t = \texttt{one\_hot}(\mathbf{c}_t). \tag{11.3.5}$$

We provide pseudocode for the algorithm to compute the commitment plan vector and the action plan matrix in Algorithm 3. An overview of the model is depicted in Figure 11.1.

11.3.2.1. *Alignment Repeat*

In order to reduce the model's computational cost, we also propose an alternative approach to computing the candidate alignment-plan matrix at every step. Specifically, we propose a model variant that reuses the alignment from the previous time-step until the commitment

Algorithm 2 Pseudocode for updating the alignment plan and commitment vector.

for $j \in \{1, \cdots |X|\}$ **do**
  for $t \in \{1, \cdots |Y|\}$ **do**
    if $g_t = 1$ **then**
      $\mathbf{c}_t = \text{softmax}(f_c(\mathbf{s}_{t-1}))$
      $\beta_t^j = f_r(\mathbf{A}_{t-1}[j])$ ▷ Read alignment plan
      $\bar{\mathbf{A}}_t[j] = f_{\text{align}}(\mathbf{s}_{t-1},\ \mathbf{h}_j,\ \beta_t^j,\ \mathbf{y}_t)$ ▷ Compute candidate alignment plan
      $\mathbf{u}_{tj} = f_{\text{up}}(\mathbf{h}_j,\ \mathbf{s}_{t-1},\ \psi_{t-1})$ ▷ Compute update gate
      $\mathbf{A}_t = (1 - \mathbf{u}_{tj}) \odot \mathbf{A}_{t-1} + \mathbf{u}_{tj} \odot \bar{\mathbf{A}}_t$ ▷ Update alignment plan
    else
      $\mathbf{A}_t = \rho(\mathbf{A}_{t-1})$ ▷ Shift alignment plan
      $\mathbf{c}_t = \rho(\mathbf{c}_{t-1})$ ▷ Shift commitment plan
    end if
    Compute the alignment as $\alpha_t = \texttt{softmax}(\mathbf{A}_t[0])$
  end for
end for

switch activates, at which time the model computes a new alignment. We call this variant *repeat, plan, attend, and generate* (rPAG). rPAG can be viewed as learning an explicit segmentation with an implicit planning mechanism in an unsupervised fashion. Repetition can reduce the computational complexity of the alignment mechanism drastically; it also eliminates the need for an explicit alignment-plan matrix, which reduces the model's memory consumption as well. We provide pseudocode for rPAG in Algorithm 3.

Algorithm 3 Pseudocode for updating the repeat alignment and commitment vector.

for $j \in \{1, \cdots |X|\}$ **do**
  for $t \in \{1, \cdots |Y|\}$ **do**
    if $g_t = 1$ **then**
      $\mathbf{c}_t = \texttt{softmax}(f_c(\mathbf{s}_{t-1}, \psi_{t-1}))$
      $\alpha_t = \texttt{softmax}(f_{\text{align}}(\mathbf{s}_{t-1},\ \mathbf{h}_j,\ \mathbf{y}_t))$
    else
      $\mathbf{c}_t = \rho(\mathbf{c}_{t-1})$ ▷ Shift the commitment vector $\mathbf{c}_{t-1}$
      $\alpha_t = \alpha_{t-1}$ ▷ Reuse the old the alignment
    end if
  end for
end for

11.3.3. Training

We use a deep output layer Pascanu *et al.* (2013a) to compute the conditional distribution over output tokens,

$$p(\mathbf{y}_t|\mathbf{y}_{<t}, \mathbf{x}) \propto \mathbf{y}_t^\top \exp(\mathbf{W}_o f_o(\mathbf{s}_t, \mathbf{y}_{t-1}, \psi_t)), \tag{11.3.6}$$

where $\mathbf{W}_o$ is a matrix of learned parameters and we have omitted the bias for brevity. Function f_o is an MLP with tanh activation.

The full model, including both the encoder and decoder, is jointly trained to minimize the (conditional) negative log-likelihood

$$\mathcal{L} = -\frac{1}{N}\sum_{n=1}^{N} \log\ p_\theta(\mathbf{y}^{(n)}|\mathbf{x}^{(n)}),$$

where the training corpus is a set of $(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})$ pairs and $\boldsymbol{\theta}$ denotes the set of all tunable parameters. As noted in (Vezhnevets *et al.*, 2016), the proposed model can learn to recompute very often which decreases the utility of planning. In order to avoid this behavior, we introduce a loss that penalizes the model for committing too often,

$$\mathcal{L}_{\text{com}} = \lambda_{\text{com}} \sum_{t=1}^{|X|} \sum_{i=0}^{k} ||\frac{1}{k} - \mathbf{c}_{ti}||_2^2, \tag{11.3.7}$$

where λ_{com} is the commitment hyperparameter and k is the timescale over which plans operate.

11.4. Experiments

Our baseline is the encoder-decoder architecture with attention described in Chung *et al.* (2016), wherein the MLP that constructs alignments conditions on the second layer hidden states, $\mathbf{h}^2$, in the two-layer decoder. The integration of our planning mechanism is analogous across the family of attentive encoder-decoder models, thus our approach can be applied more generally. In all experiments below, we use the same architecture for our baseline and the (r)PAG models. The only factor of variation is the planning mechanism. For training all models we use the Adam optimizer with initial learning rate set to 0.0002. We clip gradients with a threshold of 5 (Pascanu, Mikolov, and Bengio, 2013b) and set the number of planning steps (k) to 10 throughout. In order to backpropagate through the alignment-plan matrices and the commitment vectors, the model must maintain these in memory, increasing the computational overhead of the PAG model. However, rPAG does not suffer from these computational issues.

11.4.1. Algorithmic Task

We first compared our models on the algorithmic task from Li, Tarlow, Brockschmidt, and Zemel (2015b) of finding the "Eulerian Circuits" in a random graph. The original work used random graphs with 4 nodes only, but we found that both our baseline and the PAG model

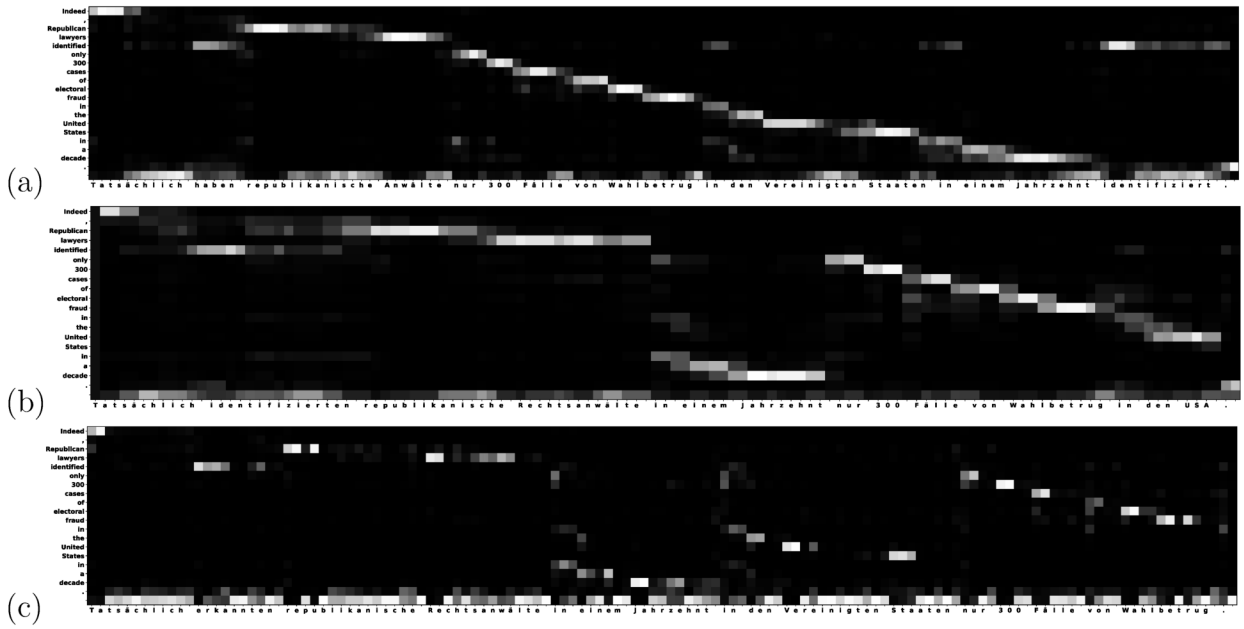


FIG. 11.2. We visualize the alignments learned by PAG in (a), rPAG in (b), and our baseline model with a 2-layer GRU decoder using $\mathbf{h}_2$ for the attention in (c). As depicted, the alignments learned by PAG and rPAG are smoother than those of the baseline. The baseline tends to put too much attention on the last token of the sequence, defaulting to this empty location in alternation with more relevant locations. Our model, however, places higher weight on the last token usually when no other good alignments exist. We observe that rPAG tends to generate less monotonic alignments in general.

solve this task very easily. We therefore increased the number of nodes to 7. We tested the baseline described above with hidden-state dimension of 360, and the same model augmented with our planning mechanism. The PAG model solves the Eulerian Circuits problem with 100% absolute accuracy on the test set, indicating that for all test-set graphs, all nodes of the circuit were predicted correctly. The baseline encoder-decoder architecture with attention performs well but significantly worse, achieving 90.4% accuracy on the test set.

11.4.2. Question Generation

SQUAD (Rajpurkar *et al.*, 2016) is a question answering (QA) corpus wherein each sample is a (document, question, answer) triple. The document and the question are given in words and the answer is a span of word positions in the document. We evaluate our planning models on the recently proposed question-generation task (Yuan, Wang, Gulcehre, Sordoni, Bachman, Subramanian, Zhang, and Trischler, 2017), where the goal is to generate a

question conditioned on a document and an answer. We add the planning mechanism to the encoder-decoder architecture proposed in (Yuan *et al.*, 2017). Both the document and the answer are encoded via recurrent neural networks, and the model learns to align the question output with the document during decoding. The pointer-softmax mechanism (Gulcehre, Ahn, Nallapati, Zhou, and Bengio, 2016d) is used to generate question words from either a shortlist vocabulary or by copying from the document. Pointer-softmax uses the alignments to predict the location of the word to copy; thus, the planning mechanism has a direct influence on the decoder's predictions.

We used 2000 examples from SQUAD's training set for validation and used the official development set as a test set to evaluate our models. We trained a model with 800 units for all GRU hidden states 600 units for word embedding. On the test set the baseline achieved 66.25 NLL while PAG got 65.45 NLL. We show the validation-set learning curves of both models in Figure 11.3.

FIG. 11.3. Learning curves for question-generation models on our development set. Both models have the same capacity and are trained with the same hyperparameters. PAG converges faster than the baseline with better stability.

11.4.3. Character-level Neural Machine Translation

Character-level neural machine translation (NMT) is an attractive research problem (Lee *et al.*, 2016; Chung *et al.*, 2016; Luong and Manning, 2016) because it addresses important issues encountered in word-level NMT. Word-level NMT systems can suffer from problems with rare words (Gulcehre *et al.*, 2016d) or data sparsity, and the existence of compound words without explicit segmentation in some language pairs can make learning alignments

between different languages and translations to be more difficult. Character-level neural machine translation mitigates these issues.

In our NMT experiments we use byte pair encoding (BPE) (Sennrich *et al.*, 2015) for the source sequence and characters at the target, the same setup described in Chung *et al.* (2016). We also use the same preprocessing as in that work.[2] We present our experimental results in Table 11.1. Models were tested on the WMT'15 tasks for English to German (En→De), English to Czech (En→Cs), and English to Finnish (En→Fi) language pairs. The table shows that our planning mechanism improves translation performance over our baseline (which reproduces the results reported in (Chung *et al.*, 2016) to within a small margin). It does this with fewer updates and fewer parameters. We trained (r)PAG for 350K updates on the training set, while the baseline was trained for 680K updates. We used 600 units in (r)PAG's encoder and decoder, while the baseline used 512 in the encoder and 1024 units in the decoder. In total our model has about 4M fewer parameters than the baseline. We tested all models with a beam size of 15.

As can be seen from Table 11.1, layer normalization (Ba *et al.*, 2016) improves the performance of PAG model significantly. However, according to our results on En→De, layer norm affects the performance of our rPAG only marginally. Thus, we decided not to train rPAG with layer norm on other language pairs.

In Figure 11.2, we show qualitatively that our model constructs smoother alignments. At each word that the baseline decoder generates, it aligns the first few characters to a word in the source sequence, but for the remaining characters places the largest alignment weight on the last, empty token of the source sequence. This is because the baseline becomes confident of which word to generate after the first few characters, and it generates the remainder of the word mainly by relying on language-model predictions. We observe that (r)PAG converges faster with the help of the improved alignments, as illustrated by the learning curves in Figure 11.4.

