Université de Montréal

Towards Learning Sentence Representation with Self-supervision

par Seyedarian Hosseini

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Juillet, 2019

Université de Montréal
Faculté des arts et des sciences

Ce mémoire intitulé:

Towards Learning Sentence Representation with Self-supervision

présenté par:

Seyedarian Hosseini

a été évalué par un jury composé des personnes suivantes:

Guy Lapalme,	président-rapporteur
Aaron Courville,	directeur de recherche
Yoshua Bengio,	membre du jury

Mémoire accepté le: ....

Résumé

Ces dernières années, il y a eu un intérêt croissant dans le domaine de l'apprentissage profond pour le traitement du langage naturel. Plusieurs étapes importantes ont été franchies au cours de la dernière décennie dans divers problèmes, tels que les systèmes de questions-réponses, le résumé de texte, l'analyse des sentiments, etc. Le pré-entraînement des modèles de langage dans une manière auto-supervisé est une partie importante de ces réalisations. Cette thèse explore un ensemble de méthodes auto-supervisées pour apprendre des représentations de phrases à partir d'une grande quantité de données non étiquetées. Nous introduisons également un nouveau modèle de mémoire augmentée pour apprendre des représentations basées sur une structure d'arbre. Nous évaluons et analysons ces représentations sur différentes tâches.

Dans le chapitre 1, nous introduisons les bases des réseaux neuronaux avant et des réseaux neuronaux récurrents. Le chapitre se poursuit avec la discussion de l'algorithme de rétropropagation pour former les réseaux neuronaux de flux avant, et la rétropropagation à travers l'algorithme de temps pour former les réseaux neuronaux récurrents. Nous discutons également de trois approches différentes dans le domaine de l'apprentissage de représentations, notamment l'apprentissage supervisé, l'apprentissage non supervisé et une approche relativement nouvelle appelée apprentissage auto-supervisé.

Dans le chapitre 2, nous discutons des principes fondamentaux du traitement automatique du langage naturel profond. Plus précisément, nous couvrons les représentations de mots, les représentations de phrases et la modélisation du langage. Nous nous concentrons sur l'évaluation et l'état actuel de la littérature pour ces concepts. Nous finissons le chapitre en discutant le pré-entraînement à grande échelle et le transfert de l'apprentissage dans la langue.

Dans le chapitre 3, nous étudions un ensemble de tâches auto-supervisées qui prend avantage de l'estimation contrastive bruitée afin d'apprendre des représentations de phrases à l'aide de données non étiquetées. Nous entraînons notre modèle sur un grand corpus et évaluons nos représentations de phrases apprises sur un ensemble de tâches du langage naturel en aval provenant du cadre SentEval. Notre modèle entraîné sur les tâches proposées surpasse les méthodes non-supervisées sur un sous-ensemble de tâches de SentEval.

Dans les chapitres 4, nous introduisons un modèle de mémoire augmentée appelé *Ordered Memory*, qui présente plusieurs améliorations par rapport aux réseaux de neurones récurrents augmentés par pile traditionnels. Nous introduisons un nouveau mécanisme d'attention de *Stick-breaking* inspiré par les *Ordered Neurons* (Shen et al., 2019) pour écrire et effacer la mémoire. Une nouvelle cellule récursive à portes est également introduite pour composer des représentations de bas niveau en des représentations de haut niveau. Nous montrons que ce modèle fonctionne bien sur la tâche d'inférence logique et la tâche ListOps, et il montre également de fortes propriétés de généralisation dans ces tâches. Enfin, nous évaluons notre modèle sur les tâches (binaire et multi-classe) SST (*Stanford Sentiment Treebank*) et rapportons des résultats comparables à l'état de l'art sur ces tâches.

Mots clés: réseaux de neurones, apprentissage profond, apprentissage de représentations, apprentissage non supervisé, traitement automatique du langage naturel, modélisation du langage, réseaux de neurones augmentés par la mémoire

Summary

In chapter 1, we introduce the basics of feed forward neural networks and recurrent neural networks. The chapter continues with the discussion of the backpropagation algorithm to train feed forward neural networks, and the backpropagation through time algorithm to train recurrent neural networks. We also discuss three different approaches in learning representations, namely supervised learning, unsupervised learning, and a relatively new approach called self-supervised learning.

In chapter 2, we talk about the fundamentals of deep natural language processing. Specifically, we cover word representations, sentence representations, and language modelling. We focus on the evaluation and current state of the literature for these concepts. We close the chapter by discussing large scale pre-training and transfer learning in language.

In chapter 3, we investigate a set of self-supervised tasks that take advantage of noise contrastive estimation in order to learn sentence representations using unlabeled data. We train our model on a large corpora and evaluate our learned sentence representations on a set of downstream natural language tasks from the SentEval framework. Our model trained on the proposed tasks outperforms unsupervised methods on a subset of tasks from SentEval.

In chapter 4, we introduce a memory augmented model called Ordered Memory with several improvements over traditional stack-augmented recurrent neural networks. We introduce a new Stick-breaking attention mechanism inspired by Ordered Neurons (Shen et al., 2019) to write in and erase from the memory. A new Gated Recursive Cell is also introduced to compose low level representations into higher level ones. We show that this model performs well on the logical inference task and the ListOps task, and it also shows strong generalization properties in these tasks. Finally, we evaluate our model on the SST (Stanford Sentiment Treebank) tasks (binary and fine-grained) and report results that are comparable with state-of-the-art on these tasks.

Keywords: neural networks, deep learning, representation learning, unsupervised learning, natural language processing, language modelling, memory augmented neural networks

Contents

List of Figures

List of Tables

List of Abbreviations

AE	Auto-Encoder
AMLP	Augmented Multi-Layer Perceptron
ANN	Artificial Neural Network
BN	Batch Normalization
BPTT	Backpropagation Through Time
CE	Cross Entropy
CR	Customer Reviews
DSAE	Denoising Sequential Auto-Encoder
ELU	Exponential Linear Unit
GD	Gradient Descent
I.I.D	Independent and Identically Distributed
INIT	Initialization
MLP	Multi-Layer Perceptron
MPQA	Multi-Perspective Question Answering
MR	Movie Reviews
MSE	Mean Squared Error
MSRP	Microsoft Research Paraphrase
NLI	Natural Language Inference
NLL	Negative Log-Likelihood
OM	Ordered Memory
ON	Ordered Neurons
PPL	Perplexity
QT	Quick Thought
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
SICK-R	Sentence Involving Compositional Knowledge Semantic Relatedness
SNLI	Stanford Natural Language Inference
SUBJ	Subjectivity/Objectivity
TREC	Text REtrieval Conference
VAE	Variational Auto-Encoder

Acknowledgments

I would like to thank my thesis advisor Aaron Courville for his mentorship and many helpful discussions, for giving me the chance to work on many interesting projects, and for sharing his knowledge with me.

I would like to extend my gratitude to my co-workers, Devon J Hjelm, Alessandro Sordoni, Yikang Shen, Shawn Tan and Soroush Mehri who taught me much during my time at MSR Montreal and Mila.

I would like to thank my parents and my brother for their love and support. I would not have gotten this far if wasn't for them.

I must also acknowledge great help and support from my friends and colleagues, Carolyne Pelletier, Amirhossein Shajeri, Mona Saghaee, Mandana Samiei, Mohammad Pezeshki and Reyhaneh Askari Hemmat.

Finally, the work reported in this thesis would not have been possible without the financial support from: Microsoft, NSERC, Calcul Quebec, Compute Canada, the Canada Research Chairs and CIFAR.

1 Neural Networks

1.1 Artificial Neuron

An artificial neuron is a function from a single or multiple inputs to a single output. Formally, considering a set of features for an example $x = \{x_1, x_2, x_3, \dots, x_k\}$ with k scalars, a set of k scalar weights $w = \{w_1, w_2, w_3, \dots, w_k\}$ which correspond to each feature, and finally a single scalar term b called bias, an artificial neuron $h(x)$ can be defined as:

$$h(\mathbf{x}) = g\Big(\sum_{i=1}^{k} w_i x_i + b\Big), \tag{1.1}$$

where function $g(.)$ is a nonlinear function called an *activation function*. The input to this function is also known as the *pre-activation*, where we refer to the weight $\mathbf{W}$ and bias $\mathbf{b}$ as parameters. Equation 1.1 can also be written in vector notation,

$$h(\mathbf{x}) = g\Big(\mathbf{w}^T\mathbf{x} + b\Big). \tag{1.2}$$

It is worth mentioning that in the case where we have N number of examples, $\mathbf{x}$ is a matrix of size $k \times N$, whereas in the single input case, it is a vector of size $k \times 1$.

1.1.1 Activation functions

In Equation 1.1 the pre-activation is a simple linear weighted sum. In order to make the function from input (or another layer) nonlinear, an activation function is applied on the weighted sum. There are many activation functions, we introduce five of them.

Sigmoid This activation function is an element-wise function that ranges be-

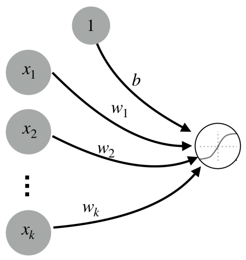


Figure 1.1 – A graphical illustration of an artificial neuron. The input is the feature vector $\mathbf{x} = \{x_1, x_2, ..., x_k\}$ to which a weight vector $\mathbf{w} = \{w_1, w_2, ..., w_k\}$ and a bias term b is assigned.

tween 0 and 1. The sigmoid function has an "S"-shaped curve. It squashes the value of the input to the range of $[0, 1]$. Its output is centered on zero. Denoting the pre-activation as z, this function is defined as follows,

$$sigmoid(z) = \frac{1}{1 + e^{-z}}. \tag{1.3}$$

Tanh This activation function is an element-wise function that ranges between -1 and 1. It squashes the value of the input to the range of $[-1, 1]$. Its output is centered on zero. Denoting the pre-activation as z, this function is defined as follows,

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}. \tag{1.4}$$

Rectifier Linear Unit (ReLU) is also an element-wise function which is quite simple. The formula is $max(0, z)$, which means that for negative inputs, the output is zero, while for positive inputs, the output is the same as the input. Despite its name, this activation function is non-linear and is one of the most common activation functions.

Exponential Linear Unit (ELU) is the same as ReLU except for the outputs of negative inputs. They are both identity functions for non-negative inputs. For negative inputs, the output of ELU slowly becomes smooth until the output is equal to $-\alpha$, whereas that is not the case for ReLU which smooths sharply. ELU has an extra constant value α in its formula which should be a positive number.

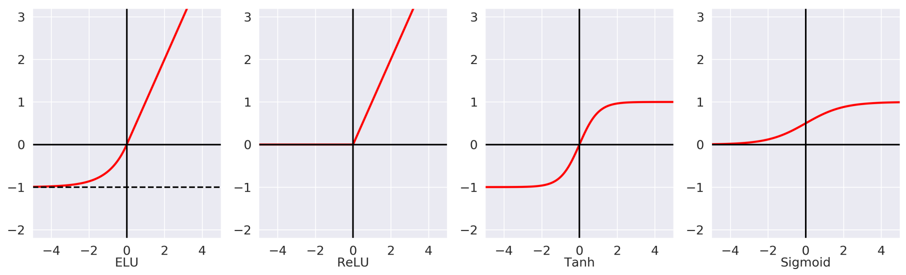


Figure 1.2 – **(a)** the Exponential Linear Unit activation function, **(b)** the Rectifier Linear Unit, **(c)** the Tanh activation function, and **(d)** the sigmoid activation function.

ELU is defined as follows:

$$g(z) = \begin{cases} \alpha(e^z - 1) & z \leq 0 \\ z & z > 0 \end{cases} \tag{1.5}$$

Empirically, optimization and training benefit from a small amount for gradient in the negative region. That is not the case for ReLU, where in the negative region the output of ReLU is always zero and therefore the gradient is zero in that region.

Softmax is a rather different and common type of activation function. It calculates the probability distribution of an event over n different outputs. It could be interpreted in a way that this function calculates the probabilities of each class over all possible classes. For instance, it would calculate the probability of a target word out of a vocabulary of words for a given input. Considering $\mathbf{z}$ as a vector of n pre-activations, this function over these n classes is defined as follows,

$$g(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{k=1}^{n} e^k} \tag{1.6}$$

A graphical illustration for the first three activation functions is shown in Figure 1.2. Note that since the Softmax activation function takes as input more that two numbers it is not shown due to visualization limitations.

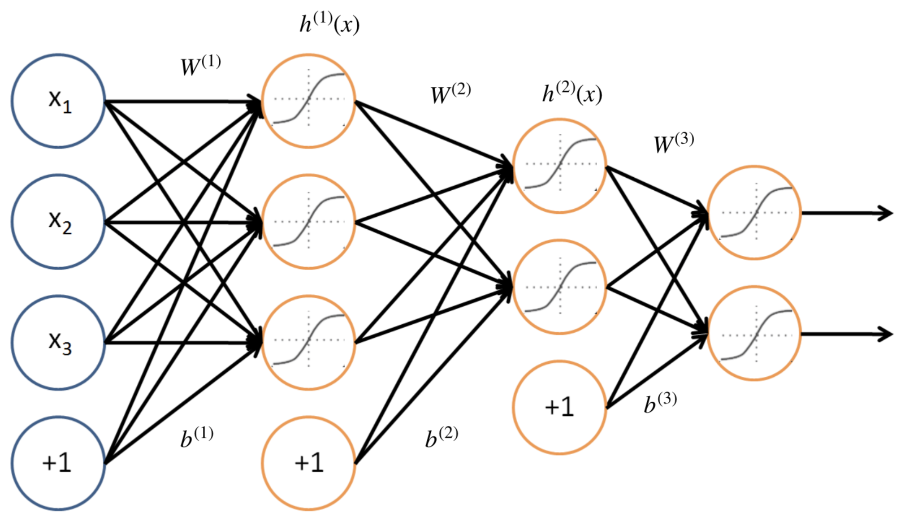


Figure 1.3 – A three layer neural network. The weight matrix $w^{(l)}$ and bias $b^{(l)}$ connects layers $(l-1)$ and (l). An activation function is applied after each weight multiplication and bias addition.

1.2 Multilayer Neural Networks

Artificial Neural Networks are structured in a layer-wise manner. In this structure, in one of the layers, neurons have different weights and biases and each one is applied on the same input. As shown in figure 1.3, once we have the output of one layer, it will act as the input to the next layer. A typical layer is defined as follows,

$$\mathbf{h}^{(l)}(\mathbf{x}) = g(\mathbf{b}^{(l)} + \mathbf{W}^{(l)}\mathbf{h}^{(l-1)}(\mathbf{x})), \tag{1.7}$$

where $\mathbf{h}^{(l-1)}$ is the nonlinear output of the l-th layer and $\mathbf{h}^{(0)} = x$. This multilayer neural network can be seen as a powerful and complicated function from inputs to outputs. This function is composed of many small functions. Multilayer neural networks are capable of approximating any measurable function to a desired accuracy given sufficient number of layers and number of neurons in each layer (Hornik et al., 1989). Finding the best parameters **W**s and **b**s remains a challenging problem which we will cover in section 1.4.

1.3 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are sequential data processing units, known as recurrent cells. They take as their input the current input example along with what they have processed in the previous time step. These models are very popular and have shown great performance in many natural language tasks (Merity et al., 2018b).

Figure 1.4 shows the unrolling of the recurrent neural network, which represents the full network written out for the whole sequence. The computation in a typical recurrent cell at time **t** can be written as

$$s_t = g(Ux_t + Ws_{t-1} + b), \tag{1.8}$$

in which function $g(.)$ is an *activation function*. Note that for a single set of inputs, $\mathbf{x}$ is a sequence therefore a matrix of size $l \times d$ while in the case of N sets of inputs, $\mathbf{x}$ is a tensor of size $N \times l \times d$.

In section 1.4.5, we will discuss how recurrent neural networks are trained using the backpropagation through time algorithm. We will also touch on a problem called vanishing gradients in RNNs introduced by Hochreiter (1998). This problem occurs during the training of RNNs dealing with long sequences, where as we go back through time to calculate the gradients, they often become smaller and smaller. In the next section we will discuss Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Unit (GRU) (Cho et al.,

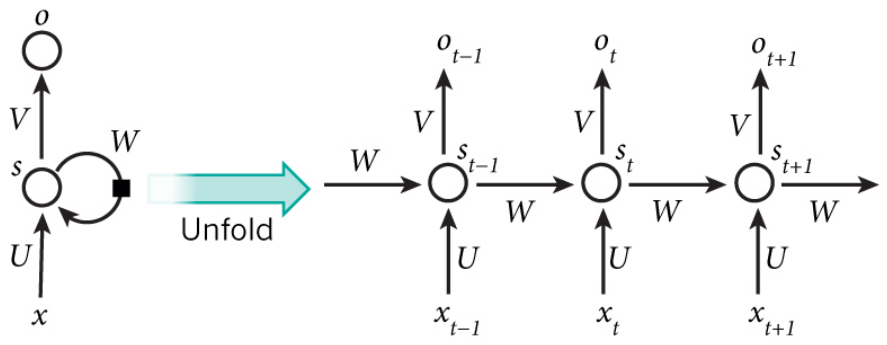


Figure 1.4 – A graphical illustration of a recurrent neural network and its unfolding in time (Figure adapted from LeCun et al. (2015))

2014), which are both effective solutions to address this problem.

