Interpretable Neural Models for Natural Language Processing

by

Tao Lei

B.S., Peking University (2010)
M.S., Massachusetts Institute of Technology (2013)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2017

Author .... Signature redacted ....
Department of Electrical Engineering and Computer Science
January 20, 2017

Certified by...... Signature redacted ....
Regina Barzilay
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by......... Signature redacted ....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair of the Committee on Graduate Students

Interpretable Neural Models for Natural Language Processing

by

Tao Lei

Submitted to the Department of Electrical Engineering and Computer Science
on January 20, 2017, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

The success of neural network models often comes at a cost of interpretability. This thesis addresses the problem by providing justifications behind the model's structure and predictions.

In the first part of this thesis, we present a class of sequence operations for text processing. The proposed component generalizes from convolution operations and gated aggregations. As justifications, we relate this component to string kernels, i.e. functions measuring the similarity between sequences, and demonstrate how it encodes the efficient kernel computing algorithm into its structure. The proposed model achieves state-of-the-art or competitive results compared to alternative architectures (such as LSTMs and CNNs) across several NLP applications.

In the second part, we learn rationales behind the model's prediction by extracting input pieces as supporting evidence. Rationales are tailored to be short and coherent, yet sufficient for making the same prediction. Our approach combines two modular components, generator and encoder, which are trained to operate well together. The generator specifies a distribution over text fragments as candidate rationales and these are passed through the encoder for prediction. Rationales are never given during training. Instead, the model is regularized by the desiderata for rationales. We demonstrate the effectiveness of this learning framework in applications such multi-aspect sentiment analysis. Our method achieves a performance over 90% evaluated against manual annotated rationales.

Thesis Supervisor: Regina Barzilay
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

The six year of work and study in the MIT NLP group has been a wonderful and unforgettable experience. I am forever grateful to my advisor Regina Barzilay, who offered me the chance to be part of the family. Her vision, advice, high standards and encouragement are invaluable to me, which has led to all the exciting research and, has made me a researcher. I could not have asked for a better advisor.

I have been also extremely fortunate to work with Tommi Jaakkola, whose brilliance has always brought me inspirations. My work would not have been possible without Regina and Tommi.

I am also very thankful to my wonderful thesis committee and collaborators – Jim Glass, Yuan Zhang, Fan Long, Martin Rinard, Yu Xin, Lluís Màrquez, Alessandro Moschitti, S.R.K. Branavan, Nate Kushman, Yonatan Belinkov, Tianxiao Shen, Hrishikesh Joshi, Youyang Gu and Amir Globerson, for their significant input or guidance on my research. I would like to thank my friends and my labmates, Yu Zhang, Danqi Chen, Karthik Narasimhan, Tahira Naseem, Yoong Keok Lee, Yevgeni Berzak, Zachary Hynes, Jiaming Luo, David Alverez Melis, Adam Yala, Nick Locasio, Wengong Jin, Darsh Shah, Christina Sauper, Kevin Yang, Abdi Dirie, Eduardo De Leon and Benson Chen, for their help and a lot interesting conversations and activities with them during my PhD study. A special thanks to our administrative assistant Marcia Davidson for her help.

I dedicate this thesis to my wonderful family: My grandpa Keqiu and my mother Yueming for their great love and support; my wife Lang for her love and company; and all the family members.

Bibliographic Note

Portions of this thesis are based on prior peer-reviews publications. Most of the work presented in Chapter 2 was originally published in Lei et al. [66] and Lei et al. [67]. We include an additional theoretical interpretation (Section 2.4) in this chapter. Chapter 3 was published in Lei et al. [64].

The code and data of the work presented in this thesis are available at `https://github.com/taolei87/rcnn`.

Contents

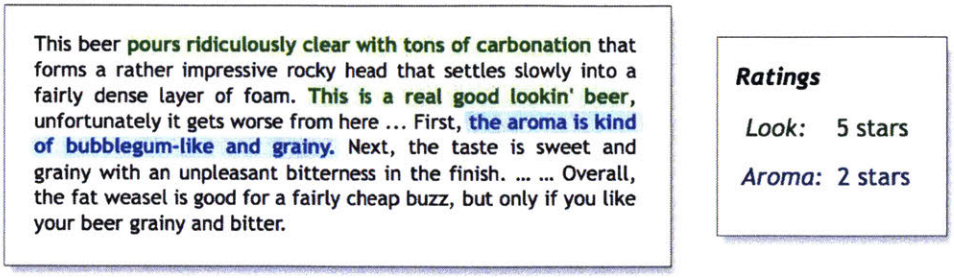
List of Figures

List of Tables

1
Introduction

Deep learning and neural networks have become de facto top performing techniques in the field of natural language processing, computer vision, and machine learning. As methods, they require only limited domain knowledge to reach respectable performance with increasing data and computation, yet permit easy architectural and operational variations to explore and to tune for top performance. In natural language processing (NLP) for instance, neural network models have been applied to applications such as sentiment classification, parsing, and machine translation among many others, where carefully-chosen neural components such as long short-term memory networks (LSTMs) [42] were shown to achieve state-of-the-art performance.

The improvement in performance, however, comes at the cost of interpretability since complex neural models offer little transparency concerning their inner workings. Consider text classification for example. A typical neural network model would take the word vectors from the input text and apply non-linear transformations over the vectors. The transformation process can be further repeated via recurrence or recursion of the network, before reaching the final classification prediction. As a consequence of this complicated procedure, the model often lacks a good explanation or understanding of its computation. This could be problematic for developing new methods to real-world applications. For example, researchers need understanding of the computation and architecture in order to extend and improve the methods. In addition, ordinary users often require justifications for the model's prediction in many application scenarios. Ideally, powerful neural models would not only yield improved



This beer pours ridiculously clear with tons of carbonation that forms a rather impressive rocky head that settles slowly into a fairly dense layer of foam. This is a real good lookin' beer, unfortunately it gets worse from here ... First, the aroma is kind of bubblegum-like and grainy. Next, the taste is sweet and grainy with an unpleasant bitterness in the finish. ... ... Overall, the fat weasel is good for a fairly cheap buzz, but only if you like your beer grainy and bitter.

Ratings	
Look:	5 stars
Aroma:	2 stars

Figure 1-1: An example of product review and associated aspect ratings. Rationales for each aspect rating are highlighted in the text.

performance, but also provide some interpretability in two regards:

- **Intepretable Component:** One kind of interpretability is being able to understand the fundamentals of certain neural models, including the architecture and the inner computation. This is important since the success of neural methods is often contingent on specific operational and architectural choices of the neural model. While it is possible to apply heuristic search over such choices for good performance [7, 48, 5, 123], the procedure is extremely inefficient and provide no experience on the applicability of the model. Hence, the expensive search procedure has to repeat for every new application. In contrast, if a model and the associated computation is better understood, we can effectively explore the choices since the insight of the model would reveal whether the component can work, and if not, what variations can work. Despite some recent efforts, most understanding of neural NLP models is empirical, focusing on performance analysis and visualization [33, 16, 51, 70]. To facilitate the development of NLP models, it is necessary to design *interpretable neural components* that can be both intuitively and theoretically justified.

- **Interpretable Prediction:** Another form of interpretability is being able to provide human-readable justifications – *rationales* – that support the model's prediction. In many applications, such as medicine, predictions are used to drive critical decisions, including treatment options. It is necessary in such cases to be able to verify and understand the underlying basis for the decisions. In product review analysis, for instance, users often want to know the reasons a product gets a positive or negative rating. Figure 1-1 illustrates a product review along with ratings on two categories or aspects. If a model in this case predicts such ratings for the two aspects, it could also identify the three highlighted text pieces as supporting rationales underlying the decisions. For example, the first two pieces, "pours ridiculously clear with tons of carbonation" and "this is a real good lookin' beer", explains the 5-star rating on the appearance aspect.

In this dissertation, I present several deep learning techniques to address the above interpretability challenges. First, I present and analyze a new class of neural operations that is specifically designed for sequential data such as text. The operation builds on traditional convolution operation [62], but incorporates two modifications that improve empirical performance on several NLP tasks. Further more, this component can be *theoretically interpreted* as a parameterized function in the reproducing kernel Hilbert space (RKHS) of string kernels. This connection provides deeper understanding of the network and explains its effectiveness on sequential modeling tasks. Secondly, with such operational component as the building block, I design a novel model framework and the associated learning method for prediction interpretability. The framework incorporates rationale generation as an integral part of the overall learning problem. The learning method optimizes the model so as to produce accurate predictions and concise rationales from the input.

I briefly describe these techniques and the evaluation below.

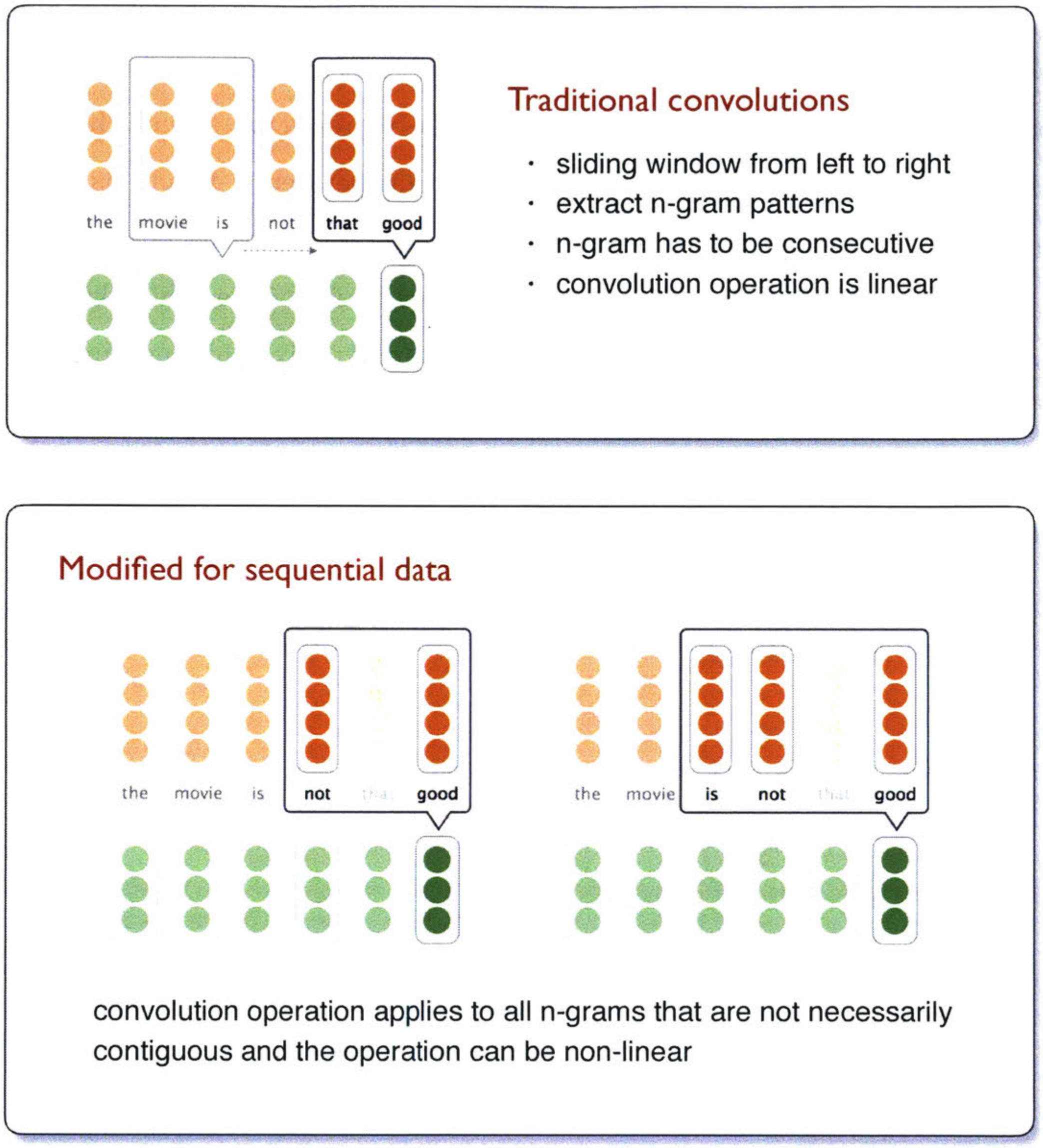
1.1 Interpretable Neural Component

Recurrent Convolution Our proposed interpretable component extends traditional convolutional neural networks to better adapt it to text processing. CNNs for text applications make use of temporal convolution operators or filters. Similar to image processing, they are applied at multiple resolutions, interspersed with non-linearities and pooling. The convolution operation itself is a linear mapping over "n-gram vectors" obtained by concatenating consecutive word (or character) representations. We argue that this basic building block can be improved in two important respects. First, many useful patterns are expressed as non-consecutive phrases, such as semantically close multi-word expressions (e.g., "not that good", "not nearly as good"). In typical CNNs, such expressions would have to come together and emerge as useful patterns after several layers of processing. Secondly, the power of n-grams derives precisely from multi-way interactions and these are clearly missed (initially) with linear operations on stacked n-gram vectors. Hence, a non-linear mapping could potentially capture more interaction between n-grams, and improve the performance in the target task [78, 52, 97].

We propose to apply feature extraction and aggregation over all n-grams that are not necessarily contiguous in the sequence. This operation, which we call *recurrent convolution*, can be performed efficiently in a *recurrent manner* via a dynamic programming-style implementation. In addition, we employ a feature extraction operation based on tensor products over word vectors, which enables us to directly tap into non-linear interactions between adjacent words. Figure 1-2 illustrates and compares the traditional convolutions and the modified version.

We evaluate models built from recurrent convolution components on several NLP tasks such as sentiment prediction and language modeling. Our model obtains state-of-the-art performance over many competitive baselines such as CNNs, GRUs and LSTMs [42, 16, 50, 55, 105, 97].

Theoretical Interpretation In addition to the intuitive justification and empirical evaluation, we also theoretically relate the inner working (computation) of the



Traditional convolutions

- sliding window from left to right
- extract n-gram patterns
- n-gram has to be consecutive
- convolution operation is linear

Modified for sequential data

convolution operation applies to all n-grams that are not necessarily contiguous and the operation can be non-linear

Figure 1-2: An illustration of traditional convolution operation and the recurrent convolution. The former uses a fixed-size sliding window that moves from left to right, and extracts features within the window (i.e. local context). The recurrent version works in a recurrent manner to pull n-gram features that may not be consecutive. In addition, the feature mapping could be non-linear.

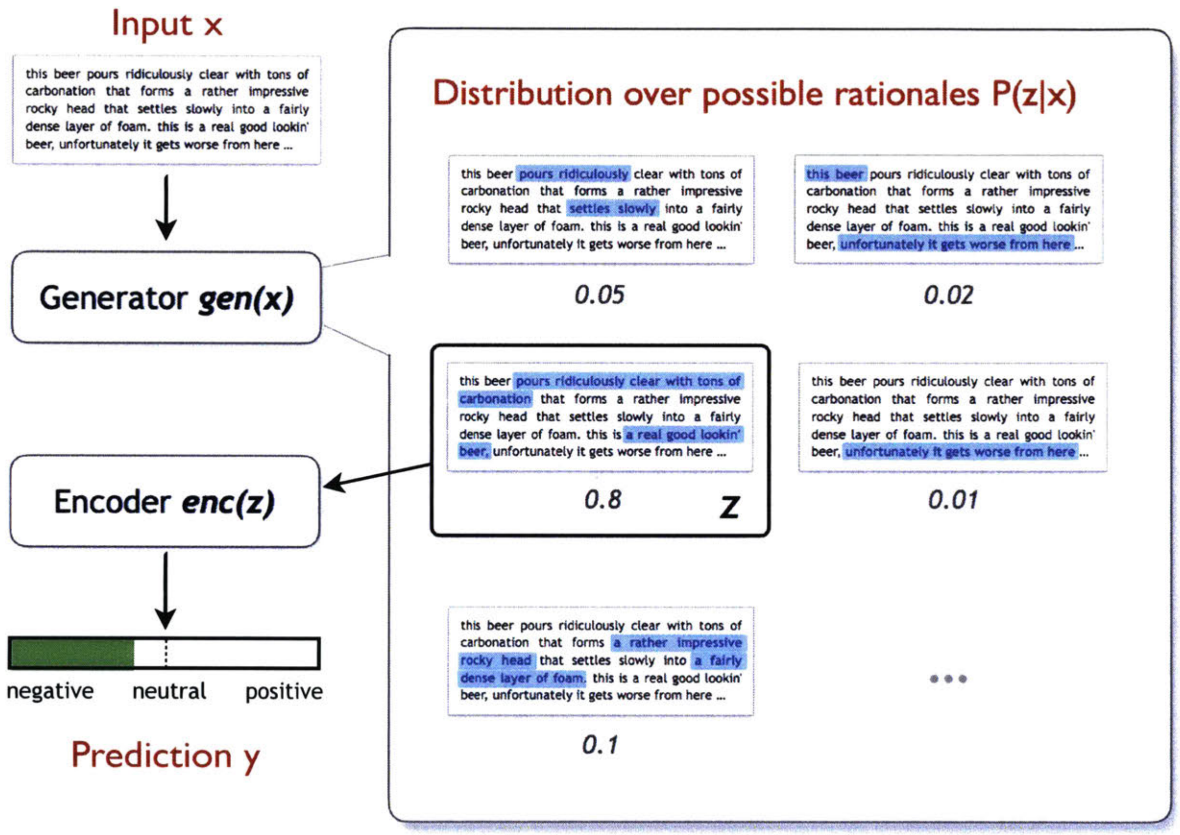
recurrent convolution module to traditional kernel methods such as string kernel [71]. This result follows recent work that shows the connection between kernels and feed-forward neural networks [35, 120]. In particular, we show that prediction functions constructed by recurrent convolutions lie in the reproducing kernel Hilbert space (RKHS) of string kernel (or its composition with other kernel functions when non-linear activation is involved). Training the neural model of this kind thus can be understood as seeking a good predictor in the space introduced by a kernel, while posing additional dimension-reduction constraint (due to fixed hidden dimension size and number of parameters). This explains the empirical effectiveness of our component for sequential model since string kernels are naturally functions measuring the similarity between sequences. Dimension reduction also explains the better generalization of the model – it is a regularization technique that was shown useful in many previous work [63, 99, 100, 27, 65].

1.2 Interpretable Prediction

Extractive Rationale Generation Prediction without justification has limited applicability. As a remedy, we propose a novel approach that learns to provide human-readable rationales as the basis of the model prediction. We formulate rationale generation as an integral part of the overall learning problem. We limit ourselves to extractive (as opposed to abstractive) rationales. From this perspective, our rationales are simply subsets of the words from the input text that satisfy two key properties. First, the selected words represent short and coherent pieces of text (e.g., phrases) and, second, the selected words must alone suffice for prediction as a substitute of the original text. In most practical applications, rationale generation must be learned entirely in an unsupervised manner. We therefore assume that our model with rationales is trained on the same data as the original neural models, without access to additional rationale annotations. In other words, target rationales are never provided during training; the intermediate step of rationale generation is guided only by the two desiderata discussed above.

Encoder-Generator Framework As shown in Figure 1-3, our model is composed of two modular components that we call the *generator* and the *encoder*. Our generator specifies a distribution over possible rationales (extracted text) and the encoder maps any such text to task specific target values. They are trained jointly to minimize a cost function that favors short, concise rationales while enforcing that the rationales alone suffice for accurate prediction.

The framework is evaluated on several real-world applications and datasets such as multi-aspect sentiment analysis on beer reviews [74]. Our approach outperforms attention-based baseline and others by a significant margin. For instance, the beer review corpus contains human annotation which identifies, for each aspect, the sentence(s) that relate to this aspect. We can therefore directly evaluate our extracted rationales using such annotation. Our model achieves extraction accuracy of 96%, as compared to 38% and 81% obtained by the bigram SVM and a neural attention baseline.



Input x
this beer pours ridiculously clear with tons of carbonation that forms a rather impressive rocky head that settles slowly into a fairly dense layer of foam. this is a real good lookin' beer, unfortunately it gets worse from here ...
Generator gen(x)
Encoder enc(z)
negative
neutral
positive
Prediction y
Distribution over possible rationales P(z|x)
0.05
0.02
0.8
z
0.01
0.1
...

