Univeristy College London
Engineering Faculty
Department of Computer Science
MSc Machine Learning

# Back-to-the-Future Networks: Referring to the Past to Predict the Future

*Michał Daniluk*

Supervised by
Dr. Sebastian Riedel
Tim Rocktäschel
Johannes Welbl

September 2016

# Abstract

Language models are very useful in many applications such as speech recognition, machine translation and text summarization. Recently, deep learning have fueled language modeling research in the past years. This dissertation examines the role of Recurrent Neural Networks in modeling sequential data, and present four novel neural network architectures, namely Recurrent Neural Network N-gram, Attention, Key-Value and Key-Value-Predict models. This study offers three hypotheses: (1) does incorporating the history context to recurrent neural networks will improve the performance of language model systems?; (2) does storing the memory in Key-Value-Predict structure will improve the performance of language model?; (3) does the attention mechanism is able to capture some linguistic phenomena?

Our most important contributions are four novel state-of-the-art models that not only outperform current state-of-the-art approaches, but also are able to give answers on our research questions. The presented models are able to deal with the problem of memory compression in recurrent neural network by incorporating the history context. Finally, we evaluate our models against current state-of-the-art techniques on two public datasets: LAMBADA and English Penn Treebank, and on our own dataset from Wikipedia articles.

For accompanying code, see
`https://www.dropbox.com/s/r3ogm7nvdxtnner/ML-Thesis-Code.zip`

# Acknowledgements

I would like to thank my supervisor Dr. Sebastian Riedel for the useful comments, remarks and engagement during this project. Furthermore I would like to express my gratitude to my co-supervisors Tim Rocktäschel and Johannes Welbl for ideas and inspirations that came from our talks. Without their passionate participation and input, the project could not have been successfully conducted.

# Contents

# List of Figures

# List of Tables

Chapter 1

# Introduction

In this chapter we give a brief introduction to the problem being discussed in this dissertation, describing reasons and motivations behind it. Furthermore, we define research hypothesis which we wish to address in this thesis. Finally, we outline the content of the project.

## 1.1 Language Modeling

Language Modeling (LM) has been a crucial task in Natural Language Processing (NLP) and Language Understanding. They were originally developed for the problem of speech recognition [28, 39], and they are also widely used in many other fundamental tasks in NLP such as machine translation [8, 63] or text summarization [17, 48]. Training better language models often improves the performance of downstream task such as BLEU score for machine translation or word error rate for speech recognition.

Assume that we have a *corpus*, which is a set of sentences in some language. For example, we may have a large amount of data from the web, or may have a data from the BBC News. Given this corpus, the language model parameters might be estimated. Then, we will define $\mathcal{V}$ as a set of all words in some language. For example, for English language we might have

$$\mathcal{V} = \{\text{the, home, big, I, house, going, am, is} \dots\}$$

In practise, the size of $\mathcal{V}$ is finite and can be large. It might contain even tens of thousands of words.

A language model is defined as a probability distribution $p(w_1, w_2, \ldots, w_n)$ over a sequence of words from $\mathcal{V}$. For example, a language model is expected to assign a higher probability to *the house is big* than to *big the is house*, because of the correct order of words. A language model also supports predicting the completion of a sentence. For instance, they are expected to assign higher probability to *I am going home* than to *I am going house*. Moreover, a language model helps in speech recognition task to distinguish between phrases that sound similar. For example, *I helped Google recognize speech* and *I helped Google wreck a nice beach* are pronounced almost the same but have different meaning and the first one is more meaningful.

Furthermore, language models are also able to extract a knowledge that a dataset may contain. For example, Serban et al. [51] trained the language model on movie subtitles and these models are able to generate a basic answers to questions about facts, people and object colors. In addition, recently proposed a sequence-to sequence models use a conditional language models [41] as their fundamental component to solve a machine translation tasks [59, 12]. Language Models are also widely used in text [58] and video [54] generation tasks.

## 1.2 Motivation

People were always dreaming about machines that are even more intelligent than humans. One of the key part of considering machine as intelligent is ability to communicate with human. We can think about several ways which leads us towards text and speech understanding by intelligent machines - we can start by understanding basic commands as *make coffee for me*, then translating text from one language to another which allows to communicate with people speaking in different languages, and then answer questions about facts from the text, finally reaching machine that can communicate with humans at the same level as other human beings.

Assuming that out goal is to build a machine that can communicate with human in natural language. The state-of-the art methods of doing it use a neural networks (called *deep learning*), which mimic learning process of humans and are a simple abstraction of human brain. In summary, the main motivation behind this work is to improve the ability of language understanding by machines.

## 1.3 Objectives

Deep learning made a big impact on language modeling research last few years. More technically, Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) [26] are current state-of-the art Language Modeling approaches. Our main goal is to analyse and implement current state-of-the-art neural network approaches to language modeling and propose a potential improvements of current approaches. Therefore, there are three main hypothesis which we wish to address in this thesis.

First, a bottleneck associated with RNNs is memory compression problem. Since the input sequence is compressed and blended into a single dense vector, it becomes difficult to accurately memorize sequences [72]. As a result, long sequences are generalized poorly by the network, while the memory is wasted for a short sequences. We hypothesize that incorporating the history context to recurrent neural networks will improve the performance of language model systems.

Second, can we use recently proposed idea of Key-Value Memory Network [42] in language modeling system? Key-value paired memories, which are used in question-answering task, are a generalization of the way context are stored in memory. The reading stage is based on value memory, and the lookup stage is the key memory. We hypothesize that storing memory in more complicated structure Key-Value-Predict will give more flexibility in predicting the next word and will improve the performance.

Third, the source of RNNs outstanding performance is poorly understood. It is difficult to understand and interpret the architecture of RNNs from the linguistic point of view [31]. As a result, the lack of this interpretability limits the ability to design better architectures. We hypothesize that attention mechanism is able to give us interpretation which previous words in the sequence were used to predict the next one.

## 1.4 Structure of the Thesis

In this thesis we tackles all our hypothesis simultaneously. In Chapter 2 we present the overview of language modeling methods. Firstly, we describe the classical approaches that have been widely used. Then, more recent deep learning approaches and their applications to language modeling are presented. Chapter 3 covers our novel methods for language modeling task. First we present our Recurrent Neural Network N-gram model which combine the strengths of neural network in generalizing with the memorization and scalability of an classical N-gram model. Then we proposed three attentive models: Attention, Key-Value and Key-Value-Predict. Following that, we take a look at number of related works and describe similarities and differences between them and our architectures. In Chapter 4 we examine the proposed approaches on real-world datasets, analyse along various linguistic dimensions that our model captures and test our hypothesis. Finally, Chapter 5 summarizes our results and discusses the future work.

Chapter 2

# Background

This chapter aims to give an overview of the approaches to language modeling problem by firstly discussed classical approaches that have been successfully used, and then looking at more recent state-of-the art deep learning techniques and their applications to the problem of Language Modeling. As Jozefowicz discussed in [31], Deep Learning and Recurrent Neural Networks have changed language modeling research in the past years. These techniques outperforms the standard approaches that have been used for years.

The chapter start by discussing Language Models evaluation metrics, then classical widely used learning approaches of building LMs are presented. Following that, different types of neural networks and their applications to LMs have been explored: feedforward neural network, recurrent neural network and long short-term memory network. Moreover, we explain problem of capturing certain linguistic phenomena. Finally, we describe attention mechanism as one way of solving this problem.

## 2.1 Evaluation of Language Models

In the literature, language models are evaluated by two most common metrics. First, they are estimate by *word error rate*. Second, most commonly, they are evaluated by computing the *perplexity*. Both metrics are a measurement of how well a language model predicts a sample.

The word error rate is used to evaluate the performance of speech recognition system. It can be computed as:

$$WER = \frac{S+D+I}{N}, \tag{2.1}$$

where $S$ is the number of substitutions, $D$ deletions, $I$ insertions between prediction of the model and the reference transcription. Word error rate is computationally expensive to calculate and it is used only in speech recognition systems, which makes it difficult to compare language models from different research sites.

On the other hand, perplexity is the most popular way of evaluating language models. The perplexity ($PPL$) of word sequence $\mathbf{w}$ can be computed trivially and it is defined as:

$$PPL = \sqrt[K]{\prod \frac{1}{P(w_i|w_{1\ldots i-1})}} = 2^{-\frac{1}{K}\sum_{i=1}^{K} log_2 P(w_i|w_{1\ldots i-1})} \tag{2.2}$$

It can be seen as the inverse of the geometric average probability assigned to each word in the dataset. A low perplexity means that model is good at predicting the sample.

## 2.2 Statistical Language Modeling

To build an efficient language model, we should use a techniques which are specialized for processing sequential data. Since the total number of possible combination of words is extremely large, language models are high-dimensional and they must work on sparse discrete space. Much work has been done to develop models that work efficiently on sparse space. For example, count-based approaches such as n-gram models have been successfully used for many years. Furthermore, the Kneser-Ney smoothed 5-gram model [34] is a strong baseline, which have challenged neural network language models.

In this section, the statistical language modeling approaches are presented. We discuss the benefits of using them and also discuss reasons why this approaches were outperformed by deep learning techniques by looking at their most significant limitations.

### 2.2.1 N-gram Language Model

The earliest successful approach to language modeling was n-gram model. It defines the probability of observing a given sequence of words $w_1, \ldots, w_m$ as:

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i | w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{m} P(w_i | w_{i-(n-1)}, \ldots, w_{i-1}) \tag{2.3}$$

Training n-gram models is based on maximizing the likelihood which can be computed simply by counting the frequency of word's occurrence:

$$P(w_i | w_1, \ldots, w_{i-1}) = \frac{\text{count}(w_{i-(n-1)}, \ldots, w_{i-1}, w_i)}{\text{count}(w_{i-(n-1)}, \ldots, w_{i-1})} \tag{2.4}$$

In other words, the probability that some word appears after a given sequence can be estimated as number of times that this sequence augmented with this word divided by frequency of given sequence in the training set. In practise, the probability is smoothed by assigning some low probability to words that does not appear in the training corpus [34]. N-gram model use a Markov assumption as an approximation of the true language model. Current word depends only on $n$ previous words, where $n = 1$ refers to *unigram* (does not take into account history), $n = 2$ to a *bigram*, $n = 3$ is a *trigram* and so on.

The most important advantages of n-gram models are simplicity, speed, generality and ease of scale them efficiently. N-gram is still a good baseline model for more advanced techniques, not only because they reach low perplexity, but also because their are computationally much less complex than, for example, neural network models.

However, on the other hand, there are some disadvantages of using n-gram models. First of all, data sparsity is a big problem in language modeling that refers to the fact that language has a lot of rare, varied and complex events, that even using an extremely big corpus, it is not possible to model language accurately using n-gram. With increasing the parameter $N$, the number of possible n-grams increases exponentially, so we need exponentially more data. What is more, n-gram models use a fixed length of context that is used to predict the next word. As a results, they are not able to learn longer context patterns. In addition, n-gram models assign a low probability to the rare words.

### 2.2.2 Class Based N-gram Model

One of the most important problem of training high order n-grams model is a data sparsity. In order to deal with this problem, Class Based Model was proposed. It allows better generalization to unseen patterns in the training data by assigning each word to a single class, and then training n-gram model on these classes. See [30] for more information about techniques of mapping words to certain classes.

Despite the fact that this approach slightly improve the perplexity, it does not reach a popularity in the real applications. One of the reason is that this class based models has high complexity during inference and the improvement in perplexity vanish with increased amout of training data [30].

### 2.2.3 Cache Language Model

Cache Language Models are design to deal with the problem of representing longer term patterns. Standard n-gram models assign a very low probability to the rare words. Cache Language Models used the observation that if certain word have already occurred in the recent history, the probability of observing it again increases. It introduces a cache component to standard n-gram model, which allows to deal with this regularity. It is usually done by training one n-gram model on recent history and another one on the whole corpus.

The main benefit of language cache models is performance improvement and easy implementation. On the other hand, the main drawback is that it increases the word rate rate in speech recognitions systems, which is explained in [19]. If the speech recognition system decodes incorrectly a word, the cache model might increase the chance of doing the same mistake again.

### 2.2.4 Decision Trees Language Models

Decision trees algorithm were first applied to language modeling by Bahl [4]. The main idea behind them is to split data by asking arbitrary binary questions about the history. For example, it is possible to divide data by asking if specific word appears in the last *N* words. However, the performance of decision trees is worse than n-gram models with the same order *N* [46]. As a result, researches started working on random forest models, which is a random combination of many decision trees [70].

Despite the fact that random forest models reduce both perplexity and word error rate compare to Kneser-Ney smoothed 5-gram model, they does not reach a popularity in the real applications. The disadvantage of this models is a high computational complexity and decreasing improvement for a large amount of training corpus.