1.3.1 Long Short-Term Memory

Long Short-Term Memory (LSTM) is a variation of recurrent neural nets with a more complicated cell architecture that uses a memory cell and gate units (Hochreiter and Schmidhuber, 1997). The LSTM cell computes a hidden state at timestep t as follows:

$$\begin{aligned} \mathbf{i}_t &= \sigma(\mathbf{W}_{i,x}\mathbf{x}_t + \mathbf{W}_{i,h}\mathbf{h}_{t-1} + \mathbf{b}_i) & \mathbf{f}_t &= \sigma(\mathbf{W}_{f,x}\mathbf{x}_t + \mathbf{W}_{f,h}\mathbf{h}_{t-1} + \mathbf{b}_f) \\ \mathbf{o}_t &= \sigma(\mathbf{W}_{o,x}\mathbf{x}_t + \mathbf{W}_{o,h}\mathbf{h}_{t-1} + \mathbf{b}_o) & \tilde{\boldsymbol{c}}_t &= \tanh\left(\mathbf{W}_{\tilde{c},x}\mathbf{x}_t + \mathbf{W}_{\tilde{c},h}\mathbf{h}_{t-1} + \mathbf{b}_{\tilde{c}}\right) \\ \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\boldsymbol{c}}_t & \mathbf{h}_t &= \mathbf{o}_t \odot \tanh\left(\mathbf{c}_t\right) \end{aligned} \tag{1.9}$$

Where $\mathbf{i}_t$ denotes the input gate, $\mathbf{o}_t$ denotes the output gate, $\mathbf{f}_t$ denotes the forget gate, $\tilde{\boldsymbol{c}}_t$ denotes the candidate cell state and $\boldsymbol{\sigma}$ denotes the *sigmoid* activation function. The LSTM cell has a sequential memory cell $\mathbf{c}$ that it can use to store and retrieve information controlled by the gating mechanism, namely, the input, output and forget gates. In order to have long-term information in the future, it has to flow through these gates for many timesteps. This long-term information has shown to be useful in many tasks (Merity et al., 2018b,a; Hochreiter and Schmidhuber, 1997; Peters et al., 2018). The architecture of the LSTM helps preserve the error that can be backpropagated through time (and layers) and help with the vanishing gradient problem (Hochreiter, 1998). This ability to avoid vanishing gradients allows these recurrent nets to be able to learn over many time steps and long sequences (Hochreiter and Schmidhuber, 1997; Pascanu et al., 2012). One popular variant of LSTM is the Gated Recurrent Unit (GRU) (Cho et al., 2014) which we use in our experiments in chapter 3. GRU replaces the input and forget gates with a single update gate. It also combines the cell state and the hidden state. This model is simpler than LSTM, and has been growing in popularity.

To sum up, recurrent neural networks can be seen as a complicated recurrent function that is applied on a sequence. A more complicated network called Long Short-Term Memory was introduced by (Hochreiter and Schmidhuber, 1997) to overcome the fundamental problems of traditional RNNs and attempt at solving many tasks at which traditional RNNs fall short. However, finding an appropriate set of parameters (**W**,**U**'s and **b**'s) is still a difficult problem. In section 1.4, we

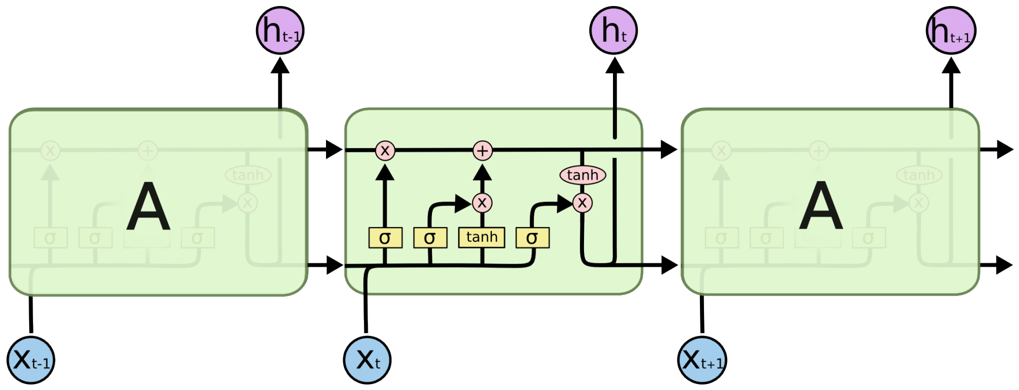


Figure 1.5 – Architecture of an LSTM cell with its gating mechanism, repeated on a sequence of three inputs. (Figure adapted from Christopher Olah's blog)

introduce the current methods for training such architectures.

1.4 Training Neural Networks

In the previous section, we introduced the structure and architectures of recurrent neural networks. As defined by the formulas, there are two sets of parameters, namely weights and biases, which need to be trained and adapted so that the neural net (or the model) acts in our desired way. This process of parameter adaptation is called *training* which we will cover in the following subsections. We will cover the methods to optimize the parameters of our model with respect to a certain loss function.

1.4.1 Loss Functions

In the optimization literature, the *loss function*, also known as the *cost function* is a function that maps a set of values or variables to a single real number, and this single real number is known as the "loss". Usually the process of training is to minimize the expectation of this loss (or maximize its negative) over the training data.

This process is also known as *Empirical Risk Minimization* in which we formulate an empirical loss function considering the outputs and parameters of the model. Let's denote the parameters of the network as θ, input to the network as $\mathbf{x}$

and the output of the network as $\hat{\mathbf{y}}$. The objective of the network will be to minimize the loss between the true (target) output $\mathbf{y}$ and the output of the network $\hat{\mathbf{y}}$. Assuming that there are N pairs of IID[1] data points $(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})$ and the output of the network with parameters θ is $\hat{\mathbf{y}}^{(n)} = f(\mathbf{x}^{(n)}; \theta)$, the cost function is defined as follows:

$$L(\mathbf{x}, \theta) = \frac{1}{N} \sum_n l(f(\mathbf{x}^{(n)}; \theta), \mathbf{y}), \tag{1.10}$$

in which, the choice of $l(.,.)$ could be different and depends on the task. It could be negative log likelihood (nll), Mean Squared Error, something more complicated such as Noise Contrastive Estimation (NCE) loss (Gutmann and Hyvärinen, 2012) or any other differentiable function.

Negative log-likelihood is usually used in a classification task with K classes, where the value of each output neuron of the network is considered to be the probability of that specific class. Formally, $f(\mathbf{x})_k = Pr(y = k|\mathbf{x})$. In this setting $l(.,.)$ is the negative log-likelihood and is as follows:

$$l(f(\mathbf{x}^{(n)}; \theta), \mathbf{y}) = -\sum_k 1_{y=k} \log f(\mathbf{x})_k = -\log f(\mathbf{x})_y, \tag{1.11}$$

where the choice of log is for numerical stability reasons and its properties help simplify the math involved. Additionally, maximizing the whole data log-likelihood is a convergent estimator of the parameters.

MSE is usually used in a regression task where the output of the model $\hat{\mathbf{y}}$ and the target y are real values. It is formulated as follows:

$$l(f(\mathbf{x}^{(n)}; \theta), \mathbf{y}) = ||f(\mathbf{x}^{(n)}; \theta) - \mathbf{y}||_F^2, \tag{1.12}$$

in which $||.||_F^2$ is the Frobenius norm, and the Frobenius norm of a vector X with n elements is defined as follows:

$$||X||_F^2 = \sqrt{\sum_{i=1}^{n} x_i^2}. \tag{1.13}$$

1. independent, identically distributed

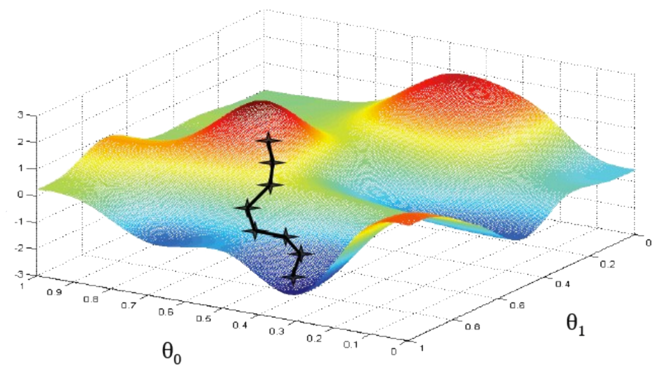


Figure 1.6 – A graphical illustration of the gradient descent algorithm, where the loss surface of a neural net with two parameters (θ_0, θ_1) is visualized. (Figure adapted from Andrew Ng's slides.)

1.4.2 Gradient Method

The most common method of optimization for neural networks is gradient based optimization, out of which, Gradient Descent and its variants are typically used. Gradient Descent finds a local minimum of a surface by taking small steps towards the direction of the gradient. This process is iterative and at each step of this process the parameters of the model are updated as follows:

$$\theta_{t+1} \leftarrow \theta_t - \eta \nabla_{\theta_t} L(\mathbf{x}, \theta_t), \tag{1.14}$$

in which η is called the *learning rate*. A graphical illustration of this algorithm is shown in figure 1.6 for a loss surface of a neural net with two parameters. The values of the two parameters are shown on two axes and on the z axis the value of the loss function is shown.

In practice, computing the gradient $\nabla_\theta L(\mathbf{x}, \theta)$ for all the data points is usually not suitable. Therefore, typically, **Stochastic (mini-batch) Gradient Descent** (SGD) is used rather than full Gradient Descent. SGD is a more practical version of Gradient Descent algorithm in which instead of computing the whole gradient $\nabla_\theta L(\mathbf{x}, \theta)$ exactly, an estimate of the gradient is calculated based on a mini-batch of randomly selected examples. Considering that these examples are randomly selected, the expected value of the minibatch gradient is the same as the exact gradient.

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

Figure 1.7 – A pseudo code for the Adam learning algorithm (stochastic optimization) (Kingma and Ba, 2015). Note that first moment and second moment are computed using two moving averages, but starting at zero. Hence, the moving averages are biased towards zero, but are corrected later.

1.4.3 Adam Learning Algorithm

The learning rate introduced in section 1.4.2 is a very important hyperparameter and plays a significant role in the training process, and one way to handle this issue is to use a learning algorithm which can change and adapt the learning rate throughout the course of training. Over the past few years many algorithms with adaptive learning rate have been introduced. In this section, we will mention *Adam* (Kingma and Ba, 2015) which is very common with recurrent neural networks and also used in most of the experiments in section 3.4.

Adam is an adaptive learning rate method, i.e., it adapts different learning rates for different parameters in the network. It uses the estimation of first and second moment of the gradient to do so. It keeps a moving average of the gradient instead of the gradient itself to take advantage of momentum. The pseudo code for this algorithm can be seen in figure 1.7.

In this algorithm there are two other hyperparameters, namely, ρ_1 and ρ_2. In practice, the value of ρ_2 is set to $1 - \frac{1}{N}$ where N is the number of mini-batches. While some problems with using Adam have been noticed in certain areas (Wilson et al., 2017), researchers still work on ways to improve it.

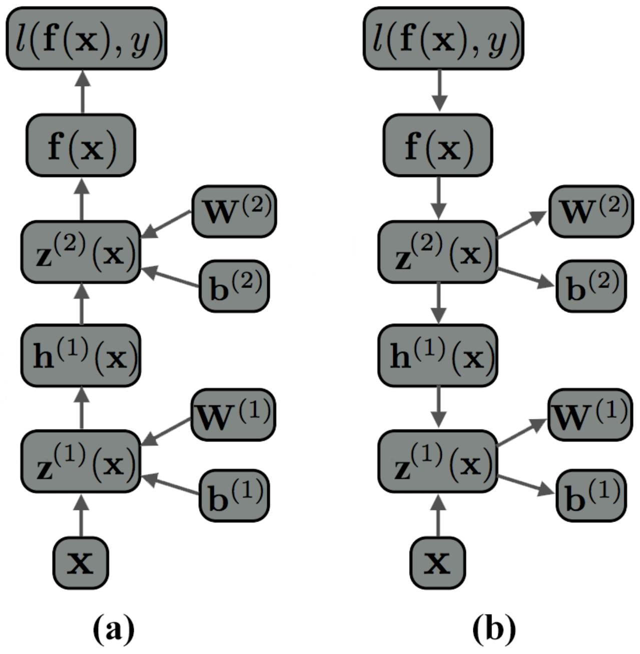

$l(\mathbf{f}(\mathbf{x}), y)$
$\mathbf{f}(\mathbf{x})$
$\mathbf{W}^{(2)}$
$\mathbf{z}^{(2)}(\mathbf{x})$
$\mathbf{b}^{(2)}$
$\mathbf{h}^{(1)}(\mathbf{x})$
$\mathbf{W}^{(1)}$
$\mathbf{z}^{(1)}(\mathbf{x})$
$\mathbf{b}^{(1)}$
$\mathbf{x}$
(a)
(b)

Figure 1.8 – (a) The forward pass and (b) the backward pass in the backpropagation algorithm on a neural network with two layers. Note that in the backward pass, since we are using the chain rule, each module should be differentiable so that we can go through them. (Figure adapted from Hugo Larochelle's slides.)

1.4.4 Backpropagation Algorithm

In the previous sections, we covered how to train a neural network parameters using the gradients of those parameters from the loss function. In this section, we introduce a well-known and efficient algorithm to calculate the gradients. Backpropagation finds the gradients in a feed forward multi-layer network iteratively with the help of the chain rule. Clearly, in order to be able to use backpropagation and chain rule, both activation functions (as seen in 1.1.1) and pre-activations should be differentiable. Back propagation has one forward and one backward pass. In the forward pass we feed the inputs to the network, which means we compute the pre-activations, activations, and finally the error (the loss function). In the backward pass, the error is propagated on activations and then the pre-activations of each layer of the network. Figure 1.8 illustrates the forward and backward passes.

Considering the network in Figure 1.8, the loss function $l(f(x), y)$ negative log-likelihood as described in sub-section 1.4.1, $\mathbf{z}^{(l)}$ and $\mathbf{h}^{(l)}$ as pre-activations and activations at layer l respectively, then we can formally write the gradient of the loss with respect to the pre-activation at layer 2 as follows:

$$\nabla_{z^{(2)}(\mathbf{x})} - \log f(\mathbf{x})_y = -(\mathbf{e}(y) - f(\mathbf{x})), \tag{1.15}$$

where $\mathbf{e}(y)$ denotes a one-hot representation, i.e. all the elements in the vector are zero except the element at index y which has a value of one. Since we want the gradients of the parameters w.r.t loss, considering that we have $\nabla_{\mathbf{W}^{(2)}} z^{(2)}(\mathbf{x}) = \mathbf{h}^{(1)}(\mathbf{x})$ and $\nabla_{\mathbf{b}^{(2)}} z^{(2)}(\mathbf{x}) = \mathbf{1}$, then we can use chain rule and derive the gradients as follows,

$$\nabla_{\mathbf{W}^{(2)}} - \log f(\mathbf{x})_y = \big(\nabla_{z^{(2)}(\mathbf{x})} - \log f(\mathbf{x})_y\big)\mathbf{h}^{(1)}(\mathbf{x})^T, \tag{1.16}$$

$$\nabla_{\mathbf{b}^{(2)}} - \log f(\mathbf{x})_y = \nabla_{z^{(2)}(\mathbf{x})} - \log f(\mathbf{x})_y. \tag{1.17}$$

Similarly, to back-propagate the gradient to the next layer of the network (the next pre-activation to be exact), we can write,

$$\nabla_{\mathbf{h}^{(1)}(\mathbf{x})} - \log f(\mathbf{x})_y = \mathbf{W}^{(2)^T}\big(\nabla_{\mathbf{z}^{(2)}(\mathbf{x})} - \log f(\mathbf{x})_y\big), \tag{1.18}$$

$$\nabla_{\mathbf{z}^{(1)}(\mathbf{x})} - \log f(\mathbf{x})_y = \big(\nabla_{\mathbf{h}^{(1)}(\mathbf{x})} - \log f(\mathbf{x})_y\big) \odot g'(\mathbf{z}^{(1)}(\mathbf{x})), \tag{1.19}$$

where $\odot$ is an element-wise multiplication (also known as Hadamard multiplication) and $g'(.)$ denotes the derivative of the activation function itself. It is clear that once we have $\nabla_{\mathbf{z}^{(1)}(\mathbf{x})} - \log f(\mathbf{x})_y$, gradients of parameters in the first layer can be computed in a similar manner.