Figure 1-3: An overview of the model framework for learning rationales. The generator module takes the original input and produces a distribution of possible rationales, e.g. possible word segments to extract. The encoder module takes the selected rationale and makes the prediction for the target task. The two components are optimized jointly during training.

1.3 Contributions

The primary contributions of this work are:

- **Designing neural components for text** We propose a class of sequence component for text processing, i.e. mapping texts into vector representations. The component generalizes from convolution operations and gated aggregations. Its empirical performance is competitive or superior compared to other convolutional or recurrent alternatives across a range of NLP tasks.

- **Theoretically justifying the component** We provide theoretical justifications that support the empirical success of the proposed architecture. Specifically, we relate this component to string kernels, i.e., functions measuring the similarity between sequences, and demonstrate how it encodes the efficient kernel computing algorithm into its structure.

- **Rationalizing neural predictions** We present a prototype framework that learns to provide supporting evidence (as rationales) for the model's prediction. The framework is modular and flexible for the choice of neural networks and the target applications. The model can be trained in unsupervised fashion, in the sense that no additional rationale annotations are required. We demonstrate both quantitatively and qualitatively the feasibility and effectiveness of the framework in various NLP applications.

We believe our work would benefit neural network-based applications and future research. The idea of incorporating specific algorithms and computations (e.g. kernel and dynamic programming) into the NN structure would help develop new components with known interpretation, therefore provide guidance of its applicability. The rationale framework would help researchers and even ordinary users to better understand, verify and communicate with advanced neural models.

1.4 Outline

The rest of this thesis is organized as follows:

- **Chapter 2** describes our novel neural component for sequence modeling, and presents theoretical and empirical studies of this neural component.

- **Chapter 3** presents our rationale generation framework and the associated training algorithm for learning the supporting evidence of a model's prediction.

- **Chapter 4** concludes the thesis, and discusses a few directions of future work.

We include algorithm derivations, theorem proofs and experimental details in the appendices.

2 From Kernels to Neural Models

In this chapter, we present a class of neural operation designed for text and sequential data in general. After briefly introducing traditional convolution operation, we tailor this operator for sequential data. We then provide theoretical interpretation of the new component by relating it to sequence kernel. Finally we demonstrate the empirical performance of the component in various NLP applications.

2.1 Introduction

The success of deep learning (and neural networks) often derives from well-chosen neural components as the building blocks. In applications where the data is sequential, the most fundamental blocks (and the associated operations) are perhaps feature extraction and aggregation. Considering CNNs as the example, the two operations are performed via temporal convolutions, which typically consist of feature mapping (*filters*) and *poolings* such as max-pooling and average-pooling. Similarly, RNNs utilize *neural gates* as the building block for feature aggregation, which adaptively learn to store or discard features. Indeed, many well-adopted RNN variants such as LSTMs and GRUs employ gatings in their internal architecture.

Despite the great empirical success in text and speech applications, there is little (theoretical) understanding of such sequential models and components. Most of the prior work has focused on intuitive explanation and empirical evaluation of the model, while theoretical interpretations and justifications are rarely provided. In contrast, feed-forward neural nets have been successfully analyzed [80, 35, 53, 83]. For instance, the expressive power and classification margin of deep rectifier networks have been studied [1]. More recently, Zhang et al. [120] showed that feed-forward nets with certain activations belong to the space (reproducing kernel Hilbert space) of kernel functions. Not surprisingly, such discovery leads to the development of new promising neural components or models [2, 119].

We aim to fill the gap for neural sequential models by introducing an operational component called *recurrent convolution* that can be both empirically and theoretically justified. Derived from traditional convolutions, the component is tailored for text and other sequential data, coupling feature mapping and sequential aggregation together. Specifically, the feature mapping is applied over all n-grams in the sequence, including those that are not contiguous. The intuition behind this is that many sequential patterns are non-consecutive expressions, such as semantically close multi-word phrases (e.g. "not that good", "not nearly as good"). Feature aggregation is performed (by summing) over all possible n-grams with a exponentially decaying weight depending on the length of the n-gram span. The decaying factor can be either constant (as in string kernel) or adaptive controlled by a neural gate (as in RNNs). Owing to a dynamic programming-style implementation, this aggregation can run efficiently in a recurrent manner, linear to the length of the sequence.

We relate recurrent convolution to kernel methods (in particular, string kernels [71]), therefore providing a theoretical basis for its empirical effectiveness. We show that the predictor constructed by one or multiple layers of recurrent convolutions (with constant decay) lie in the reproducing kernel Hilbert space (RKHS) of string kernel (and/or its composition with other kernel functions when non-linear activation is involved). Training a neural model of this kind hence can be viewed as seeking a good predictor in the space introduced by a kernel, while posing additional dimension-reduction constraint (due to fixed hidden dimension and number of parameters). The results follow and reinforce recent work that demonstrate the connection between feed-forward networks and kernels [35, 120].

We empirically demonstrate the effectiveness of our neural component in several

NLP applications. In the first application, we train neural language models by applying recurrent convolution similarly to other RNN models such as LSTMs. On the Penn Tree Bank (PTB) dataset, our model variants achieve 69~76 perplexities, being competitive or better compared to state-of-the-art neural language models. In the seond application, we evaluate our model on text classification tasks. On the Stanford Sentiment Treebank dataset, our model obtains the best performance among a variety of neural networks. For example, our best model achieves 53.2% accuracy on fine-grained classification and 89.9% on binary classification, outperforming the deep recursive model [103] and convolutional model [55]. Finally, in the third application, we compare our model with other alternatives on text retrieval task. In particular, all neural models are used as encoders to assess the similarity between text pieces (after being encoded as vectors using the neural nets). The models are compared on the testbed constructed from AskUbuntu question-answering forum. Our full model achieves a MRR of 75.6% and P@1 of 62.0%, yielding 8% absolute improvement over a standard IR baseline, and 4% over standard neural network architectures (including CNNs, LSTMs and GRUs).

The remainder of this chapter is organized as follows. We first describe related prior work on deep learning in Section 2.2. In Section 2.3 we detail the proposed operational component with background, intuitive explanation and formula derivation. Section 2.4 further establishes the theoretical interpretation of the component. We describe applications and accompanied evaluations in Section 2.5.

2.2 Related Work

2.2.1 Neural Networks for NLP

Deep neural networks have recently brought about significant advancements in natural language processing, such as language modeling [8, 76], sentiment analysis [97, 45, 60], syntactic parsing [18, 94, 10] and machine translation [4, 24, 102] among many others. Models applied in these tasks exhibit significant architectural differences. Broadly speaking, the neural architectures range from convolutional neural nets (CNNs) [18, 19, 114, 93, 118], recurrent neural networks (RNNs) [76, 49] and also recursive networks [85, 58]. Of course, the distinctions are not clear-cut since various architectures can be integrated into a single model and combined with other operational blocks such as neural attention. Nevertheless, we summarize various prior work based on this division in the following paragraphs.

Convolutional neural nets Our work most closely relates to convolutional networks. Originally been developed for computer vision [62], CNNs make use of convolution operations (filters) with pooling and layer-stacking to learn features at different levels of abstraction. To adopt CNNs in NLP applications, a number of modifications have been explored. For instance, Collobert et al. [19] use the max-over-time pooling operation to aggregate the features over the input sequence. This variant has been successfully applied to semantic parsing [114] and information retrieval [93, 31]. Kalchbrenner et al. [50] instead propose (dynamic) k-max pooling operation for modeling sentences. In addition, Kim [55] combines CNNs of different filter widths and either static or fine-tuned word vectors. In contrast to the traditional CNN models, the recurrent convolution component we proposed considers non-consecutive n-grams thereby expanding the representation capacity of the model. We also consider nonlinear interactions within n-gram snippets through the use of tensors, moving beyond direct linear projection operator used in standard CNNs. As our experiments demonstrate these advancements result in improved performance.

Recurrent neural nets While traditional CNNs decouples feature extraction and aggregation operations (i.e. convolution and pooling) as separate procedures, recurrent neural networks perform both jointly by successively transforming each input token and updating the internal feature representation (i.e. hidden states). To this end, RNN unit is a parameterized function that produces new state values given the old ones and the current input. In early work, the choice of RNN function is simply a linear transformation followed by element-wise activation, such as in RNN-based neural language model [76]. However, the simple unit often fail to produce competitive performance. More complicated and advanced architectures have therefore been adopted recently, including long short-term memory unit (LSTM) [42] and gated recurrent unit (GRU) [14, 16] for example. LSTMs and GRUs have become successful sequence modeling components in a wide range of NLP applications, such as classification [9, 21, 89], tagging [44], language generation and translation [4, 91]. Our sequence component relates to standard recurrent units in twofold. First, our recurrent convolution component (RCNN) performs feature aggregation similarly in the sense that it maintains feature representations as hidden state values and updates them after each input token. Second, the decaying factor for feature aggregation can be controlled by neural gates, similar to LSTMs and GRUs.

Recursive neural nets Sometimes it is necessary to learn distributed representations of structures, such as syntax trees and logic forms. Recursive neural networks are introduced to explicitly model structures. In recursive nets, the representations of nodes (e.g. tokens in the tree) are transformed into the representation of the parent, following the topological order (e.g. bottom-up order in the tree). Example applications of recursive neural networks includes syntactic parsing [94, 96], sentiment analysis [97, 38, 103], image retrieval [98] and question answering [44]. Li et al. [69] compare recurrent models and recursive models in various applications in order to better understand in which cases the latter is more applicable. Our proposed component is motivated by sequence-based techniques such as string kernel, and hence is more similar to recurrent architecture. However, we note that the idea can be naturally extended beyond sequential structures, for instance trees, by incorporating tree kernel into the NN structure.

2.2.2 Understanding Neural Networks

The empirical success of deep learning has motivated a great deal of research to better understand neural networks. The focus of this research direction is mostly twofold. First, parameterized functions represented as neural networks are neither convex nor concave. Optimizing such functions, in general, is believed to be computationally quite challenging. Nevertheless, training neural networks with randomly initialized parameters seems stable and delivers superior performance in practice. As a result, many recent work has been dedicated to better explain the phenomena. Secondly, a lot of empirical results have suggested that neural architectures, especially deeper networks, possess much more expressive power compared to traditional (perhaps simpler) methods. Hence, to justify the empirical success, the expressive capability of neural networks has been separately addressed, despite their own optimization challenge. We discuss some related work of the two categories in subsequent paragraphs.

Difficulty of Learning Gradient-based methods [26, 57, 117, 113] has been the dominant method of choice for optimizing neural networks. While these methods are known to converge to local optima, they turned to perform surprisingly well in practice. Recent work attempts to provide answers for such observations by analyzing the loss surface of deep neural networks [22, 15]. Based on results from random matrix theory and empirical evidence, they conjectured that the major difficulty of the optimization originates from saddle points (critical points) rather than local minima. That is, local minima are close enough to the global minimum, and the optimization method has to escape saddle points in order to reach those minima. In fact, recent work has proved this conjecture under simplified conditions. For example, Kawaguchi [53] showed that any local optimum is global optimum in deep linear networks. Hardt and Ma [34] proved that there are no saddle points for deep residual linear networks under similar circumstances.

Another practical challenge of optimizing neural models is gradient vanishing, especially for recurrent neural networks [82]. During backward propagation of the training error [90], the scale of the gradient tends to decrease (and vanish) after each layer. Hence, a sequence model can fail to capture long-term dependencies, since training signals are not even propagated for a long range at the first place. To this end, neural gates have been adopted and shown to alleviate this issue, such as in LSTMs and GRUs. With an identity connection (scaled by the neural gate) to the previous states, gradient weights are passed more directly during back-propagation. In our proposed recurrent convolution component, the state values (i.e. feature representations) are aggregated and updated through the decaying factor, which is architecturally similar to those neural gates. In this sense, learning of our component is no harder than that of other alternative structures.

Expressive Power of Neural Nets In addition to the learning problem, a lot of work has focused on understanding the expressive capability of neural networks. For example, the decision boundary (e.g. classification regions) of deep feed-forward networks can grow exponentially with the number of the layers [80, 83]. As a conjecture, functions that can be represented by deeper networks are not easily represented by shallow models [23]. For instance, Cohen et al. [17] proved this result for a specific CNN variant (SimNets) using matrix algebra and measure theory. Another line of research demonstrates the connection between neural nets and kernels [35, 120]. For instance, Zhang et al. [120] showed that predictor functions represented as deep feed-forward networks belong to the reproducing kernel Hilbert space (RKHS) of recursive kernels, under practical assumptions.

Theoretical understanding of existing models often lead to the development of new (neural) methods [1, 2, 119]. Our work in this sense is motivated by kernel methods, and share similar spirit with the recent progress on neural networks [104, 120]. For instance, Tamar et al. [104] encodes value iteration algorithm into the proposed neural component, while our component integrates the dynamic programming implementation of string kernels, therefore expanding the computational and expressive power of traditional convolutional networks. In addition, following Zhang et al. [120], we also show that our component indeed learns a function from the RKHS of string kernels, and/or the RKHS of generalized (resursive) string kernels when a deeper architecture is employed.

2.3 Model

In this section, we present our recurrent convolution component (RCNN) for sequence modeling. We begin by briefly describing traditional convolution operation in the context of text processing, and then introduce recurrent convolution to encode non-consecutive n-gram features. Thereafter, we proceed to derive an efficient way to compute feature mappings via dynamic programming.

2.3.1 Background

Let $\mathbf{x}_1, \cdots, \mathbf{x}_L \in \mathbb{R}^d$ be the input sequence such as a document or sentence. Here L is the length of the sequence and each $\mathbf{x}_i$ is a vector representing the i^{th} word. The (consecutive) *n-gram* vector ending at position j is obtained by simply concatenating the corresponding word vectors i.e., $\mathbf{v}_j := [\mathbf{x}_{j-n+1}; \mathbf{x}_{j-n+2}; \cdots; \mathbf{x}_j]$. Note here, out-of-index words are simply set to all zeros.

The traditional convolution operator is parameterized by filter matrix $\mathbf{W} \in \mathbb{R}^{h \times nd}$ which can be thought of as n smaller filter matrices, i.e. $\mathbf{W}^{(i)} \in \mathbb{R}^{h \times d}, i \in \{1, .., n\}$. These smaller matrices are applied to each $\mathbf{x}_i$ in vector $\mathbf{v}_j$ to obtain a (linearly) transformed representation $\mathbf{W}\mathbf{v}_j \in \mathbb{R}^h$:

$$\mathbf{W}\mathbf{v}_j := \sum_{i=1}^{n} \mathbf{W}^{(i)} \mathbf{x}_{j-n+i}$$

For the entire sequence level, the operator maps each n-gram vector $\mathbf{v}_j$ in the input sequence to $\mathbf{W}\mathbf{v}_j \in \mathbb{R}^h$ so that the input sequence $\mathbf{x}$ is transformed into a sequence of feature representations,

$$[\mathbf{W}\mathbf{v}_1, \cdots, \mathbf{W}\mathbf{v}_L] \quad \in \quad \mathbb{R}^{L \times h}$$

The resulting feature values are often passed through non-linearities such as the hyper-tangent (element-wise) as well as aggregated or reduced by "sum-over" or "max-pooling" operations for later (similar stages) of processing.

The overall architecture can be easily modified by replacing the basic n-gram vectors and the convolution operation with other feature mappings. Indeed, we introduce two possible modifications which we believe can better capture feature interaction within a sequence. These two modifications are detailed in the next two sub-sections.

2.3.2 Non-consecutive N-gram Features

Traditional convolution uses consecutive n-grams in the feature map. Non-consecutive n-grams may nevertheless be helpful since phrases such as "not good", "not so good" and "not nearly as good" express similar sentiments but involve variable spacings between the key words. Variable spacings are not effectively captured by fixed n-grams.

We apply the feature-mapping in a weighted manner to all n-grams thereby gaining access to patterns such as "not ... good". For example, let $\mathbf{z}[i,j,k] \in \mathbb{R}^h$ denote the feature representation corresponding to a 3-gram $(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k)$ of words in positions i, j, and k along the sequence. This vector is calculated as,

$$\mathbf{z}[i,j,k] \;=\; \mathbf{W}^{(1)}\mathbf{x}_i + \mathbf{W}^{(2)}\mathbf{x}_j + \mathbf{W}^{(3)}\mathbf{x}_k$$

We will aggregate these vectors into an $h-$dimensional feature representation at each position in the sequence. The idea is similar to neural bag-of-words models where the feature representation for a document or sentence is obtained by averaging (or summing) of all the word vectors. In our case, we define the aggregate representation $\mathbf{z}_3[t]$ in position t as the weighted sum of all 3-gram feature representations ***ending exactly at position*** t, i.e.,

$$\begin{aligned}\mathbf{z}_3[t] &= \sum_{i<j<k=t} \mathbf{z}[i,j,k] \cdot \lambda^{(k-j-1)+(j-i-1)} \\ &= \sum_{i<j<k=t} \mathbf{z}[i,j,k] \cdot \lambda^{k-i-2} \qquad (2.1)\end{aligned}$$

where $\lambda \in [0,1)$ is a decay factor that down-weights 3-grams with longer spans (i.e., 3-grams that skip more in-between words). The way of feature aggregation is not unique. For instance, we can alternatively sum over 3-gram features ***up to position*** t, including those in previous positions. We denote this aggregate representation as $\mathbf{c}_3[t]$, and compute it similarly with the decay factor λ,

$$\begin{aligned} \mathbf{c}_3[k] &= \sum_{i<j<k\leq t} \mathbf{z}[i,j,k] \cdot \lambda^{(t-k)+(k-j-1)+(j-i-1)} \\ &= \sum_{i<j<k\leq} \mathbf{z}[i,j,k] \cdot \lambda^{t-i-2} \end{aligned} \tag{2.2}$$

where an extra term $\lambda^{(t-k)}$ is introduced to penalize 3-grams which end before the target position t (hence skipping words after).

Notice that, as $\lambda \to 0$ all non-consecutive 3-grams are omitted, $\mathbf{c}_3[t] = \mathbf{z}_3[t] = \mathbf{z}[t-2, t-1, t]$, and the model acts like a traditional convolution with only consecutive n-grams. When $\lambda > 0$, however, $\mathbf{c}_3[t]$ and $\mathbf{z}_3[t]$ are weighted averages of many 3-grams with variable spans.

2.3.3 Non-linear Feature Mapping

Typical $n-$gram feature mappings (Sec 2.3.1) where concatenated word vectors are mapped linearly to feature coordinates may be insufficient to directly capture relevant information in the $n-$gram. As a remedy, we replace concatenation with a tensor product. Consider again a 3-gram $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ and the corresponding tensor product $\mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3$. The result of the tensor product is a 3-way array of coordinate interactions such that each ijk entry of the tensor is given by the product of the corresponding coordinates of the word vectors

$$(\mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3)_{ijk} = x_{1i} \cdot x_{2j} \cdot x_{3k}$$

Here $\otimes$ denotes the tensor product operator. The tensor product of a 2-gram analogously gives a two-way array or matrix $\mathbf{x}_1 \otimes \mathbf{x}_2 \in \mathbb{R}^{d \times d}$. The n-gram tensor can be seen as a direct generalization of the typical concatenated vector[1].

Tensor-based non-linear mapping Since each n-gram in the sequence is now expanded into a high-dimensional tensor using tensor products, the set of filters are analogously maintained as high-order tensors. In other words, our filters are linear mappings over the higher dimensional interaction terms rather than the original word coordinates.

Consider mapping the 3-gram $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ into a feature representation. Each filter is a 3-way tensor with dimensions $d \times d \times d$. The set of h filters, denoted as T, is a 4-way tensor of dimension $d \times d \times d \times h$, where each d^3 slice of T represents a single filter and h is the number of such filters, i.e., the feature dimension. The resulting $h-$dimensional feature representation $\mathbf{z} \in \mathbb{R}^h$ for the 3-gram $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ is obtained by multiplying the filter $\mathbf{T}$ and the 3-gram tensor as follows. The l^{th} coordinate of $\mathbf{z}$ is given by

$$\begin{aligned} z_l &= \sum_{ijk} \mathbf{T}_{ijkl} \cdot (\mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \mathbf{x}_3)_{ijk} \\ &= \sum_{ijk} \mathbf{T}_{ijkl} \cdot x_{1i} \cdot x_{2j} \cdot x_{3k} \end{aligned} \tag{2.3}$$

The formula is equivalent to summing over all the third-order polynomial interaction terms where tensor $\mathbf{T}$ stores the coefficients.

