Université de Montréal

Compression in Sequence to Sequence Learning for Natural Language Processing

par

Gabriele Prato

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

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

décembre 2019

Université de Montréal

Faculté des études supérieures et postdoctorales

Ce mémoire intitulé

Compression in Sequence to Sequence Learning for Natural Language Processing

présenté par

Gabriele Prato

a été évalué par un jury composé des personnes suivantes :

Mikhail Bessmeltsev
(président-rapporteur)

Alain Tapp
(directeur de recherche)

Gena Hahn
(membre du jury)

Mémoire accepté le :
30 juin 2019

Sommaire

Dans ce travail, nous proposons une méthode presque sans perte d'information pour encoder de longues séquences de texte ainsi que toutes leurs sous-séquences en des représentations riches en information. Nous testons notre méthode sur une tâche d'analyse de sentiments et obtenons de bons résultats avec les vecteurs de sous-phrases et de phrases. Ce travail présente aussi l'utilisation de la distillation de connaissance et de la quantification pour compresser le modèle de Transformer [Vaswani et al., 2017] pour la tâche de traduction. Nous sommes, au mieux de nos connaissances, les premiers à démontrer que le Transformer avec ses poids quantifiés à 8-bits peut obtenir un score BLEU aussi bon qu'avec ses poids de précisions pleines. De plus, en combinant la distillation de connaissance avec la quantification des poids, nous pouvons entraîner de plus petits réseaux Transformer et obtenir un taux de compression jusqu'à 12.59x, en ne perdant que seulement 2.51 BLEU sur la tâche de traduction WMT 2014 Anglais-Français, comparativement au modèle de base.

Le chapitre 1 introduit des concepts d'apprentissage machine pour le traitement des langues naturelles, concepts qui sont essentiels à la compréhension des deux papiers présentés dans cette thèse. Chapitre 2 et 3 couvrent respectivement chaque papier, avant de conclure par le chapitre 4.

Mots clés : apprentissage machine, apprentissage profond, traitement des langues naturelles, séquence à séquence, compression

Summary

In this work, we propose a near lossless method for encoding long sequences of texts as well as all of their sub-sequences into feature rich representations. We test our method on sentiment analysis and show good performance across all sub-sentence and sentence embeddings. This work also demonstrates the use of knowledge distillation and quantization to compress the original Transformer model [Vaswani et al., 2017] for the translation task. We are, to the best of our knowledge, the first to show that 8-bit quantization of the weights of the Transformer can achieve the same BLEU score as the full-precision model. Furthermore, when we combine knowledge distillation with weight quantization, we can train smaller Transformer networks and achieve up to 12.59x compression while losing only 2.51 BLEU off the baseline on the WMT 2014 English-to-French translation task.

Chapter 1 introduces machine learning concepts for natural language processing which are essential to understanding both papers presented in this thesis. Chapter 2 and 3 cover each paper respectively, before finally concluding with chapter 4.

Keywords: machine learning, deep learning, natural language processing, sequence to sequence, compression

Table des matières

Sommaire iii

Summary iv

Liste des tableaux viii

Table des figures ix

Remerciements xi

Chapitre 1. Introduction 1

1.1. General Purpose Sentence Embeddings 1

1.2. Compression of Neural Machine Translation Networks 1

Chapitre 2. Background 2

2.1. Tokenization 2

2.1.1. Full Word 2

2.1.2. ***n***-gram 3

2.1.3. Byte pair encoding 3

2.1.4. WordPiece 4

2.2. Word Embeddings 4

2.2.1. Word2vec 4

2.2.1.1. Skip-gram model 5

2.2.2. FastText 5

2.2.3. MUSE 6

2.2.4. GloVe 6

2.2.5. Learning From Scratch . . . 7
2.2.6. Positional Embeddings . . . 7
2.3. Sentence Embeddings . . . 8
2.3.1. Skip-Thought . . . 8
2.3.2. InferSent . . . 9
2.3.3. GenSen . . . 10
2.4. Sequence to Sequence Learning . . . 10
2.5. Recurrent Neural Networks . . . 11
2.5.1. Sequence to Sequence . . . 12
2.5.2. Stacked RNN . . . 13
2.5.3. Bidirectional RNN . . . 14
2.5.4. Long short-term memory . . . 15
2.5.5. Gated Recurrent Unit . . . 16
2.6. Attention . . . 17
2.7. Transformers . . . 18
2.7.1. Scaled Dot-Product Attention . . . 19
2.7.2. Multi-Head Attention . . . 20
2.7.3. Current State-Of-The-Art . . . 20
Chapitre 3. Related Work . . . 22
3.1. Representation Learning . . . 22
3.2. Neural Machine Translation . . . 23
3.3. Quantization . . . 24
3.4. Knowledge Distillation . . . 25
Chapter 4. Towards Lossless Encoding of Sentences . . . 26
4.1. Recursive Autoencoder . . . 27

4.1.1. Model Architecture 27
4.1.2. Step Encoding 28
4.1.3. Input Representation 29
4.1.4. Mean Squared Error 29
4.1.5. Implementation Details 29
4.2. Experiments 29
4.2.1. Autoencoding 30
4.2.2. Sentiment Analysis 31
Chapitre 5. Compressing Tranformers for Neural Machine Translation via Knowledge Distillation and Quantization 36
5.1. Compressing Transformers 37
5.1.1. Weight Quantization 37
5.1.2. Knowledge Distillation 38
5.1.3. Quantization & Distillation 38
5.2. Experiments 38
5.2.1. Transformer Variants 39
5.2.2. English to French Translation 40
5.2.3. Results 40
Chapitre 6. Conclusion 41
6.1. Future Work 41
Bibliographie 43

Liste des tableaux

4.1 Mean squared error loss of stacked LSTMs and our RAE model for different embedding sizes. All models are trained on the autoencoding task for 20 epochs and models of same embedding size have the same capacity. MSE is computed on the BookCorpus dev set [Zhu et al., 2015], between the input GloVe embeddings [Pennington et al., 2014] and output embeddings. 34

4.2 SST-5 and SST-2 performance on all and root nodes respectively. Model results in the first section are from the Stanford Treebank paper [Socher et al., 2013]. GenSen and $\text{BERT}_{\text{BASE}}$ results are from [Subramanian et al., 2018] and [Devlin et al., 2018] respectively. 34

5.1 Results of the different Transformer [Vaswani et al., 2017] variants on the WMT 2014 English-French corpus. Perplexity is per token, computed on the development set[2]. BLEU is measured with `multi-bleu.pl`[1] on `newstest2014`[2]. 39

Table des figures

2.1 The InferSent method for natural language inference. The premise and hypothesis are encoded into an embedding each of their own, then concatenated along with element-wise absolute difference and product, before finally being classified as either entailment, contradiction or neutral. Taken from [Conneau et al., 2017a]. . 9

2.2 Sequence to sequence example where a sequence of english words gets encoded into a representation which is then decoded to its french equivalent. 11

2.3 The recurrent neural network architecture. For each element x_t of the input sequence, a hidden state h_t is computed using the current input element as well as the previously computed hidden state h_{t-1}. In this example, an output y_t is also computed for each element of the input sequence. 11

2.4 Two recurrent neural networks in an sequence to sequence setup. One RNN is the encoder and the other the decoder. Each RNN has its own set of weights. This is an example setup, different variations are possible. 12

2.5 Two stacked recurrent neural networks computing an input sequence x_1 to x_3 and output the output sequence y_1 to y_3. Each RNN has its own set of weights W^1, b^1 and W^2, b^2 respectively. 13

2.6 Example of a bidirectional recurrent neural network setup, where one RNN computes the input sequence in the original order and the second RNN computes it in the reverse order. h_0 and g_4 are both padding hidden states. The final outputs y_t are computed using the h_t and g_t hidden states. 14

2.7 Visualization of the computations performed in a LSTM cell. h_{t-1} and h_t are the previous and the current hidden state and c_{t-1} and c_t are the previous and current cell state. 15

2.8 Visualization of the operations performed in a GRU cell. The cell computes the hidden state h_t by taking into input x_t and the previously computed hidden state h_{t-1} 16

2.9 The Transformer architecture. Taken from [Vaswani et al., 2017] 18

2.10 Left is the Scaled Dot-Product Attention mechanism and right the Multi-head attention mechanism. Taken from [Vaswani et al., 2017] 19

4.1 Example of our recursive autoencoder with an input sequence of length three. The encoder recursively takes two embeddings and outputs one until a single one is left and the decoder takes one embedding and outputs two until there are as many as in the original sequence. 27

4.2 Accuracy comparison of different embedding sizes (300, 512, 1024 and 2048) for different sequence lengths. Left is our recursive autoencoder and right a stacked LSTM. An output embedding is counted as correct if the closest embedding out of all the vocabulary is its corresponding input embedding. 31

4.3 Accuracy comparison of different embedding sizes (300, 512, 1024 and 2048) for different sequence lengths. Left is our recursive autoencoder and right a stacked LSTM. An output embedding is counted as correct if the closest embedding out of all the vocabulary is its corresponding input embedding. 32

4.4 Accuracy comparison of our RAE model versus a stacked LSTM for embedding sizes 512 and 1024. Models of same embedding size have the same capacity. 33

4.5 Difference in accuracy when counting an output embedding as correct if the corresponding input embedding is in the five closest versus the closest. Comparison is done on our RAE model with embedding sizes 1024 and 2048. 35

Remerciements

I would like to thank the following people : Prof. Alain Tapp, Prof. Sarath Chandar, Mathieu Duchesneau, Ella Charlaix, Mehdi Rezagholizadeh, Devansh Arpit, Sai Rajeswar, Sandeep Subramanian, Tom Bosc, Rosemary Ke, Akram Erraqabi, Karttikeya Mangalam, Konrad Żołna, Mikołaj Bińkowski, Samuel Lavoie, Evan Racah, Ankesh Anand, Michael Noukhovitch, Arian Hosseini, Chen Xing, Joseph Paul Cohen, Vinayak Tantia, César Laurent, Pierre-Luc Vaudry, David Poellhuber, Prof. Yoshua Bengio and everyone past and present at Mila, for being such great co-authors and colleagues, for their invaluable guidance and contribution and for building such a great research lab.

I am grateful to Marc-Olivier Pelletier, Charles Royer, Patrice Rousseau, Philippe Mondor and Jean-François Dionne, whom I've had to let down many times in order to make this happen.

Last but not least, thank you mom and dad for being such wonderful parents.

Chapitre 1

Introduction

This chapter covers the motivation behind the two proposed methods in this work.

1.1. General Purpose Sentence Embeddings

Compressing information by encoding it into a fixed size representation in such a way that perfect decoding is possible is challenging. Instead, most of the existing sentence encoding methods focus more on learning encoding such that the encoded representations are good enough for the downstream tasks. In the following work, we focus on perfectly decodable encoding of sentences which will be very useful in designing good generative models that can generate longer sentences.

1.2. Compression of Neural Machine Translation Networks

State-of-the-art neural machine translation methods make use of an enormous amount of parameters and thus inference on edge-devices such as smartphones is impractical unless compressed. All work on the compression of Transformers [Vaswani et al., 2017] has so far shown a drop in BLEU score. In this work, we compare the effects of knowledge distillation and quantization on Transformers, as well as the combination of both methods. To the best of our knowledge, we are the first to show that it is possible to compress the Transformer network without any drop in BLEU score.

Chapitre 2

Background

Before diving into the research segment of this thesis and to better understand the work done, some basic elements of natural language processing in machine learning need to be covered. This chapter introduces core elements of most machine learning methods in NLP. Starting with basic text tokenization, various word embedding methods for processing the text input of neural networks follow, then an overview of numerous approaches to computing sentence embeddings. An explanation of sequence to sequence learning, recurrent neural networks and its variations ensues, as well as a description of the attention mechanism. The chapter ends with the very recent and popular Transformer architecture.

2.1. Tokenization

The question of how to split the text is as old as natural language processing. The following subsections detail the most popular methods for tokenization of text in machine learning.

2.1.1. Full Word

This is a very basic approach, were each word is considered a token. Although simple, this method has been used in much successful work up to very recently. Its advantage is its easy combination with pretrained word embedding corpuses. The disadvantage is that it ignores the internal structure of words. For example, information learned about the word *eat* could be useful for the word *eaten*. Not that word tokenization forbids such information to be used between two or more words. Some embedding methods could make use of such

information for embedding generation. The process can be easier though with more flexible tokenization methods.

2.1.2. *n*-gram

The n-gram tokenization with $n = 1$ is simply character tokenization. For example, the 1-gram tokenization of the sentence *I bike.* would produce the tokenized sequence :

<I>, < >, <b>, <i>, <k>, <e>, <.>

This is usually inefficient and underperforms other tokenization methods. Generally, $n > 1$ is a better choice. The method starts at the very first character of the sequence and takes the first n characters as the first token. Then, the tokenization window moves by one character to the right and the second token is again the next n characters. This process continues until the window reaches the end of the sequence. For example, 3-gram tokenization of the sentence *I bike.* gives the following tokenized sequence :

<I b>, < bi>, <bik>, <ike>, <ke.>

Note the character overlap. This happens for $n > 1$, as the tokenization window only moves by one character at a time.

2.1.3. Byte pair encoding

Sennrich et al. [2016] proposed using byte pair encoding (BPE) [Gage, 1994] but for grouping characters instead of bytes. The method iteratively learns a vocabulary of character n-grams tokens. Before beginning, a special end of word symbol is added to every word to be able to reconstruct the original sequence back from the tokenized sequence. The initial vocabulary contains all the characters in the training corpus as well as the special end of word symbol. The training corpus is then tokenized with this vocabulary. Then the most frequent pair of consecutive tokens is added to the vocabulary and each occurrence of that pair is replaced by the new token. For example, if the most frequent token pair is <A> followed by <B>, then <AB> will be added to the vocabulary and all such occurrences will be replace by <AB>. Then, token co-occurrences are counted again and the most frequent token pair is again added to the vocabulary and replaced by the new token. This process repeats until the vocabulary grows to the desired size. An important point is that words are

separate entities, meaning no tokens may be paired between words and are also not counted as adjacent.