## 2.3 Neural Network Language Model

Deep learning have been used in language modeling task for a long time. The first known work of using neural networks to language modeling was done by Jeff Elman in 1990 [16]. After ten years, neural network language model with a comparison to classical approaches was proposed by Yoshua Bengio [5], who used feedforward neural networks with fixed length context. This work was further investigated by Goodman [20], who showed that it provides a significant improvement compare to standard approaches (n-grams, class-based models). Later, Holger Shwenk [50], who followed this approach, has shown that neural network based LMs outperforms other methods in speech recognition tasks. More recently, Mikolov [39] proposed a new recurrent neural network based language model (*RNN LM*) and compared against good baseline systems. Their results have shown that is is possible to obtain around 50% reduction of perplexity by using a mixture of several RNN LMs, compared to current state of the art backoff language model.

Although there are several difference in the neural network based language models that have been widely applied so far, there are some common principles:

- The representations of words is embedded to the fixed length vector.
- A softmax function is used to produced a correctly normalized probability distribution.
- The cross entropy error is minimized during the training.

Neural network language models learn both the probability distribution of given sequence of words and a vector representation, which is called embedding, for every word in vocabulary. As a results, neural network language model is able to learn embeddings which carry important linguistic information. For example, if the representation of words *big* and *large* share many attributes, then model will give similar prediction for the sequence that contains word *big* or *large* at the same position in the sentence. In summary, neural networks have the ability to learn word representation which carry important linguistic information. As a result, using learned word embeddings solves the problem of data sparsity and explain better performance for set of words which have never been seen in the training corpus.

### 2.3.1 Feedforward Neural Network Based Language Model

Bengio et al. [5] proposed a feedforward neural network architecture (see Fig. 2.1) which aims to model natural language. The model proposed by Bengio consists of

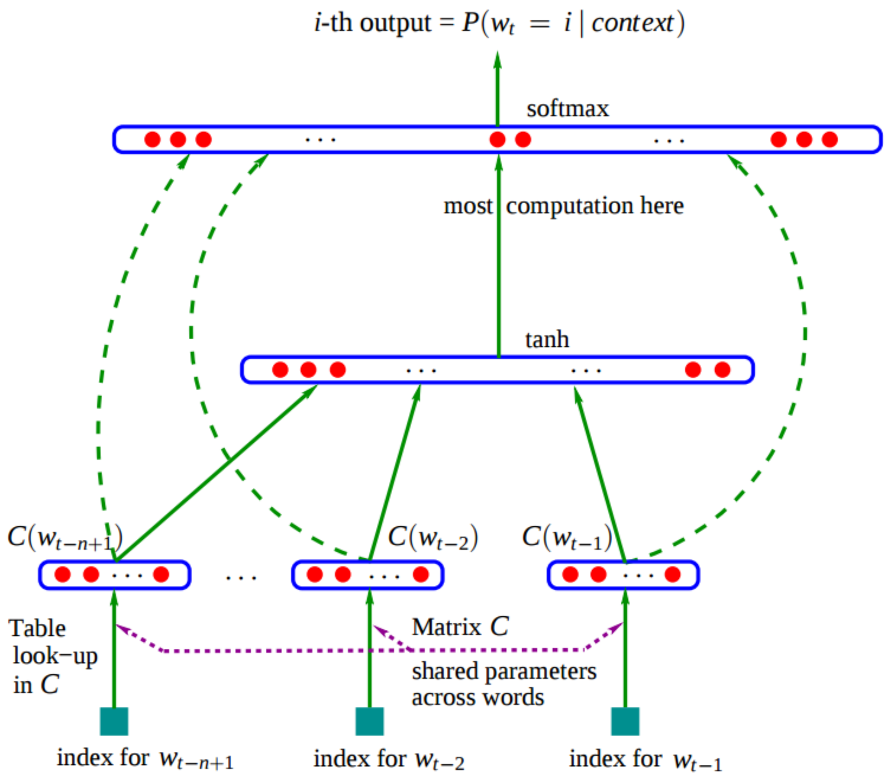


**Figure 2.1:** Architecture of Bengio's neural Language Model [5]

input, projection, hidden and output layers. First, the input layer takes last $n-1$ words from recent history and encodes them using one-hot encoding. As a next step, the projection layer is used to learned a word embeddings by multiplying an one-hot vector with a trained projection matrix. A hidden layer follows that applied a non-linear function to the concatenation of word embedding to $n-1$ previous words. After the hidden layer, the output layer calculates a probability distribution of the next word.

The most significant disadvantage of Bengio's approach is that a feedforward network has to use a fixed length of history that needs to be specified before training. Thus, similar to n-grams, they are based on Markov assumption. In practise, it means that neural network can see only five to ten previous words to predict the next one.

### 2.3.2 Recurrent Neural Networks Based Language Model

Recurrent Neural Networks (RNN) based language model has been proposed to overcome a potential limitations of feedforward neural network Language Model such as the limited representation of history. The history in feedforward neural networks is just $n-1$ previous words, whereas the hidden layer of RNN represents unbounded previous history which is learned during the training. As a results, the most significant benefit of recurrent neural network over feedforward network is the ability to learn longer context patterns. Advanced patterns, that rely on words at variable position in the history, can be model more effectively with RNN, because they can remember some specific words in the hidden state, while feedforward network uses different parameters for each position of the word in the history. The hidden state is considered as a *memory* which capture information which has been presented so far. However, it is also claimed that, in practise, learning a long term dependencies can be difficult [6].

More precisely, recurrent neural network is a type of neural network where connections between neurons make a directed cycle. The idea behind RNNs is that the state of the hidden units depends on previous state of the network. The same operation is performed for every element of the sequence. The typical architecture of RNN is presented on Figure 2.2. It shows that the hidden state receives input

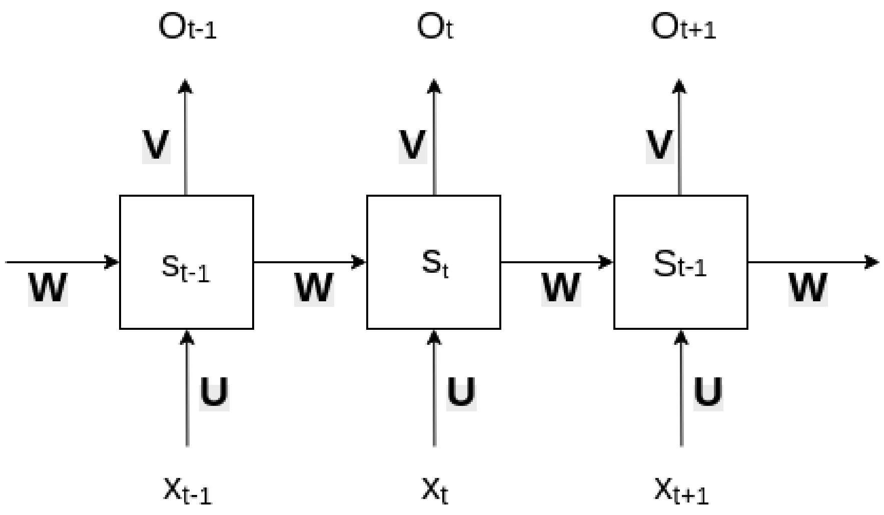


**Figure 2.2:** A three time-steps Recurrent Neural Network

value from previous hidden state and from current input. In contrast, feedforward neural network receives input only from the current input word, as shown in Figure 2.1.

At each time step, the hidden state $s_t$ is calculated from two inputs: previous hidden state $s_{t-1}$ and input at current time-step $x_t$, which is a embedding of current word. Then non-linear transformation is applied to obtain the output:

$$s_t = f(Ux_t + Ws_{t-1}) \tag{2.5}$$

where $U$ and $W$ are trained projection matrices, $f$ is a non-linear function such as *tanh* or *ReLU*. The output of the hidden layer at time $t$ is calculated from current state $s_t$ as:

$$o_t = \text{softmax}(Vs_t) \tag{2.6}$$

where $V$ is a trained projection matrix. It represents the probability distribution of the next word over all words in vocabulary. The softmax function, that ensures that the outputs form a valid probability distribution, is defined as:

$$g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}} \tag{2.7}$$

RNN can be extended by stacking multiple recurrent hidden layers on top of each other [49, 27, 22]. This model, called stacked RNN or multi-layer RNN, gives a higher learning capacity and encourage each hidden layer to operate at a different timescale.

### 2.3.3 LSTM Networks

Recurrent Neural Networks with long short-term memory (LSTM) unit was originally proposed by Hochreiter [26]. They has a track record of success in many NLP tasks such as machine translation [59], language modeling [73], constituency parsing [64], sentence compression [47]. LSTM is a type of activation unit (see Fig. 2.3), which replace the standard hidden unit in RNN network. Instead of having a

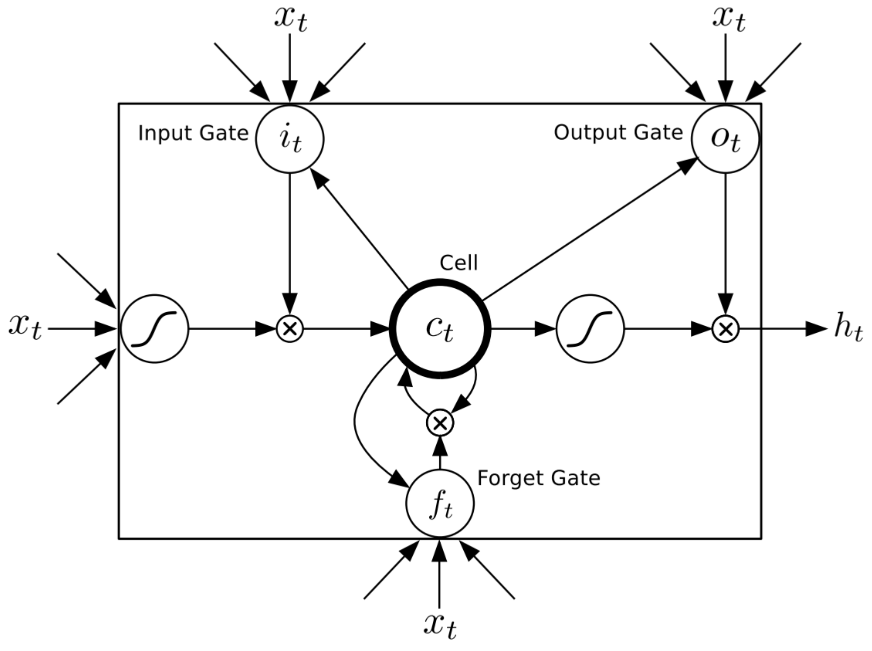


**Figure 2.3:** Long Short-term Memory Cell [22]

single neural network layer, LSTM unit consists a memory cell that can store information for a long period of time, as well the three types of boolean gates which control the flow of information into and out of the memory cell: forget gate (Eq. 2.8), input gate (Eq. 2.9) and output gate (Eq. 2.10). Given an current input vector $x_t$, the previous output vector $h_{t-1}$ and cell state $c_{t-1}$, an LSTM computes the next output $h_t$ and cell state $c_t$ as:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{2.8}$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{2.9}$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.10}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c[h_{t-1}, x_t] + b_c) \tag{2.11}$$

$$h_t = o_t \odot \tanh(C_t) \tag{2.12}$$

where $W_f$, $W_i$, $W_o$, $W_c$ are trained projection matrices and $b_f$, $b_i$, $b_o$, $b_c$ are trained biases that parametrize the gates and transformation of the input and $\odot$ denotes the element-wise multiplication of two vectors.

All gates are a learned sigmoidal units. They have exactly the same equations with different parameter matrices. Its inputs are the previous hidden state $h_{t-1}$ and current input $x_t$. *Forget gate* (2.8) allows to decide what information is going to be erase from the cell state. It defines how much of the previous state is thrown away. For example, in the language modeling problem, the cell state may include the gender of the person to use a correct pronouns. After seeing a new subject, this gate allows to forget the gender of the old subject. This gate was not introduce in the original LSTM [26], but was proposed by Gers et al. [18]. *Input gate* (2.9) is responsible for deciding how much of the newly computed stated for the current input is updated. *Output gate* determines which parts of the memory cell should be output as input to the next layer and for the next time step.

Compared to the standard RNN, the LSTM separates the memory from the hidden state. As a result LSTMs have been shown to outperform vanilla RRNs on many tasks, including language modeling [57]. Furthermore, LSTM networks were designed to learn long-term dependencies more easily than the simple recurrent architecture. Moreover, they deal with the gradient vanishing problem, because memory cells in the LSTM are additive with respect to time. Although LSTM does not deal with the gradient exploding problem, simple optimization strategies as gradient clipping [45] works well in practise.

### 2.3.4 Learning Algorithm

Neural Language models are trained using the cross-entropy criterion, which is equivalent to maximum likelihood, by minimizing the objective function

$$J = \sum_{i=1}^{N} \log P(w_i | w_{i-1}, \ldots, w_{i-n+1}) \tag{2.13}$$

where $N$ is the number of previous words in the sequence.

Finding the minimum of the objective function is the optimization problem. The most common approach to solve it is a stochastic gradient descent algorithm. More recently, most advanced algorithm, the Adam (Adaptive Moment Estimation) [33] optimizer is used in training. It is an extension of stochatistic gradient descent, which computes adaptive learning rates for each parameter and uses a momentum.

