Unknown date | Unknown author

Modern Deep Learning Techniques Applied to Natural Language Processing by Authors

Chapter 1

Introduction

Natural language processing (NLP) is a theory-motivated range of computational techniques for the automatic analysis and representation of human language. NLP research has evolved from the era of punch cards and batch processing, in which the analysis of a sentence could take up to 7 minutes, to the era of Google and the likes of it, in which millions of webpages can be processed in less than a second (Cambria and White, 2014). NLP enables computers to perform a wide range of natural language related tasks at all levels, ranging from parsing and part-of-speech (POS) tagging, to machine translation and dialogue systems.

Deep learning architectures and algorithms have already made impressive advances in fields such as computer vision and pattern recognition. Following this trend, recent NLP research is now increasingly focusing on the use of new deep learning methods (see Figure 1). For decades, machine learning approaches targeting NLP problems have been based on shallow models (e.g., SVM and logistic regression) trained on very high dimensional and sparse features. In the last few years, neural networks based on dense vector representations have been producing superior results on various NLP tasks. This trend is sparked by the success of word embeddings (Mikolov et al., 2010, 2013a) and deep learning methods (Socher et al., 2013). Deep learning enables multi-level automatic feature representation learning. In contrast, traditional machine learning based NLP systems liaise heavily on hand-crafted features. Such hand-crafted features are time-consuming and often incomplete.

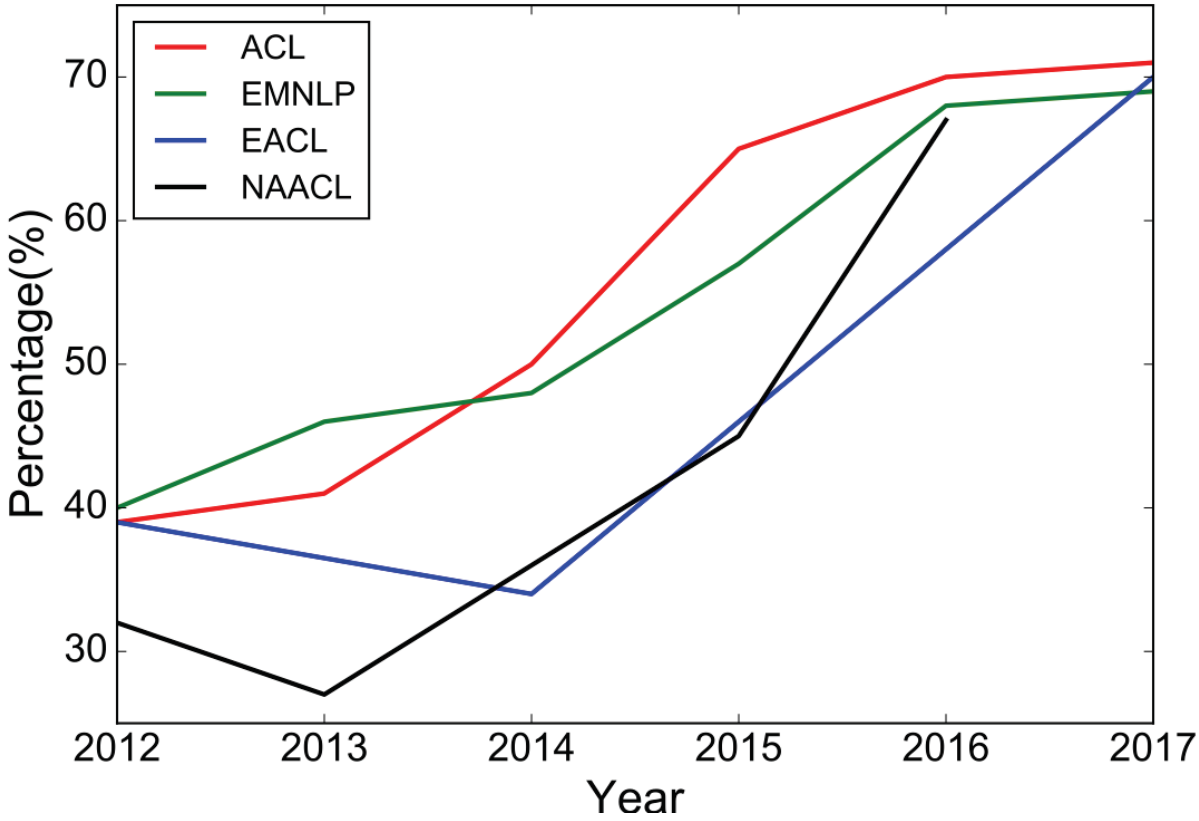


Figure 1: Percentage of deep learning papers in ACL, EMNLP, EACL, NAACL over the last 6 years (long papers).

Collobert et al. (2011) demonstrated that a simple deep learning framework outperforms most state-of-the-art approaches in several NLP tasks such as named-entity recognition (NER), semantic role labeling (SRL), and POS tagging. Since then, numerous complex deep learning based algorithms have been proposed to solve difficult NLP tasks. We review major deep learning related models and methods applied to natural language tasks such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), and recursive neural networks. We also discuss memory-augmenting strategies, attention mechanisms and how unsupervised models, reinforcement learning methods and recently, deep generative models have been employed for language-related tasks.

To the best of our knowledge, this work is the first of its type to comprehensively cover the most popular deep learning methods in NLP research today. The work by Goldberg (2016) only presented the basic principles for applying neural networks to NLP in a tutorial manner. We believe this paper will give readers a more comprehensive idea of current practices in this domain.

The structure of the paper is as follows: Section 2 introduces the concept of distributed representation, the basis of sophisticated deep learning models; next, Sections 3, 4, and 5 discuss popular models such as convolutional, recurrent, and recursive neural networks, as well as

their use in various NLP tasks; following, Section 6 lists recent applications of reinforcement learning in NLP and new developments in unsupervised sentence representation learning; later, Section 7 illustrates the recent trend of coupling deep learning models with memory modules; finally, Section 8 summarizes the performance of a series of deep learning methods on standard datasets about major NLP topics.

Chapter 2

Distributed Representation

Statistical NLP has emerged as the primary option for modeling complex natural language tasks. However, in its beginning, it often used to suffer from the notorious *curse of dimensionality* while learning joint probability functions of language models. This led to the motivation of learning distributed representations of words existing in low-dimensional space (Bengio et al., 2003).

A. Word Embeddings

Distributional vectors or word embeddings (Figure 2) essentially follow the distributional hypothesis, according to which words with similar meanings tend to occur in similar context. Thus, these vectors try to capture the characteristics of the neighbors of a word. The main advantage of distributional vectors is that they capture similarity between words. Measuring similarity between vectors is possible, using measures such as cosine similarity. Word embeddings are often used as the first data processing layer in a deep learning model. Typically, word embeddings are pre-trained by optimizing an auxiliary objective in a large unlabeled corpus, such as predicting a word based on its context (Mikolov et al., 2013b, a), where the learned word vectors can capture general syntactical and semantic information. Thus, these embeddings have proven to be efficient in capturing context similarity, analogies and due to its smaller dimensionality, are fast and efficient in computing core NLP tasks.

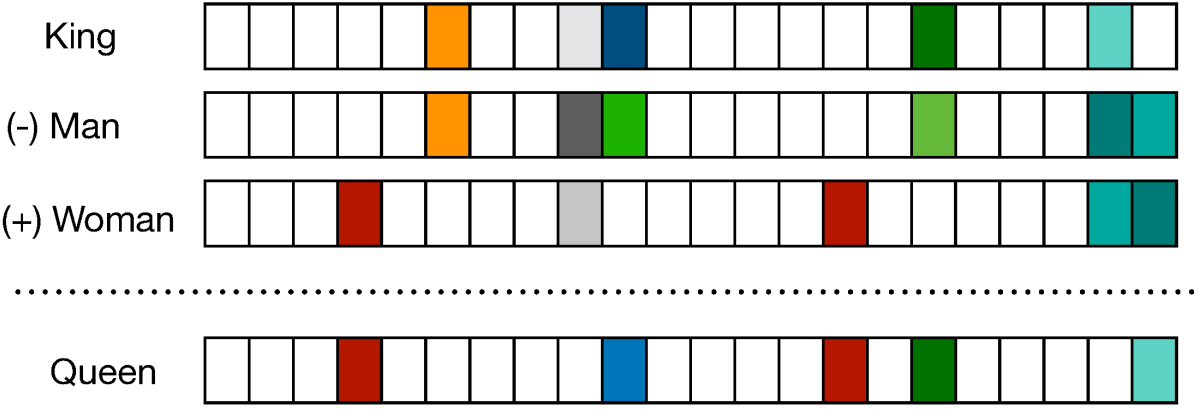


Figure 2: Distributional vectors represented by a -dimensional vector where , where is size of Vocabulary. Figure Source: http://veredshwartz.blogspot.sg.

Over the years, the models that create such embeddings have been shallow neural networks and there has not been need for deep networks to create good embeddings. However, deep learning based NLP models invariably represent their words, phrases and even sentences using these embeddings. This is in fact a major difference between traditional word count based models and deep learning based models. Word embeddings have been responsible for state-of-the-art results in a wide range of NLP tasks (Bengio and Usunier, 2011; Socher et al., 2011; Turney and Pantel, 2010; Cambria et al., 2017).

For example, Glorot et al. (2011) used embeddings along with stacked denoising autoencoders for domain adaptation in sentiment classification and Hermann and Blunsom (2013) presented combinatory categorical autoencoders to learn the compositionality of sentence. Their wide usage across the recent literature shows their effectiveness and importance in any deep learning model performing a NLP task.

Distributed representations (embeddings) are mainly learned through context. During 1990s, several research developments (Elman, 1991) marked the foundations of research in distributional semantics. A more detailed summary of these early trends is provided in (Glenberg and Robertson, 2000; Dumais, 2004). Later developments were adaptations of these early works, which led to creation of topic models like latent Dirichlet allocation (Blei et al., 2003) and language models (Bengio et al., 2003). These works laid out the foundations of representation learning.

In 2003, Bengio et al. (2003) proposed a neural language model which learned distributed representations for words (Figure 3). Authors argued that these word representations, once compiled into sentence

representations using joint probability of word sequences, achieved an exponential number of semantically neighboring sentences. This, in turn, helped in generalization since unseen sentences could now gather higher confidence if word sequences with similar words (in respect to nearby word representation) were already seen.

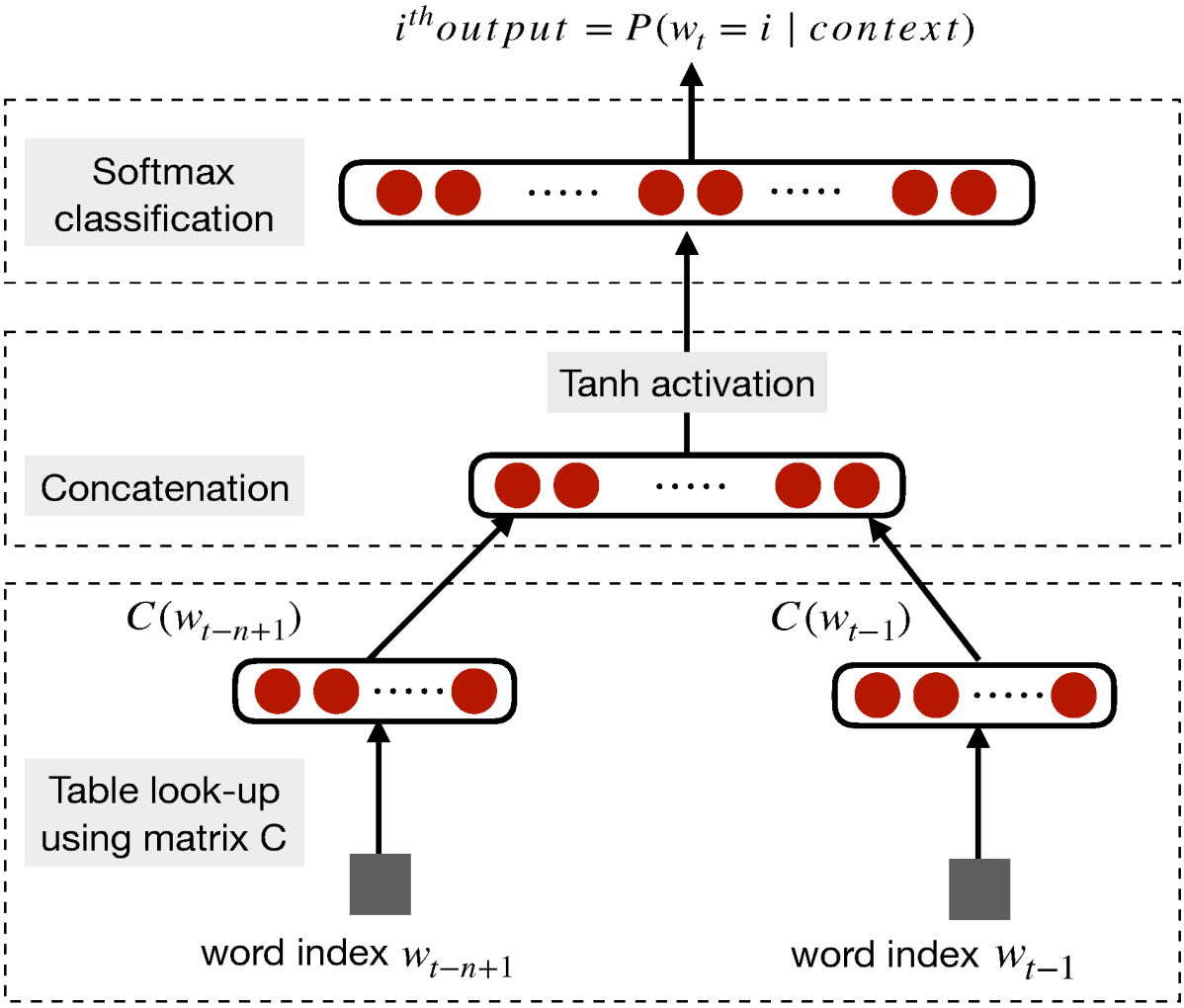


Figure 3: Neural Language Model (Figure reproduced from Bengio et al. (2003)). is the word embedding.

Collobert and Weston (2008) was the first work to show the utility of pre-trained word embeddings. The authors proposed a neural network architecture that forms the foundation to many current approaches. The work also establishes word embeddings as a useful tool for NLP tasks. However, the immense popularization of word embeddings was arguably due to (Mikolov et al., 2013), who proposed the continuous bag-of-words (CBOW) and skip-gram models to efficiently construct high-quality distributed vector representations. Propelling their popularity was the unexpected side effect of the vectors exhibiting compositionality, i.e., adding two word vectors results in a vector that is a semantic composite of the individual words, e.g., 'man' + 'royal' = 'king'. The theoretical justification for this behavior was recently given by Gittens et al. (2017), which stated that compositionality is seen only

when certain assumptions are held, e.g., the assumption that words need to be uniformly distributed in the embedding space.

Pennington et al. (2014) is another famous word embedding method which is essentially a "count-based" model. Here, the word co-occurrence count matrix is preprocessed by normalizing the counts and log-smoothing them. This matrix is then factorized to get lower dimensional representations which is done by minimizing a "reconstruction loss".

Below, we provide a brief description of the word2vec method proposed by Mikolov et al., (2013).

B. Word2vec

Word embeddings were revolutionized by Mikolov et al. (2013b, a) who proposed the CBOW and skip-gram models. CBOW computes the conditional probability of a target word given the context words surrounding it across a window of size . On the other hand, the skip-gram model does the exact opposite of the CBOW model, by predicting the surrounding context words given the central target word. The context words are assumed to be located symmetrically to the target words within a distance equal to the window size in both directions. In unsupervised settings, the word embedding dimension is determined by the accuracy of prediction. As the embedding dimension increases, the accuracy of prediction also increases until it converges at some point, which is considered the optimal embedding dimension as it is the shortest without compromising accuracy.

Let us consider a simplified version of the CBOW model where only one word is considered in the context. This essentially replicates a bigram language model.