1.4.5 Backpropagation Through Time

In the previous section we covered the backpropagation algorithm for feed forward networks. Computing the gradients for parameters of the recurrent neural nets are use a similar algorithm called backpropagation through time (BPTT). This will ultimately compute the gradients of the RNN by applying the backpropagation algorithm to the unrolled graph of the RNN (as seen in Figure 1.4). Once the gradients are obtained, one can use a gradient based method to train an RNN (e.g. The algorithm in sub-section 1.4.3). Consistent with our notation in equation 1.8, let's assume that we compute the output of the RNN as follows,

$$o_t = b_o + V s_t \tag{1.20}$$

$$\hat{y}_t = softmax(o_t) \tag{1.21}$$

Then if we assume that the loss at time step t is negative log-likelihood, we can write the gradient of the loss w.r.t the pre-activation as

$$(\nabla_{o_t} L) = \frac{\partial L^{(t)}}{\partial o^t} = \hat{y}^{(t)} - e(y^{(t)}) \tag{1.22}$$

Note that in the computation graph of the RNN, the same weights (U, W) are used in all of the time steps. So in order to find the gradients we would move backward starting from the end of the sequence,

$$\nabla_{s(\tau)} L = V^T \nabla_{o(\tau)} L \tag{1.23}$$

where τ is the time step at which the loss is defined (the last step). Now, we iterate backwards and propagate the gradients through time from this time step to $t = 1$. It is worth noting that when computing the gradients for hidden state s_t (for $t < \tau$), the gradient signals come from both the next step's hidden state $s^{(t+1)}$ and from the output at that step o^t. Therefore the gradient is calculated as follows,

$$\nabla_{s^{(t)}} L = \big(\frac{\partial s^{(t+1)}}{\partial s^{(t)}}\big)^T (\nabla_{s^{(t+1)} L}) + \big(\frac{\partial o^{(t)}}{\partial s^{(t)}}\big)^T (\nabla_{o^{(t)} L}) \tag{1.24}$$

$$= W^T \text{diag}\big(1 - \big(s^{(t+1)}\big)^2\big)(\nabla_{s^{(t+1)} L}) + V^T (\nabla_{o^{(t)} L}) \tag{1.25}$$

in which $\text{diag}\big(1 - \big(s^{(t+1)}\big)^2\big)$ is a matrix with values $1 - (s_i^{(t+1)})^2$ on its diagonal. Once the gradients of these intermediate hidden state of the unrolled RNN has been found, the gradients for the parameters can be obtained. First the gradients on biases are as follows,

$$\nabla_{b_o} L = \sum_t \big(\frac{\partial o^{(t)}}{\partial b_o}\big)^T \nabla_{o^{(t)}} L = \sum_t \nabla_{o^{(t)}} L, \tag{1.26}$$

$$\nabla_b L = \sum_t \big(\frac{\partial s^{(t)}}{\partial b^{(t)}}\big)^T \nabla_{s^{(t)}} L = \sum_t \text{diag}\big(1 - \big(s^{(t)}\big)^2\big) \nabla_{h^{(t)}} L \tag{1.27}$$

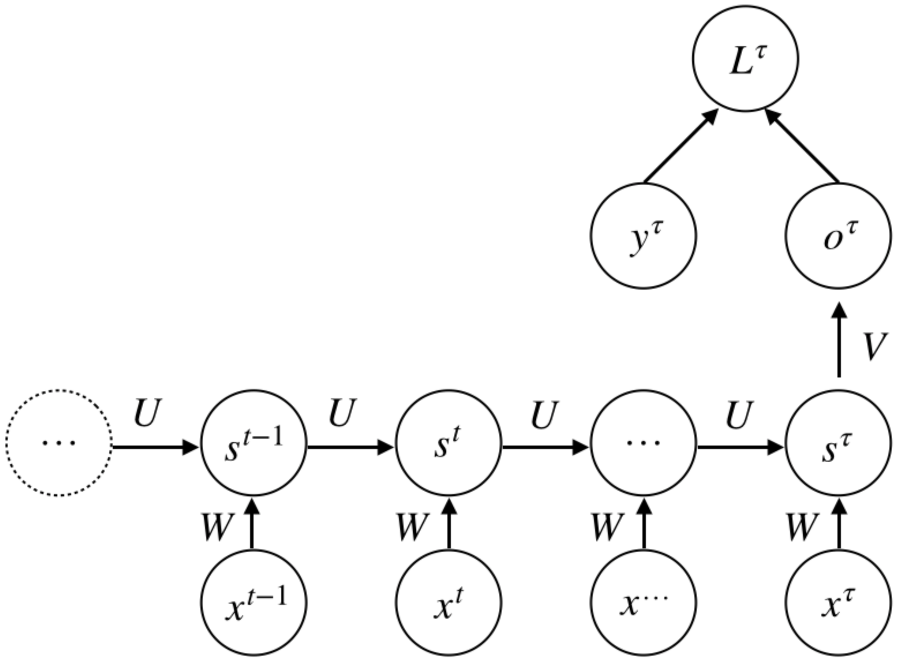


Figure 1.9 – Illustration of an RNN unfolded with an output at step τ at the end of the sequence. If there was an output at another time step (e.g. at t) then gradient on the output o^t can be computed by backpropagating from there. (Figure adapted from Deep Learning Book (Goodfellow et al., 2016).)

Note that we use the notation $\mathbf{b}^{(t)}$ as a copy of $\mathbf{b}$ but only used at time step t. Concretely, $\nabla b^{(t)}$ denotes the contribution of that parameter at time step t to the gradient. This notation is used because the parameters are shared across all the time steps. Considering this notation, the gradients for the rest of the parameters are as follows,

$$\nabla_V L = \sum_t \sum_i (\frac{\partial L}{\partial o_i^{(t)}}) \nabla_{V^{(t)}} o_i^{(t)} = \sum_t (\nabla_{o^{(t)}} L) s^{(t)^T}, \tag{1.28}$$

$$\nabla_W L = \sum_t \sum_i (\frac{\partial L}{\partial s_i^{(t)}}) \nabla_{W^{(t)}} s_i^{(t)} = \sum_t \text{diag}\big(1 - \big(s^{(t)}\big)^2\big)(\nabla_{s^{(t)}} L) s^{(t-1)^T}, \tag{1.29}$$

$$\nabla_U L = \sum_t \sum_i (\frac{\partial L}{\partial s_i^{(t)}}) \nabla_{U^{(t)}} s_i^{(t)} = \sum_t \text{diag}\big(1 - \big(s^{(t)}\big)^2\big)(\nabla_{s^{(t)}} L) x^{(t)^T} \tag{1.30}$$

It is worth mentioning that if $\mathbf{x}^{(t)}$ has parameters (e.g. trainable word embeddings) we would need to compute the gradients with respect to $\mathbf{x}^{(t)}$ as well for training.

The vanishing gradient problem happens during training of RNNs for long sequences. As it was discussed, to find the gradients of the weights in an RNN we need to go back through time (through the unrolled graph of the RNN) and apply the backpropagation algorithm. Some activation functions, like the sigmoid activation function, have this property that a large change in the input will cause a small change in the output, therefore the derivative of it is small. When the inputs of these activation functions (e.g. sigmoid function) becomes smaller or larger, the derivative becomes really small and close to zero. Since these activation functions are used in each time step in RNNs, many small derivatives are multiplied together, so the gradient decreases exponentially as we go back in time, and this causes problems for training RNNs. Based on the activation function that is being used, one could get exploding gradients instead of vanishing gradients, where the derivatives are very large and since many large derivatives are multiplied together the gradient will be too large. LSTM and GRU architectures were designed in a way to deal with these problems, and both are widely used architectures nowadays.

1.5 Representation Learning

The performance of many machine learning algorithms depends on how the data is represented. For instance, a feature representation of some data that separates different classes well in the feature space can cause ideal learning using a simple linear classifier in this representation space. In the past, it has been common for human experts to manually engineer and create the features for the data to make the machine learning algorithms work. This is both expensive and difficult, and we would ideally like to create end-to-end trainable models that automatically learn the best features for the task at hand on their own.

This representation learning can be done in different ways. In this section we will discuss 1) supervised learning, 2) unsupervised learning and 3) self-supervised learning.

1.5.1 Supervised Learning

In supervised learning, the assumption is that both the input and the desired label or target value are provided for the training procedure. This way, training would simply be finding a function that correctly maps the input to the target. Two typical categories of supervised learning problems are *classification* and *regression* problems. In classification, the target is a discrete and categorical variable, and in regression, the target is a continuous variable. Neural networks are mostly used for classification problems in supervised learning. Handwritten digit recognition, sentiment classification and object detection are instances of classification. Supervised learning approaches can also be categorised into discriminative and generative models. A discriminative model learns the conditional probability distribution $P(y|x)$, where x is the input data and the task is to predict the labels y. In classification, this amounts to finding decision boundary between classes. Generative models learn the joint probability distribution $P(x, y)$ and model the actual distribution of each class. *Logistic Regression* and *Support Vector Machines* are examples of discriminative learning approaches, and typical generative model approaches are *Naive Bayes* and *Gaussian Mixture Models*.

Despite the fact that impressive results have been attained with deep supervised learning, deep models require an extremely large amount of data that is labeled in order to perform well. Unsupervised learning, on the other hand, does not require labeled data, and therefore has gained a lot of interest.

1.5.2 Unsupervised Learning

The training data in unsupervised learning is a set of input points without any labels or target values. The idea is to find and learn a useful representation of the data without any pre-existing label or information for the model. *Clustering* is a very common unsupervised learning method in machine learning, in which the goal is to group similar examples together. This method is based on finding commonalities in the data and acting based on such commonalities when presented with a new input. In deep learning, *manifold learning*, *density estimation* and *unsupervised feature extraction* are the most common methods of unsupervised learning. Density estimation is to construct an estimate and adapt its parameters to recover the underlying probability of data P_{data} using the training examples.

Manifold learning is a dimensionality reduction method where the goal is to model a complicated distribution P_{data} with another distribution P_z in which z is a low-dimensional latent variable, representing principal factors of variation of data.

1.5.3 Self-supervised Learning

Self-supervised learning is a family of tasks where the output labels are intrinsically provided by the data itself by taking advantage of a relation between parts of the data or different views of it. For instance, learning to predict a missing word given the context sentence around it, or inversely, learning to predict surrounding words given a center word as input (also known as *skip-gram* (Mikolov et al., 2013)). This learning technique does not require manual labeling of the data; therefore massive amounts of unlabeled data is available to train models with self-supervised tasks. Unsupervised learning and self-supervised learning are closer to the way humans learn in the natural world, which appears to be with no (or very little) explicit supervision.

In this chapter, the fundamentals of feed forward neural networks and recurrent neural networks have been covered. We also discussed how to train these neural networks using the gradient method and backpropagation algorithm. Chapter 3 relies heavily on using recurrent neural networks to process sentences and the Adam learning algorithm to help train them. Further, three approaches for representation learning were discussed. In the next chapter, we will discuss some models which learn useful representations for language using self-supervision.

2 Deep Natural Language Processing

Natural Language Processing and Understanding is a very active area of research. Over the past couple of years an explosion in the amount of data available with increasingly powerful computation resources has emerged from many methods of transfer learning and new architectures which has significantly improved the state-of-the-art on a wide variety of NLP tasks (Peters et al., 2018; Radford et al., 2018; Yang et al., 2019; Kiros et al., 2015; Devlin et al., 2019).

In this chapter, we first introduce word and sentence representations, then we discuss the fundamentals of language modelling and recent related work. Finally, large scale pre-training is introduced and some of the recent work in that domain is discussed.

2.1 Word Representations

Neural Networks have been very successful in many natural language tasks, such as machine translation (Vaswani et al., 2017), question answering (Yang et al., 2019; Peters et al., 2018) and natural language inference (Liu et al., 2019; Yang et al., 2019). In order to apply these neural networks on text, words are represented in a vector space. Each word is represented by a vector that represents a point in an N-dimensional space. Early approaches used very sparse vector representations of the size of the vocabulary with a single 1 at the index of the word in the vocabulary. This one-hot representation does not capture any similarity between two words. So when the model sees the word ”cat” in a context during training, it cannot use the learned information when seeing the word ”kitten” at test time in the same context. Instead, many methods were introduced to represent a word based on the context in which it usually appears (Mikolov et al., 2013; Pennington et al., 2014). These methods learn an embedding matrix where its columns are the learned word vectors that capture distributional semantics and co-occurrence statistics. There is another method where the embedding matrix is randomly initialized and trained as a part of the model. If the tasks has enough training data this method would generally work well.

2.2 Sentence Representations

With the success of word representations, there have been some attempts to encode a sentence into a fixed-length vector (Kiros et al., 2015; Hill et al., 2016; Logeswaran and Lee, 2018). However, some researchers critically disagree with the idea of a fixed-length sentence vector as Raymond J. Mooney said "You can't cram the meaning of a whole %&!$ing sentence into a single $&!*ing vector!". Despite these critics, sentence embeddings have achieved great results in different tasks, such as machine translation (Cho et al., 2014; Sutskever et al., 2014) and natural language inference (Williams et al., 2018). Kiros et al. (2015); Conneau et al. (2017); Logeswaran and Lee (2018) have proposed universal sentence embeddings which are trained on a large amount of text and then used in many different applications.

Table 2.1 – Subset of SentEval benchmarks along with the type of the task, labels, and size of the train and test split. 2 and 6 are number of classes in tasks, and [0,5] means that labels are scores between 0 and 5.

Task	Type	Output	Train size	Test size
MR (Pang and Lee, 2005)	movie review	2	11k	11k
CR (Hu and Liu, 2004)	product review	2	4k	4k
SUBJ (Pang and Lee, 2004)	subjectivity classification	2	10k	10k
MPQA (Wiebe et al., 2005)	opinion polarity	2	11k	11k
TREC (Voorhees and Buckland, 2012)	question type classification	6	6k	0.5k
MSRP (Dolan et al., 2004)	paraphrase detection	2	4.1k	1.7k
SICK-R (Marelli et al., 2014)	semantic textual similarity	[0,5]	4.5k	4.9k

2.2.1 Evaluation

The quality of fixed-length sentence embeddings are typically evaluated using a suite of natural language processing tasks called SentEval (Conneau and Kiela, 2018). This toolkit uses sentence embeddings as features on a set of natural language inference and sentence similarity tasks, which includes binary and multi-class classification. These tasks require understanding of sentence semantics to be solved. Table 2.1 shows a subset of SentEval tasks which will be used to evaluate our sentence embeddings in the next chapter. Here is a summary of the tasks in this subset along with examples:

— **Movie Reviews (MR)** is a binary sentiment analysis of movie reviews. Example: *"there are no special effects , and no hollywood endings."* – Output: positive.

— **Customer Reviews (CR)** is a binary sentiment analysis of customer product's reviews. Example: *"while light , it will not easily go in small handbags or pockets."* – Output: negative.

— **Subjectivity/Objectivity (SUBJ)** is a binary classification task where the goal is to classify the sentences as subjective or objective. Example: *"the story bogs down in a mess of purposeless violence."* – Output: subjective.

— **Multi-Perspective Question Answering (MPQA)** is a binary classification of evaluation polarity. Example: *"did not adequately"* – Output: negative.

— **Text REtrieval Conference (TREC)** is a multi-class classification of question types. Example: *"When did the American Civil War end?"* – Output: NUM:date.

— **Microsoft Research Paraphrase (MSRP)** is a binary classification task where the goal is to classify whether a pair of sentences are paraphrases of each other or not. Example: *"British police claim they have been taking statements from more than 30 of the women."*, *"British military police have also been taking statements from 30 of the women who have made the allegations."* – Output: paraphrase.

— **Sentence Involving Compositional Knowledge Semantic Relatedness (SICK-R)** is a task where the goal is to measure the relatedness between sentences. The output is from 0 (not related) to 5 (related). Example: *"Two people are kissing near a crowd"*, *"A lady is being kissed by a* *man"* – Output: 4.1.

Moreover, learning the compositional property of language is also an important attribute that sentence representation learning methods should have (Dasgupta et al., 2018). In the context of language, a model with compositional property is able to find the meaning of a complex phrase from the meaning of its constituent phrases. Learning compositionality is discussed in more details in section 4.1. Dasgupta et al. (2018) present a new dataset to evaluate compositionality in sentence representations with a natural language inference task, where the task is to classify a pair of sentences into *entailment*, *neutral* or *contradiction* classes. They create the examples in three different categories, and the relation (entailment, neutral or contradiction) between pairs of sentences in each example could be changed only by changing the order of the words in each sentence.

— **Same category** in which the difference of the pairs is only in the order of the words.

  Entailment:

  s1: the billionaire is more weary than the writer.

  s2: the billionaire is more weary than the writer.

  Contradiction:

  s1: the billionaire is more weary than the writer.

  s2: the writer is more weary than the billionaire.

— **More-Less category** in which the difference of the pairs is in whether they have the word "more" or the word "less".

  Contradiction:

  s1: the woman is less lazy than the fashion blogger.

  s2: the woman is more lazy than the fashion blogger.

  Entailment:

  s1: the woman is less lazy than the fashion blogger.

  s2: the fashion blogger is more lazy than the woman.

— **Not category** in which the difference of the pairs is in having the word "not".

  Entailment:

  s1: the woman is not more cheerful than the nurse.

  s2: the nurse is more cheerful than the woman.

  Contradiction:

s1: the woman is not more cheerful than the nurse.

s2: the woman is more cheerful than the nurse.

We use this dataset to evaluate the sentence representations from our proposed model in chapter 4. The results are shown in section A.1.

2.2.2 Related Work

In the section, recent work in learning sentence representations is discussed. These works can be categorized into two groups, 1) unlabeled data approaches and 2) labeled data approaches.

Unlabeled Data Approaches

Kiros et al. (2015) introduced the skip-thought objective for learning sentence representations by predicting the adjacent sentences word by word. In their model an RNN encoder is used to encode a source sentence and the output of this RNN is used in a decoder RNN to predict the words of the context sentences.

Gan et al. (2016) proposed a similar encoder-decoder based model to learn generic sentence representations using CNNs. In their method, a convolution neural network is used as an encoder to map the source sentence into a fixed length vector, which is fed into a decoder LSTM for reconstruction and predicting adjacent sentences. In addition, they propose a hierarchical model to encode and predict multiple following sentences.

Le and Mikolov (2014b) proposed another method called Paragraph Vector for fixed length embeddings from sentences, paragraphs or documents by predicting words in the documents.

Hill et al. (2016) introduce the sequential denoising autoencoder approach on variable length sentences. In their method, they introduce noise into a sentence by removing some words or switching bigrams with some probability. Their LSTM based encoder-decoder model is trained with the denoising objective to predict the original uncorrupted sentence.

Results from (Kiros et al., 2015) and (Logeswaran and Lee, 2018) suggest that rich sentence semantics can be obtained from neighbouring sentences. One drawback of these is that they are slow to train. Hill et al. (2016) propose the FastSent approach which predicts the adjacent sentences based on a bag of words (BOW) representation of a source sentence.

Labeled Data Approaches

The following approaches are on structured data rather than raw text. These methods are usually trained for a specific task. Hill et al. (2016) propose a method to map dictionary definitions of words to the embeddings of the corresponding defined words.

Hermann and Blunsom (2014) propose a multilingual framework, in which their model is trained to assign close embeddings (via inner product minimization between embeddings) to paired sentences between the two languages and far embeddings for sentences that are not aligned.

In a more recent work, Conneau et al. (2017) show that their sentence representations from a model trained only on natural language inference corpora outperform the previous methods on a set of unseen tasks referred to as ”transfer tasks”.

Subramanian et al. (2018) propose a multi-task learning setup in which they train a model with multiple training objectives, such as skip-thought, neural machine translation, natural language inference and constituency parsing to combine the benefits of these tasks into a single model. Hashimoto et al. (2017) also proposed a multi-task framework and a strategy to train the model at different depth; however, their goal is not re-usable generic sentence representations that transfer.