11.5. Conclusion

In this work we addressed a fundamental issue in neural generation of long sequences by integrating *planning* into the alignment mechanism of sequence-to-sequence architectures. We proposed two different planning mechanisms: PAG, which constructs explicit plans in the form of stored matrices, and rPAG, which plans implicitly and is computationally cheaper. The (r)PAG approach empirically improves alignments over long input sequences. We demonstrated our models' capabilities through results on character-level machine translation,

[2]Our implementation is based on the code available at `https://github.com/nyu-dl/dl4mt-cdec`

	Model	Layer Norm	Dev	Test 2014	Test 2015
En→De	Baseline	✗	21.57	21.33	**23.45**
	Baseline†	✗	21.4	21.16	22.1
	Baseline†	✓	21.65	21.69	22.55
	PAG	✗	21.92	21.93	22.42
		✓	**22.44**	**22.59**	**23.18**
	rPAG	✗	21.98	22.17	22.85
		✓	22.33	22.35	22.83
En→Cs	Baseline	✗	17.68	19.27	16.98
	Baseline†	✓	19.1	21.35	18.79
	PAG	✗	18.9	20.6	18.88
		✓	**19.44**	**21.64**	**19.48**
	rPAG	✗	18.66	21.18	19.14
En→Fi	Baseline	✗	11.19	-	10.93
	Baseline†	✓	11.26	-	10.71
	PAG	✗	12.09	-	11.08
		✓	**12.85**	-	**12.15**
	rPAG	✗	11.76	-	11.02

TAB. 11.1. The results of different models on WMT'15 task on English to German, English to Czech and English to Finnish language pairs. We report BLEU scores of each model computed via the *multi-blue.perl* script. The best-score of each model for each language pair appears in bold-face. We use *newstest2013* as our development set, *newstest2014* as our "Test 2014" and *newstest2015* as our "Test 2015" set. $\binom{\dagger}{}$ denotes the results of the baseline that we trained using the hyperparameters reported in Chung *et al.* (2016) and the code provided with that paper. For our baseline, we only report the median result, and do not have multiple runs of our models.

an algorithmic task, and question generation. In machine translation, models with planning outperform a state-of-the-art baseline on almost all language pairs using fewer parameters. We also showed that our model outperforms baselines with the same architecture (minus planning) on the question-generation and algorithmic tasks. The introduction of planning also improves training convergence.

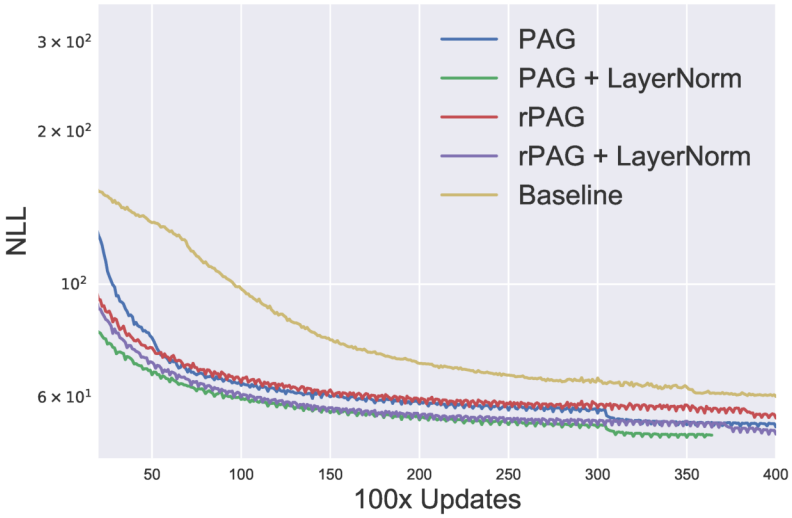


FIG. 11.4. Learning curves for different models on WMT'15 for En→De. Models with the planning mechanism converge faster than our baseline (which has larger capacity).

Chapter 12

PROLOGUE TO THE FIFTH ARTICLE

12.1. Article Details

Noisy Activation Functions, Caglar Gulcehre, Marcin Moczulski, Misha Denil, Caglar Gulcehre, International Conference on Machine Learning (ICML) 2016.
Personal Contribution:
I have first came up with the idea of using piecewise linear activation functions with noise to model the discrete decisions in neural networks with backpropagation. However, my experiments in that direction did not work very well. I decided to try this with neural networks to emulate the sigmoid and tanh type of saturating activation functions in order to overcome some of the issues training them. After getting some encouraging results, I have started discussing this with Marcin Moczulski and Yoshua Bengio. From the discussions with Marcin, we have decided to learn standard deviation and the mean of the noise in a parametrized form. Yoshua Bengio suggested us to run experiments with annealing the noise to establish the relationship between our method and the continuation methods. Marcin Moczulski ran the experiments on "Learning to Execute" and Penntreebank dataset. I have run the rest of the experiments. Misha greatly helped us writing the paper. Yoshua Bengio has written the part of the paper related to the continuation methods.

12.2. Context

Piecewise linear activation functions such as rectified linear units (ReLU) have contributed to the success of deep learning in vision problems. Inspired from those results, we decided to explore the possibility of extending those results for recurrent neural networks. Hannun *et al.* (2014) obtained state of art on speech recognition by replacing sigmoid activation functions of the vanilla recurrent neural networks with a clipped ReLU. We extended this to the gates of the LSTMs and GRUs. Furthermore, the noise injected to the activation function prevents the activations to get stuck in the saturated regime. Following the continuation based approaches as proposed by (Mobahi, 2016), we have also shown that annealing the noise injected to

the activation function has a profound continuation like effect on the training. Our method mainly improves the training of the RNNs, but since the noise is injected into the activation function, it also has a regularization effect as well.

The paper is published at ICML 2016 and has over 26 citations.

12.3. Contributions

We propose to exploit the injection of appropriate noise so that the gradients may flow easily, even if the noiseless application of the activation function would yield zero gradient. Large noise will dominate the noise-free gradient and allow stochastic gradient descent to explore more. By adding noise only to the problematic parts of the activation function, we allow the optimization procedure to explore the boundary between the degenerate (saturating) and the well-behaved parts of the activation function. We also establish connections to simulated annealing, when the amount of noise is annealed down, making it easier to optimize hard objective functions. We find experimentally that replacing such saturating activation functions by noisy variants helps training in many contexts, yielding state-of-the-art or competitive results on different datasets and tasks, especially when training seems to be the most difficult, e.g., when curriculum learning is necessary to obtain good results.

12.4. Recent Developments

We have extended our analysis on the relationship between the continuation and noise injected into the activation function in our more recent work (Gulcehre, Moczulski, Visin, and Bengio, 2016b). In this paper, we have shown that annealing the noise in the activations for each layer with a different rate can be an effective way to train neural networks. Furthermore, recurrent batch normalization (Cooijmans *et al.*, 2017) and layer normalization (Ba *et al.*, 2016) methods are proposed to deal with the similar issue regarding to saturating activation functions of the recurrent neural networks.

Chapter 13

NOISY ACTIVATION FUNCTIONS

13.1. Introduction

The introduction of the piecewise-linear activation functions such as ReLU and Maxout Goodfellow, Warde-Farley, Mirza, Courville, and Bengio (2013b) units had a profound effect on deep learning, and was a major catalyst in allowing the training of much deeper networks. It is thanks to ReLU that for the first time it was shown (Glorot, Bordes, and Bengio, 2011) that deep purely supervised networks can be trained, whereas using tanh nonlinearity only allowed to train shallow networks. A plausible hypothesis about the recent surge of interest on these piecewise-linear activation functions (Glorot *et al.*, 2011), is due to the fact that they are easier to optimize with SGD and backpropagation than smooth activation functions, such as sigmoid and tanh. The recent successes of piecewise linear functions is particularly evident in computer vision, where the ReLU has become the default choice in convolutional networks.

We propose a new technique to train neural networks with activation functions which strongly saturate when their input is large. This is mainly achieved by injecting noise to the activation function in its saturated regime and learning the level of noise. Using this approach, we have found that it was possible to train neural networks with much wider family of activation functions than previously. Adding noise to the activation function has been considered for ReLU units and was explored in Bengio *et al.* (2013b); Nair and Hinton (2010) for feed-forward networks and Boltzmann machines to encourage units to explore more and make the optimization easier.

More recently there has been a resurgence of interest in more elaborated "gated" architectures such as LSTMs Hochreiter and Schmidhuber (1997) and GRUs Cho *et al.* (2014a), but also encompassing neural attention mechanisms that have been used in the NTM Graves *et al.* (2014), Memory Networks Weston *et al.* (2015a), automatic image captioning Xu *et al.* (2015b), video caption generation Yao *et al.* (2015) and wide areas of applications LeCun, Bengio, and Hinton (2015). A common thread running through these works is the use of

soft-saturating non-linearities, such as the sigmoid or softmax, to emulate the hard decisions of digital logic circuits. In spite of its success, there are two key problems with this approach.

(1) Since the non-linearities still saturate there are problems with vanishing gradient information flowing through the gates; and

(2) since the non-linearities only *softly* saturate they do not allow one to take hard decisions.

Although gates often operate in the soft-saturated regime Karpathy *et al.* (2015); Bahdanau *et al.* (2014); Hermann *et al.* (2015) the architecture prevents them from being fully open or closed. We follow a novel approach to address both of these problems. Our method addresses the second problem through the use of hard-saturating nonlinearities, which allow gates to make perfectly on or off decisions when they saturate. Since the gates are able to be completely open or closed, no information is lost through the leakiness of the soft-gating architecture.

By introducing hard-saturating nonlinearities, we have exacerbated the problem of gradient flow, since gradients in the saturated regime are now precisely zero instead of being negligible. However, by introducing noise into the activation function which can grow based on the magnitude of saturation, we encourage random exploration. Our work builds up on the existing literature on the noise injection methods to the piecewise-linear activation functions (Bengio *et al.*, 2013b; Nair and Hinton, 2010; Xu, Wang, Chen, and Li, 2015a).

At test time the noise in the activation functions can be replaced with its expectation. As our experiments show, the resulting deterministic networks outperform their soft-saturating counterparts on a wide variety of tasks, and allow to reach state-of-the-art performance by simple drop-in replacement of the nonlinearities in existing training code.

The technique that we propose, addresses the difficulty of optimization and having hard-activations at the test time for gating units and we propose a way of performing simulated annealing for neural networks.

Hannun *et al.* (2014); Le *et al.* (2015) used ReLU activation functions with simple RNNs. In this paper, we successfully show that, it is possible to use piecewise-linear activation functions with gated recurrent networks such as LSTM and GRU's.

13.2. Saturating Activation Functions

Definition 13.2.1. ***(Activation Function)***. *An activation function is a function* $h : \mathbb{R} \to \mathbb{R}$ *that is differentiable almost everywhere.*

Definition 13.2.2. ***(Saturation)***. *An activation function $h(x)$ with derivative $h'(x)$ is said to right (resp. left) saturate if its limit as $x \to \infty$ (resp. $x \to -\infty$) is zero. An activation function is said to saturate (without qualification) if it both left and right saturates.*

Most common activation functions used in recurrent networks (for example, tanh and sigmoid) are saturating. In particular they are soft saturating, meaning that they achieve saturation only in the limit.

Definition 13.2.3. ***(Hard and Soft Saturation)***. *Let c be a constant such that $x > c$ implies $h'(x) = 0$ and left hard saturates when $x < c$ implies $h'(x) = 0$, $\forall x$. We say that $h(\cdot)$ hard saturates (without qualification) if it both left and right hard saturates. An activation function that saturates but achieves zero gradient only in the limit is said to soft saturate.*

We can construct hard saturating versions of soft saturating activation functions by taking a first-order Taylor expansion about zero and clipping the results to an appropriate range.

For example, expanding tanh and sigmoid around 0, with $x \approx 0$, we obtain linearized functions u^t and u^s of tanh and sigmoid respectively:

$$\text{sigmoid}(x) \approx u^s(x) = 0.25x + 0.5 \tag{13.2.1}$$

$$\tanh(x) \approx u^t(x) = x. \tag{13.2.2}$$

Clipping the linear approximations result to,

$$\text{hard-sigmoid}(x) = \max(\min(u^s(x),\ 1),\ 0) \tag{13.2.3}$$

$$\text{hard-tanh}(x) = \max(\min(u^t(x),\ 1),\ -1). \tag{13.2.4}$$

The motivation behind this construction is to introduce linear behavior around zero to allow gradients to flow easily when the unit is not saturated, while providing a crisp decision in the saturated regime.