Directly maintaining the filters as full tensors leads to parametric explosion. Indeed, the size of the tensor $\mathbf{T}$ (i.e. $d^n \times h$) would be too large even for typical low-dimensional word vectors where, e.g., $d = 300$. To this end, we assume a *low-rank factorization* of the tensor $\mathbf{T}$, represented in the Kruskal form. Specifically, $\mathbf{T}$ is decomposed into a sum of h rank-1 tensors

$$\mathbf{T} = \sum_{r=1}^{h} \mathbf{W}_r^{(1)} \otimes \mathbf{W}_r^{(2)} \otimes \mathbf{W}_r^{(3)} \otimes \mathbf{O}_r$$

[1] To see this, consider word vectors with a "bias" term $\mathbf{x}_i{}' = [\mathbf{x}_i; 1]$. The tensor product of n such vectors includes the concatenated vector as a subset of tensor entries but, in addition, contains all up to n^{th}-order interaction terms.

where $\mathbf{W}^{(i)} \in \mathbb{R}^{h \times d}, i = 1, .., 3$ are smaller parameter matrices similar to those used in a linear filter (shown later), and $\mathbf{O} \in \mathbb{R}^{h \times h}$ is an output matrix (with respect to the last mode of our tensor $\mathbf{T}$). $\mathbf{W}_r^{(i)}$ (and $\mathbf{O}_r$) denotes the r^{th} row of the matrix. Note that, for simplicity, we have assumed that the number of rank-1 components in the decomposition is equal to the feature dimension h. We could optionally set $\mathbf{O} = \mathbf{I}$ (i.e. the identity matrix) for simplicity as well. These assumptions would not affect subsequent algorithm derivations or analyses. Plugging the low-rank factorization into Eq.(2.3), the feature-mapping can be rewritten in a vector form as

$$\begin{aligned} \mathbf{z} &= \mathbf{O}^\top \left(\mathbf{W}^{(1)}\mathbf{x}_1 \odot \mathbf{W}^{(2)}\mathbf{x}_2 \odot \mathbf{W}^{(3)}\mathbf{x}_3 \right) && (2.4) \\ &= \mathbf{W}^{(1)}\mathbf{x}_1 \odot \mathbf{W}^{(2)}\mathbf{x}_2 \odot \mathbf{W}^{(3)}\mathbf{x}_3 && \text{(when } \mathbf{O} \text{ set to } \mathbf{I}\text{)} \end{aligned}$$

where $\odot$ is the element-wise product such that, e.g., $(\mathbf{a} \odot \mathbf{b})_k = a_k b_k$ for $\mathbf{a}, \mathbf{b} \in \mathbb{R}^m$. Note that while $\mathbf{W}^{(1)}\mathbf{x}_1$ (similarly $\mathbf{W}^{(2)}\mathbf{x}_2$ and $\mathbf{W}^{(3)}\mathbf{x}_3$) is a linear mapping from each word $\mathbf{x}_1$ (similarly $\mathbf{x}_2$ and $\mathbf{x}_3$) into a h-dimensional feature space, higher-order terms and non-linearity arise from the element-wise products.

Table 2.1 summarizes the proposed operations (modifications) for recurrent convolutions. Those equations are presented in the case of 3-grams for ease of reading, but can be extended to in general to any case of n-grams. We also present the normalized version of feature aggregation, in which the factors $Z_3[t]$ and $C_3[t]$ are introduced so the weights of $\mathbf{z}[i, j, k]$ terms sum to one.

2.3.4 Recurrent Computation via Dynamic Programming

Directly calculating $\mathbf{z}_3[\cdot]$ and $\mathbf{c}_3[\cdot]$ shown in Table 2.1 by enumerating all 3-grams would require $O(L^2)$ and $O(L^3)$ feature-mapping operations respectively. In general, directly enumerating all n-grams would require $O(L^n)$ operations. We can, however, evaluate the features more efficiently by relying on the associative and distributive properties of the feature operation.

The efficient way of computing $\mathbf{z}_3[\cdot]$ and $\mathbf{c}_3[\cdot]$ can be derived and implemented via dynamic programming (i.e. breaking the problem into a series of smaller sub-

Feature mapping:

$$\begin{aligned}
\mathbf{z}[i,j,k] &= \mathbf{W}^{(1)}\mathbf{x}_i + \mathbf{W}^{(2)}\mathbf{x}_j + \mathbf{W}^{(3)}\mathbf{x}_k && \text{(additive)} \\
\mathbf{z}[i,j,k] &= \mathbf{W}^{(1)}\mathbf{x}_i \odot \mathbf{W}^{(2)}\mathbf{x}_j \odot \mathbf{W}^{(3)}\mathbf{x}_k && \text{(multiplicative)} \\
\mathbf{z}[i,j,k] &= \mathbf{O}^\top \left(\mathbf{W}^{(1)}\mathbf{x}_i \odot \mathbf{W}^{(2)}\mathbf{x}_j \odot \mathbf{W}^{(3)}\mathbf{x}_k\right) && \text{(multiplicative, with } \mathbf{O}\text{)}
\end{aligned}$$

Aggregation:

$$\begin{aligned}
\mathbf{z}_3[t] &= \sum_{i<j<t} \mathbf{z}[i,j,t] \cdot \lambda^{t-i-2} && \text{(exactly at position t)} \\
\mathbf{c}_3[t] &= \sum_{i<j<k\leq t} \mathbf{z}[i,j,k] \cdot \lambda^{t-i-2} && \text{(up to position t)}
\end{aligned}$$

Normalized Aggregation:

$$\begin{aligned}
\mathbf{z}_3[t] &= \sum_{i<j<t} \mathbf{z}[i,j,t] \cdot \frac{\lambda^{t-i-2}}{Z_3[t]} && \text{(exactly at position t)} \\
Z_3[t] &= \sum_{i<j<t} \lambda^{t-i-2} && \\
\mathbf{c}_3[t] &= \sum_{i<j<k\leq t} \mathbf{z}[i,j,k] \cdot \frac{\lambda^{t-i-2}}{C_3[t]} && \text{(up to position t)} \\
C_3[t] &= \sum_{i<j<k\leq t} \lambda^{t-i-2} &&
\end{aligned}$$

Table 2.1: A summarization of proposed operations in RCNN for 3-grams. Compared to the traditional convolution, the feature mapping operation can be multiplicative (and hence non-linear) to model interactions between $(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k)$, and feature aggregation is performed over all n-gram combinations. $Z_3[t]$ and $C_3[t]$ are normalization factors that eliminate the bias introduced by sequence length.

problems). Specifically, consider computing $\mathbf{z}_3[t]$ and $\mathbf{c}_3[t]$ for the multiplicative mapping (Table 2.1, second equation). Plugging the feature mapping equation into the aggregation equation, we obtain,

$$\begin{aligned}
\mathbf{z}_3[t] &= \sum_{i<j<t} \lambda^{t-i-2} \cdot \left(\mathbf{W}^{(1)}\mathbf{x}_i \odot \mathbf{W}^{(2)}\mathbf{x}_j \odot \mathbf{W}^{(3)}\mathbf{x}_t\right) \\
\mathbf{c}_3[t] &= \sum_{i<j<k\leq t} \lambda^{t-i-2} \cdot \left(\mathbf{W}^{(1)}\mathbf{x}_i \odot \mathbf{W}^{(2)}\mathbf{x}_j \odot \mathbf{W}^{(3)}\mathbf{x}_k\right)
\end{aligned}$$

As auxiliary dynamic programming tables (sub-problems), we can analogously define $\mathbf{z}_1[t]$ and $\mathbf{z}_2[t]$ (similarly $\mathbf{c}_1[t]$ and $\mathbf{c}_2[t]$) for 1-grams and 2-grams respectively,

$$\begin{aligned}
\mathbf{z}_1[t] &= \mathbf{W}^{(1)}\mathbf{x}_t \\
\mathbf{c}_1[t] &= \sum_{i\leq t} \lambda^{t-i} \cdot \mathbf{W}^{(1)}\mathbf{x}_i \\
\mathbf{z}_2[t] &= \sum_{i<t} \lambda^{t-i-1} \cdot \left(\mathbf{W}^{(1)}\mathbf{x}_i \odot \mathbf{W}^{(2)}\mathbf{x}_t\right) \\
\mathbf{c}_2[t] &= \sum_{i<j\leq t} \lambda^{t-i-1} \cdot \left(\mathbf{W}^{(1)}\mathbf{x}_i \odot \mathbf{W}^{(2)}\mathbf{x}_j\right)
\end{aligned}$$

These dynamic programming tables can be computed in a ***recurrent manner***, according to the following formulas:

$$\begin{aligned}
\mathbf{z}_1[t] &= \mathbf{W}^{(1)}\mathbf{x}_t \\
\mathbf{c}_1[t] &= \lambda \cdot \mathbf{c}_1[t-1] + \mathbf{z}_1[t] \\
\mathbf{z}_2[t] &= \mathbf{c}_1[t-1] \odot \mathbf{W}^{(2)}\mathbf{x}_t \\
\mathbf{c}_2[t] &= \lambda \cdot \mathbf{c}_2[t-1] + \mathbf{z}_2[t] \\
\mathbf{z}_3[t] &= \mathbf{c}_2[t-1] \odot \mathbf{W}^{(3)}\mathbf{x}_t \\
\mathbf{c}_3[t] &= \lambda \cdot \mathbf{c}_3[t-1] + \mathbf{z}_3[t]
\end{aligned}$$

The derivation of this DP algorithm for any n-grams is presented in Appendix A.1.

Note for $\mathbf{c}_i[t]$ terms, we can substitute $\mathbf{z}_i[t]$ with the corresponding right hand side term to remove the dependence on $\mathbf{z}_i[t]$. That is,

$$\begin{aligned}
\mathbf{c}_1[t] &= \lambda \cdot \mathbf{c}_1[t-1] + \mathbf{W}^{(1)}\mathbf{x}_t \\
\mathbf{c}_2[t] &= \lambda \cdot \mathbf{c}_2[t-1] + \left(\mathbf{c}_1[t-1] \odot \mathbf{W}^{(2)}\mathbf{x}_t\right) \\
\mathbf{c}_3[t] &= \lambda \cdot \mathbf{c}_3[t-1] + \left(\mathbf{c}_2[t-1] \odot \mathbf{W}^{(3)}\mathbf{x}_t\right)
\end{aligned}$$

As we can see, the above formulas are structurally similar to those recurrent formulas used in RNNs. As a final refinement, we will apply element-wise activations after each representation, similar to standard CNNs and RNNs, i.e. $\mathbf{h}[t] = g\left(\mathbf{c}_3[t]\right)^2$ for a selected activation function $g()$ such as tanh() and ReLU().

The recurrent computation for other choices of feature mapping and aggregation listed in Table 2.1 can be derived similarly, and only differs from the above version very slightly. We present various implementations in Table 2.2. Overall, the n-gram feature aggregation can be performed in $O(Ln)$ matrix multiplication/addition operations, and remains linear in the sequence length.

2.3.5 Extensions

We discuss a few extensions of RCNNs in this section. One is a gated version which uses neural gates to learn *adaptive decay weights* from the context. Another one uses max-pooling instead of weighted average-pooling for feature aggregation.

Adaptive Gated Decay We refine RCNNs by learning context dependent decay weights. For example, if the current input word provides no relevant information (e.g., symbols, functional words), the model should ignore it by incorporating the token with a vanishing weight. In contrast, important content words (such as sentiment words "good" and "bad" for sentiment analysis) should be included with much larger weights. To achieve this effect we introduce *neural gates* similar to LSTMs to specify

[2] We can alternatively use $\mathbf{z}[t]$ instead of $\mathbf{c}[t]$ as the input to the activation function, when we are only interested in n-gram features exactly ends at position t.

Recurrent computation of RCNN:

(a) Multiplicative mapping, aggregation un-normalized:

$$\begin{aligned}
\mathbf{c}_1[t] &= \lambda \cdot \mathbf{c}_1[t-1] + \mathbf{W}^{(1)}\mathbf{x}_t & \mathbf{z}_1[t] &= \mathbf{W}^{(1)}\mathbf{x}_t \\
\mathbf{c}_2[t] &= \lambda \cdot \mathbf{c}_2[t-1] + \left(\mathbf{c}_1[t-1] \odot \mathbf{W}^{(2)}\mathbf{x}_t\right) & \mathbf{z}_2[t] &= \mathbf{c}_1[t-1] \odot \mathbf{W}^{(2)}\mathbf{x}_t \\
\mathbf{c}_3[t] &= \lambda \cdot \mathbf{c}_3[t-1] + \left(\mathbf{c}_2[t-1] \odot \mathbf{W}^{(3)}\mathbf{x}_t\right) & \mathbf{z}_3[t] &= \mathbf{c}_2[t-1] \odot \mathbf{W}^{(3)}\mathbf{x}_t
\end{aligned}$$

(b) Multiplicative mapping, aggregation normalized:

$$\begin{aligned}
\mathbf{c}_1[t] &= \lambda \cdot \mathbf{c}_1[t-1] + (1-\lambda) \cdot \mathbf{W}^{(1)}\mathbf{x}_t & \mathbf{z}_1[t] &= \mathbf{W}^{(1)}\mathbf{x}_t \\
\mathbf{c}_2[t] &= \lambda \cdot \mathbf{c}_2[t-1] + (1-\lambda) \cdot \left(\mathbf{c}_1[t-1] \odot \mathbf{W}^{(2)}\mathbf{x}_t\right) & \mathbf{z}_2[t] &= \mathbf{c}_1[t-1] \odot \mathbf{W}^{(2)}\mathbf{x}_t \\
\mathbf{c}_3[t] &= \lambda \cdot \mathbf{c}_3[t-1] + (1-\lambda) \cdot \left(\mathbf{c}_2[t-1] \odot \mathbf{W}^{(3)}\mathbf{x}_t\right) & \mathbf{z}_3[t] &= \mathbf{c}_2[t-1] \odot \mathbf{W}^{(3)}\mathbf{x}_t
\end{aligned}$$

(c) Additive mapping, aggregation normalized:

$$\begin{aligned}
\mathbf{c}_1[t] &= \lambda \cdot \mathbf{c}_1[t-1] + (1-\lambda) \cdot \mathbf{W}^{(1)}\mathbf{x}_t & \mathbf{z}_1[t] &= \mathbf{W}^{(1)}\mathbf{x}_t \\
\mathbf{c}_2[t] &= \lambda \cdot \mathbf{c}_2[t-1] + (1-\lambda) \cdot \left(\mathbf{c}_1[t-1] + \mathbf{W}^{(2)}\mathbf{x}_t\right) & \mathbf{z}_2[t] &= \mathbf{c}_1[t-1] + \mathbf{W}^{(2)}\mathbf{x}_t \\
\mathbf{c}_3[t] &= \lambda \cdot \mathbf{c}_3[t-1] + (1-\lambda) \cdot \left(\mathbf{c}_2[t-1] + \mathbf{W}^{(3)}\mathbf{x}_t\right) & \mathbf{z}_3[t] &= \mathbf{c}_2[t-1] + \mathbf{W}^{(3)}\mathbf{x}_t
\end{aligned}$$

Final activation:

$$\begin{aligned}
\mathbf{h}[t] &= g\left(\mathbf{c}_3[t]\right) \quad \text{or} \quad \mathbf{h}[t] = g\left(\mathbf{z}_3[t]\right) \\
\mathbf{h}[t] &= g\left(\mathbf{O}^\top \mathbf{c}_3[t]\right) \quad \text{or} \quad \mathbf{h}[t] = g\left(\mathbf{O}^\top \mathbf{z}_3[t]\right) \qquad \text{(with output matrix } \mathbf{O}\text{)}
\end{aligned}$$

Table 2.2: An summarization of RCNN variants and associated equations. Again we present these equations in the context of 3-grams, but they can be easily generalized to any n-gram cases.

when and how to average the observed signals.

Consider the additive and normalized RCNN in Table 2.2 (c) as the example, the resulting adaptive version is,

$$\begin{aligned}
\mathbf{c}_1[t] &= \lambda_t \odot \mathbf{c}_1[t-1] \;+\; (1-\lambda_t) \odot \left(\mathbf{W}^{(1)}\mathbf{x}_t\right) \\
\mathbf{c}_2[t] &= \lambda_t \odot \mathbf{c}_2[t-1] \;+\; (1-\lambda_t) \odot \left(\mathbf{c}_1[t-1] + \mathbf{W}^{(2)}\mathbf{x}_t\right) \\
\mathbf{c}_3[t] &= \lambda_t \odot \mathbf{c}_3[t-1] \;+\; (1-\lambda_t) \odot \left(\mathbf{c}_2[t-1] + \mathbf{W}^{(3)}\mathbf{x}_t\right) \\
\lambda_t &= \sigma(\mathbf{W}^{\lambda}\mathbf{x}_t + \mathbf{U}^{\lambda}\mathbf{h}[t-1] + \mathbf{b}^{\lambda})
\end{aligned}$$

where $\sigma(\cdot)$ is the sigmoid function and $\mathbf{h}[t-1]$ is the representation at the previous position $t-1$ after applying the non-linear activation (see Table 2.2). $\mathbf{W}^{\lambda}$, $\mathbf{U}^{\lambda}$ and $\mathbf{b}^{\lambda}$ are additional parameters associated with the neural gate. The difference between the adaptive version and the original version is the choice of λ at each input. The decay value λ_t is controlled and parametrized by a neural gate and responds directly to the previous state $\mathbf{h}[t-1]$ and the input token.

The adaptive version can still be seen as a generalization of traditional CNNs by incorporating similar mechanism from recent recurrent structures. When the gate $\lambda_t = 0$ (vector) for all t, the model computes traditional convolution operations, i.e. $\mathbf{c}_3[t] = \mathbf{W}^{(1)}\mathbf{x}_{t-2} + \mathbf{W}^{(2)}\mathbf{x}_{t-1} + \mathbf{W}^{(3)}\mathbf{x}_t$. As $\lambda_t > 0$, however, $\mathbf{c}_n[t]$ becomes the sum of an exponential number of terms, enumerating all possible n-grams within $\mathbf{x}_1, \cdots, \mathbf{x}_t$ (seen by expanding the formulas).

In the evaluation section, we demonstrate that learning context-dependent decay values indeed improve the model's performance.

Max-pooling over Non-consecutive N-grams The RCNN component we presented so far aggregate features in a weighted manner. In traditional CNNs however, it is also quite common to use max-pooling (instead of average-pooling). To this end, the second extension of the component considers max-pooling in the context of non-consecutive n-grams.

Specifically, let $z_3[t][l]$ be the l^{th} entry of vector $\mathbf{z}_3[t]$ (and also $c_3[t][l]$ and $z[i,j,k][l]$

for vectors $\mathbf{c}_3[t]$ and $\mathbf{z}[i, j, k]$ respectively). The max-pooling operation when applied over all n-grams can be defined as

$$\begin{aligned} z_3[t][l] &= \max_{i<j<t} \left\{ z[i, j, t][l] \cdot \lambda^{t-i-2} \right\} \\ &= \max_{i<j<t} \left\{ \left(\mathbf{W}^{(1)}\mathbf{x}_i + \mathbf{W}^{(2)}\mathbf{x}_j + \mathbf{W}^{(3)}\mathbf{x}_t\right)[l] \cdot \lambda^{t-i-2} \right\} && \text{(additive)} \\ \text{or,} \quad &= \max_{i<j<t} \left\{ \left(\mathbf{W}^{(1)}\mathbf{x}_i \odot \mathbf{W}^{(2)}\mathbf{x}_j \odot \mathbf{W}^{(3)}\mathbf{x}_t\right)[l] \cdot \lambda^{t-i-2} \right\} && \text{(multiplicative)} \end{aligned}$$

The operations for $z_n[t][l]$ (and $c_n[t][l]$) for n-grams can be defined analogously. Again, these values can be calculated efficiently using dynamic programming[3]. This variant have been adopted in recent work [81]. We refer readers to Nguyen and Grishman [81] for more discussion and evaluation.