An advantage of BPE is that it is language independent. For bilingual machine translation, one could train a BPE tokenizer per language or train a single tokenizer on a corpus containing sequences in both languages. With BPE, there is also no out-of-vocabulary words, as the basic character tokens assures total coverage. This tokenization method is used by many popular and recent work in natural language processing [Gehring et al., 2017, Vaswani et al., 2017], as is WordPiece, which is presented in the next subsection.

2.1.4. WordPiece

Schuster and Nakajima [2012] proposed a tokenization method very similar to byte pair encoding, called WordPiece. Instead of merging token pairs based on frequency, the proposed solution is to train a language model on the tokenized corpus and merge token pairs which increase the likelihood the most. A language model is then retrained and the process continues until the vocabulary grows to the desired size, just like BPE.

2.2. Word Embeddings

Word embeddings serve as a more information rich representation of an input sequence than word ids. They encode information about a word into a fixed size vector. When training a neural network with such embeddings, one can use pretrained word embeddings or start with randomly initialized embeddings and learn them jointly while training the neural network. Of course, one could also start with pretrained embeddings and fine-tune them for the task currently training on. In the case of learning embeddings from scratch or fine-tuning pretrained embeddings, the general method is to simply compute gradient for these input embeddings and update them, just like any other weight in the neural network. There is a vast number of methods to compute word embeddings. The most popular are covered in the following subsections.

2.2.1. Word2vec

Word2vec [Mikolov et al., 2013b] is an unsupervised method for learning word embeddings, meaning input dataset does not require any labelling of responses. In the case of text for example, only the raw text is needed. By training a neural network on language

modeling, Word2vec uses the representations it learns to better predict surrounding words. The original paper proposes two variants of the language modeling task. The first one called Continuous Bag-of-Words (CBOW) gives to the model the surrounding words as input and then the model predicts the missing word. This can be, for example, a sentence in which one of the words has been blanked out. The second one, called the Skip-gram model, predicts surrounding words given a single word as input. The latter has empirically been shown to perform best [Mikolov et al., 2013b].

2.2.1.1. *Skip-gram model*

Given a training corpus with the vocabulary $w_1, \ldots, w_T$, we want the model to maximize the following log probability :

$$\sum_{t=1}^{T} \sum_{w_c \in C_t} \log p(w_c|w_t) \tag{2.2.1}$$

where C_t is the set of words surrounding w_t, for example in a sequence of text. For $p(w_c|w_t)$, the basic skip-gram method defines it as the softmax function :

$$p(w_c|w_t) = \frac{e^{s(w_t,w_c)}}{\sum_{j=1}^{W} e^{s(w_t,j)}} \tag{2.2.2}$$

where s is the following scoring function :

$$s(w_t,w_c) = v_{w_t}^{\top} v_{w_c} \tag{2.2.3}$$

and v_{w_t} and v_{w_c} are the word embedding vectors of w_t and w_c respectively.

Mikolov et al. [2013a] proposed to use negative sampling instead of a softmax. In this approach, instead of computing a probability on the whole vocabulary, the model needs to correctly predict if a given word is a surrounding word or not. The following binary logistic loss is used :

$$\log\left(1 + e^{-s(w_t,w_c)}\right) + \sum_{w_n \in N_t} \log\left(1 + e^{s(w_t,w_n)}\right) \tag{2.2.4}$$

where N_t are the randomly drawn negative samples not surrounding w_t.

2.2.2. FastText

An improvement to the Word2vec method mentioned above was proposed by Bojanowski et al. [2016]. Contrary to the original approach, FastText does not ignore the internal structure of words. Instead of learning a single vector representation per word, an embedding is

learned per character n-gram. A word embedding is the sum of its character n-gram vectors. The special boundary characters < and > are also added at the beginning and end of the word and the complete word also gets its own embedding. For example, with $n = 3$, the embedding for the word *prized* will be the sum of the embeddings of the following tokens :

<pr, pri, riz, ize, zed, ed>, <prized>

FastText also uses a different scoring function :

$$s(w_t, w_c) = \sum_{z \in Z_{w_t}} z^\top v_{w_c} \tag{2.2.5}$$

where Z_{w_t} is the set of embeddings of a given word w_t, while v_{w_c} is the sum of the embeddings of the word w_c.

FastText pretrained embeddings trained on Common Crawl and Wikipedia are available in multiple languages at `https://fasttext.cc/docs/en/crawl-vectors.html`.

2.2.3. MUSE

MUSE [Conneau et al., 2017b] is an unsupervised method for learning bilingual pairs of word embeddings. The idea is to transform the embeddings of one language to fit as much as possible the embeddings of the target language. For a word with a translation in both languages, the corresponding embedding in each language should be as close as possible to one another. More formally, given a language pair and their word embeddings X and Y respectively, the method learns a mapping W to minimize an adversarial loss between X and Y.

Multiple MUSE embeddings all aligned in a single vector space and trained from FastText embeddings are available at `https://github.com/facebookresearch/MUSE`.

2.2.4. GloVe

The GloVe method proposed by Pennington et al. [2014] learns word embeddings via a global word-word co-occurrence matrix of the training corpus. Two sets of vectors w, $\tilde{w}$ and their corresponding biases b and $\tilde{b}$ are learned. The final word embeddings are the sum of w and $\tilde{w}$. The model is trained via the following cost function :

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right) \left(w_i^\top \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2 \tag{2.2.6}$$

where V is the vocabulary size, X_{ij} is the number of times the word j appears in the context of the word i and f :

$$f(x) = \begin{cases} (x/x_{\max})^\alpha & \text{if } x < x_{\max} \\ 1 & \text{else} \end{cases} \tag{2.2.7}$$

where $x_{\max}$ and α are fixed constants. The original paper [Pennington et al., 2014] empirically found $x_{\max} = 100$ and $\alpha = 3/4$ to work best.

Multiple versions of GloVe embeddings, pretrained on different corpuses (Wikipedia, Gigaword, Common Crawl, Twitter), are available at `https://nlp.stanford.edu/projects/glove/`.

2.2.5. Learning From Scratch

Many recent state-of-the-art NLP models do not use any pretrained word embeddings [Vaswani et al., 2017, Subramanian et al., 2018, Radford, 2018, Devlin et al., 2018, Liu et al., 2019b]. They start from randomly initialized word embeddings and learn these representations jointly with the other weights of the model. Although, similarly to using pretrained representations, some of these methods begin by pretraining their whole model on unsupervised tasks to learn good representations before fine-tuning on downstream tasks [Radford, 2018, Devlin et al., 2018, Liu et al., 2019b].

2.2.6. Positional Embeddings

Some natural language processing methods encode word level positional information into embeddings so as to help the neural network. Specifically, some models by design cannot infer any order in the input sequence. For example, in the Transformer [Vaswani et al., 2017] architecture, if no positional information was given to the model, mixing up input words in any order would make no difference with the correctly ordered original input (see section 2.7 for more details). Not all models require being given positional information. For example, recurrent neural networks can model a past context thanks to the hidden state (see section 2.5).

There are many ways to encode positional information : scalars, one-hots, learned embeddings, complex functions [Vaswani et al., 2017] (see chapter 4). So far, no method has been established as superior, but providing models with positional information usually provides a

boost in performance (see section 4.1.2). Most methods either sum or concatenate positional information to the input. For example, the Transformer network [Vaswani et al., 2017] sums with the input word embeddings the positional fixed embeddings computed with :

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d}) \tag{2.2.8}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d}) \tag{2.2.9}$$

where pos is the position of the embedding in the sequence, i is the dimension of the vector and d is the size of the positional embedding, in this case the size of the word embeddings, since both need to be summed. The advantage of using such a function is that hopefully the model can better behave with unseen lengths compared with methods like one-hot encoding.

More simple approaches, like in section 4.1.2, simply encode positional information with a scalar or into a one-hot and concatenate it with the input. Concatenation is also more "powerful" than summing with the input, as it lets the model decide how this positional information should impact the original input.

Just as with word embeddings, it is also possible to learn positional embeddings.

2.3. Sentence Embeddings

It is the end goal in machine learning to have a model generalize well to as many tasks as possible. In this respect, work in machine learning NLP has been focused on learning sentence embeddings which can be useful for as many tasks as possible. So far, there has been no clear best type of method for doing so. Successful methods range from unsupervised learning to multi-task supervised learning and many different neural network architectures have been shown to perform well. In the following subsections, three popular such methods are explained.

2.3.1. Skip-Thought

One of the first method for learning general purpose sentence embeddings, Skip-Thought [Kiros et al., 2015], pushed for vectors which could be useful for as many downstream tasks as possible. This fully unsupervised method trained a language model on the BookCorpus [Zhu et al., 2015] dataset. First, an encoder RNN encodes a sentence into a fixed sized representation which is then used by two decoder RNNs, whom each decode the previous sentence

and the next sentence respectively. The trained model is then tested on 8 downstream tasks with the sole training of a linear classifier on top of the sentence embedding for tasks requiring it, no fine-tuning of any other weights. The eight evaluation tasks are the following : semantic relatedness, paraphrase detection, image-sentence ranking, two sentiment analysis tasks, subjectivity/objectivity classification, opinion polarity and question-type classification. These downstream tasks and more can be found in the SentEval benchmark [Conneau and Kiela, 2018].

For a totally unsupervised method, the Skip-Thought approach performed very well compared to other methods, on par with the then state-of-the-art methods. Its good performance on all tasks shows the robustness of the learned sentence embeddings.

2.3.2. InferSent

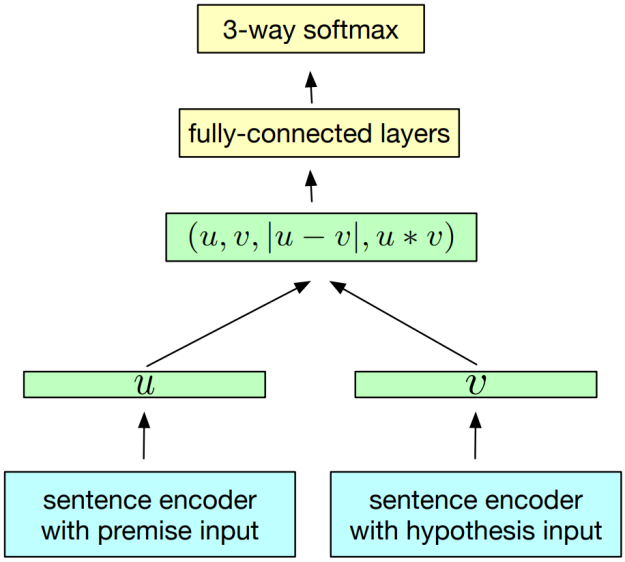

Figure 2.1. The InferSent method for natural language inference. The premise and hypothesis are encoded into an embedding each of their own, then concatenated along with element-wise absolute difference and product, before finally being classified as either entailment, contradiction or neutral. Taken from [Conneau et al., 2017a].

Compared to Skip-Thought [Kiros et al., 2015], the InferSent method [Conneau et al., 2017a] trains on a supervised task before evaluating its sentence embeddings on downstream tasks. Training is done on SNLI [Bowman et al., 2015], a natural language inference corpus

where given two sentences, a premise and a hypothesis, the model must classify between entailment, contradiction or neutral. Conneau et al. [2017a] compared the performance of seven different models, all trained on the SNLI task and then evaluated on the downstream tasks. The seven models where : LSTM, GRU, bidirectional GRU and bidirectional LSTM both with either mean or max pooling, a self-attentive network and a hierarchical convolutional network. In all cases, an encoder encodes both the premise and the hypothesis separately, generating an embedding for both. Then, the two representations are concatenated, along with the absolute values of their difference and their element-wise product. Finally, a fully connected network classifies the concatenation. Figure 2.1 presents this architecture via a graph. The trained models are then evaluated on the following downstream tasks : sentiment analysis, subjectivity/objectivity classification, opinion polarity, question-type, paraphrase detection, entailment and semantic relatedness, semantic text similarity and caption-image retrieval. These tasks can all be found in a single benchmark : SentEval [Conneau and Kiela, 2018].

The bidirectional LSTM with max pooling got the best results out of all models and the method achieved state-of-the-art results on most downstream tasks.

2.3.3. GenSen

Subramanian et al. [2018] proposed to learn generalizable sentence embeddings by training a sequence to sequence model on multiple supervised and unsupervised tasks. Their method uses a bidirectional GRU as the encoder, shared by all training tasks, and a unidirectional GRU per task as the decoder. They train the model on the following tasks : Skip-Thought (see section 2.3.1), neural machine translation, constituency parsing and natural language inference. The method obtained state-of-the-art results on the SentEval benchmark [Conneau and Kiela, 2018], beating Skip-Thought vectors and InferSent (see previous sections).

2.4. Sequence to Sequence Learning

Sequence to sequence learning [Sutskever et al., 2014] encodes a sequence and uses the encoded information to decode another sequence. Use cases are tasks such as machine translation, summarization, question answering. Figure 2.2 shows a simplified example where an

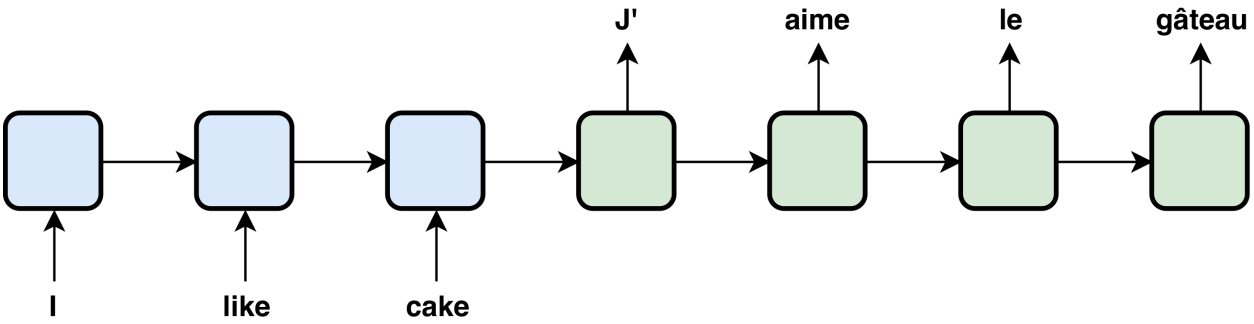

Figure 2.2. Sequence to sequence example where a sequence of english words gets encoded into a representation which is then decoded to its french equivalent.

english sentence gets encoded and then decoded back into a french sentence. Many other architectures than the one depicted in Figure 2.2 exist. The following sections go into details explaining some neural network architectures and their usage in sequence to sequence learning.