The ability of the hard-sigmoid and hard-tanh to make crisp decisions comes at the cost of exactly 0 gradients in the saturated regime. This can cause difficulties during training: a small but not infinitesimal change of the pre-activation (before the nonlinearity) may help to reduce the objective function, but this will not be reflected in the gradient.

In the rest of the document we will use $h(x)$ to refer to a generic activation function and use $u(x)$ to denote its linearization based on the first-order Taylor expansion about zero. hard-sigmoid saturates when $x \leqslant -2$ or $x \geqslant 2$ and hard-tanh saturates when $x \leqslant -1$ or $x \geqslant 1$. We denote the threshold by x_t. Absolute values of the threshold are $x_t = 2$ for hard-sigmoid and $x_t = 1$ for the hard-tanh.

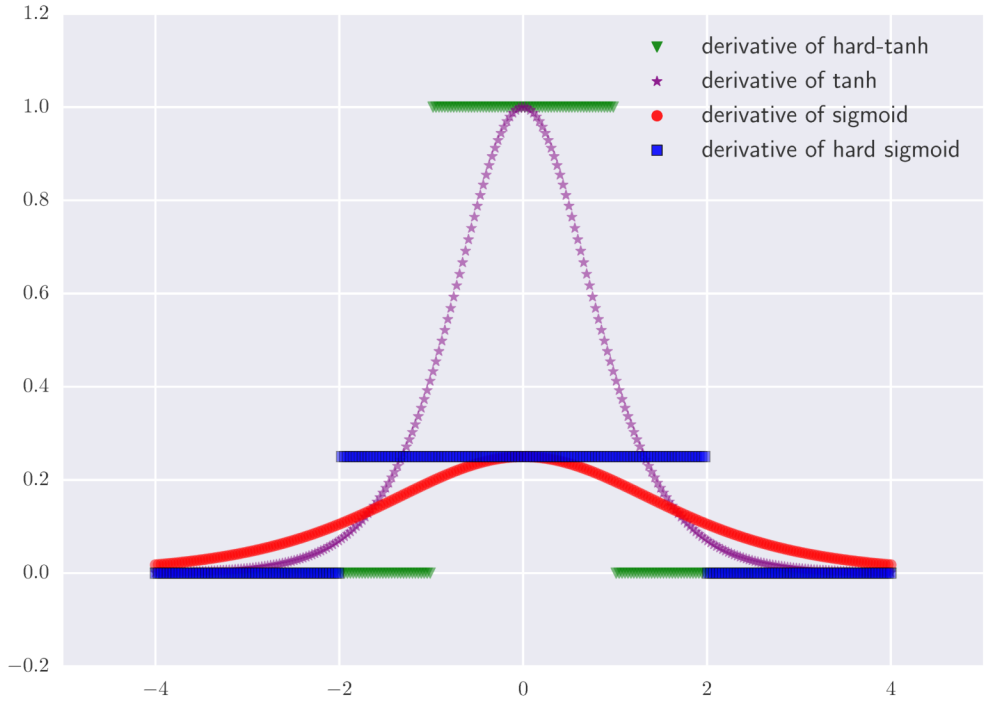


FIG. 13.1. The plot for derivatives of different activation functions.

The highly non-smooth gradient descent trajectory may bring the parameters into a state that pushes the activations of a unit towards the 0 gradient regime for a particular example, from where it may become difficult to escape and the unit may get stuck in the 0 gradient regime.

When units saturate and gradients vanish, an algorithm may require many training examples and a lot of computation to recover.

13.3. Annealing with Noisy Activation Functions

Consider a noisy activation function $\phi(x, \xi)$ in which we have injected iid noise ξ, to replace a saturating nonlinearity such as the hard-sigmoid and hard-tanh introduced in the previous section. In the next section we describe the proposed noisy activation function which has been used for our experiments, but here we want to consider a larger family of such noisy activation functions, when we use a variant of stochastic gradient descent (SGD) for training.

Let ξ have variance σ^2 and mean 0. We want to characterize what happens as we gradually anneal the noise, going from large noise levels ($\sigma \to \infty$) to no noise at all ($\sigma \to 0$).

Furthermore, we will assume that ϕ is such that when the noise level becomes large, so does its derivative with respect to x:

$$\lim_{|\xi|\to\infty} |\frac{\partial \phi(x,\xi)}{\partial x}| \to \infty. \tag{13.3.1}$$

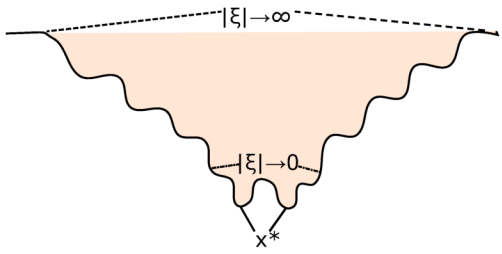


FIG. 13.2. *An example of a one-dimensional, non-convex objective function where a simple gradient descent will behave poorly. With large noise* $|\xi| \to \infty$, *SGD can escape from saddle points and bad local-minima as a result of exploration. As we anneal the noise level* $|\xi| \to 0$, *SGD will eventually converge to one of the local-minima* x^*.

In the 0 noise limit, we recover a deterministic nonlinearity, $\phi(x, 0)$, which in our experiments is piecewise linear and allows us to capture the kind of complex function we want to learn. As illustrated in Figure 13.2, in the large noise limit, large gradients are obtained because backpropagating through ϕ gives rise to large derivatives. Hence, the noise drowns the signal: the example-wise gradient on parameters is much larger than it would have been with $\sigma = 0$. SGD therefore just sees noise and can move around anywhere in parameter space without "seeing" any trend.

Annealing is also related to the signal to noise ratio where SNR can be defined as the ratio of the variance of noise σ_{signal} and σ_{noise}, $SNR = \frac{\sigma_{\text{signal}}}{\sigma_{\text{noise}}}$. If $SNR \to 0$, the model will do pure random exploration. As we anneal SNR will increase, and when σ_{noise} converges to 0, the only source of exploration during the training will come from the noise of Monte Carlo estimates of stochastic gradients.

This is precisely what we need for methods such as simulated annealing (Kirkpatrick, Jr., , and Vecchi, 1983) and continuation methods (Allgower and Georg, 1980) to be helpful, in the context of the optimization of difficult non-convex objectives. With high noise, SGD is free to explore all parts of space. As the noise level is decreased, it will prefer some regions where the signal is strong enough to be "visible" by SGD: given a finite number of SGD steps, the noise is not averaged out, and the variance continues to dominate. Then as the noise level is reduced SGD spends more time in "globally better" regions of parameter space. As it

approaches to zero we are fine-tuning the solution and converging near a minimum of the noise-free objective function. A related approach of adding noise to gradients and annealing the noise was investigated in Neelakantan, Vilnis, Le, Sutskever, Kaiser, Kurach, and Martens (2015) as well. Ge, Huang, Jin, and Yuan (2015) showed that SGD with annealed noise will globally converge to a local-minima for non-convex objective functions in polynomial number of iterations. Recently, Mobahi (2016) propose an optimization method that applies Gaussian smoothing on the loss function such that annealing weight noise is a Monte Carlo estimator of that.

13.4. Adding Noise when the Unit Saturates

A novel idea behind the proposed noisy activation is that **the amount of noise added to the nonlinearity is proportional to the magnitude of saturation of the nonlinearity**. For hard-sigmoid(x) and hard-tanh(x), due to our parametrization of the noise, that translates into the fact that the noise is only added when the hard-nonlinearity saturates. This is different from previous proposals such as the noisy rectifier from Bengio *et al.* (2013b) where noise is added just before a rectifier (ReLU) unit, independently of whether the input is in the linear regime or in the saturating regime of the nonlinearity.

The motivation is to keep the training signal clean when the unit is in the non-saturating (typically linear) regime and provide some noisy signal when the unit is in the saturating regime.

$h(x)$ refer to hard saturation activation function such as the hard-sigmoid and hard-tanh introduced in Sec. 13.2, we consider noisy activation functions of the following form:

$$\phi(x, \xi) = h(x) + s \tag{13.4.1}$$

and $s = \mu + \sigma\xi$. Here ξ is an iid random variable drawn from some generating distribution, and the parameters μ and σ (discussed below) are used to generate a location scale family from ξ.

Intuitively when the unit saturates we pin its output to the threshold value t and add noise. The exact behavior of the method depends on the type of noise ξ and the choice of μ and σ, which we can pick as functions of x in order to let some gradients be propagated even when we are in the saturating regime.

A desirable property we would like ϕ to approximately satisfy is that, in expectation, it is equal to the hard-saturating activation function, i.e.

$$\mathrm{E}_{\xi\sim\mathcal{N}(0,1)}[\phi(x, \xi)] \approx h(x) \tag{13.4.2}$$

If the ξ distribution has zero mean then this property can be satisfied by setting $\mu = 0$, but for biased noise it will be necessary to make other choices for μ. In practice, we used slightly biased ϕ with good results.

Intuitively we would like to add more noise when x is far into the saturated regime, since a large change in parameters would be required desaturate h. Conversely, when x is close to the saturation threshold a small change in parameters would be sufficient for it to escape. To that end we make use of the difference between the original activation function h and its linearization u

$$\Delta = h(x) - u(x) \tag{13.4.3}$$

when choosing the scale of the noise. See Eqs.13.2.1 for definitions of u for the hard-sigmoid and hard-tanh respectively. The quantity Δ is zero in the unsaturated regime, and when h saturates it grows proportionally to the distance between $|x|$ and the saturation threshold x_t. We also refer $|\Delta|$ as the magnitude of the saturation.

We experimented with different ways of scaling σ with Δ, and empirically found that the following formulation performs better:

$$\begin{aligned} \sigma(x) &= c\ (g(p\Delta) - 0.5)^2 \\ g(x) &= \text{sigmoid}(x). \end{aligned} \tag{13.4.4}$$

In Equation 13.4.4 a free scalar parameter p is learned during the course of training. By changing p, the model is able to adjust the magnitude of the noise and that also effects the sign of the gradient as well. The hyper-parameter c changes the scale of the standard deviation of the noise.

13.4.1. Derivatives in the Saturated Regime

In the simplest case of our method we draw ξ from an unbiased distribution, such as a standard normal. In this case we choose $\mu = 0$ to satisfy Equation 13.4.2 and therefore we will have,

$$\phi(x,\ \xi) = h(x) + \sigma(x)\xi$$

Due to our parameterization of $\sigma(x)$, when $|x| \leq x_t$ our stochastic activation function behaves exactly as the linear function $u(x)$, leading to familiar territory. Because Δ will be 0. Let us for the moment restrict our attention to the case when $|x| > x_t$ and h saturates. In this

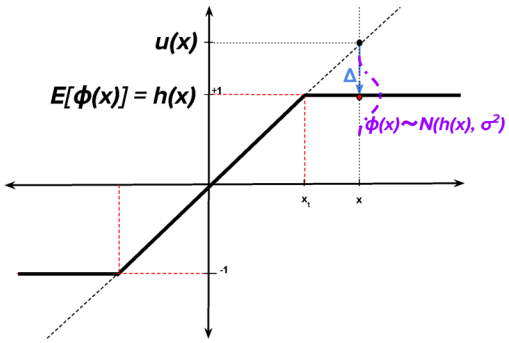


FIG. 13.3. A simple depiction of adding Gaussian noise on the linearized activation function, which brings the average back to the hard-saturating nonlinearity $h(x)$, in bold. Its linearization is $u(x)$ and the noisy activation is ϕ. The difference $h(x) - u(x)$ is Δ which is a vector indicates the discrepancy between the linearized function and the actual function that the noise is being added to $h(x)$. Note that, Δ will be zero, at the non-saturating parts of the function where $u(x)$ and $h(u)$ matches perfectly.

case the derivative of $h(x)$ is precisely zero, however, if we condition on the sample ξ we have

$$\phi'(x, \xi) = \frac{\partial}{\partial x}\phi(x, \xi) = \sigma'(x)\xi \tag{13.4.5}$$

which is non-zero almost surely.

In the non-saturated regime, where $\phi'(x, \xi) = h'(x)$ the optimization can exploit the linear structure of h near the origin in order to tune its output. In the saturated regime the randomness in ξ drives exploration, and gradients still flow back to x since the scale of the noise still depends on x. To reiterate, we get gradient information at every point in spite of the saturation of h, and the variance of the gradient information in the saturated regime depends on the variance of $\sigma'(x)\xi$.

13.4.2. Pushing Activations towards Linear Regime

An unsatisfying aspect of the formulation with unbiased noise is that, depending on the value of ξ occasionally the gradient of ϕ will point the wrong way. This can cause a backwards message that would push x in a direction that would worsen the objective function on average over ξ. Intuitively we would prefer these messages to "push back" the saturated unit towards a non-saturated state where the gradient of $h(x)$ can be used safely.