[3] However, it becomes difficult when the output matrix **O** is introduced in multiplicative mapping.

2.4 Theoretical Interpretation

In the previous section, we have presented the recurrent convolution (RCNN) as a sequence modeling component, and discussed about the intuition behind its construction. In this section, we give more theoretical justifications for it – why RCNN would be good sequence model. When the decay factor λ is a constant, we show that RCNN encodes sequence kernels (a.k.a string kernels), i.e. functions measuring sequence similarity, as the central part of its computation. As a result, we show that any classification function built from one or several RCNN layers belongs to the reproducing kernel Hilbert space (RKHS) introduced by sequence kernels. Finally we discuss the generalized case when the decay factor becomes adaptive (controlled by neural gates).

2.4.1 Background

Notations We define a sequence (or a string) of tokens (e.g. a sentence) as $\mathbf{x}_{1:L} = \{\mathbf{x}_i\}_{i=1}^{L}$ where $\mathbf{x}_i \in \mathbb{R}^d$ represents its i^{th} element and $|\mathbf{x}| = L$ denotes the length. Whenever it is clear from the context, we will omit the subscript and directly use $\mathbf{x}$ (and $\mathbf{y}$) to denote a sequence. For a pair of vectors (or matrices) $\mathbf{u}, \mathbf{v}$, we denote $\langle \mathbf{u}, \mathbf{v} \rangle = \sum_k u_k v_k$ as their inner product. For a kernel function $\mathcal{K}_i(\cdot, \cdot)$ with subscript i, we use $\phi_i(\cdot)$ to denote its underlying mapping, i.e. $\mathcal{K}_i(\mathbf{x}, \mathbf{y}) = \langle \phi_i(\mathbf{x}), \phi_i(\mathbf{y}) \rangle = \phi_i(\mathbf{x})^\top \phi_i(\mathbf{y})$.

Sequence Kernels A family of functions called string kernels measures the similarity between two strings (sequences) by counting shared subsequences (see Lodhi et al. [71]). For example, let $\mathbf{x}$ and $\mathbf{y}$ be two strings, a 2-gram string kernel $\mathcal{K}_2(\mathbf{x}, \mathbf{y})$ counts the number of 2-grams $\mathbf{x}_i\mathbf{x}_j$ and $\mathbf{y}_k\mathbf{y}_l$ such that $\mathbf{x}_i\mathbf{x}_j = \mathbf{y}_k\mathbf{y}_l$,

$$\mathcal{K}_2(\mathbf{x}, \mathbf{y}) = \sum_{1 \le i < j \le |\mathbf{x}|} \sum_{1 \le k < l \le |\mathbf{y}|} \lambda^{j-i-1} \lambda^{l-k-1} \left[\mathbb{1}(\mathbf{x}_i = \mathbf{y}_k) \cdot \mathbb{1}(\mathbf{x}_j = \mathbf{y}_l) \right] \qquad (2.5)$$

where $\lambda \in [0, 1)$ is a decay factor penalizing non-contiguous substrings.

The kernel function presented above assign decay weights to substrings regardless of the positions they appear in the sequence. In many temporal predictions such as language modeling, substrings (i.e. patterns) appear later may have higher impact and should be assigned higher weights for prediction, for instance,

$$\mathcal{K}_2(\mathbf{x},\mathbf{y}) = \sum_{1\le i<j\le|\mathbf{x}|}\ \sum_{1\le k<l\le|\mathbf{y}|} \lambda^{|\mathbf{x}|-i-1}\lambda^{|\mathbf{y}|-k-1}\left[\mathbb{1}(\mathbf{x}_i=\mathbf{y}_k)\cdot\mathbb{1}(\mathbf{x}_j=\mathbf{y}_l)\right] \tag{2.6}$$

where the decay weights $\lambda^{|\mathbf{x}|-i-1}$ and $\lambda^{|\mathbf{y}|-k-1}$ are determined based on the distance from the 2-grams to the end. Note the kernel functions can be generalized to n-grams when $n \neq 2$. For simplicity, we use 2-gram ($n = 2$) as the illustrative example for discussion.

In our case, each token in the sequence is a vector (such as one-hot encoding of a word or a feature vector), we shall replace the exact match $\mathbb{1}(\mathbf{u}=\mathbf{v})$ by the inner product $\langle\mathbf{u},\mathbf{v}\rangle$. To this end, the kernel function (2.6) can be rewritten as,

$$\begin{aligned}
\mathcal{K}_2(\mathbf{x},\mathbf{y}) &= \sum_{1\le i<j\le|\mathbf{x}|}\ \sum_{1\le k<l\le|\mathbf{y}|} \lambda^{|\mathbf{x}|-i-1}\lambda^{|\mathbf{y}|-k-1}\left\langle\mathbf{x}_i,\mathbf{y}_k\right\rangle\left\langle\mathbf{x}_j,\mathbf{y}_l\right\rangle \\
&= \sum_{1\le i<j\le|\mathbf{x}|}\ \sum_{1\le k<l\le|\mathbf{y}|} \lambda^{|\mathbf{x}|-i-1}\lambda^{|\mathbf{y}|-k-1}\left\langle\mathbf{x}_i\otimes\mathbf{x}_j,\mathbf{y}_k\otimes\mathbf{y}_l\right\rangle \\
&= \left\langle \sum_{1\le i<j\le|\mathbf{x}|}\lambda^{|\mathbf{x}|-i-1}\mathbf{x}_i\otimes\mathbf{x}_j,\ \sum_{1\le k<l\le|\mathbf{y}|}\lambda^{|\mathbf{y}|-k-1}\mathbf{y}_k\otimes\mathbf{y}_l\right\rangle
\end{aligned} \tag{2.7}$$

where $\mathbf{x}_i\otimes\mathbf{x}_j\in\mathbb{R}^{d\times d}$ (and similarly $\mathbf{y}_k\otimes\mathbf{y}_l$) is the outer-product. The above equality uses the fact that $\langle\mathbf{x}_i,\mathbf{y}_k\rangle\cdot\langle\mathbf{x}_j,\mathbf{y}_l\rangle = \langle\mathbf{x}_i\otimes\mathbf{x}_j,\mathbf{y}_k\otimes\mathbf{y}_l\rangle$. In other words, the underlying mapping of kernel $\mathcal{K}_2()$ defined above is $\phi_2(\mathbf{x}) = \sum_{1\le i<j\le|\mathbf{x}|}\lambda^{|\mathbf{x}|-i-1}\mathbf{x}_i\otimes\mathbf{x}_j$.

The kernel function (2.5) can be derived the same way with the different decay weight. Another variant is to use *partial matching score*, e.g. an additive term $\langle\mathbf{x}_i,\mathbf{y}_k\rangle + \langle\mathbf{x}_j,\mathbf{y}_l\rangle$ instead of the multiplicative version in Eq.(2.7). As we will see later in the section, these variants have a one-to-one correspondence with the RCNN variants presented in the previous section (see examples in Table 2.2).

2.4.2 One-layer RCNN as Kernel Computation

We now proceed to present our theoretical results. Consider an n-gram RCNN with multiplicative and un-normalized features (see Table 2.2 (a)), the internal feature states are computed via the recurrent formulas,

$$\begin{aligned}
\mathbf{c}_1[t] &= \lambda \cdot \mathbf{c}_1[t-1] + \left(\mathbf{W}^{(1)}\mathbf{x}_t\right) \\
\mathbf{c}_2[t] &= \lambda \cdot \mathbf{c}_2[t-1] + \left(\mathbf{c}_1[t-1] \odot \mathbf{W}^{(2)}\mathbf{x}_t\right) \\
&\cdots \\
\mathbf{c}_n[t] &= \lambda \cdot \mathbf{c}_n[t-1] + \left(\mathbf{c}_{n-1}[t-1] \odot \mathbf{W}^{(n)}\mathbf{x}_t\right)
\end{aligned}$$

For $j \in \{1, .., n\}$, let $c_j[t][i]$ be the i-th entry of state vector $\mathbf{c}_j[t]$, $\mathbf{w}_i^{(j)}$ represents the i-th row of filter matrix $\mathbf{W}^{(j)}$. Define $\mathbf{w}_{i,j} = \{\mathbf{w}_i^{(1)}, \mathbf{w}_i^{(2)}, ..., \mathbf{w}_i^{(j)}\}$ as the string constructed by taking the i-th row from each matrix $\mathbf{W}^{(1)}, .., \mathbf{W}^{(j)}$.

Theorem 1. *Let $\mathbf{x}_{1:t}$ be the prefix of $\mathbf{x}$ consisting of first t tokens, and $\mathcal{K}_j$ be the string kernel of j-gram shown in Eq.(2.7). The RCNN state values satisfy,*

$$c_j[t][i] \;=\; \mathcal{K}_j\left(\mathbf{x}_{1:t}, \mathbf{w}_{i,j}\right) = \langle \phi_j(\mathbf{x}_{1:t}), \phi_j(\mathbf{w}_{i,j}) \rangle$$

for any $j \in \{1, .., n\}$, $t \in \{1, .., |\mathbf{x}|\}$.

See Appendix A.2 for the proof of this theorem. In other words, the RCNN layer evaluates the kernel function given a sequence prefix $\mathbf{x}_{1:t}$ and the fuzzy sequence $\mathbf{w}_{i,j}$. This means any linear combination of the state values (a classifier $\theta^\top \mathbf{c}_n[t]$ for example), as a function of the input $\mathbf{x}$, will lie in the function space (i.e. reproducing kernel Hilbert space) introduced by string kernel $\mathcal{K}_1, ..., \mathcal{K}_n$.

Similarly, there are corresponding kernel functions for other variants of RCNN discussed in Section 2.3. Table 2.3 lists the kernels and underlying mappings for RCNNs listed in Table 2.2.

Applying Non-linear Activation In practice, a non-linear activation function such as polynomial activation or sigmoid-like activation is added to the internal states

Underlying string kernel for 2-gram RCNN:

(a) Multiplicative mapping, aggregation un-normalized:

$$\mathcal{K}_2(\mathbf{x},\mathbf{y}) = \sum_{1\le i<j\le|\mathbf{x}|}\ \sum_{1\le k<l\le|\mathbf{y}|} \lambda^{|\mathbf{x}|-i-1}\,\lambda^{|\mathbf{y}|-k-1}\ \langle \mathbf{x}_i,\mathbf{y}_k\rangle\cdot\langle \mathbf{x}_j,\mathbf{y}_l\rangle$$

$$\phi_2(\mathbf{x}) = \sum_{1\le i<j\le|\mathbf{x}|} \lambda^{|\mathbf{x}|-i-1}\ \mathbf{x}_i\otimes\mathbf{x}_j$$

(b) Multiplicative mapping, aggregation <u>normalized</u>:

$$\mathcal{K}_2(\mathbf{x},\mathbf{y}) = \frac{1}{Z}\sum_{1\le i<j\le|\mathbf{x}|}\ \sum_{1\le k<l\le|\mathbf{y}|} \lambda^{|\mathbf{x}|-i-1}\,\lambda^{|\mathbf{y}|-k-1}\ \langle \mathbf{x}_i,\mathbf{y}_k\rangle\cdot\langle \mathbf{x}_j,\mathbf{y}_l\rangle$$

$$\text{s.t.}\quad Z = \sum_{1\le i<j\le|\mathbf{x}|}\ \sum_{1\le k<l\le|\mathbf{y}|} \lambda^{|\mathbf{x}|-i-1}\,\lambda^{|\mathbf{y}|-k-1}$$

$$\phi_2(\mathbf{x}) = \frac{1}{Z'}\sum_{1\le i<j\le|\mathbf{x}|} \lambda^{|\mathbf{x}|-i-1}\ \mathbf{x}_i\otimes\mathbf{x}_j$$

$$\text{s.t.}\quad Z' = \sum_{1\le i<j\le|\mathbf{x}|} \lambda^{|\mathbf{x}|-i-1}$$

(c) <u>Additive</u> mapping, aggregation normalized:

$$\mathcal{K}_2(\mathbf{x},\mathbf{y}) = \frac{1}{Z}\sum_{1\le i<j\le|\mathbf{x}|}\ \sum_{1\le k<l\le|\mathbf{y}|} \lambda^{|\mathbf{x}|-i-1}\,\lambda^{|\mathbf{y}|-k-1}\ \left(\langle \mathbf{x}_i,\mathbf{y}_k\rangle + \langle \mathbf{x}_j,\mathbf{y}_l\rangle\right)$$

$$\text{s.t.}\quad Z = \sum_{1\le i<j\le|\mathbf{x}|}\ \sum_{1\le k<l\le|\mathbf{y}|} \lambda^{|\mathbf{x}|-i-1}\,\lambda^{|\mathbf{y}|-k-1}$$

$$\phi_2(\mathbf{x}) = \frac{1}{Z'}\sum_{1\le i<j\le|\mathbf{x}|} \lambda^{|\mathbf{x}|-i-1}\ [\mathbf{x}_i,\mathbf{x}_j]$$

$$\text{s.t.}\quad Z' = \sum_{1\le i<j\le|\mathbf{x}|} \lambda^{|\mathbf{x}|-i-1}$$

Table 2.3: Kernel functions and associated mappings for RCNNs shown in Table 2.2. $[\mathbf{x}_i,\mathbf{x}_j]$ denotes the concatenation of two vectors.

to produce the final output state $\mathbf{h}[t]$. It turns out that many activations are also functions in RKHS of certain kernel functions, including the quadratic activation $\sigma_{sq}(x) = x^2$, sigmoid-like and ReLU-like activations (see Shalev-Shwartz et al. [92], Zhang et al. [120, 121]). Given these facts, RCNN layer with non-linear activations can also be related to RKHS as well. We give the formal statements below.

Lemma 1. (Shalev-Shwartz et al. [92], Zhang et al. [120]) *Let* $\mathbf{x}$ *and* $\mathbf{w}$ *be multi-dimensional vectors with finite norm. Consider the function* $f(\mathbf{x}) := \sigma(\mathbf{w}^\top \mathbf{x})$ *with non-linear activation* $\sigma(\cdot)$*. For activation functions such as polynomials and sigmoid function* σ_{erf}*, there exists kernel functions* $\mathcal{K}_\sigma(\cdot,\cdot)$ *and the underlying mapping* $\phi_\sigma(\cdot)$ *such that* $f(x)$ *is in the reproducing kernel Hilbert space of* $\mathcal{K}_\sigma(\cdot,\cdot)$*, i.e.,*

$$f(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x}) = \langle \phi_\sigma(\mathbf{x}), \psi(\mathbf{w}) \rangle$$

for some mapping $\psi(\mathbf{w})$ *constructed from* $\mathbf{w}$*. In particular,* $\mathcal{K}_\sigma(\mathbf{x}, \mathbf{y})$ *can be the inverse-polynomial kernel* $\frac{1}{2-\langle \mathbf{x},\mathbf{y} \rangle}$ *for the above activations.*

Theorem 2. *For one RCNN layer with non-linear activation* $\sigma(\cdot)$ *discussed above, assuming the final output state* $\mathbf{h}[t] = \sigma(\mathbf{c}_n[t])$ *(or* ***any linear combination*** *of* $\{\mathbf{c}_i[t]\}, i = 1, .., n$*), then* $\mathbf{h}[t][i]$ *as a function of input* $\mathbf{x}$ *belongs to the RKHS introduced by the composition of* $\mathcal{K}_\sigma(\cdot,\cdot)$ *and string kernel* $\mathcal{K}_n(\cdot,\cdot)$*.*

Here a kernel composition $\mathcal{K}_{\sigma,n}(\mathbf{x}, \mathbf{y})$ *is defined with the underlying mapping* $\mathbf{x} \mapsto \phi_\sigma(\phi_n(\mathbf{x}))$*, and hence* $\mathcal{K}_{\sigma,n}(\mathbf{x}, \mathbf{y}) = \phi_\sigma(\phi_n(\mathbf{x}))^\top \phi_\sigma(\phi_n(\mathbf{y}))$*.*

Theorem 2 is the corollary of Lemma 1 and Theorem 1, since $\mathbf{h}[t][i] = \sigma(\mathbf{c}_n[t][i]) = \sigma(\mathcal{K}_n(\mathbf{x}_{1:t}, \mathbf{w}_{i,j})) = \langle \phi_\sigma(\phi_n(\mathbf{x}_{1:t})), \tilde{\mathbf{w}}_{i,j} \rangle$ and $\phi_\sigma(\phi_n(\cdot))$ is the mapping for the composed kernel. The same proof applies when $\mathbf{h}[t]$ is a linear combination of all $\mathbf{c}_i[t]$ since kernel functions are close under addition.

2.4.3 Deep RCNN Models

We now address the case when multiple RCNN layers are stacked to construct deeper networks. That is, the output states $\mathbf{h}^{(l)}[t]$ of the l-th layer are fed to the $(l+1)$-th RCNN layer as the input sequence. We show that layer stacking corresponds to recursive kernel construction (i.e. $(l+1)$-th kernel is defined on top of l-th kernel), which has been proved for feed-forward networks [120].

We first generalize the sequence kernel definition to enable recursive construction. Notice that the definition in Eq.(2.7) uses the inner product $\langle \mathbf{x}_i, \mathbf{y}_k \rangle$ as a "subroutine" to measure the similarity between substructures (e.g. tokens) within the sequences. We can therefore replace it with other similarity measures introduced by kernels. In particular, let $\mathcal{K}^{(1)}(\cdot,\cdot)$ be the kernel function introduced by a single RCNN layer. The generalized sequence kernel can be recursively defined as,

$$\begin{aligned}
&\mathcal{K}^{(l+1)}(\mathbf{x},\mathbf{y}) \qquad\qquad\qquad\qquad\qquad\qquad \textit{for } l = 1,2,\cdots,L-1 \\
&= \sum_{1\le i<j\le|\mathbf{x}|} \sum_{1\le k<l\le|\mathbf{y}|} \lambda^{|\mathbf{x}|-i-1+|\mathbf{y}|-k-1} \; \mathcal{K}^{(l)}(\mathbf{x}_{1:i},\mathbf{y}_{1:k}) \; \mathcal{K}^{(l)}(\mathbf{x}_{1:j},\mathbf{y}_{1:l}) \\
&= \sum_{1\le i<j\le|\mathbf{x}|} \sum_{1\le k<l\le|\mathbf{y}|} \lambda^{|\mathbf{x}|-i-1+|\mathbf{y}|-k-1} \left\langle \phi^{(l)}(\mathbf{x}_{1:i}), \phi^{(l)}(\mathbf{y}_{1:k}) \right\rangle \left\langle \phi^{(l)}(\mathbf{x}_{1:j}), \phi^{(l)}(\mathbf{y}_{1:l}) \right\rangle \\
&= \left\langle \sum_{1\le i<j\le|\mathbf{x}|} \lambda^{|\mathbf{x}|-i-1} \phi^{(l)}(\mathbf{x}_{1:i}) \otimes \phi^{(l)}(\mathbf{x}_{1:j}), \sum_{1\le k<l\le|\mathbf{y}|} \lambda^{|\mathbf{y}|-k-1} \phi^{(l)}(\mathbf{y}_{1:k}) \otimes \phi^{(l)}(\mathbf{y}_{1:l}) \right\rangle
\end{aligned} \tag{2.8}$$

where $\phi^{(l)}(\cdot)$ is the underlying mapping for kernel $\mathcal{K}^{(l)}(\cdot,\cdot)$ and L is the maximum level of recursive construction (or equivalently the number of stacking layers). Similarly, when non-linear activation $\sigma(\cdot)$ is involved, we shall replace $\phi^{(l)}(\cdot)$ with the composition $\phi_\sigma(\phi^{(l)}(\cdot))$, and we denote the associated kernel as $\mathcal{K}_\sigma^{(l)}(\cdot,\cdot)$. Based on this definition, a deeper model can be interpreted as kernel computation as well.