2.5. Recurrent Neural Networks

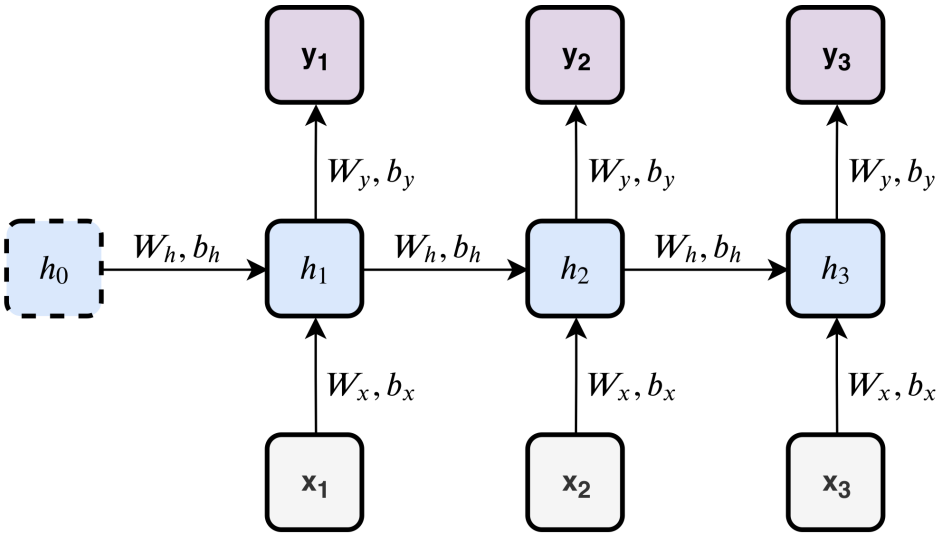

Figure 2.3. The recurrent neural network architecture. For each element x_t of the input sequence, a hidden state h_t is computed using the current input element as well as the previously computed hidden state h_{t-1}. In this example, an output y_t is also computed for each element of the input sequence.

Recurrent neural networks are a class of neural network architectures which iteratively apply a shared neural network "block" or cell to an input sequence. The RNN cell is applied to each element of the input sequence and computes a hidden state. These hidden states are also given as input to an RNN cell, where each cell takes both the current element t of the input sequence and the previous hidden state $t-1$. Figure 2.3 shows an example of this architecture for a sequence of length 3. Since no hidden states were computed prior to h_1, the h_0 hidden state is taken as input. This h_0 can be anything, but the values of this original hidden state are usually set to 0. As can be seen in Figure 2.3, the same set of weights are used to compute every x_t. Same for every h_t and y_t, the same set of weights are reused, where W are weights matrices and b the biases. Each hidden state h_t is computed the following way :

$$h_t = \tanh(W_x x_t + b_x + W_h h_{t-1} + b_h) \tag{2.5.1}$$

where tanh is the hyperbolic tangent activation function. The output y_t is computed by :

$$y_t = W_y h_t + b_y \tag{2.5.2}$$

Note that depending on the task, it is not necessary to compute an y_t for every x_t. Some tasks will only require to compute an y_t for the last element of the input sequence.

This is a very general description of the recurrent neural network architecture. Many variations exist. The most popular of these are detailed in the following sections.

2.5.1. Sequence to Sequence

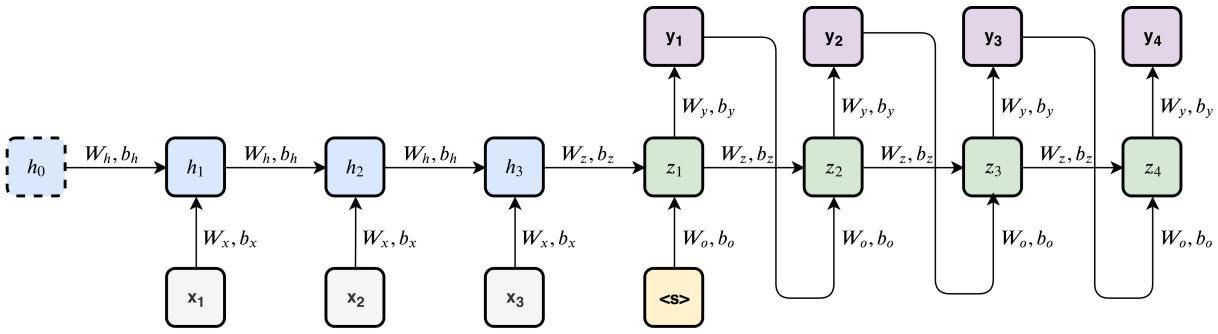

Figure 2.4. Two recurrent neural networks in an sequence to sequence setup. One RNN is the encoder and the other the decoder. Each RNN has its own set of weights. This is an example setup, different variations are possible.

The usual sequence to sequence setup (see section 2.4) with recurrent neural networks has one RNN as the encoder and another RNN as the decoder. The encoder takes as input a sequence of a certain length and encodes it to a fixed representation, the hidden state. Then, the decoder takes as input what is usually a start of sequence token and the final hidden state computed by the encoder to decode an output sequence. The output sequence can be of any length. For example, the stop condition could be when an outputted y_t corresponds to an end of sequence token. For the decoder, apart from the start of sequence token, every other input to the RNN cell is the previous output. Sometimes, the ground truth is given as input instead of the previous output. This is called teacher forcing. Figure 2.4 shows two RNNs in a sequence to sequence setup, where each RNN has its own set of weights and biases.

2.5.2. Stacked RNN

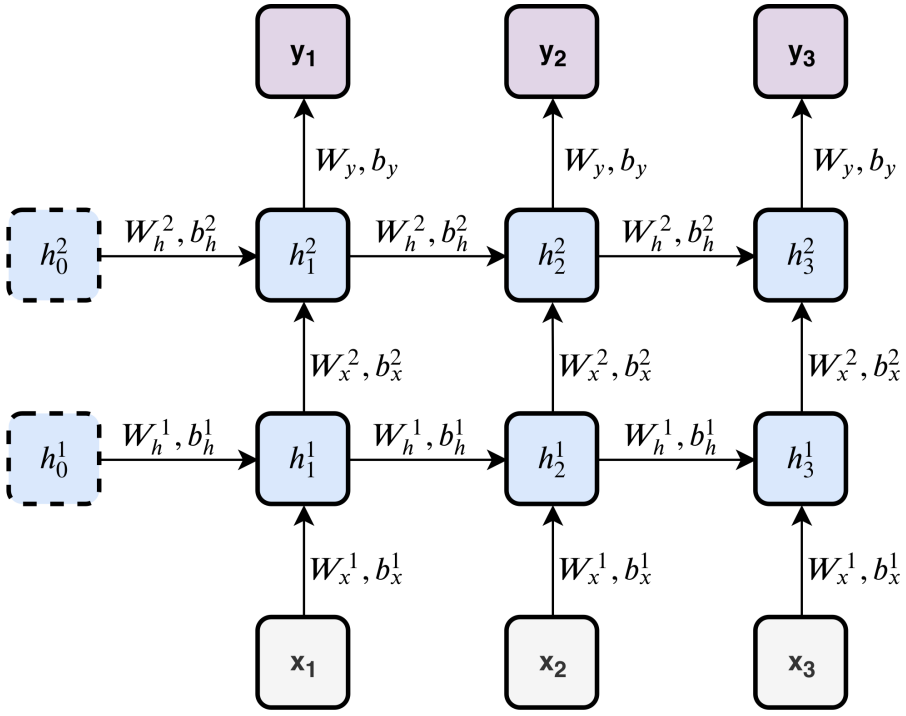

Figure 2.5. Two stacked recurrent neural networks computing an input sequence x_1 to x_3 and output the output sequence y_1 to y_3. Each RNN has its own set of weights W^1, b^1 and W^2, b^2 respectively.

One way to increase the depth of recurrent neural networks is to stack them one on top of the other [Schmidhuber, 1992]. The first RNN takes as input the input sequence just as in the regular RNN setup, any other RNN on top takes the hidden states computed by the RNN under them as input and the final RNN's hidden states are what is usually used to compute the final output. Figure 2.5 shows an example of two stacked RNNs. Each RNN has its own set of weights in this case. Of course, weight sharing is also possible.

2.5.3. Bidirectional RNN

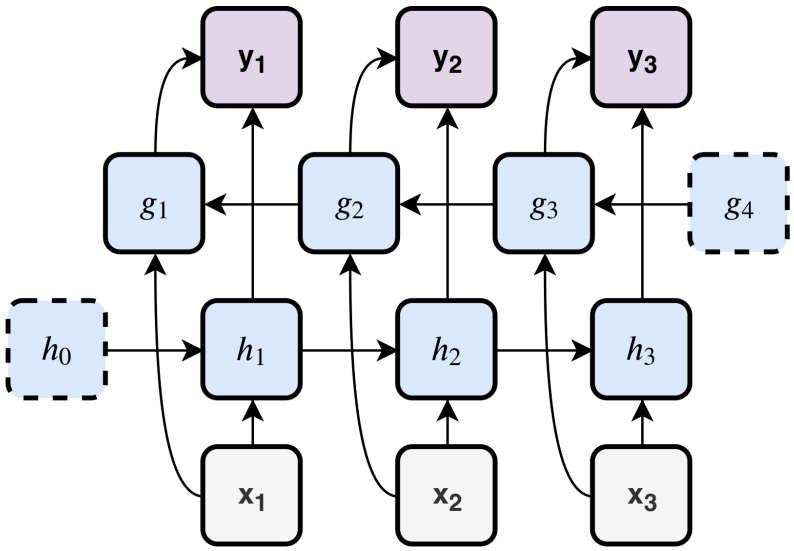

Figure 2.6. Example of a bidirectional recurrent neural network setup, where one RNN computes the input sequence in the original order and the second RNN computes it in the reverse order. h_0 and g_4 are both padding hidden states. The final outputs y_t are computed using the h_t and g_t hidden states.

Because the original recurrent neural network setup is unidirectional, the hidden state computed on the first few elements of a sequence get less information about the rest of the sequence than the last few elements. When only the last computed output is important, it is less of an issue, but for tasks where an output for each input element is necessary, then the unidirectionality of RNNs is an issue. Bidirectional RNNs counter this problem by using a second RNN to compute the original sequence in the reverse order. Figure 2.6 shows an example of a bidirectional setup. The two RNNs in this example could share the same weights or each have their own. The final outputs y_t can be computed in many ways, using both the

h_t and g_t as input. For example, the hidden states can be meaned, max pooled or simply concatenated and passed through a linear layer.

2.5.4. Long short-term memory

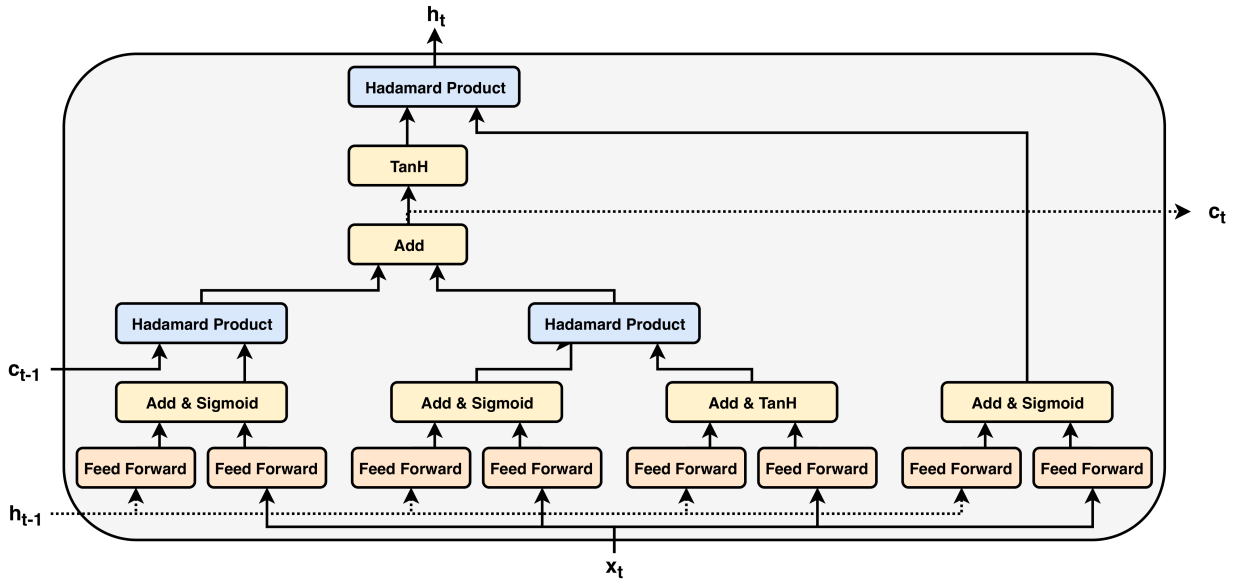

Figure 2.7. Visualization of the computations performed in a LSTM cell. h_{t-1} and h_t are the previous and the current hidden state and c_{t-1} and c_t are the previous and current cell state.