2.3 Language Models

Language Modelling is closely related to many natural language processing tasks. At the core, the goal of language modelling is to estimate the probability of a sequence of words (or tokens) $p(x_1, \ldots, x_T)$. Having this estimator can help tasks like machine translation (Koehn et al., 2003) or speech recognition (Bahl et al., 1990) in choosing the most probable sentence given many possible sentences. Formally, the factorization for language modelling is as follows,

$$p(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t | x_1, \ldots, x_{t-1}) \tag{2.1}$$

It is also worth mentioning that language models are evaluated by *perplexity* measurement,

$$ppl = 2^{\frac{1}{T}\sum_{t=1}^{T} -\log_2 p(x_t|x_1,\ldots,x_{t-1})} \tag{2.2}$$

This is a measurement that shows how well a probability model predicts the data. A lower value shows a model is better at predicting the data. In practice, since a cross-entropy loss is usually used for these models perplexity can be an exponentiated cross-entropy.

Traditionally, this was done using *n-gram* models. This model predicts the probability of the next word based on a context window of size $n-1$:

$$p(x_1,\ldots,x_T) = \prod_{t=1}^{T} p(x_t|x_{t-1-n},\ldots,x_{t-1}) \tag{2.3}$$

where $p(x_t|x_{t-1-n},\ldots,x_{t-1})$ is obtained by counting the occurrences of x_t after the window and then normalized as follows,

$$p(x_1,\ldots,x_T) = \frac{C(x_{t-1-n},\ldots,x_{t-1},x_t)}{C(x_{t-1-n},\ldots,x_{t-1})} \tag{2.4}$$

But the main problem with this approach is that as the number of words in the window grows, the count matrix will grow exponentially $O(|V|^n)$, where V is the set of all the words in the dataset.

Bengio et al. (2003) introduced a neural language model that leverages *word embeddings* and neural networks. These embeddings are vectors that act as representations for words. Words that appear in similar context will have similar embedding vectors (Mikolov et al., 2013). The neural language model uses these semantic and syntactic closeness information in order to predict. But this model lacks memory or history of what has happened before, so recurrent neural networks language models (RNN LM) were introduced to capture long-term dependencies in sequences (Mikolov et al., 2010) and make predictions based on what has happened until now.

Merity et al. (2018b) investigates the methods and strategies for regularizing and optimizing LSTM word-level language models. They show that using hidden to hidden dropout as recurrent regularization helps improve the model's performance.

They also introduce a variant of SGD algorithm that help reduce perplexities on the Penn Treebank (Mikolov et al., 2010) and WikiText-2 datasets.

Dai et al. (2019) introduce a Transformer model in the setting of language modelling that can learn dependencies beyond a fixed length vanilla Transformers.

2.4 Large Scale Pre-training and Transfer Learning

The success of many natural language processing methods depends on their ability to extract representations for content and meaning of each input sentence. The representations and the encoding components involved are typically trained directly for the target task in mind. This approach could be effective on some tasks with a lot of data (Rajpurkar et al., 2018; Hassan et al., 2018), but this is only plausible for a few NLP tasks with more than millions of training examples. This has spiked interest in a technique called transfer learning, in which knowledge from a source setting is extracted and reused in a target setting. Specifically, in this technique, a model is first trained (pre-trained) in one domain with a lot data, then it is adapted to a different domain, which typically has less data than the source domain. Given the success of word embeddings and image encoders

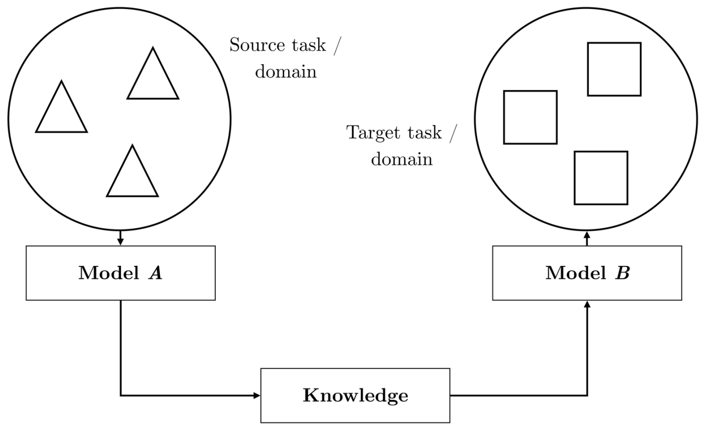


Figure 2.1 – A depiction of the transfer learning procedure (Figure adapted from Sebastian Ruder's blog.)

(Zamir et al., 2018), it is plausible to believe this could be beneficial. It is shown in the literature that language model pre-training is shown to be very effective at capturing a variety of linguistic phenomena from a large corpus (Peters et al., 2018; Devlin et al., 2019; Peters et al., 2018). This knowledge is then used (transferred) to initialize and then train a model to perform well on a natural language processing task. One property that makes these self-supervised methods attractive is that since it does not require labeled data, vast amounts of available unlabeled data can be used to pre-train models.

These methods extend the classic supervised machine learning paradigm to take advantage of data from multiple tasks or domains to train a model with better generalization capabilities (Peters et al., 2018; Devlin et al., 2019; Peters et al., 2018). Specifically, these methods in NLP combine a lot of data from the internet with self-supervised methods as pre-training steps before fine-tuning for a downstream task of interest.

Peters et al. (2018) introduced ELMo as a deep contextualized word representation model. The word vectors are learned from a combination of the hidden states of a deep bidirectional language model pre-trained on 1 Billion Word Benchmark (Chelba et al., 2014) data. The bi-directional language model is a combination of a backward and forward language model.

Radford et al. (2018) pre-trains a uni-directional Transformer (Vaswani et al., 2017) model with a language modelling criteria. Their model is later fine-tuned on several supervised tasks. They report very good results on some of the tasks they tested their model on.

Devlin et al. (2019) Take a similar approach by pre-training a Transformer model in order to get a language representation model called BERT (Bidirectional Transformers for Language Understanding). Their model is pre-trained on unlabeled text to get representations by conditioning on both left and right context. They pre-train BERT using two unsupervised tasks, namely, Masked Language Modelling and Next Sentence Prediction (NSP). Specifically, in Masked LM they simply mask out some percentage of the input tokens at random, and then predict those masked tokens using the bidirectional representations. NSP is designed to capture the relationship between two sentences. It is a binary task where the goal is to predict whether sentence B is the actual sentence that follows sentence A. They report exceedingly good results on several natural language tasks when they fine-tune their model on those tasks.

Radford et al. (2018) train a very large Transformer model, called GPT-2, with 1.5B parameters that achieves state-of-the-art results on several language modelling datasets when evaluated in a zero-shot setting. They also show that their model is capable of generating very coherent texts. They demonstrate that a model trained with a LM objective can perform well in zero-shot downstream tasks without fine-tuning the parameters.

Yang et al. (2019) introduced a generalized pre-training method. While BERT's masked language modelling pre-training task is a denoising autoencoding task, XL-Net, their proposed model, is pre-trained with an autoregressive permutation language modelling method on different permutations of a factorization order (e.g. $x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow \cdots \rightarrow x_T$). They achieve impressive results on many natural language processing tasks such as question answering, sentiment analysis and natural language inference.

2.5 Conclusion

In this chapter, fundamentals of deep natural language processing, such as word embeddings and sentence representations were introduced. We also discussed transfer learning and the benefits of self supervision in pre-training a model. Based on the studies in the previous section, the process of training a model with a self supervised method (e.g. language modelling) seems to learn some structures in the language as a byproduct, which are very helpful when it comes to prediction on downstream supervised tasks. This motivates our work discussed in the next chapter, in which we leverage a set of self-supervised tasks to learn sentence representations from unlabeled data that perform well in many different applications.

3 Learning Sentence Representation with Self-supervision

3.1 Introduction

Finding better and efficient sentence representations is still an active area of research in deep natural language processing and much remains to be done. In recent years, several methods have been proposed for learning sentence representations (Kiros et al., 2015; Logeswaran and Lee, 2018; Subramanian et al., 2018). There are two major groups for these methods. The first group are methods that provide sentence embeddings trained in an unsupervised (or self-supervised) manner, such as SkipThought (Kiros et al., 2015), ParagraphVector (Le and Mikolov, 2014a), Quick Thought (Logeswaran and Lee, 2018), etc.

In the other group, models are trained on a specific task or on a set of tasks using supervised training (Subramanian et al., 2018; McCann et al., 2017). The appeal of the first group is that one can use them in a semi-supervised setting, while taking advantage of large unlabeled data.

In this work we propose and investigate a set of self-supervised tasks to learn sentence representations in an unsupervised setting by maximizing the mutual information between different views. This self-supervision framework allows us to train our model on a large corpora (over 74 million sentences).

The mutual information (MI) maximization objective has been shown to result in effective and flexible unsupervised representations (Hjelm et al., 2019; Bachman et al., 2019). We use MI to explore a set of pre-training tasks designed to capture information and structure from a large unlabeled corpora. We evaluate our sentence representations on SentEval, a suite of natural language processing tasks (Conneau and Kiela, 2018).

Section 3.2 details our proposed self-prediction tasks. We also provide an ablation study on different tasks and evaluate our sentence representations on SentEval. We report results and analysis in section 3.4.

3.2 Approach

In this section, we describe the proposed pre-training tasks. Our objectives mainly work in a discriminative manner where the model attempts to find a target embedding from a set of candidates. We explored the following approaches:

1. Quick Thoughts (Logeswaran and Lee, 2018)
2. Self Prediction
3. Global to Local Prediction

3.2.1 Noise-contrastive estimation

We train our proposed tasks using noise-contrastive estimation (Gutmann and Hyvärinen, 2012). NCE is based on formulating a density estimation to a probabilistic binary classification. This reduces down to training a logistic regression classifier to discriminate between examples from the data distribution (positive samples) and examples from the noise distribution (negative samples). What makes NCE appealing is that it allows fitting an unnormalized model which makes computation time independent of the number of possible output values, e.g. vocabulary size when predicting the next word, or the number of possible sentences, here.

Suppose we want to learn the distribution of sentences for some specific context sentence c, denoted by $P^c(s)$, where s denotes a sentence. The training data would be tuples of the context and neighbour sentence $[(c_1, s_1), (c_2, s_2), \cdots (c_n, s_n)]$. The probabilistic binary classification problem has its positive samples from the training data, and its negative samples from a noise distribution $P_{neg}(s)$. There is freedom in choosing the noise distribution. Let's assume that the distribution of all the different unigram sentences in our dataset $[s_4, c_{10}, s_6, c \ldots]$ is our noise distribution. Let's also assume that $P^c(\text{pos} = 1|s)$ is the probability that a given sentence comes from the data (i.e. is in the context of c). We use our model, with parameters θ, to estimate this probability as follows,

$$P^c(\text{pos} = 1|s, \theta) = \frac{P^c_\theta(s)}{P^c_\theta(s) + P_{neg}(s)} \tag{3.1}$$

We can fit the model by maximizing the average log of the probability above over the training data and noise samples, and we optimize:

$$L = -\mathbb{E}_D \Big[\log P^c(\text{pos} = 1 | s, \theta) \Big] \tag{3.2}$$

Where D is our training data. Minimizing the loss in 3.2 maximizes a lower bound on mutual information. For more details regarding the relationship between mutual information and NCE see van den Oord et al. (2018) and Poole et al. (2019).

3.2.2 Quick Thoughts

The goal of this task is to capture information at sentence level by using the representation of one sentence to predict the representation of its neighbouring sentences. One benefit of doing this prediction in the representation space (rather than in raw data space, e.g. generating the neighbouring sentences) is that the model is free to focus on the important aspects of the sentence and ignore the irrelevant parts of it. This task is to discriminate between the sentences that appear in the context of a specific sentence and those that do not. In this task, given a pair of consecutive sentences $(S_{\text{src}}, S_{\text{trg}})$ in our dataset D, the objective is to discriminate between S_{trg} that follows S_{src} and the sentences that don't follow S_{src} (aka negative samples). Both sentences are encoded separately using two bi-directional GRUs (Cho et al., 2014), Enc_{src} and Enc_{trg}, the source encoder and target encoder respectively. The source sentence S_{src} is encoded using Enc_{src} and the target sentence S_{trg} is encoded using Enc_{trg}. The sentences are fed as inputs to the GRUs and the final hidden states of the forward and backward GRUs are concatenated and interpreted as the representation for sentences. Let S_{neg} be a set of sentences that do not appear in the context of the source sentence S_{src}, as negative samples, we can write the NCE loss,

$$L_{\text{QT}} = -\mathbb{E}_D \Big[\log \frac{\exp(\text{sim}(h_{\text{src}}, h_{\text{trg}}))}{\sum\limits_{c \in \{S_{\text{trg}}\} \cup S_{\text{neg}}} \exp(\text{sim}(h_{\text{src}}, h_c))} \Big] \tag{3.3}$$

Where h_{src}, h_{trg} and h_c are source, target and candidate sentence representations respectively. $\{S_{\text{trg}}, S_{\text{neg}}\}$ denotes a set containing the target sentence and all the negative examples. sim is also a similarity function for which we use inner product $sim(i, j) = i^T j$.

s : His aching eyes shot open , and a startled yelp passed his lips . More hands laid ahold of him and dragged him out of the sea and aboard the overturned boat , laying him out like a trophy kill .

$\hat{s}_{src}$: [MASK] aching eyes , [MASK] [MASK] and [MASK] startled glad Malma passed lips More [MASK] laid [MASK] hands [MASK] and him him dragged [MASK] [MASK] and [MASK] the [MASK] Minyera [MASK] , [MASK] laying [MASK] disorganised [MASK] [MASK] [MASK] [MASK] .

$\hat{s}_{trg}$: His eyes aching Sympathy [MASK] shot [MASK] a [MASK] yelp [MASK] [MASK] lips . [MASK] [MASK] [MASK] [MASK] him ahold dragged and him out [MASK] the [MASK] [MASK] aboard the [MASK] [MASK] [MASK] , him like [MASK] [MASK] [MASK] kill [MASK]

Figure 3.1 – An example of the corruption applied on a sentence to obtain $\hat{s}_{\text{src}}$ and $\hat{s}_{\text{trg}}$.

3.2.3 Self Prediction

This task is designed to make the model robust to noise and small jitters in the sentence sequence. Specifically, given a sentence $\mathbf{s}$, we corrupt it twice differently to obtain two different corrupted versions of sentence $\mathbf{s}$, denoted by $\hat{s}_{\text{src}}$ and $\hat{s}_{\text{trg}}$ (figure 3.1). the task consists of discriminating $\hat{s}_{\text{trg}}$ from other sentences (negative samples) given $\hat{s}_{\text{src}}$. The corruption procedure is done in two steps. The first step is done similar to (Devlin et al., 2019) by replacing some percentage of the words in the sentence with either a random word, the [MASK] token or the original word. Specifically, we choose 50% of the words in a sentence, and replace 80% of them with [MASK] token, replace 10% of them with a random word from the vocabulary and we keep the other 10% unchanged. The second step consists of shuffling a word in a sentence with another word chosen from the set of the same word and its next k words with a uniform probability. In our experiments we set $k = 3$. Figure 3.1 shows an example of the result of this corruption.

Once $\hat{s}_{\text{src}}$ is encoded using Enc_{src} and $\hat{s}_{\text{trg}}$ is encoded using Enc_{trg}, a similar NCE loss can be written,

$$L_{\text{SP}} = -\mathbb{E}_D \Big[\log \frac{\exp(\text{sim}(\hat{h}_{\text{src}}, \hat{h}_{\text{trg}}))}{\sum\limits_{c \in \{\hat{s}_{\text{trg}}\} \cup S_{\text{neg}}} \exp(\text{sim}(\hat{h}_{\text{src}}, h_c))} \Big] \tag{3.4}$$

Where S_{neg} denotes a set of negative examples, for which we sample from the corrupted version of all the sentences in our dataset excluding $\mathbf{s}$. $\hat{h}_{\text{src}}$ and $\hat{h}_{\text{trg}}$ are representations for corrupted sentences $\hat{s}_{\text{src}}$ and $\hat{s}_{\text{trg}}$ obtained from Enc_{src} and Enc_{trg} respectively. The similarity function is same as in section 3.2.2 the inner

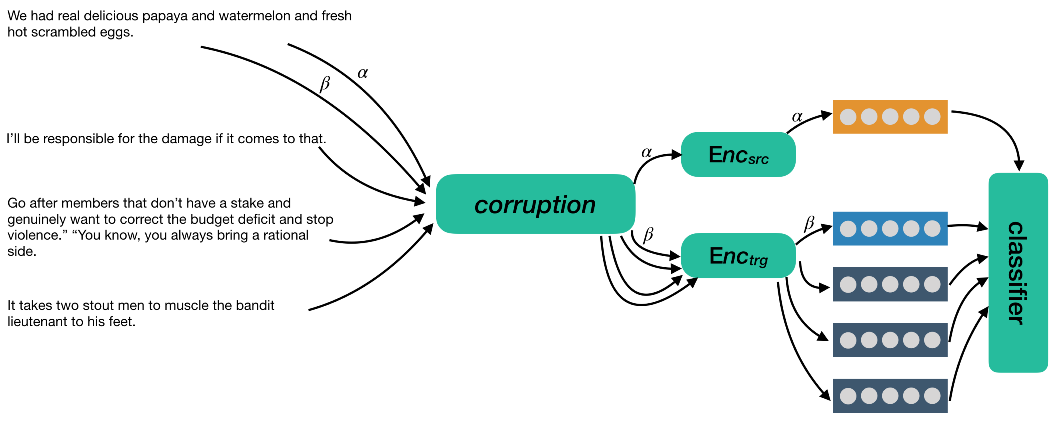

We had real delicious papaya and watermelon and fresh hot scrambled eggs.
I'll be responsible for the damage if it comes to that.
Go after members that don't have a stake and genuinely want to correct the budget deficit and stop violence." "You know, you always bring a rational side.
It takes two stout men to muscle the bandit lieutenant to his feet.
corruption
α
β
Enc_src
Enc_trg
classifier

Figure 3.2 – Overview of the Self Prediction task in which a sentence is corrupted in two different ways (two corruptions are denoted by α and β). Once the α and β corrupted versions of the sentence are obtained, one of them will be encoded using the source encoder and the other using the target encoder. The classifier should discriminate the correct representation, which is the representation of the β-corrupted sentence from the target encoder, based on the representation of the α-corrupted sentence from the source encoder. Other sentences in the batch are also corrupted and encoded. They will be used as negative examples.