Theorem 3. *For a RCNN model with L stacking layers and activation function $\sigma(\cdot)$, assuming the final output state $\mathbf{h}^{(l)}[t] = \sigma(\mathbf{c}_n[t])$ (or any linear combination of $\{\mathbf{c}_i^{(l)}[t]\}, i = 1,..,n$), then for any $l = 1,\cdots,L$,*

(i) $\mathbf{c}_n^{(l)}[t][i]$ as a function of input $\mathbf{x}$ belongs to the RKHS of kernel $\mathcal{K}^{(l)}(\cdot,\cdot)$;

(ii) $\mathbf{h}^{(l)}[t][i]$ belongs to the RKHS of kernel $\mathcal{K}_\sigma^{(l)}(\cdot,\cdot)$.

The proof of this theorem is included in Appendix A.3.

2.4.4 Adaptive Decay using Neural Gates

The sequence kernel and neural network discussed so far use a constant decay value regardless of the current input token. However, this is often not the case since the importance of the input tokens can vary across the context or the applications. One extension is to make use of neural gates that adaptively control the decay factor, i.e. $\lambda_t = \sigma(\mathbf{U}\mathbf{x}_t)$ or $\lambda_t = \sigma(\mathbf{U}\mathbf{x}_t + \mathbf{V}\mathbf{h}[t-1])$ where $\sigma(\cdot)$ is a sigmoid function.

When the decay only depends on the input ($\mathbf{x}_t$ or any context), the theoretical results and the sequence kernels can be extended accordingly. Specifically, let's again assume Lemma 1 applies to the sigmoid function $\sigma(\cdot)$ that is used to calculate the adaptive decay value. Let $\mathcal{K}_\sigma(\cdot,\cdot)$ be the corresponding kernel function whose RKHS contains $\sigma(\cdot)$, and let us assume $0 \leq \mathcal{K}_\sigma(\cdot,\cdot) \leq 1$. Let $\mathbf{x}_{1:t}$ and $\mathbf{y}_{1:t'}$ be two sequences. We can define a sequence kernel and the underlying mapping in a recursive manner,

$$\mathcal{K}(\mathbf{x}_{1:t}, \mathbf{y}_{1:t'}) = \mathcal{K}_\sigma(\mathbf{x}_t, \mathbf{y}_{t'}) \cdot \mathcal{K}(\mathbf{x}_{1:t-1}, \mathbf{y}_{1:t'-1}) + \langle \mathbf{x}_t, \mathbf{y}_{t'} \rangle$$

$$\phi(\mathbf{x}_{1:t}) = [\mathbf{x}_t, \phi(\mathbf{x}_{1:t-1}) \otimes \phi_\sigma(\mathbf{x}_t)]$$

The corresponding neural operation is then

$$\mathbf{c}[t] = \lambda_t \odot \mathbf{c}[t-1] + \mathbf{W}\mathbf{x}_t$$

Similarly, for a deep kernel and network, the inner product $\langle \mathbf{x}_t, \mathbf{y}_{t'} \rangle$ can be replaced by a base kernel function (e.g. the one associated with shallow network as we discussed in the previous section).

Although working this out is mathematically more involved, we can understand the adaptive version in an simpler way. Since Lemma 1 applies to $\sigma(\cdot)$, it can be also expressed as an inner product in the space introduced by a kernel mapping $\sigma(\mathbf{u}^\top \mathbf{x}_t) = \langle \phi_\sigma(\mathbf{x}_t), \psi(\mathbf{u}) \rangle$. Let's write $\phi_\sigma(\mathbf{x}_t)$ and $\psi(\mathbf{u})$ as $\tilde{\mathbf{x}}_t$ and $\tilde{\mathbf{u}}$ for short. Note that the internal hidden state $\mathbf{c}[t]$ is computed in the form such as $\mathbf{c}[t] = \lambda_t \odot \mathbf{c}[t-1] + \mathbf{W}\mathbf{x}_t$. If we plug in $\lambda_t = \sigma(\mathbf{U}\mathbf{x}_t)$ and expand the formula, we can rewrite each entry of the state $\mathbf{c}[t][i]$ as,

$$\begin{aligned}
\mathbf{c}[t][i] &= \mathbf{c}[t-1][i] \cdot \lambda_t[i] + \langle \mathbf{w}_i, \mathbf{x}_t \rangle \\
&= \mathbf{c}[t-1][i] \cdot \langle \tilde{\mathbf{u}}_i, \tilde{\mathbf{x}}_t \rangle + \langle \mathbf{w}_i, \mathbf{x}_t \rangle \\
&= \mathbf{c}[t-2][i] \cdot \langle \tilde{\mathbf{u}}_i, \tilde{\mathbf{x}}_{t-1} \rangle \cdot \langle \tilde{\mathbf{u}}_i, \tilde{\mathbf{x}}_t \rangle + \langle \mathbf{w}_i, \mathbf{x}_{t-1} \rangle \cdot \langle \tilde{\mathbf{u}}_i, \tilde{\mathbf{x}}_t \rangle + \langle \mathbf{w}_i, \mathbf{x}_t \rangle \\
&= \cdots + \langle \mathbf{w}_i, \mathbf{x}_{t-2} \rangle \cdot \langle \tilde{\mathbf{u}}_i, \tilde{\mathbf{x}}_{t-1} \rangle \cdot \langle \tilde{\mathbf{u}}_i, \tilde{\mathbf{x}}_t \rangle + \langle \mathbf{w}_i, \mathbf{x}_{t-1} \rangle \cdot \langle \tilde{\mathbf{u}}_i, \tilde{\mathbf{x}}_t \rangle + \langle \mathbf{w}_i, \mathbf{x}_t \rangle
\end{aligned}$$

In other words, the state $\mathbf{c}[t][i]$ becomes a polynomial that involves unbounded high-order terms over all previous inputs $\mathbf{x}_1 \cdots \mathbf{x}_t$. In contrast, if the decay value is constant, the n-th internal layer $\mathbf{c}_n[t]$ would be (at most) a n-order polynomial.

The above results suggest that an adaptive RCNN has more expressive power than the constant version. We also conjecture that the network gains additional power when the decay depends on both the input and the previous hidden state. We will empirically demonstrate this in the experiments.

To summarize Section 2.4, we show that RCNN models have a close connection with string kernels (and their generalization). Specifically, the function space introduced by the kernels contain all the predictor constructed from RCNNs (e.g. $\theta^\top \mathbf{h}_n[t]$). Training a RCNN model in this sense can be interpreted as finding a good predictor in that function space under the RCNN parameterization.[4] The result suggests why RCNNs (and perhaps similar recurrent structures) would be a good sequential component, since the underlying kernels are indeed a suitable choice for sequences.

[4] This is different to traditional kernel methods, where the parameterization is a linear combination of representers associated with each training example.

2.5 Applications and Evaluations

In this section, we apply the recurrent convolution component to various NLP tasks and empirically evaluate its performance against other neural network models. These tasks include language modeling, text classification and text retrieval. We discuss each of them in a separate subsection.

2.5.1 Language Modeling on PTB

Dataset and Setup We analyze the performance of RCNNs and alternative neural networks on the language modeling task. We use the Penn Tree Bank (PTB) corpus as the benchmark. The dataset contains about 1 million tokens in total. We use the standard train/development/test split of this dataset and limit the vocabulary size to 10,000.

We run two sets of experiments using Adam [57] and SGD as the optimizer respectively. In the first setup, we aim to explore different versions of RCNNs shown in Table 2.2 and use either state $\mathbf{c}[t]$ or $\mathbf{z}[t]$ as the pre-activation output state. We also train RCNNs with the adaptive decay controlled by neural gates (Section 2.3.5). The model's behavior can be sensitive to the parameter initialization, especially for the multiplicative feature mapping (e.g. $\mathbf{W}^{(1)}\mathbf{x}_i \odot \mathbf{W}^{(2)}\mathbf{x}_j$).[5] For a fair comparison between different models, we use Adam optimization method to alleviate the initialization issue. We train LSTM models under the same setting, and also include CNN models for comparison since CNNs are a special case of our model (corresponding to decay value $\lambda = 0$).

In the second setup, we compare with state-of-the-art models on this dataset. To this end, we follow the previous work of Zaremba et al. [116] and use SGD with gradient clipping [82] and learning rate decay as the optimization strategy. Specifically, the initial learning rate is 1.0 and is decreased by a constant factor after a certain epoch.

[5] Due to multiplication $\mathbf{u} \odot \mathbf{v}$, the gradient w.r.t one vector depends on the values of the other vector. This leads to easy gradient vanishing or explosion in vanilla SGD optimization. This limits the applicability of multiplicative RCNNs. A future work would be to find a good strategy for optimizing the network.

These choices are crucial for the PTB dataset. In addition, we add highway connections [101] between the input and the output of each RCNN layer in order to train deeper model that has multiple RCNN layers. We back-propagate the gradient with an unroll size of 35 and use the standard dropout [41] as the regularization. For each model run, we train a maximum of 50 epochs and select the best epoch based on the performance on the development set. The evaluation metric is word-level perplexity. The detailed experimental setup is shown in Appendix B.1.

Results Table 2.4 shows the results of each model when trained using Adam optimizer. The RCNN model with the adaptive decay achieves better performance compared to RCNN with a constant decay and the LSTM model.[6] In addition, the RCNN component clearly outperforms its counterpart, the traditional CNNs, getting perplexities that are 30 to 40 points lower compared to the latter. This illustrates the effectiveness of specifically tailoring a neural component for sequential data.

Table 2.4 also shows the results of two additional RCNN variants. First, our theoretical results suggest that any linear combination of the internal states is a valid function in the RKHS. We can simply use the sum of these states as the pre-activation output (i.e. $\sum_{i=1}^{n} \mathbf{c}_i[t]$, corresponding to combining 1-gram to n-gram kernels). Second, recent work found that the neural output gate is necessary in language modeling. The output states of our model can be re-scaled by an output gate, similar to that in LSTMs. As shown in the table, these modifications lead to improved results.

Figure 2-1 analyzes the importance of hyper-parameters for RCNN models. As shown in the figure, deeper RCNN models (i.e. stacking multiple RCNN layers) achieve better performance compared to single-layer models, indicating better representational power. Figure 2-1 also suggests that the multiplicative feature mapping (e.g. $\mathbf{W}^{(1)}\mathbf{x}_1 \odot \mathbf{W}^{(2)}\mathbf{x}_2$) performs better than the additive linear mapping. This is also true for traditional CNNs – the multiplicative filters decrease the perplexity by 10 points, as demonstrated in Table 2.4. However, the multiplicative version is much

[6] Best version is the multiplicative, normalized version using $\mathbf{c}_n[t]$ as the pre-activation output, presented in Table 2.2 (b). Other versions achieve worse results, but not too much.

Model	d	$\|\theta\|$	**Test PPL**
CNN, 3-gram, linear	250	5.4m	141.7
CNN, 2-gram, multiplicative	250	5.4m	132.8
LSTM	225	4.9m	107.6
RCNN, 2-gram, $\lambda = 0.8$	240	5.2m	115.2
RCNN, 2-gram, λ adaptive	230	5.0m	103.9
LSTM	650	16.4m	92.3
RCNN, 2-gram, $\lambda = 0.8$	650	16.4m	106.2
RCNN, 2-gram, λ adaptive	650	16.4m	91.8
+ $\mathbf{h}[t] = \tanh(\mathbf{c}_1[t] + \mathbf{c}_2[t])$	650	16.4m	89.6
+ $\mathbf{h}[t] = \tanh(\mathbf{c}_1[t] + \mathbf{c}_2[t])$ and outgate	650	18.1m	85.9

Table 2.4: Results of CNN, LSTM and RCNN using Adam optimizer. d is the hidden dimension and $|\theta|$ denotes the number of parameters. Best results of each category are reported. For CNNs, we explore 1 to 3 layers with 2-gram or 3-gram feature maps. For LSTMs and RCNNs, we explore 1 or 2 layers. The last two rows present the results of two additional variants: (a) using the sum of all internal states as the pre-activation output and (b) adding an output gate similar to that in LSTMs.

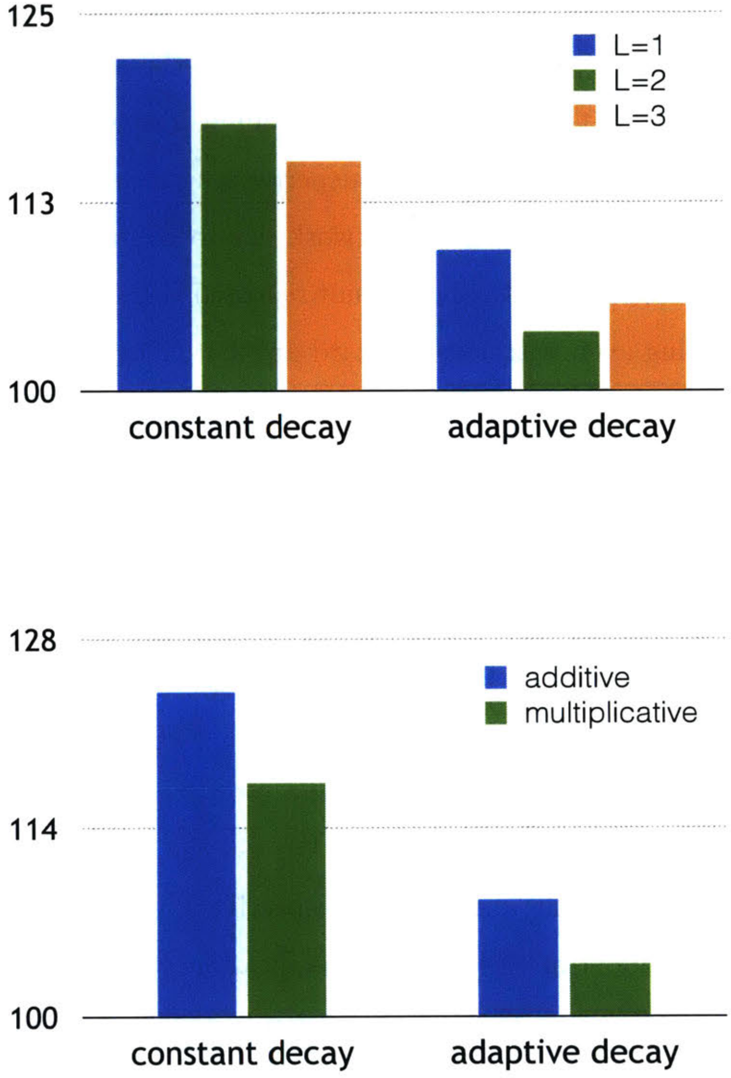


Figure 2-1: Analyses of RCNN models using Adam optimizer on the language modeling task. We compare the variants shown in Table 2.2 with either a constant or an adaptive decay. We show the test perplexity of small networks (5m parameters). Best runs of each category are reported here. Top: deeper architecture (via stacking RCNN layers) exhibits better expressive power and delivers better performance. Bottom: multiplicative feature mapping performs better on language modeling task.

more sensitive to the optimization method and parameter initialization.[7] This limits the applicability of the multiplicative mapping.

Table 2.5 compares our model with various state-of-the-art models. Our results are competitive to state-of-the-art results. For example, a small model with 5 million parameters achieves a test perplexity of 76, outperforming many results achieved using much larger network. By increasing the network size to 20 million, we obtain a test perplexity of 69.7, being close to the best result reported [122, 123]. The large models are prone to overfitting even with the standard dropout. Our results may be further improved with better regularization techniques such as the variational dropout [30], which have been shown to improve several points of perplexity on PTB.

2.5.2 Sentiment Classification

Datasets We evaluate our model on sentence sentiment classification task. We use the Stanford Sentiment Treebank benchmark [97]. The dataset consists of 11855 parsed English sentences annotated at both the root (i.e. sentence) level and the phrase level using 5-class fine-grained labels. We use the standard 8544/1101/2210 split for training, development and testing respectively. Following previous work, we also evaluate our model on the binary classification variant of this benchmark, ignoring all neutral sentences. The binary version has 6920/872/1821 sentences for training, development and testing.

Baselines We compare our model with a wide range of neural network models on the sentence sentiment classification task. Most of these models fall into either the category of recursive neural networks (RNNs) or the category of convolutional neural networks (CNNs). The recursive neural network baselines include standard **RNN** [95], **RNTN** with a small core tensor in the composition function [97], the deep recursive model **DRNN** [43] and the most recent recursive model using long-short-term-memory units **RLSTM** [103]. These recursive models assume the input

[7] For example, the training cost may remain high due to gradient vanishing. This happens more often for models with more layers and larger n ($n > 2$).

Model	$\lvert\theta\rvert$	**Test PPL**
LSTM (medium) [116]	20m	82.7
LSTM (large) [116]	66m	78.4
Character CNN [56]	19m	78.9
Variational LSTM (medium) [30]	20m	78.6
Variational LSTM (large) [30]	66m	73.4
Variational LSTM (shared emb) [86]	51m	73.2
Pointer Sentinel-LSTM [75]	21m	70.9
Variational RHN (9 layers, shared emb) [122]	24m	66.0
Neural Net Search (variational, shared emb) [123]	25m	64.0
RCNN ($\lambda = 0.8$)	5m	84.3
RCNN (λ constants trained as parameters)	5m	77.0
RCNN (λ adaptive based on $\mathbf{x}$)	5m	74.2
RCNN (λ adaptive based on $\mathbf{x}$)	20m	70.9
RCNN (λ adaptive based on $\mathbf{x}$ and $\mathbf{h}$)	5m	73.6
RCNN (λ adaptive based on $\mathbf{x}$ and $\mathbf{h}$)	20m	69.2

Table 2.5: Comparison with state-of-the-art results on PTB. $|\theta|$ denotes the number of parameters. Following these work, we use SGD with gradient clipping and learning rate decay during training. We also share the input and output word embedding matrix [86]. However, variational dropout regularization has not been explored.

sentences are represented as parse trees. As a benefit, they can readily utilize annotations at the phrase level. In contrast, convolutional neural networks are trained on sequence-level, taking the original sequence and its label as training input. Such convolutional baselines include the dynamic CNN with k-max pooling **DCNN** [50] and the convolutional model with multi-channel **CNN-MC** by Kim [55]. To leverage the phrase-level annotations in the Stanford Sentiment Treebank, all phrases and the corresponding labels are added as separate instances when training the sequence models. We report results based on this strategy.

Word vectors The word vectors are pre-trained on much larger unannotated corpora to achieve better generalization given limited amount of training data [107]. In particular, we use the publicly available 300-dimensional GloVe word vectors trained on the Common Crawl with 840B tokens [84]. This choice of word vectors follows most recent work, such as **DAN** [45] and **RLSTM** [103]. The word vectors are normalized to unit norm (i.e. $\|w\|_2^2 = 1$) and are fixed in the experiments without fine-tuning.

Hyperparameter setting Our model configuration largely follows our prior work [66]. We stack several RCNN layers and apply dropout on the word embeddings and the output states of each layer. We use mean-pooling to average the output states across different word positions and fed the averaged vector as the input to the final softmax layer. We perform a few independent runs to explore different random initializations. The detailed experimental setup is shown in Appendix B.2.

Results Table 2.6 presents the performance of our model and other baseline methods on Stanford Sentiment Treebank benchmark. Our best model obtains the highest performance, achieving 53.2% and 89.9% test accuracies on fine-grained and binary tasks respectively. Our model with only a constant decay factor also obtains quite high accuracy, outperforming other baseline methods shown in the table.