A simple way to achieve this is to make sure that the noise ξ is always positive and adjust its sign manually to match the sign of x. In particular we could set

$$d(x) = -\operatorname{sgn}(x)\operatorname{sgn}(1 - \alpha)$$

$$s = \mu(x) + d(x)\sigma(x)|\xi|.$$

where ξ and σ are as before and sgn is the sign function, such that $\text{sgn}(x)$ is 1 if x is greater than or equal to 0 otherwise it is -1. We also use the absolute value of ξ in the reparametrization of the noise, such that the noise is being sampled from a half-Normal distribution. We ignored the sign of ξ, such that the direction that the noise pushes the activations are determined by $d(x)$ and it will point towards $h(x)$. Matching the sign of the noise to the sign of x would ensure that we avoid the sign cancellation between the noise and the gradient message from backpropagation. $\text{sgn}(1-\alpha)$ is required to push the activations towards $h(x)$ when the bias from α is introduced.

In practice we use a hyperparameter α that influences the mean of the added term, such that α near 1 approximately satisfies the above condition, as seen in Fig. 13.4. We can rewrite the noisy term s in a way that the noise can either be added to the linearized function or $h(x)$. The relationship between Δ, $u(x)$ and $h(x)$ is visualized Figure 13.3 can be expressed as in Eqn 13.4.6.

We have experimented with different types of noise. Empirically, in terms of performance we found, half-normal and normal noise to be better. In Eqn 13.4.6, we provide the formulation for the activation function where $\epsilon = |\xi|$ if the noise is sampled from half-normal distribution, $\epsilon = \xi$ if the noise is sampled from normal distribution.

$$\phi(x, \xi) = u(x) + \alpha\Delta + d(x)\sigma(x)\epsilon \tag{13.4.6}$$

By using Eqn 13.4.6, we arrive at the noisy activations, which we used in our experiments.

$$\phi(x, \xi) = \alpha h(x) + (1-\alpha)u(x) + d(x)\sigma(x)\epsilon \tag{13.4.7}$$

As can be seen in Eqn 13.4.7, there are three paths that gradients can flow through the neural network, the linear path ($u(x)$), nonlinear path ($h(x)$) and the stochastic path ($\sigma(x)$). The flow of gradients through these different pathways across different layers makes the optimization of our activation function easier.

At test time, we used the expectation of Eqn 13.4.7 in order to get deterministic units,

$$\text{E}_\xi[\phi(x, \xi)] = \alpha h(x) + (1-\alpha)u(x) + d(x)\sigma(x)\text{E}_\xi[\epsilon] \tag{13.4.8}$$

If $\epsilon = \xi$, then $\text{E}_\xi[\epsilon]$ is 0. Otherwise if $\epsilon = |\xi|$, then $\text{E}_\xi[\epsilon]$ is $\sqrt{\frac{2}{\pi}}$.

To illustrate the effect of α and noisy activation of the hard-tanh, We provide plots of our stochastic activation functions in Fig 13.4.

Algorithm 4 Noisy Activations with Half-Normal Noise for Hard-Saturating Functions

$\Delta \leftarrow h(x) - u(x)$
$d(x) \leftarrow -\operatorname{sgn}(x)\operatorname{sgn}(1-\alpha)$
$\sigma(x) \leftarrow c\ (g(p\Delta) - 0.5)^2$
$\xi \sim \mathcal{N}(0,\ 1)$
$\phi(x, \xi) \leftarrow \alpha h(x) + (1-\alpha)u(x) + (d(x)\sigma(x)|\xi|)$

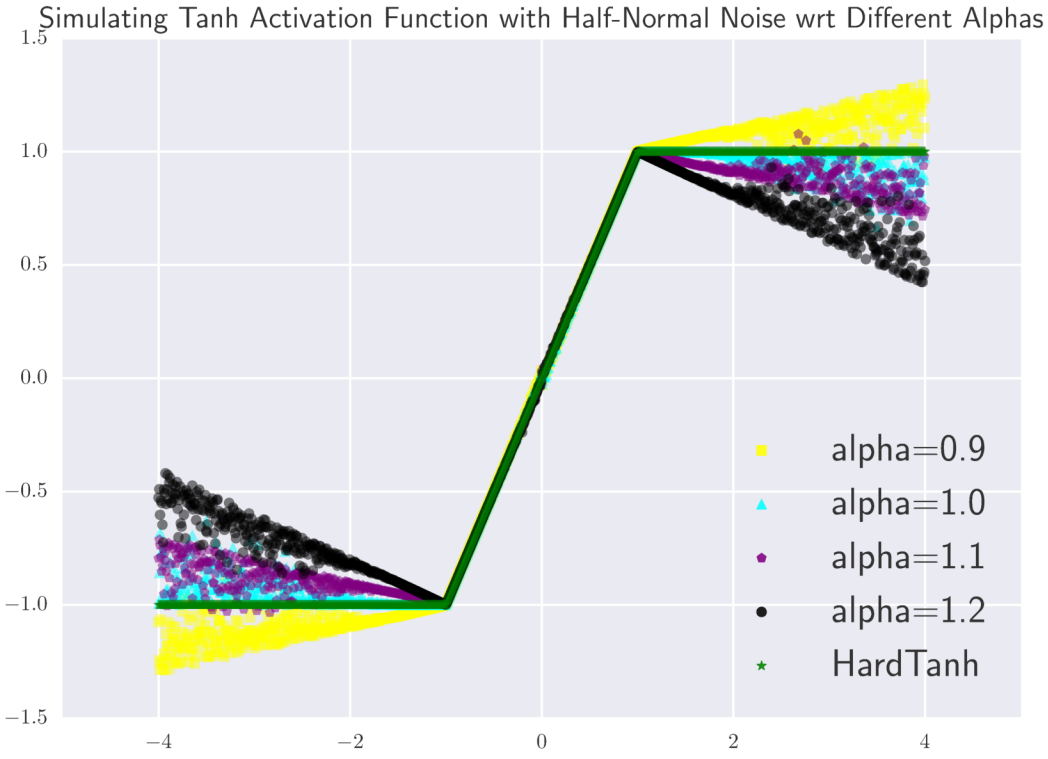


FIG. 13.4. Stochastic behavior of the proposed noisy activation function with different α values and with noise sampled from the Normal distribution, approximating the hard-tanh nonlinearity (in bold green).

13.5. Adding Noise to Input of the Function

Adding noise with fixed standard deviation to the input of the activation function has been investigated for ReLU activation functions Nair and Hinton (2010); Bengio *et al.* (2013b).

$$\phi(x, \xi) = h(x + \sigma\xi) \text{ and } \xi \sim \mathcal{N}(0, 1). \tag{13.5.1}$$

In Eqn 13.5.1, we provide a parametrization of the noisy activation function. σ can be either learned as in Eqn 13.4.4 or fixed as a hyperparameter.

The condition in Eqn 13.3.1 is satisfied only when σ is learned. Experimentally we found small values of σ to work better. When σ is fixed and small, as x gets larger and further away from the threshold x_t, noise will less likely be able to push the activations back to the linear regime. We also investigated the effect of injecting input noise when the activations saturate:

$$\phi(x, \xi) = h(x + \mathbf{1}_{|x| \geq |x_t|}(\sigma\xi)) \text{ and } \xi \sim \mathcal{N}(0, 1). \tag{13.5.2}$$

13.6. Experimental Results

In our experiments, we used noise only during training: at test time we replaced the noise variable with its expected value. We performed our experiments with just a drop-in replacement of the activation functions in existing experimental setups, without changing the previously set hyper-parameters. Hence it is plausible one could obtain better results by performing a careful hyper-parameter tuning for the models with noisy activation functions. In all our experiments, we initialized p uniform randomly from the range $[-1,\ 1]$.

We provide experimental results using noisy activations with normal **(NAN)**, half-normal noise **(NAH)**, normal noise at the input of the function **(NANI)**, normal noise at the input of the function with learned σ **(NANIL)** and normal noise injected to the input of the function when the unit saturates **(NANIS)**. Codes for different types of noisy activation functions can be found at `https://github.com/caglar/noisy_units`.

13.6.1. Exploratory Analysis

As a sanity-check, we performed small-scale control experiments, in order to observe the behavior of the noisy units.

We trained 3−layer MLP on a dataset generated from a mixture of 3 Gaussian distributions with different means and standard deviations. Each layer of the MLP contains 8-hidden units. Both the model with tanh and noisy− tanh activations was able to solve this task almost perfectly. By using the learned p values, in Figure 13.5 and 13.6, we showed the scatter plot of the activations of each unit at each layer and the derivative function of each unit at each layer with respect to its input.

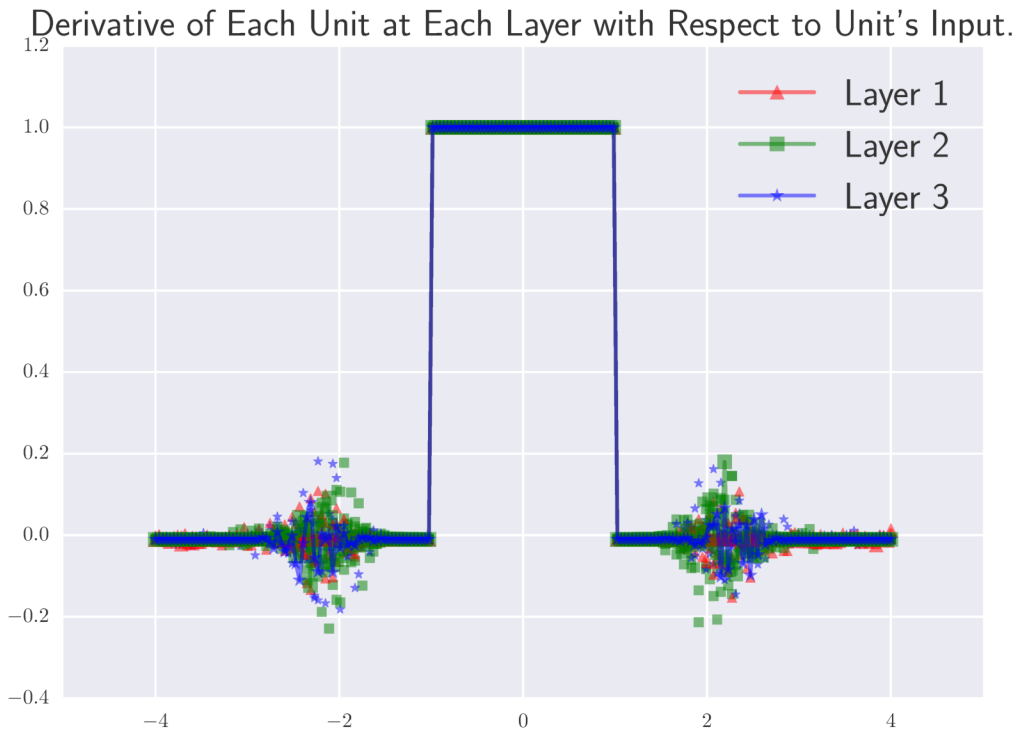


FIG. 13.5. Derivatives of each unit at each layer with respect to its input for a three-layered MLP trained on a dataset generated by three normal distributions with different means and standard deviations. In other words learned $\frac{\partial \phi(x_i^k, \xi_i^k)}{\partial x_i^k}$ at the end of training for i^{th} unit at k^{th} layer. ξ^k is sampled from Normal distribution with $\alpha = 1$.

13.6.2. Learning to Execute

The problem of predicting the output of a short program introduced by Zaremba and Sutskever (2014) [1] proved challenging for modern deep learning architectures. The authors had to use curriculum learning (Bengio, Louradour, Collobert, and Weston, 2009b) to let the model capture knowledge about the easier examples first and increase the level of difficulty of the examples further down the training.

We replaced all sigmoid and tanh non-linearities in the reference model with their noisy counterparts. We changed the default gradient clipping to 5 from 10 in order to avoid numerical stability problems. When evaluating a network, the length (number of lines) of

[1]The code is residing at `https://github.com/wojciechz/learning_to_execute`. We thank authors for making it publicly available.