Backpropagation algorithm [25] is used to train a feedforward neural network. However, a more common approach to train RNN is Back-propagation Through Time (BPTT) [66]. Recurrent Neural Network share the same parameters **U,V,W** across over all steps. In BPTT, the gradient is calculated by first treating all parameters as independent, and then running the standard backpropagation algorithm. Finally, all the corresponding gradients are averaged. Thus, they are tied together. In practise, instead of backpropagate the error back to the beginning, the algorithm is run over a fixed window, which results in an approximation of the true gradient.

BPTT has the problem of the vanishing/exploding gradient, that is is a difficulty in training in neural networks that propagate gradient for a long number of steps. The exploding gradient problem, introduced by Bengio [6], refers to large increase in the norm of gradient during training. If gradient increase too fast during training and it will cause a memory overflow problem. The vanishing gradient problem refers to the opposite behaviour. If the value of gradient go exponentially quickly to norm 0, it will make the model impossible to learn long distance dependencies.

To deal with the problem of exploding gradient, the gradient clipping method was proposed by Pascanu et al. [45]. The gradient is set back to a small number after it exceeds a threshold value using the following update rule:

$$g = \frac{\text{theshold}}{||g||} g \tag{2.14}$$

There are two main approaches to deal with vanishing gradient problem. Firstly, we can use previously described LSTM units. Secondly, instead of using classical sigmoid activation function in RNN, Rectified Linear Units (ReLUs) can be used.

Neural networks tends to overfit and they requires good regularization. In addition to classic regularization methods $L_1$ and $L_2$, Dropout [53] is recently the most powerful for neural networks. It works by dropping out hidden units in a neural network along with its incoming and outgoing connections. The hidden units are removed randomly with some defined probability. Zaremba et al. showed in [73] how to correctly apply dropout to Reccurent Neural Networks. According to them dropout should be applied only to inputs and outputs of the network.

## 2.4 Understanding LSTM

The question which appears with LSTM is to what extent the LSTMs is able to memorize sequences and capture certain linguistic phenomena. Although LSTMs can produce the list of all previous hidden states, the next state is always computed from the current state, making all previous states redundant. It makes this computations in a Markovian manner, but LSTMs can in theory model unbounded sequence without any Markov assumption. It is common belief that LSTM networks are able to learn a long term dependencies using the gating mechanism. However, it is not clear that it applies to real-world data with the common use stochastic gradient descent and BPTT learning algorithm. Evidence of learning long term dependencies mainly comes from evaluating LSTMs in downstream task such as learning the recursive structure of the corpus [7].

Despite the huge popularity of LSTM in language modeling, the understanding and justification why it works is a challenge. This raises concerns of the limits of our ability to design better architectures. There have been several attempts to understand the component of LSTM. Greff [24] and Chung [14] analyzed the effect on performance of removing various gates and connections. The results suggest that forget and output gates are the most important part of LSTMs. Zaremba and Jozefowicz [71] evaluate ten thousands of different RNN architectures and find that the initialization of forget gate bias has crucial effect on LSTM's performance. However, this analysis is limited to examining the effects on global level of the perplexity on the final test set.

While these results are very important to choose the appropriate LSTM architecture, they do not find what information is captured by hidden states of LSTM. RNN architecture make it difficult to analyze what information is retained at their hidden states at each time step. Karpathy [32] provides the first empirical exploration of the interpretation of cell states of LSTMs during the the predictions of next word. Their experiments has revealed the existence of interpretable cells that keep track of long-range dependencies, such as quotes and brackets.

In practise it has proven challenging to LSTM network to learn long linguistic patterns, because of memory compression problem. The whole input sequence is compressed into a single dense vector. Thus, storing past information requires a large memory capacity. As a result, long sequences are generalized poorly by the network, while the memory is wasted for a short sequences.

## 2.5 Attention based Models

Recent work attempts to address latter limitations using attention mechanism, which is one of the most interesting architectural innovation in neural networks. Many of them show that attention techniques improve the performance of many deep learning models [43, 74]. It has already demonstrated a success, particularly in computer vision tasks: digit classification [43], handwriting synthesis [21], image captioning

[69]. But only recently attention mechanism was introduces into recurrent neural networks architectures that are typically used in NLP and have been successfully applied to NLP tasks. Weston et al. [67] proposed a neural networks based model called Memory Networks. It uses an external memory which are read and written on the fly with respect to the attention. Since then, many solutions have been proposed to solve NLP tasks such as speech recognition [13], recognising textual entailment [47], machine translation [3], question answering [42, 56] and language modeling [61].

Attention mechanism allows model to access previous information without having to store it in RNN hidden state. More precisely, attention mechanism is giving the access to its internal memory, which are the previous hidden states. It allows network to refer back to the previous inputs, instead of forcing to encode all information into one fixed-length vector. Model can learn which previous outputs are useful to predict the next word. There are two ways to get attention from the sequence (following [65]): by word by word attention (called *impatient*) or by the whole sequence (called *attentive*). We can also distinguish between *hard* and *soft* attention. *Hard* attention mechanism gives a discrete selections of memory locations, which has disadvantage that it makes the whole model not differentiable with respect to its inputs. As a results, reinforcement learning is used to solve learning problem. On the other hand, more commonly used, *soft* attention means that the network compute the attention output as a weighted combination of all memory locations, instead of value from a single discrete location. Making attention mechanism *soft* has the advantage that the network can be trained using a backpropagation algorithm.

In summary, the attention mechanism is designed to solve the memory compression problem in recurrent neural networks by giving the access to its previous hidden states. Attention Model is able to learn to assign weights to different parts of the input sequence instead of treating all input items equally. This is not only a way to improve the performance, but it is also a powerful tool of visualization.

Chapter 3

# Model Design

In this chapter we present different neural models for language modeling task. First of all, we describe the vanilla LSTM model. Then we proposed four new architectures of Recurrent Neural Network. The first one, Recurrent Neural Network N-gram model, is a simple combination of RNN with N-gram approach. The next three models: Attention, Key-Value, Key-Value-Predict models are based on RNN with attention mechanism. Following that, we take a look at number of related works and describe similarities and differences between them and our architectures.

## 3.1 Vanilla LSTM Model

Our model takes a discrete set of inputs $\mathbf{w_1}, \ldots \mathbf{w_N}$. Then model converts the entire set of $\{\mathbf{w_i}\}$ into embedding vectors $\{\mathbf{x_i}\}$ of dimension $d$ which are computed by embedding each $\mathbf{w_i}$ in continuous space using an embedding matrix $\mathbf{A}$ of size $d \times V$, where $V$ is a size of vocabulary and $d$ is the embedding size. Since the word embedding are learned during the training, the model can represent the knowledge about particular words. The embedding matrix $\mathbf{A}$ is initialized randomly and the model is learned to differentiate the meaning of words.

The set of word embeddings $\mathbf{x_1}, \ldots \mathbf{x_N}$ is then processed via LSTM units to output vectors $\mathbf{h_i}$, which are used to compute the probability distribution of the next word over all words in vocabulary. To compute this efficiently, the resulting vector is defined as:

$$Wh_i + b \tag{3.1}$$

where $\mathbf{W} \in \mathbb{R}^{V \times k}$ is a trained projection matrix, and $\mathbf{b} \in \mathbb{R}^{V}$ is a trained bias vector. The probability distribution over all words in vocabulary is computed by feeding the resulting vector into the softmax function.

The model tries to maximize the probability of predicting the correct word at every time-step by maximizing the averaged log probability of the whole corpus:

$$J = \frac{1}{N} \sum_{i=1}^{N} \log p(w_i) \tag{3.2}$$

## 3.2 RNNN Model

Inspired by recent works [36, 37, 9, 68, 29, 52] that has shown that RNNs are great in combination with N-grams, as they combine the strengths of neural network in generalizing with the memorization and scalability of an N-gram model, we introduce Recurrent Neural Network N-gram model with LSTM unit (called RNNN). The RNNN model architecture is illustrated on Figure 3.1. The idea is to use differ-

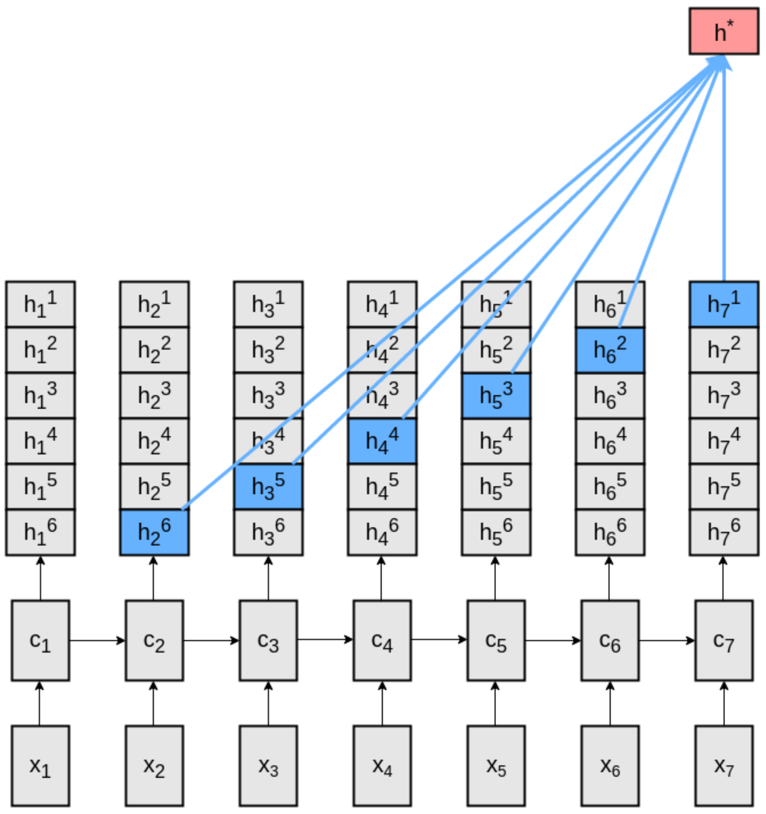

h*
h1^1 h1^2 h1^3 h1^4 h1^5 h1^6
h2^1 h2^2 h2^3 h2^4 h2^5 h2^6
h3^1 h3^2 h3^3 h3^4 h3^5 h3^6
h4^1 h4^2 h4^3 h4^4 h4^5 h4^6
h5^1 h5^2 h5^3 h5^4 h5^5 h5^6
h6^1 h6^2 h6^3 h6^4 h6^5 h6^6
h7^1 h7^2 h7^3 h7^4 h7^5 h7^6
c1 c2 c3 c4 c5 c6 c7
x1 x2 x3 x4 x5 x6 x7

**Figure 3.1:** Architecture of Recurrent Neural Network 5-gram model.

ent parts of past output vectors to predict words on different positions in the feature.

Each output vector $\mathbf{h_j}$ is divided into $N$ parts $[\mathbf{h_j^1}, \ldots, \mathbf{h_j^N}]$. Let $\mathbf{C} \in \mathbb{R}^k$ be a matrix consisting of different parts of $N+1$ previous vectors $[\mathbf{h_j^1}, \mathbf{h_{j-1}^2}, \ldots, \mathbf{h_{j-N}^{N+1}}]$. The final representation of the new output vector is obtain from non-linear combination of different parts of $N+1$ previous output vectors using

$$h^* = \tanh(W^C C) \qquad h^* \in \mathbb{R}^{\frac{k}{N+1}} \tag{3.3}$$

where $\mathbf{W^C} \in \mathbb{R}^{\frac{k}{N+1} \times k}$ is trained projection matrix. Note that the first part of the output vector is used to predict the next word, the second part is used to predict the word after the next one, the $N$th part is used to predict the word on the position $N+1$ in the future.

## 3.3 Attention Model

The Attention LSTM model architecture is presented on the Figure 3.2. It introduces a new attention layer to the vanilla LSTM model, which takes $N$ previous output vectors of hidden states and produces new representation of the output vector which are then used to predict the next word.