Similar to the experiments of the language modeling application, we also investigate the impact of hyperparameters in the model performance. To this end, we train

Model	**Fine**	**Binary**
RNN (Socher et al. [95])	43.2	82.4
RNTN (Socher et al. [97])	45.7	85.4
DRNN (Irsoy and Cardie [43])	49.8	86.8
RLSTM (Tai et al. [103])	51.0	88.0
DCNN (Kalchbrenner et al. [50])	48.5	86.9
CNN-MC (Kim [55])	47.4	88.1
Bi-LSTM (Tai et al. [103])	49.1	87.5
LSTMN (Cheng et al. [12])	47.9	87.0
PVEC (Le and Mikolov [61])	48.7	87.8
DAN (Iyyer et al. [44])	48.2	86.8
DMN (Kumar et al. [59])	52.1	88.6
RCNN, $\lambda = 0.5$ (Lei et al. [66])	51.2	88.6
RCNN, λ adaptive on $\mathbf{x}$	51.4	89.2
RCNN, λ adaptive on $\mathbf{x}$ and $\mathbf{h}$	53.2	89.9

Table 2.6: Comparison between our RCNN model and other baseline methods on Stanford Sentiment Treebank. The top block lists recursive neural network models, the second block are convolutional network models and recurrent neural models, and the third block contains other baseline methods, including the paragraph-vector model [61], the deep averaging network model [45] and the dynamic memory network [59].

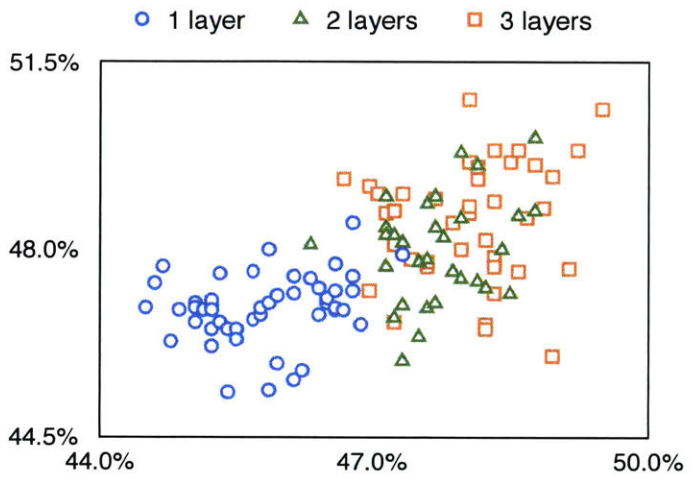


Figure 2-2: Dev accuracy (x-axis) and test accuracy (y-axis) of independent runs of our model on fine-grained sentiment classification task. Deeper architectures achieve better accuracies.

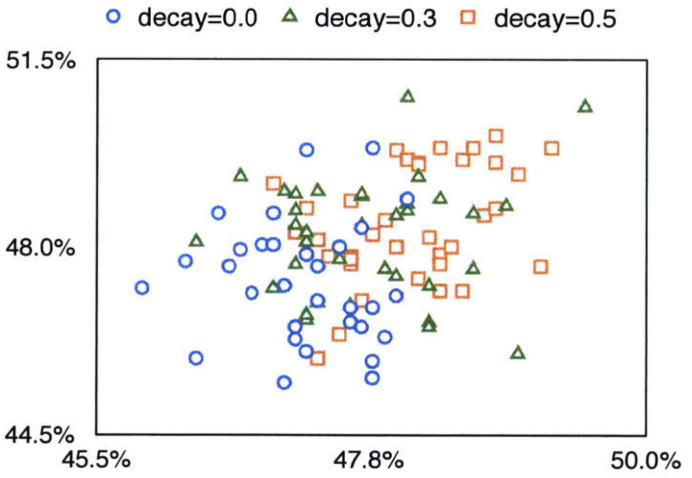


Figure 2-3: Comparison of our model variations in sentiment classification task when considering consecutive n-grams only (decaying factor $\lambda = 0$) and when considering non-consecutive n-grams ($\lambda > 0$). Modeling non-consecutive n-gram features leads to better performance.

many independent runs exploring various hyperparameter configurations. To reduce the total running time, we use the RCNN models with constant decay value and exclude phrase-level annotations during training. We plot the fine-grained sentiment classification accuracies on both the development set and the test set obtained during this hyperparameter grid search.

Figure 2-2 illustrates how the number of feature layers impacts the model performance. As shown in the figure, adding higher-level features by stacking more layers clearly improves the classification accuracy across various other hyperparameter settings and initializations. Again, this observation demonstrates deeper RCNN architectures possess more representational power compared to shallow models.

We also analyze the effect of modeling non-consecutive n-grams. Figure 2-3 splits the model accuracies according to the choice of span decaying factor λ. Note when $\lambda = 0$, the model only applies feature extractions to consecutive n-grams and hence becomes a traditional CNNs. As shown in Figure 2-3, this setting leads to consistent performance drop. This result confirms the importance of handling non-consecutive n-gram patterns.

Finally we conduct case studies on the learned sentiments. Figure 2-4 gives examples of input sentences and the corresponding predictions of our model in fine-grained sentiment classification. To see how our model captures the sentiment at different local context, we apply the learned softmax activation (i.e. the output layer) to the extracted features (i.e. the hidden states $\mathbf{h}^{(l)}[i]$) at each position. That is, for each index i, we obtain the local sentiment $p = \text{softmax}\left(\mathbf{W}^\top[\mathbf{h}^{(1)}[i]; ...; \mathbf{h}^{(L)}[i]]\right)$. We plot the expected sentiment scores $\sum_{s=-2}^{2} s \cdot p(s)$, where a score of 2 means "very positive", 0 means "neutral" and -2 means "very negative". As shown in the figure, our model successfully learns negation and double negation. The model also identifies positive and negative segments appearing in the sentence.

2.5.3 Similar Question Retrieval

Our third application is a text retrieval task for community-based question answering. Given a user input question post and a set of candidate questions, the goal is to rank

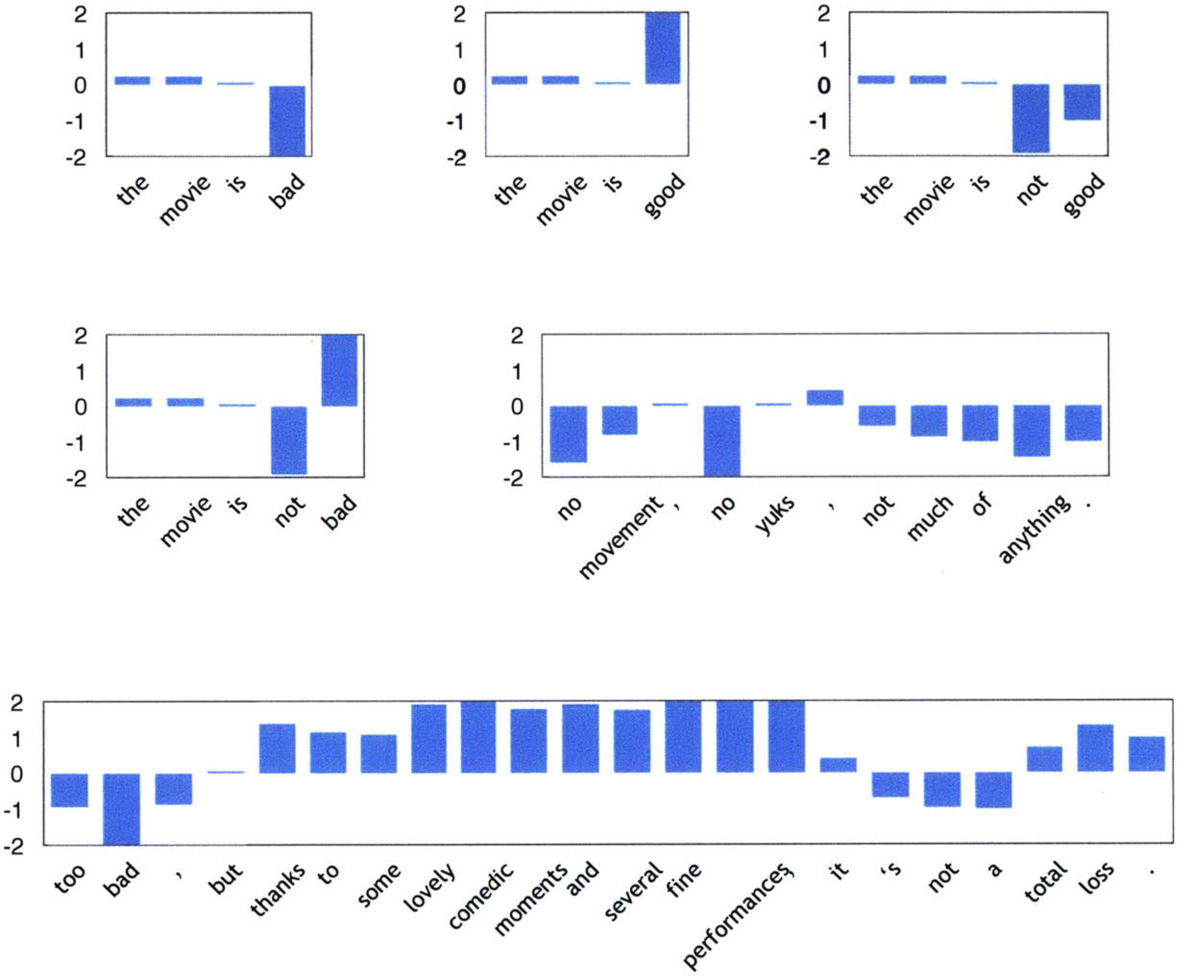
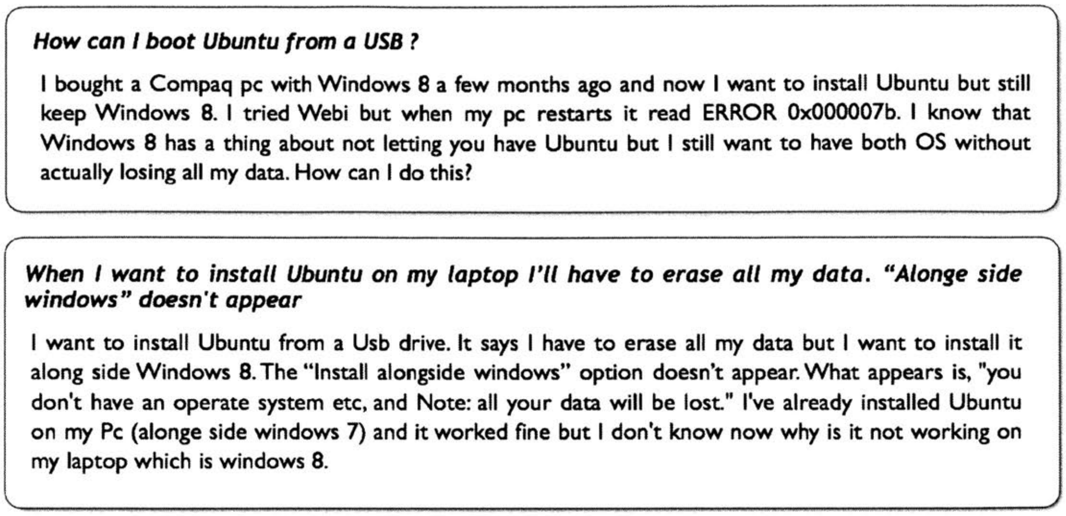


Figure 2-4: Example sentences and their sentiments predicted by our RCNN model trained with root labels. Our model successfully identifies negation, double negation and phrases with different sentiment in one sentence.



How can I boot Ubuntu from a USB ?

I bought a Compaq pc with Windows 8 a few months ago and now I want to install Ubuntu but still keep Windows 8. I tried Webi but when my pc restarts it read ERROR 0x000007b. I know that Windows 8 has a thing about not letting you have Ubuntu but I still want to have both OS without actually losing all my data. How can I do this?

When I want to install Ubuntu on my laptop I'll have to erase all my data. "Alonge side windows" doesn't appear

I want to install Ubuntu from a Usb drive. It says I have to erase all my data but I want to install it along side Windows 8. The "Install alongside windows" option doesn't appear. What appears is, "you don't have an operate system etc, and Note: all your data will be lost." I've already installed Ubuntu on my Pc (alonge side windows 7) and it worked fine but I don't know now why is it not working on my laptop which is windows 8.

Figure 2-5: An example of similar posts taken from AskUbuntu question answering forum. Both questions ask about installing Ubuntu from USB. A question post consists of a *title* section and a detailed *body* section.

these candidates according to their semantic relevance to the input post. Figure 2-5 gives an example pair of relevant questions pertaining to booting Ubuntu using a USB stick. A large portion of the body contains tangential details that are idiosyncratic to this user, such as references to Compaq pc, Webi and the error message mentioned in the first question. Not surprisingly, these features are not repeated in the second question in Figure 2-5. The extraneous detail makes the task quite difficult since it can easily confuse simple word-matching algorithms.

Model Framework To better access the semantic relevance of questions, we adopt neural networks as encoders to map question posts into vector representations, and measure the relevance based on the cosine similarity of these vectors (Figure 2-6 (a)).[8] The model is optimized to maximize the similarities of similar questions against non-similar ones (using user-marked relevant questions from the forum), therefore has potentials to outperform unsupervised word-matching algorithms.

[8] The representations of the question title and body are averaged to get a single representation of the question.

Corpus	# of unique questions	167,765
	Avg length of title	6.7
	Avg length of body	59.7
Training	# of unique questions	12,584
	# of user-marked pairs	16,391
Dev	# of query questions	200
	# of annotated pairs	200×20
	Avg # of positive pairs per query	5.8
Test	# of query questions	200
	# of annotated pairs	200×20
	Avg # of positive pairs per query	5.5

Table 2.7: Various statistics from our training, development, and test sets derived from the Sept. 2014 Stack Exchange AskUbuntu dataset.

The number of questions in a question answering forum far exceeds user annotations of pairs of similar questions. To further leverage the data and the potential of neural networks, we train an auto-encoder constructed by pairing the encoder model (such as our RCNNs) with an corresponding decoder (of the same type). As illustrated in Figure 2-6 (b), the resulting encoder-decoder architecture is akin to those used in machine translation [], which represent a conditional language model P(title|context). In our case, the context can be any of (a) the original title itself, (b) the question body and (c) the title/body of a similar question. All possible (title, context) pairs are used during training to optimize the likelihood of the words (and their order) in the titles. The encoders pre-trained in this manner are subsequently fine-tuned to optimize the cosine similarity (as described above).

Dataset We use the Stack Exchange AskUbuntu dataset used in prior work [25]. This dataset contains 167,765 unique questions, each consisting of a question title and a question body [9], and a set of user-marked similar question pairs. We provide various statistics from this dataset in Table 2.7.

[9] We truncate the body section at a maximum of 100 words

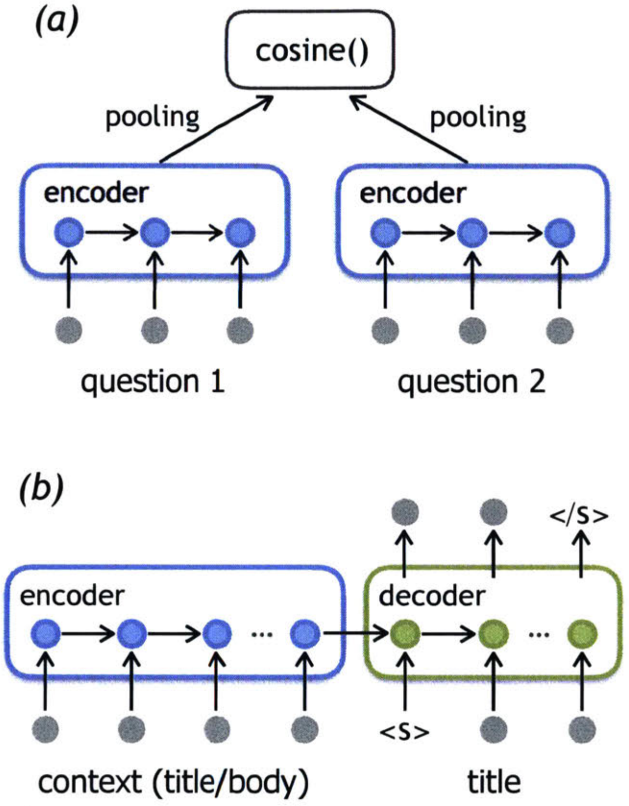


Figure 2-6: An illustration of the neural network architecture for question retrieval: (a) the neural encoder maps questions into vector representations and measures their semantic relevance via cosine similarity; (b) the encoder-decoder model is pre-trained to better initialize the encoder. We explore various choices of neural networks as the encoder and decoder, including for example LSTMs, GRUs and our proposed RCNNs.

Evaluation Setup User-marked similar question pairs on QA sites are often known to be incomplete. In order to evaluate this in our dataset, we took a sample set of questions paired with 20 candidate questions retrieved by a search engine trained on the AskUbuntu data. The search engine used is the well-known BM25 model [88]. Our manual evaluation of the candidates showed that only 5% of the similar questions were marked by users, with a precision of 79%. Clearly, this low recall would not lead to a realistic evaluation if we used user marks as our gold standard. Instead, we make use of expert annotations carried out on a subset of questions.

We use user-marked similar pairs as positive pairs in training since the annotations have high precision and do not require additional manual annotations. This allows us to use a much larger training set. We use random questions from the corpus paired with each query question p_i as negative pairs in training. We randomly sample 20 questions as negative examples for each p_i during each epoch.

We re-constructed the new dev and test sets consisting of the first 200 questions from the dev and test sets provided by dos Santos et al. [25]. For each of the above questions, we retrieved the top 20 similar candidates using BM25 and manually annotated the resulting 8K pairs as similar or non-similar.[10]

Baselines and Evaluation Metrics We evaluated neural network models including **CNNs**, **LSTMs**, **GRUs** and **RCNNs** by comparing with the following baselines:

- **BM25**, we used the BM25 similarity measure provided by Apache Lucene.
- **TF-IDF**, we ranked questions using cosine similarity based on a vector-based word representation for each question.
- **SVM**, we trained a re-ranker using SVM-Light [46] with a linear kernel incorporating several similarity measures from the DKPro similarity package [6].

We evaluated the models based on the following IR metrics: Mean Average Precision (MAP), Mean Reciprocal Rank (MRR), Precision at 1 (P@1), and Precision at 5

[10] The annotation task was initially carried out by two expert annotators, independently. The initial set was refined by comparing the annotations and asking a third judge to make a final decision on disagreements. After a consensus on the annotation guidelines was reached (producing a Cohen's kappa of 0.73), the overall annotation was carried out by only one expert.

	d	$\|\theta\|$	n
LSTMs	240	423K	-
GRUs	280	404K	-
CNNs	667	401K	3
RCNNs	400	401K	2

Table 2.8: Configuration of neural models. d is the hidden dimension, $|\theta|$ is the number of parameters and n is the filter width (n-gram features).

(P@5).

Hyper-parameters We performed an extensive hyper-parameter search to identify the best model for the baselines and neural network models. For the **TF-IDF** baseline, we tried n-gram feature order $n \in \{1, 2, 3\}$ with and without stop words pruning. For the **SVM** baseline, we used the default SVM-Light parameters whereas the dev data is only used to increase the training set size when testing on the test set. We also tried to give higher weight to dev instances but this did not result in any improvement.

For all the neural network models, we used Adam [57] as the optimization method with the default setting suggested by the authors. We optimized other hyper-parameters with the following range of values: learning rate $\in \{1e-3, 3e-4\}$, dropout [41] probability $\in \{0.1, 0.2, 0.3\}$, CNN and RCNN feature width $\in \{2, 3, 4\}$. We also tuned the pooling strategies and ensured each model has a comparable number of parameters. The default configurations of LSTMs, GRUs, CNNs and RCNNs are shown in Table 2.8. We use the additive and normalized version of RCNNs with adaptive neural gates. We used MRR to identify the best training epoch and the model configuration. For the same model configuration, we report average performance across 5 independent runs.[11]

[11] For a fair comparison, we also pre-train 5 independent models for each configuration and then fine tune these models. We use the same learning rate and dropout rate during pre-training and fine-tuning.

Word Vectors We ran word2vec [77] to obtain 200-dimensional word embeddings using all Stack Exchange data (excluding StackOverflow) and a large Wikipedia corpus. The word vectors are fixed to avoid over-fitting across all experiments.

Results Table 2.9 shows the performance of the baselines and the neural encoder models on the question retrieval task. The results show that our full model, RCNNs with pre-training, achieves the best performance across all metrics on both the dev and test sets. For instance, the full model gets a P@1 of 62.0% on the test set, outperforming the word matching-based method BM25 by over 8 percent points. Further, our RCNN model also outperforms the other neural encoder models and the baselines across all metrics. This superior performance indicates that the use of non-consecutive filters and a varying decay is effective in improving traditional neural network models. Table 2.9 also demonstrates the performance gain from pre-training the RCNN encoder. The RCNN model when pre-trained on the entire corpus consistently gets better results across all the metrics.