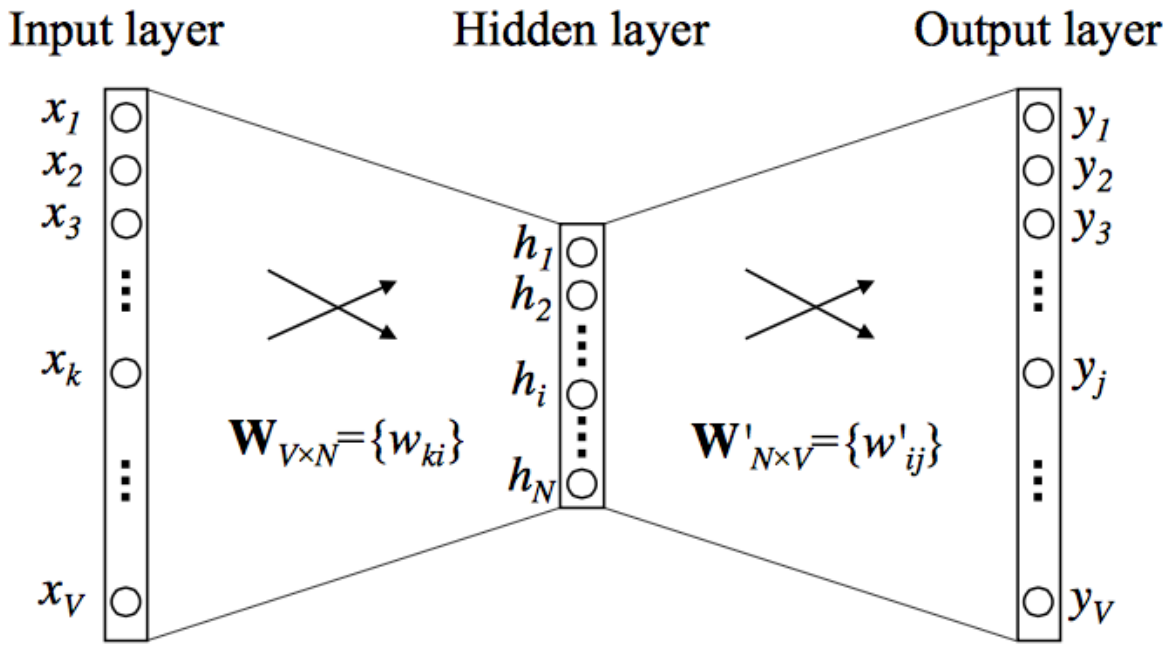


Figure 4: Model for CBOW (Figure source: Rong (2014))

As shown in Figure 4 , the CBOW model is a simple fully connected neural network with one hidden layer. The input layer, which takes the one-hot vector of context word has neurons while the hidden layer has neurons. The output layer is softmax of all words in the vocabulary. The layers are connected by weight matrix and , respectively. Each word from the vocabulary is finally represented as two learned vectors and , corresponding to context and target word representations, respectively. Thus, word in the vocabulary will have

Overall, for any word with given context word as input,

The parameters are learned by defining the objective function as the log-likelihood and finding its gradient as

In the general CBOW model, all the one-hot vectors of context words are taken as input simultaneously, i.e,

One limitation of individual word embeddings is their inability to represent phrases (Mikolov et al., 2013), where the combination of two or more words (e.g., idioms like “hot potato” or named entities such as “Boston Globe”) does not represent the combination of meanings of individual words. One solution to this problem, as explored by Mikolov et al. (2013), is to identify such phrases based on word co-occurrence and train embeddings for them separately. More recent methods have explored directly learning n-gram embeddings from unlabeled data (Johnson and Zhang, 2015).

Another limitation comes from learning embeddings based only on a small window of surrounding words, sometimes words such as *good* and *bad* share almost the same embedding (Socher et al., 2011), which is problematic if used in tasks such as sentiment analysis (Wang et al., 2015). At times these embeddings cluster semantically-similar words which have opposing sentiment polarities. This leads the downstream model used for the sentiment analysis task to be unable to identify this contrasting polarities leading to poor performance. Tang et al. (2014) addresses this problem by proposing sentiment specific word embedding (SSWE). Authors incorporate the supervised sentiment polarity of text in their loss functions while learning the embeddings.

A general caveat for word embeddings is that they are highly dependent on the applications in which it is used. Labutov and Lipson (2013) proposed task specific embeddings which retrain the word embeddings to align them in the current task space. This is very important as training embeddings from scratch requires large amount of time and resource. Mikolov et al. (2013) tried to address this issue by proposing *negative sampling* which is nothing but frequency-based sampling of negative terms while training the word2vec model.

Traditional word embedding algorithms assign a distinct vector to each word. This makes them unable to account for polysemy. In a recent work, Upadhyay et al. (2017) provided an innovative way to address this deficit. The authors leveraged multilingual parallel data to learn multi-sense word embeddings. For example, the English word bank, when translated to French provides two different words: *banque* and *banc* representing financial and geographical meanings, respectively. Such multilingual distributional information helped them in accounting for polysemy.

Table 1 provides a directory of existing frameworks that are frequently used for creating embeddings which are further incorporated into deep learning models.

Framework	Language	URL
S-Space	Java	https://github.com/fozziethebeat/S-Space
Semanticvectors	Java	https://github.com/semanticvectors/semanticvectors
Gensim	Python	https://radimrehurek.com/gensim/
Pydsm	Python	https://github.com/jimmycallin/pydsm
Dissect	Python	http://clic.cimec.unitn.it/composes/toolkit/
FastText	Python	https://fasttext.cc/

Framework	Language	URL
Elmo	Python	https://tfhub.dev/google/elmo/2

Table 1: Frameworks providing embedding tools and methods.

C. Character Embeddings

Word embeddings are able to capture syntactic and semantic information, yet for tasks such as POS-tagging and NER, intra-word morphological and shape information can also be very useful. Generally speaking, building natural language understanding systems at the character level has attracted certain research attention (Kim et al., 2016; Dos Santos and Gatti, 2014; Santos and Guimaraes, 2015; Santos and Zadrozny, 2014). Better results on morphologically rich languages are reported in certain NLP tasks. Santos and Guimaraes (2015) applied character-level representations, along with word embeddings for NER, achieving state-of-the-art results in Portuguese and Spanish corpora. Kim et al. (2016) showed positive results on building a neural language model using only character embeddings. Ma et al. (2016) exploited several embeddings, including character trigrams, to incorporate prototypical and hierarchical information for learning pre-trained label embeddings in the context of NER.

A common phenomenon for languages with large vocabularies is the unknown word issue or out-of-vocabulary word (OOV) issue. Character embeddings naturally deal with it since each word is considered as no more than a composition of individual letters. In languages where text is not composed of separated words but individual characters and the semantic meaning of words map to its compositional characters (such as Chinese), building systems at the character level is a natural choice to avoid word segmentation (Chen et al., 2015). Thus, works employing deep learning applications on such languages tend to prefer character embeddings over word vectors (Zheng et al., 2013). For example, Peng et al. (2017) proved that radical-level processing could greatly improve sentiment classification performance. In particular, the authors proposed two types of Chinese radical-based hierarchical embeddings, which incorporate not only semantics at radical and character level, but also sentiment information. Bojanowski et al. (2016) also tried to improve the representation of words by using character-level information in morphologically-rich languages. They approached the skip-gram method by representing words as bag-of-characters n-grams. Their work thus had the effectiveness of the skip-gram model

along with addressing some persistent issues of word embeddings. The method was also fast, which allowed training models on large corpora quickly. Popularly known as *FastText*, such a method stands out over previous methods in terms of speed, scalability, and effectiveness.

Apart from character embeddings, different approaches have been proposed for OOV handling. Herbelot and Baroni (2017) provided OOV handling on-the-fly by initializing the unknown words as the sum of the context words and refining these words with a high learning rate. However, their approach is yet to be tested on typical NLP tasks. Pinter et al. (2017) provided an interesting approach of training a character-based model to recreate pre-trained embeddings. This allowed them to learn a compositional mapping form character to word embedding, thus tackling the OOV problem.

Despite the ever growing popularity of distributional vectors, recent discussions on their relevance in the long run have cropped up. For example, Lucy and Gauthier (2017) has recently tried to evaluate how well the word vectors capture the necessary facets of conceptual meaning. The authors have discovered severe limitations in perceptual understanding of the concepts behind the words, which cannot be inferred from distributional semantics alone. A possible direction for mitigating these deficiencies will be grounded learning, which has been gaining popularity in this research domain.

D. Contextualized Word Embeddings

The quality of word representations is generally gauged by its ability to encode syntactical information and handle polysemic behavior (or word senses). These properties result in improved semantic word representations. Recent approaches in this area encode such information into its embeddings by leveraging the context. These methods provide deeper networks that calculate word representations as a function of its context.

Traditional word embedding methods such as Word2Vec and Glove consider all the sentences where a word is present in order to create a global vector representation of that word. However, a word can have completely different senses or meanings in the contexts. For example, lets consider these two sentences - 1) “The *bank* will not be accepting cash on Saturdays” 2) “The river overflowed the *bank*.”. The word senses of *bank* are different in these two sentences depending on its context. Reasonably, one might want two different vector

representations of the word *bank* based on its two different word senses. The new class of models adopt this reasoning by diverging from the concept of global word representations and proposing contextual word embeddings instead.

Embedding from Language Model (ELMo) (Peters et al., 2018) is one such method that provides deep contextual embeddings. ELMo produces word embeddings for each context where the word is used, thus allowing different representations for varying senses of the same word. Specifically, for different sentences where a word is present, ELMo generates different representations of i.e., .

The mechanism of ELMo is based on the representation obtained from a bidirectional language model. A bidirectional language model (biLM) constitutes of two language models (LM) 1) *forward LM* and 2) *backward LM*. A forward LM takes input representation for each of the token and passes it through layers of forward LSTM to get representations where . Each of these representations, being hidden representations of recurrent neural networks, is context dependent. A forward LM can be seen as a method to model the joint probability of a sequence of tokens: . At a timestep the forward LM predicts the next token given the previous observed tokens . This is typically achieved by placing a softmax layer on top of the final LSTM in a forward LM. On the other hand, a backward LM models the same joint probability of the sequence by predicting the previous token given the future tokens: . In other words, a backward LM is similar to forward LM which processes a sequence with the order being reversed. The training of the biLM model involves modeling the log-likelihood of both the sentence orientations. Finally, hidden representations from both LMs are concatenated to compose the final token vectors (Mousa et al., 2017).

For each token, ELMo extracts the intermediate layer representations from the biLM and performs a linear combination based on the given downstream task. A -layer biLM contains set of representations as shown below -

Here, is the token representation at the lowest level. One can use either character or word embeddings to initialize . For other values of ,

ELMo flattens all layers in R in a single vector such that -

In Eq. 8, is the softmax-normalized weight vector to combine the representations of different layers. is a hyperparameter which helps in

optimization and task specific scaling of the ELMo representation. ELMo produces varied word representations for the same word in different sentences. According to Peters et al. (2018), it is always beneficial to combine ELMo word representations with standard global word representations like Glove and Word2Vec.

Off-late, there has been a surge of interest in pre-trained language models for myriad of natural language tasks (Dai et al., 2015). Language modeling is chosen as the pre-training objective as it is widely considered to incorporate multiple traits of natural language understanding and generation. A good language model requires learning complex characteristics of language involving syntactical properties and also semantical coherence. Thus, it is believed that unsupervised training on such objectives would infuse better linguistic knowledge into the networks than random initialization. The *generative pre-training* and *discriminative fine-tuning* procedure is also desirable as the pre-training is unsupervised and does not require any manual labeling.

Radford et al. (2018) proposed similar pre-trained model, the OpenAI-GPT, by adapting the *Transformer* (see section 4-E). Recently, Delvin et al. (2018) proposed BERT which utilizes a transformer network to pre-train a language model for extracting contextual word embeddings. Unlike ELMo and OpenAI-GPT, BERT uses different pre-training tasks for language modeling. In one of the tasks, BERT randomly masks a percentage of words in the sentences and only predicts those masked words. In the other task, BERT predicts the next sentence given a sentence. This task in particular tries to model the relationship among two sentences which is supposedly not captured by traditional bidirectional language models. Consequently, this particular pre-training scheme helps BERT to outperform state-of-the-art techniques by a large margin on key NLP tasks such as QA, Natural Language Inference (NLI) where understanding relation among two sentences is very important. We discuss the impact of these proposed models and the performance achieved by them in section 8-A.

The described approaches for contextual word embeddings promises better quality representations for words. The pre-trained deep language models also provide a headstart for downstream tasks in the form of transfer learning. This approach has been extremely popular in computer vision tasks. Whether there would be similar trends in the

NLP community, where researchers and practitioners would prefer such models over traditional variants remains to be seen in the future.

Chapter 3

Convolutional Neural Networks

Following the popularization of word embeddings and its ability to represent words in a distributed space, the need arose for an effective feature function that extracts higher-level features from constituting words or n-grams. These abstract features would then be used for numerous NLP tasks such as sentiment analysis, summarization, machine translation, and question answering (QA). CNNs turned out to be the natural choice given their effectiveness in computer vision tasks (Krizhevsky et al., 2012; Razavian et al., 2014; Jia et al., 2014).

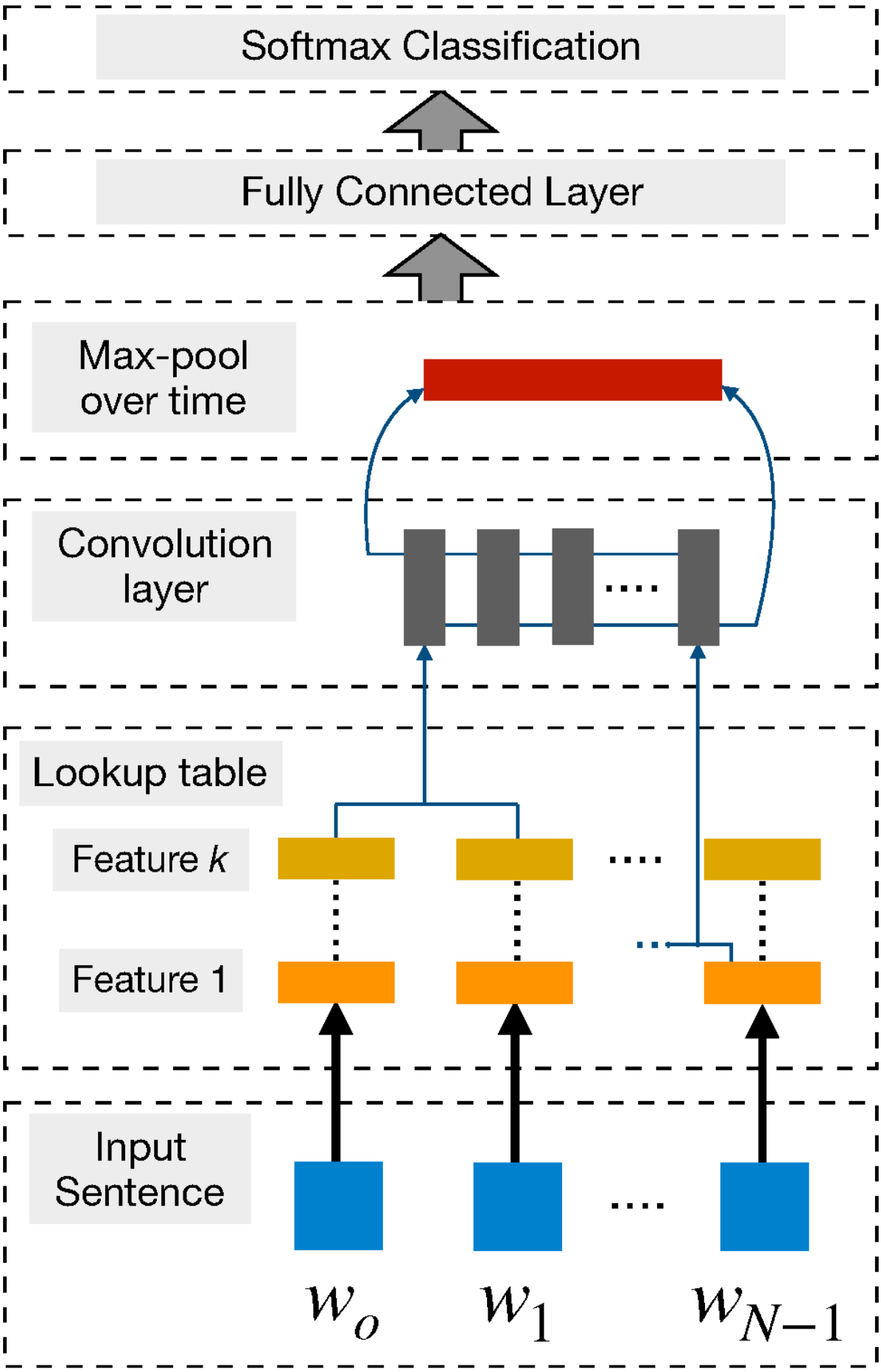


Figure 5: CNN framework used to perform word wise class prediction (Figure source: Collobert and Weston (2008))

The use of CNNs for sentence modeling traces back to Collobert and Weston (2008). This work used multi-task learning to output multiple predictions for NLP tasks such as POS tags, chunks, named-entity tags,

semantic roles, semantically-similar words and a language model. A look-up table was used to transform each word into a vector of user-defined dimensions. Thus, an input sequence of words was transformed into a series of vectors by applying the look-up table to each of its words (Figure 5).

This can be thought of as a primitive word embedding method whose weights were learned in the training of the network. In (Collobert et al., 2011), Collobert extended his work to propose a general CNN-based framework to solve a plethora of NLP tasks. Both these works triggered a huge popularization of CNNs amongst NLP researchers. Given that CNNs had already shown their mettle for computer vision tasks, it was easier for people to believe in their performance.

CNNs have the ability to extract salient n-gram features from the input sentence to create an informative latent semantic representation of the sentence for downstream tasks. This application was pioneered by Collobert et al. (2011), Kalchbrenner et al. (2014), Kim (2014), which led to a huge proliferation of CNN-based networks in the succeeding literature. Below, we describe the working of a simple CNN-based sentence modeling network:

A. Basic CNN

1. Sentence Modeling

For each sentence, let represent the word embedding for the word in the sentence, where is the dimension of the word embedding. Given that a sentence has words, the sentence can now be represented as an embedding matrix . Figure 6 depicts such a sentence as an input to the CNN framework.