Long short-term memory [Hochreiter and Schmidhuber, 1997] proposes to change the RNN cell for something more complex. First off, it has three "gates", controlled by a sigmoid function :

$$i_t = \sigma(W_{ix}x_t + b_{ix} + W_{ih}h_{t-1} + b_{ih}) \tag{2.5.3}$$

$$f_t = \sigma(W_{fx}x_t + b_{fx} + W_{fh}h_{t-1} + b_{fh}) \tag{2.5.4}$$

$$o_t = \sigma(W_{ox}x_t + b_{ox} + W_{oh}h_{t-1} + b_{oh}) \tag{2.5.5}$$

The LSTM cell also has a cell state, computed with the two gates f_t and i_t :

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{2.5.6}$$

where $\odot$ is the Hadamard product and g_t is computed exactly like the hidden state of the regular RNN cell :

$$g_t = \tanh(W_{gx}x_t + b_{gx} + W_{gh}h_{t-1} + b_{gh}) \tag{2.5.7}$$

Finally, the hidden state of the LSTM is computed with the Hadamard product between the o_t gate and the cell state :

$$h_t = o_t \odot \tanh(c_t) \tag{2.5.8}$$

Figure 2.7 shows the operations performed in a LSTM cell.

Apart from how the hidden state is computed, the LSTM makes no other changes over the regular recurrent neural network. LSTMs usually perform better than a regular RNN [Kiros et al., 2015, Wu et al., 2016b].

2.5.5. Gated Recurrent Unit

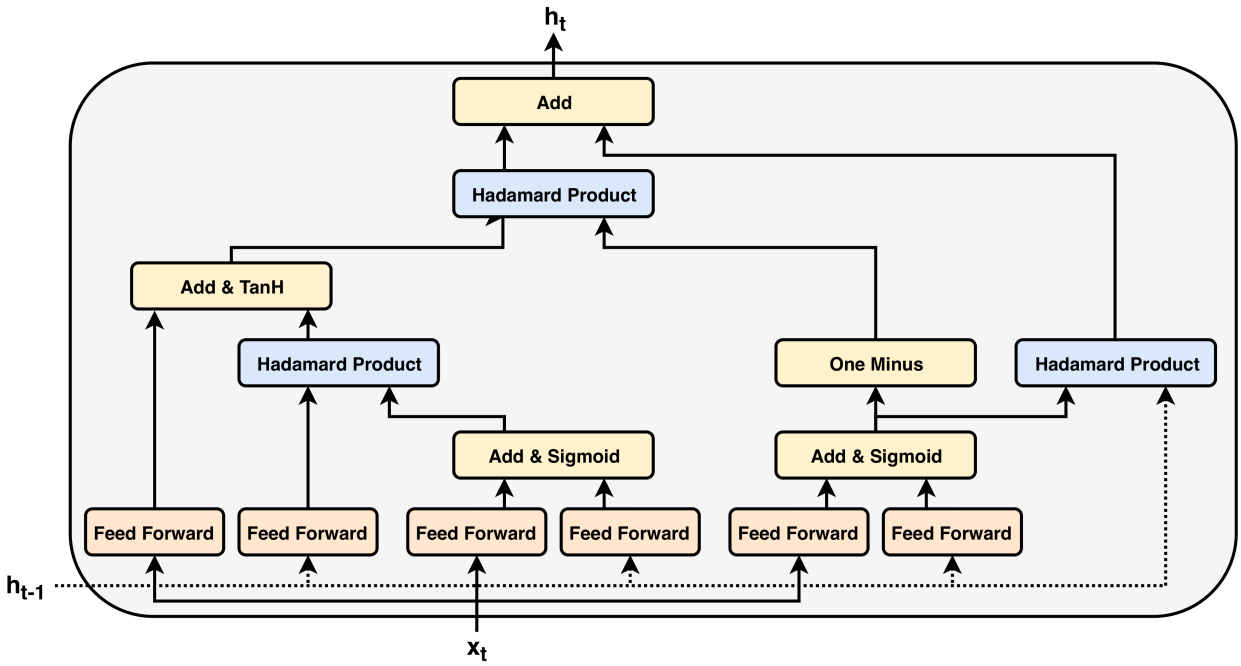

Figure 2.8. Visualization of the operations performed in a GRU cell. The cell computes the hidden state h_t by taking into input x_t and the previously computed hidden state h_{t-1}

The gated recurrent unit aims at simplifying the LSTM cell while being on par with performance. It has only two gates instead of three :

$$r_t = \sigma(W_{rx}x_t + b_{rx} + W_{rh}h_{t-1} + b_{rh}) \tag{2.5.9}$$

$$z_t = \sigma(W_{zx}x_t + b_{zx} + W_{zh}h_{t-1} + b_{zh}) \tag{2.5.10}$$

where σ is the sigmoid activation function. The GRU cell also has no cell state. Like the regular RNN cell, it only computes a hidden state :

$$h_t = (1 - z_t) \odot n_t + z_t \odot h_{t-1} \tag{2.5.11}$$

where $\odot$ is the Hadamard product and n_t is computed by :

$$n_t = \tanh(W_{nx}x_t + b_{nx} + r_t \odot (W_{nh}h_{t-1} + b_{nh})) \tag{2.5.12}$$

Figure 2.8 shows the operations performed in a GRU cell.

As with the LSTM, a GRU network only computes the hidden state differently than a regular recurrent neural network. No other changes are made to the original method. GRU networks have been shown to perform better than regular RNNs and their performance being on par with LSTMs [Chung et al., 2014].

2.6. Attention

Bahdanau et al. [2014] introduced an attention mechanism used in the context of sequence to sequence learning (see section 2.5.1). By allowing the decoder to access information on the original sequence other than the last hidden state computed by the encoder, Bahdanau et al. removed the burden of encoding all of the information of the original sequence into a single vector. This is done by computing context vectors in the decoder. When computing the hidden states z_t with the decoder, along with the previous hidden state z_{t-1} and output y_{t-1}, the decoder takes into input the context vector c_t :

$$z_t = f(x_{t-1}, z_{t-1}, c_t) \tag{2.6.1}$$

where f is the function used to compute a hidden state of the decoder.

The context vector c_t is computed the following way : first an energy function is computed between the previous hidden state z_{t-1} and every hidden state h_i of the encoder :

$$e_{ti} = g(z_{t-1}, h_i) \; \forall i \in \{1, \cdots, T\} \tag{2.6.2}$$

where g is a function of choice, like a linear layer for example. Then, the softmax is computed for each e_{ti} :

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_{j=1}^{T_x} \exp(e_{tj})} \tag{2.6.3}$$

We thus get, for the current z_t, one α_{ti} for each h_i and then the context vector c_t is simply summing all h_i multiplied by their α_{ti} scalar :

$$c_t = \sum_{i=1}^{T_x} \alpha_{ti} h_i \tag{2.6.4}$$

This section describe the original attention mechanism, but multiple others exist, like multiplicative attention [Luong et al., 2015] and the scaled dot-product attention, which is described in section 2.7.1. Attention is used by most state-of-the-art methods in NLP [Vaswani et al., 2017, Devlin et al., 2018, Liu et al., 2019a], with the Transformer architecture being based purely on self-attention (see next section).

2.7. Transformers

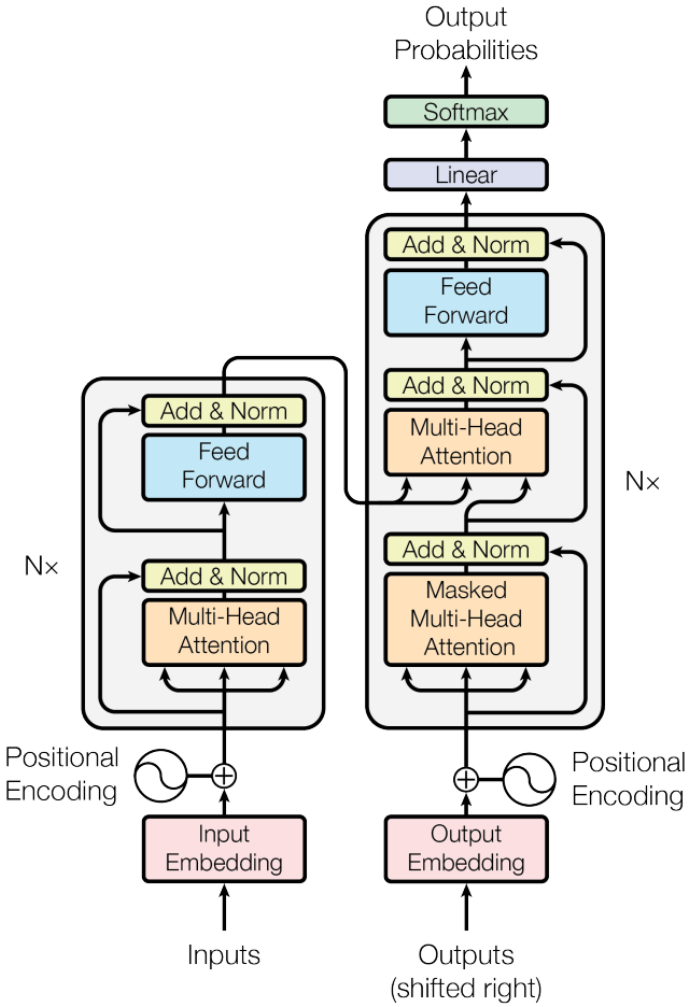

Figure 2.9. The Transformer architecture. Taken from [Vaswani et al., 2017]

Vaswani et al. [2017] introduced a neural network architecture based on a multi-head self-attention mechanism. They presented their architecture as a neural machine translation method. First, an encoder stack applies N Transformer layers, where each layer applies the multi-head attention mechanism, followed by a residual connection, then a feed forward layer and finally another residual connection. The ouput of each encoder layer contains as many embeddings as the original input sequence. When the encoder is done, the decoder stack then decodes one word of the output sequence at a time. Each decoder layer has two multi-head attention sub-layers instead of one. The first one computes attention between the input of the decoder and itself, while the second one computes the attention between the decoder input and the final encoder output. In the neural machine translation setting, the Transformer thus encodes the original sequence into a sequence of the same length that the decoder then uses to decode an output sequence in the target language.

2.7.1. Scaled Dot-Product Attention

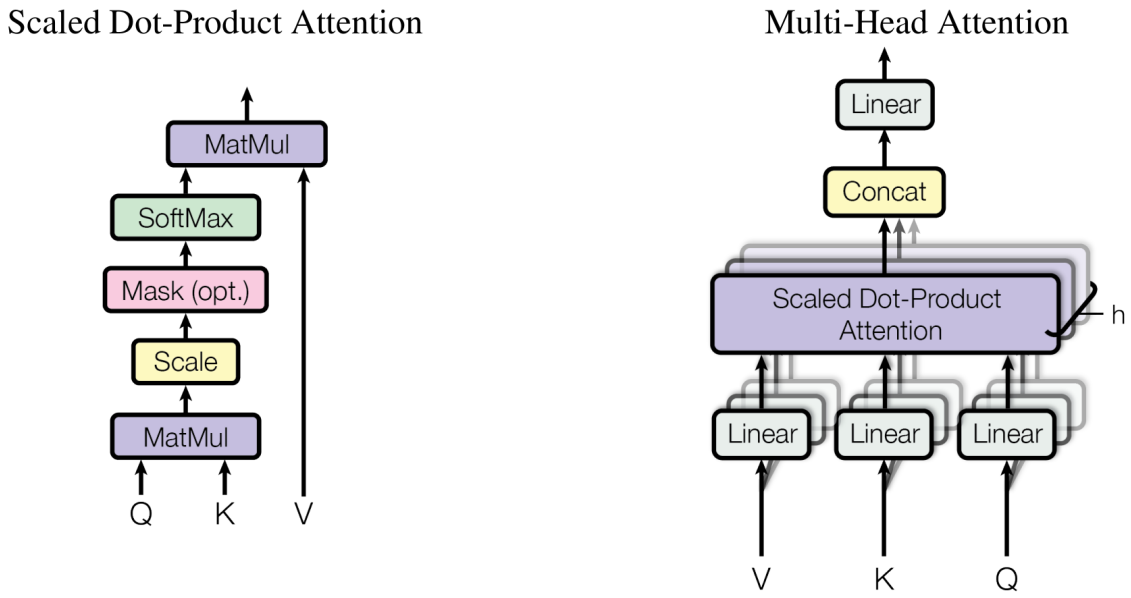

Figure 2.10. Left is the Scaled Dot-Product Attention mechanism and right the Multi-head attention mechanism. Taken from [Vaswani et al., 2017]

The self-attention mechanism is a scaled dot-product attention. For each input embedding, a query, key, and value vector is computed. All queries packed in matrix Q, all keys in

K, and values in V. The attention is then computed as following :

$$Attention(Q, K, V) = softmax\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{2.7.1}$$

where d_k is the dimension of a key vector. The output matrix is then unpacked into as many embeddings as the input, one embedding per row. Figure 2.10 shows the operations performed by the Scaled Dot-Product Attention.

2.7.2. Multi-Head Attention

Multi-head attention is simply multiple computations of attention. Each "head" has a different set of weights to compute Q, K and V. The output of each attention head is concatenated and then fed to a linear layer. Figure 2.10 shows the operations performed in multi-head attention.

2.7.3. Current State-Of-The-Art

Devlin et al. [2018] used the encoder part of the Transformer architecture to achieve state-of-the-art results on multiple natural language processing tasks. They first trained the network on BookCorpus [Zhu et al., 2015] and Wikipedia for two unsupervised tasks: language modeling and predicting, given a sentence pair A and B, if B is the sentence following A or a random sentence. They then individually fine-tuned this pretrained model on 11 natural language understanding tasks, the GLUE benchmark [Wang et al., 2018] and got state-of-the-art results. The method was also used to achieve state-of-the-art results on multiple other NLP tasks.

Very similar to BERT, Liu et al. [2019b] train a Transformer architecture via the same pretraining method as BERT, but then train their model on 9 GLUE tasks, sharing and updating the Transformer weights for all tasks and using task specific layers at the top of the architecture. They call their method MT-DNN. Liu et al. [2019a] improved the results of the MT-DNN network by using an ensemble of MT-DNN networks for each task to train a single of MT-DNN to generalize well on all tasks. This teacher-student method is called knowledge distillation [Hinton et al., 2015]. The MT-DNN method is the current state-of-the-art on the GLUE benchmark [Wang et al., 2018].