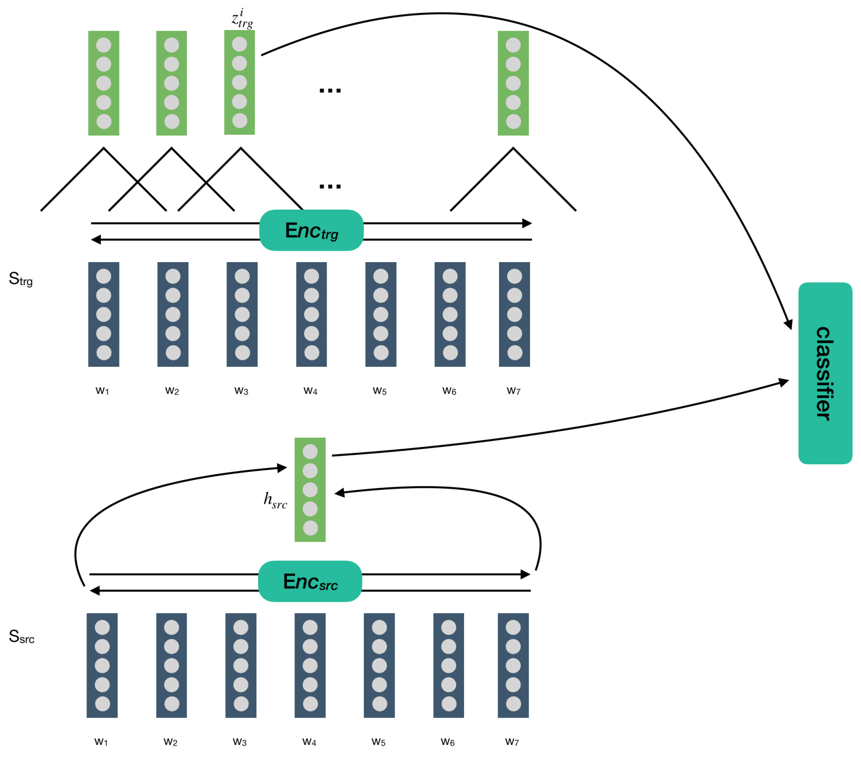
product. Figure 3.2 illustrates an overview of this task.

3.2.4 Global to Local

The intuition behind this task is to bridge the gap between prediction at sentence level and word level. The Quick Thought and Self Prediction tasks predict at sentence level and the masked language modelling pre-training task from Devlin et al. (2019) predicts at word level. This task is designed to do predictions at a sub sentence or phrase level. Given a pair of consecutive sentences $(S_{\text{src}}, S_{\text{trg}})$ in our dataset D, similar to section 3.2.2 both sequences are fed into two bi-directional GRUs, Enc_{src} and Enc_{trg}. Let h^i_{trg} denote the concatenation of the intermediate hidden representation of the forward and backward target GRUs at step i. Then, the sequence of all hidden states $[h^1_{\text{trg}}, h^2_{\text{trg}}, \dots, h^T_{\text{trg}}]$ is fed into a two layer CNN (Lecun et al., 1998) block with a kernel width of 3 and stride of 1. We will denote the output as $[z^1_{\text{trg}}, z^2_{\text{trg}}, \dots, z^T_{\text{trg}}]$. S_{src} is encoded using Enc_{src} similar to section 3.2.2 to obtain h_{src}. Given h_{src}, the task is to discriminate between the intermediate representations of S_{trg}, denoted by z^i_{trg}, and the intermediate representations of sentences that don't follow S_{src}. Note that in this task, negative examples are also intermediate representations, i.e. outputs of the two layer CNN block, from sentences that don't follow S_{src}. Given this notation the NCE loss is as follows,

$$L_{\text{G2L}} = -\mathbb{E}_D \Big[\sum_{i=1}^{T} \log \frac{\exp(\text{sim}(h_{\text{src}}, z^i_{\text{trg}}))}{\sum_{z_c \in \{z^i_{\text{trg}}\} \cup S_{\text{neg}}} \exp(\text{sim}(h_{\text{src}}, z_c))} \Big] \tag{3.5}$$

Where S_{neg} is the set of negative examples, which are sampled from the distribution of intermediate representations (obtained in the same way as z^i_{trg}s) of all the sentences in our dataset excluding S_{trg}. Figure 3.3 shows an overview of this task. An L2-norm of both representations h_{src} and h_{trg} is added to the loss for regularization. At test time, to encode a sentence s we use the concatenation of the output representation from both source and target encoders $[\text{Enc}_{\text{src}}(s), \text{Enc}_{\text{trg}}(s)]$ as (Logeswaran and Lee, 2018) did in their experiments.



z^i_{trg}
...
...
Enc_{trg}
S_{trg}
w_1 w_2 w_3 w_4 w_5 w_6 w_7
classifier
h_{src}
Enc_{src}
S_{src}
w_1 w_2 w_3 w_4 w_5 w_6 w_7

Figure 3.3 – Overview of the Global to Local task. The source sentence is encoded using the source encoder, whereas the target sentence is first fed through the target encoder and then the two layer CNN network to get z_{trg} of all positions.

3.3 Experimental Setup

3.3.1 Data

Our experimental setup is very similar to Logeswaran and Lee (2018). We trained our models on the BookCorpus (Kiros et al., 2015) dataset. It consists of 74 million sentences. We use the case-sensitive vocabulary of size 100k for this dataset from Logeswaran and Lee (2018).

3.3.2 Training

We conducted our experiments with a batch size of 400. Each example in a batch is a pair of sentences from the corpus. Given that there is no order between examples in the batch, we use all the sentences in the batch (instead of sampling from the whole dataset) to form the set of negative examples S_{neg} for each task as described in the section 3.2. Logeswaran and Lee (2018) states that this simple random strategy for choosing negative samples performs as well as other more complicated strategies. We train the models for 10 epochs with the Adam optimizer and a learning rate of $5e-4$. GRU weights are initialized using uniform Xavier initialization (Glorot and Bengio, 2010) and word embeddings are initialized from GloVe embeddings (Pennington et al., 2014). For the experiments with more than one pre-training task we sample two tasks for each batch at the same time and compute the loss for those tasks. We also use a coefficient of $1e-5$ for the L2 regularization.

3.3.3 Evaluation

We use the SentEval toolkit to evaluate our sentence embeddings as features. Following (Logeswaran and Lee, 2018) we evaluate our embeddings on the tasks listed in table 2.1.

There is no *dev* split for the binary tasks MR, CR, SUBJ and MPQA. A 10-fold cross validation is carried out to report the test scores for these tasks. The rest of the tasks each have a *dev* split which are used to choose the best regularization hyperparameter. We use the evaluation code and strategies from (Logeswaran and Lee, 2018) and (Kiros et al., 2015). In this strategy, a softmax or logistic classifier is simply trained on top of features/embeddings from the model. These embeddings are kept fixed and only the classification layer is trained.

3.4 Results & Discussion

Table 3.1 – Scores of different unsupervised models on downstream tasks from SentEval. GloVe bag-of-words model (Logeswaran and Lee, 2018), Denoising auto-encoders and FastSent (Hill et al., 2016) and MC-QT (Logeswaran and Lee, 2018) are baselines. Scores were obtained from Logeswaran and Lee (2018). Higher numbers are better for all the columns. Last column shows the average of scores on all of the downstream tasks. MC-QT is the MultiChannel-QT model from Logeswaran and Lee (2018).

Model	**MR**	**CR**	**SUBJ**	**MPQA**	**TREC**	**MSRP**		**SICK**		**Avg**
						Acc	F1	r	ρ	
GloVe BoW	78.1	80.4	91.9	87.8	85.2	72.5	81.1	0.764	0.687	80.2
SDAE	67.6	74.0	89.3	81.3	77.6	76.4	83.4	N/A	N/A	N/A
FastSent	71.8	78.4	88.7	81.5	76.8	72.2	80.3	N/A	N/A	N/A
MC-QT	80.4	85.2	93.9	89.4	**92.8**	**76.9**	**84.0**	**0.868**	**0.801**	**85.5**
QT+SP+G2L	79.97	84.7	**94.58**	89.47	92.0	74.67	82.02	0.867	0.798	84.87
QT+SP-G2L	**80.42**	84.86	94.12	89.42	91.2	74.72	81.76	0.866	0.798	84.76
QT+G2L-SP	80.21	**85.6**	94.38	**89.51**	90.4	74.96	81.89	0.861	0.794	84.71
SP+G2L-QT	79.87	84.25	93.59	89.01	92.0	75.77	82.5	0.864	0.796	84.77

Downstream Tasks Table 3.1 shows scores of multiple models on the SentEval downstream tasks from table 2.1. All the approaches compared in this table learn representations from unlabeled data. GloVe bag-of-words sentence representations have dimension of 300. Dimensions for SDAE and FastSent are 2400 and lower than 500 respectively. MC-QT and the rest of the models have dimensionality of 4800. Ablating the three tasks QT, SP and G2L leads to the importance or gains from the corresponding task. Results show that having all three tasks perform better than removing each one of them, but worse than MC-QT. Moreover, it can be seen that removing SP had the biggest effect on the performance, and removing QT had the smallest effect. This shows the importance of each task in training, but the relationship between these pre-training tasks could be more complex during the pre-training phase.

Table 3.2 – Comparison against supervised models on downstream tasks from SentEval.

Model	**MR**	**CR**	**SUBJ**	**MPQA**	**SST-2**	**TREC**	**MSRP**		**SICK**
							Acc	F1	r
GenSen	**82.5**	**87.7**	94.0	**90.9**	83.2	**93.0**	**78.6**	**84.4**	**0.888**
InferSent	81.1	86.3	92.4	90.2	84.6	88.2	76.2	83.1	0.884
DictRep	76.7	78.7	90.7	87.2	-	81.0	68.4	76.8	-
CaptionRep	61.9	69.3	77.4	70.8	-	72.2	-	-	-
NMT (En to Fr)	64.7	70.1	84.9	81.5	-	82.8	-	-	-
QT+SP+G2L	79.97	84.7	**94.58**	89.47	**84.62**	92.0	74.67	82.02	0.867

Table 3.2 shows our approach against supervised models that learn their representations from labeled data. DictRep, CaptionRep and NMT (En to Fr) are models introduced by Hill et al. (2016) which learn representations from mapping words and their definition, matching images and caption and neural machine translation. InferSent model is from Conneau et al. (2017) which is trained on the natural language inference task. GenSen is the model from Subramanian et al. (2018) which is trained in a multi-task setup on NLI, Skip-thought, NMT, constituency parsing. GenSen has a strong performance on most of the tasks, and it is trained on approximately 124M sentence pairs (including the 74M sentences of BookCorpus). Our model outperforms GenSen on the SUBJ and SST-2 tasks.

Nearest Neighbours To have a better idea of the embedding space and its quality we extracted nearest neighbours of random query sentences. Table 3.3 shows a random query sentence from the BookCorpus and the corresponding nearest neighbours from a pool of sentences from BookCorpus for the QT+SP+G2L model. It is interesting how close in meaning the query and the neighbours are. In the second example it is interesting how the representations for the query and N3 are close while their syntactic properties are not that close.

Table 3.3 – Nearest neighbours retrieved by our model from a random pool of sentences of BookCorpus dataset.

QT+SP+G2L	
Query	Patrick ' s stomach tightened a bit .
N1	Patrick hung up and went back to wait and pray for the happy ending of that adventure to come soon .
N2	He had a feeling about all that .
N3	He felt a pressure on him .
Query	Any other detail was lost in the moment of panic .
N1	The danger was imminent .
N2	They were caught off guard and had started to panic .
N3	She was unable to control the shivering in her body .
Query	He could see that her mind was working through whether or not to accept his offer , and he wondered if women were even allowed to read here .
N1	He could read her too well , and she was already feeling raw and exposed .
N2	He wanted to ask about something else , but was finding it difficult to speak .
N3	He nodded , and she wondered if this uncomfortable conversation was finally at an end .

3.5 Conclusion and Future Work

This chapter explores a set of pre-training tasks and takes advantage of mutual information maximization to learn representations in an unsupervised method. These tasks, namely, quick thoughts objective, self prediction and global to local prediction (described in section 3.2) find structure that is effective in transfer tasks. These ”transfer tasks” are downstream tasks which are not used during training/pre-training. We evaluate our representations on downstream tasks from SentEval (table 2.1) toolkit against models and approaches that use unlabeled data as well as those that use labeled/structured data.

An ablation study is done on these tasks and based on the results we conclude that:

— Applying all three tasks together results in better overall performance.

— Removing the self prediction task led to the largest drop in the average score. Surprisingly, removing the quickthought task had the smallest effect on the average scores on the downstream tasks.

— Our approach achieves somewhat comparable results to supervised methods which learn their representations from structured data and achieves better scores in the SST-2 and SUBJ tasks.

— The nearest neighbour sentences retrieved show that the sentence representations learned by our model captures the meaning and semantics of sentences rather than just focusing on the syntax.

We hope that these explorations and comparisons between different tasks will help us to have a better understanding of self-supervision, unsupervised task design and architecture choices.

Prologue to the Article

Ordered Memory.Yikang Shen, Arian Hosseini, Shawn Tawn, Zhouhan Lin, Alessandro Sordoni, and Aaron Courville. Accepted in the 33rd conference on Neural Information Processing Systems, NeurIPS, 2019

An important goal in human-like AI is to have natural language processing models that take the compositional property of language into account. This allows the models to understand the meaning of an expression from the meaning of its parts (Montague, 1970; Dowty, 2007). Studies have shown the failure of neural networks in achieving compositionality (Gershman and Tenenbaum, 2015; Lake and Baroni, 2018; Loula et al., 2018). Dasgupta et al. (2018) propose a new dataset for the natural language inference task that requires compositionality in order to be solved. They evaluate the (state-of-the-art) sentence representations from InferSent (Conneau et al., 2017) and report poor results on their dataset, showing that there is a gap between sentence representation methods and humans in learning compositionality. Ordered Memory is a step towards closing this gap. Specifically, the model presented in this work is trying to compute a sentence representation[1] from a tree structure. We did not try to pre-train the proposed model with self-supervised tasks, but this idea can be explored in future work.

Personal Contribution. I contributed to architecture design and implementations. Then, I implemented and carried out experiments on ListOps, SCAN and Stanford Sentiment Treebank (SST). Yikang and I conducted different experiments on SST (binary and fine-grained). We implemented the majority of the code in pytorch (Paszke et al., 2017). I contributed to the writing of the paper as well, but most of the paper is written by Yikang Shen and Shawn Tawn, with valuable inputs, reviews and rewritings from our co-authors Alessandro Sordoni and Zhohan Lin, and my supervisor Prof. Aaron Courville.

1. The input sequences in experiments 4.4.1 and 4.4.2 are in fact utterances, but one could consider them to be sentences.

4 Ordered Memory

4.1 Introduction

A long-sought after goal in natural language processing is to build models that account for the compositional nature of language — granting them an ability to understand complex, unseen expressions from the meaning of simpler, known expressions (Montague, 1970; Dowty, 2007). Despite being successful in language generation tasks, recurrent neural networks (RNNs, Elman 1990) fail at tasks that explicitly require and test compositional behavior (Lake and Baroni, 2018; Loula et al., 2018). In particular, Bowman et al. (2015), and later Bahdanau et al. (2019) give evidence that, by exploiting the appropriate compositional structure of the task, models can generalize better to out-of-distribution test examples. Results from Andreas et al. (2016) also indicate that recursively composing smaller modules results in better representations. The remaining challenge, however, is learning the underlying structure and the rules governing composition from the observed data alone. This is often referred to as the *grammar induction* (Chen, 1995; Cohen et al., 2011; Roark, 2001; Chelba and Jelinek, 2000; Williams et al., 2018).

Fodor and Pylyshyn (1988) claim that "cognitive capacities always exhibit certain symmetries, so that the ability to entertain a given thought implies the ability to entertain thoughts with semantically related contents," and use the term *systematicity* to describe this phenomenon. Exploiting known symmetries in the structure of the data has been a useful technique for achieving good generalization capabilities in deep learning, particularly in the form of convolutions (Fukushima, 1980), which leverage parameter-sharing. If we consider architectures used in Socher et al. (2013) or Tai et al. (2015), the same recursive operation is performed at known points along the input where the substructures are meant to be composed. Could symmetries in the structure of natural language data be learned and exploited by models that operate on them?

In recent years, many attempts have been made in this direction using neural network architectures (Grefenstette et al., 2015; Bowman et al., 2016; Williams et al., 2018; Yogatama et al., 2018; Shen et al., 2019; Dyer et al., 2016). These models typically augment a recurrent neural network with a stack and a buffer which operate in a similar way to how a shift-reduce parser builds a parse-tree. While some assume that ground-truth trees are available for supervised learning (Bowman et al., 2016; Dyer et al., 2016), others use reinforcement learning (RL) techniques to learn the optimal sequence of shift reduce actions in an unsupervised fashion (Yogatama et al., 2018).

To avoid some of the challenges of RL training (Havrylov et al., 2019), some approaches use a *continuous stack* (Grefenstette et al., 2015; Joulin and Mikolov, 2015; Yogatama et al., 2018). These can usually only perform one push or pop action per time step, requiring different mechanisms — akin to adaptive computation time (ACT, Graves 2016; Jernite et al. 2017) — to perform the right number of shift and reduce steps to express the correct parse. In addition, continuous stack models tend to "blur" the stack due to performing a "soft" shift of either the pointer to the head of the stack (Grefenstette et al., 2015), or all the values in the stack (Joulin and Mikolov, 2015; Yogatama et al., 2018). Finally, while these previous models can learn to manipulate a stack, they lack the capability to *lookahead* to future tokens before performing the stack manipulation for the current time step.