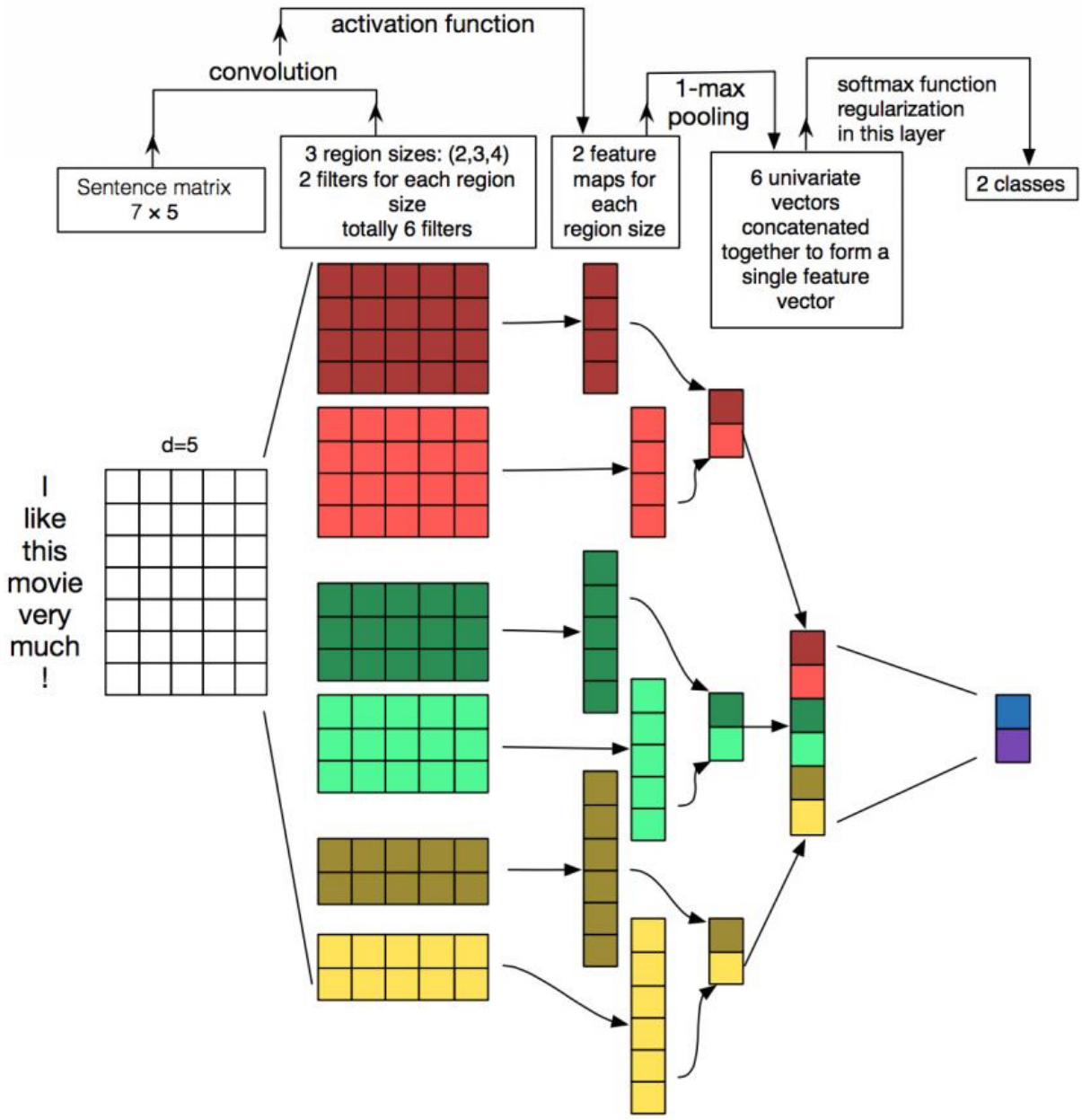


Figure 6: CNN modeling on text (Figure source: [Zhang and Wallace (2015)](#))

Let refer to the concatenation of vectors . Convolution is performed on this input embedding layer. It involves a *filter* which is applied to a window of words to produce a new feature. For example, a feature is generated using the window of words by

Here, is the bias term and is a non-linear activation function, for example the hyperbolic tangent. The filter is applied to all possible windows using the same weights to create the feature map.

In a CNN, a number of convolutional filters, also called kernels (typically hundreds), of different widths slide over the entire word embedding matrix. Each kernel extracts a specific pattern of n-gram. A convolution layer is usually followed by a max-pooling strategy, , which subsamples the input typically by applying a max operation on each filter. This strategy has two primary reasons.

Firstly, max pooling provides a fixed-length output which is generally required for classification. Thus, regardless the size of the filters, max pooling always maps the input to a fixed dimension of outputs. Secondly, it reduces the output's dimensionality while keeping the most salient n-gram features across the whole sentence. This is done in a translation invariant manner where each filter is now able to extract a particular feature (e.g., negations) from anywhere in the sentence and add it to the final sentence representation.

The word embeddings can be initialized randomly or pre-trained on a large unlabeled corpora (as in Section 2). The latter option is sometimes found beneficial to performance, especially when the amount of labeled data is limited (Kim, 2014). This combination of convolution layer followed by max pooling is often stacked to create deep CNN networks. These sequential convolutions help in improved mining of the sentence to grasp a truly abstract representations comprising rich semantic information. The kernels through deeper convolutions cover a larger part of the sentence until finally covering it fully and creating a global summarization of the sentence features.

2. Window Approach

The above-mentioned architecture allows for modeling of complete sentences into sentence representations. However, many NLP tasks, such as NER, POS tagging, and SRL, require word-based predictions. To adapt CNNs for such tasks, a window approach is used, which assumes that the tag of a word primarily depends on its neighboring words. For each word, thus, a fixed-size window surrounding itself is assumed and the sub-sentence ranging within the window is considered. A standalone CNN is applied to this sub-sentence as explained earlier and predictions are attributed to the word in the center of the window. Following this approach, Poria et al. (2016) employed a multi-level deep CNN to tag each word in a sentence as a possible aspect or non-aspect. Coupled with a set of linguistic patterns, their ensemble classifier managed to perform well in aspect detection.

The ultimate goal of word-level classification is generally to assign a sequence of labels to the entire sentence. In such cases, structured prediction techniques such as conditional random field (CRF) are sometimes employed to better capture dependencies between adjacent class labels and finally generate cohesive label sequence giving maximum score to the whole sentence (Kirillov et al., 2015).

To get a larger contextual range, the classic window approach is often coupled with a time-delay neural network (TDNN) (Waibel et al., 1989). Here, convolutions are performed across all windows throughout the sequence. These convolutions are generally constrained by defining a kernel having a certain width. Thus, while the classic window approach only considers the words in the window around the word to be labeled, TDNN considers all windows of words in the sentence at the same time. At times, TDNN layers are also stacked like CNN architectures to extract local features in lower layers and global features in higher layers (Collobert et al., 2011).

B. Applications

In this section, we present some of the crucial works that employed CNNs on NLP tasks to set state-of-the-art benchmarks in their respective times.

Kim (2014) explored using the above architecture for a variety of sentence classification tasks, including sentiment, subjectivity and question type classification, showing competitive results. This work was quickly adapted by researchers given its simple yet effective network. After training for a specific task, the randomly initialized convolutional kernels became specific n-gram feature detectors that were useful for that target task (Figure 7) . This simple network, however, had many shortcomings with the CNN's inability to model long distance dependencies standing as the main issue.

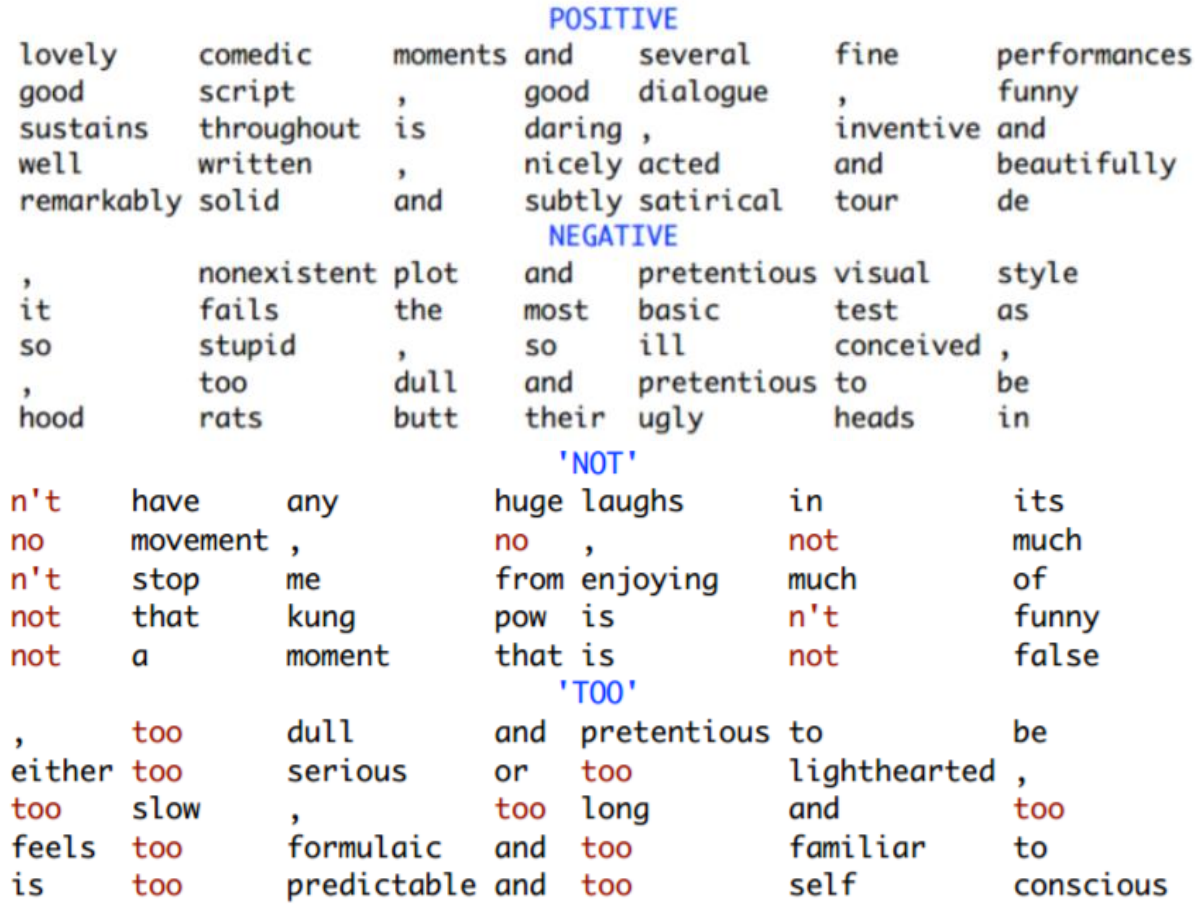


Figure 7: Top 7-grams by four learned 7-gram kernels; each kernel is sensitive to a specific kind of 7-gram (Figure Source: Kalchbrenner et al. (2014))

This issue was partly handled by Kalchbrenner et al. (2014), who published a prominent paper where they proposed a dynamic convolutional neural network (DCNN) for semantic modeling of sentences. They proposed dynamic k-max pooling strategy which, given a sequence selects the most active features. The selection preserved the order of the features but was insensitive to their specific positions (Figure 8). Built on the concept of TDNN, they added this dynamic k-max pooling strategy to create a sentence model. This combination allowed filters with small width to span across a long range within the input sentence, thus accumulating crucial information across the sentence. In the induced subgraph (Figure 8), higher order features had highly variable ranges that could be either short and focused or global and long as the input sentence. They applied their model on multiple tasks, including sentiment prediction and question type classification, achieving significant results. Overall, this work commented on the range of individual kernels while trying to model contextual semantics and proposed a way to extend their reach.

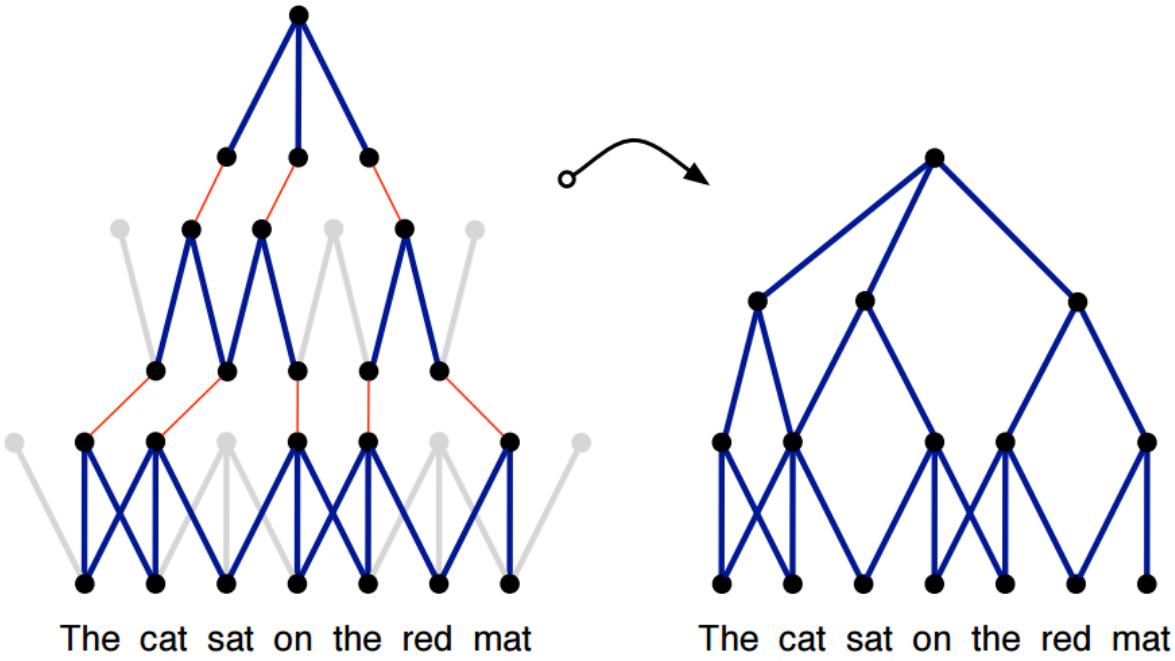


Figure 8: DCNN subgraph. With dynamic pooling, a filter with small width at the higher layers can relate phrases far apart in the input sentence (Figure Source: Kalchbrenner et al. (2014))

Tasks involving sentiment analysis also require effective extraction of aspects along with their sentiment polarities (Mukherjee and Liu, 2012). Ruder et al. (2016) applied a CNN where in the input they concatenated an aspect vector with the word embeddings to get competitive results. CNN modeling approach varies amongst different length of texts. Such differences were seen in many works like Johnson and Zhang (2015), where performance on longer text worked well as opposed to shorter texts. Wang et al. (2015) proposed the usage of CNN for modeling representations of short texts, which suffer from the lack of available context and, thus, require extra efforts to create meaningful representations. The authors proposed semantic clustering which introduced multi-scale semantic units to be used as external knowledge for the short texts. CNN was used to combine these units and form the overall representation. In fact, this requirement of high context information can be thought of as a caveat for CNN-based models. NLP tasks involving microtexts using CNN-based methods often require the need of additional information and external knowledge to perform as per expectations. This fact was also observed in (Poria et al., 2016), where authors performed sarcasm detection in Twitter texts using a CNN network. Auxiliary support, in the form of pre-trained networks trained on emotion, sentiment and personality datasets was used to achieve state-of-the-art performance.

CNNs have also been extensively used in other tasks. For example, Denil et al. (2014) applied DCNN to map meanings of words that constitute a sentence to that of documents for summarization. The DCNN learned convolution filters at both the sentence and document level, hierarchically learning to capture and compose low-level lexical features into high-level semantic concepts. The focal point of this work was the introduction of a novel visualization technique of the learned representations, which provided insights not only in the learning process but also for automatic summarization of texts.

CNN models are also suitable for certain NLP tasks that require semantic matching beyond classification (Hu et al., 2014). A similar model to the above CNN architecture (Figure 6) was explored in (Shen et al., 2014) for information retrieval. The CNN was used for projecting queries and documents to a fixed-dimension semantic space, where cosine similarity between the query and documents was used for ranking documents regarding a specific query. The model attempted to extract rich contextual structures in a query or a document by considering a temporal context window in a word sequence. This captured the contextual features at the word n-gram level. The salient word n-grams is then discovered by the convolution and max-pooling layers which are then aggregated to form the overall sentence vector.

In the domain of QA, Yih et al. (2014) proposed to measure the semantic similarity between a question and entries in a knowledge base (KB) to determine what supporting fact in the KB to look for when answering a question. To create semantic representations, a CNN similar to the one in Figure 6 was used. Unlike the classification setting, the supervision signal came from positive or negative text pairs (e.g., query-document), instead of class labels. Subsequently, Dong et al. (2015) introduced a multi-column CNN (MCCNN) to analyze and understand questions from multiple aspects and create their representations. MCCNN used multiple column networks to extract information from aspects comprising answer types and context from the input questions. By representing entities and relations in the KB with low-dimensional vectors, they used question-answer pairs to train the CNN model so as to rank candidate answers. Severyn and Moschitti (2016) also used CNN network to model optimal representations of question and answer sentences. They proposed additional features in the embeddings in the form of relational information given by matching words between the question and answer pair. These parameters were tuned by the

network. This simple network was able to produce comparable results to state-of-the-art methods.

CNNs are wired in a way to capture the most important information in a sentence. Traditional max-pooling strategies perform this in a translation invariant form. However, this often misses valuable information present in multiple facts within the sentence. To overcome this loss of information for multiple-event modeling, Chen et al. (2015) proposed a modified pooling strategy: dynamic multi-pooling CNN (DMCNN). This strategy used a novel dynamic multi-pooling layer that, as the name suggests, incorporates event triggers and arguments to reserve more crucial information from the pooling layer.

CNNs inherently provide certain required features like local connectivity, weight sharing, and pooling. This puts forward some degree of invariance which is highly desired in many tasks. Speech recognition also requires such invariance and, thus, Abdel-Hamid et al. (2014) used a hybrid CNN-HMM model which provided invariance to frequency shifts along the frequency axis. This variability is often found in speech signals due to speaker differences. They also performed limited weight sharing which led to a smaller number of pooling parameters, resulting in lower computational complexity. Palaz et al. (2015) performed extensive analysis of CNN-based speech recognition systems when given raw speech as input. They showed the ability of CNNs to directly model the relationship between raw input and phones, creating a robust automatic speech recognition system.