Radford et al. [2019] train a Transformer network as a language model on their huge WebText corpus and achieve state-of-the-art results on seven language modelling datasets, without any training on any of them.

Finally, one of the drawbacks of the Transformer network is its heavy computational cost. Ott et al. [2018] propose solutions to this problem. Another issue with the architecture is its fixed size input. To deal with long sequences, Dai et al. [2019] propose to use the Transformer architecture in a recurrent manner, while keeping track of the previous computations via a hidden state, similar to recurrent neural networks.

Chapitre 3

Related Work

In this chapter, we cover related work in representation learning, neural machine translation, quantization and knowledge distillation.

3.1. Representation Learning

Early efforts such as [Hinton and Salakhutdinov, 2006] have shown autoencoders to effectively yield compressed input representations. Pollack [1990] was the first to propose using autoencoders recursively. Such models have been shown to be useful for a multitude of tasks. Luong et al. [2013] use recursive neural networks and neural language models to better represent rare words via morphemes. Socher et al. [2011a] use recursive autoencoders for paraphrase detection, learning sentence embeddings [Socher et al., 2010] and syntactic parsing. Socher et al. [2011b] also use a recursive autoencoder to build a tree structure based on error reconstruction. Additionally, Socher et al. [2012] use a matrix-vector RNN to learn semantic relationships present in natural language and show good performance on such task as well as sentiment classification. Then, Socher et al. [2013] introduced the Recursive Neural Tensor Network, trained on a their proposed Sentiment Treebank corpus to better deal with negating sub-sequences for better sentiment classification. Recently, Kokkinos and Potamianos [2017] proposed Structural Attention to build syntactic trees and improve even further performance on SST. Parse trees do alleviate the burden of learning the syntactic structure of text, but these methods limit the number of generated embeddings to the number of nodes in the parse tree. The proposed method in this work does not have such a restriction as all possible syntactic tree can be simultaneously represented by the architecture.

Convolutional Neural Networks [LeCun et al., 1989] have been used in natural language processing as well. Convolutions work well for extracting low and high level text features and building sequence representations. Lai et al. [2015] proposed to use CNNs recurrently and show good performance on various language tasks. Zhang et al. [2015], Dos Santos and Gatti de Bayser [2014] both train CNNs on character level for sentiment analysis, while Johnson and Zhang [2014] work on word level. Kalchbrenner et al. [2014] propose a Dynamic Convolutional Neural Network for semantic modelling of sentences and apply their model to sentiment prediction. The model proposed in this work is very similar to 1D CNNs. Though, it uses a multilayer perceptron in parallel instead of a kernel to extract meaningful information out of the layer's input.

Much progress has been made in recent years in the field of general purpose sentence embeddings. Fixed length representations of sentence wide context are learned with the objective of serving for a wide range of downstream tasks. Conneau et al. [2017a] trained a bidirectional LSTM on the AllNLI natural language inference corpus [Bowman et al., 2015, Williams et al., 2017] producing embeddings that generalized well on the SentEval [Conneau and Kiela, 2018] benchmark. Following this trend, Subramanian et al. [2018] trained a GRU [Cho et al., 2014a] on Skip-thought vectors [Kiros et al., 2015], neural machine translation, parsing and natural language inference to get even better downstream task results. More recently, Devlin et al. [2018], Liu et al. [2019b,a] use Transformers [Vaswani et al., 2017] to produce sentence wide context embeddings for each input token and get state-of-the-art results on multiple natural language processing tasks. Dai et al. [2019] improve the Transformer method by recursively applying it to fixed length segments of text while using a hidden state to model long dependencies. One downside to these sentence embedding generation methods is that the context is always sequence wide. The proposed model in this work computes a sentence embedding as well as an embedding for all possible sub-sentences of the sequence with sub-sentence wide context only. All embeddings generated throughout our architecture are constructed the same way and thus share the same properties.

3.2. Neural Machine Translation

Neural machine translation methods have achieved impressive results lately [Wu et al., 2016b, Gehring et al., 2017, Ott et al., 2018]. This end-to-end approach to machine translation

was first proposed by Kalchbrenner and Blunsom [2013], Sutskever et al. [2014], Cho et al. [2014], with Bahdanau et al. [2014] introducing an attention mechanism soon after. Multiple improvements to their approach have been proposed, such as multiplicative attention [Luong et al., 2015] and more recently multi-head self-attention [Vaswani et al., 2017]. The latter's novel Transformer architecture achieved state-of-the-art results on the WMT 2014 English-French and WMT 2014 English-German corpus. Inspiring a new wave of work, state-of-the-art of numerous natural language processing tasks reached new heights [Devlin et al., 2018, Liu et al., 2019b].

3.3. Quantization

Quantization is the process of lowering numerical precision, which allows representation of numerical values with fewer bits. The focus of early work was simpler hardware deployment [Fiesler et al., 1993, Tang and Kwan, 1993, Marchesi et al., 1993]. With ever larger neural networks, quantization has of lately served as a compression method [Gong et al., 2014, Han et al., 2015, Hubara et al., 2016, Polino et al., 2018]. Multiple approaches have been explored, such as binary [Courbariaux et al., 2016], ternary [Lin et al., 2015, Li et al., 2016], learned [Zhang et al., 2018] and uniform affine [Jacob et al., 2018a] quantization and in combination with pruning [Han et al., 2015]. The practice has been extended to a multitude different architectures. Rastegari et al. [2016] apply binary quantization to the filters and convolutional layers of CNNs [LeCun et al., 1989], while Wu et al. [2015] quantize both kernels and fully connected layers. Similarly, Zhou et al. [2016] use low bitwidth weights in convolution kernels. Ott et al. [2016] explore binary and ternary quantization of RNNs [Jordan, 1990] and introduce their novel exponential quantization method. Wang et al. [2018] propose the use of different quantization methods per RNN component. Hubara et al. [2016] apply quantization to both the weights and the activations of RNNs and LSTMs [Hochreiter and Schmidhuber, 1997]. He et al. [2016] propose modifications to the gates and interlinks of quantized LSTM and GRU cells [Cho et al., 2014b]. Wu et al. [2016b] reduce the quantization errors of their deep LSTM network by restraining the values of the residual connections and cell states. Ott et al. [2018] train Transformers with mixed-precision and get state-of-the-art on the WMT 14 English-French corpus. Cheong and Daniel [2019] apply k-means quantization to the Transformer network, as well as iterative magnitude pruning. Fan [2019]

also use binary quantization on the Transformer, as well as ranged base linear quantization. Tierno [2019] quantize both the weights and the inputs of Transformer layers.

3.4. Knowledge Distillation

Another method which can be used to compress neural networks is called knowledge distillation [Hinton et al., 2015]. The approach can be used to distill the knowledge of a neural network to a smaller one [Kim and Rush, 2016, Tucker et al., 2016, Mishra and Marr, 2017, Reddy et al., 2017]. The technique has also been combined with quantization [Wu et al., 2016a, Polino et al., 2018]. Related to Transformers, Liu et al. [2019a] use an ensemble of these, each trained on a specific task and then train a single Transformer on all tasks. Knowledge distillation between different types of neural networks has also been tried. Senellart et al. [2018] use a Transformer as a teacher and a recurrent neural network as the student. Likewise, Chia et al. [2018] distill from a Transformer to a CNN.

Chapter 4

Towards Lossless Encoding of Sentences

Gabriele Prato Mathieu Duchesneau Sarath Chandar Alain Tapp

Accepted at ACL 2019.

Contribution: Mathieu Duchesneau and Prof. Alain Tapp initially came up with a first version of this project. Then, me, Prof. Alain Tapp and Prof. Sarath Chandar further developed the idea. I wrote all the code and did all experiments found in the paper. I also wrote the paper with the help of Prof. Sarath Chandar.

Affiliation:

— Gabriele Prato, Mila, Université de Montréal

— Mathieu Duchesneau, Mila, Université de Montréal

— Sarath Chandar, Mila, Université de Montréal

— Alain Tapp, Mila, Université de Montréal

4.1. Recursive Autoencoder

We introduce our recursive autoencoding approach in this section. First we define our model's architecture and how each encoding and decoding recursion is performed. We then describe how the model keeps track of the recursion steps, followed by a description of how the input is represented. We also explain the advantages of using the mean squared error loss for our method. Finally, we dive into the implementation details.

4.1.1. Model Architecture

Our model is a recursive auto-encoder. Figure 4.1 shows an example of our architecture for a sequence of length three.

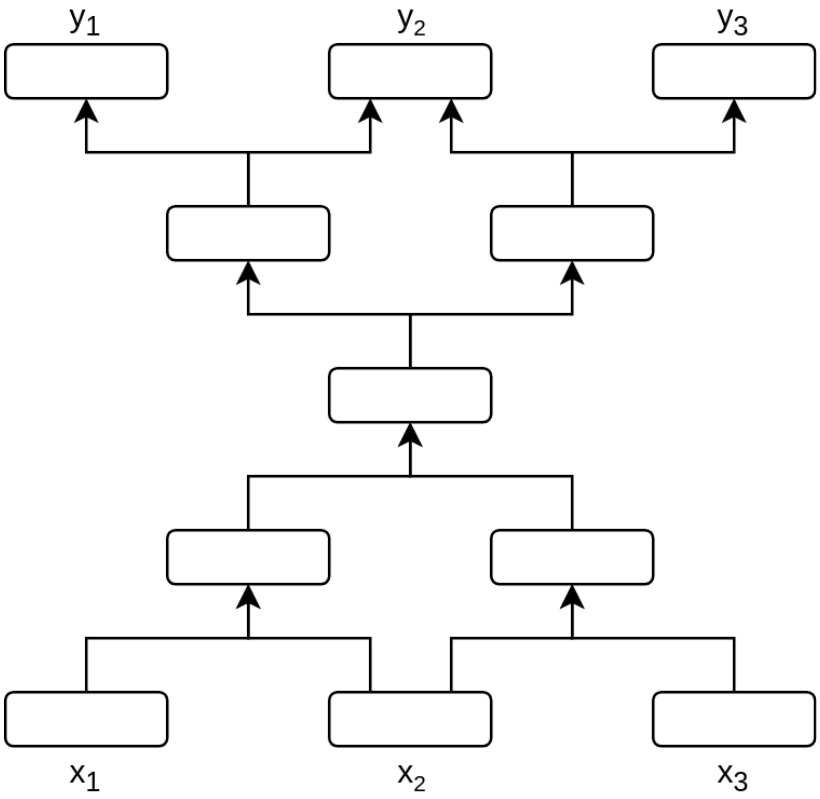

Figure 4.1. Example of our recursive autoencoder with an input sequence of length three. The encoder recursively takes two embeddings and outputs one until a single one is left and the decoder takes one embedding and outputs two until there are as many as in the original sequence.

The encoder takes an input sequence $\{x_1, \cdots, x_n\}$, where n is the sequence length of the layer's input, and outputs a sequence $\{y_1, \cdots, y_{n-1}\}$. The same $\{y_1, \cdots, y_{n-1}\}$ is then used as input for the next recursion until the output sequence contains only a single element y_1, the sentence embedding. The recursion performs the following operation:

$$y_i = \text{MLP}_{enc}\left([x_i; x_{i+1}]\right) \forall i \in \{1, \cdots, n-1\} \tag{4.1.1}$$

where MLP_{enc} is a shared multilayer perceptron and $[x_i; x_{i+1}]$ is the concatenation of the embeddings x_i and x_{i+1}. MLP_{enc} is shared throughout all of the encoding recursion steps.

For decoding, it is the inverse procedure of recursively transforming an input sequence $\{x_1, \cdots, x_n\}$ into an output sequence $\{y_1, \cdots, y_{n+1}\}$:

$$[y_i; y'_{i+1}] = \text{MLP}_{dec}\left(x_i\right) \forall i \in \{1, \cdots, n\} \tag{4.1.2}$$

where MLP_{dec} is the shared multilayer perceptron used by all decoding recursive steps and $[y_i; y'_{i+1}]$ is an embedding twice the size of x_i, which we then split into two embeddings y_i and y'_{i+1}, each of the same size as x_i. Since we obtain two embeddings y_i and y'_{i+1} for each x_i, we will have the following embeddings: y_1, $\{y_2, \cdots, y_n\}$, $\{y'_2, \cdots, y'_n\}$ and y'_{n+1}. We merge the overlapping sets by computing the mean:

$$y_i = \frac{y_i + y'_i}{2} \ \forall i \in \{2, \cdots, n\} \tag{4.1.3}$$

and set $y_{n+1} = y'_{n+1}$. We now have a single set of embeddings $\{y_1, \cdots, y_{n+1}\}$. Both *max* and *mean* functions gave similar results, hence we stick with *mean* throughout all experiments. The output embeddings are then used as input for the next decoding recursion until we get as many elements as the original input sequence.

4.1.2. Step Encoding

To help the recursive autoencoder keep track of the number of recursive steps which were applied to an embedding, we concatenate to the input of MLP_{enc} the number of the current recursive step as a scalar, starting from 1 for the first recursion, as well as a one-hot of that scalar with custom bucket sizes: {1, 2, 3-4, 5-7, ...}. All buckets after 5-7 are also of size 3. We found this combination of both scalar and one-hot to give best results. When decoding, we also concatenate to the input of MLP_{dec} this scalar and one-hot, but instead of increasing our recursive step count, we subtract one to it after each recursive decoding step.

4.1.3. Input Representation

We use uncased GloVe embeddings [Pennington et al., 2014] of size 300 to represent the initial input sequence words, which are then passed through a learned resizing multilayer perceptron (MLP_{in}) before given as input to the encoder. The output of the decoder is also passed through a different learned resizing multilayer perceptron (MLP_{out}) to get back to the GloVe embedding size. We use a vocabulary of 337k words throughout all tasks.