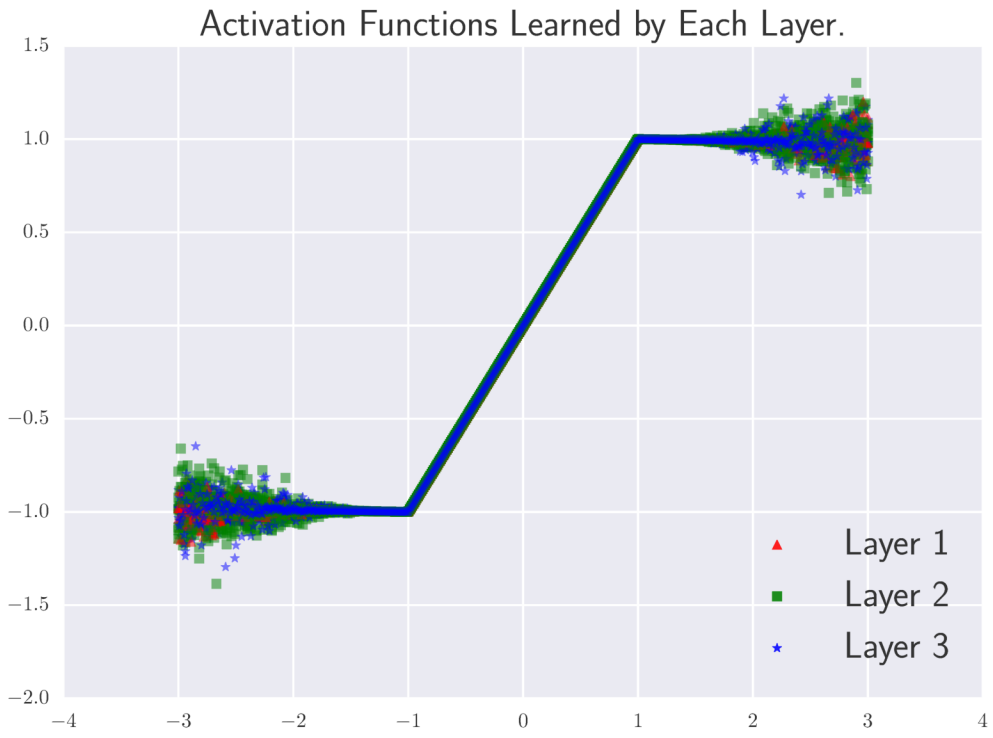


FIG. 13.6. Activations of each unit at each layer of a three-layer MLP trained on a dataset generated by three normal distributions with different means and standard deviations. In other words learned $\phi(x_i^k, \xi_i^k)$ at the end of training for i^{th} unit at k^{th} layer. ξ^k is sampled from Half-Normal distribution with $\alpha = 1$.

the executed programs was set to 6 and nesting was set to 3, which are default settings in the released code for these tasks. Both the reference model and the model with noisy activations were trained with "combined" curriculum which is the most sophisticated and the best performing one.

Our results show that applying the proposed activation function leads to better performance than that of the reference model. Moreover it shows that the method is easy to combine with a non-trivial learning curriculum. The results are presented in Table 13.1 and in Figure 13.8

TAB. 13.1. Performance of the noisy network on the *Learning to Execute* / task. Just changing the activation function to the proposed noisy one yielded about 2.5% improvement in accuracy.

Model name	Test Accuracy
Reference Model	46.45%
Noisy Network(**NAH**)	**48.09%**

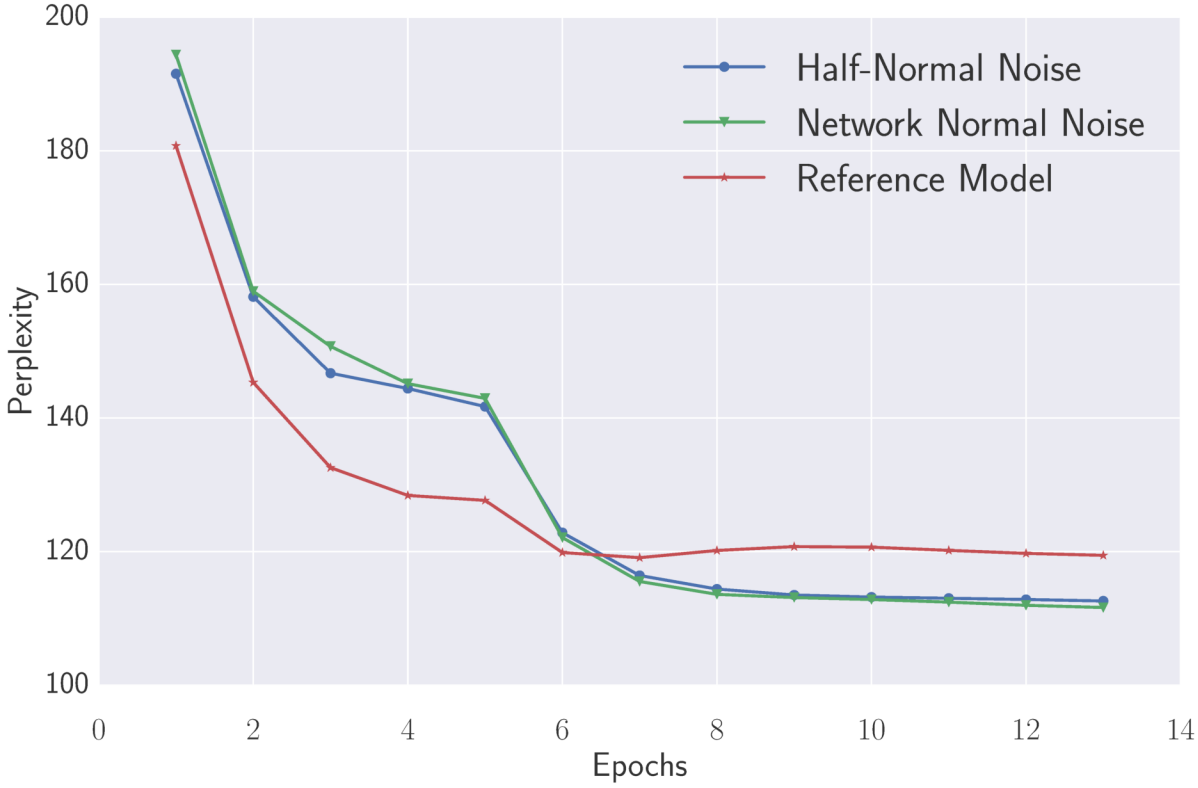


FIG. 13.7. Learning curves of validation perplexity for the LSTM language model on word level on Penntreebank dataset.

13.6.3. Penntreebank Experiments

We trained a 2−layer word-level LSTM language model on Penntreebank. We used the same model proposed by Zaremba *et al.* (2014). [2]We simply replaced all sigmoid and tanh units with noisy hard-sigmoid and hard-tanh units. The reference model is a well-finetuned strong baseline from Zaremba *et al.* (2014). For the noisy experiments we used exactly the

[2]We used the code provided in `https://github.com/wojzaremba/lstm`

FIG. 13.8. Training curves of the reference model (Zaremba and Sutskever, 2014) and its noisy variant on the "Learning To Execute" problem. The noisy network converges faster and reaches a higher accuracy, showing that the noisy activations help to better optimize for such hard to optimize tasks.

same setting, but decreased the gradient clipping threshold to 5 from 10. We provide the results of different models in Table 13.2. In terms of validation and test performance we did not observe big difference between the additive noise from Normal and half-Normal distributions, but there is a substantial improvement due to noise, which makes this result the new state-of-the-art on this task, as far as we know.

13.6.4. Neural Machine Translation Experiments

We have trained a neural machine translation (NMT) model on the Europarl dataset with the neural attention model Bahdanau *et al.* (2014).[3] We have replaced all sigmoid and tanh units with their noisy counterparts. We have scaled down the weight matrices initialized to be orthogonal scaled by multiplying with 0.01. Evaluation is done on the `newstest2011` test set.

[3]Again, we have used existing code, provided in `https://github.com/kyunghyuncho/dl4mt-material`, and only changed the nonlinearities

Tab. 13.2. Penntreebank word-level comparative perplexities. We only replaced in the code from Zaremba *et al.* (2014) the sigmoid and tanh by corresponding noisy variants and observe a substantial improvement in perplexity, which makes this the state-of-the-art on this task.

	Valid ppl	Test ppl
Noisy LSTM + **NAN**	111.7	**108.0**
Noisy LSTM + **NAH**	112.6	**108.7**
LSTM (Reference)	119.4	115.6

Tab. 13.3. Image Caption Generation on Flickr8k. This time we added noisy activations in the code from Xu *et al.* (2015b) and obtain substantial improvements on the higher-order BLEU scores and the METEOR metric, as well as in NLL. Soft attention and hard attention here refers to using backprop versus REINFORCE when training the attention mechanism. We fixed $\sigma = 0.05$ for NANI and $c = 0.5$ for both NAN and NANIL.

	BLEU -1	BLEU-2	BLEU-3	BLEU-4	METEOR	Test NLL
Soft Attention (Sigmoid and Tanh) (Baseline)	**67**	44.8	29.9	19.5	18.9	40.33
Soft Attention (**NAH Sigmoid & Tanh**)	66	**45.8**	**30.69**	**20.9**	**20.5**	40.17
Soft Attention (**NAH Sigmoid & Tanh** wo dropout)	64.9	44.2	**30.7**	**20.9**	20.3	**39.8**
Soft Attention (**NANI Sigmoid & Tanh**)	66	45.0	30.6	20.7	**20.5**	40.0
Soft Attention (**NANIL Sigmoid & Tanh**)	66	44.6	30.1	20.0	**20.5**	39.9
Hard Attention (Sigmoid and Tanh)	67	45.7	31.4	21.3	19.5	-

Tab. 13.4. Neural machine Translation on Europarl. Using existing code from Bahdanau *et al.* (2014) with nonlinearities replaced by their noisy versions, we find much improved performance (2 BLEU points is considered significant for machine translation). We also see that simply using the hard versions of the nonlinearities buys about half of the gain.

	Valid nll	BLEU
Sigmoid and Tanh NMT (Reference)	65.26	20.18
Hard-Tanh and Hard-Sigmoid NMT	64.27	21.59
Noisy (**NAH**) Tanh and Sigmoid NMT	**63.46**	**22.57**

All models are trained with early-stopping. We also compare with a model with hard-tanh and hard-sigmoid units and our model using noisy activations was able to outperform both, as shown in Table 13.4. Again, we see a substantial improvement (more than 2 BLEU points) with respect to the reference for English to French machine translation.

13.6.5. Image Caption Generation Experiments

We evaluated our noisy activation functions on a network trained on the Flickr8k dataset. We used the soft neural attention model proposed in Xu *et al.* (2015b) as our reference model.[4] We scaled down the weight matrices initialized to be orthogonal scaled by multiplying with 0.01. As shown in Table 13.3, we were able to obtain better results than the reference model and our model also outperformed the best model provided in Xu *et al.* (2015b) in terms of Meteor score.

(Xu *et al.*, 2015b)'s model was using dropout with the ratio of 0.5 on the output of the LSTM layers and the context. We have tried both with and without dropout, as in Table 13.3, we observed improvements with the addition of dropout to the noisy activation function. But the main improvement seems to be coming with the introduction of the noisy activation functions since the model without dropout already outperforms the reference model.

13.6.6. Experiments with Continuation

We performed experiments to validate the effect of annealing the noise to obtain a continuation method for neural networks.

We designed a new task where, given a random sequence of integers, the objective is to predict the number of unique elements in the sequence. We use an LSTM network over the input sequence, and performed a time average pooling over the hidden states of LSTM to obtain a fixed-size vector. We feed the pooled LSTM representation into a simple (one hidden-layer) ReLU MLP in order to predict the unique number of elements in the input sequence. In the experiments we fixed the length of input sequence to 26 and the input values are between 0 and 10. In order to anneal the noise, we started training with the scale hyperparameter of the standard deviation of noise with $c = 30$ and annealed it down to 0.5 with the schedule of $\frac{c}{\sqrt{t+1}}$ where t is being incremented at every 200 minibatch updates. When noise annealing is combined with a curriculum strategy (starting with short sequences first and gradually increase the length of the training sequences), the best models are obtained.

As a second test, we used the same annealing procedure in order to train a Neural Turing Machine (NTM) on the associative recall task Graves *et al.* (2014). We trained our model with a minimum of 2 items and a maximum of 16 items. We show results of the NTM with noisy activations in the controller, with annealed noise, and compare with a regular NTM in terms of validation error. As can be seen in Figure 13.9, the network using noisy activation

[4]We used the code provided at `https://github.com/kelvinxu/arctic-captions`.

TAB. 13.5. Experimental results on the task of finding the unique number of elements in a random integer sequence. This illustrates the effect of annealing the noise level, turning the training procedure into a continuation method. Noise annealing yields better results than the curriculum.