Tasks like machine translation require perseverance of sequential information and long-term dependency. Thus, structurally they are not well suited for CNN networks, which lack these features. Nevertheless, Tu et al. (2015) addressed this task by considering both the semantic similarity of the translation pair and their respective contexts. Although this method did not address the sequence perseverance problem, it allowed them to get competitive results amongst other benchmarks.

Overall, CNNs are extremely effective in mining semantic clues in contextual windows. However, they are very data heavy models. They include a large number of trainable parameters which require huge training data. This poses a problem when scarcity of data arises. Another persistent issue with CNNs is their inability to model long-distance contextual information and preserving sequential order in their representations (Kalchbrenner et al., 2014; Tu et al., 2015). Other

networks like recursive models (explained below) reveal themselves as better suited for such learning.

Chapter 4

Recurrent Neural Networks

RNNs (Elman, 1990) use the idea of processing sequential information. The term "recurrent" applies as they perform the same task over each instance of the sequence such that the output is dependent on the previous computations and results. Generally, a fixed-size vector is produced to represent a sequence by feeding tokens one by one to a recurrent unit. In a way, RNNs have "memory" over previous computations and use this information in current processing. This template is naturally suited for many NLP tasks such as language modeling (Mikolov et al., 2010, 2011; Sutskever et al., 2011), machine translation (Liu et al., 2014; Auli et al., 2013; Sutskever et al., 2014), speech recognition (Robinson et al., 1996; Graves et al., 2013; Graves and Jaitly, 2014; Sak et al., 2014), and image captioning (Karpathy and Fei-Fei, 2015). This made RNNs increasingly popular for NLP applications in recent years.

A. Need for Recurrent Networks

In this section, we analyze the fundamental properties that favored the popularization of RNNs in a multitude of NLP tasks. Given that an RNN performs sequential processing by modeling units in sequence, it has the ability to capture the inherent sequential nature present in language, where units are characters, words or even sentences. Words in a language develop their semantical meaning based on the previous words in the sentence. A simple example stating this would be the difference in meaning between "dog" and "hot dog". RNNs are tailor-made for modeling such context dependencies in language and similar sequence modeling tasks, which resulted to be a strong motivation for researchers to use RNNs over CNNs in these areas.

Another factor aiding RNN's suitability for sequence modeling tasks lies in its ability to model variable length of text, including very long sentences, paragraphs and even documents (Tang et al., 2015). Unlike CNNs, RNNs have flexible computational steps that provide better modeling capability and create the possibility to capture unbounded

context. This ability to handle input of arbitrary length became one of the selling points of major works using RNNs (Chung et al., 2014).

Many NLP tasks require semantic modeling over the whole sentence. This involves creating a gist of the sentence in a fixed dimensional hyperspace. RNN's ability to summarize sentences led to their increased usage for tasks like machine translation (Cho et al., 2014) where the whole sentence is summarized to a fixed vector and then mapped back to the variable-length target sequence.

RNN also provides the network support to perform time distributed joint processing. Most of the sequence labeling tasks like POS tagging (Santos and Zadrozny, 2014) come under this domain. More specific use cases include applications such as multi-label text categorization (Chen et al., 2017), multimodal sentiment analysis (Poria et al., 2017; Zadeh et al., 2017; Tong et al., 2017), and subjectivity detection (Chaturvedi et al., 2017).

The above points enlist some of the focal reasons that motivated researchers to opt for RNNs. However, it would be gravely wrong to make conclusions on the superiority of RNNs over other deep networks. Recently, several works provided contrasting evidence on the superiority of CNNs over RNNs. Even in RNN-suited tasks like language modeling, CNNs achieved competitive performance over RNNs (Dauphin et al., 2016). Both CNNs and RNNs have different objectives when modeling a sentence. While RNNs try to create a composition of an arbitrarily long sentence along with unbounded context, CNNs try to extract the most important n-grams. Although they prove an effective way to capture n-gram features, which is approximately sufficient in certain sentence classification tasks, their sensitivity to word order is restricted locally and long-term dependencies are typically ignored.

Yin et al. (2017) provided interesting insights on the comparative performance between RNNs and CNNs. After testing on multiple NLP tasks that included sentiment classification, QA, and POS tagging, they concluded that there is no clear winner: the performance of each network depends on the global semantics required by the task itself.

Below, we discuss some of the RNN models extensively used in the literature.

B. RNN models

1. Simple RNN

In the context of NLP, RNNs are primarily based on Elman network (Elman, 1990) and they are originally three-layer networks. Figure 9 illustrates a more general RNN which is unfolded across time to accommodate a whole sequence. In the figure, is taken as the input to the network at time step and represents the hidden state at the same time step. Calculation of is based as per the equation:

Thus, is calculated based on the current input and the previous time step's hidden state. The function is taken to be a non-linear transformation such as and account for weights that are shared across time. In the context of NLP, typically comprises of one-hot encodings or embeddings. At times, they can also be abstract representations of textual content. illustrates the output of the network which is also often subjected to non-linearity, especially when the network contains further layers downstream.

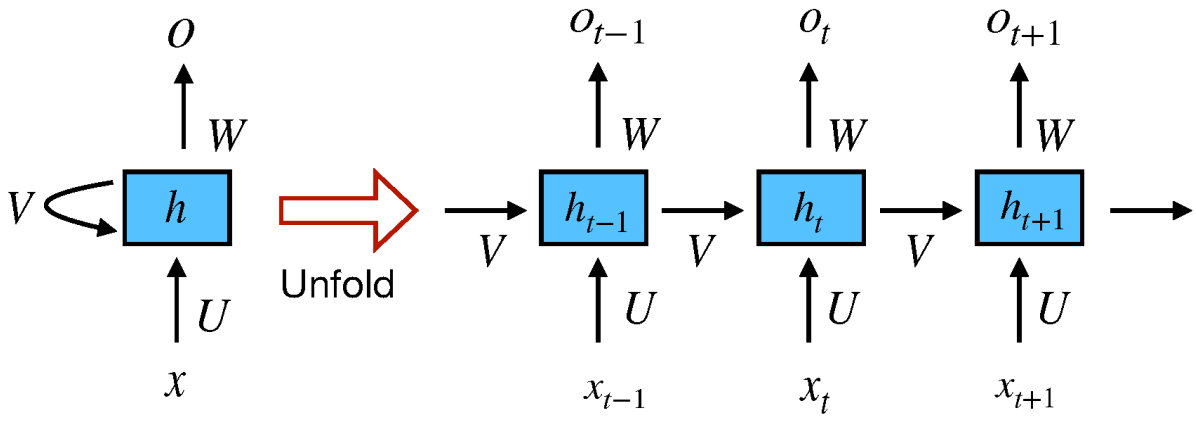


Figure 9: Simple RNN network (Figure Source: Le Cun et al. (2015))

The hidden state of the RNN is typically considered to be its most crucial element. As stated before, it can be considered as the network's memory element that accumulates information from other time steps. In practice, however, these simple RNN networks suffer from the infamous *vanishing gradient* problem, which makes it really hard to learn and tune the parameters of the earlier layers in the network.

This limitation was overcome by various networks such as long short-term memory (LSTM), gated recurrent units (GRUs), and residual networks (ResNets), where the first two are the most used RNN variants in NLP applications.

2. Long Short-Term Memory

LSTM (Hochreiter and Schmidhuber, 1997; Gers et al., 1999) (Figure 10) has additional "forget" gates over the simple RNN. Its unique mechanism enables it to overcome both the *vanishing* and *exploding* gradient problem.

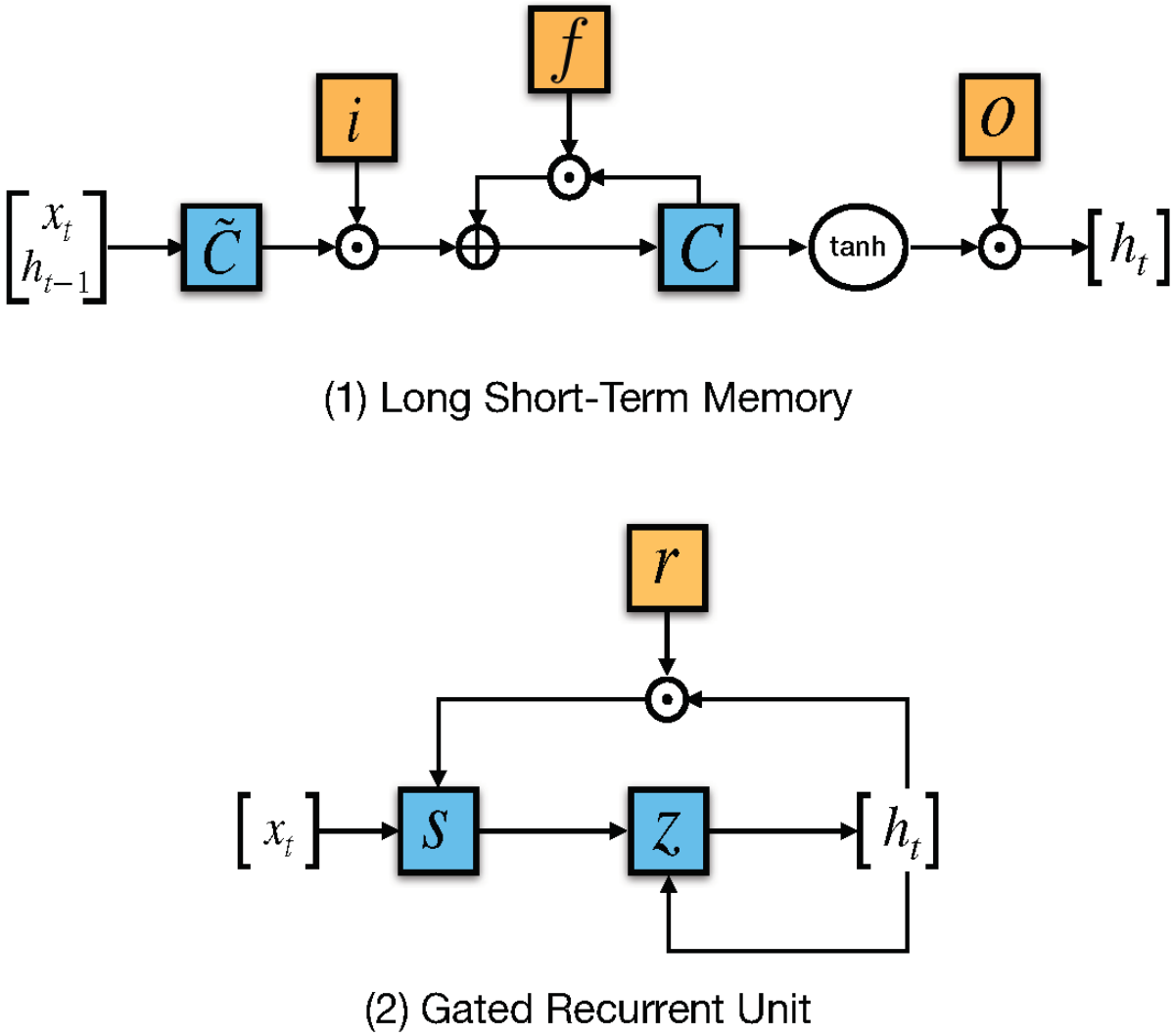


Figure 10: Illustration of an LSTM and GRU gate (Figure Source: Chung et al. (2014))

Unlike the vanilla RNN, LSTM allows the error to back-propagate through unlimited number of time steps. Consisting of three gates: input, forget and output gates, it calculates the hidden state by taking a combination of these three gates as per the equations below:

3. Gated Recurrent Units

Another gated RNN variant called GRU (Cho et al., 2014) (Figure 10}) of lesser complexity was invented with empirically similar performances to LSTM in most tasks. GRU comprises of two gates, reset gate and update gate, and handles the flow of information like an LSTM sans a memory unit. Thus, it exposes the whole hidden content without any control. Being less complex, GRU can be a more efficient RNN than LSTM. The working of GRU is as follows:

Researchers often face the dilemma of choosing the appropriate gated RNN. This also extends to developers working in NLP. Throughout the history, most of the choices over the RNN variant tended to be heuristic. Chung et al. (2014) did a critical comparative evaluation of the three RNN variants mentioned above, although not on NLP tasks. They evaluated their work on tasks relating to polyphonic music modeling and speech signal modeling. Their evaluation clearly demonstrated the superiority of the gated units (LSTM and GRU) over the traditional simple RNN (in their case, using activation) (Figure 11). However, they could not make any concrete conclusion about which of the two gating units was better. This fact has been noted in other works too and, thus, people often leverage on other factors like computing power while choosing between the two.

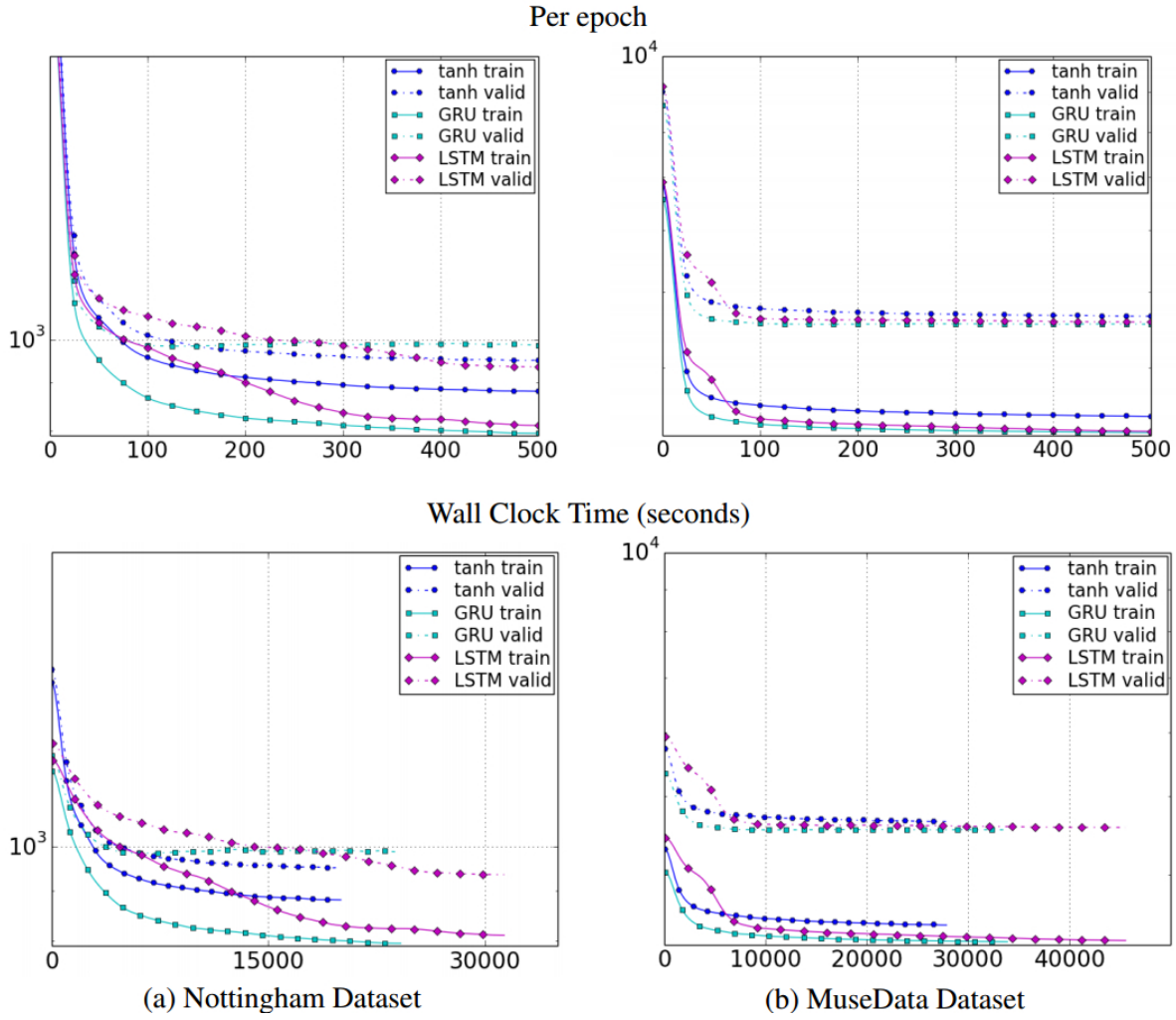


Figure 11: Learning curves for training and validation sets of different types of units with respect to (top) the number of iterations and (bottom) the wall clock time. y-axis corresponds to the negative log likelihood of the model shown in log-scale (Figure source: Chung et al. (2014))

C. Applications

1. RNN for word-level classification

RNNs have had a huge presence in the field of word-level classification. Many of their applications stand as state of the art in their respective tasks. Lample et al. (2016) proposed to use bidirectional LSTM for NER. The network captured arbitrarily long context information around the target word (curbing the limitation of a fixed window size) resulting in two fixed-size vector, on top of which another fully-connected layer was built. They used a CRF layer at last for the final entity tagging.