4.1.4. Mean Squared Error

To compute the loss between input GloVe embeddings and the output embeddings, we use the mean squared error (MSE) loss. Obtaining an MSE of 0 would mean our method is lossless, which would not necessarily be the case with the cross entropy loss. MSE also allows us to work with a vocabulary far larger than what is usually the case, as the common classification layer plus cross entropy loss setup tends to have issues with large vocabularies.

4.1.5. Implementation Details

The two embeddings given as input to MLP_{enc} are each of size d_{emb}, as is also its output embedding. Same for MLP_{dec}, the input embedding is of size d_{emb} and the two output embeddings are each of size d_{emb}. Both multilayer perceptrons have one hidden layer of size $\frac{2}{3}d_{emb}$, halfway between the input and output size. We apply LayerNorm [Lei Ba et al., 2016] on the output of each layers of the MLPs, followed by a ReLU activation. The input and output resizing modules MLP_{in} and MLP_{out} also have one hidden layer halfway the size of their input and output. They also use ReLU activations, except for MLP_{out}'s last layer. No LayerNorm is used in these resizing components. We test four different d_{emb} embedding sizes in section 4.2.1.

4.2. Experiments

In this section, we first present the autoencoding results. Then we present the results on sentiment analysis using our sentence encoding on the Stanford Sentiment Treebank dataset [Socher et al., 2013].

4.2.1. Autoencoding

As a first experiment, we tested our model on the autoencoding task. Training was done on the BookCorpus [Zhu et al., 2015] dataset, comprising eleven thousand books and almost one billion words. At test time, we measured accuracy by computing the MSE distance between an output embedding and the entire vocabulary. We count an output embedding as "correct" if the closest embedding out of all the vocabulary of size 337k is its corresponding input embedding.

For the autoencoder, we tried four embedding sizes: 300, 512, 1024 and 2048. In all cases, models are given GloVe embeddings of size 300 as input. They also all output embeddings of size 300. Reconstruction accuracy is shown for different sequence lengths in Figure 4.2. With an embedding size of 2048, the model is able to reproduce near perfectly sequences of up to 40 tokens. Longer sentences aren't able to do better and have on average 39 correct tokens. This results in model accuracy linearly going down after a certain threshold, as can be seen in Figure 4.2.

To demonstrate how good our model is at reconstruction, we trained a stacked LSTM on the same autoencoding task. Figure 4.3 shows performance of LSTM models for embedding sizes 300, 512 and 1024. All LSTMs have two encoder and two decoder layers. The 1024 variant seems to have reached a saturation point, as it performs similarly to the 512 version. All RAEs and LSTMs were trained for 20 epochs and models with same embedding size have the same capacity. Figure 4.4 shows a better side by side comparison of the RAE and the LSTM for embedding sizes 512 and 1024. Table 4.1 shows the MSE loss of all models on the dev set after 20 epochs. The LSTM with an embedding size of 1024 only slightly achieves lower MSE than the RAE with embedding size 300.

When the output and input embeddings don't match as nearest, they are usually close. Figure 4.5 shows the gain in accuracy for the 1024 and 2048 variants when considering an output embedding as correct if the input embedding is in the five closest to the output, out of all the vocabulary. For the 1024 version, we see on average an increase in accuracy of 2.7%, while for the 2048 variant, the gain only starts to get noticeable for sequences longer than 30, with an overall average increase of 1.4%.

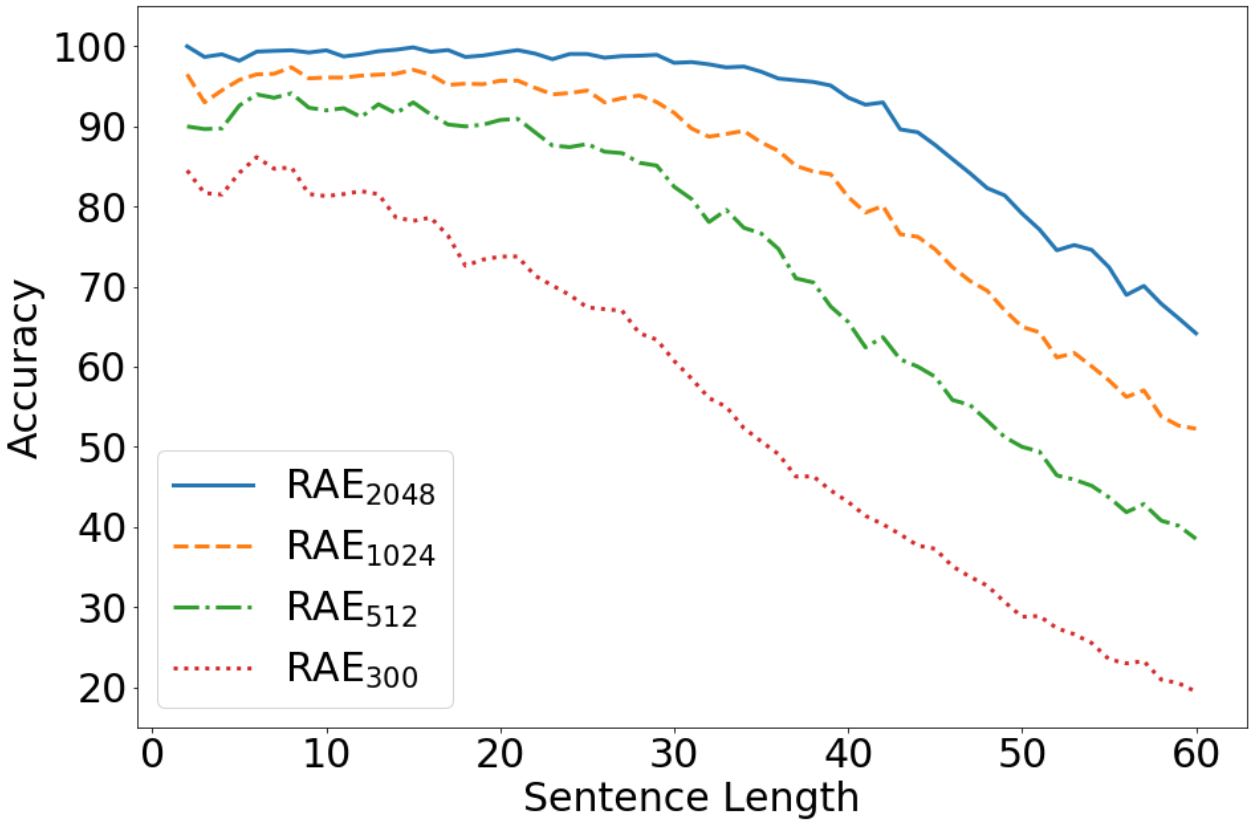

Figure 4.2. Accuracy comparison of different embedding sizes (300, 512, 1024 and 2048) for different sequence lengths. Left is our recursive autoencoder and right a stacked LSTM. An output embedding is counted as correct if the closest embedding out of all the vocabulary is its corresponding input embedding.

4.2.2. Sentiment Analysis

With strong autoencoding performance, one would think that features get deeply encoded into the representation, making it difficult to easily extract them back, which is crucial for a great number of tasks. To this end, we test our architecture on the sentiment analysis task.

The Stanford Sentiment Treebank [Socher et al., 2013] is a sentiment classification task where each sample in the dataset is a sentence with its corresponding sentiment tree. Each node in the tree is human annotated, with the leaves representing the sentiment of the words, all the way up to the root node, representing the whole sequence. Comparison is usually done on a binary or five label classification task, ranging from negative to positive. Most models are usually by design only able to classify the root node, while our architecture allows

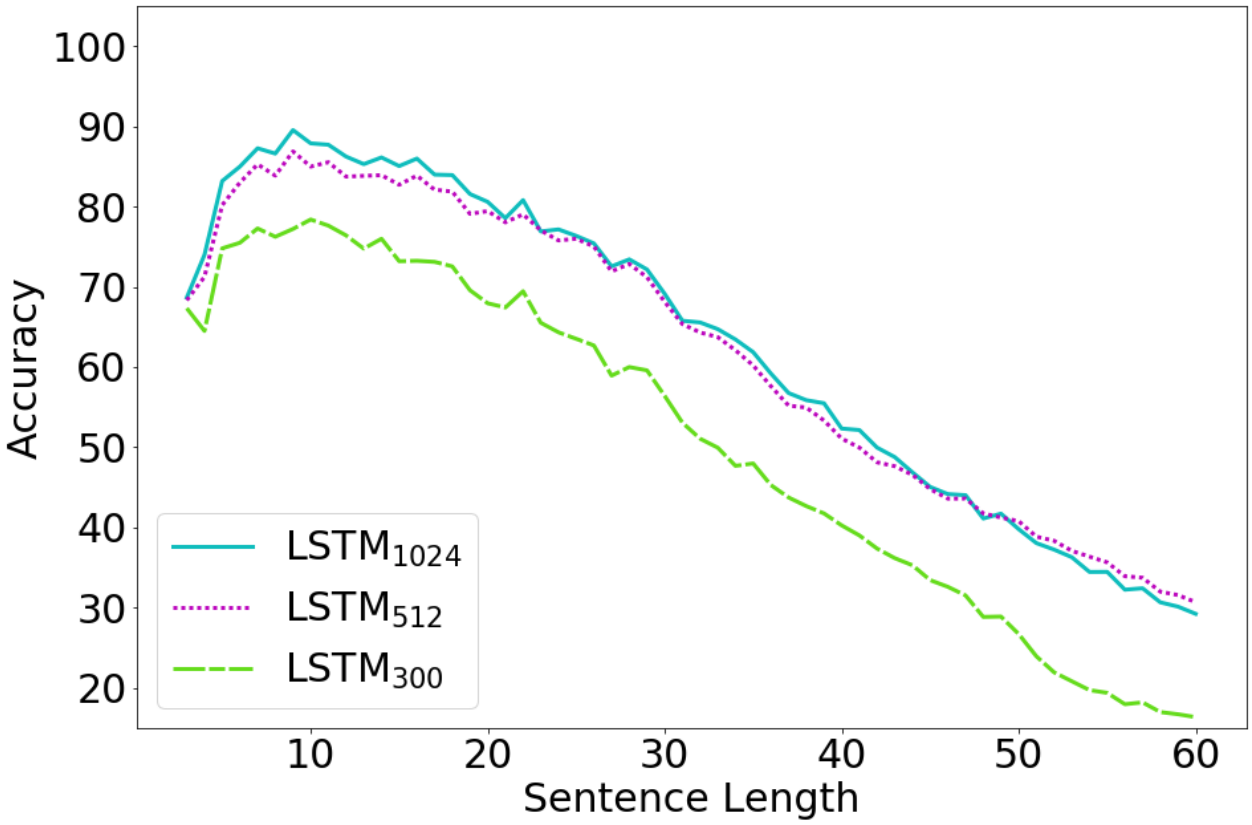

Figure 4.3. Accuracy comparison of different embedding sizes (300, 512, 1024 and 2048) for different sequence lengths. Left is our recursive autoencoder and right a stacked LSTM. An output embedding is counted as correct if the closest embedding out of all the vocabulary is its corresponding input embedding.

classification of every node in the tree. We use a linear layer on top of each embedding in the encoder to classify sentiment.

We present in Table 4.2 results for fine-grained sentiment analysis on all nodes as well as comparison with recent state-of-the-art methods on binary sentiment classification of the root node. For the five class sentiment task, we compare our model with the original Sentiment Treebank results and beat all the models. In order to compare our approach with state-of-the-art methods, we also trained our model on the binary classification task with sole classification of the root node. Other presented models are GenSen [Subramanian et al., 2018] and $\text{BERT}_{\text{BASE}}$ [Devlin et al., 2018]. Both these recent methods perform extremely well on multiple natural language processing tasks. We set the RAE embedding size d_{emb} to 1024. Larger embedding sizes did not improve the accuracy of our model for this task.

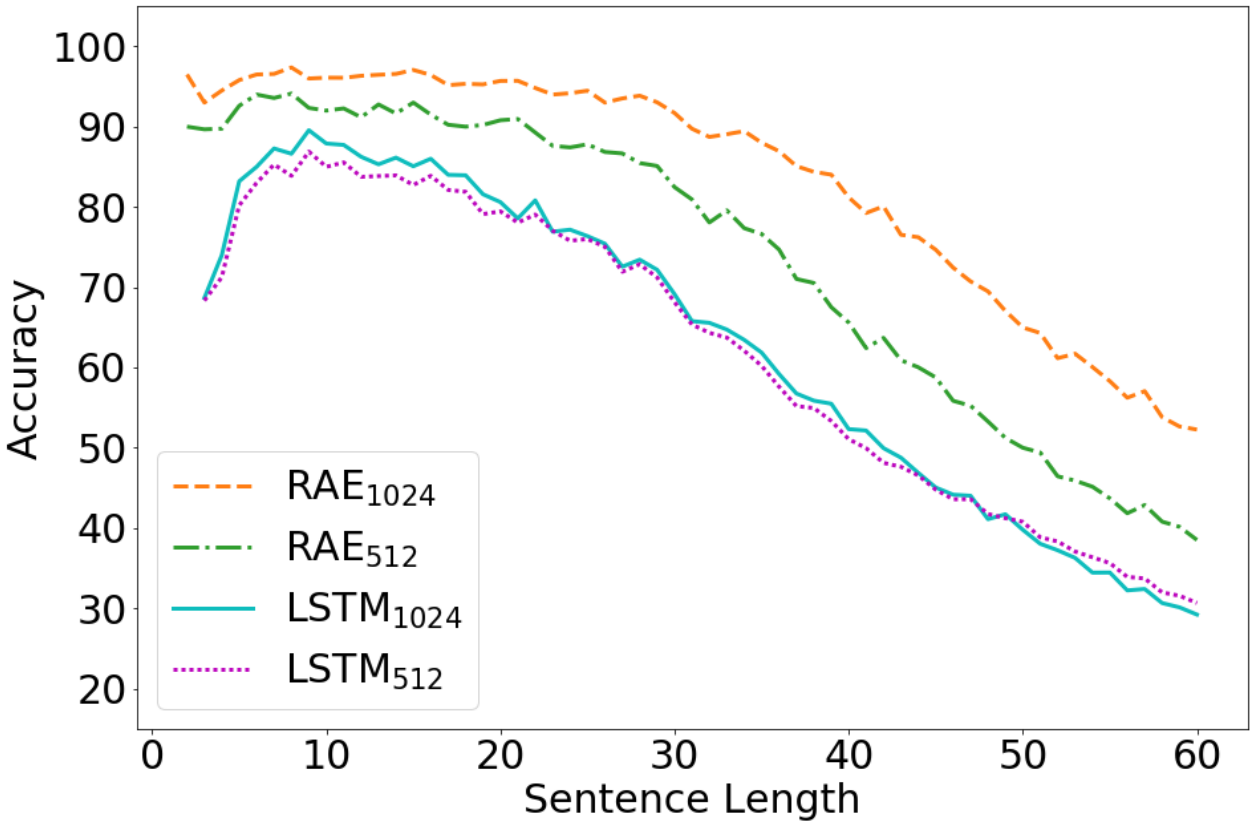

Figure 4.4. Accuracy comparison of our RAE model versus a stacked LSTM for embedding sizes 512 and 1024. Models of same embedding size have the same capacity.