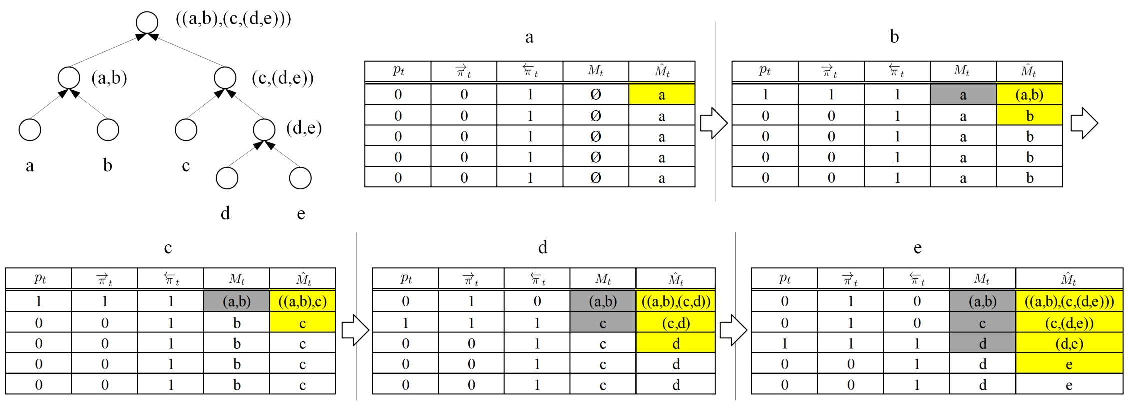
In this paper, we propose a novel architecture: *Ordered Memory* (OM), which includes a new Stick-breaking Attention mechanism and a new Gated Recurrent Cell. We demonstrate that our method generalizes for synthetic tasks where the ability to parse is crucial to solving them. In the Logical inference dataset (Bowman et al., 2015), we show that our model can systematically generalize to unseen combination of operators. In the ListOps dataset (Nangia and Bowman, 2018), we show that our model can learn to solve the task with an order of magnitude less training examples than the baselines. The parsing experiments shows that our method can effectively recover the latent tree structure of the both tasks with very high accuracy. We also perform experiments on the Stanford Sentiment Treebank, in both binary classification and fine-grained settings (SST-2 & SST-5), and find that we achieve comparative results to the current benchmarks.

4.2 Related Work

Composition with recursive structures has been shown to work well for certain types of tasks. Pollack (1990) first suggests their use with distributed representations. Later, Socher et al. (2013) shows their effectiveness on sentiment analysis tasks. Recent work has demonstrated that recursive composition of sentences is crucial to systematic generalisation (Bowman et al., 2015; Bahdanau et al., 2019). Kuncoro et al. (2018) also demonstrate that architectures like Dyer et al. (2016) handle syntax-sensitive dependencies better for language-related tasks.

Schützenberger (1963) first showed an equivalence between push-down automata (stack-augmented automatons) and context-free grammars. Knuth (1965) introduced the notion of a shift-reduce parser that uses a stack for a subset of formal languages that can be parsed from left to right. This technique for parsing has been applied to natural language as well: Shieber (1983) applies it to English, using assumptions about how native English speakers parse sentences to remove ambiguous parse candidates. More recently, Maillard et al. (2019) shows that a soft tree could emerge from all possible tree structures through back propagation.

The idea of using neural networks to control a stack is not new. Zeng et al. (1994) uses gradient estimates to learn to manipulate a stack using a neural network. Das et al. (1992) and Mozer and Das (1993) introduced the notion of a *continuous stack* in order for the model to be fully differentiable. Much of the recent work with stack-augmented networks built upon the development of neural attention (Graves, 2013; Bahdanau et al., 2015; Weston et al., 2015). Graves et al. (2014) proposed methods for reading and writing using a head, along with a "soft" shift mechanism. Apart from using attention mechanisms, Grefenstette et al. (2015) proposed a neural stack where the push and pop operations are made to be differentiable, which worked well in synthetic datasets. Yogatama et al. (2017) proposes RL-SPINN where the discrete stack operations are directly learned by reinforcement learning.



a

p_t	$\overrightarrow{\pi}_t$	$\overleftarrow{\pi}_t$	M_t	$\hat{M}_t$
0	0	1	Ø	a
0	0	1	Ø	a
0	0	1	Ø	a
0	0	1	Ø	a
0	0	1	Ø	a

b

p_t	$\overrightarrow{\pi}_t$	$\overleftarrow{\pi}_t$	M_t	$\hat{M}_t$
1	1	1	a	(a,b)
0	0	1	a	b
0	0	1	a	b
0	0	1	a	b
0	0	1	a	b

c

p_t	$\overrightarrow{\pi}_t$	$\overleftarrow{\pi}_t$	M_t	$\hat{M}_t$
1	1	1	(a,b)	((a,b),c)
0	0	1	b	c
0	0	1	b	c
0	0	1	b	c
0	0	1	b	c

d

p_t	$\overrightarrow{\pi}_t$	$\overleftarrow{\pi}_t$	M_t	$\hat{M}_t$
0	1	0	(a,b)	((a,b),(c,d))
1	1	1	c	(c,d)
0	0	1	c	d
0	0	1	c	d
0	0	1	c	d

e

p_t	$\overrightarrow{\pi}_t$	$\overleftarrow{\pi}_t$	M_t	$\hat{M}_t$
0	1	0	(a,b)	((a,b),(c,(d,e)))
0	1	0	c	(c,(d,e))
1	1	1	d	(d,e)
0	0	1	d	e
0	0	1	d	e

Figure 4.1 – An example run of the OM model. Let the input sequence a, b, c, d, e and its hierarchical structure be as shown in the figure. Ideally, the OM model will output the values shown in the above tables. The occupied slots in M_t are highlighted in gray. The yellow slots in $\hat{M}_t$ are slots that can be attended on in time-step $t+1$. At the first time-step ($t=1$), the model will initialize the candidate memory $\hat{M}_1$ with input a and the memory M_0 with zero vectors. At $t=2$, the model attends on the last memory slot to compute M_1 (Eqn. 4.5), followed by $\hat{M}_2$ (Eqn. 4.7). At $t=3$, given the input c, the model will attend on the last slot. Consequently the memory slot for b is erased by $\overrightarrow{\pi}_3$. Given Eqns. 4.6 and 4.7, our model will recursively compute every slot in the candidate memory $\hat{M}_t^i$ to include information from $\hat{M}_t^{i-1}$ and M_{t-1}^i. Since the cell$(\cdot)$ function only takes 2 inputs, the actual computation graph is a binary tree.

4.3 Model

The OM model actively maintains a stack and processes the input from left to right, with a one-step lookahead in the sequence. This allows the OM model to decide the local structure more accurately, much like a shift-reduce parser (Knuth, 1965).

At a given point t in the input sequence $\boldsymbol{x}$ (the t-th time-step), we have a memory of candidate sub-trees spanning the non-overlapping sub-sequences in $x_1, \ldots, x_{t-1}$, with each sub-tree being represented by one slot in the memory stack. We also maintain a memory stack of sub-trees that contains $x_1, \ldots, x_{t-2}$. We use the input x_t to choose its parent node from our previous candidate sub-trees. The descendant sub-trees of this new sub-tree (if they exist) are removed from the memory stack, and this new sub-tree is then added. We then build the new candidate sub-trees that include x_t using the current input and the memory stack. In what follows, we describe the OM model in detail. To facilitate a clearer description, a discrete attention scheme is assumed, but only "soft" attention is used in both the training and evaluation of this model.

Let D be the dimension of each memory slot and N be the number of memory slots. At time-step t, the model takes four inputs:

— M_{t-1}: a memory matrix of dimension $N \times D$, where each occupied slot is a distributed representation for sub-trees spanning the non-overlapping subsequences in $x_1, ..., x_{t-2}$;

— $\hat{M}_{t-1}$: a matrix of dimension $N \times D$ that contains representations for candidate subtrees that include the leaf node x_{t-1};

— $\overrightarrow{\pi}_{t-1}$: a vector of dimension N, where each element indicate whether the respective slot in M_{t-1} is occupied by a subtree.

— x_t: a vector of dimension D_{in}, the input at time-step t.

The model first transforms x_t to a D dimension vector.

$$\tilde{x}_t = LN(Wx_t + b) \tag{4.1}$$

where $LN(\cdot)$ is the layer normalization function (Ba et al., 2016).

To select the candidate representations from $\hat{M}_{t-1}$, the model uses $\tilde{x}_t$ as its query to attend on $\hat{M}_{t-1}$:

$$p_t = \mathrm{Att}(\tilde{x}_t, \hat{M}_{t-1}, \overrightarrow{\pi}_{t-1}) \tag{4.2}$$

$$\overrightarrow{\pi}^i_t = \sum_{j \le i} p^j_t \tag{4.3}$$

$$\overleftarrow{\pi}^i_t = \sum_{j \ge i} p^j_t \tag{4.4}$$

where p_t is a distribution over different memory slots in $\hat{M}_{t-1}$ and p^j_t is the probability on the j-th slot. Att is the attention mechanism that gives a categorical probability distribution over memory slots. This function is described in 4.3.1. Intuitively, p_t can be viewed as a pointer to the head of the stack, $\overrightarrow{\pi}_t$ is an indicator value over where the stack exists, and $\overleftarrow{\pi}_t$ indicates where in the memory will be forgotten (and replaced). $1 - \overleftarrow{\pi}_t$ is an indicator over what will be kept in the memory slots.

To compute M_t, we combine $\hat{M}_{t-1}$ and M_{t-1} through:

$$M^i_t = M^i_{t-1} \cdot (1 - \overleftarrow{\pi})^i + \hat{M}^i_{t-1} \cdot \overleftarrow{\pi}^i_t, \quad \forall i \tag{4.5}$$

```
Data: $x_1, ..., x_T$
Result: $o_T^N$
initialize $M_0, \hat{M}_0$;
for $i \leftarrow 1$ to $T$ do
    $\tilde{x}_t = LN(Wx_t + b)$;
    $p_t = \mathrm{Att}(\tilde{x}_t, \hat{M}_{t-1}, \overrightarrow{\pi}_{t-1})$;
    $\overrightarrow{\pi}_t^i = \sum_{j \le i} p_t^j$;
    $\overleftarrow{\pi}_t^i = \sum_{j \ge i} p_t^j$;
    $\hat{M}_t^0 = \tilde{x}_t$;
    for $i \leftarrow 1$ to $N$ do
        $M_t^i = M_{t-1}^i \cdot (1 - \overleftarrow{\pi}_t)^i + \hat{M}_{t-1}^i \cdot \overleftarrow{\pi}_t^i$;
        $o_t^i = \mathrm{cell}(M_t^i, \hat{M}_t^{i-1})$;
        $\hat{M}_t^i = \tilde{x}_t \cdot (1 - \overrightarrow{\pi}_t)^i + o_t^i \cdot \overrightarrow{\pi}_t^i$;
    end
end
return $o_T^N$;
```

Algorithm 1: Ordered Memory algorithm. The attention function $\mathrm{Att}(\cdot)$ is defined in section 4.3.1. The recursive cell function $\mathrm{cell}(\cdot)$ is defined in section 4.3.2.

Suppose p_t points at a memory slot y_t in $\hat{m}$. Then $\overleftarrow{\pi}_t$ will write $\hat{M}_{t-1}^i$ to M_t^i for $i \le y_t$, and $(1 - \overleftarrow{\pi}_t)$ will write M_{t-1}^i to M_t^i for $i > y_t$. In other words, Eqn. 4.5 copies everything from M_{t-1} to the current timestep, up to the where p_t is pointing.

We believe that this is a crucial point that differentiates our model from past stack-augmented models like Yogatama et al. (2017) and Joulin and Mikolov (2015). Both constructions have the 0-th slot as the top of the stack, and perform a convex combination of each slot in the memory / stack given the action performed. More concretely, a distribution over the actions that is not sharp (e.g. 0.5 for pop) will result in a weighted sum of an un-popped stack and a pop stack, resulting in a blurred memory state. Compounded, this effect can make such models hard to train. In our case, because $(1 - \overleftarrow{\pi}_t)^i$ is non-decreasing with i, its value will accumulate to 1 at or before N. This results in a full copy, guaranteeing that the earlier states are retained. This full retention of earlier states may play a part in the training process, as it is a strategy also used in Gulcehre et al. (2017), where all the memory slots are filled before any erasing or writing takes place.

To compute candidate memories for time step t, we recurrently update all memory slots with

$$o^i = \mathrm{cell}(M_t^i, \hat{M}_t^{i-1}) \tag{4.6}$$

$$\hat{M}_t^i = \tilde{x}_t \cdot (1 - \overrightarrow{\pi}_t)^{i+1} + o_t^i \cdot \overrightarrow{\pi}_t^i, \forall i \tag{4.7}$$

where $\hat{M}_t^0$ is x_t. The cell$(\cdot)$ function can be seen as a recursive composition function in a recursive neural network (Socher et al., 2013). We propose a new cell function in section 4.3.2.

The output of time step t is the last memory slot $\hat{M}_t^N$ of the new candidate memory, which summarizes all the information from $x_1, ..., x_t$ using the induced structure. The pseudo-code for the OM algorithm is shown in Algorithm 1.

4.3.1 Stick-Breaking Attention Mechanism

Instead of the cumax$(\cdot)$ function used in Shen et al. (2019) and the softmax$(\cdot)$ function, we use a stick breaking formulation to model the categorical distribution for selecting the split point to erase/preserve memory.

Given the projected input $\tilde{x}_t$ and candidate memory $\hat{M}_{t-1}^i$, we feed every $(\tilde{x}_t, \hat{M}_{t-1}^i)$ pair into a feed-forward network:

$$\alpha_t^i = \mathbf{w}_2^{Att} \tanh\left(\mathbf{W}_1^{Att} \begin{bmatrix} \hat{M}_{t-1}^i \\ \tilde{x}_t \end{bmatrix} + b_1\right) + b_2 \tag{4.8}$$

$$\beta_t^i = \exp\left(\alpha_t^i - \max_j \alpha_t^j\right) \tag{4.9}$$

where $\mathbf{W}_1^{Att}$ is $N \times 2N$ matrix, $\mathbf{w}_2^{Att}$ is N dimension vector, and the output β_t^i is a scalar. We further mask the β_t with the cumulative probability from the previous time step to prevent the model attending on non-existent parts of the stack:

$$\hat{\beta}_t^i = \beta_t^i \overrightarrow{\pi}_{t-1}^{i+1} \tag{4.10}$$

where $\overrightarrow{\pi}_{t-1}^{N+1} = 1$ and $\overrightarrow{\pi}_0^{\leq N} = 0$. We can then compute the probability distribution:

$$p_t^i = \hat{\beta}_t^i \cdot \prod_{j=1}^{i-1} \left(1 - \hat{\beta}_t^j\right) \tag{4.11}$$

This formulation bears similarity to the method used for the multi-pop mechanism seen in Yogatama et al. (2018).

4.3.2 Gated Recursive Cell

Instead of using the recursive cell proposed in TreeLSTM (Tai et al., 2015) and RNTN (Socher et al., 2010), we propose a new gated recursive cell, which is inspired by the feed-forward layer in Transformer (Vaswani et al., 2017). The inputs M_t^i and $\hat{M}_t^{i-1}$ are concatenated and fed into a fully connect feed-forward network:

$$\begin{bmatrix} v_t^i \\ h_t^i \\ c_t^i \\ u_t^i \end{bmatrix} = \mathbf{W}_2^{Cell} \,\text{ReLU}\left(\mathbf{W}_1^{Cell} \begin{bmatrix} \hat{M}_t^{i-1} \\ M_t^i \end{bmatrix} + b_1\right) + b_2 \tag{4.12}$$

Like the TreeLSTM, we compute the output with a gated combination of the inputs and u_t^i:

$$o_t^i \quad = \quad LN(\sigma(v_t^i) \odot \hat{M}_t^{i-1} + \sigma(h_t^i) \odot M_t^i + \sigma(c_t^i) \odot u_t^i) \tag{4.13}$$

where v_t^i is the vertical gate that controls the input from previous slot, h_t^i is horizontal gate that controls the input from previous time step, cg_t^i is cell gate that control the u_t^i, o_t^i is the output of cell function, and $LN(\cdot)$ share the same parameters with the one used in the Eqn. 4.1.

4.3.3 Relations to ON-LSTM and Shift-reduce Parser

Ordered Memory is implemented following the principles introduced in Ordered Neurons (Shen et al., 2019). Our model is related to ON-LSTM in several aspects: 1) The memory slots are similar to the chunks in ON-LSTM, when a higher ranking memory slot is forgotten/updated, all lower ranking memory slots should likewise be forgotten/updated; 2) ON-LSTM uses the monotonically non-decreasing master forget gate to preserve long-term information while erasing short term information. The OM model uses the cumulative probability $\overrightarrow{\pi}_t$; 3) Similarly, the master input gate used by ON-LSTM to control the writing of new information into the memory is replaced with the reversed cumulative probability $\overleftarrow{\pi}_t$ in the OM model.

At the same time, the internal mechanism of OM can be seen as a continuous version of a shift-reduce parser. At time step t, a shift-reduce parser could perform zero or several reduce steps to combine the heads of stack, then shift the word t into stack. Similarly, the OM can perform zero or several reduce steps. A single reduce step is implemented with the Gated Recursive Cell. This cell combines $\hat{M}_t^{i-1}$, the output of previous reduce step, and M_t^i, the next element in stack, into $\hat{M}_t^i$, the representation for new sub-tree. The number of reduce steps is modeled with the stick-breaking attention mechanism. The probability distribution p_t models the position of the head of stack after all necessary reduce operations are performed. The shift operations are implemented as copying the current input word x_t into candidate memory $\hat{M}_t$.

The upshot of drawing connections between our model and the shift-reduce parser is interpretability: We can approximately infer the computation graph constructed by our model with Algorithm 2 (see appendix). The algorithm can be used for the latent tree induction tasks in (Williams et al., 2018).