	Test Error %
LSTM+MLP(Reference)	33.28
Noisy LSTM+MLP(**NAN**)	31.12
Curriculum LSTM+MLP	14.83
Noisy LSTM+MLP(**NAN**) Annealed Noise	**9.53**
Noisy LSTM+MLP(**NANIL**) Annealed Noise	20.94

converges much faster and nails the task, whereas the original network failed to approach a low error.

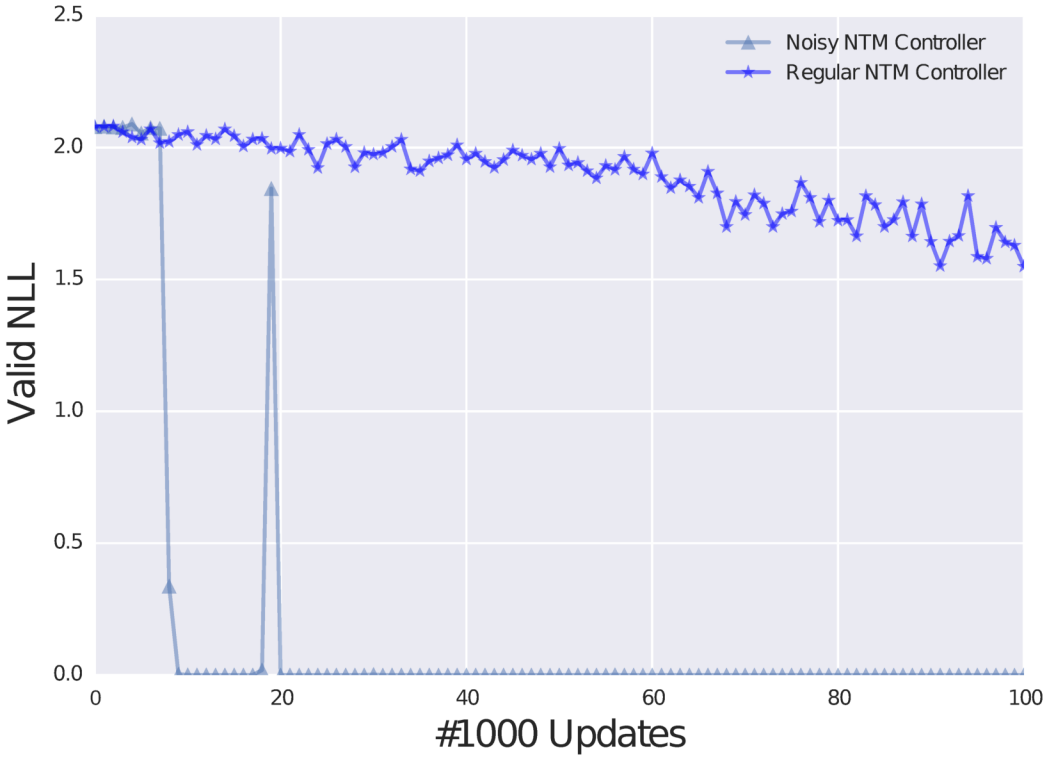


FIG. 13.9. Validation learning curve of NTM on Associative recall task evaluated over items of length 2 and 16. The NTM with noisy controller converges much faster and solves the task.

13.7. Conclusion

Nonlinearities in neural networks are both a blessing and a curse. A blessing because they allow to represent more complicated functions and a curse because that makes the optimization more difficult. For example, we have found in our experiments that using a hard version (hence more nonlinear) of the sigmoid and tanh nonlinearities often improved results. In the past, various strategies have been proposed to help deal with the difficult optimization problem involved in training some deep networks, including curriculum learning, which is an approximate form of continuation method. Earlier work also included softened versions of the nonlinearities that are gradually made harder during training. Motivated by this prior work, we introduce and formalize the concept of noisy activations as a general framework for injecting noise in nonlinear functions so that large noise allows SGD to be more exploratory. We propose to inject the noise to the activation functions either at the input of the function or at the output where unit would otherwise saturate, and allow gradients to flow even in that case. We show that our noisy activation functions are easier to optimize. It also, achieves better test errors, since the noise injected to the activations also regularizes the model as well. Even with a fixed noise level, we found the proposed noisy activations to outperform their sigmoid or tanh counterpart on different tasks and datasets, yielding state-of-the-art or competitive results with a simple modification, for example on PennTreebank. In addition, we found that annealing the noise to obtain a continuation method could further improved performance.

Chapter 14

CONCLUSION

The focus of this thesis is my recent works in the direction of improving "Natural Language Understanding" systems. In order to tackle this problem, we focus on five different aspects where improvements are achieved,

(1) **Improving the Ways to Deal with Rare Words**: In (Gulcehre *et al.*, 2016d), we have proposed a method to enable NLP models to learn to copy some of the words from the context, instead of directly predicting them via the softmax output. This type of method has been successful in summarization and question answering (See *et al.*, 2017; Nallapati *et al.*, 2016b).

(2) **Improving the Memory Aspect of the Algorithms for NLU**: We have discussed issues with existing deep learning approaches on natural language understanding systems when learning to represent long-term dependencies. In (Gulcehre, Chandar, Cho, and Bengio, 2016a) and in the followup work (Gulcehre *et al.*, 2017a), we have explored different models and methods to deal with the problems of learning long-term dependencies for neural networks.

(3) **Improving the Reasoning**: Most of the important applications of NLU rely on the performance of the reasoning abilities of the model. The reasoning can be either in the form of inferring relationships between different entities (Sukhbaatar *et al.*, 2015; Hermann and Blunsom, 2014) or based on the information that is not directly represented in the input.

(4) **Improving the Training of NLU Models**: Most of the NLU tasks can be basically cast as sequence processing problems (Winograd, 1972; Hermann *et al.*, 2015; Hermann, Hill, Green, Wang, Faulkner, Soyer, Szepesvari, Czarnecki, Jaderberg, Teplyashin, *et al.*, 2017). Some of the sequence processing problems with long-term dependencies or the ones that involve complicated reasoning can be difficult to learn. It is possible to tackle those training issues either by changing the architecture, e.g.: by changing the activation function or improving the optimization methods. In (Gulcehre *et al.*, 2016c) and (Gulcehre *et al.*, 2016b), we both explored changing the activation function

and introducing a curriculum for the sequence models. In (Gulcehre *et al.*, 2016a), we have proposed a curriculum learning strategy for the memory models with the discrete addressing method.

(5) **Improving the Sample Efficiency**: The sequence to sequence models are notoriously known to require lots of training data in order to perform well in a particular task. However, for some domains, it might be difficult to obtain large enough labeled training sets. In (Gulcehre *et al.*, 2015), we have explored the use of uni-lingual language modeling corpora in order to improve the performance of the sequence to sequence models, mainly on the low-resource translation tasks.

14.1. Problems of Training Sequence to Sequence Models for NLU

The sequence to sequence models with or without attention mechanisms have been very successful in several different applications for NLU. However, there are still several problems with the existing models. In this section, we summarize some of them and the future work should focus on solving some of those issues. I will not propose any solutions or why they are happening but point out some of the empirical observations that we come across a lot when we are generating text with "Sequence to Sequence" models.

14.1.1. Models can Hallucinate

In most of the applications of the sequence to sequence models, in particular, when we have a very strong decoder, the model tends to ignore the context and generate the target sequence, just by using the language model in the decoder. This tends to cause the model just generate translations that are completely irrelevant. In our work (Gulcehre, Dutil, Trischler, and Bengio, 2017b), we have proposed a mechanism to improve the attention that prevents the decoder to ignore the context. However, more research needs to be done to improve our models.

14.1.2. Repetitions in the Generated Samples

A common problem that occurs and we have observed in the outputs of the neural machine translation systems is that the models tend to stutter and repeat a very common phrase again and again towards the end of the sequence. This tends to happen in particular if the model is being trained on low data regime. For example,

“The president of United States is Barrack Obama, Barrack Obama Barrack Obama ..."

14.1.3. Teacher Forcing

As we have discussed earlier, teacher forcing is needed to train the recurrent neural language models with the maximum likelihood objective. However, one of the downsides of this approach is the model never gets exposed to its own predictions during the training as an input. However, during test-time, we provide the models own predictions as input. This discrepancy between the training and the test time tends to create a deficiency in the generated samples once the model starts to make mistakes and eventually the whole sampling trajectory diverges. The approaches such as scheduled sampling (Bengio, Vinyals, Jaitly, and Shazeer, 2015) have been proposed to address this issue, but the problem is still far from being solved completely.

14.1.4. Lack of Diversity

One of the most pressing issue with the generated samples is the lack of diversity in the outputs of the sequence-to-sequence models. This issue has been more pronounced in the applications such as Dialogue Generation. It is well-known that in practice neural dialogue models tend to generate less diverse, short and trivial dialogues (Sordoni, Galley, Auli, Brockett, Ji, Mitchell, Nie, Gao, and Dolan, 2015; Serban, García-Durán, Gulcehre, Ahn, Chandar, Courville, and Bengio, 2016b; Vinyals and Le, 2015). There have been several works recently on addressing this issue (Li, Galley, Brockett, Gao, and Dolan, 2015a; Shao, Gouws, Britz, Goldie, Strope, and Kurzweil, 2017), however, the issue is still far from being solved.

Bibliography

Allgower, E. L. and Georg, K. (1980). *Numerical Continuation Methods. An Introduction*. Springer-Verlag.

Amodei, D., Anubhai, R., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Chen, J., Chrzanowski, M., Coates, A., Diamos, G., *et al.* (2015). Deep speech 2: End-to-end speech recognition in english and mandarin. *arXiv preprint arXiv:1512.02595*.

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. (2016). Learning to compose neural networks for question answering. *arXiv preprint arXiv:1601.01705*.

Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Zitnick, C. L., and Parikh, D. (2015). VQA: visual question answering. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 2425–2433.

Arjovsky, M., Shah, A., and Bengio, Y. (2016). Unitary evolution recurrent neural networks. *ICML 2016*.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations (ICLR)*.

Bahdanau, D., Brakel, P., Xu, K., Goyal, A., Lowe, R., Pineau, J., Courville, A., and Bengio, Y. (2016a). An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*.

Bahdanau, D., Chorowski, J., Serdyuk, D., Brakel, P., and Bengio, Y. (2016b). End-to-end attention-based large vocabulary speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 4945–4949. IEEE.

Baldi, P. and Hornik, K. (1989). Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, **2**(1), 53–58.

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. (2015). Automatic differentiation in machine learning: a survey. *arXiv preprint arXiv:1502.05767*.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179.

Bengio, Y. *et al.* (2009a). Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, **2**(1), 1–127.

Bengio, Y. and Senécal, J.-S. (2008). Adaptive importance sampling to accelerate training of a neural probabilistic language model. *Neural Networks, IEEE Transactions on*, **19**(4), 713–722.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, **5**(2), 157–166.

Bengio, Y., Ducharme, R., and Vincent, P. (2000). A neural probabilistic language model. In *Advances in Neural Information Processing Systems*, pages 932–938.

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *J. Mach. Learn. Res.*, **3**, 1137–1155.

Bengio, Y., LeCun, Y., *et al.* (2007). Scaling learning algorithms towards ai. *Large-scale kernel machines*, **34**(5), 1–41.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009b). Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM.

Bengio, Y., Boulanger-Lewandowski, N., and Pascanu, R. (2013a). Advances in optimizing recurrent networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8624–8628. IEEE.

Bengio, Y., Léonard, N., and Courville, A. (2013b). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Bengio, Y., Courville, A., and Vincent, P. (2013c). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, **35**(8), 1798–1828.

Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. ACM.

Bolukbasi, T., Chang, K.-W., Zou, J. Y., Saligrama, V., and Kalai, A. T. (2016). Man is to computer programmer as woman is to homemaker? debiasing word embeddings. In *Advances in Neural Information Processing Systems*, pages 4349–4357.

Bordes, A., Usunier, N., Chopra, S., and Weston, J. (2015). Large-scale simple question answering with memory networks. *arXiv preprint arXiv:1506.02075*.

Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer.

Bottou, L., Curtis, F. E., and Nocedal, J. (2016). Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838*.

Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. (2015). A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*.

Bozsahin, C. (2012). *Combinatory linguistics*. Walter de Gruyter.

Brown, P. F., Pietra, V. J. D., Pietra, S. A. D., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, **19**(2), 263–311.

Cettolo, M., Girardi, C., and Federico, M. (2012). Wit3: Web inventory of transcribed and translated talks. *Proceedings of the 16th Conference of the European Association for Machine Translation (EAMT)*, pages 261–268.

Chan, W., Jaitly, N., Le, Q. V., and Vinyals, O. (2015). Listen, attend and spell. *arXiv preprint arXiv:1508.01211*.