In this setting, the RAE has 11M parameters, while the models we compare with, GenSen and $\text{BERT}_{\text{BASE}}$, have respectively 100M and 110M parameters. Both our model and GenSen fail to beat the RNTN model for the SST-2 task. We see an improvement in accuracy when combining both methods' embeddings, surpassing every model in the SST paper, while being close to $\text{BERT}_{\text{BASE}}$'s performance.

Training solely on sentiment classification had same performance as jointly training on the autoencoding task, as the latter had no impact on the sentiment analysis performance. Joint training though had a small impact on reconstruction.

Model	d_{emb}	MSE (dev)
LSTM	300	0.0274
	512	0.0231
	1024	0.0191
RAE	300	0.0208
	512	0.0124
	1024	0.0075
	2048	**0.0019**

Table 4.1. Mean squared error loss of stacked LSTMs and our RAE model for different embedding sizes. All models are trained on the autoencoding task for 20 epochs and models of same embedding size have the same capacity. MSE is computed on the BookCorpus dev set [Zhu et al., 2015], between the input GloVe embeddings [Pennington et al., 2014] and output embeddings.

Model	SST-5 (All)	SST-2 (Root)
NB	67.2	81.8
SVM	64.3	79.4
BiNB	71.0	83.1
VecAvg	73.3	80.1
RNN	79.0	82.4
MV-RNN	78.7	82.9
RNTN	80.7	85.4
RAE	**81.07**	83
GenSen	-	84.5
RAE + GenSen	-	86.43
$\text{BERT}_{\text{BASE}}$	-	**93.5**

Table 4.2. SST-5 and SST-2 performance on all and root nodes respectively. Model results in the first section are from the Stanford Treebank paper [Socher et al., 2013]. GenSen and $\text{BERT}_{\text{BASE}}$ results are from [Subramanian et al., 2018] and [Devlin et al., 2018] respectively.

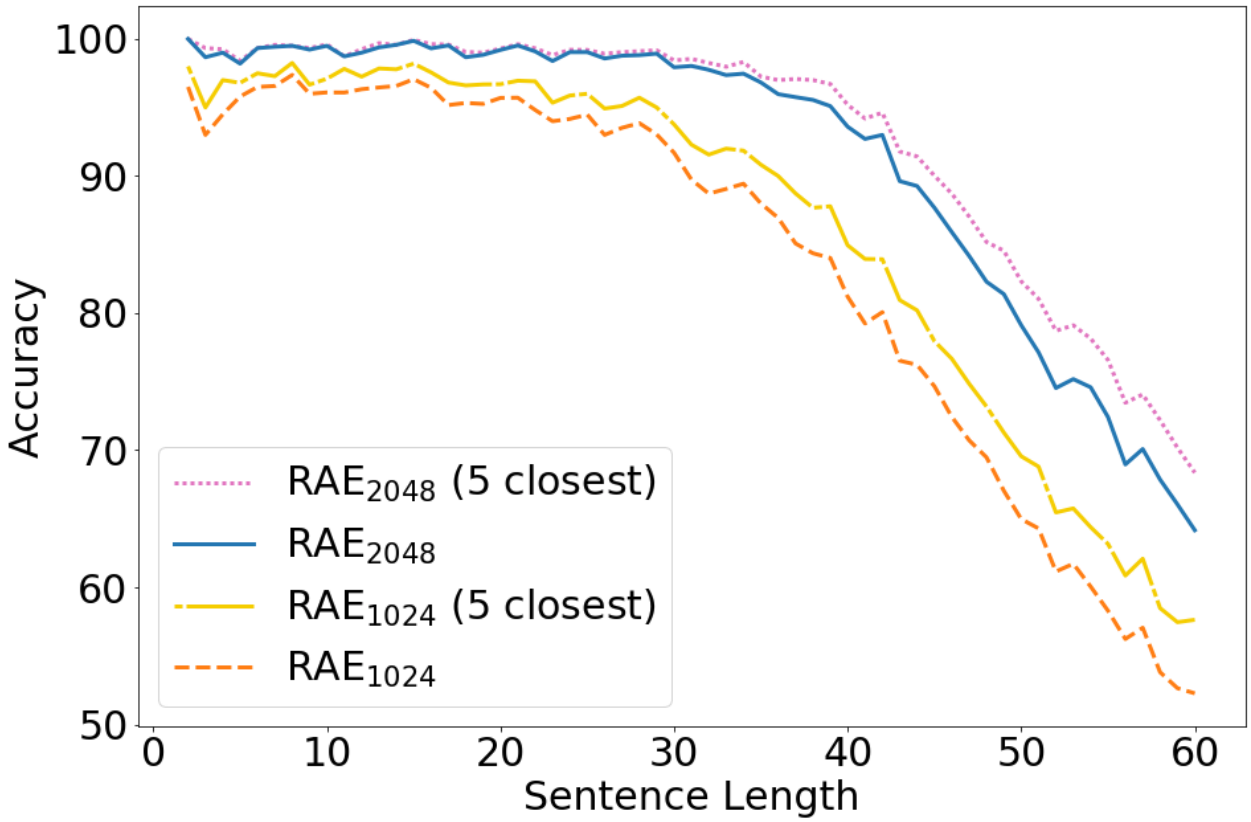

Figure 4.5. Difference in accuracy when counting an output embedding as correct if the corresponding input embedding is in the five closest versus the closest. Comparison is done on our RAE model with embedding sizes 1024 and 2048.

Chapitre 5

Compressing Tranformers for Neural Machine Translation via Knowledge Distillation and Quantization

Gabriele Prato Ella Charlaix Mehdi Rezagholizadeh
Eyyüb Sari Vahid Partovi Nia

In submission for EMNLP 2019.

This work was done while interning at Huawei Noah's Ark Lab, Montreal Research Centre.

Contribution : My supervisor Mehdi Rezagholizadeh initially came up with the idea. Everyone then contributed to further develop it. Me, Ella Charlaix and Eyyüb Sari wrote all the code and did all experiments found in the paper. Me and Ella Charlaix wrote the paper, everyone else provided feedback.

Affiliation :
— Gabriele Prato, Mila, Université de Montréal
— Ella Charlaix, Huawei Noah's Ark Lab, Montreal Research Centre
— Mehdi Rezagholizadeh, Huawei Noah's Ark Lab, Montreal Research Centre
— Eyyüb Sari, Huawei Noah's Ark Lab, Montreal Research Centre
— Vahid Partovi Nia, Huawei Noah's Ark Lab, Montreal Research Centre

5.1. Compressing Transformers

First, we detail the quantization method we apply to the Transformer. We then define how we use knowledge distillation in our setting. Finally, we describe how both methods are combined for greater compression of the Transformer.

5.1.1. Weight Quantization

Weight Quantization is a compression method which consists in reducing the precision of a network's weights by representing their values with fewer bits. By quantizing the weights to k-bit, the space required to store the model reduces by a factor of $\frac{32}{k}$.

The uniform quantization procedure applied to our model is based on [Jacob et al., 2018b]. During the training phase, the weight values are mapped to the closest quantization level using a scaling factor s computed as :

$$s = \frac{b - a}{2^k - 1} \tag{5.1.1}$$

with a and b respectively the minimum and maximum values of the weight matrix being quantized and 2^k the number of quantization levels associated with the k-bit quantization. However, this quantization is only simulated during the training phase as the values are mapped back to the original domain after the precision loss. This fake quantization process is performed during the forward pass as follows :

$$w_q = \left\lfloor \frac{w - a}{s} \right\rceil s + a \tag{5.1.2}$$

with w a real-valued weight, w_q its simulated k-bit version and $\lfloor \cdot \rceil$ the rounding to the nearest integer operator. We applied the straight-through estimator (STE), introduced by Hinton [2012] in his lectures, to approximate the gradient of the quantization function. Our quantization procedure adds, for each quantized layer, two additional parameters s and a that are both stored in floating-point. These two parameters are however negligible in comparison with the resulting compression.

The Transformer's weights we quantize correspond to each Multi-Head Attention and Position-wise Feed-Forward layers as well as the embedding layer. We did not quantize the layers' biases, as they account for a small fraction of the number of parameters while being added to many output activations. Likewise, we did not quantize the weights of the layer norms.

5.1.2. Knowledge Distillation

A simple method for compressing neural networks is to train smaller models. The knowledge distillation approach by Hinton et al. [2015] allows us to do so while mitigating the loss in performance. This method incorporates the soft-labels generated by a teacher model in the student's loss function, jointly with the true labels. More specifically, we first train our teacher network as the *base* Transformer configuration specified by Vaswani et al. [2017]. We then train our student network, a smaller Transformer model which we detail in section 5.2.1, with the following loss function L :

$$L = w_s H\left(\frac{z}{T}, y'\right) + w_t H(z, y) \tag{5.1.3}$$

where H is the cross entropy loss, z the logits computed by the student and y the true labels. The weights w_s and w_t are used to control the ratio between the soft-label and true-label loss. The temperature T is the same as the one used to compute the teacher's soft targets y' :

$$y' = softmax\left(\frac{z'}{T}\right) \tag{5.1.4}$$

where z' are the logits computed by the teacher network. For further details, we refer the reader to [Hinton et al., 2015].

5.1.3. Quantization & Distillation

As a guideline, we follow the Quantized Distillation approach proposed by Polino et al. [2018], although we do not use bucketing in our quantization method. No modifications need to be made to the knowledge distillation nor the quantization method for both methods to be used jointly. Gradients are computed for the quantized weights, with respect to the distillation loss, and gradient step is applied to the full precision weights. The teacher network is not quantized, only the student.

5.2. Experiments

We first detail the Transformer variants we experimented with. We then go over the training procedure. Finally we discuss the results on the machine translation task.

	d_{model}	d_{ff}	h	PPL (dev)	BLEU (test)	compression
base [Vaswani et al., 2017]	512	2048	8	-	38.1	1x
base				**4.43**	38.08	
base [8-bit Quantized]				4.50	**38.13**	4x
medium	256	1024	4	5.62	34.24	3.15x
medium [Knowledge Distilled]				5.57	35.30	
medium [Knowledge Distilled] [8-bit Quantized]				5.78	35.57	12.59x
small	128	512	4	8.02	29.81	7.53x
small [Knowledge Distilled]				7.78	30.21	
small [Knowledge Distilled] [8-bit Quantized]				8.02	30.17	**30.11x**

Table 5.1. Results of the different Transformer [Vaswani et al., 2017] variants on the WMT 2014 English-French corpus. Perplexity is per token, computed on the development set[2]. BLEU is measured with `multi-bleu.pl`[1] on `newstest2014`[2].

5.2.1. Transformer Variants

We trained three variants of the Transformer network, which we will refer to as base, medium and small. The base model has the exact same configuration as the one in [Vaswani et al., 2017]. The two other variants are exactly the same as base, except for the following : medium has $d_{\text{model}} = 256$ and $d_{\text{ff}} = 1024$ and small has $d_{\text{model}} = 128$ and $d_{\text{ff}} = 512$. Also, both medium and small only have 4 attention heads. For implementation details, we refer the reader to the Transformer paper [Vaswani et al., 2017].

For the base model, we tested with and without quantization, while for medium and small we tried knowledge distillation, knowledge distillation in combination with quantization and also a regular version of each model. In all cases, we use 8-bit quantization. Only the weights are quantized, as quantizing both the activations and weights gave much worse results. For knowledge distillation, we use a base model trained for 10 epochs as the teacher. We found a temperature $T = 1$ to give best results.

5.2.2. English to French Translation

All models are trained on the WMT 2014 English-French corpus, containing ∼36M sentence pairs. We use byte-pair encoding [Sennrich et al., 2016] for tokenization and a shared source-target vocabulary of 32000 case-sensitive tokens. Each training batch was composed of about 40000 tokens, with about equal length sequences. The base models were trained for 5 epochs, medium for 4 and small for 3.

5.2.3. Results

We present all of our results in Table 5.1. All quantized or knowledge distilled models are trained from scratch, no pre-training was done. Perplexity is per token, computed on the development set and BLEU measured with `multi-bleu.pl` [1] on the `newstest2014` [2] test set. We used beam search with a beam size of 4 and a length penalty of 0.6, as in Vaswani et al. [2017]. We did not average checkpoints. The reported compression rate is the difference between the number of weights × bit precision of a model with the *base* model.

Our 8-bit quantized base model has no drop in BLEU compared to the baseline, while being compressed by a factor of 4. With the medium model, BLEU drops by 3.84. Knowledge distillation rises the medium's BLEU score by 1.06 point. Quantizing the knowledge distilled medium model increases compression from 3.15x to 12.59x, while scoring 1.33 more BLEU than our medium baseline. The small model goes down to 29.81 BLEU, with a compression ratio of 7.53. Again, knowledge distillation slightly helps. Adding quantization, we get no loss in BLEU compared to the small baseline, while boosting compression up to 30x. This is only 3% the number of bits the base transformer uses, yet still gives decent translations (see Appendix A).