Table 4.1 – Test accuracy of the models, trained on operation lengths of ≤ 6, with their out-of-distribution results shown here (lengths 7-12). We ran 5 different runs of our models, giving the error bounds in the last row. The F_1 score is the parsing score with respect to the ground truth tree structure. The TreeCell is a recursive neural network based on the Gated Recursive Cell function proposed in section 4.3.2. For the Transformer and Universal Transformer, we follow the entailment architecture introduced in Radford et al. (2018). The model takes `<start> sentence1 <delim> sentence2 <extract>` as input, then use the vector representation for `<extract>` position at last layer for classification. *The results for RRNet were taken from Jacob et al. (2018).

Model	**Number of Operations**						**Sys. Gen.**		
	7	8	9	10	11	12	A	B	C
Sequential sentence representation									
LSTM	88	84	80	78	71	69	84	60	59
RRNet*	84	81	78	74	72	71	–	–	–
ON-LSTM	91	87	85	81	78	75	70	63	60
Inter sentence attention									
Transformer	51	52	51	51	51	48	53	51	51
Universal Transformer	51	52	51	51	51	48	53	51	51
Our acc	98 ± 0.4	97 ± 0.5	96 ± 0.8	94 ± 0.8	94 ± 1.5	92 ± 0.7	94	91	81
Our F1	87.9 ± 8.9								
Recursive NN + ground-truth structure									
TreeLSTM	94	92	92	88	87	86	91	84	76
TreeCell	98	96	96	95	93	92	95	95	90
TreeRNN	98	98	97	96	95	96	94	92	86

4.4 Experiments

We evaluate the tree learning capabilities of our model on two datasets: logical inference (Bowman et al., 2015) and ListOps (Nangia and Bowman, 2018). In these experiments, we infer the trees with our model by using Alg. 2 and compare them with the ground-truth trees used to generate the data. We evaluate parsing performance using the F_1 score[1]. We also evaluate our model on Stanford Sentiment Treebank (SST), which is the sequential labeling task described in Socher et al. (2013).

4.4.1 Logical Inference

The logical inference task described in Bowman et al. (2015) has a vocabulary of six words and three logical operations, `or`, `and`, `not`. The task is to classify the relationship of two logical clauses into seven mutually exclusive categories. We use

1. All parsing scores are given by `Evalb` https://nlp.cs.nyu.edu/evalb/

a multi-layer perceptron (MLP) with $(h_1, h_2, h_1 \circ h_2, |h_1 - h_2|)$ as input, where h_1 and h_2 are the $\hat{M}_T^N$ of their respective input sequences. We compare our model with LSTM, RRNet (Jacob et al., 2018), ON-LSTM (Shen et al., 2019), Tranformer (Vaswani et al., 2017), Universal Transformer (Dehghani et al., 2019), TreeLSTM (Tai et al., 2015), TreeRNN (Bowman et al., 2015), and TreeCell. We used the same hidden state size for our model and baselines for proper comparison. Hyper-parameters can be found in Appendix A.4. The model is trained on sequences containing up to 6 operations and tested on sequences with higher number (7-12) of operations.

The Transformer models were implemented by modifying the code from the Annotated Transformer [2]. The number of Transformer layers are the same as the number of slots in Ordered Memory. Unfortunately, we were not able to successfully train a Transformer model on the task, resulting in a model that only learns the marginal over the labels. We also tried to used Transformer as a sentence embedding model, but to no avail. Tran et al. (2018) achieves similar results, suggesting this could be a problem intrinsic to self-attention mechanisms for this task.

Length Generalization Tests The TreeRNN model represents the best results achievable if the structure of the tree is known. The TreeCell experiment was performed as a control to isolate the performance of using the cell$(\cdot)$ composition function versus using both using cell$(\cdot)$ and learning the composition with OM. The performance of our model degrades only marginally with increasing number of operations in the test set, suggesting generalization on these longer sequences never seen during training.

Parsing results There is a variability in parsing performance over several runs under different random seeds, but the model's ability to generalize to longer sequences remains fairly constant. The model learns a slightly different method of composition for consecutive operations. Perhaps predictably, these are variations that do not affect the logical composition of the subtrees. The source of different parsing results can be seen in Figure 4.2. The results suggest that these latent structures are still valid computation graphs for the task, in spite of the variations.

2. http://nlp.seas.harvard.edu/2018/04/03/attention.html

Table 4.2 – Partitions of the Logical Inference task from Bowman et al. (2014). Each partitions include a training set filtered out all data points that match the rule indicated in **Excluded**, and a test set formed by matched data points.

Part.	**Excluded**	**Training size**	**Test set example**
A	`* ( and (not a) ) *`	128,969	`f (and (not a))`
B	`* ( and (not *) ) *`	87,948	`c (and (not (a (or b))))`
C	`* ( {and,or} (not *) ) *`	51,896	`a (or (e (and c)))`
Full		135,529	

Systematic Generalization Tests Inspired by Loula et al. (2018), we created three splits of the original logical inference dataset with increasing levels of difficulty. Each consecutive split removes a superset of the previously excluded clauses, creating a harder generalization task. Each model is then trained on the ablated training set, and tested on examples unseen in the training data. As a result, the different test splits have different numbers of data points. Table 4.2 contains the details of the individual partitions.

The results are shown in the right section of Table 4.1 under Sys. Gen. Each column labeled A, B, and C are the model's aggregated accuracies over the unseen operation lengths. As with the length generalization tests, the models with the known tree structure performs the best on unseen structures, while sequential models degrade quickly as the tests get harder. Our model greatly outperforms all the other sequential models, performing slightly below the results of TreeRNN and TreeCell on the different partitions.

Combined with the parsing results, and our model's performance on these generalization tests, we believe this is strong evidence that the model has both (i) learned to exploit symmetries in the structure of the data by learning a good cell$(\cdot)$ function, and (ii) learned where and how to apply said function by operating its stack memory.

4.4.2 ListOps

Nangia and Bowman (2018) build a dataset with nested summary operations on lists of single digit integers. The sequences comprise of the operators `MAX`, `MIN`, `MED`, and `SUM_MOD`. The output is also an integer in $[0, 9]$ As an example, the input:

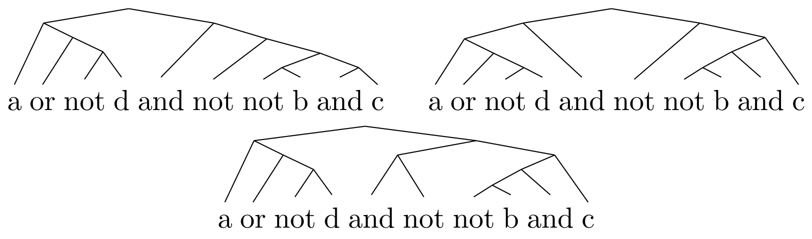

a or not d and not not b and c
a or not d and not not b and c
a or not d and not not b and c

Figure 4.2 – Variations in induced parse trees under different runs of the logical inference experiment. The left most tree is the ground truth and one of induced structures. We have removed the parentheses in the original sequence for this visualization. It is interesting to note that the different structures induced by our model are all valid computation graphs to produce the correct results.

`[MAX 2 9 [MIN 4 7 ] 0 ]` has the solution `9`. As the task is formulated to be easily solved with a correct parsing strategy, the task provides an excellent test-bed to diagnose models that perform tree induction. The authors binarize the structure by choosing the subtree corresponding to each list to be left-branching: the model would first take into account the operator, and then proceed to compute the summary statistic within the list. A right-branching parse would require the entire list to be maintained in the model's hidden state.

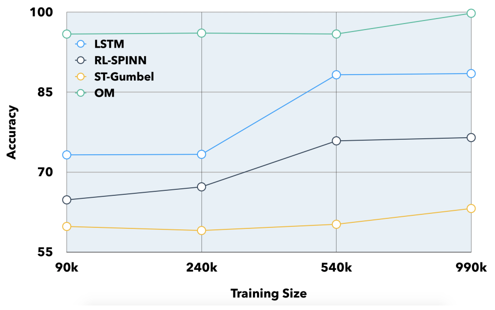
Our model achieves 99.9% accuracy, and an F_1 score of 100% on the model's induced parse tree (See Table 4.3a). This result is consistent across 3 different runs of the same experiment. In Nangia and Bowman (2018), the authors perform an experiment to verify the effect of training set size on the latent tree models. As the latent tree models (RL-SPINN and ST-Gumbel) need to parse the input successfully to perform well on the task, the better performance of the LSTM than those models indicate that the size of the dataset does not affect the ability to learn to parse much for those models. Our model seems to be more data efficient and solves the task even when only training on a subset of 90k examples (Fig. 4.3b).

4.4.3 Stanford Sentiment Treebank

The Stanford Sentiment Treebank is a classification task described in Socher et al. (2013). There are two settings: SST-2, which reduces the task down to a positive or negative label for each sentence (the neutral sentiment sentences are ignored), and SST-5, which is a fine-grained classification task which has 5 labels

Model	Accuracy	F_1
LSTM*	71.5±1.5	–
RL-SPINN*	60.7±2.6	71.1
Gumbel Tree-LSTM*	57.6±2.9	57.3
Transformer	57.4±0.4	–
Universal Transformer	71.5±7.8	–
Havrylov et al. (2019)	99.2±0.5	–
Ours	**99.9±0.02**	**100**

(a)



(b)

Figure 4.3 – (a) shows the accuracy of different models on the ListOps dataset. All models have 128 dimensions. Results for models with * are taken from Nangia and Bowman (2018). (b) shows our model accuracy on the ListOps task when varying the the size of the training set.

Table 4.3 – Accuracy results of models on the SST.

	SST-2	SST-5
Sequential sentence representation & other methods		
Radford et al. (2017)	91.8	52.9
Peters et al. (2018)	–	54.7
Brahma (2018)	91.2	56.2
Devlin et al. (2019)	94.9	–
Liu et al. (2019)	95.6	–
Recursive NN + ground-truth structure		
Tai et al. (2015)	88.0	51.0
Munkhdalai and Yu (2017)	89.3	53.1
Looks et al. (2017)	89.4	52.3
Recursive NN + latent / learned structure		
Choi et al. (2018)	90.7	53.7
Havrylov et al. (2019)	90.2±0.2	51.5±0.4
Ours (Glove)	90.4	52.2
Ours (ELMO)	92.0	55.2

for each sentence.

Current state-of-the-art models use pretrained contextual embeddings Radford et al. (2018); McCann et al. (2017); Peters et al. (2018). Building on ELMO Peters et al. (2018), we achieve a performance comparable with the current state-of-the-art for both SST-2 and SST-5 settings. However, it should be noted that our model is a sentence representation model. Table 4.3 lists our and related work's respective performance on the SST task in both settings.

4.5 Conclusion

In this paper, we introduce the Ordered Memory architecture. The model is conceptually close to previous stack-augmented RNNs, but with two important differences: 1) we replace the pop and push operations with a Stick-breaking Attention and a new writing and erasing mechanism inspired by Ordered Neurons (Shen et al., 2019); 2) we also introduce a new Gated Recursive Cell to compose lower level representations into higher level one. On the logical inference and ListOps tasks, we show that the model learns the proper tree structures required to solve them. As a result, the model can effectively generalize to longer sequence and combination of operators that is unseen in the training set, and the model is data efficient. We also demonstrate that our results on the SST are comparable with state-of-the-art models.

5 Conclusion

In this thesis, we first discussed neural networks as powerful functions to learn the underlying representations in data. We also touched on recurrent neural networks as sequential data processing units and how to train them. Moreover, we discussed the emerging field of self-supervised representation learning, and focused on its success and potential for use in natural language processing. Lastly, to close the background section, we discussed word representations, sentence representations and the typical approach to evaluate the utility of sentence embeddings.

In the first work presented, we explore a set of self-supervised learning tasks as pre-training methods to learn sentence representations from a large unlabeled corpora. These pre-training tasks (explained in section 3.2) are Quick Thought, Self Prediction and Global to Local prediction. We evaluate the representations learned from these pre-training tasks on a set of downstream tasks from SentEval and compare our approach with existing methods. Our approach achieves better results on MR, CR, SUBJ and MPQA compared to the unsupervised models. It also achieves somewhat comparable results to supervised methods in the literature. However, we realized that pre-training on these self-supervised tasks did not result in outstanding gain in performance. We concluded that we need to explore a larger suite of self-supervised methods for pre-training.

In the second work presented, we introduce a new architecture called Ordered Memory which bears a resemblance to stack-augmented RNNs. However, it has two major differences. The first one is that the push and pop operations are replaced with Stick-breaking attention mechanism, and writing and erasing are based on a new mechanism. The second difference is the introduced Gated Recursive Cell used to compose low level representations to form high level ones. In this work, we focused on utilizing the compositional structure of the tasks to have better generalization abilities (Bowman et al., 2015; Bahdanau et al., 2019). As a result, the OM model shows good generalization properties in the logical inference and ListOps tasks on examples that are unseen during training. We also report comparable with state-of-the-art results on the Stanford Sentiment Treebank. The OM model finds a sentence representation by composing parts of a sequence (or sentence) in a tree structure; however, we did not achieve impressive results on the natural language inference dataset from Dasgupta et al. (2018), for which they claim the model requires to learn compositionality to solve it. We concluded that we need to pre-train the OM model on a larger dataset as well as using a wider class of objectives in pre-training.

Learning representations from unlabeled data is a promising alley and can be a very powerful instrument for machine learning. In this thesis, we showed some methods of learning representations using self-supervised methods. However, self-supervised learning is still in its early stages and there is still disagreement about its definition as well as how to evaluate these methods. Although pre-training with self-supervised methods have shown impressive results in natural language tasks, there is still much to be done and understood. What are these self-supervision methods learning? Are they just memorizing large amounts of data that they are trained on? Do the models learn important concepts like compositionality using these methods? We hope that future work can build on top of our findings and results to answer such critical questions.

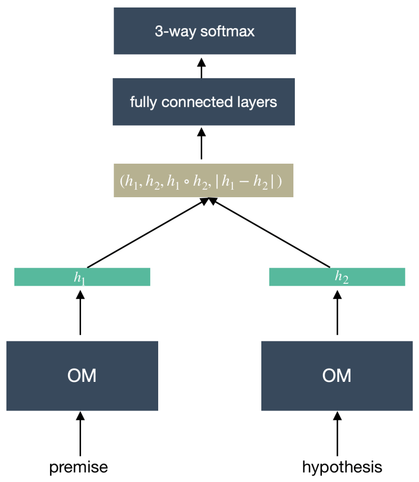
A Appendix

A.1 Compositionality in sentence embeddings

In this section we evaluate the performance of sentence representations from the OM model on the natural language inference task from Dasgupta et al. (2018) (described in section 2.2.1). Following Dasgupta et al. (2018), in this experiment a classifier model is first trained on the SNLI (Stanford Natural Language Inference) training set (Bowman et al., 2015). This task is a three-way classification task, in which pairs of sentences are classified into 'entailment', 'contradiction', or 'neutral'. Similar to the logical inference experiment in section 4.4.1, we use an MLP with $(h_1, h_2, h_1 \circ h_2, |h_1 - h_2|)$ as input (see figure A.1b), where h_1 and h_2 are the $\hat{M}_T^N$ of their respective input sequences (premise and hypothesis). We train the classifier model using GloVe embeddings (Pennington et al., 2014) on SNLI with a hidden size of 1120 dimensions, and same number of parameters as the InferSent model in Dasgupta et al. (2018) for proper comparison. Our trained model achieves 81.36% accuracy on validation and 80.77% accuracy on the test set of SNLI. Next, we evaluate the classifier on the proposed Comparisons dataset from Dasgupta et al. (2018). Table A.1a shows the performance of each model on different categories of the Comparisons dataset. Our model outperforms InferSent on the *more/less* and *not* categories; however, we did not achieve impressive results in this experiment.

Category	Ours	BOW-MLP	InferSent
same	40.42	50.0	**50.37**
more/less	**62.28**	30.24	50.35
not	**52.40**	48.98	45.24

(a)



(b)

Figure A.1

Figure A.2 – (a) Performance on different categories of the Comparisons dataset. (b) NLI classifier architecture using OM as sentence encoder.

A.2 Tree induction algorithm

```
Data: $p_1, ..., p_T$
Result: T
initialize queue $= [w_2, ..., w_T]$
   stack $= [w_1], h = \text{argmax}(p_1) - 1$;
for $i \leftarrow 2$ to $T$ do
    $y_i = \text{argmax}(p_i)$;
    $d = y_i - h$;
    if $d > 0$ then
        for $j \leftarrow 1$ to $d$ do
            if len(stack) $< 2$ then
                Break;
            end
            $e_1 =$ stack.pop();
            $e_2 =$ stack.pop();
            stack.push(node($e_1, e_2$));
        end
    end
    stack.push(queue.popleft());
    $h = y_i - 1$;
end
while len(stack) $> 2$ do
    $e_1 =$ stack.pop();
    $e_2 =$ stack.pop();
    stack.push(node($e_1, e_2$));
end
```

Algorithm 2: Shift-reduce parsing algorithm for generate parsing tree from Ordered Memory model. Here we greedily choose the argmax(p_t) as the head of stack for each slot.

A.3 Dynamic Computation Time

Given Eqn. 4.7, we can see that some o_t^is are multiplied with $\overrightarrow{\pi}_t^i$. It may not necessary to compute the cell function (Eqn. 4.6) if the cumulative probability $\overrightarrow{\pi}_t^i$ is smaller than a certain threshold. This threshold actively controls the number of computation steps that we need to perform for each time step. In our experiments, we set the threshold to be 10^{-5}. This idea of dynamically modulating the number of computational step is similar to Adaptive Computation Time (ACT) in Graves (2016), which attempts to learn the number of computation steps required that is dependent on the input. However, the author does not demonstrate savings in computation time. In Tan and Sim (2016), the authors implement a similar mechanism, but demonstrate computational savings only at test time.

A.4 Hyperparameters

Table A.1 – The hyperparameters used in the various experiments described. D is the dimension of each slot in the memory. There are 4 different dropout rates for different parts of the model: *In* dropout is applied at the embedding level input to the OM model. *Out* dropout is applied at the layers in the MLP before the final classification task. *Attention* dropout is applied at the layers inside stick-breaking attention mechanism. *Hidden* dropout is applied at various other points in the OM architecture.