We analyze the effect of various pooling strategies for the neural network encoders. As shown in Table 2.10, our RCNN model outperforms other neural models regardless of the two pooling strategies explored. We also observe that simply using the last hidden state as the final representation achieves better results for the RCNN model.

Table 2.11 compares the performance of the TF-IDF baseline and the RCNN model when using question titles only or when using question titles along with question bodies. TF-IDF's performance changes very little when the question bodies are included (MRR and P@1 are slightly better but MAP is slightly worse). However, we find that the inclusion of the question bodies improves the performance of the RCNN model, achieving a 1% to 3% improvement with both model variations. The RCNN model's greater improvement illustrates the ability of the model to pick out components that pertain most directly to the question being asked from the long, descriptive question bodies.

We also analyze the effect of pre-training the encoder-decoder network. Note that, during pre-training, the last hidden states generated by the neural encoder are used by

Method	Dev				Test			
	MAP	MRR	P@1	P@5	MAP	MRR	P@1	P@5
BM25	52.0	66.0	51.9	42.1	56.0	68.0	53.8	42.5
TF-IDF	54.1	68.2	55.6	45.1	53.2	67.1	53.8	39.7
SVM	53.5	66.1	50.8	43.8	57.7	71.3	57.0	43.3
CNNs, mean	58.5	71.1	58.4	46.4	57.6	71.4	57.6	43.2
LSTMs, mean	58.4	72.3	60.0	46.4	56.8	70.1	55.8	43.2
GRUs, mean	59.1	74.0	62.6	47.3	57.1	71.4	57.3	43.6
RCNNs, last	59.9	74.2	63.2	48.0	60.7	72.9	59.1	45.0
LSTMs, pt, mean	58.3	71.5	59.3	47.4	55.5	67.0	51.1	43.4
GRUs, pt, last	59.3	72.2	59.8	48.3	59.3	71.3	57.2	44.3
RCNNs, pt, last	**61.3***	**75.2**	**64.2**	**50.3***	**62.3***	**75.6***	**62.0**	**47.1***

Table 2.9: Comparative results of all methods on the question similarity task. *pt* stands for pre-training. For neural network models, we show the best average performance across 5 independent runs and the corresponding pooling strategy. Statistical significance with $p < 0.05$ against other types of model is marked with $*$.

Method	Dev				Test			
	MAP	MRR	P@1	P@5	MAP	MRR	P@1	P@5
CNNs, max	57.8	69.9	56.6	47.7	59.6	73.1	59.6	45.4
CNNs, mean	58.5	71.1	58.4	46.4	57.6	71.4	57.6	43.2
LSTMs, pt, mean	58.3	71.5	59.3	47.4	55.5	67.0	51.1	43.4
LSTMs, pt, last	57.6	71.0	58.1	47.3	57.6	69.8	55.2	43.7
GRUs, pt, mean	57.5	69.9	57.1	46.2	55.5	67.3	52.4	42.8
GRUs, pt, last	59.3	72.2	59.8	48.3	59.3	71.3	57.2	44.3
RCNNs, pt, mean	59.3	73.6	61.7	48.6	58.9	72.3	57.3	45.3
RCNNs, pt, last	**61.3**	**75.2**	**64.2**	**50.3**	**62.3**	**75.6**	**62.0**	**47.1**

Table 2.10: Choice of pooling strategies. LSTMs, GRUs and RCNNs are all pretrained. *max*, *mean* and *last* stand for the three different pooling strategies.

TF-IDF	MAP	MRR	P@1
title only	54.3	66.8	52.7
title + body	53.2	67.1	53.8
RCNNs, mean-pooling	**MAP**	**MRR**	**P@1**
title only	56.0	68.9	55.7
title + body	58.5	71.7	56.7
RCNNs, last state	**MAP**	**MRR**	**P@1**
title only	58.2	70.7	56.6
title + body	60.7	72.9	59.1

Table 2.11: Comparision between model variants on the test set when question bodies are used or not used.

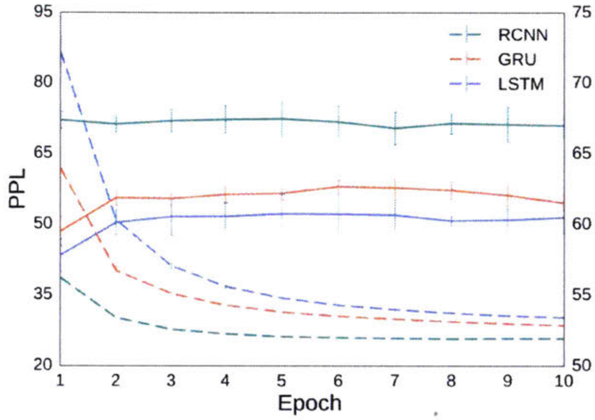


Figure 2-7: Perplexity (dotted lines) on a heldout portion of the corpus versus MRR on the dev set (solid lines) during pre-training. Variances across 5 runs are shown as vertical bars.

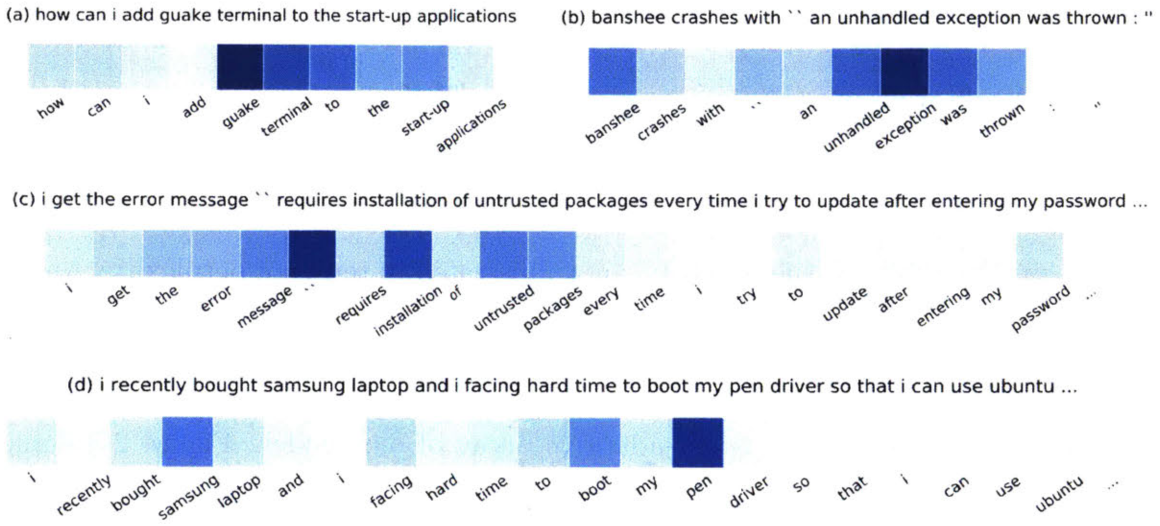
the decoder to reproduce the question titles. It would be interesting to see how such states capture the meaning of questions. To this end, we evaluate MRR on the dev set using the last hidden states of the question titles. We also test how the encoder captures information from the question bodies to produce the distilled summary, i.e. titles. To do so, we evaluate the perplexity of the trained encoder-decoder model on a heldout set of the corpus, which contains about 2000 questions.

As shown in Figure 2-7, the representations generated by the RCNN encoder perform quite well, resulting in a perplexity of 25 and over 68% MRR without the subsequent fine-tuning. Interestingly, the LSTM and GRU networks obtain similar perplexity on the heldout set, but achieve much worse MRR for similar question retrieval. For instance, the GRU encoder obtains only 63% MRR, 5% worse than the RCNN model's MRR performance. As a result, the LSTM and GRU encoder do not benefit clearly from pre-training, as suggested in Table 2.9.

The inconsistent performance difference may be explained by two hypotheses. One is that the perplexity is not suitable for measuring the similarity of the encoded text, thus the power of the encoder is not illustrated in terms of perplexity. Another hypothesis is that the LSTM and GRU encoder may learn non-linear representations therefore their semantic relatedness can not be directly accessed by cosine similarity.

Finally, we analyze the gated adaptive decay of our model. Figure 2-9 demonstrates at each word position t how much input information is taken into the model by the adaptive weights $1 - \lambda_t$. The average of weights in the vector decreases as t increments, suggesting that the information encoded into the state vector saturates when more input are processed. On the other hand, the largest value in the weight vector remains high throughout the input, indicating that at least some information has been stored in $\mathbf{h}[t]$ and $\mathbf{c}[t]$.

We also conduct a case study on analyzing the neural gate. Since directly inspecting the 400-dimensional decay vector is difficult, we train a model that uses a *scalar decay* instead. As shown in Figure 2-8, the model learns to assign higher weights to application names and quoted error messages, which intuitively are important pieces of a question in the AskUbuntu domain.


(a) how can i add guake terminal to the start-up applications

(b) banshee crashes with `` an unhandled exception was thrown : ''

(c) i get the error message `` requires installation of untrusted packages every time i try to update after entering my password ...

(d) i recently bought samsung laptop and i facing hard time to boot my pen driver so that i can use ubuntu ...

Figure 2-8: Visualizations of $1 - \lambda_t$ of our model on several question pieces from the data set. λ_t is set to a *scalar value* (instead of 400-dimension vector) to make the visualization simple. The corresponding model is a simplified variant, which is about 4% worse than our full model.

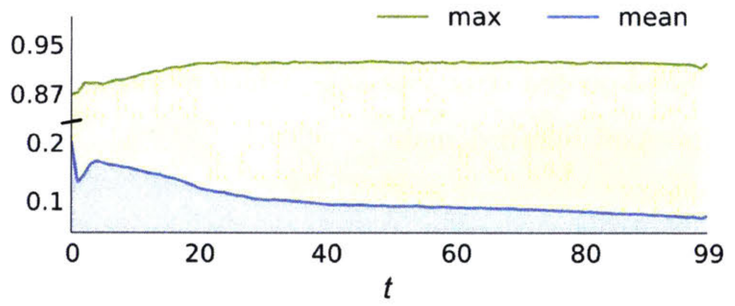


Figure 2-9: The maximum and mean value of the 400-dimentional weight vector $1-\lambda_t$ at each step (word position) t. Values are averaged across all questions in the dev and test set.

3

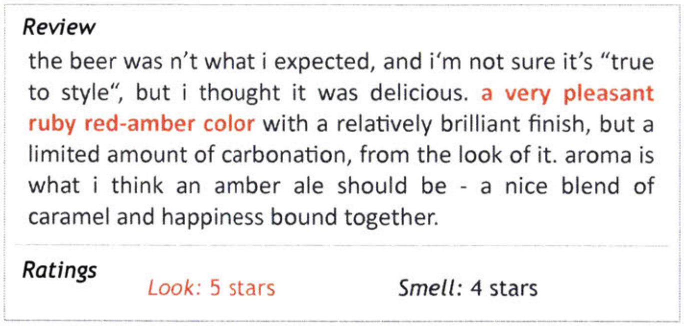
Rationalizing Neural Predictions

Prediction without justification has limited applicability. In this chapter, we learn to extract pieces of input text as justifications – rationales – that are tailored to be short and coherent, yet sufficient for making the same prediction. We first describe our model framework, which consists two modular components, generator and encoder. Thereafter, we present our experiments on multi-aspect sentiment analysis and similar text retrieval. We conclude and discuss some extensions in the last section.

3.1 Introduction

Many recent advances in NLP problems have come from formulating and training expressive and elaborate neural models. This includes models for sentiment classification, parsing, and machine translation among many others. The gains in accuracy have, however, come at the cost of interpretability since complex neural models offer little transparency concerning their inner workings. In many applications, such as medicine, predictions are used to drive critical decisions, including treatment options. It is necessary in such cases to be able to verify and understand the underlying basis for the decisions. Ideally, complex neural models would not only yield improved performance but would also offer interpretable justifications – rationales – for their predictions.

In this chapter, we propose a novel approach to incorporating rationale generation as an integral part of the overall learning problem. We limit ourselves to extractive (as



Review

the beer was n't what i expected, and i'm not sure it's "true to style", but i thought it was delicious. **a very pleasant ruby red-amber color** with a relatively brilliant finish, but a limited amount of carbonation, from the look of it. aroma is what i think an amber ale should be - a nice blend of caramel and happiness bound together.

Ratings

Look: 5 stars

Smell: 4 stars

Figure 3-1: An example of a review with ranking in two categories. The rationale for Look prediction is shown in bold.

opposed to abstractive) rationales. From this perspective, our rationales are simply subsets of the words from the input text that satisfy two key properties. First, the selected words represent short and coherent pieces of text (e.g., phrases) and, second, the selected words must alone suffice for prediction as a substitute of the original text. More concretely, consider the task of multi-aspect sentiment analysis. Figure 3-1 illustrates a product review along with user rating in terms of two categories or aspects. If the model in this case predicts five star rating for color, it should also identify the phrase *"a very pleasant ruby red-amber color"* as the rationale underlying this decision.

In most practical applications, rationale generation must be learned entirely in an unsupervised manner. We therefore assume that our model with rationales is trained on the same data as the original neural models, without access to additional rationale annotations. In other words, target rationales are never provided during training; the intermediate step of rationale generation is guided only by the two desiderata discussed above. Our model is composed of two modular components that we call the generator and the encoder. Our generator specifies a distribution over possible rationales (extracted text) and the encoder maps any such text to task specific target values. They are trained jointly to minimize a cost function that favors short, concise rationales while enforcing that the rationales alone suffice for accurate prediction.

The notion of what counts as a rationale may be ambiguous in some contexts and the task of selecting rationales may therefore be challenging to evaluate. We focus on two domains where ambiguity is minimal (or can be minimized). The first scenario concerns with multi-aspect sentiment analysis exemplified by the beer review corpus [74]. A smaller test set in this corpus identifies, for each aspect, the sentence(s) that relate to this aspect. We can therefore directly evaluate our predictions on the sentence level with the caveat that our model makes selections on a finer level, in terms of words, not complete sentences. The second scenario concerns with the problem of retrieving similar questions. The extracted rationales should capture the main purpose of the questions. We can therefore evaluate the quality of rationales as a compressed proxy for the full text in terms of retrieval performance. Our model achieves high performance on both tasks. For instance, on the sentiment prediction task, our model achieves extraction accuracy of 96%, as compared to 38% and 81% obtained by the bigram SVM and a neural attention baseline.

3.2 Related Work

3.2.1 Developing Interpretable Models

Developing sparse interpretable models is of considerable interest to the broader research community[68, 54]. The need for interpretability is even more pronounced with recent neural models. Efforts in this area include analyzing and visualizing state activation [40, 51, 70], learning sparse interpretable word vectors [29], and linking word vectors to semantic lexicons or word properties [28, 37].

Beyond learning to understand or further constrain the network to be directly interpretable, one can estimate interpretable proxies that approximate the network. Examples include extracting "if-then" rules [106] and decision trees [20] from trained networks. More recently, Ribeiro et al. [87] propose a model-agnostic framework where the proxy model is learned only for the target sample (and its neighborhood) thus ensuring locally valid approximations.

Our work differs from these both in terms of what is meant by an explanation and how they are derived. In our case, an explanation consists of a concise yet sufficient portion of the text where the mechanism of selection is learned jointly with the predictor.

3.2.2 Attention-based Neural Networks

Attention-based models offer another means to explicate the inner workings of neural models [4, 13, 73, 11, 110, 112]. Such models have been successfully applied to many NLP problems, improving both prediction accuracy as well as visualization and interpretability [91, 89, 39]. Xu et al. [111] introduced a stochastic attention mechanism together with a more standard soft attention on image captioning task.

Our rationale extraction can be understood as a type of stochastic attention although architectures and objectives differ. Moreover, we compartmentalize rationale generation from downstream encoding so as to expose knobs to directly control types of rationales that are acceptable, and to facilitate broader modular use in other applications.

3.2.3 Rationale-based Classification

Finally, we contrast our work with rationale-based classification [115, 72, 119] which seek to improve prediction by relying on richer annotations in the form of human-provided rationales. For example, Zaidan et al. [115] include rationales as additional supervision in a SVM classifier, and improve the sentiment prediction performance on movie reviews. Zhang et al. [119] train a sentence-level classifier using rationale annotations and use the trained model for sentence-level weighted average-pooling in a document-level CNN model. In contrast to the prior work, we assume that rationales are never given during training. Instead, the goal is to learn to generate them.

3.3 Model

3.3.1 Extractive Rationale Generation

We formalize here the task of extractive rationale generation and illustrate it in the context of neural models. To this end, consider a typical NLP task where we are provided with a sequence of words as input, namely $\mathbf{x} = \{x_1, \cdots, x_l\}$, where each $x_t \in \mathbb{R}^d$ denotes the vector representation of the i-th word. The learning problem is to map the input sequence $\mathbf{x}$ to a target vector in $\mathbb{R}^m$. For example, in multi-aspect sentiment analysis each coordinate of the target vector represents the response or rating pertaining to the associated aspect. In text retrieval, on the other hand, the target vectors are used to induce similarity assessments between input sequences. Broadly speaking, we can solve the associated learning problem by estimating a complex parameterized mapping $\mathbf{enc}(\mathbf{x})$ from input sequences to target vectors. We call this mapping an *encoder*. The training signal for these vectors is obtained either directly (e.g., multi-sentiment analysis) or via similarities (e.g., text retrieval). The challenge is that a complex neural encoder $\mathbf{enc}(\mathbf{x})$ reveals little about its internal workings and thus offers little in the way of justification for why a particular prediction was made.

In extractive rationale generation, our goal is to select a subset of the input sequence as a *rationale*. In order for the subset to qualify as a rationale it should satisfy two criteria: 1) the selected words should be interpretable and 2) they ought to suffice to reach nearly the same prediction (target vector) as the original input. In other words, a rationale must be short and sufficient. We will assume that a short selection is interpretable and focus on optimizing sufficiency under cardinality constraints.

We encapsulate the selection of words as a *rationale generator* which is another parameterized mapping $\mathbf{gen}(\mathbf{x})$ from input sequences to shorter sequences of words. Thus $\mathbf{gen}(\mathbf{x})$ must include only a few words and $\mathbf{enc}(\mathbf{gen}(\mathbf{x}))$ should result in nearly the same target vector as the original input passed through the encoder or $\mathbf{enc}(\mathbf{x})$. We can think of the generator as a tagging model where each word in the input receives a binary tag pertaining to whether it is selected to be included in the rationale. In our case, the generator is probabilistic and specifies a distribution over possible selections.

Figure 3-2 illustrates our generator component.

The rationale generation task is entirely unsupervised in the sense that we assume no explicit annotations about which words should be included in the rationale. Put another way, the rationale is introduced as a latent variable, a constraint that guides how to interpret the input sequence. The encoder and generator are trained jointly, in an end-to-end fashion so as to function well together.

3.3.2 Encoder and Generator

We use multi-aspect sentiment prediction as a guiding example to instantiate the two key components – the encoder and the generator. The framework itself generalizes to other tasks.