RNNs have also shown considerable improvement in language modeling over traditional methods based on count statistics. Pioneering work in this field was done by Graves (2013), who introduced the effectiveness of RNNs in modeling complex sequences with long range context structures. He also proposed deep RNNs where multiple layers of hidden states were used to enhance the modeling. This work established the usage of RNNs on tasks beyond the context of NLP. Later, Sundermeyer et al. (2015) compared the gain obtained by replacing a feed-forward neural network with an RNN when conditioning the prediction of a word on the words ahead. In their work, they proposed a typical hierarchy in neural network architectures where feed-forward neural networks gave considerable improvement over traditional count-based language models, which in turn were superseded by RNNs and later by LSTMs. An important point that they mentioned was the applicability of their conclusions to a variety of other tasks such as statistical machine translation (Sundermeyer et al., 2014).

2. RNN for sentence-level classification

Wang et al. (2015) proposed encoding entire tweets with LSTM, whose hidden state is used for predicting sentiment polarity. This simple strategy proved competitive to the more complex DCNN structure by Kalchbrenner et al. (2014) designed to endow CNN models with ability to capture long-term dependencies. In a special case studying negation phrase, the authors also showed that the dynamics of LSTM gates can capture the reversal effect of the word *not*.

Similar to CNN, the hidden state of an RNN can also be used for semantic matching between texts. In dialogue systems, Lowe et al. (2015) proposed to match a message with candidate responses with

Dual-LSTM, which encodes both as fixed-size vectors and then measure their inner product as the basis to rank candidate responses.

3. RNN for generating language

A challenging task in NLP is generating natural language, which is another natural application of RNNs. Conditioned on textual or visual data, deep LSTMs have been shown to generate reasonable task-specific text in tasks such as machine translation, image captioning, etc. In such cases, the RNN is termed a decoder.

In (Sutskever et al., 2014), the authors proposed a general deep LSTM encoder-decoder framework that maps a sequence to another sequence. One LSTM is used to encode the ``source'' sequence as a fixed-size vector, which can be text in the original language (machine translation), the question to be answered (QA) or the message to be replied to (dialogue systems). The vector is used as the initial state of another LSTM, named the decoder. During inference, the decoder generates tokens one by one, while updating its hidden state with the last generated token. Beam search is often used to approximate the optimal sequence.

Sutskever et al. (2014) experimented with 4-layer LSTM on a machine translation task in an end-to-end fashion, showing competitive results. In (Vinyals and Le, 2015), the same encoder-decoder framework is employed to model human conversations. When trained on more than 100 million message-response pairs, the LSTM decoder is able to generate very interesting responses in the open domain. It is also common to condition the LSTM decoder on additional signal to achieve certain effects. In (Li et al., 2016), the authors proposed to condition the decoder on a constant persona vector that captures the personal information of an individual speaker. In the above cases, language is generated based mainly on the semantic vector representing textual input. Similar frameworks have also been successfully used in image-based language generation, where visual features are used to condition the LSTM decoder (Figure 12).

Visual QA is another task that requires language generation based on both textual and visual clues. Malinowski et al. (2015) were the first to provide an end-to-end deep learning solution where they predicted the answer as a set of words conditioned on the input image modeled by a CNN and text modeled by an LSTM (Figure 13).

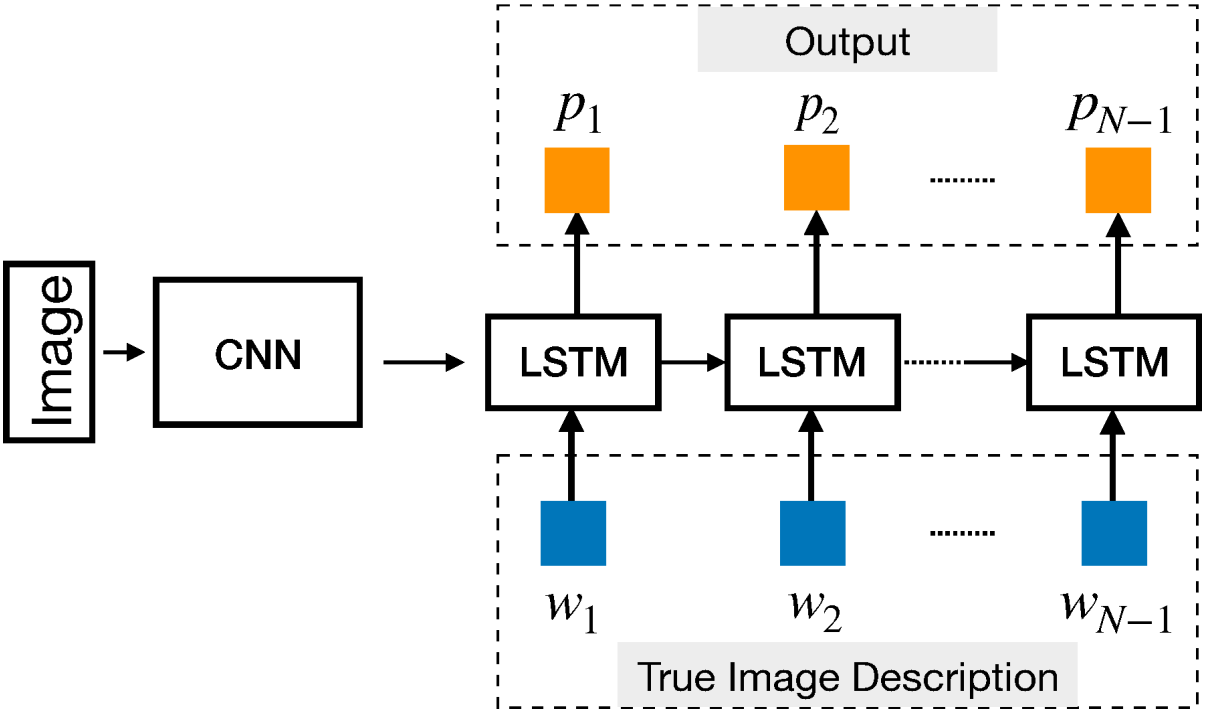


Figure 12: LSTM decoder combined with a CNN image embedder to generate image captioning (Figure source: Vinyals et al. (2015))

Kumar er al. (2015) tackled this problem by proposing an elaborated network termed dynamic memory network (DMN), which had four sub-modules. The idea was to repeatedly attend to the input text and image to form episodes of information improved at each iteration. Attention networks were used for fine-grained focus on input text phrases.

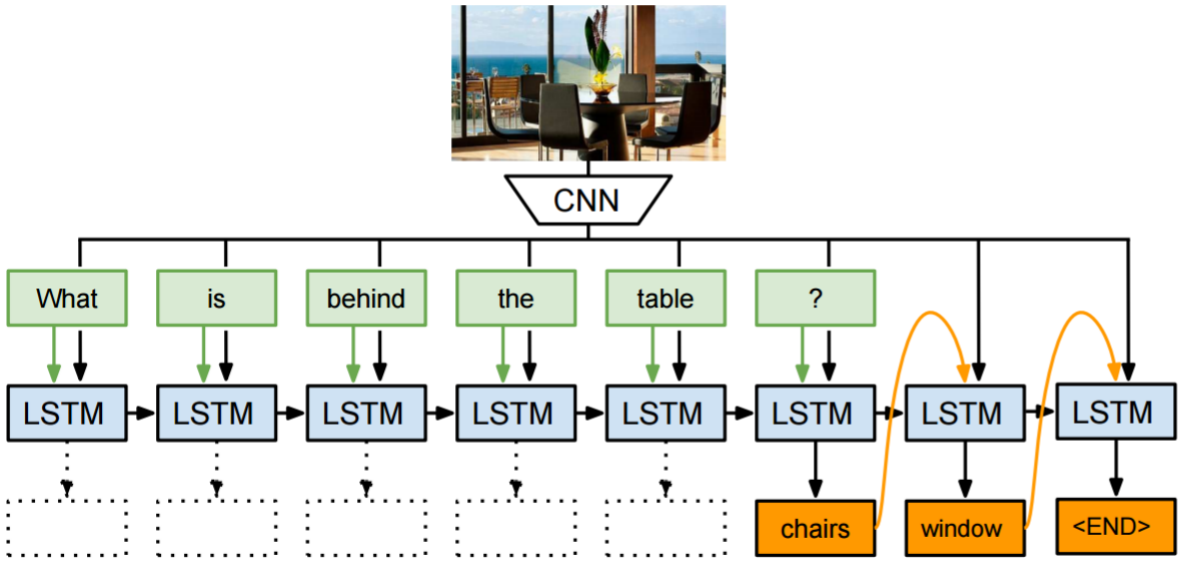


Figure 13: Neural-image QA (Figure source: Malinowski et al. (2015))

D. Attention Mechanism

One potential problem that the traditional encoder-decoder framework faces is that the encoder at times is forced to encode information which might not be fully relevant to the task at hand. The problem arises also

if the input is long or very information-rich and selective encoding is not possible.

For example, the task of text summarization can be cast as a sequence-to-sequence learning problem, where the input is the original text and the output is the condensed version. Intuitively, it is unrealistic to expect a fixed-size vector to encode all information in a piece of text whose length can potentially be very long. Similar problems have also been reported in machine translation (Bahdanau et al., 2014).

In tasks such as text summarization and machine translation, certain alignment exists between the input text and the output text, which means that each token generation step is highly related to a certain part of the input text. This intuition inspires the attention mechanism. This mechanism attempts to ease the above problems by allowing the decoder to refer back to the input sequence. Specifically during decoding, in addition to the last hidden state and generated token, the decoder is also conditioned on a "context" vector calculated based on the input hidden state sequence.

Bahdanau et al. (2014) first applied the attention mechanism to machine translation, which improved the performance especially for long sequences. In their work, the attention signal over the input hidden state sequence is determined with a multi-layer perceptron by the last hidden state of the decoder. By visualizing the attention signal over the input sequence during each decoding step, a clear alignment between the source and target language can be demonstrated (Figure 14).

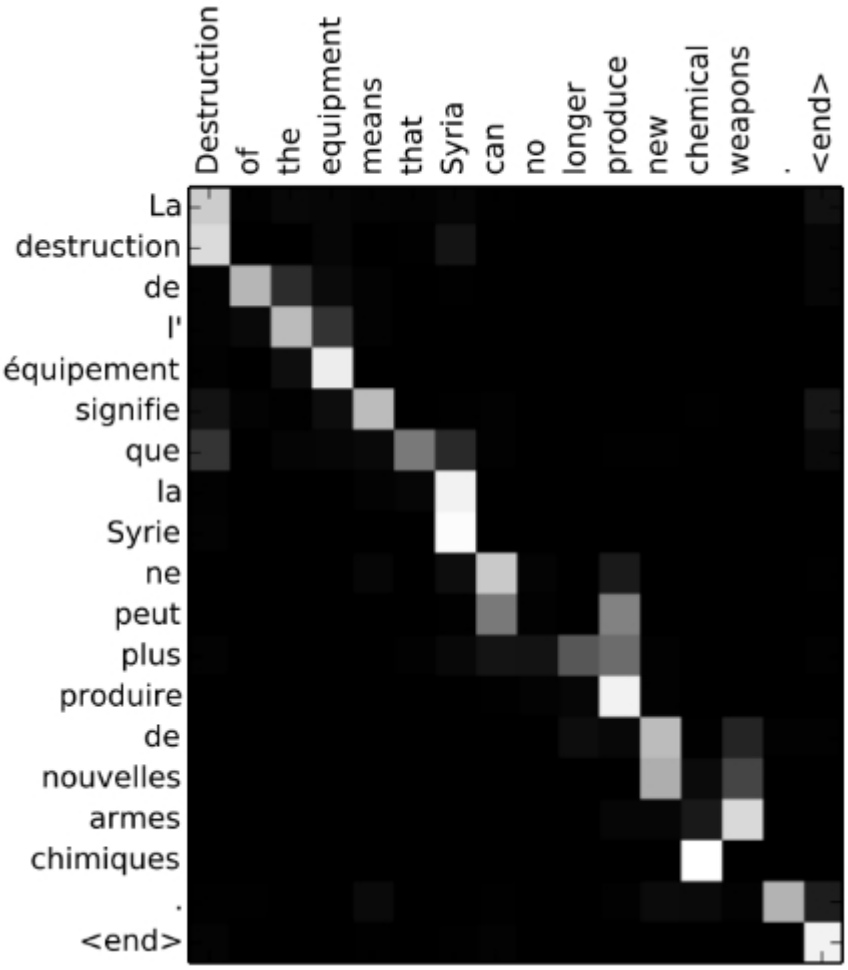


Figure 14: Word alignment matrix (Figure source: Bahdanau et al. (2014))

A similar approach was applied to the task of summarization by Rush et al. (2015) where each output word in the summary was conditioned on the input sentence through an attention mechanism. The authors performed abstractive summarization which is not very conventional as opposed to extractive summarization, but can be scaled up to large data with minimal linguistic input.

In image captioning, Xu et al. (2015) conditioned the LSTM decoder on different parts of the input image during each decoding step. Attention signal was determined by the previous hidden state and CNN features. In (Vinyals et al., 2015), the authors casted the syntactical parsing problem as a sequence-to-sequence learning task by linearizing the parsing tree. The attention mechanism proved to be more data-efficient in this work. A further step in referring to the input sequence was to directly copy words or sub-sequences of the input onto the output sequence under a certain condition (Vinyals et al., 2015), which was useful in tasks such as dialogue generation and text summarization. Copying or generation was chosen at each time step during decoding (Paulus et al. (2017)).

In aspect-based sentiment analysis, Wang et al. (2016) proposed an attention-based solution where they used aspect embeddings to provide additional support during classification (Figure 15). The attention module focused on selective regions of the sentence which affected the aspect to be classified. This can be seen in Figure 16 where, for the aspect *service* in (a), the attention module dynamically focused on the phrase "fastest delivery times" and in (b) with the aspect *food*, it identified multiple key-points across the sentence that included "tasteless" and "too sweet". Recently, Ma et al. (2018) augmented LSTM with a hierarchical attention mechanism consisting of a target-level attention and a sentence-level attention to exploit commonsense knowledge for targeted aspect-based sentiment analysis.

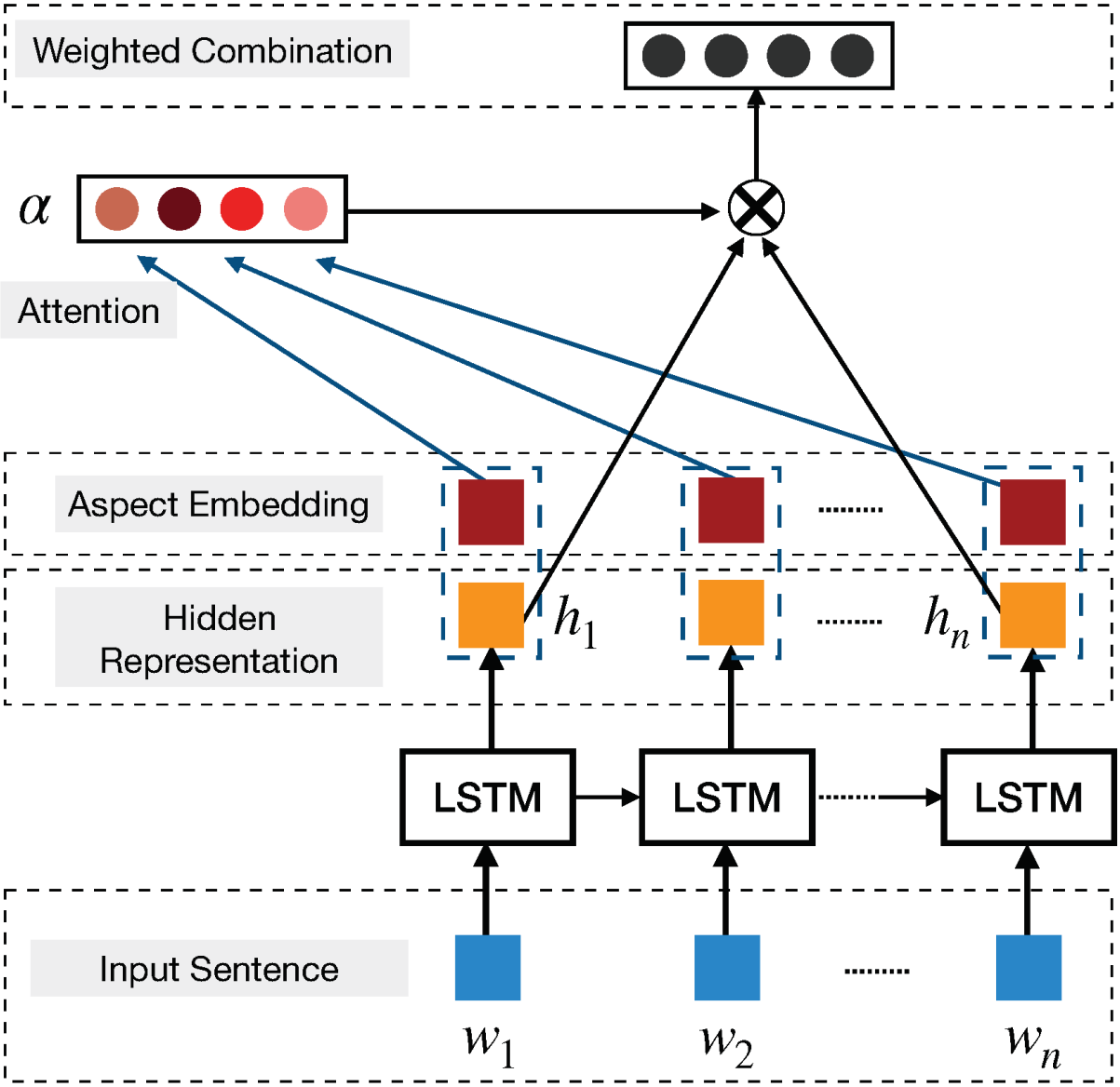
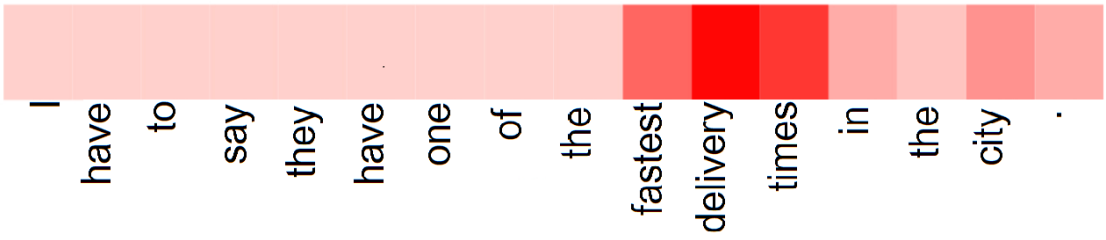
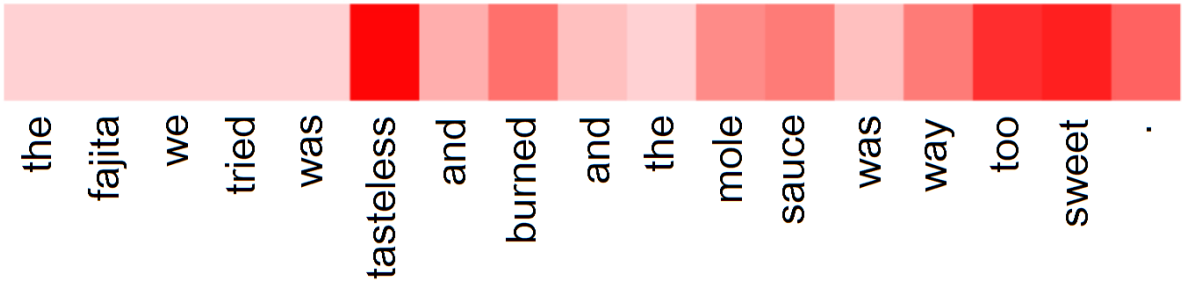


Figure 15: Aspect classification using attention (Figure source: Wang et al. (2016))



(a) the aspect of this sentence: *service*



(b) the aspect of this sentence: food

Figure 16: Focus of attention module on the sentence for certain aspects (Figure source: Wang et al. (2016))

On the other hand, Tang et al. (2016) adopted a solution based on a memory network (also known as MemNet (Weston et al., 2014)), which employed multiple-hop attention. The multiple attention computation layer on the memory led to improved lookup for most informational regions in the memory and subsequently aided the classification. Their work stands as the state of the art in this domain.