Chandar, S., Ahn, S., Larochelle, H., Vincent, P., Tesauro, G., and Bengio, Y. (2016). Hierarchical memory networks. *arXiv preprint arXiv:1605.07427*.

Chen, D., Bolton, J., and Manning, C. D. (2016a). A thorough examination of the cnn/daily mail reading comprehension task. *arXiv preprint arXiv:1606.02858*.

Chen, T., Xu, B., Zhang, C., and Guestrin, C. (2016b). Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*.

Cheng, J. and Lapata, M. (2016). Neural summarization by extracting sentences and words. *arXiv preprint arXiv:1603.07252*.

Chiang, D. (2005). A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 263–270. Association for Computational Linguistics.

Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. (2014a). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *In EMNLP*.

Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014b). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on information theory*, **2**(3), 113–124.

Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. (2015). The loss surfaces of multilayer networks. In *AISTATS*.

Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015). Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems*, pages 577–585.

Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, **abs/1412.3555**.

Chung, J., Cho, K., and Bengio, Y. (2016). A character-level decoder without explicit segmentation for neural machine translation. *arXiv preprint arXiv:1603.06147*.

Cole, D. (2009). The chinese room argument. *Stanford Encyclopedia of Philosophy*.

Collins, M. and Koo, T. (2005). Discriminative reranking for natural language parsing. *Computational Linguistics*, **31**(1), 25–70.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, **12**(Aug), 2493–2537.

Cooijmans, T., Ballas, N., Laurent, C., and Courville, A. (2017). Recurrent batch normalization. *ICLR 2017, Toullone France*.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, **2**(4), 303–314.

Dai, J., He, K., and Sun, J. (2015). Convolutional feature masking for joint object and stuff segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3992–4000.

Damerau, F. J. (1971). *Markov models and linguistic theory: an experimental study of a model for English*. Number 95. Mouton De Gruyter.

Dauphin, Y., de Vries, H., and Bengio, Y. (2015). Equilibrated adaptive learning rates for non-convex optimization. In *Advances in Neural Information Processing Systems*, pages 1504–1512.

Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941.

Dekel, O., Gilad-Bachrach, R., Shamir, O., and Xiao, L. (2011). Optimal distributed online prediction. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 713–720.

Devlin, J., Uesato, J., Bhupatiraju, S., Singh, R., Mohamed, A.-r., and Kohli, P. (2017). Robustfill: Neural program learning under noisy i/o. *arXiv preprint arXiv:1703.07469*.

Dietterich, T. G. (2000). Hierarchical reinforcement learning.

Dodge, J., Gane, A., Zhang, X., Bordes, A., Chopra, S., Miller, A., Szlam, A., and Weston, J. (2015). Evaluating prerequisite qualities for learning end-to-end dialog systems. *CoRR*, **abs/1511.06931**.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, **14**(2), 179–211.

Elman, J. L. (1993). Learning and development in neural networks: The importance of starting small. *Cognition*, **48**(1), 71–99.

Firat, O., Cho, K., and Bengio, Y. (2016). Multi-way, multilingual neural machine translation with a shared attention mechanism. *arXiv preprint arXiv:1601.01073*.

Ge, R., Huang, F., Jin, C., and Yuan, Y. (2015). Escaping from saddle points—online stochastic gradient for tensor decomposition. *arXiv preprint arXiv:1503.02101*.

Gemici, M., Hung, C.-C., Santoro, A., Wayne, G., Mohamed, S., Rezende, D. J., Amos, D., and Lillicrap, T. (2017). Generative temporal models with memory. *arXiv preprint arXiv:1702.04649*.

Gillick, D., Brunk, C., Vinyals, O., and Subramanya, A. (2015). Multilingual language processing from bytes. *arXiv preprint arXiv:1512.00103*.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323.

Goh, G. (2017). Why momentum really works. *Distill*.

Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013a). Maxout networks. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1319–1327.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep learning. Book in preparation for MIT Press.

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013b). Maxout networks. *arXiv preprint arXiv:1302.4389*.

Grave, E., Joulin, A., and Usunier, N. (2016). Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*.

Graves, A. (2011). Practical variational inference for neural networks. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2348–2356.

Graves, A. and Schmidhuber, J. (2009). Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in neural information processing systems*, pages 545–552.

Graves, A., Jaitly, N., and Mohamed, A.-R. (2013). Hybrid speech recognition with deep bidirectional LSTM. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 273–278.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., *et al.* (2016). Hybrid computing using a neural network with dynamic external memory. *Nature*, **538**(7626), 471–476.

Greensmith, E., Bartlett, P. L., and Baxter, J. (2004). Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, **5**(Nov), 1471–1530.

Grefenstette, E., Hermann, K. M., Suleyman, M., and Blunsom, P. (2015). Learning to transduce with unbounded memory. In *Advances in Neural Information Processing Systems*, pages 1819–1827.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2016). Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*.

Griewank, A. (1992). Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods and software*, **1**(1), 35–54.

Gu, J., Lu, Z., Li, H., and Li, V. O. (2016). Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393*.

Gulcehre, C., Firat, O., Xu, K., Cho, K., Barrault, L., Lin, H.-C., Bougares, F., Schwenk, H., and Bengio, Y. (2015). On using monolingual corpora in neural machine translation. *arXiv preprint arXiv:1503.03535*.

Gulcehre, C., Chandar, S., Cho, K., and Bengio, Y. (2016a). Dynamic neural turing machine with soft and hard addressing schemes. *arXiv preprint arXiv:1607.00036*.

Gulcehre, C., Moczulski, M., Visin, F., and Bengio, Y. (2016b). Mollifying networks. *arXiv preprint arXiv:1608.04980*.

Gulcehre, C., Moczulski, M., Denil, M., and Bengio, Y. (2016c). Noisy activation functions. *arXiv preprint arXiv:1603.00391*.

Gulcehre, C., Ahn, S., Nallapati, R., Zhou, B., and Bengio, Y. (2016d). Pointing the unknown words. *arXiv preprint arXiv:1603.08148*.

Gulcehre, C., Chandar, S., and Bengio, Y. (2017a). Memory augmented neural networks with wormhole connections. *arXiv preprint arXiv:1701.08718*.

Gulcehre, C., Dutil, F., Trischler, A., and Bengio, Y. (2017b). Plan, attend, generate: Character-level neural machine translation with planning. In *Proceedings of Neural Information Processing (NIPS)*.

Gulcehre, C., Sotelo, J., Moczulski, M., and Bengio, Y. (2017c). A robust adaptive stochastic gradient method for deep learning. *arXiv preprint arXiv:1703.00788*.

Gutmann, M. U. and Hyvärinen, A. (2012). Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *The Journal of Machine Learning Research*, **13**(1), 307–361.

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., *et al.* (2014). Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*.

Hardt, M., Recht, B., and Singer, Y. (2015). Train faster, generalize better: Stability of stochastic gradient descent. *arXiv preprint arXiv:1509.01240*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

Henaff, M., Weston, J., Szlam, A., Bordes, A., and LeCun, Y. (2016). Tracking the world state with recurrent entity networks. *arXiv preprint arXiv:1612.03969*.

Hermann, K. and Blunsom, P. (2014). Multilingual distributed representations without word alignment. In *Proceedings of the Second International Conference on Learning Representations (ICLR 2014)*.

Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701.

Hermann, K. M., Hill, F., Green, S., Wang, F., Faulkner, R., Soyer, H., Szepesvari, D., Czarnecki, W., Jaderberg, M., Teplyashin, D., *et al.* (2017). Grounded language learning in a simulated 3d world. *arXiv preprint arXiv:1706.06551*.

Hill, F., Bordes, A., Chopra, S., and Weston, J. (2015). The goldilocks principle: Reading children's books with explicit memory representations. *arXiv preprint arXiv:1511.02301*.

Hinton, G. E., Rumelhart, D., and McClelland, J. (1984). Distributed representations. In *"Parallel distributed processing"*, chapter 3. MIT.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint*

arXiv:1207.0580.

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, page 91.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, **9.8**, 1735–1780.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, **2**(5), 359–366.

Huber, P. J. (1964). Robust estimation of a location parameter. *Ann. Math. Statist.*, **35**(1), 73–101.

Iyyer, M., Boyd-Graber, J. L., Claudino, L. M. B., Socher, R., and Daumé III, H. (2014). A neural network for factoid question answering over paragraphs. In *EMNLP*, pages 633–644.

Jang, E., Gu, S., and Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Jean, S., Cho, K., Memisevic, R., and Bengio, Y. (2014). On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007*.

Jean, S., Cho, K., Memisevic, R., and Bengio, Y. (2015). On using very large target vocabulary for neural machine translation.

Johnson, J., Hariharan, B., van der Maaten, L., Hoffman, J., Fei-Fei, L., Zitnick, C. L., and Girshick, R. B. (2017). Inferring and executing programs for visual reasoning. In *ICCV*, pages 3008–3017.

Jordan, M. I. (1986). Serial order: A parallel distributed processing approach. *Advances in psychology*, **121**, 471–495.

Joshi, A. K. and Schabes, Y. (1997). Tree-adjoining grammars. In *Handbook of formal languages*, pages 69–123. Springer.

Joulin, A. and Mikolov, T. (2015). Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in Neural Information Processing Systems*, pages 190–198.

Jurafsky, D. (2000). Speech and language processing: An introduction to natural language processing. *Computational linguistics, and speech recognition*.

Kalchbrenner, N. and Blunsom, P. (2013). Recurrent continuous translation models. In *Proceedings of the ACL Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1700–1709. Association for Computational Linguistics.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Karpathy, A., Johnson, J., and Fei-Fei, L. (2015). Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*.

Kawaguchi, K. (2016). Deep learning without poor local minima. In *Advances In Neural Information Processing Systems*, pages 586–594.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. and Adam, J. B. (2015). A method for stochastic optimization. In *International Conference on Learning Representation*.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*.

Kirkpatrick, S., Jr., C. D. G., , and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, **220**, 671–680.

Koehn, P. (2009). *Statistical machine translation*. Cambridge University Press.

Koehn, P. (2010). *Statistical Machine Translation*. Cambridge University Press, New York, NY, USA.

Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 48–54, Stroudsburg, PA, USA. Association for Computational Linguistics.

Kukich, K. (1992). Techniques for automatically correcting words in text. *ACM Computing Surveys (CSUR)*, **24**(4), 377–439.

Kumar, A., Irsoy, O., Su, J., Bradbury, J., English, R., Pierce, B., Ondruska, P., Gulrajani, I., and Socher, R. (2015). Ask me anything: Dynamic memory networks for natural language processing. *CoRR*, **abs/1506.07285**.

Le, Q. V., Jaitly, N., and Hinton, G. E. (2015). A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, **521**(7553), 436–444.

Lee, J., Cho, K., and Hofmann, T. (2016). Fully character-level neural machine translation without explicit segmentation. *arXiv preprint arXiv:1610.03017*.

Lenat, D. B., Prakash, M., and Shepherd, M. (1985). Cyc: Using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. *AI magazine*, **6**(4), 65.

Li, J., Galley, M., Brockett, C., Gao, J., and Dolan, B. (2015a). A diversity-promoting objective function for neural conversation models. *arXiv preprint arXiv:1510.03055*.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. (2015b). Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2,

pages 1150–1157. Ieee.

Luo, Y., Chiu, C.-C., Jaitly, N., and Sutskever, I. (2016). Learning online alignments with continuous rewards policy gradient. *arXiv preprint arXiv:1608.01281*.

Luong, M.-T. and Manning, C. D. (2016). Achieving open vocabulary neural machine translation with hybrid word-character models. *arXiv preprint arXiv:1604.00788*.

Luong, M.-T., Sutskever, I., Le, Q. V., Vinyals, O., and Zaremba, W. (2015a). Addressing the rare word problem in neural machine translation. In *Proceedings of ACL*.

Luong, M.-T., Pham, H., and Manning, C. D. (2015b). Effective approaches to attention-based neural machine translation. *In Proceedings Of The Conference on Empirical Methods for Natural Language Processing (EMNLP 2015)*.

Luong, M.-T., Le, Q. V., Sutskever, I., Vinyals, O., and Kaiser, L. (2015c). Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*.

Luong, T., Sutskever, I., Le, Q. V., Vinyals, O., and Zaremba, W. (2014). Addressing the rare word problem in neural machine translation. *arXiv preprint arXiv:1410.8206*.

Maddison, C. J., Mnih, A., and Teh, Y. W. (2016). The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*.

Manning, C. D., Manning, C. D., and Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT press.

Matthews, D., Behne, T., Lieven, E., and Tomasello, M. (2012). Origins of the human pointing gesture: a training study. *Developmental science*, **15**(6), 817–829.