Let $\mathbf{Y} \in \mathbb{R}^{k \times L}$ be a matrix consisting of output vectors $[\mathbf{h_1} \ldots \mathbf{h_L}]$ that LSTM produces when reading the last $L$ words, where $k$ is the size of the hidden state of the LSTMs. Furthermore, let $\mathbf{H} \in \mathbb{R}^{k \times L}$ be a matrix consisting final output vector that is multiplied $L$ times $[\mathbf{h_N}, \ldots, \mathbf{h_N}]$. The attention weights $\alpha \in \mathbb{R}^L$ are computed from non-linear transformation of the LSTM outputs and the vector $\mathbf{r} \in \mathbb{R}^k$ is a weighted representation of previous output vectors. This can be modeled as:

$$M = \tanh(W^Y Y + W^h H) \qquad M \in \mathbb{R}^{k \times L} \tag{3.4}$$

$$\alpha = \mathrm{softmax}(w^T M) \qquad \alpha \in \mathbb{R}^L \tag{3.5}$$

$$r = Y \alpha^T \qquad r \in \mathbb{R}^k \tag{3.6}$$

where $\mathbf{W^y}$, $\mathbf{W^h} \in \mathbb{R}^{k \times k}$ are trained projection matrices and $\mathbf{w} \in \mathbb{R}^k$ is a trained vector. Note that each row of matrix $\mathbf{M}$ corresponds to intermediate attention rep-

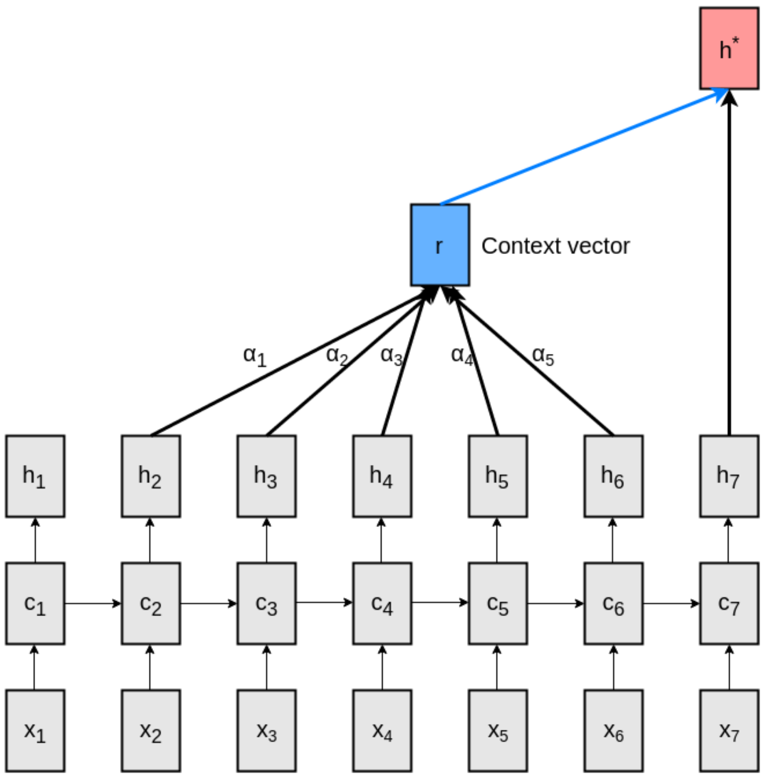


**Figure 3.2:** Architecture of Attentive LSTM model.

resentation $m_i$ of the $i$th word in the sequence and it is obtained from non-linear combination of transformed final output vector $\mathbf{h_N}$ and the sequence's output vector $\mathbf{h_i}$ ($i$th column in matrix $\mathbf{Y}$).

The final representation of attention output is computed as a non-linear combination of the attention weighted representation $\mathbf{r}$ of previous outputs and the final output vector $\mathbf{h_N}$ as:

$$h^* = \tanh(W^p r + W^x h_N) \qquad h^* \in \mathbb{R}^k \tag{3.7}$$

where $\mathbf{W^p}$, $\mathbf{W^x}$ are trained projection matrices.

## 3.4 Key-Value Attention Model

Key-Value Attention network gives the model greater flexibility for predicting the next word and helps reduce the gap between computing the attention weight and calculating the convex combination of output vectors. To encourage such behaviour we apply key-value decomposition of the output vector similar to Miller et al. [42]. The difference is that they converted question embedding into key-value paired memory, whereas we define output vectors of the LSTM network as key-value pairs. In our case, key part of the output vectors is used to compute distribution of attention weights $\alpha$, while the value part is used to calculate weighted representation of previous output vectors and to define the final attention output vector $\mathbf{h}^*$. The Key-Value model architecture is presented on the Figure 3.3.

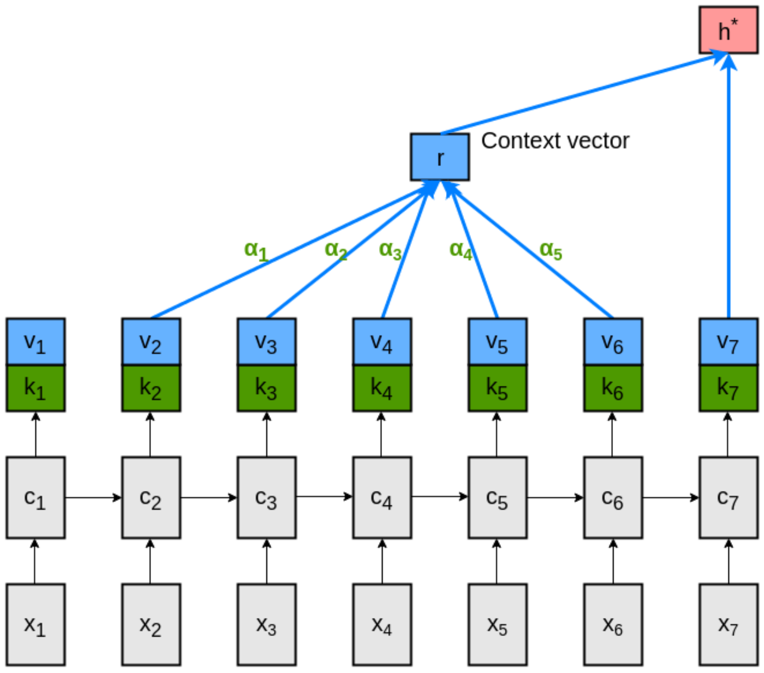


**Figure 3.3:** Architecture of Key-Value model.

In other words, key memory is designed to decide which previous output vectors are important, while the value memory is designed to compute the final output vector. This can me modeled as:

$$M = \tanh(W^Y Y + W^h H) \qquad M \in \mathbb{R}^{\frac{k}{2} \times L} \qquad (3.8)$$

$$\alpha = \text{softmax}(w^T M) \qquad \alpha \in \mathbb{R}^L \qquad (3.9)$$

$$r = Y\alpha^T \qquad r \in \mathbb{R}^{\frac{k}{2}} \qquad (3.10)$$

Note that the output vector $h_i$ contains pair of vector $(\mathbf{k_i}, \mathbf{v_i})$. The difference between

Key-Value Attention Model and previous described Attention Model is that matrix $\mathbf{Y} \in \mathbb{R}^{\frac{k}{2} \times L}$ consist of key part of output vectors $[\mathbf{k_1} \dots \mathbf{k_L}]$ that LSTM produces when reading the last $L$ words. Matrix $\mathbf{H} \in \mathbb{R}^{\frac{k}{2} \times L}$ consists of key part of the final output vector that is multiplied $L$ times $[\mathbf{k_N}, \dots, \mathbf{k_N}]$.

The final representation of attention output is computed as a non-linear combination of the attention weighted representation $\mathbf{r}$ of previous outputs and the value part of the final output vector $\mathbf{v_N}$ as:

$$h^* = \tanh(W^p r + W^x v_N) \qquad h^* \in \mathbb{R}^{\frac{k}{2}} \tag{3.11}$$

## 3.5 Key-Value-Predict Attention LSTM

The Key-Value-Predict Attention model is an extension of the Key-Value Attention Model. To give more flexibility to our model, we divided output vector of the recurrent neural network into three parts: key, value and predict $(\mathbf{k_i}, \mathbf{v_i}, \mathbf{p_i})$ (see architecture on Fig. 3.4). Key and value memory have the same application as in the

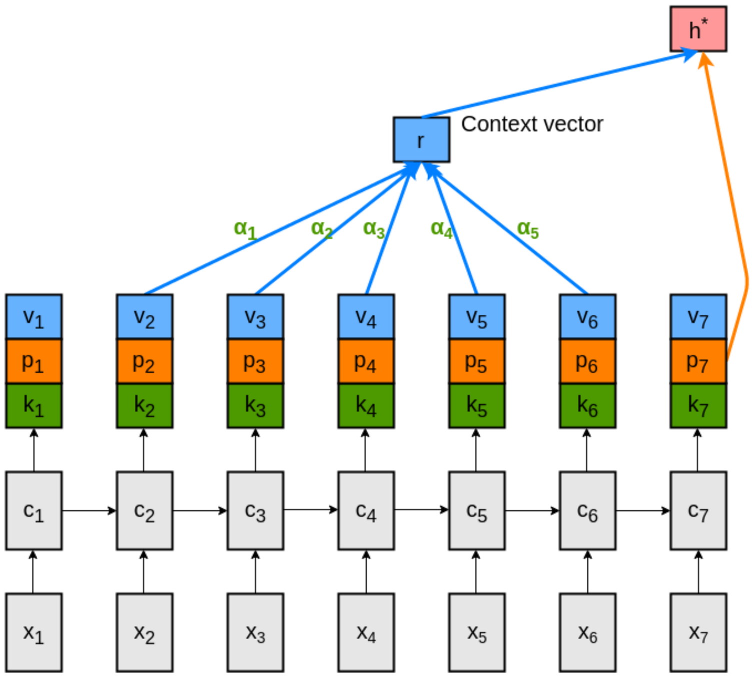


**Figure 3.4:** Architecture of Key-Value-Predict model.

Key-Value model. The predict part is only used to compute the final representation of attention output from the final output vector. Key-Value-Predict model uses the same structure and weights to compute attention distribution $\alpha$ and weighted representation **r** of previous output vectors as Key-Value Attention Model:

$$M = \tanh(W^Y Y + W^h H) \qquad M \in \mathbb{R}^{\frac{k}{3} \times L} \tag{3.12}$$

$$\alpha = \text{softmax}(w^T M) \qquad \alpha \in \mathbb{R}^L \tag{3.13}$$

$$r = Y \alpha^T \qquad r \in \mathbb{R}^{\frac{k}{3}} \tag{3.14}$$

The final output vector $\mathbf{h_i}$ contains three vectors $(\mathbf{k_i}, \mathbf{v_i}, \mathbf{p_i})$. Matrix $\mathbf{Y} \in \mathbb{R}^{\frac{k}{3} \times L}$ again consist of key part of output vectors $[\mathbf{k_1} \ldots \mathbf{k_L}]$ that LSTM produces when reading the last $L$ words. As in the Key-Value Attention model, matrix $\mathbf{H} \in \mathbb{R}^{\frac{k}{3} \times L}$ consists of key part of the final output vector that is multiplied $L$ times $[\mathbf{k_N}, \ldots, \mathbf{k_N}]$. The difference is that the final representation of attention output is computed as a non-linear combination of the attention weighted representation **r** of previous outputs and the **predict** part of the final output vector $\mathbf{p_N}$ as:

$$h^* = \tanh(W^p r + W^x p_N) \qquad h^* \in \mathbb{R}^{\frac{k}{3}} \tag{3.15}$$

## 3.6 Related Methods

A number of recent works have been done to improve the performance of language models by using RNNs or LSTMs-based models [14, 21, 35, 38, 2]. The memory in these models is incorporated in the hidden state of the network, which leads to memory compression problem and does not allow to learn complex patterns. Furthermore, the LSTM itself has a local memory cell. More recently, Cheng et al. [11] proposed a Long Short-Term Memory architecture, which differs from previous works that it has a memory tape instead of fixed memory cell. Compared to standard LSTM, they additionally introduce an attention layer to compute the adaptive memory and hidden representation. However, for language modeling task, they carry experiment only on small PTB data set, which does not have many long term dependencies. Our model differs from these models in that it does not enhance the internal memory of an LSTM.

Recent works attempts to address the limitation of capturing a long distance dependencies using external memories. The first attempts of using the memory in neural networks have been done by Steinbuch [55] and Taylor [60] by performing nearest-neighbor operations on input vectors and then fitting parametric models to the retrieved sets. More recently, Weston et al. [67] introduce Memory Networks that explicitly segregate memory storage from the computation of the neural network. Sukhbaatar et al. [56] trained their model end-to-end with a memory addressing mechanism related to soft attention. Recently proposed Neural Turing Machine [23] also combine neural networks with external memory resources, which can interact with by attentional process. Their model has shown promising results in simple sequence tasks such as copying and sorting. A common theme across this works is that they use external memories that interact with neural network, whereas our model directly uses an attention mechanism over past outputs states of the network.

Closely related to our Attention model is a Recurrent Memory Network (RMN) recently proposed by Tran et al. [61]. It combines the strengths of both LSTM and Memory Network [56]. In RMN, the Memory Block (MB) accesses the most recent input words and selectively attend to relevant words for predicting the next word given the current LSTM cell. The Memory Block consists of two lookup table $\mathbf{M}$ and $\mathbf{C}$ of size $|V| \times d$, where $|V|$ is the size of vocabulary and $d$ is dimension of word embedding. Matrix $\mathbf{M}$ was used to compute attention distribution, while matrix $\mathbf{C}$ was used to compute a context vector representation of input vectors. On the one hand, there are some similarities between our model and Recurrent Memory Network. First of all, we also use the attention mechanism which allows to attend previous words. However, our model attend over past output vectors, while RMN attend over past input word embeddings. RMN also uses the idea from Key-Value Memory Network [42] by using different lookup tables to compute attention distribution and to compute context vector representation. On the other hand, our model differs from RMN in many aspects. Firstly, RMN uses a global memory with two lookup tables which are used in attention mechanism. In contrast, we use local memory which is represents by Attention Layer. It allows our model to better learn a context of word. For example, there are many words with different meaning in different context such as *bark* which represent the same vector in the global embeddings memory, but different vector in Attention Layer memory. Secondly, RMN model has much more trainable parameters than our model by introducing two big learned lookup tables of size $|V| \times d$ each. For example, if the vocabulary size is 77K and embedding and hidden size are both 200, RMN model introduce at least 31 mln parameters, whereas parameters introduced by our model does not depend on vocabulary size. In our case, model only consist of trainable projection matrices of size about 160K.