1. `https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl`

2. `https://www.statmt.org/wmt14/translation-task.html`

Chapitre 6

Conclusion

In the first paper, we introduced a recursive autoencoder method for generating sentence and sub-sentence representations. Decoding from a single embedding and working with a 337k vocabulary, we manage to get near perfect reconstruction for sequences of up to 40 word tokens and very good reconstruction for longer sequences. Capitalizing on our model's architecture, we showed our method to perform well on sentiment analysis and more precisely its advantage when classifying sentiment trees.

In the second paper, we applied knowledge distillation and quantization to the Transformer network. To the best of our knowledge, we are the first to do so, as well as the first work showing Transformers can be quantized without any drop in BLEU. While compressing the network by a factor of 4, we gained 0.05 BLEU over the baseline on WMT 2014 English-French. We gained further compression by training smaller variants of the Transformer architecture via knowledge distillation and achieved 12.59x while losing only 2.51 BLEU.

6.1. Future Work

Continuing in the direction of training our model on different NLP tasks, we would like our representations to generalize well on downstream tasks while maintaining their reconstruction property. We would also like to further explore the usage of sub-sentence representations in natural language processing. Finally, we would like to learn our sentence embeddings' latent space, similarly to Subramanian et al. [2018]'s method, so as to leverage our autoencoder's strong reconstruction ability and generate very long sequences of text.

Secondly, we plan on exploring lower bit-precision quantization methods for the Transformer architecture. We would also like to explore other compression methods, such as neural network pruning. Finally, we would like to improve the benefits of distillation by leveraging the knowledge of better teachers, such as an ensemble of Transformers and larger networks.

Finally, instead of focusing on learning good text representations, a broader work direction would be to learn good representations of the world. The purpose for such is much more general. Good world representations can be useful in a wide range of fields, such as natural languages, reinforcement learning, planning, world modeling and more general machine learning research.

Bibliographie

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv e-prints*, art. arXiv :1409.0473, Sep 2014.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching Word Vectors with Subword Information. *arXiv e-prints*, art. arXiv :1607.04606, Jul 2016.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. *arXiv e-prints*, art. arXiv :1508.05326, Aug 2015.

Robin Cheong and Robel Daniel. transformers.zip : Compressing Transformers with Pruning and Quantization. Technical report, Stanford University, Stanford, California, 2019. URL `https://web.stanford.edu/class/cs224n/reports/custom/15763707.pdf`.

Yew Ken Chia, Sam Witteveen, and Martin Andrews. Transformer to cnn : Label-scarce distillation for efficient text classification. 2018.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation : Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, October 2014. Association for Computational Linguistics. doi : 10.3115/v1/W14-4012. URL `https://www.aclweb.org/anthology/W14-4012`.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the Properties of Neural Machine Translation : Encoder-Decoder Approaches. *arXiv e-prints*, art. arXiv :1409.1259, Sep 2014a.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv e-prints*, art. arXiv :1406.1078, Jun 2014b.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv e-prints*, art. arXiv :1412.3555, Dec 2014.

Alexis Conneau and Douwe Kiela. Senteval : An evaluation toolkit for universal sentence representations. *arXiv preprint arXiv :1803.05449*, 2018.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. *arXiv e-prints*, art. arXiv :1705.02364, May 2017a.

Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Word Translation Without Parallel Data. *arXiv e-prints*, art. arXiv :1710.04087, Oct 2017b.

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized Neural Networks : Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv e-prints*, art. arXiv :1602.02830, Feb 2016.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-XL : Attentive Language Models Beyond a Fixed-Length Context. *arXiv e-prints*, art. arXiv :1901.02860, Jan 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv e-prints*, art. arXiv :1810.04805, Oct 2018.

Cicero Dos Santos and Maira Gatti de Bayser. Deep convolutional neural networks for sentiment analysis of short texts. 08 2014.

Chaofei Fan. Quantized Transformer. Technical report, Stanford University, Stanford, California, 2019. URL `https://web.stanford.edu/class/cs224n/reports/custom/15742249.pdf`.

Emile Fiesler, Amar Choudry, and H John Caulfield. A weight discretization paradigm for optical neural networks. *Proceedings of SPIE - The International Society for Optical Engineering*, 03 1993. doi : 10.1117/12.20700.

Philip Gage. A new algorithm for data compression. *C Users J.*, 12(2) :23–38, February 1994. ISSN 0898-9788. URL `http://dl.acm.org/citation.cfm?id=177910.177914`.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional Sequence to Sequence Learning. *arXiv e-prints*, art. arXiv :1705.03122, May 2017.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing Deep Convolutional Networks using Vector Quantization. *arXiv e-prints*, art. arXiv :1412.6115, Dec 2014.

Song Han, Huizi Mao, and William J. Dally. Deep Compression : Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv e-prints*, art. arXiv :1510.00149, Oct 2015.

Qinyao He, He Wen, Shuchang Zhou, Yuxin Wu, Cong Yao, Xinyu Zhou, and Yuheng Zou. Effective Quantization Methods for Recurrent Neural Networks. *arXiv e-prints*, art. arXiv :1611.10176, Nov 2016.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786) :504–507, 2006. ISSN 0036-8075. doi : 10.1126/science.1127647. URL `http://science.sciencemag.org/content/313/5786/504`.

Geoffrey Hinton. Neural networks for machine learning, 2012. Coursera, video lectures.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. *arXiv e-prints*, art. arXiv :1503.02531, Mar 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 :1735–80, 12 1997. doi : 10.1162/neco.1997.9.8.1735.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized Neural Networks : Training Neural Networks with Low Precision Weights and Activations. *arXiv e-prints*, art. arXiv :1609.07061, Sep 2016.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018a.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018b.

Rie Johnson and Tong Zhang. Effective Use of Word Order for Text Categorization with Convolutional Neural Networks. *arXiv e-prints*, art. arXiv :1412.1058, Dec 2014.

Michael I. Jordan. Artificial neural networks. chapter Attractor Dynamics and Parallelism in a Connectionist Sequential Machine, pages 112–127. IEEE Press, Piscataway, NJ, USA, 1990. ISBN 0-8186-2015-3. URL http://dl.acm.org/citation.cfm?id=104134.104148.

Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D13-1176.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A Convolutional Neural Network for Modelling Sentences. *arXiv e-prints*, art. arXiv :1404.2188, Apr 2014.

Yoon Kim and Alexander M. Rush. Sequence-Level Knowledge Distillation. *arXiv e-prints*, art. arXiv :1606.07947, Jun 2016.

Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. Skip-Thought Vectors. *arXiv e-prints*, art. arXiv :1506.06726, Jun 2015.

Filippos Kokkinos and Alexandros Potamianos. Structural Attention Neural Networks for improved sentiment analysis. *arXiv e-prints*, art. arXiv :1701.01811, Jan 2017.

Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification, 2015. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9745/9552.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4) :541–551, December 1989. ISSN 0899-7667. doi : 10.1162/neco.1989.1.4.541. URL http://dx.doi.org/10.1162/neco.1989.1.4.541.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *arXiv e-prints*, art. arXiv :1607.06450, Jul 2016.

Fengfu Li, Bo Zhang, and Bin Liu. Ternary Weight Networks. *arXiv e-prints*, art. arXiv :1605.04711, May 2016.

Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. Neural Networks with Few Multiplications. *arXiv e-prints*, art. arXiv :1510.03009, Oct 2015.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Improving Multi-Task Deep Neural Networks via Knowledge Distillation for Natural Language Understanding. *arXiv*

e-prints, art. arXiv :1904.09482, Apr 2019a.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-Task Deep Neural Networks for Natural Language Understanding. *arXiv e-prints*, art. arXiv :1901.11504, Jan 2019b.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. *arXiv e-prints*, art. arXiv :1508.04025, Aug 2015.

Thang Luong, Richard Socher, and Christopher Manning. Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 104–113. Association for Computational Linguistics, 2013. URL `http://aclweb.org/anthology/W13-3512`.

M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini. Fast neural networks without multipliers. *IEEE Transactions on Neural Networks*, 4(1) :53–62, Jan 1993. ISSN 1045-9227. doi : 10.1109/72.182695.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv e-prints*, art. arXiv :1301.3781, Jan 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. *arXiv e-prints*, art. arXiv :1310.4546, Oct 2013b.

Asit Mishra and Debbie Marr. Apprentice : Using Knowledge Distillation Techniques To Improve Low-Precision Network Accuracy. *arXiv e-prints*, art. arXiv :1711.05852, Nov 2017.

Joachim Ott, Zhouhan Lin, Ying Zhang, Shih-Chii Liu, and Yoshua Bengio. Recurrent Neural Networks With Limited Numerical Precision. *arXiv e-prints*, art. arXiv :1608.06902, Aug 2016.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. Scaling Neural Machine Translation. *arXiv e-prints*, art. arXiv :1806.00187, Jun 2018.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove : Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL `http://www.aclweb.org/anthology/D14-1162`.

Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *arXiv e-prints*, art. arXiv :1802.05668, Feb 2018.

Jordan B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1) :77 – 105, 1990. ISSN 0004-3702. doi : https://doi.org/10.1016/0004-3702(90)90005-K. URL http://www.sciencedirect.com/science/article/pii/000437029090005K.

Alec Radford. Improving language understanding by generative pre-training. 2018.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net : ImageNet Classification Using Binary Convolutional Neural Networks. *arXiv e-prints*, art. arXiv :1603.05279, Mar 2016.

Bhargava Reddy, Ye-Hoon Kim, Sojung Yun, Chanwon Seo, and Junik Jang. Real-time driver drowsiness detection for embedded system using model compression of deep neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

Jürgen Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Comput.*, 4(2) :234–242, March 1992. ISSN 0899-7667. doi : 10.1162/neco.1992.4.2.234. URL http://dx.doi.org/10.1162/neco.1992.4.2.234.

Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *International Conference on Acoustics, Speech and Signal Processing*, pages 5149–5152, 2012.

Jean Senellart, Dakun Zhang, Bo WANG, Guillaume KLEIN, Jean-Pierre Ramatchandirin, Josep Crego, and Alexander Rush. OpenNMT system description for WNMT 2018 : 800 words/sec on a single-core CPU. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 122–128, Melbourne, Australia, July 2018. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W18-2715.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi : 10.18653/v1/P16-1162. URL https://www.aclweb.org/anthology/P16-1162.

Richard Socher, Christopher D. Manning, and Andrew Y. Ng. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *In Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, 2010.

Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 801–809, USA, 2011a. Curran Associates Inc. ISBN 978-1-61839-599-3. URL http://dl.acm.org/citation.cfm?id=2986459.2986549.

Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2011b.

Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1201–1211. Association for Computational Linguistics, 2012. URL http://aclweb.org/anthology/D12-1110.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642. Association for Computational Linguistics, 2013. URL http://aclweb.org/anthology/D13-1170.

Sandeep Subramanian, Sai Rajeswar Mudumba, Alessandro Sordoni, Adam Trischler, Aaron C Courville, and Chris Pal. Towards text generation with adversarially learned neural outlines. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7551–7563. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7983-towards-text-generation-with-adversarially-learned-neural-outlines.pdf.

Sandeep Subramanian, Adam Trischler, Yoshua Bengio, and Christopher J Pal. Learning General Purpose Distributed Sentence Representations via Large Scale Multi-task Learning.

arXiv e-prints, art. arXiv :1804.00079, Mar 2018.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014. URL `http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf`.

C. Z. Tang and H. K. Kwan. Multilayer feedforward neural networks with single powers-of-two weights. *IEEE Transactions on Signal Processing*, 41(8) :2724–2727, Aug 1993. ISSN 1053-587X. doi : 10.1109/78.229903.

Andrew Tierno. Quantized Transformer. Technical report, Stanford University, Stanford, California, 2019. URL `https://web.stanford.edu/class/cs224n/reports/custom/15848474.pdf`.

George Tucker, Minhua Wu, Ming Sun, Sankaran Panchapagesan, Gengshen Fu, and Shiv Vitaladevuni. Model compression applied to small-footprint keyword spotting. In *INTERSPEECH*, pages 1878–1882, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv e-prints*, art. arXiv :1706.03762, Jun 2017.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE : A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. *arXiv e-prints*, art. arXiv :1804.07461, Apr 2018.

Peiqi Wang, Xinfeng Xie, Lei Deng, Guoqi Li, Dongsheng Wang, and Yuan Xie. Hitnet : Hybrid ternary recurrent neural network. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 604–614. Curran Associates, Inc., 2018. URL `http://papers.nips.cc/paper/7341-hitnet-hybrid-ternary-recurrent-neural-network.pdf`.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. *arXiv e-prints*, art. arXiv :1704.05426, Apr 2017.

Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized Convolutional Neural Networks for Mobile Devices. *arXiv e-prints*, art. arXiv :1512.06473, Dec

2015.

Xundong Wu, Yong Wu, and Yong Zhao. Binarized Neural Networks on the ImageNet Classification Task. *arXiv e-prints*, art. arXiv :1604.03058, Apr 2016a.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's Neural Machine Translation System : Bridging the Gap between Human and Machine Translation. *arXiv e-prints*, art. arXiv :1609.08144, Sep 2016b.

Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. LQ-Nets : Learned Quantization for Highly Accurate and Compact Deep Neural Networks. *arXiv e-prints*, art. arXiv :1807.10029, Jul 2018.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 649–657. Curran Associates, Inc., 2015. URL `http://papers.nips.cc/paper/5782-character-level-convolutional-networks-for-text-classification.pdf`.

Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. DoReFa-Net : Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv e-prints*, art. arXiv :1606.06160, Jun 2016.

Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies : Towards story-like visual explanations by watching movies and reading books. In *arXiv preprint arXiv :1506.06724*, 2015.