Task	**Memory size**	**#slot**	**Dropout**				**Batch size**	**Embedding**	
			In	*Out*	*Hidden*	*Attention*		Size	Pretrained
Logic	200	15	0.2	0.2	0.2	0.2	128	200	None
ListOps	128	21	0.1	0.1	0.1	0.1	128	128	None
SST(Glove)	300	15	0.3	0.4	0.2	0.2	128	300	Glove
SST(ELMO)	300	15	0.3	0.2	0.2	0.3	128	1024	ELMo

Bibliography

Andreas, J., M. Rohrbach, T. Darrell, and D. Klein (2016). Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 39–48.

Ba, J. L., J. R. Kiros, and G. E. Hinton (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bachman, P., R. D. Hjelm, and W. Buchwalter (2019). Learning representations by maximizing mutual information across views. *ArXiv abs/1906.00910*.

Bahdanau, D., K. Cho, and Y. Bengio (2015). Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Bahdanau, D., S. Murty, M. Noukhovitch, T. H. Nguyen, H. de Vries, and A. C. Courville (2019). Systematic generalization: What is required and can it be learned? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Bahl, L. R., F. Jelinek, and R. L. Mercer (1990). A maximum likelihood approach to continuous speech recognition. In *Readings in speech recognition*, pp. 308–319. Elsevier.

Bengio, Y., R. Ducharme, P. Vincent, and C. Jauvin (2003). A neural probabilistic language model. *Journal of machine learning research 3*(Feb), 1137–1155.

Bowman, S. R., G. Angeli, C. Potts, and C. D. Manning (2015). A large annotated corpus for learning natural language inference. In L. Màrquez, C. Callison-Burch, J. Su, D. Pighin, and Y. Marton (Eds.), *Proceedings of the 2015 Conference on*

Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015, pp. 632–642. The Association for Computational Linguistics.

Bowman, S. R., J. Gauthier, A. Rastogi, R. Gupta, C. D. Manning, and C. Potts (2016, August). A fast unified model for parsing and sentence understanding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Berlin, Germany, pp. 1466–1477. Association for Computational Linguistics.

Bowman, S. R., C. D. Manning, and C. Potts (2015). Tree-structured composition in neural networks without tree-structured architectures. In *Proceedings of the 2015th International Conference on Cognitive Computation: Integrating Neural and Symbolic Approaches - Volume 1583*, COCO'15, Aachen, Germany, pp. 37–42. CEUR-WS.org.

Bowman, S. R., C. Potts, and C. D. Manning (2014). Recursive neural networks for learning logical semantics. *CoRR abs/1406.1827*.

Brahma, S. (2018). Suffix bidirectional long short-term memory. *CoRR abs/1805.07340*.

Chelba, C. and F. Jelinek (2000). Structured language modeling. *Computer Speech & Language 14*(4), 283–332.

Chelba, C., T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson (2014). One billion word benchmark for measuring progress in statistical language modeling. In *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*, pp. 2635–2639.

Chen, S. F. (1995). Bayesian grammar induction for language modeling. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, pp. 228–235. Association for Computational Linguistics.

Cho, K., B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio (2014, October). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014*

Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, pp. 1724–1734. Association for Computational Linguistics.

Choi, J., K. M. Yoo, and S.-g. Lee (2018). Learning to compose task-specific tree structures. In *Proceedings of the 2018 Association for the Advancement of Artificial Intelligence (AAAI). and the 7th International Joint Conference on Natural Language Processing (ACL-IJCNLP).*

Cohen, S. B., D. Das, and N. A. Smith (2011). Unsupervised structure prediction with non-parallel multilingual guidance. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 50–61. Association for Computational Linguistics.

Conneau, A. and D. Kiela (2018). Senteval: An evaluation toolkit for universal sentence representations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018.*

Conneau, A., D. Kiela, H. Schwenk, L. Barrault, and A. Bordes (2017). Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pp. 670–680.

Dai, Z., Z. Yang, Y. Yang, J. G. Carbonell, Q. V. Le, and R. Salakhutdinov (2019). Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pp. 2978–2988.

Das, S., C. L. Giles, and G.-Z. Sun (1992). Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. In *Proceedings of The Fourteenth Annual Conference of Cognitive Science Society. Indiana University*, pp. 14.

Dasgupta, I., D. Guo, A. Stuhlmüller, S. Gershman, and N. Goodman (2018). Evaluating compositionality in sentence embeddings. In C. Kalish, M. A. Rau,

X. J. Zhu, and T. T. Rogers (Eds.), *Proceedings of the 40th Annual Meeting of the Cognitive Science Society, CogSci 2018, Madison, WI, USA, July 25-28, 2018*. cognitivesciencesociety.org.

Dehghani, M., S. Gouws, O. Vinyals, J. Uszkoreit, and L. Kaiser (2019). Universal transformers. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Devlin, J., M. Chang, K. Lee, and K. Toutanova (2019). BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pp. 4171–4186.

Dolan, B., C. Quirk, and C. Brockett (2004). Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *Proceedings of the 20th International Conference on Computational Linguistics*, COLING '04, Stroudsburg, PA, USA. Association for Computational Linguistics.

Dowty, D. (2007). Compositionality as an empirical problem. In *In Chris Barker and Pauline Jacobson (eds.) Direct Compositionality*, pp. 23–101.

Dyer, C., A. Kuncoro, M. Ballesteros, and N. A. Smith (2016). Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 199–209.

Elman, J. L. (1990). Finding structure in time. *Cognitive science 14*(2), 179–211.

Fodor, J. A. and Z. W. Pylyshyn (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition 28*(1-2), 3–71.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics 36*(4), 193–202.

Gan, Z., Y. Pu, R. Henao, C. Li, X. He, and L. Carin (2016). Unsupervised learning of sentence representations using convolutional neural networks. *CoRR abs/1611.07897*.

Gershman, S. and J. B. Tenenbaum (2015). Phrase similarity in humans and machines. In *CogSci*. Citeseer.

Glorot, X. and Y. Bengio (2010). Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Graves, A. (2016). Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*.

Graves, A., G. Wayne, and I. Danihelka (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Grefenstette, E., K. M. Hermann, M. Suleyman, and P. Blunsom (2015). Learning to transduce with unbounded memory. In *Advances in Neural Information Processing Systems*, pp. 1828–1836.

Gulcehre, C., S. Chandar, and Y. Bengio (2017). Memory augmented neural networks with wormhole connections. *arXiv preprint arXiv:1701.08718*.

Gutmann, M. U. and A. Hyvärinen (2012, February). Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *J. Mach. Learn. Res. 13*(1), 307–361.

Hashimoto, K., C. Xiong, Y. Tsuruoka, and R. Socher (2017). A joint many-task model: Growing a neural network for multiple NLP tasks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pp. 1923–1933.

Hassan, H., A. Aue, C. Chen, V. Chowdhary, J. Clark, C. Federmann, X. Huang, M. Junczys-Dowmunt, W. Lewis, M. Li, S. Liu, T. Liu, R. Luo, A. Menezes, T. Qin, F. Seide, X. Tan, F. Tian, L. Wu, S. Wu, Y. Xia, D. Zhang, Z. Zhang, and M. Zhou (2018). Achieving human parity on automatic chinese to english news translation. *CoRR abs/1803.05567*.

Havrylov, S., G. Kruszewski, and A. Joulin (2019). Cooperative learning of disjoint syntax and semantics. *In Proc. of NAACL-HLT*.

Hermann, K. M. and P. Blunsom (2014). Multilingual distributed representations without word alignment. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, Canada.*

Hill, F., K. Cho, and A. Korhonen (2016). Learning distributed representations of sentences from unlabelled data. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pp. 1367–1377.

Hill, F., K. Cho, A. Korhonen, and Y. Bengio (2016). Learning to understand phrases by embedding the dictionary. *Transactions of the Association for Computational Linguistics 4*, 17–30.

Hjelm, R. D., A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio (2019). Learning deep representations by mutual information estimation and maximization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019.*

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 6*(2), 107–116.

Hochreiter, S. and J. Schmidhuber (1997). Long short-term memory. *Neural computation 9*(8), 1735–1780.

Hornik, K., M. Stinchcombe, and H. White (1989). Multilayer feedforward networks are universal approximators. *Neural networks 2*(5), 359–366.

Hu, M. and B. Liu (2004). Mining and summarizing customer reviews. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, New York, NY, USA, pp. 168–177. ACM.

Jacob, A. P., Z. Lin, A. Sordoni, and Y. Bengio (2018). Learning hierarchical structures on-the-fly with a recurrent-recursive model for sequences. In *Proceedings of The Third Workshop on Representation Learning for NLP*, pp. 154–158.

Jernite, Y., E. Grave, A. Joulin, and T. Mikolov (2017). Variable computation in recurrent neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Joulin, A. and T. Mikolov (2015). Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in neural information processing systems*, pp. 190–198.

Kingma, D. P. and J. Ba (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kiros, R., Y. Zhu, R. Salakhutdinov, R. S. Zemel, R. Urtasun, A. Torralba, and S. Fidler (2015). Skip-thought vectors. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 3294–3302.

Knuth, D. E. (1965). On the translation of languages from left to right. *Information and control 8*(6), 607–639.

Koehn, P., F. Och, and D. Marcu (2003). Statistical Phrase-Based Translation. In *NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, Morristown, NJ, pp. 48–54. Association for Computational Linguistics: Association for Computational Linguistics.

Kuncoro, A., C. Dyer, J. Hale, D. Yogatama, S. Clark, and P. Blunsom (2018). Lstms can learn syntax-sensitive dependencies well, but modeling structure makes them better. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Volume 1, pp. 1426–1436.

Lake, B. M. and M. Baroni (2018). Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 2879–2888.

Le, Q. and T. Mikolov (2014a). Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pp. II–1188–II–1196. JMLR.org.

Le, Q. V. and T. Mikolov (2014b). Distributed representations of sentences and documents. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pp. 1188–1196.

LeCun, Y., Y. Bengio, and G. Hinton (2015, 5). Deep learning. *Nature 521*(7553), 436–444.

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pp. 2278–2324.

Liu, X., P. He, W. Chen, and J. Gao (2019). Multi-task deep neural networks for natural language understanding. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28-August 2, 2019, Volume 1: Long Papers*, pp. 4487–4496.

Liu, Y., M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov (2019). Roberta: A robustly optimized BERT pretraining approach. *CoRR abs/1907.11692*.

Logeswaran, L. and H. Lee (2018). An efficient framework for learning sentence representations. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Looks, M., M. Herreshoff, D. Hutchins, and P. Norvig (2017). Deep learning with dynamic computation graphs. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings.*

Loula, J., M. Baroni, and B. M. Lake (2018). Rearranging the familiar: Testing compositional generalization in recurrent networks. In *Proceedings of the Workshop: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2018, Brussels, Belgium, November 1, 2018*, pp. 108–114.

Maillard, J., S. Clark, and D. Yogatama (2019). Jointly learning sentence embeddings and syntax with unsupervised tree-lstms. *Natural Language Engineering 25*(4), 433–449.

Marelli, M., S. Menini, M. Baroni, L. Bentivogli, R. Bernardi, and R. Zamparelli (2014, May). A SICK cure for the evaluation of compositional distributional semantic models. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*, Reykjavik, Iceland, pp. 216–223. European Languages Resources Association (ELRA).

McCann, B., J. Bradbury, C. Xiong, and R. Socher (2017). Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, pp. 6297–6308.

Merity, S., N. S. Keskar, and R. Socher (2018a). An analysis of neural language modeling at multiple scales. *CoRR abs/1803.08240.*

Merity, S., N. S. Keskar, and R. Socher (2018b). Regularizing and optimizing LSTM language models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings.*

Mikolov, T., K. Chen, G. S. Corrado, and J. Dean (2013). Efficient estimation of word representations in vector space. In *ICLR Workshop.*

Mikolov, T., M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur (2010). Recurrent neural network based language model. In *INTERSPEECH 2010,* *11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pp. 1045–1048.

Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean (2013). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pp. 3111–3119.

Montague, R. (1970). Universal grammar. *Theoria 36*(3), 373–398.

Mozer, M. C. and S. Das (1993). A connectionist symbol manipulator that discovers the structure of context-free languages. In *Advances in neural information processing systems*, pp. 863–870.

Munkhdalai, T. and H. Yu (2017). Neural tree indexers for text understanding. In *Proceedings of the conference. Association for Computational Linguistics. Meeting*, Volume 1, pp. 11. NIH Public Access.

Nangia, N. and S. R. Bowman (2018). Listops: A diagnostic dataset for latent tree learning. In S. R. Cordeiro, S. Oraby, U. Pavalanathan, and K. Rim (Eds.), *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 2-4, 2018, Student Research Workshop*, pp. 92–99. Association for Computational Linguistics.

Pang, B. and L. Lee (2004). A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA. Association for Computational Linguistics.

Pang, B. and L. Lee (2005). Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of ACL*, pp. 115–124.

Pascanu, R., T. Mikolov, and Y. Bengio (2012). Understanding the exploding gradient problem. *CoRR abs/1211.5063*.

Paszke, A., S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer (2017). Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop.*

Pennington, J., R. Socher, and C. D. Manning (2014). Glove: Global vectors for word representation. In A. Moschitti, B. Pang, and W. Daelemans (Eds.), *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pp. 1532–1543. ACL.

Peters, M. E., M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer (2018). Deep contextualized word representations. In M. A. Walker, H. Ji, and A. Stent (Eds.), *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pp. 2227–2237. Association for Computational Linguistics.

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence 46*(1-2), 77–105.

Poole, B., S. Ozair, A. Van Den Oord, A. Alemi, and G. Tucker (2019, 09–15 Jun). On variational bounds of mutual information. In K. Chaudhuri and R. Salakhutdinov (Eds.), *Proceedings of the 36th International Conference on Machine Learning*, Volume 97 of *Proceedings of Machine Learning Research*, Long Beach, California, USA, pp. 5171–5180. PMLR.

Radford, A., R. Jozefowicz, and I. Sutskever (2017). Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*.

Radford, A., K. Narasimhan, T. Salimans, and I. Sutskever (2018). Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf*.

Radford, A., J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever (2018). Language models are unsupervised multitask learners. *URL* *https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf*.

Rajpurkar, P., R. Jia, and P. Liang (2018). Know what you don't know: Unanswerable questions for squad. In I. Gurevych and Y. Miyao (Eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*, pp. 784–789. Association for Computational Linguistics.

Roark, B. (2001). Probabilistic top-down parsing and language modeling. *Computational linguistics 27*(2), 249–276.

Schützenberger, M. P. (1963). On context-free languages and push-down automata. *Information and control 6*(3), 246–264.

Shen, Y., S. Tan, A. Sordoni, and A. C. Courville (2019). Ordered neurons: Integrating tree structures into recurrent neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Shieber, S. M. (1983). Sentence disambiguation by a shift-reduce parsing technique. In *Proceedings of the 21st annual meeting on Association for Computational Linguistics*, pp. 113–118. Association for Computational Linguistics.

Socher, R., C. D. Manning, and A. Y. Ng (2010). Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, Volume 2010, pp. 1–9.

Socher, R., A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642.

Subramanian, S., A. Trischler, Y. Bengio, and C. J. Pal (2018). Learning general purpose distributed sentence representations via large scale multi-task learning.

In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Sutskever, I., O. Vinyals, and Q. V. Le (2014). Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 27*, pp. 3104–3112. Curran Associates, Inc.

Tai, K. S., R. Socher, and C. D. Manning (2015). Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pp. 1556–1566. The Association for Computer Linguistics.

Tan, S. and K. C. Sim (2016). Towards implicit complexity control using variable-depth deep neural networks for automatic speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5965–5969. IEEE.

Tran, K. M., A. Bisazza, and C. Monz (2018). The importance of being recurrent for modeling hierarchical structure. In E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii (Eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pp. 4731–4736. Association for Computational Linguistics.

van den Oord, A., Y. Li, and O. Vinyals (2018). Representation learning with contrastive predictive coding. *CoRR abs/1807.03748*.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin (2017). Attention is all you need. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 5998–6008.

Voorhees, E. M. and L. P. Buckland (Eds.) (2012). *Proceedings of The Twenty-First Text REtrieval Conference, TREC 2012, Gaithersburg, Maryland, USA, November 6-9, 2012*, Volume Special Publication 500-298. National Institute of Standards and Technology (NIST).

Weston, J., S. Chopra, and A. Bordes (2015). Memory networks. In Y. Bengio and Y. LeCun (Eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.*

Wiebe, J., T. Wilson, and C. Cardie (2005). Annotating expressions of opinions and emotions in language. *Language Resources and Evaluation 1*(2), 0.

Williams, A., A. Drozdov, and S. R. Bowman (2018). Do latent tree learning models identify meaningful structure in sentences? *Transactions of the Association of Computational Linguistics, TACL. 6*, 253–267.

Williams, A., N. Nangia, and S. R. Bowman (2018). A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pp. 1112–1122.

Wilson, A. C., R. Roelofs, M. Stern, N. Srebro, and B. Recht (2017). The marginal value of adaptive gradient methods in machine learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 30*, pp. 4148–4158. Curran Associates, Inc.

Yang, Z., Z. Dai, Y. Yang, J. G. Carbonell, R. Salakhutdinov, and Q. V. Le (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR abs/1906.08237*.

Yogatama, D., P. Blunsom, C. Dyer, E. Grefenstette, and W. Ling (2017). Learning to compose words into sentences with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings.* OpenReview.net.

Yogatama, D., Y. Miao, G. Melis, W. Ling, A. Kuncoro, C. Dyer, and P. Blunsom (2018). Memory architectures in recurrent neural network language models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Zamir, A. R., A. Sax, W. B. Shen, L. J. Guibas, J. Malik, and S. Savarese (2018). Taskonomy: Disentangling task transfer learning. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 3712–3722.

Zeng, Z., R. M. Goodman, and P. Smyth (1994). Discrete recurrent neural networks for grammatical inference. *IEEE Transactions on Neural Networks 5*(2), 320–330.