Our Attention Model is also related to Rocktaschel et al. [47]. In that work, the attention mechanism were used for recognizing textual entailment. A similar attention model was also used by Bahdanau et al. [3] for machine translation. Xu et al. [69] also proposed similar attention model that learns to describe the content of the image. Our Attention model uses an analogous attention mechanism but for language modeling task. However, we proposed a new architecture of Key-Value-Predict attention model, which gives more flexibility than basic attention model.

Damavandi et al. [15] presented a hybrid language model integrating n-grams and neural networks for speech recognition task. Thus, it combines generalization ability of neural networks and memorization capacity of n-gram model. This model has shown a 7% relative reduction in word error rate on their Italian dataset. The difference between their models and our is that they explicitly used n-grams counts as inputs to neural network, whereas we incorporate the history context of the network.

Chapter 4

# Experiments

In this chapter we evaluate the performance of proposed models against variety of state-of-the-art approaches. We use two public datasets: English Penn Treebank and Lambada [44]. Furthermore, we created our on dataset from Wikipedia articles. All experiments were run using TensorFlow library [1].

## 4.1 Datasets

Our experiments are performed on three different dataset. First, for historical reasons, we carried out the experiments on **English Penn Treebank** dataset (*PTB*). It is a relatively small corpus, but it is used in almost all language models comparison. The dataset contains about 1 million tokens and vocabulary size of 10K. The average length of the sentence is 21. We follow the common approach [39] and trained on sections 0-20 (1M words), used a sections 21-22 (80K words) for validation and 23-24 (90K words) for testing.

Second, recently proposed **LAMBADA** dataset [44] was used. The LAMBADA data set incorporate broad context of natural language text. It consists of passages composed of a context and target sentence. The context size, which is on average 4.6 sentences, is the minimum number of complete sentences before the target sentence such that they cumulatively contain at least 50 tokens. The LAMBADA dataset is created from Book Corpus [75]. The training corpus contains 2662 novels and 203 million words, while validation set contains about 361 000 words. The test set will be released at the time of the competition organised by them.

Third, we created our own data set from Wikipedia articles. We randomly drawn 7513 articles from Wikipedia corpus which are in one of the following category: People, Cities, Countries, Universities, Novels. The reason behind this categories is that this articles often refer to previous information and it incorporate a long context of the words. The training, validation and test sets consist of 22.5 mln, 1.2 mln and 1.2 mln words respectively. Dataset was preprocessed by removing all numbers to $N$ symbol (similar to PTB dataset). We also added a special token before and after each article, which allows to do not attend over words from previous articles while processing the current one. We restricted the vocabulary of the Wikipedia dataset to 77K most frequent words in the training set by replacing less common words by UNK symbol. It converges 97% words in both development and test sets. Table 4.1 summarizes the data used in our experiments:

**Table 4.1:** Datasets statistics - number of words in every set. $|V|$ denotes the size of the vocabulary.

| Dataset | Number of words | | | |
|---|---|---|---|---|
| | Train | Validation | Test | $\|V\|$ |
| PTB | 888K | 70K | 78K | 10K |
| LAMBADA | 203M | 361K | - | 77K |
| Wikipedia | 22.5M | 1.2M | 1.2M | 77K |

## 4.2 Training Procedure

We use ADAM [33] for optimization with a first momentum coefficient of 0.9, a second momentum coefficient of 0.999[2] (configuration recommended by Kingma and Ba [33]) and with learning rate of 0.001. The gradients was clipped during the training such their norm is bounded by 5 [45]. In all experiments the LSTM network were unrolled for 20 steps, and mini-batch size was set to 64 for PTB and Wikipedia dataset and to 96 for LAMBADA corpus. The bias of the LSTM's forget gate was initialized to 1 [31], while other parameters are initialized uniformly in range $(-0.1, 0.1)$.

Back-propagation Through time was used to train the network. For Wikipedia dataset, the hidden states of LSTMs are reset to zeros at the beginning of new article.

We also ensure that our models does not attend over words from previous article. We take the best configuration based on performance on the validation set and evaluate it on test set.

## 4.3 Results

In the following section we present our experiments in language modeling. First, we report the perplexity results for each of our models against state-of-the-art and baseline models. Second, we discuss their performance on predicting different categories of words such an nouns or verbs. Third, we analyse the attention distribution of our attentive models, and then we visualize and discuss the attention patterns of the presented attentive models.

### 4.3.1 Performance

In this set of experiments we compared proposed Recurrent Neural Network N-gram, Attention, Key-Value and Key-Value-Predict models against a variety of state-of-the-art models. First of all, we present perplexity results for the standard LSTM models with different number of layers. Furthermore, we implemented Recurrent-Memory model [61] with temporal matrix and gating composition function (RM+tM-g). Finally, we used a Tensorflow implementation of LSTMN architecture [11].

First, we compare the performance of our models to a range of state-of-the-art models on Wikipedia dataset. Table 4.2 summarizes the results. Embedding and hidden sizes are denoted by $w$ and $k$ respectively. The attention window, which is number of past output vectors used by attentive models, is denoted by $a$. The total number of model parameters, including word representations, is denoted by $\theta_{W+M}$ (without word representations $\theta_M$). The embedding and hidden size of models were chosen to have at most the same number of parameters as vanilla LSTM model. To ensure a comparable number of parameters to vanilla LSTM model, we have adjusted both size of the word vectors and hidden size of the network. The attention window and parameter $N$ for RNNN model were chosen so that they have the smallest perplexity on validation set.

**Table 4.2:** Perplexity results on development and test sets from Wikipedia corpus. Embedding size of the input word is denoted by $w$, the hidden size of the network by $k$, and the attention window by $a$. The total number of model parameters, including word representations, is denoted by $\theta_{W+M}$ (without word representations $\theta_M$).

| Model | w | k | a | $\theta_{W+M}$ | $\theta_M$ | Dev | Test |
|---|---|---|---|---|---|---|---|
| LSTM | 300 | 300 | - | 47.0M | 23.9M | 83.2 | 85.2 |
| LSTM layers 2 | 300 | 300 | - | 47.7M | 24.6M | 88.3 | 90.8 |
| LSTM layers 3 | 300 | 300 | - | 48.4M | 25.3M | 89.5 | 91.8 |
| RM(+tM-g) | 300 | 300 | 20 | 93.7M | 24.4M | 76.7 | 78.4 |
| LSTMN | 296 | 296 | 5 | 47.6M | 24.2M | 93.2 | 96.4 |
| RNN-3 | 266 | 1064 | - | 47.0M | 26.5M | 74.0 | **74.9** |
| Attention | 298 | 298 | 5 | 47.0M | 23.9M | 79.8 | 81.2 |
| Key-Value | 289 | 578 | 15 | 46.9M | 23.8M | 76.4 | 78.2 |
| Key-Value-Predict | 278 | 834 | 5 | 46.9M | 23.8M | **73.6** | 75.5 |

Results on the Wikipedia corpus of proposed attentive models with different attention window are summarized in Table 4.3. The parameters of models are the same as presented in Table 4.2. Finally, Table 4.4 summarizes the results of RNNN model with different parameter $N$. To ensure a comparable number of parameters to vanilla LSTM model, the embedding and hidden size of models were chosen to

**Table 4.3:** Perplexity results of attentive models on Wikipedia test set for different attention window.

| Attention window | 1 | 3 | 5 | 8 | 10 | 15 | 20 |
|---|---|---|---|---|---|---|---|
| Attention | **81.1** | 81.7 | 81.2 | 82.1 | 83.1 | 82.8 | 84.0 |
| Key-Value | 78.3 | 79.6 | 79.3 | 79.2 | 78.5 | **78.2** | 79.9 |
| Key-Value-Predict | 75.7 | 76.1 | **75.5** | 75.5 | 76.0 | 76.0 | 77.5 |
| RM(+tM-g) | 83.5 | 82.1 | 80.5 | 80.6 | 80.3 | 80.1 | **78.4** |
| LSTMN | 97.6 | 96.8 | **96.4** | 97.5 | 99.9 | 101.7 | 103.1 |

**Table 4.4:** Perplexity results of RNNN models on development and test sets from Wikipedia corpus for different parameter $N$. Embedding size of the input word is denoted by $w$, the hidden size of the network by $k$. The total number of model parameters, including word representations, is denoted by $\theta_{W+M}$ (without word representations $\theta_M$).

| Model | N | w | k | $\theta_{W+M}$ | $\theta_M$ | Dev | Test |
|---|---|---|---|---|---|---|---|
| RNN-1 | 1 | 290 | 580 | 47.0M | 24.7M | 76.1 | 78.0 |
| RNN-2 | 2 | 279 | 837 | 47.0M | 25.5M | 74.4 | 76.3 |
| RNN-3 | 3 | 266 | 1064 | 47.0M | 26.5M | **74.0** | **74.9** |
| RNN-4 | 4 | 253 | 1265 | 47.0M | 27.5M | 74.7 | 76.8 |

have at most the same number of parameters as vanilla LSTM model.

**RNNN** We found that even by small modification of Recurrent Neural Network such as incorporating Recurrent Neural Network 1-gram model, gives an improvement of 8.5 percentage points in perplexity over vanilla LSTM model. We argue that this is due giving the model ability to use one part of the memory to predict word after the next one. Specifically, the model use one part of the output vector to predict the current word, and the second part to predict the word after the current prediction. Moreover, tuning the parameter $N$ to 3, which allows to use combination of 3 past output vectors, gives an improvement of 12.1% over baseline LSTM model.

**Attention** By incorporating more complicated attention mechanism we found a 4.7 percentage point over vanilla LSTM but it does not outperform RNNN model. The attention model gives access to its previous output vectors and learns which previous outputs are useful to predict the next word. The motivation for that model is to enable it to learn a long term context dependencies. However, attending bigger number of output vectors does not improve the performance of the model. It leads to the conclusion that attention model does not learn a long term dependencies in the data. In fact, the model with attention window 1 have reached the lowest perplexity. Hence, attention model with windows size 1 does not learn attention distribution, because it uses only current and previous output vectors. We suspect that this is due to not enough size of the corpus and model is not able to learn a long term dependencies.

**Key-Value** Enabling the model to decompose the output vector into key-value paired memory improves the perplexity by 8.2 percentage points over baseline LSTM model and by 0.3 percentage point over state-of-the-art RM(+tM-g) model. Key-Value model is an extension of Attention model. It allows to use different memory parts to compute the attention distribution weights and to calculate weighted representation of previous output vectors. Similar to Attention Model, the performance of Key Value model does not improve significantly with increasing attention window.

**Key-Value-Predict** By incorporating Key-Value-Predict decomposition of output vectors we found a 11.4 percentage point improvement over a baseline LSTM model , and a 3.7 percentage point increase over RM(+tM-g) model which is considered as current state-of-the art model. Introducing the third part to the output vectors gives model more flexibility by using different memory to compute weighted representation of previous output vectors and to compute the final representation from final output vector. Therefore, when predicting the next word, the model use different parts of memory to three tasks : computing attention distribution, compute weighted representation of attended vectors and predicting the next word from final output vector. However, the performance of Key-Value-Predict model does not improve by significant margin with increasing the attention window. It is similar behaviour to our previous attentive models.

In summary, Recurrent Neural Network 3-gram model gives the biggest improvement over vanilla LSTM model. In addition, it also outperforms the state-of-the-art RM(+tM-g) model by 4.5 percentage points. Note that that RM(+tM-g) model uses almost two times more parameters that our models. It is due to it introduces two big trained lookup tables to learn representations of word embeddings. However, our attentive models does not improve the performance over RNNN model. It is surprising that attentive model does not significantly improve their performance with increasing the attention window. It seems that they do not learn a long term dependencies. However, we will show in the qualitative analysis (see 4.4) that they are able to capture long-term dependencies.