Given the intuitive applicability of attention modules, they are still being actively investigated by NLP researchers and adopted for an increasing number of applications.

E. Parallelized Attention: The Transformer

Both CNNs and RNNs have been crucial in sequence transduction applications involving the encoder-decoder architecture. Attention-based mechanisms, as described above, have further boosted the capabilities of these models. However, one of the bottlenecks suffered by these architectures is the sequential processing at the encoding step. To address this, Vaswani et al. (2017) proposed the *Transformer* which dispensed the recurrence and convolutions involved in the encoding step entirely and based models only on attention mechanisms to capture the global relations between input and output. As a result, the overall architecture became more parallelizable and required lesser

time to train along with positive results on tasks ranging from translation to parsing.

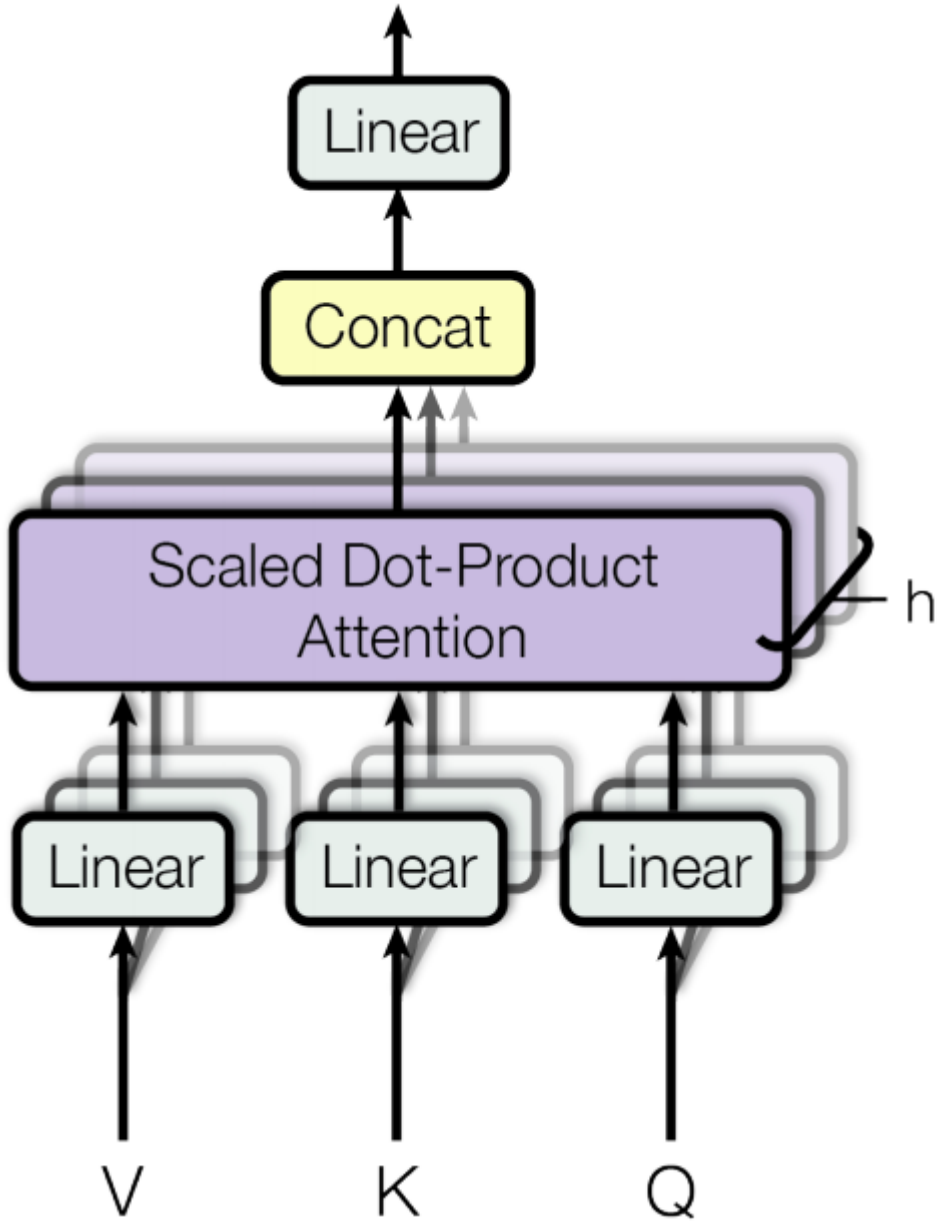


Figure 17: Multi-head Attention: Vaswani et al. (2017)

The Transformer consists stacked layers in both encoder and decoder components. Each layer has two sub-layers comprising *multi-head attention layer* (Figure 17) followed by a position-wise feed forward network. For set of queries , keys and values , the multi-head attention module performs attention times where the computation can be seen as:

here, and are projection parameters. Incorporating other techniques such as residual connections (He et al., 2016), layer normalization (Ba et al., 2016), dropouts, positional encodings, and others, the model achieves state-of-the-art results in English-German and English-French translation and constituency parsing.

Chapter 5

Recursive Neural Networks

Recursive neural networks represent a natural way to model sequences. Arguably, however, language exhibits a natural recursive structure, where words and sub-phrases combine into phrases in a hierarchical manner. Such structure can be represented by a constituency parsing tree. Thus, tree-structured models have been used to better make use of such syntactic interpretations of sentence structure (Socher et al., 2013). Specifically, in a recursive neural network, the representation of each non-terminal node in a parsing tree is determined by the representations of all its children.

A. Basic model

In this section, we describe the basic structure of recursive neural networks. As shown in Figure 17 and 18, the network defines a compositional function on the representations of phrases or words (or) to compute the representation of a higher-level phrase (or). The representations of all nodes take the same form.

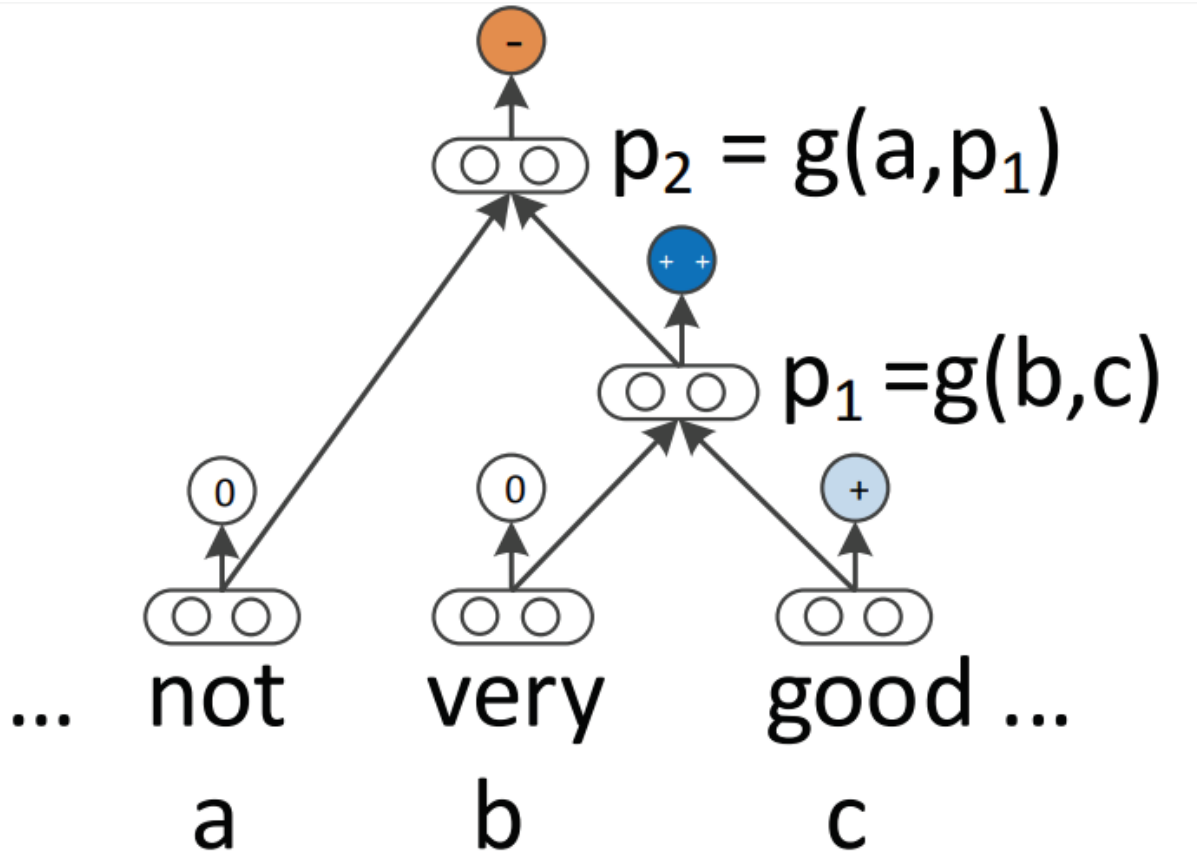


Figure 17: Recursive neural networks for phrase-level sentiment classification (Figure source: Socher et al. (2013))

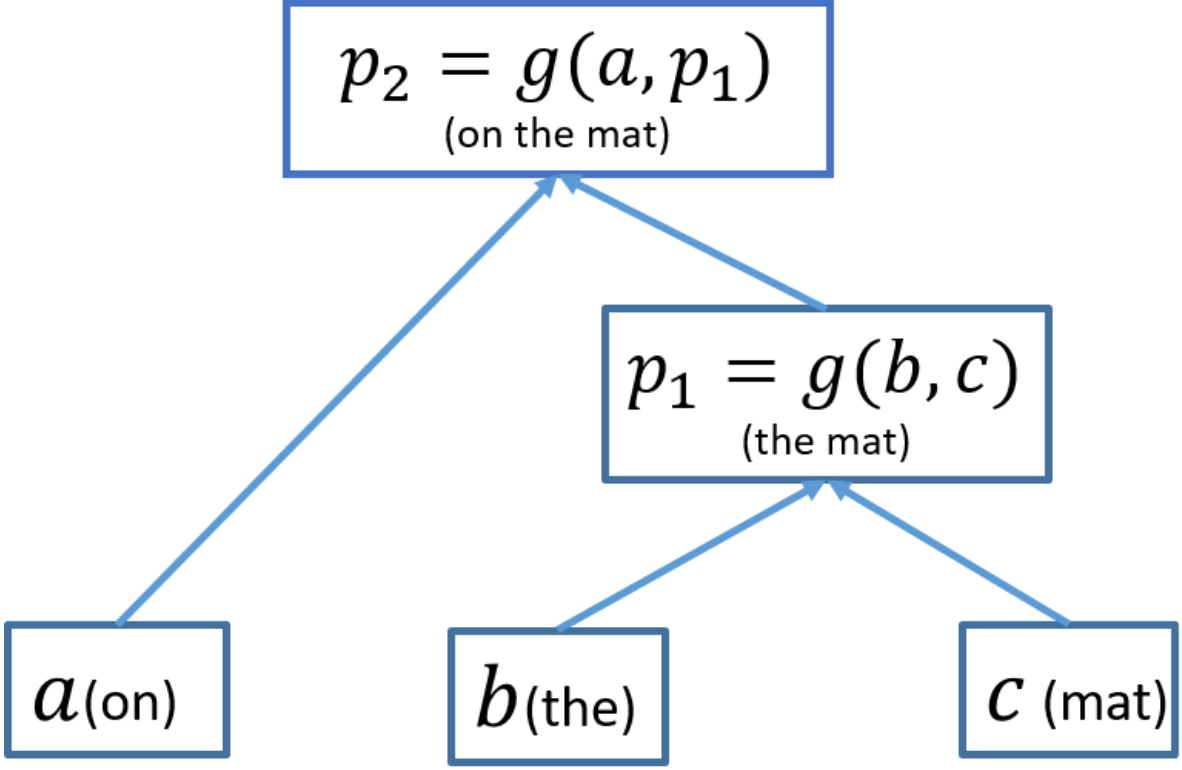


Figure 18: Recursive neural networks iteratively form high-level representation from lower-level representations.

In (Socher et al., 2013), the authors described multiple variations of this model. In its simplest form, is defined as:

in which the representation for each node is a -dimensional vector and .

Another variation is the MV-RNN (Socher et al., 2012). The idea is to represent every word and phrase as both a matrix and a vector. When two constituents are combined, the matrix of one is multiplied with the vector of the other:

in which , , and . Compared to the vanilla form, MV-RNN parameterizes the compositional function with matrices corresponding to the constituents.

The recursive neural tensor network (RNTN) is proposed to introduce more interaction between the input vectors without making the number of parameters exceptionally large like MV-RNN. RNTN is defined by:

where is a tensor that defines multiple bilinear forms.

B. Applications

One natural application of recursive neural networks is parsing (Socher et al., 2011). A scoring function is defined on the phrase representation

to calculate the plausibility of that phrase. Beam search is usually applied for searching the best tree. The model is trained with the max-margin objective (Taskar et al., 2004).

Based on recursive neural networks and the parsing tree, Socher et al. (2013)) proposed a phrase-level sentiment analysis framework (Figure 19), where each node in the parsing tree can be assigned a sentiment label.

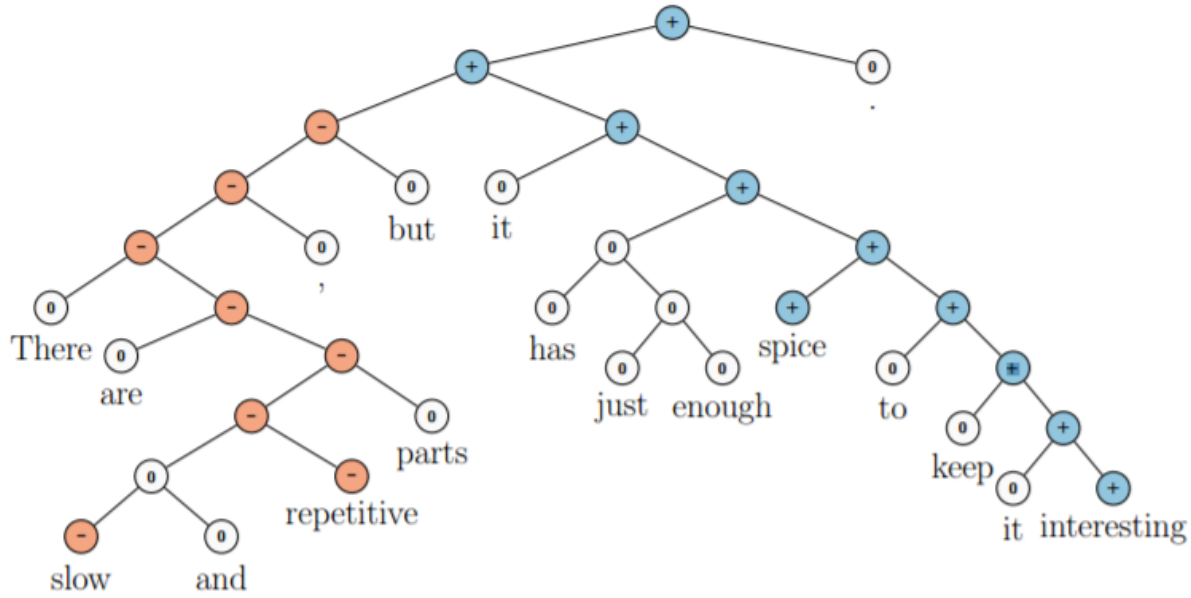


Figure 19: Recursive neural networks applied on a sentence for sentiment classification. Note that "but" plays a crucial role on determining the sentiment of the whole sentence (Figure source: Socher et al. (2013))

Socher et al. (2012) classified semantic relationships such as cause-effect or topic-message between nominals in a sentence by building a single compositional semantics for the minimal constituent including both terms. Bowman et al. (2014) proposed to classify the logical relationship between sentences with recursive neural networks. The representations for both sentences are fed to another neural network for relationship classification. They show that both vanilla and tensor versions of the recursive unit performed competitively in a textual entailment dataset.