McGill, M. and Perona, P. (2017). Deciding how to decide: Dynamic routing in artificial neural networks. *arXiv preprint arXiv:1703.06217*.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. (2016). Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Mermelstein, P. (1976). Distance measures for speech recognition, psychological and instrumental. *Pattern recognition and artificial intelligence*, **116**, 374–388.

Mikolov, T., Kombrink, S., Deoras, A., Burget, L., and Cernocky, J. (2011). Rnnlm-recurrent neural network language modeling toolkit. In *Proc. of the 2011 ASRU Workshop*, pages 196–201.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Miller, A., Fisch, A., Dodge, J., Karimi, A.-H., Bordes, A., and Weston, J. (2016a). Key-value memory networks for directly reading documents. *arXiv preprint arXiv:1606.03126*.

Miller, A., Fisch, A., Dodge, J., Karimi, A., Bordes, A., and Weston, J. (2016b). Key-value memory networks for directly reading documents. *CoRR*, **abs/1606.03126**.

Mnih, A. and Gregor, K. (2014). Neural variational inference and learning in belief networks. *International Conference on Machine Learning, ICML*.

Mnih, A. and Kavukcuoglu, K. (2013). Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in Neural Information Processing Systems*, pages 2265–2273.

Mnih, V., Heess, N., Graves, A., *et al.* (2014). Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212.

Mobahi, H. (2016). Training recurrent neural networks by diffusion. *arXiv preprint arXiv:1601.04114*.

Mohit, B., Hwa, R., and Lavie, A. (2010). Using variable decoding weight for language model in statistical machine translation. AMTA.

Montufar, G. F., Pascanu, R., Cho, K., and Bengio, Y. (2014). On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932.

Morin, F. and Bengio, Y. (2005). Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. Citeseer.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

Nallapati, R., Zhou, B., Gulcehre, C., Xiang, B., *et al.* (2016a). Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*.

Nallapati, R., Zhai, F., and Zhou, B. (2016b). Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. *arXiv preprint arXiv:1611.04230*.

Nam, H., Ha, J.-W., and Kim, J. (2016). Dual attention networks for multimodal reasoning and matching. *arXiv preprint arXiv:1611.00471*.

Narayan, S., Papasarantopoulos, N., Lapata, M., and Cohen, S. B. (2017). Neural extractive summarization with side information. *arXiv preprint arXiv:1704.04530*.

Neelakantan, A., Vilnis, L., Le, Q. V., Sutskever, I., Kaiser, L., Kurach, K., and Martens, J. (2015). Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*.

Nguyen, T., Rosenberg, M., Song, X., Gao, J., Tiwary, S., Majumder, R., and Deng, L. (2016). Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint*

arXiv:1611.09268.

Och, F. J. and Ney, H. (2004). The alignment template approach to statistical machine translation. *Computational linguistics*, **30**(4), 417–449.

Och, F. J., Tillmann, C., Ney, H., *et al.* (1999). Improved alignment models for statistical machine translation. In *Proc. of the Joint SIGDAT Conf. on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 20–28.

Pascanu, R., Mikolov, T., and Bengio, Y. (2012). On the difficulty of training recurrent neural networks. *arXiv preprint arXiv:1211.5063*.

Pascanu, R., Gulcehre, C., Cho, K., and Bengio, Y. (2013a). How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013b). On the difficulty of training recurrent neural networks. *ICML (3)*, **28**, 1310–1318.

Pascanu, R., Montufar, G., and Bengio, Y. (2013c). On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv:*`1312.6098[cs.LG]`.

Pascanu, R., Gulcehre, C., Cho, K., and Bengio, Y. (2014). How to construct deep recurrent neural networks. In *Proceedings of the Second International Conference on Learning Representations (ICLR 2014)*.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, **4**(5), 1–17.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training.

Rae, J. W., Hunt, J. J., Harley, T., Danihelka, I., Senior, A., Wayne, G., Graves, A., and Lillicrap, T. P. (2016). Scaling memory-augmented neural networks with sparse reads and writes. In *Advances in NIPS*.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Reed, S. and de Freitas, N. (2016). Neural programmer-interpreters. *ICLR 2016*.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.

Rocktäschel, T., Grefenstette, E., Hermann, K. M., Kočiskỳ, T., and Blunsom, P. (2015). Reasoning about entailment with neural attention. *arXiv preprint arXiv:1509.06664*.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, **65**(6), 386.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986a). Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986b). Learning representations by back-propagating errors. *Nature*, **323**(6088), 533–536.

Rush, A. M., Chopra, S., and Weston, J. (2015a). A neural attention model for abstractive sentence summarization. *CoRR*, **abs/1509.00685**.

Rush, A. M., Chopra, S., and Weston, J. (2015b). A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 379–389.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A modern approach*, volume 25. Prentice-Hall, Egnlewood Cliffs.

Sagun, L., Bottou, L., and LeCun, Y. (2016). Singularity of the hessian in deep learning. *arXiv preprint arXiv:1611.07476*.

Sak, H., Güngör, T., and Saraçlar, M. (2007). Morphological disambiguation of turkish text with perceptron algorithm. In *Computational Linguistics and Intelligent Text Processing*, pages 107–118. Springer.

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. (2016). One-shot learning with memory-augmented neural networks. *ICML 2016*.

Saxe, A. M., McClelland, J. L., and Ganguli, S. (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*.

Schaul, T., Zhang, S., and LeCun, Y. (2013). No more pesky learning rates. *ICML (3)*, **28**, 343–351.

Schmidhuber, J. and Heil, S. (1995). Predictive coding with neural nets: Application to text compression. In *Advances in neural information processing systems*, pages 1047–1054.

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, **45**(11), 2673–2681.

Schwenk, H. (2007). Continuous space language models. *Comput. Speech Lang.*, **21**(3), 492–518.

See, A., Liu, P. J., and Manning, C. D. (2017). Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*.

Semeniuta, S., Severyn, A., and Barth, E. (2016). Recurrent dropout without memory loss. *arXiv preprint arXiv:1603.05118*.

Sennrich, R., Haddow, B., and Birch, A. (2015). Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Seo, M., Min, S., Farhadi, A., and Hajishirzi, H. (2016). Query-reduction networks for question answering. *arXiv preprint arXiv:1606.04582*.

Serban, I. V., Sordoni, A., Bengio, Y., Courville, A., and Pineau, J. (2016a). Building end-to-end dialogue systems using generative hierarchical neural network models. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*.

Serban, I. V., García-Durán, A., Gulcehre, C., Ahn, S., Chandar, S., Courville, A., and Bengio, Y. (2016b). Generating factoid questions with recurrent neural networks: The 30m factoid question-answer corpus. *Proc. of ACL*.

Shao, Y., Gouws, S., Britz, D., Goldie, A., Strope, B., and Kurzweil, R. (2017). Generating high-quality and informative conversation responses with sequence-to-sequence models. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2200–2209.

Shuyo, N. (2010). Language detection library for java.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International Conference on Machine Learning (ICML)*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., *et al.* (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, **529**(7587), 484–489.

Simmons, R. F., Klein, S., and McConlogue, K. (1964). Indexing and dependency logic for answering english questions. *Journal of the Association for Information Science and Technology*, **15**(3), 196–204.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sonoda, S. and Murata, N. (2015). Neural network with unbounded activation functions is universal approximator. *Applied and Computational Harmonic Analysis*.

Sordoni, A., Galley, M., Auli, M., Brockett, C., Ji, Y., Mitchell, M., Nie, J.-Y., Gao, J., and Dolan, B. (2015). A neural network approach to context-sensitive generation of conversational responses. *arXiv preprint arXiv:1506.06714*.

Sparck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, **28**(1), 11–21.

Steedman, M. and Baldridge, J. (2011). Combinatory categorial grammar. *Non-Transformational Syntax: Formal and Explicit Models of Grammar*, pages 181–224.

Sukhbaatar, S., Szlam, A., Weston, J., and Fergus, R. (2015). End-to-end memory networks. *arXiv preprint arXiv:1503.08895*.

Sun, G., Giles, C. L., and Chen, H. (1997). The neural network pushdown automaton: Architecture, dynamics and training. In *Adaptive Processing of Sequences and Data Structures, International Summer School on Neural Networks*, pages 296–345.

Sutskever, I., Martens, J., Dahl, G. E., and Hinton, G. E. (2013). On the importance of initialization and momentum in deep learning. *ICML (3)*, **28**, 1139–1147.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. (2016). Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, **4**.

Tomasello, M., Carpenter, M., and Liszkowski, U. (2007). A new look at infant pointing. *Child development*, **78**(3), 705–722.

Trischler, A., Wang, T., Yuan, X., Harris, J., Sordoni, A., Bachman, P., and Suleman, K. (2016). Newsqa: A machine comprehension dataset. *arXiv preprint arXiv:1611.09830*.

Vezhnevets, A., Mnih, V., Agapiou, J., Osindero, S., Graves, A., Vinyals, O., and Kavukcuoglu, K. (2016). Strategic attentive writer for learning macro-actions. In *Advances in Neural Information Processing Systems*, pages 3486–3494.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. (2017). Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*.

Vinyals, O. and Le, Q. (2015). A neural conversational model. *arXiv preprint arXiv:1506.05869*.

Vinyals, O., Fortunato, M., and Jaitly, N. (2015). Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700.

Visin, F., Kastner, K., Courville, A., Bengio, Y., Matteucci, M., and Cho, K. (2015). Reseg: A recurrent neural network for object segmentation. *CoRR, abs/1511.07053*, **2**.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, **78**(10), 1550–1560.

Werbos, P. J. (1994). *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting*, volume 1. John Wiley & Sons.

Weston, J., Chopra, S., and Bordes, A. (2015a). Memory networks. *In Proceedings Of The International Conference on Representation Learning (ICLR 2015)*. In Press.

Weston, J., Bordes, A., Chopra, S., and Mikolov, T. (2015b). Towards ai-complete question answering: a set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*.

Wierstra, D., Foerster, A., Peters, J., and Schmidhuber, J. (2007). Solving deep memory pomdps with recurrent policy gradients. In *International Conference on Artificial Neural Networks*, pages 697–706. Springer.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, **8**(3-4), 229–256.

Williams, R. J. and Peng, J. (1990). An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, **2**(4), 490–501.

Winograd, T. (1972). Understanding natural language. *Cognitive psychology*, **3**(1), 1–191.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., *et al.* (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Xiong, C., Zhong, V., and Socher, R. (2016a). Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604*.

Xiong, C., Merity, S., and Socher, R. (2016b). Dynamic memory networks for visual and textual question answering. *CoRR*, **abs/1603.01417**.

Xu, B., Wang, N., Chen, T., and Li, M. (2015a). Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. (2015b). Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057.

Yang, G. (2016). Lie access neural turing machine. *arXiv preprint arXiv:1602.08671*.

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., and Hovy, E. (2016). Hierarchical attention networks for document classification. In *Proceedings of NAACL-HLT*, pages 1480–1489.

Yao, L., Torabi, A., Cho, K., Ballas, N., Pal, C., Larochelle, H., and Courville, A. (2015). Describing videos by exploiting temporal structure. In *Computer Vision (ICCV), 2015 IEEE International Conference on*. IEEE.

Yılmaz, E., El-Kahlout, I. D., Aydın, B., Ozil, Z. S., and Mermer, C. (2013). Tubitak turkish-english submissions for iwslt 2013. *Proceedings of the 10th International Workshop on Spoken Language Translation (IWSLT)*, pages 152–159.

Yuan, X., Wang, T., Gulcehre, C., Sordoni, A., Bachman, P., Subramanian, S., Zhang, S., and Trischler, A. (2017). Machine comprehension by text-to-text neural question generation.

arXiv preprint arXiv:1705.02012.

Zaremba, W. and Sutskever, I. (2014). Learning to execute. *arXiv preprint arXiv:1410.4615*.

Zaremba, W. and Sutskever, I. (2015). Reinforcement learning neural turing machines. *arXiv preprint arXiv:1505.00521*, **362**.

Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Zaremba, W., Mikolov, T., Joulin, A., and Fergus, R. (2015). Learning simple algorithms from examples. *arXiv preprint arXiv:1511.07275*.

Zeiler, M. D. (2012). ADADELTA: An adaptive learning rate method. *arXiv:*`1212.5701 [cs.LG]`.

Zhang, W., Yu, Y., and Zhou, B. (2015). Structured memory for neural turing machines. *arXiv preprint arXiv:1510.03931*.

Zhu, Q., Yeh, M.-C., Cheng, K.-T., and Avidan, S. (2006). Fast human detection using a cascade of histograms of oriented gradients. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1491–1498. IEEE.

Zoph, B., Deniz, Y., Jonathan, M., and Knight, K. (2016). Transfer learning for low-resource neural machine translation. *CoRR*, **abs/1604.02201**.