Then, we carry out experiments on LAMBADA dataset, which is about 10 times bigger than our own Wikipedia dataset, and incorporate broad context of natural language text. Test set is not released yet, so we use part of the training set as a validation set and actual development corpus as our test set. We also compare perplexities on control set which contains randomly sampled 5K passages of the same shape and size as the ones used to build test set, but without filtering them in any way. Results on the LAMBADA corpus are summarized in Table 4.5. Due to large size of the corpus and limited computational resources we report the performance

**Table 4.5:** Perplexity results on development, test and control sets from LAMBADA corpus. Embedding size of the input word is denoted by $w$, the hidden size of the network by $k$, and the attention window by $a$. The total number of model parameters, including word representations, is denoted by $\theta_{W+M}$ (without word representations $\theta_M$).

| Model | w | k | a | $\theta_{W+M}$ | $\theta_M$ | Dev | Test | Control |
|---|---|---|---|---|---|---|---|---|
| LSTM | 300 | 300 | - | 47.0M | 23.9M | 63.1 | 75.0 | 81.8 |
| LSTM layers 2 | 300 | 300 | - | 47.7M | 24.6M | 67.2 | 76.9 | 84.8 |
| LSTM layers 3 | 300 | 300 | - | 48.4M | 25.3M | 66.4 | 75.5 | 83.8 |
| RM(+tM-g) | 300 | 300 | 20 | 93.7M | 24.4M | 58.9 | 71.5 | 77.5 |
| LSTMN | 296 | 296 | 5 | 47.0M | 24.2M | 69.2 | 80.7 | 88.2 |
| RNN-4 | 253 | 1265 | - | 47.0M | 27.5M | **54.3** | **70.9** | **76.7** |
| Attention | 298 | 298 | 5 | 47.0M | 23.9M | 60.5 | 74.9 | 80.9 |
| Key-Value | 289 | 578 | 15 | 46.9M | 23.8M | 59.4 | 74.2 | 79.5 |
| Key-Value-Predict | 278 | 834 | 5 | 46.9M | 23.8M | 56.2 | 71.5 | 77.7 |

of different attention windows only for our best attentive model - Key-Value-Predict (see Table 4.6).

**Table 4.6:** Perplexity results of Key-Value-Predict model on LAMBADA test set for different attention window.

| Attention window | 1 | 3 | 5 | 10 | 15 | 20 |
|---|---|---|---|---|---|---|
| Dev | 57.5 | 57.3 | **56.2** | 61.2 | 61.3 | 60.1 |
| Test | 71.6 | 72.9 | **71.5** | 73.4 | 72.8 | 72.1 |
| Control | 77.8 | 78.3 | **77.7** | 81.5 | 80.9 | 77.9 |

The attention window for other models were chosen the same as in previous experiments on Wikipedia dataset. Finally, Table 4.7 summarizes the results of Recurrent Neural Network N-gram model with different parameter $N$.

**Table 4.7:** Perplexity results of RNNN models on development, test and control sets from LAMBADA corpus for different parameter $N$. Embedding size of the input word is denoted by $w$, the hidden size of the network by $k$. The total number of model parameters, including word representations, is denoted by $\theta_{W+M}$ (without word representations $\theta_M$).

| Model | N | w | k | $\theta_{W+M}$ | $\theta_M$ | Dev | Test | Control |
|---|---|---|---|---|---|---|---|---|
| RNN-1 | 1 | 290 | 580 | 47.0M | 24.7M | 57.3 | 73.6 | 78.7 |
| RNN-2 | 2 | 279 | 837 | 47.0M | 25.2M | 58.2 | 72.8 | 78.7 |
| RNN-3 | 3 | 266 | 1064 | 47.0M | 26.5M | 56.0 | 71.0 | 77.3 |
| RNN-4 | 4 | 253 | 1265 | 47.0M | 27.5M | **54.3** | **70.9** | **76.7** |

**RNNN** Allowing the model to use different parts of past output vectors to predict words on different position in the feature improves the performance of predicting the next word. With increasing the number of used past output vectors, the perplexity decreases. The best performance is achieved for Recurrent Neural Network 4-gram model. It decreases the perplexity by 5.5 and 6.2 percentage points on test and control set respectively over vanilla LSTM model. Moreover, it improves the performance over current state-of-the-art RM(+tM-g) model by 0.8 percentage point on test set and by 1 percentage point on control set.

**Attention** By incorporating the attention mechanism we found a 0.1 percentage point improvement on test set and 1.1 percentage point on control set over vanilla LSTM model. The attention model has worse performance than our other proposed models. We suspect that this is due to less flexibility of the model in language modeling. Hence, using the same memory for computing the attention distribution and for calculating the context representation of past output vectors might lead to noise in the training process.

**Key-Value** Enabling the model to use a Key-Value decomposition of the output vectors improves the performance by 1.0 and 2.8 percentage points on test and control set respectively over vanilla LSTM model. We argue that this is due to the model being able to use different memory parts to compute the attention distribution weights and to calculate weighted representation of previous output vectors.

**Key-Value-Predict** We found that incorporating Key-Value-Predict decomposition of output vectors decreases the perplexity by 4.7 percentage points on test set and by 5.0 on control set over LSTM model. However, it does not outperform the state-of-the-art RM(+tM-g) model. We suspect that this is due to the fact that RM(+tM-g) decreases the perplexity with increasing the attention window, whereas our Key-Value-Predict model does not improve the performance by significant margin with increasing the attention windows.

In summary, Recurrent Neural Network 4-gram models gives the biggest improvement and outperforms current state-of-the-art approaches on both test and control sets. Similar to results on Wikipedia dataset, increasing the attention window does not improve the performance of attentive language models.

Finally, for the historical reasons, we compare our models on English Penn Treebank dataset. This dataset is tiny - about 25 times smaller Wikipedia data set, and 225 times smaller than LAMBADA dataset. However, still many state-of-the-art papers compare their results only on this dataset. Table 4.8 summarizes the results of our models against state-of-the-art approaches on PTB dataset.

**Table 4.8:** Perplexity results on training, development and test sets from PTB corpus. Embedding size of the input word is denoted by $w$, the hidden size of the network by $k$, and the attention window by $a$. The total number of model parameters, including word representations, is denoted by $\theta_{W+M}$ (without word representations $\theta_M$).

| Model | w | k | a | $\theta_{W+M}$ | $\theta_M$ | Train | Dev | Test |
|---|---|---|---|---|---|---|---|---|
| LSTM | 300 | 300 | - | 6.7M | 3.7M | **55.7** | 122.8 | 116.3 |
| LSTM layers 2 | 300 | 300 | - | 7.5M | 4.5M | 75.0 | 141.9 | 131.8 |
| LSTM layers 3 | 300 | 300 | - | 8.2M | 5.2M | 99.0 | 160.0 | 148.7 |
| RM(+tM-g) | 300 | 300 | 20 | 13.3M | 4.3M | 57.5 | 121.1 | 115.6 |
| LSTMN | 277 | 277 | 5 | 6.7M | 3.9M | 89.0 | 151.4 | 145.9 |
| RNN-1 | 252 | 504 | - | 6.7M | 4.2M | 59.3 | 122.0 | 117.3 |
| Attention | 287 | 287 | 4 | 6.7M | 3.8M | 60.9 | **119.7** | **114.3** |
| Key-Value | 249 | 498 | 3 | 6.7M | 4.2M | 60.4 | 121.0 | 116.4 |
| Key-Value-Predict | 215 | 645 | 3 | 6.7M | 4.5M | 65.3 | 122.9 | 118.6 |

Results on the PTB corpus of proposed attentive models with different attention window are summarized in Table 4.9. The parameters of models are the same

**Table 4.9:** Perplexity results of attentive models on PTB test set for different attention window.

| Attention Window | 1 | 2 | 3 | 4 | 5 | 10 | 15 | 20 |
|---|---|---|---|---|---|---|---|---|
| Attention | 114.8 | 114.4 | 114.4 | **114.3** | 115.2 | 116.0 | 115.7 | 115.9 |
| Key-Value | 116.8 | 116.7 | **116.4** | 117.8 | 117.0 | 117.5 | 118.3 | 118.5 |
| Key-Value-Predict | 118.8 | 118.7 | 118.6 | **117.8** | 119.5 | 121.0 | 120.4 | 121.3 |
| RM (+Tm-g) | 117.8 | 118.1 | 117.2 | 117.3 | 117.2 | 116.4 | 115.7 | **115.6** |
| LSTMN | 154.6 | 147.8 | **145.1** | 145.9 | 145.1 | 146.2 | 145.8 | 147.5 |

as presented in Table 4.8. Finally, results of Recurrent Neural Network N-gram model with different parameter $N$ are summarized in Table 4.10

**RNNN** In contrast to performance on previous datasets, enabling the model to use different parts of output vector to predict words on different position in the

**Table 4.10:** Perplexity results of RNNN models on training, development and test sets from PTB corpus for different parameter $N$. Embedding size of the input word is denoted by $w$, the hidden size of the network by $k$. The total number of model parameters, including word representations, is denoted by $\theta_{W+M}$ (without word representations $\theta_M$).

| Model | N | w | k | $\theta_{W+M}$ | $\theta_M$ | Train | Dev | Test |
|---|---|---|---|---|---|---|---|---|
| RNN-1 | 1 | 252 | 504 | 6.7M | 4.2M | **59.3** | **122.0** | **117.3** |
| RNN-2 | 2 | 216 | 648 | 6.7M | 4.5M | 63.6 | 124.3 | 120.5 |
| RNN-3 | 3 | 188 | 752 | 6.7M | 4.8M | 67.6 | 126.6 | 121.8 |
| RNN-4 | 4 | 165 | 825 | 6.7M | 5.1M | 86.8 | 133.0 | 128.8 |
| RNN-5 | 5 | 147 | 882 | 6.7M | 5.2M | 76.5 | 133.7 | 129.3 |

future does not improve the performance of RNNN model over LSTM model. In addition, increasing the parameter $N$ decreases the performance of the model. We suspect that this is due to small amount of data. The second reason might be that to ensure the same number of parameters, we dramatically decrease size of word embeddings and size of final output vector representation. For example, for RNN-5 we decrease both sizes of word embeddings and final output vector from 300 to 147. As a result, these models have to represent words and output vectors by smaller number of parameters.

**Attention** Enabling the model to attend over previous output vectors improve the perplexity by 1.7 percentage point over LSTM model and outperform RM(+tM-g) by 1.1 percentage point. Again, increasing the size of attention window does not improve the performance by significant margin. The lowest perplexity gives the model which attends over 4 previous output vectors.

**Key-Value** Allowing the model to use different memory for computing attention distribution and for calculating the context representation of past output vectors does not improve the performance over baseline LSTM model. We suspect that the reason behind it is the same as for low performance of RNNN model - to ensure the same number of parameters we again decrease the size of embedding vectors and hidden output size from 300 to 249. It leads to not efficient representation of the word and output vector which seems to be crucial for PTB dataset.

**Key-Value-Predict** In contrast to results on previous datasets, incorporating the decomposition of the memory to key, value, predict parts, decreases the performance of the network. We suspect that this is again due to tuning the size of embedding and output vectors representation from 300 to 215. Thus model is not able to represent words efficiently.

To prove our hypothesis about weak performance of our models on PTB dataset, we run the same models without changing the size of word embeddings and final output vectors. The results are presented in Tables 4.11 Presented models

**Table 4.11:** Perplexity results of over parameterized models on training, development and test sets from PTB corpus. Embedding size of the input word is denoted by $w$, the hidden size of the network by $k$, and the attention window by $a$. The total number of model parameters, including word representations, is denoted by $\theta_{W+M}$ (without word representations $\theta_M$).

| Model | w | k | a | $\theta_{W+M}$ | $\theta_M$ | Train | Dev | Test |
|---|---|---|---|---|---|---|---|---|
| RNN-1 | 300 | 600 | - | 8.4M | 5.4M | 57.2 | **117.4** | **112.4** |
| RNN-2 | 300 | 900 | - | 10.6M | 7.6M | 61.4 | 118.1 | 115.1 |
| RNN-3 | 300 | 1200 | - | 13.6M | 10.6M | 60.7 | 120.6 | 115.8 |
| Attention | 300 | 300 | 4 | 7.1M | 4.1M | 59.7 | 119.1 | 114.0 |
| Key-Value | 300 | 600 | 3 | 8.5M | 5.5M | 63.0 | 119.7 | 114.5 |
| Key-Value-Predict | 300 | 900 | 4 | 10.7M | 7.7M | 65.1 | 118.0 | 114.0 |

outperform baseline LSTM model, but they are incomparable, because of different number of total parameters. Note the our models produces output vector of different size than hidden size of the LSTM network. For example, Key-Value-Predict model use only only one third of the output vectors to produce final representation.

### 4.3.2 Performance Analysis

As a next step towards comparison of different types of models, we look at the performance of predicting different types of words such as: nouns, verbs, adjectives, adverbs, personal pronouns, determiners. Results on Wikipedia corpus are summarized in Table 4.12. Note that the parameters of the model are the same as reported in Table 4.2.