To avoid the gradient vanishing problem, LSTM units have also been applied to tree structures in (Tai et al., 2015). The authors showed improved sentence representation over linear LSTM models, as clear improvement in sentiment analysis and sentence relatedness test was observed.

Chapter 6

Deep Reinforced Models and Deep Unsupervised Learning

A. Reinforcement learning for sequence generation

Reinforcement learning is a method of training an agent to perform discrete actions before obtaining a reward. In NLP, tasks concerning language generation can sometimes be cast as reinforcement learning problems.

In its original formulation, RNN language generators are typically trained by maximizing the likelihood of each token in the ground-truth sequence given the current hidden state and the previous tokens. Termed "teacher forcing", this training scheme provides the real sequence prefix to the generator during each generation (loss evaluation) step. At test time, however, ground-truth tokens are then replaced by a token generated by the model itself. This discrepancy between training and inference, termed "exposure bias" (Bengio et al., 2015; Ranzato et al., 2015), can yield errors that can accumulate quickly along the generated sequence.

Another problem with the word-level maximum likelihood strategy, when training auto-regressive language generation models, is that the training objective is different from the test metric. It is unclear how the n-gram overlap based metrics (BLEU, ROUGE) used to evaluate these tasks (machine translation, dialogue systems, etc.) can be optimized with the word-level training strategy. Empirically, dialogue systems trained with word-level maximum likelihood also tend to produce dull and short-sighted responses (Li et al., 2016), while text summarization tends to produce incoherent or repetitive summaries (Paulus et al., 2017).

Reinforcement learning offers a prospective to solve the above problems to a certain extent. In order to optimize the non-differentiable evaluation metrics directly, Ranzato et al. (2015) applied the REINFORCE algorithm (Williams, 1992) to train RNN-based models for several sequence generation tasks (e.g., text summarization, machine translation and image captioning), leading to improvements compared to previous supervised learning methods. In such a framework, the generative model (RNN) is viewed as an agent, which interacts with the external environment (the words and the context vector it sees as input at every time step). The parameters of this agent

defines a policy, whose execution results in the agent picking an action, which refers to predicting the next word in the sequence at each time step. After taking an action the agent updates its internal state (the hidden units of RNN). Once the agent has reached the end of a sequence, it observes a reward. This reward can be any developer-defined metric tailored to a specific task. For example, Li et al. (2016) defined 3 rewards for a generated sentence based on ease of answering, information flow, and semantic coherence.

There are two well-known shortcomings of reinforcement learning. To make reinforcement learning tractable, it is desired to carefully handle the state and action space (Young et al., 2010, 2013), which in the end may restrict expressive power and learning capacity of the model. Secondly, the need for training the reward functions makes such models hard to design and measure at run time (Su et al., 2011, 2016).

Another approach for sequence-level supervision is to use the adversarial training technique (Goodfellow et al., 2014), where the training objective for the language generator is to fool another discriminator trained to distinguish generated sequences from real sequences. The generator G and the discriminator D are trained jointly in a min-max game which ideally leads to G, generating sequences indistinguishable from real ones. This approach can be seen as a variation of generative adversarial networks in (Goodfellow et al., 2014), where G and D are conditioned on certain stimuli (for example, the source image in the task of image captioning). In practice, the above scheme can be realized under the reinforcement learning paradigm with policy gradient. For dialogue systems, the discriminator is analogous to a human Turing tester, who discriminates between human and machine-produced dialogues (Li et al., 2017).

B. Unsupervised sentence representation learning

Similar to word embeddings, distributed representation for sentences can also be learned in an unsupervised fashion. The result of such unsupervised learning are "sentence encoders", which map arbitrary sentences to fixed-size vectors that can capture their semantic and syntactic properties. Usually an auxiliary task has to be defined for the learning process.

Similar to the skip-gram model (Mikolov et al., 2013) for learning word embeddings, the skip-thought model (Kiros et al., 2015) was proposed for learning sentence representation, where the auxiliary task was to

predict two adjacent sentences (before and after) based on the given sentence. The seq2seq model was employed for this learning task. One LSTM encoded the sentence to a vector (distributed representation). Two other LSTMs decoded such representation to generate the target sequences. Standard seq2seq training process was used. After training, the encoder could be seen as a generic feature extractor (word embeddings were also learned in the same time).

Kiros et al. (2015) verified the quality of the learned sentence encoder on a range of sentence classification tasks, showing competitive results with a simple linear model based on the static feature vectors. However, the sentence encoder can also be fine-tuned in the supervised learning task as part of the classifier. Dai and Le (2015) investigated the use of the decoder to reconstruct the encoded sentence itself, which resembled an autoencoder (Rumelhart et al., 1985).

Language modeling could also be used as an auxiliary task when training LSTM encoders, where the supervision signal came from the prediction of the next token. Dai and Le (2015) conducted experiments on initializing LSTM models with learned parameters on a variety of tasks. They showed that pre-training the sentence encoder on a large unsupervised corpus yielded better accuracy than only pre-training word embeddings. Also, predicting the next token turned out to be a worse auxiliary objective than reconstructing the sentence itself, as the LSTM hidden state was only responsible for a rather short-term objective.

C. Deep generative models

Recent success in generating realistic images has driven a series of efforts on applying deep generative models to text data. The promise of such research is to discover rich structure in natural language while generating realistic sentences from a latent code space. In this section, we review recent research on achieving this goal with variational autoencoders (VAEs) (Kingma and Welling, 2013) and generative adversarial networks (GANs) (Goodfellow et al., 2014).

Standard sentence autoencoders, as in the last section, do not impose any constraint on the latent space, as a result, they fail when generating realistic sentences from arbitrary latent representations (Bowman et al., 2015). The representations of these sentences may often occupy a small region in the hidden space and most of regions in the hidden space do not necessarily map to a realistic sentence (Zhang

et al., 2016). They cannot be used to assign probabilities to sentences or to sample novel sentences (Bowman et al., 2015).

The VAE imposes a prior distribution on the hidden code space which makes it possible to draw proper samples from the model. It modifies the autoencoder architecture by replacing the deterministic encoder function with a learned posterior recognition model. The model consists of encoder and generator networks which encode data examples to latent representation and generate samples from the latent space, respectively. It is trained by maximizing a variational lower bound on the log-likelihood of observed data under the generative model.

Bowman et al. (2015) proposed an RNN-based variational autoencoder generative model that incorporated distributed latent representations of entire sentences (Figure 20). Unlike vanilla RNN language models, this model worked from an explicit global sentence representation. Samples from the prior over these sentence representations produced diverse and well-formed sentences.

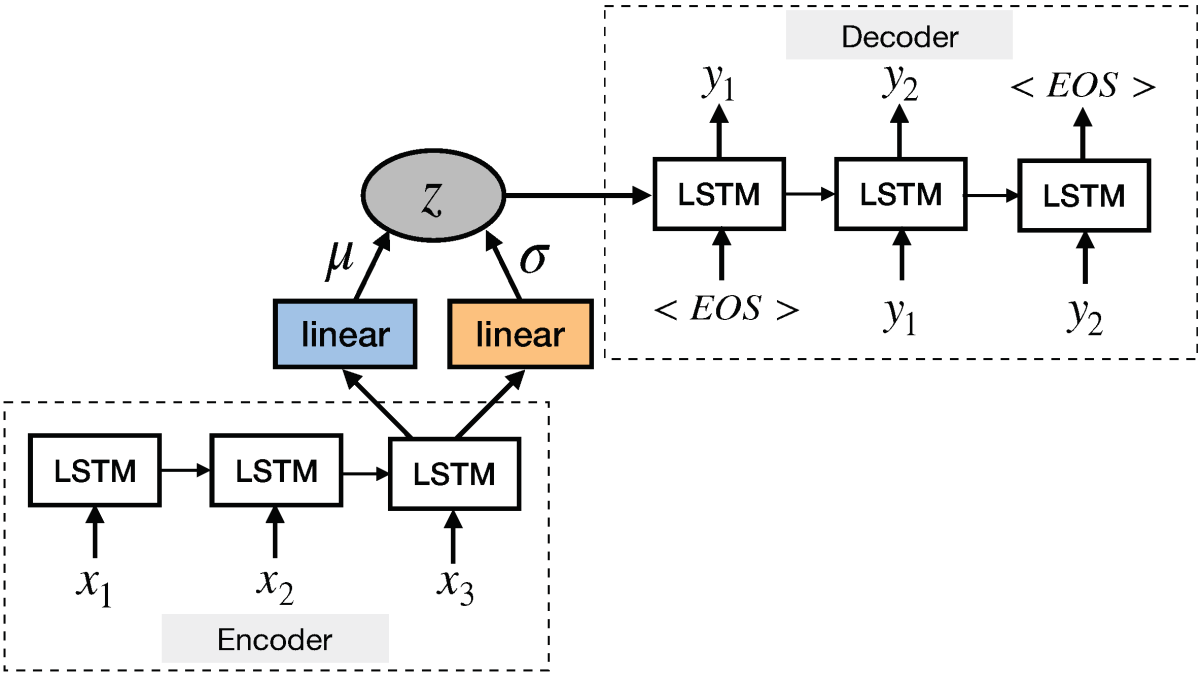


Figure 20: RNN-based VAE for sentence generation (Figure source: Bowman et al. (2015))

Hu et al. (2017) proposed generating sentences whose attributes are controlled by learning disentangled latent representations with designated semantics. The authors augmented the latent code in the VAE with a set of structured variables, each targeting a salient and independent semantic feature of sentences. The model incorporated VAE and attribute discriminators, in which the VAE component trained

the generator to reconstruct real sentences for generating plausible text, while the discriminators forced the generator to produce attributes coherent with the structured code. When trained on a large number of unsupervised sentences and a small number of labeled sentences, Hu et al. (2017) showed that the model was able to generate plausible sentences conditioned on two major attributes of English: tense and sentiment.

GAN is another class of generative model composed of two competing networks. A generative neural network decodes latent representation to a data instance, while the discriminative network is simultaneously taught to discriminate between instances from the true data distribution and synthesized instances produced by the generator. GAN does not explicitly represent the true data distribution .

Zhang et al (2016) proposed a framework for employing LSTM and CNN for adversarial training to generate realistic text. The latent code was fed to the LSTM generator at every time step. CNN acted as a binary sentence classifier which discriminated between real data and generated samples. One problem with applying GAN to text is that the gradients from the discriminator cannot properly back-propagate through discrete variables. In (Zhang et al., 2016), this problem was solved by making the word prediction at every time "soft" at the word embedding space. Yu et al. (2017) proposed to bypass this problem by modeling the generator as a stochastic policy. The reward signal came from the GAN discriminator judged on a complete sequence, and was passed back to the intermediate state-action steps using Monte Carlo search.

The evaluation of deep generative models has been challenging. For text, it is possible to create oracle training data from a fixed set of grammars and then evaluate generative models based on whether (or how well) the generated samples agree with the predefined grammar (Rajeswar et al., 2017). Another strategy is to evaluate BLEU scores of samples on a large amount of unseen test data. The ability to generate similar sentences to unseen real data is considered a measurement of quality (Yu et al., 2017).

Chapter 7

Memory-Augmented Networks

The attention mechanism stores a series of hidden vectors of the encoder, which the decoder is allowed to access during the generation of each token. Here, the hidden vectors of the encoder can be seen as entries of the model's "internal memory". Recently, there has been a surge of interest in coupling neural networks with a form of memory, which the model can interact with.

In (Weston et al., 2014), the authors proposed memory networks for QA tasks. In synthetic QA, a series of statements (memory entries) were provided to the model as potential supporting facts to the question. The model learned to retrieve one entry at a time from memory based on the question and previously retrieved memory. In large-scale realistic QA, a large set of commonsense knowledge in the form of (subject, relation, object) triples were used as memory. Sukhbaatar et al. (2015) extended this work and proposed end-to-end memory networks, where memory entries were retrieved in a "soft" manner with attention mechanism, thus enabling end-to-end training. Multiple rounds (hops) of information retrieval from memory were shown to be essential to good performance and the model was able to retrieve and reason about several supporting facts to answer a specific question (Figure 21). Sukhbaatar et al. (2015) also showed a special use of the model for language modeling, where each word in the sentence was seen as a memory entry. With multiple hops, the model yielded results comparable to deep LSTM models.

Story (1: 1 supporting fact)	Support	Hop 1	Hop 2	Hop 3
Daniel went to the bathroom.		0.00	0.00	0.03
Mary travelled to the hallway.		0.00	0.00	0.00
John went to the bedroom.		0.37	0.02	0.00
John travelled to the bathroom.	yes	0.60	0.98	0.96
Mary went to the office.		0.01	0.00	0.00
Where is John? Answer: bathroom	**Prediction: bathroom**			

Story (16: basic induction)	Support	Hop 1	Hop 2	Hop 3
Brian is a frog.	yes	0.00	0.98	0.00
Lily is gray.		0.07	0.00	0.00
Brian is yellow.	yes	0.07	0.00	1.00
Julius is green.		0.06	0.00	0.00
Greg is a frog.	yes	0.76	0.02	0.00
What color is Greg? Answer: yellow	**Prediction: yellow**			

Figure 21: Multiple supporting facts were retrieved from the memory in order to answer a specific question using an attention mechanism. The first hop uncovered the need for additional hops (Figure source: Sukhbaatar et al. (2015))

Furthermore, dynamic memory networks (DMN) (Kumar et al., 2015) improved upon previous memory-based models by employing neural network models for input representation, attention, and answer mechanisms. The resulting model was applicable to a wide range of NLP tasks (QA, POS tagging, and sentiment analysis), as every task could be cast to the memory, question, answer triple format. Xiong et al. (2016) applied the same model to visual QA and proved that the memory module was applicable to visual signals.

Chapter 8

Performance of Different Models on Different NLP Tasks

We summarize the performance of a series of deep learning methods on standard datasets developed in recent years on 7 major NLP topics in Tables 2-7. Our goal is to show the readers common datasets used in the community and state-of-the-art results with different models.

A. POS tagging

The WSJ-PTB (the Wall Street Journal part of the Penn Treebank Dataset) corpus contains 1.17 million tokens and has been widely used for developing and evaluating POS tagging systems. Giménez and Marquez (2004) employed one-against-all SVM based on manually-defined features within a seven-word window, in which some basic n-gram patterns were evaluated to form binary features such as: "previous word is *the*", "two preceding tags are DT NN", etc. One characteristic of the POS tagging problem was the strong dependency between adjacent tags. With a simple left-to-right tagging scheme, this method modeled dependencies between adjacent tags only by feature engineering. In an effort to reduce feature engineering, Collobert et al. (2011) relied on only word embeddings within the word window by a multi-layer perceptron. Incorporating CRF was proven useful in (Collobert et al., 2011). Santos and Zadrozny (2014) concatenated word embeddings with character embeddings to better exploit morphological clues. In (Santos and Zadrozny, 2014), the authors did not consider CRF, but since word-level decision was made on a context window, it can be seen that dependencies were modeled implicitly. Huang et al. (2015) concatenated word embeddings and manually-designed word-level features and employed bidirectional

LSTM to model arbitrarily long context. A series of ablative analysis suggested that bi-directionality and CRF both boosted performance. Andor et al. (2016) showed a transition-based approach that produces competitive result with a simple feed-forward neural network. When applied to sequence tagging tasks, DMNs (Kumar et al., 2015) essentially allowed for attending over the context multiple times by treating each RNN hidden state as a memory entry, each time focusing on different parts of the context.

Paper	Model	WSJ-PTB (per-token accuracy %)
Giménez and Marquez (2004)	SVM with manual feature pattern	97.16
Collobert et al. (2011)	MLP with word embeddings + CRF	97.29
Santos and Zadrozny (2014)	MLP with character+word embeddings	97.32
Huang et al. (2015)	LSTM	97.29
Huang et al. (2015)	Bidirectional LSTM	97.40
Huang et al. (2015)	LSTM-CRF	97.54
Huang et al. (2015)	Bidirectional LSTM-CRF	97.55
Andor et al. (2016)	Transition-based neural network	97.45
Kumar et al. (2015)	DMN	97.56

Table 2: POS tagging

B. Parsing

There are two types of parsing: dependency parsing, which connects individual words with their relations, and constituency parsing, which iteratively breaks text into sub-phrases. Transition-based methods are a popular choice since they are linear in the length of the sentence. The parser makes a series of decisions that read words sequentially from a buffer and combine them incrementally into the syntactic structure (Chen and Manning, 2014). At each time step, the decision is made based on a stack containing available tree nodes, a buffer containing unread words and the obtained set of dependency arcs. Chen and Manning (2014) modeled the decision making at each time step with a neural network with one hidden layer. The input layer contained embeddings of certain words, POS tags and arc labels, which came from the stack, the buffer and the set of arc labels.

Tu et al. (2015) extended the work of Chen and Manning (2014) by employing a deeper model with 2 hidden layers. However, both Tu et al. (2015) and Chen and Manning (2014) relied on manual feature selecting from the parser state, and they only took into account a limited number of latest tokens. Dyer et al. (2015) proposed stack-LSTMs to model arbitrarily long history. The end pointer of the stack changed position as the stack of tree nodes could be pushed and popped. Zhou et al. (2017) integrated beam search and contrastive learning for better optimization.

Transition-based models were applied to constituency parsing as well. Zhu et al. (2013) based each transition action on features such as the POS tags and constituent labels of the top few words of the stack and the buffer. By uniquely representing the parsing tree with a linear sequence of labels, Vinyals et al. (2015) applied the seq2seq learning method to this problem.

Paper	Model	WSJ (UAS/LAS)
Chen and Manning (2014)	Fully-connected NN with features including POS	91.80/89.60
Weiss et al. (2015)	Deep fully-connected NN with features including POS	94.30/92.40
Dyer et al. (2015)	Stack-LSTM	93.10/90.90
Zhou et al. (2017)	Beam contrastive model	93.31/92.37

Table 3.1: Dependency Parsing (UAS/LAS = Unlabeled/labeled Attachment Score; WSJ = The Wall Street Journal Section of Penn Treebank)

Paper	Model	WSJ (F1 %)
Petrov et al. (2006)	Probabilistic context-free grammars (PCFG)	91.80
Socher et al. (2011)	Recursive neural networks	90.29
Zhu et al. (2013)	Feature-based transition parsing	91.30
Vinyals et al. (2015)	seq2seq learning with LSTM+Attention	93.50

Table 3.2: Constituency Parsing

C. Named-Entity Recognition

CoNLL 2003 has been a standard English dataset for NER, which concentrates on four types of named entities: people, locations, organizations and miscellaneous entities. NER is one of the NLP problems where lexicons can be very useful. Collobert et al. (2011) first achieved competitive results with neural structures augmented by gazetteer features. Chiu and Nichols (2015) concatenated lexicon features, character embeddings and word embeddings and fed them as input to a bidirectional LSTM. On the other hand, Lample et al. (2016) only relied on character and word embeddings, with pre-training embeddings on large unsupervised corpora, they achieved competitive results without using any lexicon. Similar to POS tagging, CRF also boosted the performance of NER, as demonstrated by the comparison in (Lample et al., 2016). Overall, we see that bidirectional LSTM with CRF acts as a strong model for NLP problems related to structured prediction.

Passos et al. (2014) proposed to modify skip-gram models to better learn entity-type related word embeddings that can leverage information from relevant lexicons. Luo et al. (2015) jointly optimized the entities and the linking of entities to a KB. Strubell et al. (2017) proposed to use dilated convolutions, defined over a wider effective input width by skipping over certain inputs at a time, for better parallelization and context modeling. The model showed significant speedup while retaining accuracy.

Paper	Model	CoNLL 2003 (F1 %)
Collobert et al. (2011)	MLP with word embeddings+gazetteer	89.59
Passos et al. (2014)	Lexicon Infused Phrase Embeddings	90.90
Chiu and Nichols (2015)	Bi-LSTM with word+char+lexicon embeddings	90.77
Luo et al. (2015)	Semi-CRF jointly trained with linking	91.20
Lample et al. (2016)	Bi-LSTM with word+char embeddings	89.15
Lample et al. (2016)	Bi-LSTM-CRF with word+char embeddings	90.94
Strubell et al. (2017)	Dilated CNN with CRF	90.54

Table 4: Named-Entity Recognition

D. Semantic Role Labeling

Semantic role labeling (SRL) aims to discover the predicate-argument structure of each predicate in a sentence. For each target verb (predicate), all constituents in the sentence which take a semantic role of the verb are recognized. Typical semantic arguments include Agent, Patient, Instrument, etc., and also adjuncts such as Locative, Temporal, Manner, Cause, etc. (Zhou and Xu, 2015). Table 5 shows the performance of different models on the CoNLL 2005 & 2012 datasets.

Traditional SRL systems consist of several stages: producing a parse tree, identifying which parse tree nodes represent the arguments of a given verb, and finally classifying these nodes to determine the corresponding SRL tags. Each classification process usually entails extracting numerous features and feeding them into statistical models (Collobert et al., 2011).

Given a predicate, Täckström et al. (2015) scored a constituent span and its possible role to that predicate with a series of features based on the parse tree. They proposed a dynamic programming algorithm for efficient inference. Collobert et al. (2011) achieved comparable results with a convolution neural networks augmented by parsing information provided in the form of additional look-up tables. Zhou and Xu (2015) proposed to use bidirectional LSTM to model arbitrarily long context, which proved to be successful without any parsing tree information. He et al. (2017) further extended this work by introducing highway connections (Srivastava et al., 2015), more advanced regularization and ensemble of multiple experts.

Paper	Model	CoNLL 2005 (F1 %)	CoNLL 2012 (F1 %)
Collobert et al. (2011)	CNN with parsing features	76.06	
Täckström et al. (2015)	Manual features with DP for inference	78.60	79.40
Zhou and Xu (2015)	Bidirectional LSTM	81.07	81.27
He et al. (2017)	Bidirectional LSTM with highway connections	83.20	83.40

Table 5: Semantic Role Labeling

E. Sentiment Classification

The Stanford Sentiment Treebank (SST) dataset contains sentences taken from the movie review website Rotten Tomatoes. It was proposed by Pang and Lee (2005) and subsequently extended by Socher et al. (2013). The annotation scheme has inspired a new dataset for sentiment analysis, called CMU-MOSI, where sentiment is studied in a multimodal setup (Zadeh et al., 2013).

(Socher et al., 2013) and (Tai et al., 2015) were both recursive networks that relied on constituency parsing trees. Their difference shows the effectiveness of LSTM over vanilla RNN in modeling sentences. On the other hand, tree-LSTM performed better than linear bidirectional LSTM, implying that tree structures can potentially better capture the syntactical property of natural sentences. Yu et al. (2017) proposed to refine pre-trained word embeddings with a sentiment lexicon, observing improved results based on (Tai et al., 2015).

Kim (2014) and Kalchbrenner et al. (2014) both used convolutional layers. The model (Kim, 2014) was similar to the one in Figure 5, while Kalchbrenner et al. (2014) constructed the model in a hierarchical manner by interweaving k-max pooling layers with convolutional layers.

Paper	Model	SST-1	SST-2
Socher et al. (2013)	Recursive Neural Tensor Network	45.7	85.4
Kim (2014)	Multichannel CNN	47.4	88.1
Kalchbrenner et al. (2014)	DCNN with k-max pooling	48.5	86.8
Tai et al. (2015)	Bidirectional LSTM	48.5	87.2
Le and Mikolov (2014)	Paragraph Vector	48.7	87.8
Tai et al. (2015)	Constituency Tree-LSTM	51.0	88.0
Yu et al. (2017)	Tree-LSTM with refined word embeddings	54.0	90.3
Kumar et al. (2015)	DMN	52.1	88.6

Table 6: Sentiment Classification (SST-1 = Stanford Sentiment Treebank, fine-grained 5 classes Socher et al. (2013); SST-2: the binary version of SST-1; Numbers are accuracies (%))

F. Machine Translation

The phrase-based SMT framework (Koehn et al., 2003) factorized the translation model into the translation probabilities of matching

phrases in the source and target sentences. Cho et al. (2014) proposed to learn the translation probability of a source phrase to a corresponding target phrase with an RNN encoder-decoder. Such a scheme of scoring phrase pairs improved translation performance. Sutskever et al. (2014), on the other hand, re-scored the top 1000 best candidate translations produced by an SMT system with a 4-layer LSTM seq2seq model. Dispensing the traditional SMT system entirely, Wu et al. (2016) trained a deep LSTM network with 8 encoder and 8 decoder layers with residual connections as well as attention connections. (Wu et al., 2016) then refined the model by using reinforcement learning to directly optimize BLEU scores, but they found that the improvement in BLEU scores by this method did not reflect in human evaluation of translation quality. Recently, Gehring et al. (2017) proposed a CNN-based seq2seq learning model for machine translation. The representation for each word in the input is computed by CNN in a parallelized style for the attention mechanism. The decoder state is also determined by CNN with words that are already produced. Vaswani et al. (2017) proposed a self-attention-based model and dispensed convolutions and recurrences entirely.

Paper	Model	WMT2014 English2German	WMT2014 English2French
Cho et al. (2014)	Phrase table with neural features		34.50
Sutskever et al. (2014)	Reranking phrase-based SMT best list with LSTM seq2seq		36.50
Wu et al. (2016)	Residual LSTM seq2seq + Reinforcement learning refining	26.30	41.16
Gehring et al. (2017)	seq2seq with CNN	26.36	41.29
Vaswani et al. (2017)	Attention mechanism	28.40	41.00

Table 7: Machine translation (Numbers are BLEU scores)

G. Question Answering

QA problems take many forms. Some rely on large KBs to answer open-domain questions, while others answer a question based on a few sentences or a paragraph (reading comprehension). For the former, we list (see Table 8) several experiments conducted on a large-scale QA

dataset introduced by (Fader et al., 2013), where 14M commonsense knowledge triples are considered as the KB. Each question can be answered with a single-relation query. For the latter, we consider (see Table 8) (1) the synthetic dataset of *bAbI* (Weston et al., 2015), which requires the model to reason over multiple related facts to produce the right answer. It contains 20 synthetic tasks that test a model's ability to retrieve relevant facts and reason over them. Each task focuses on a different skill such as *basic coreference* and *size reasoning*. (2) The Stanford Question Answering Dataset (*SQuAD*) (Rajpurkar et al., 2016), consisting of 100,000+ questions posed by crowdworkers on a set of Wikipedia articles. The answer to each question is a segment of text from the corresponding article.

The central problem of learning to answer single-relation queries is to find the single supporting fact in the database. Fader et al. (2013) proposed to tackle this problem by learning a lexicon that maps natural language patterns to database concepts (entities, relations, question patterns) based on a question paraphrasing dataset. Bordes et al. (2014) embedded both questions and KB triples as dense vectors and scored them with inner product.

Weston et al. (2014) took a similar approach by treating the KB as long-term memory, while casting the problem in the framework of a memory network. On the *bAbI* dataset, Sukhbaatar et al. (2015) improved upon the original memory networks model (Weston et al., 2014) by making the training procedure agnostic of the actual supporting fact, while Kumar et al. (2015) used neural sequence models (GRU) instead of neural bag-of-words models as in (Sukhbaatar et al., 2015) and (Weston et al., 2014) to embed memories.

For models on the *SQuAD* dataset, the goal is to determine the start point and end point of the answer segment. Chen et al. (2017) encoded both the question and the words in context using LSTMs and used a bilinear matrix for calculating the similarity between the two. Shen et al. (2017) proposed Reasonet, a model that read a document repeatedly with attention on different parts each time until a satisfying answer is found. Yu et al., 2018 replaced RNNs with convolution and self-attention for encoding the question and the context with significant speed improvement.

Paper	Model	bAbI (Mean accuracy %)	Farbes (Accuracy %)	SQuAD (EM/F1 %)

Paper	Model	bAbI (Mean accuracy %)	Farbes (Accuracy %)	SQuAD (EM/F1 %)
Fader et al. (2013)	Paraphrase-driven lexicon learning		0.54	
Bordes et al. (2014)	Weekly supervised embedding		0.73	
Weston et al. (2014)	Memory networks	93.3	0.93	
Sukhbaatar et al. (2015)	End-to-end memory networks	88.4		
Kumar et al. (2015)	DMN	93.6		
Chen et al. (2017)	Document Reader			70.0/79.0
Shen et al. 2016)	ReasoNet			69.1/78.9
Yu et al. (2018)	QAnet			76.2/84.6

Table 8: Question answering

H. Dialogue Systems

Two types of dialogue systems have been developed: generation-based models and retrieval-based models.

In Table 9, the Twitter Conversation Triple Dataset is typically used for evaluating generation-based dialogue systems, containing 3-turn Twitter conversation instances. One commonly used evaluation metric is BLEU (Papineni et al., 2002), although it is commonly acknowledged that most automatic evaluation metrics are not completely reliable for dialogue evaluation and additional human evaluation is often necessary. Ritter et al. (2011) employed the phrase-based statistical machine translation (SMT) framework to "translate" the message to its appropriate response. Sordoni et al. (2015) reranked the 1000 best responses produced by SMT with a context-sensitive RNN encoder-decoder framework, observing substantial gains. Li et al. (2015) reported results on replacing the traditional maximum log likelihood training objective with the maximum mutual information training objective, in an effort to produce interesting and diverse responses, both of which are tested on a 4-layer LSTM encoder-decoder framework.

The response retrieval task is defined as selecting the best response from a repository of candidate responses. Such a model can be evaluated by the recall1@ metric, where the ground-truth response is mixed with random responses. The Ubuntu dialogue dataset was constructed by scraping multi-turn Ubuntu trouble-shooting dialogues from an online chatroom (Lowe et al., 2015). Lowe et al. (2015) used LSTMs to encode the message and response, and then inner product of the two sentence embeddings is used to rank candidates.

Zhou et al. (2016) proposed to better exploit the multi-turn nature of human conversation by employing the LSTM encoder on top of sentence-level CNN embeddings, similar to (Serban et al., 2016). Dodge et al. (2015) cast the problem in the framework of a memory network, where the past conversation was treated as memory and the latest utterance was considered as a "question" to be responded to. The authors showed that using simple neural bag-of-word embedding for sentences can yield competitive results.

Paper	Model	Twitter Conversation Triple Dataset (BLEU)	Ubuntu Dialogue Dataset (recall 1@10 %)
Ritter et al. (2011)	SMT	3.60	
Sordoni et al. (2015)	SMT+neural reranking	4.44	
Li et al. (2015)	LSTM seq2seq	4.51	
Li et al. (2015)	LSTM seq2seq with MMI objective	5.22	
Lowe et al. (2015)	Dual LSTM encoders for semantic matching		55.22
Dodge et al. (2015)	Memory networks		63.72
Zhou et al. (2016)	Sentence-level CNN-LSTM encoder		66.15

Table 9: Dialogue systems

I. Contextual Embeddings

In this section, we explore some of the recent results based on contextual embeddings as explained in section 2-D. ELMo has contributed significantly towards the recent advancement of NLP. In various NLP tasks, ELMo outperformed state of the art by significant margin (Table 10). However, latest mode BERT surpass ELMo to establish itself as the state-of-the-art in multiple tasks as summarized in Table 11.

Task	Previous SOTA	Previous SOTA Results	Baseline	ELMo + Baseline	Increase (Absolute/Relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.70 (+- 0.17)	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.3 (+- 0.19)	90.15	92.22 (+- 0.10)	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 (+- 0.5)	3.3 / 6.8%

Table 10: Comparison of ELMo + Baseline with the previous state of the art (SOTA) on various NLP tasks. The table has been adapted from Peters et al. (2018). SOTA results have been taken from Peters et al. (2018); SQUAD (Rajpurkar et al., 2016): QA task; SNLI (Bowman et al., 2015): Stanford Natural Language Inference task; SRL (Zhou and Xu, 2015): Semantic Role Labelling; Coref (Pradhan et al., 2012): Coreference Resolution; NER (Tjong et al., 2003): Named Entity Recognition; SST-5 (Socher et al., 2013): Stanford Sentiment Treebank 5-class classification;

Task	BiLSTM + ELMo + Attn	BERT
QNLI	79.9	91.1
SST-2	90.9	94.9
STS-B	73.3	86.5
RTE	56.8	70.1
SQuAD	85.8	91.1
NER	92.2	92.8

Table 11: QNLI (Wang et al., 2018): Question Natural Language Inference task; SST-2 (Socher et al., 2013): Stanford Sentiment Treebank binary classification; STS-B (Cer et al., 2017): Semantic Textual Similarity Benchmark; RTE (Bentivogli and Clark, 2009): Recognizing Textual Entailment; SQuAD (Rajpurkar et al., 2016): QA task; NER (Tjong et al., 2003): Named Entity Recognition.

Chapter 9

Conclusion

Deep learning offers a way to harness large amount of computation and data with little engineering by hand (LeCun et al., 2015). With distributed representation, various deep models have become the new state-of-the-art methods for NLP problems. Supervised learning is the most popular practice in recent deep learning research for NLP. In many real-world scenarios, however, we have unlabeled data which require advanced unsupervised or semi-supervised approaches. In cases where there is lack of labeled data for some particular classes or the appearance of a new class while testing the model, strategies like zero-shot learning should be employed. These learning schemes are still in their developing phase but we expect deep learning based NLP research to be driven in the direction of making better use of unlabeled data. We expect such trend to continue with more and better model designs. We expect to see more NLP applications that employ reinforcement learning methods, e.g., dialogue systems. We also expect to see more research on multimodal learning (Baltrušaitis et al., 2017) as, in the real world, language is often grounded on (or correlated with) other signals.

Finally, we expect to see more deep learning models whose internal memory (bottom-up knowledge learned from the data) is enriched with an external memory (top-down knowledge inherited from a KB). Coupling symbolic and sub-symbolic AI will be key for stepping forward in the path from NLP to natural language understanding. Relying on machine learning, in fact, is good to make a ‘good guess’ based on past experience, because sub-symbolic methods encode correlation and their decision-making process is probabilistic. Natural language understanding, however, requires much more than that. To use Noam Chomsky’s words, “you do not get discoveries in the sciences by taking huge amounts of data, throwing them into a computer and

doing statistical analysis of them: that's not the way you understand things, you have to have theoretical insights".