We found that the most difficult is to predict are adjectives. Hence, they are very challenging for language models. Their meaning can vary depending on the context in which they appear. The performance of predicting adjectives is the best for Key-Value-Predict model, which outperforms baseline LSTM model by 16.5 percentage points. On the other hand, the easiest to predict are determiners. It is

**Table 4.12:** Perplexity results on predicting different category of words on Wikipedia test set.

| Model | Nouns | Verbs | Adjectives | Adverbs | Personal Pronouns | Determiners |
|---|---|---|---|---|---|---|
| LSTM | 650.9 | 299.7 | 1141.7 | 510.3 | 24.4 | 9.6 |
| LSTM layers 2 | 804.3 | 306.4 | 1234.8 | 511.0 | 25.7 | 9.4 |
| LSTM layers 3 | 843.2 | 301.3 | 1227.9 | 511.3 | 22.9 | 9.4 |
| RM(+tM-g) | 578.0 | 267.4 | 1039.2 | **456.8** | 24.4 | 9.4 |
| LSTMN | 807.1 | 342.0 | 1345.4 | 570.5 | 26.8 | 10.0 |
| RNN-3 | 558.9 | **250.3** | 967.6 | 458.6 | **20.3** | 8.9 |
| Attention | 619.4 | 274.5 | 1074.2 | 501.7 | 24.5 | **8.8** |
| Key-Value | 589.2 | 264.9 | 1070.3 | 493.6 | 22.3 | 9.2 |
| Key-Value-Predict | **555.9** | 264.7 | **953.7** | 468.2 | 22.4 | 8.8 |

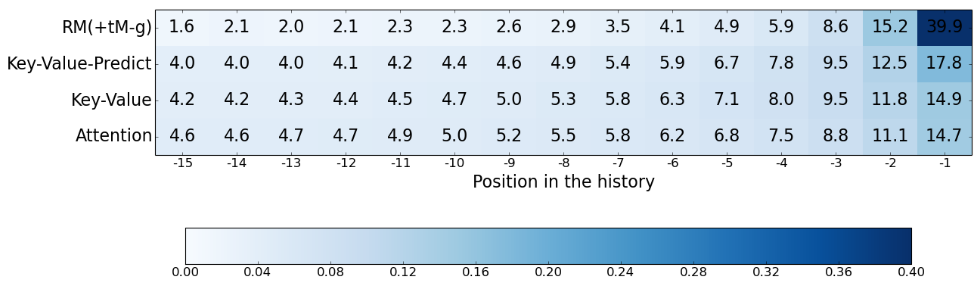
expected behaviour, because the amount of different determiners is small and their are always followed by noun. Finally, Recurrent Neural Network 3-gram model outperforms other models on predicting verbs by 16.5 percentage points over LSTM model, and on predicting personal pronouns by 16.8 percentage points.

## 4.4 Qualitative Analysis

As previous results have shown, increasing the size of attention windows does not significantly improve the performance of our attentive models. However, we found that it is instructive to analyse which output representations the model is attending over when predicting the next word. Note that interpretation of attention distribution have to be taken with care because attentive models are not forced to rely only on representation obtained from attention (see Eq. 3.7, 3.11, 3.15). In this section, we analyse attention distribution, as well as visualize and discuss the attention patterns of the presented attentive models. We use Wikipedia dataset and models with the same parameters as in Table 4.2, with the exception of the size of attention window which is set to 15.

### 4.4.1 Positional analysis

As a first step in analysing attention distribution, we look at the average attention weights of past output vectors positions (see Fig 4.1). Note that the RM(+tM-g) [61] model attends over input word embeddings starting from the current one, whereas



| | -15 | -14 | -13 | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RM(+tM-g) | 1.6 | 2.1 | 2.0 | 2.1 | 2.3 | 2.3 | 2.6 | 2.9 | 3.5 | 4.1 | 4.9 | 5.9 | 8.6 | 15.2 | 39.9 |
| Key-Value-Predict | 4.0 | 4.0 | 4.0 | 4.1 | 4.2 | 4.4 | 4.6 | 4.9 | 5.4 | 5.9 | 6.7 | 7.8 | 9.5 | 12.5 | 17.8 |
| Key-Value | 4.2 | 4.2 | 4.3 | 4.4 | 4.5 | 4.7 | 5.0 | 5.3 | 5.8 | 6.3 | 7.1 | 8.0 | 9.5 | 11.8 | 14.9 |
| Attention | 4.6 | 4.6 | 4.7 | 4.7 | 4.9 | 5.0 | 5.2 | 5.5 | 5.8 | 6.2 | 6.8 | 7.5 | 8.8 | 11.1 | 14.7 |

**Figure 4.1:** Average attention weights distribution on Wikipedia test set for RM(+tM-g), Attention, Key-Value and Key-Value-Predict models. The values denotes the percentage points of average attention weights. Rightmost positions represent most recent history.

our attentive models attend over output vectors starting from the previous one. The attention mass of all models tends to concentrate at the right most position (current input for RM(+tM-g), previous output vector for others) and decreases when moving further away in the history. However, about 40 % attention weights of RM(+tM-g) model concentrates on the current word, whereas only approximately 18 % attention weights of our best model, Key-Value-Predict, concentrates on the closest output vector. The worse performance of RM(+tM-g) can be explained by concentrating on current word, which is already used to produce current hidden state of the LSTM, so it does not deal with the memory compression problem. On the other hand, our attentive models more often attend over long distance output vectors. It means that proposed models learned to attend over further output vectors in the history which leads to capture long-term dependencies.

For Key-Value-Predict model we randomly sampled 70 sets of words from Wikipedia test set, and visualize attention distribution over last 15 output vectors (see Fig. 4.2). In many cases, attention weights concentrates on last few words, however many long distance words also receive noticeable attention weights. For many predicted words, attention weights are distributed evenly over past output vectors, which leads to the conclusion that in such cases LSTM hidden state contained enough information to predict the next word.

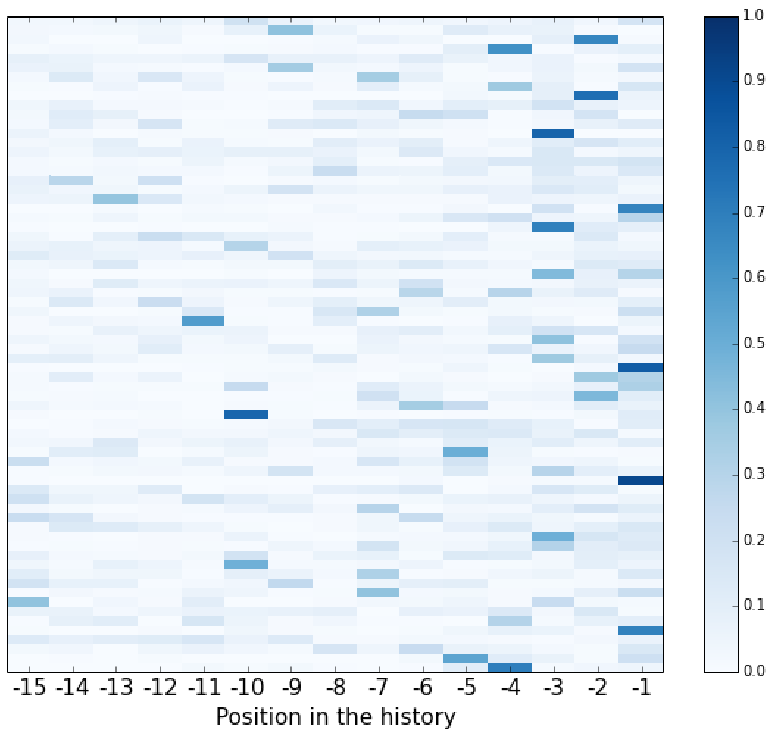


**Figure 4.2:** Visualization of attention weights for randomly chosen sequences from Wikipedia test set for Key-Value-Predict model. Rightmost positions represent most recent history.

### 4.4.2 Linguistic phenomena analysis

We manually inspect examples that shows certain linguistic phenomena. Figures 4.3-4.5 show to what extent the attentive models focuses on contextual representations of the sequences. Note that the models attend over output vector which are produced after processing visualized words. We also present the perplexity of predicting the next word and top five word predictions with with the respective log-probabilities for different models. We found that our attentive models are able to capture some long term dependencies. Example (1) from Fig. 4.3 shows that our Attention, Key-Value and RNN-3 models are able to capture information about gender of the subject. The attentive models focus on words *Anne Marie* while predicting the correct word *her* which refers to attended words. However, LSTM model and Key-Value-Predict have worse performance in this case. Furthermore, example (2) shows the ability of focusing on broad context of the sequence. Recurrent Neural Network N-gram and Attentive models are able assign bigger probability to the correct word than LSTM model. They capture the information that correct prediction *mother* refers to previously mentioned word *father*.

Figure 4.4 illustrates examples that capture distant dependencies. Both examples (3) and (4) show that our attentive models refer back to distant word in the history while predicting the correct one. Our proposed models outperform vanilla LSTM model on predicting the next word on the sequence (3) by significant margin. The attentive models are able to predict the correct word *mosque* which was previously mentioned. Example (4) shows that our attentive models correctly assign a big probability to the close bracket and focus on distant open bracket, whereas vanilla LSTM and RNN-3 models do not capture this information.

Finally, Figure 4.5 shows results that capture morphological dependencies. Both examples (5) and (6) show that attentive models are able to attend over distant plural noun which refers to the predicted word *were*. On the other hand, LSTM model does not capture that it refers to plural noun and as a results assign bigger probability to the word *was*.

Attention Model

... pastor at UNK and the family moved there . Anne Marie UNK was from UNK ;

Key-Value Model

... pastor at UNK and the family moved there . Anne Marie UNK was from UNK ;

Key-Value-Predict Model

... pastor at UNK and the family moved there . Anne Marie UNK was from UNK ;

(1)

Correct next word: **her**
Perplexity:
LSTM: 42.2 Attention: **5.9** Key-Value: 8.2 Key-Value-Predict: 33.0 RNN-3: 12.5

Top 5 predictions:

| | | | | | |
|---|---|---|---|---|---|
| LSTM | he (-3.16) | his (-3.24) | in (-3.66) | the (-3.71) | and (-3.87) |
| Attention | **her** (-2.57) | she (-2.70) | the (-3.83) | and (-3.90) | in (-5.61) |
| Key-Value | she (-2.34) | **her** (-3.03) | and (-3.35) | the (-4.02) | Maria (-4.39) |
| Key-Value-Predict | the (-2.91) | in (-3.86) | UNK (-4.02) | and (-4.13) | they (-4.62) |
| RNN-3 | she (-1.80) | the (-3.56) | **her** (-3.64) | he (-3.69) | and (-4.32) |

Attention Model

... David ) and John Mustaine . His father was of French and Finnish descent and his

Key-Value Model

... David ) and John Mustaine . His father was of French and Finnish descent and his

Key-Value-Predict Model

... David ) and John Mustaine . His father was of French and Finnish descent and his

(2)

Correct next word: **mother**
Perplexity:
LSTM: 3.0 Attention: 1.8 Key-Value: **1.2** Key-Value-Predict: 1.6 RNN-3: 1.9

Top 5 predictions:

| | | | | | |
|---|---|---|---|---|---|
| LSTM | father (-1.25) | **mother** (-1.59) | family (-4.27) | younger (-5.05) | wife (-5.43) |
| Attention | **mother** (-0.84) | father (-3.13) | grandfather (-4.02) | son (-4.62) | brother (-5.41) |
| Key-Value | **mother** (-0.24) | father (-3.88) | grandfather (-6.02) | wife (-6.77) | paternal (-7.70) |
| Key-Value-Predict | **mother** (-0.69) | father (-3.13) | maternal (-4.28) | grandfather (-4.51) | parents (-5.67) |
| RNN-3 | **mother** (-0.95) | father (-2.63) | paternal (-3.77) | parents (-4.53) | maternal (-4.71) |

Legend: 0 10 20 30 40 50 60 70 80 90 100

**Figure 4.3:** Results on manually selected examples that capture information about gender of the subject. It shows to what extent the attentive models focuses on contextual representations of the sequence. The perplexities and top five word predictions with with the respective log-probabilities for different models are presented. Legend shows the percentage points of attending weights.

Attention Model

... mosque in Texas is located in Denton , about north of Downtown Dallas . The oldest

Key-Value Model

... mosque in Texas is located in Denton , about north of Downtown Dallas . The oldest

Key-Value-Predict Model

... mosque in Texas is located in Denton , about north of Downtown Dallas . The oldest

(3)

Correct next word: **mosque**
Perplexity:
LSTM: 49.2 Attention: 6.1 Key-Value: 4.5 Key-Value-Predict: **2.7** RNN-3: 5.1

Top 5 predictions:

| | | | | | |
|---|---|---|---|---|---|
| LSTM | extant (-3.91) | and (-4.11) | city (-5.09) | surviving (-5.13) | building (-5.14) |
| Attention | **mosque** (-2.62) | is (-3.38) | building (-4.12) | in (-5.18) | was (-5.24) |
| Key-Value | **mosque** (-2.17) | building (-3.45) | church (-4.10) | synagogue (-5.04) | golf (-5.61) |
| Key-Value-Predict | **mosque** (-1.44) | , (-4.26) | synagogue (-4.56) | building (-4.78) | church (-4.81) |
| RNN-3 | building(-2.31) | **mosque** (-2.36) | church (-3.62) | community (-5.01) | synagogue (-5.08) |

Attention Model

... Glory ) [ first medal on wearer 's right depicted in photo of Morse with medals

Key-Value Model

... Glory ) [ first medal on wearer 's right depicted in photo of Morse with medals

Key-Value-Predict Model

... Glory ) [ first medal on wearer 's right depicted in photo of Morse with medals

(4)

Correct next word: **]**
Perplexity:
LSTM: 1119.8 Attention: **3.3** Key-Value: 8.8 Key-Value-Predict: 4.3 RNN-3: 330.0

Top 5 predictions:

| | | | | | |
|---|---|---|---|---|---|
| LSTM | . (-2.64) | , (-2.72) | and (-2.73) | in (-3.31) | of (-4.34) |
| Attention | ] (-1.73) | ) (-2.17) | on (-3.47) | in (-4.25) | and (-4.81) |
| Key-Value | in (-2.46) | ] (-3.13) | and (-3.16) | , (-3.77) | ) (-3.93) |
| Key-Value-Predict | ) (-2.06) | ] (-2.10) | , (-3.39) | and (-3.64) | on (-4.70) |
| RNN-3 | and (–3.41) | in (-3.56) | . (-3.62) | , (-3.63) | of (-3.67) |

Legend: 0 10 20 30 40 50 60 70 80 90 100

**Figure 4.4:** Results on manually selected examples that capture distant dependencies. It shows to what extent the attentive models focuses on contextual representations of the sequence. The perplexities and top five word predictions with with the respective log-probabilities for different models are presented. Legend shows the percentage points of attending weights.

Attention Model

His works , the greater part of which originally appeared in “ Crelle ’s Journal ” ,

Key-Value Model

His works , the greater part of which originally appeared in “ Crelle ’s Journal ” ,

Key-Value-Predict Model

His works , the greater part of which originally appeared in “ Crelle ’s Journal ” ,

(5)

Correct next word: **were**
Perplexity:
LSTM: 17.8 Attention: 6.8 Key-Value: 3.8 Key-Value-Predict: **1.9** RNN-3: 8.1

Top 5 predictions:

| | | | | | |
|---|---|---|---|---|---|
| LSTM | “ (-3.37) | was (-3.57) | published (-4.12) | **were** (-4.15) | and (-4.83) |
| Attention | **were** (-2.77) | are (-3.37) | was (-3.40) | and (-3.65) | have (-4.21) |
| Key-Value | **were** (-1.91) | was (-3.24) | are (-3.89) | included (-5.08) | is (-5.24) |
| Key-Value-Predict | **were** (-0.92) | are (-3.31) | was (-4.91) | have (-5.30) | had (-6.47) |
| RNN-3 | **were** (-2.04) | was (-3.51) | had (-3.77) | became (-5.07) | are (-5.30) |

Attention Model

and American flags . The first words transmitted by Samuel Morse that day in Puerto Rico

Key-Value Model

and American flags . The first words transmitted by Samuel Morse that day in Puerto Rico

Key-Value-Predict Model

and American flags . The first words transmitted by Samuel Morse that day in Puerto Rico

(6)

Correct next word: **were**
Perplexity:
LSTM: 9.9 Attention: 7.3 Key-Value: 14.4 Key-Value-Predict: **3.0** RNN-3: 7.2

Top 5 predictions:

| | | | | | |
|---|---|---|---|---|---|
| LSTM | , (-2.94) | was (-3.08) | **were** (-3.31) | in (-3.62) | had (-4.42) |
| Attention | **were** (-2.87) | was (-2.92) | , (-3.03) | are (-3.10) | is (-3.86) |
| Key-Value | , (-3.18) | ’s (-3.44) | **were** (-3.85) | and (-4.71) | found (-5.14) |
| Key-Value-Predict | **were** (-1.56) | are (-2.39) | , (-3.89) | is (-5.00) | was (-5.02) |
| RNN-4 | **were** (-2.85) | ’s (-2.89) | are (-3.01) | is (-3.87) | , (-4.13) |

Legend: 0 10 20 30 40 50 60 70 80 90 100

**Figure 4.5:** Results on manually selected examples that capture morphological dependencies. It shows to what extent the attentive models focuses on contextual representations of the sequence. The perplexities and top five word predictions with with the respective log-probabilities for different models are presented. Legend shows the percentage points of attending weights.

Chapter 5

# Conclusions

In this thesis we have proposed four novel recurrent neural network architectures for language modeling, namely Recurrent Neural Network N-gram, Attention, Key-Value and Key-Value-Predict models. All models were trained on LAMBADA, PTB and our own Wikipedia dataset.

Overall, the results of the thesis have been very satisfying. First, we have shown that incorporating the history context to recurrent neural network improves the performance of language model. All our models compute the new representation of the final output vector using the previous output vectors of LSTM. We designed a Recurrent Neural Network N-gram model and three more complicated attentive models: Attention, Key-Value and Key-Value-Predict. Interestingly, we have presented that our simple modification of recurrent neural network, RNNN model, achieve the same performance as our the best attentive model - Key-Value-Predict and they both outperforms other models by significant margin.

Moreover we have proved that our Key-Value-Predict structure gives more flexibility in predicting the next word and it improves the performance over Key-Value paired memory. Key-Value-Predict model use different parts of memory while predicting the next word: the attention distribution is computed based on *key* memory, the context weighted representation of attended vectors is calculated from *value* memory, and the the next word is predicted from *predict* part of current output vector. We suspect that this is due to giving model more flexibility and model is able to learn three different calculations simultaneously.

In addition, we have visualized that our attentive models are able to capture certain linguistic phenomena. It allows us to discover which words are important for word prediction. For example, we can capture some syntactic dependencies such as open and close brackets. Another examples have shown that attentive models are able to focus on distant noun, for instance *works*, while predicting the word *were* which refers to attended noun.

As the extension of present work, it would be worthwhile exploring how using the mixture of our Recurrent Neural Network N-gram and Key-Value-Predict model will improve the performance of language model. Both models achieve competitive results, but they use different techniques. In addition, we can apply some heuristics to force our attentive models to focus on more distant words. For example, we can allow attentive models to attend only capitalized words, which we believe might be more important for predicting the new one. Another example can be to do not allow attentive models to attend over $N$ previous words. Finally, it would be interesting to explore how our models could help to improve the performance of related tasks such as machine translation or text summarization.

# Bibliography

[1] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Christopher G Atkeson and Stefan Schaal. Memory-based neural networks for robot learning. *Neurocomputing*, 9(3):243–269, 1995.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] Lalit R Bahl, Peter F Brown, Peter V de Souza, and Robert L Mercer. A tree-based statistical language model for natural language speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(7):1001–1008, 1989.

[5] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *journal of machine learning research*, 3(Feb):1137–1155, 2003.

[6] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[7] Samuel R Bowman, Christopher D Manning, and Christopher Potts. Tree-structured composition in neural networks without tree-structured architectures. *arXiv preprint arXiv:1506.04834*, 2015.

[8] Peter F Brown, John Cocke, Stephen A Della Pietra, Vincent J Della Pietra, Fredrick Jelinek, John D Lafferty, Robert L Mercer, and Paul S Roossin. A statistical approach to machine translation. *Computational linguistics*, 16(2):79–85, 1990.

[9] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

[10] Yanqing Chen, Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. The expressive power of word embeddings. *arXiv preprint arXiv:1301.3226*, 2013.

[11] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*, 2016.

[12] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[13] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems*, pages 577–585, 2015.

[14] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[15] Babak Damavandi, Shankar Kumar, Noam Shazeer, and Antoine Bruguier. Nn-grams: Unifying neural network and n-gram language models for speech recognition. *arXiv preprint arXiv:1606.07470*, 2016.

[16] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[17] Katja Filippova, Enrique Alfonseca, Carlos Colmenares, Lukasz Kaiser, and Oriol Vinyals. Sentence compression by deletion with lstms. 2015.

[18] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[19] Joshua T Goodman. A bit of progress in language modeling. *Computer Speech & Language*, 15(4):403–434, 2001.

[20] Joshua T Goodman. A bit of progress in language modeling. *Computer Speech & Language*, 15(4):403–434, 2001.

[21] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[22] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[23] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[24] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *arXiv preprint arXiv:1503.04069*, 2015.

[25] GE Hinton, DE Rumelhart, and RJ Williams. Learning internal representations by back-propagating errors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, 1, 1985.

[26] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[27] Herbert Jaeger. Discovering multiscale dynamical features with hierarchical echo state networks. *Jacobs University Bremen, Tech. Rep*, 2007.

[28] Frederick Jelinek. *Statistical methods for speech recognition.* MIT press, 1997.

[29] Shihao Ji, SVN Vishwanathan, Nadathur Satish, Michael J Anderson, and Pradeep Dubey. Blackout: Speeding up recurrent neural network language models with very large vocabularies. *arXiv preprint arXiv:1511.06909*, 2015.

[30] T Joshua and JT Goodman. A bit of progress in language modeling extended version. *Machine Learning and Applied Statistics Group Microsoft Research. Technical Report, MSR-TR-2001-72*, 2001.

[31] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

[32] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[33] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[34] Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 181–184. IEEE, 1995.

[35] Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. A clockwork rnn. *arXiv preprint arXiv:1402.3511*, 2014.

[36] Tomáš Mikolov. Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April*, 2012.

[37] Tomas Mikolov, Anoop Deoras, Stefan Kombrink, Lukas Burget, and Jan Cernockỳ. Empirical evaluation and combination of advanced language modeling techniques. In *INTERSPEECH*, number s 1, pages 605–608, 2011.

[38] Tomas Mikolov, Armand Joulin, Sumit Chopra, Michael Mathieu, and Marc'Aurelio Ranzato. Learning longer memory in recurrent neural networks. *arXiv preprint arXiv:1412.7753*, 2014.

[39] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

[40] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, volume 13, pages 746–751, 2013.

[41] Tomas Mikolov and Geoffrey Zweig. Context dependent recurrent neural network language model. In *SLT*, pages 234–239, 2012.

[42] Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. *arXiv preprint arXiv:1606.03126*, 2016.

[43] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems*, pages 2204–2212, 2014.

[44] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and

Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

[45] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318, 2013.

[46] Gerasimos Potamianos and Frederick Jelinek. A study of n-gram and decision tree letter language modeling methods. *Speech Communication*, 24(3):171–192, 1998.

[47] Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, and Phil Blunsom. Reasoning about entailment with neural attention. *arXiv preprint arXiv:1509.06664*, 2015.

[48] Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*, 2015.

[49] Jürgen Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992.

[50] Holger Schwenk. Continuous space language models. *Computer Speech & Language*, 21(3):492–518, 2007.

[51] Iulian V Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. Hierarchical neural network generative models for movie dialogues. *arXiv preprint arXiv:1507.04808*, 2015.

[52] Noam M Shazeer, Joris Pelemans, and Ciprian Chelba. Sparse non-negative matrix language modeling for skip-grams. 2015.

[53] Nitish Srivastava. *Improving neural networks with dropout*. PhD thesis, University of Toronto, 2013.

[54] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using lstms. *CoRR, abs/1502.04681*, 2, 2015.

[55] Karl Steinbuch and UAW Piske. Learning matrices and their applications. *IEEE Transactions on Electronic Computers*, (6):846–862, 1963.

[56] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448, 2015.

[57] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Interspeech*, pages 194–197, 2012.

[58] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011.

[59] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[60] WK Taylor. Pattern recognition by means of automatic analogue apparatus. *Proceedings of the IEE-Part B: Radio and Electronic Engineering*, 106(26):198–209, 1959.

[61] Ke Tran, Arianna Bisazza, and Christof Monz. Recurrent memory network for language modeling. *arXiv preprint arXiv:1601.01272*, 2016.

[62] Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics, 2010.

[63] Ashish Vaswani, Yinggong Zhao, Victoria Fossum, and David Chiang. Decoding with large-scale neural language models improves translation. In *EMNLP*, pages 1387–1392. Citeseer, 2013.

[64] Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pages 2773–2781, 2015.

[65] Bingning Wang, Kang Liu, and Jun Zhao. Inner attention based recurrent neural networks for answer selection.

[66] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[67] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

[68] Will Williams, Niranjani Prasad, David Mrva, Tom Ash, and Tony Robinson. Scaling recurrent neural network language models. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5391–5395. IEEE, 2015.

[69] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2(3):5, 2015.

[70] Peng Xu and Frederick Jelinek. Random forests and the data sparseness problem in language modeling. *Computer Speech & Language*, 21(1):105–152, 2007.

[71] Wojciech Zaremba. An empirical exploration of recurrent network architectures. 2015.

[72] Wojciech Zaremba and Ilya Sutskever. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014.

[73] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[74] Yin Zheng, Richard S Zemel, Yu-Jin Zhang, and Hugo Larochelle. A neural autoregressive approach to attention-based recognition. *International Journal of Computer Vision*, 113(1):67–79, 2015.

[75] Yukun Zhu, Ryan Kiros, Rich 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 *Proceedings of the IEEE International Conference on Computer Vision*, pages 19–27, 2015.