Encoder enc(·): Given a training instance $(\mathbf{x}, \mathbf{y})$ where $\mathbf{x} = \{\mathbf{x}_t\}_{t=1}^{l}$ is the input text sequence of length l and $\mathbf{y} \in [0, 1]^m$ is the target m-dimensional sentiment vector, the neural encoder predicts $\tilde{\mathbf{y}} = \mathbf{enc}(\mathbf{x})$. If trained on its own, the encoder would aim to minimize the discrepancy between the predicted sentiment vector $\tilde{\mathbf{y}}$ and the gold target vector $\mathbf{y}$. We will use the squared error (i.e. L_2 distance) as the loss function,

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = \|\tilde{\mathbf{y}} - \mathbf{y}\|_2^2 = \|\mathbf{enc}(\mathbf{x}) - \mathbf{y}\|_2^2$$

The encoder could be realized in many ways such as a recurrent neural network. For example, let $\mathbf{h}[t] = f_e(\mathbf{x}_t, \mathbf{h}[t-1])$ denote a parameterized recurrent unit mapping input word $\mathbf{x}_t$ and previous state $\mathbf{h}[t-1]$ to next state $\mathbf{h}[t]$. The target vector is then generated on the basis of the final state reached by the recurrent unit after processing all the words in the input sequence. Specifically,

$$\begin{aligned} \mathbf{h}[t] &= f_e(\mathbf{x}_t, \mathbf{h}[t-1]), \qquad t = 1, \ldots, l \\ \tilde{\mathbf{y}} &= \sigma_e(\mathbf{W}^e \mathbf{h}[l] + \mathbf{b}^e) \end{aligned}$$

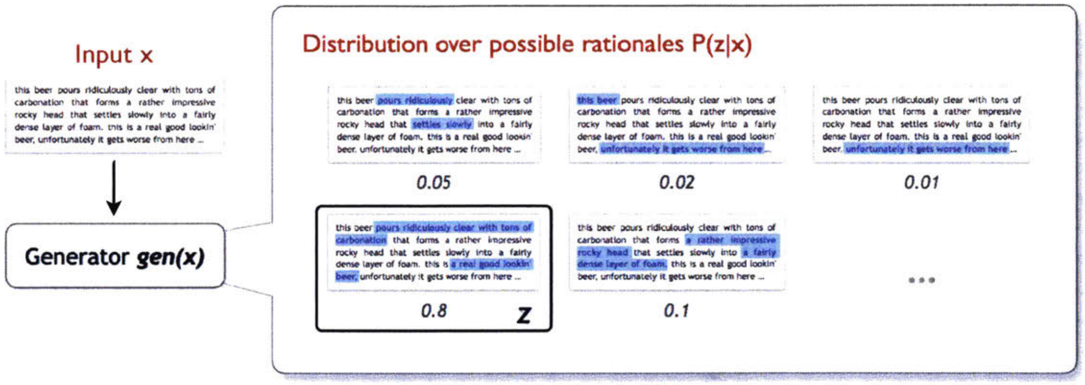

Input x
this beer pours ridiculously clear with tons of carbonation that forms a rather impressive rocky head that settles slowly into a fairly dense layer of foam. this is a real good lookin' beer, unfortunately it gets worse from here ...
Generator gen(x)
Distribution over possible rationales P(z|x)
this beer pours ridiculously clear with tons of carbonation that forms a rather impressive rocky head that settles slowly into a fairly dense layer of foam. this is a real good lookin' beer. unfortunately it gets worse from here ...
0.05
this beer pours ridiculously clear with tons of carbonation that forms a rather impressive rocky head that settles slowly into a fairly dense layer of foam. this is a real good lookin' beer, unfortunately it gets worse from here ...
0.02
this beer pours ridiculously clear with tons of carbonation that forms a rather impressive rocky head that settles slowly into a fairly dense layer of foam. this is a real good lookin' beer. unfortunately it gets worse from here ...
0.01
this beer pours ridiculously clear with tons of carbonation that forms a rather impressive rocky head that settles slowly into a fairly dense layer of foam. this is a real good lookin' beer, unfortunately it gets worse from here ...
0.8
z
this beer pours ridiculously clear with tons of carbonation that forms a rather impressive rocky head that settles slowly into a fairly dense layer of foam. this is a real good lookin' beer, unfortunately it gets worse from here ...
0.1
...

Figure 3-2: An illustration of the generator component. The generator specifies the distribution of rationales for a given input document.

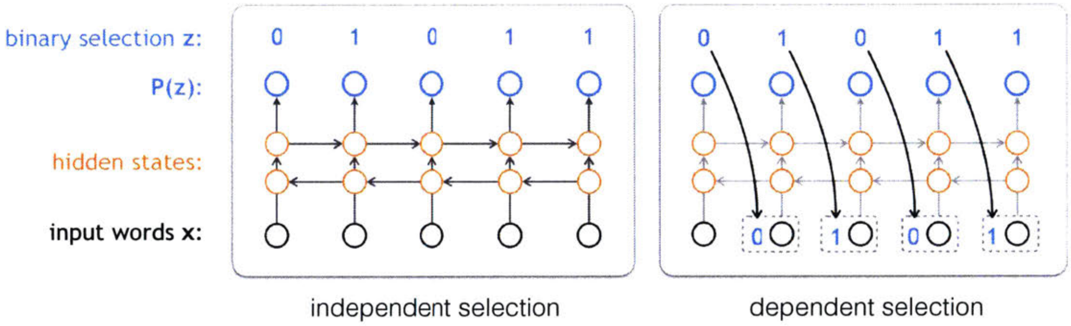


Figure 3-3: Two possible neural implementations of the generator component.

Generator $\mathbf{gen}(\cdot)$: The rationale generator extracts a subset of text from the original input $\mathbf{x}$ to function as an interpretable summary. Thus the rationale for a given sequence $\mathbf{x}$ can be equivalently defined in terms of binary variables $\mathbf{z} = \{z_1, \cdots, z_l\}$ where each $z_t \in \{0, 1\}$ indicates whether word $\mathbf{x}_t$ is selected or not (see Figure 3-2). From here on, we will use $\mathbf{z}$ to specify the binary selections and thus $(\mathbf{z}, \mathbf{x})$ is the actual rationale generated (selections, input). We will use generator $\mathbf{gen}(\mathbf{x})$ as synonymous with a probability distribution over binary selections, i.e., $\mathbf{z} \sim \mathbf{gen}(\mathbf{x}) \equiv p(\mathbf{z}|\mathbf{x})$ where the length of $\mathbf{z}$ varies with the input $\mathbf{x}$.

In a simple generator, the probability that the t^{th} word is selected can be assumed to be conditionally independent from other selections given the input $\mathbf{x}$. That is, the joint probability $p(\mathbf{z}|\mathbf{x})$ factors according to

$$p(\mathbf{z}|\mathbf{x}) = \prod_{t=1}^{l} p(z_t|\mathbf{x}) \qquad \text{(independent selection)}$$

The component distributions $p(z_t|\mathbf{x})$ can be modeled using a shared bi-directional recurrent neural network. Specifically, let $\overrightarrow{f}()$ and $\overleftarrow{f}()$ be the forward and backward recurrent unit, respectively, then

$$\begin{aligned}\overrightarrow{\mathbf{h}}[t] &= \overrightarrow{f}\left(\mathbf{x}_t, \overrightarrow{\mathbf{h}}[t-1]\right) \\ \overleftarrow{\mathbf{h}}[t] &= \overleftarrow{f}\left(\mathbf{x}_t, \overleftarrow{\mathbf{h}}[t-1]\right) \\ p(z_t|\mathbf{x}) &= \sigma_z\left(\mathbf{W}^z\left[\overrightarrow{\mathbf{h}}[t]; \overleftarrow{\mathbf{h}}[t]\right] + \mathbf{b}^z\right)\end{aligned}$$

Independent but context dependent selection of words is often sufficient. However, the model is unable to select phrases or refrain from selecting the same word again if already chosen. To this end, we also introduce a dependent selection of words,

$$p(\mathbf{z}|\mathbf{x}) = \prod_{t=1}^{l} p(z_t|\mathbf{x}, z_1 \cdots z_{t-1})$$

which can be also expressed as a recurrent neural network. To this end, we introduce another hidden state $\mathbf{s}[t]$ whose role is to couple the selections. For example,

$$\begin{aligned}p(z_t|\mathbf{x}, \mathbf{z}_{1,t-1}) &= \sigma_z\left(\mathbf{W}^z\left[\overrightarrow{\mathbf{h}}[t]; \overleftarrow{\mathbf{h}}[t]; \mathbf{s}[t-1]\right] + \mathbf{b}^z\right) \\ \mathbf{s}[t] &= f_z\left(\left[\overrightarrow{\mathbf{h}}[t]; \overleftarrow{\mathbf{h}}[t]; z_t\right], \mathbf{s}[t-1]\right)\end{aligned}$$

Figure 3-3 illustrates the independent and dependent implementations of selections.

Joint objective: A rationale in our definition corresponds to the selected words, i.e., $\{\mathbf{x}_k|\mathbf{z}_k = 1\}$. We will use $(\mathbf{z}, \mathbf{x})$ as the shorthand for this rationale and, thus, $\mathbf{enc}(\mathbf{z}, \mathbf{x})$ refers to the target vector obtained by applying the encoder to the rationale as the input. Our goal here is to formalize how the rationale can be made short and meaningful yet function well in conjunction with the encoder. Our generator and encoder are learned jointly to interact well but they are treated as independent units for modularity.

The generator is guided in two ways during learning. First, the rationale that it produces must suffice as a replacement for the input text. In other words, the target vector (sentiment) arising from the rationale should be close to the gold sentiment. The corresponding loss function is given by

$$\mathcal{L}(\mathbf{z}, \mathbf{x}, \mathbf{y}) = \|\mathbf{enc}(\mathbf{z}, \mathbf{x}) - \mathbf{y}\|_2^2$$

Note that the loss function depends directly (parametrically) on the encoder but only indirectly on the generator via the sampled selection.

Second, we must guide the generator to realize short and coherent rationales. It should select only a few words and those selections should form phrases (consecutive words) rather than represent isolated, disconnected words. We therefore introduce an additional regularizer over the selections

$$\Omega(\mathbf{z}) = \lambda_1 \|\mathbf{z}\| + \lambda_2 \sum_t |z_t - z_{t-1}|$$

where the first term penalizes the number of selections while the second one discourages transitions (encourages continuity of selections). Note that this regularizer also depends on the generator only indirectly via the selected rationale. This is because it is easier to assess the rationale once produced rather than directly guide how it is obtained.

Our final cost is the combination of the two, $\text{cost}(\mathbf{z}, \mathbf{x}, \mathbf{y}) = \mathcal{L}(\mathbf{z}, \mathbf{x}, \mathbf{y}) + \Omega(\mathbf{z})$. Since the selections are not provided during training, we minimize the expected cost:

$$\min_{\theta_e, \theta_g} \sum_{(\mathbf{x}, \mathbf{y}) \in D} \mathbb{E}_{\mathbf{z} \sim \mathbf{gen}(\mathbf{x})} \left[\text{cost}(\mathbf{z}, \mathbf{x}, \mathbf{y})\right]$$

where θ_e and θ_g denote the set of parameters of the encoder and generator, respectively, and D is the collection of training instances. Our joint objective encourages the generator to compress the input text into coherent summaries that work well with the associated encoder it is trained with.

Minimizing the expected cost is challenging since it involves summing over all the possible choices of rationales $\mathbf{z}$. This summation could potentially be made feasible with additional restrictive assumptions about the generator and encoder. However, we assume only that it is possible to efficiently sample from the generator.

Doubly stochastic gradient We now derive a sampled approximation to the gradient of the expected cost objective. This sampled approximation is obtained separately for each input text $\mathbf{x}$ so as to work well with an overall stochastic gradient method. Consider therefore a training pair $(\mathbf{x}, \mathbf{y})$. For the parameters of the generator θ_g, the gradient is

$$\begin{aligned} & \frac{\partial \mathbb{E}_{\mathbf{z} \sim \mathbf{gen}(\mathbf{x})} \left[\text{cost}(\mathbf{z}, \mathbf{x}, \mathbf{y}) \right]}{\partial \theta_g} \\ = \quad & \sum_{\mathbf{z}} \text{cost}(\mathbf{z}, \mathbf{x}, \mathbf{y}) \cdot \frac{\partial p(\mathbf{z}|\mathbf{x})}{\partial \theta_g} \\ = \quad & \sum_{\mathbf{z}} \text{cost}(\mathbf{z}, \mathbf{x}, \mathbf{y}) \cdot \frac{\partial p(\mathbf{z}|\mathbf{x})}{\partial \theta_g} \cdot \frac{p(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})} \end{aligned}$$

Using the fact $(\log f(\theta))' = f'(\theta)/f(\theta)$, we can rewrite the gradient as,

$$\begin{aligned} & \sum_{\mathbf{z}} \text{cost}(\mathbf{z}, \mathbf{x}, \mathbf{y}) \cdot \frac{\partial p(\mathbf{z}|\mathbf{x})}{\partial \theta_g} \cdot \frac{p(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})} \\ = \quad & \sum_{\mathbf{z}} \text{cost}(\mathbf{z}, \mathbf{x}, \mathbf{y}) \cdot \frac{\partial \log p(\mathbf{z}|\mathbf{x})}{\partial \theta_g} \cdot p(\mathbf{z}|\mathbf{x}) \\ = \quad & \mathbb{E}_{z \sim \mathbf{gen}(\mathbf{x})} \left[\text{cost}(\mathbf{z}, \mathbf{x}, \mathbf{y}) \frac{\partial \log p(\mathbf{z}|\mathbf{x})}{\partial \theta_g} \right] \end{aligned}$$

The last term is the expected gradient where the expectation is taken with respect to the generator distribution over rationales $\mathbf{z}$. Therefore, we can simply sample a few rationales $\mathbf{z}$ from the generator $\mathbf{gen}(\mathbf{x})$ and use the resulting average gradient in an overall stochastic gradient method. A sampled approximation to the gradient with respect to the encoder parameters θ_e can be derived similarly,

$$\begin{aligned}
&\frac{\partial \mathbb{E}_{\mathbf{z}\sim\mathbf{gen}(\mathbf{x})}\left[\text{cost}(\mathbf{z},\mathbf{x},\mathbf{y})\right]}{\partial \theta_e} \\
&\quad = \sum_{\mathbf{z}} \frac{\partial \text{cost}(\mathbf{z},\mathbf{x},\mathbf{y})}{\partial \theta_e} \cdot p(\mathbf{z}|\mathbf{x}) \\
&\quad = \mathbb{E}_{z\sim\mathbf{gen}(\mathbf{x})}\left[\frac{\partial \text{cost}(\mathbf{z},\mathbf{x},\mathbf{y})}{\partial \theta_e}\right]
\end{aligned}$$

Choice of recurrent unit We employ the recurrent convolution (RCNN) proposed in the previous chapter. Specifically, we use the adaptive version with additive and normalized 2-gram feature mapping,

$$\begin{aligned}
\mathbf{c}_1[t] &= \lambda_t \odot \mathbf{c}_1[t-1] + (1-\lambda_t) \odot (\mathbf{W}_1 \mathbf{x}_t) \\
\mathbf{c}_2[t] &= \lambda_t \odot \mathbf{c}_2[t-1] + (1-\lambda_t) \odot (\mathbf{c}_1[t-1] + \mathbf{W}_2 \mathbf{x}_t) \\
\mathbf{h}[t] &= \tanh(\mathbf{c}_2[t] + \mathbf{b}) \\
\lambda_t &= \sigma(\mathbf{W}^\lambda \mathbf{x}_t + \mathbf{U}^\lambda \mathbf{h}[t-1] + \mathbf{b}^\lambda)
\end{aligned}$$

As shown in the evaluation in the previous chapter, the RCNN component works remarkably well in classification and retrieval applications compared to other alternatives such CNNs and LSTMs. We use it for all the recurrent units introduced in our model.

3.4 Applications and Evaluations

We evaluate the proposed joint model on two NLP applications: (1) multi-aspect sentiment analysis on product reviews and (2) similar text retrieval on AskUbuntu question answering forum.

3.4.1 Multi-aspect Sentiment Analysis

Dataset We use the BeerAdvocate[1] review dataset used in prior work [74].[2] This dataset contains 1.5 million reviews written by the website users. The reviews are naturally multi-aspect – each of them contains multiple sentences describing the *overall* impression or one particular aspect of a beer, including *appearance*, *smell* (aroma), *palate* and the *taste*. In addition to the written text, the reviewer provides the ratings (on a scale of 0 to 5 stars) for each aspect as well as an overall rating. The ratings can be fractional (e.g. 3.5 stars), so we normalize the scores to $[0, 1]$ and use them as the (only) supervision for regression.

McAuley et al. [74] also provided sentence-level annotations on around 1,000 reviews. Each sentence is annotated with one (or multiple) aspect label, indicating what aspect this sentence covers. We use this set as our test set to evaluate the precision of words in the extracted rationales.

Table 3.1 shows several statistics of the beer review dataset. The sentiment correlation between any pair of aspects (and the overall score) is quite high, getting 63.5% on average and a maximum of 79.1% (between the *taste* and *overall* score). If directly training the model on this set, the model can be confused due to such strong correlation. We therefore perform a preprocessing step, picking "less correlated" examples from the dataset.[3] This gives us a de-correlated subset for each aspect, each containing about 80k to 90k reviews. We use 10k as the development set. We focus on three aspects since the fourth aspect *taste* still gets $> 50\%$ correlation with the

[1] `www.beeradvocate.com`

[2] `http://snap.stanford.edu/data/web-BeerAdvocate.html`

[3] Specifically, for each aspect we train a simple linear regression model to predict the rating of this aspect given the ratings of the other four aspects. We then keep picking reviews with largest prediction error until the sentiment correlation in the selected subset increases dramatically.

Number of reviews	1580k
Avg length of review	144.9
Avg correlation between aspects	63.5%
Max correlation between two aspects	79.1%
Number of annotated reviews	994

Table 3.1: Statistics of the beer review dataset.

overall sentiment.

Sentiment Prediction Before training the joint model, it is worth assessing the neural encoder separately to check how accurately the neural network predicts the sentiment. To this end, we compare neural encoders with bigram SVM model, training medium and large SVM models using 260k and all 1580k reviews respectively. As shown in Table 3.2, the recurrent neural network models outperform the SVM model for sentiment prediction and also require less training data to achieve the performance. The LSTM and RCNN units obtain similar test error, getting 0.0094 and 0.0087 mean squared error respectively. The RCNN unit performs slightly better and uses less parameters. Based on the results, we choose the RCNN encoder network with 2 stacking layers and 200 hidden states.

To train the joint model, we also use RCNN unit with 200 states as the forward and backward recurrent unit for the generator **gen**(). The dependent generator has one additional recurrent layer. For this layer we use 30 states so the dependent version still has a number of parameters comparable to the independent version. The two versions of the generator have 358k and 323k parameters respectively.

Figure 3-4 shows the performance of our joint dependent model when trained to predict the sentiment of all aspects. We vary the regularization λ_1 and λ_2 to show various runs that extract different amount of text as rationales. Our joint model gets performance close to the best encoder run (with full text) when few words are extracted.

	D	d	l	$\|\theta\|$	MSE
SVM	260k	-	-	2.5M	0.0154
SVM	1580k	-	-	7.3M	0.0100
LSTM	260k	200	2	644k	0.0094
RCNN	260k	200	2	323k	**0.0087**

Table 3.2: Comparing neural encoders with bigram SVM model. MSE is the mean squared error on the test set. D is the amount of data used for training and development. d stands for the hidden dimension, l denotes the depth of network and $|\theta|$ denotes the number of parameters (number of features for SVM).

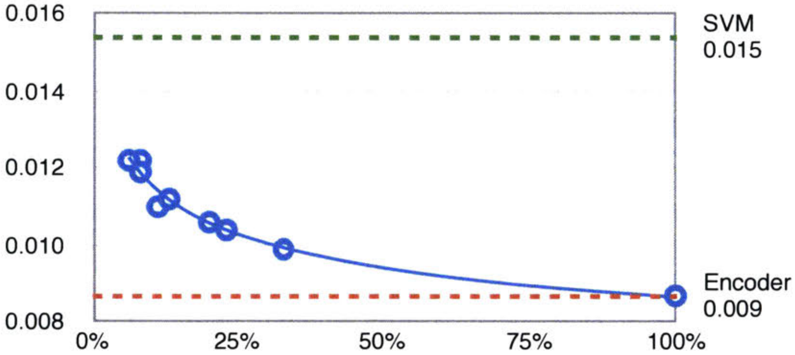


Figure 3-4: Mean squared error of all aspects on the test set (y-axis) when various percentages of text are extracted as rationales (x-axis). 220k training data is used. Our model using only the extracted rationales is getting close performance to the neural model using full text as the input.

Method	Appearance		Smell		Palate	
	%prec	%sel	%prec	%sel	%prec	%sel
SVM	38.3	13	21.6	7	24.9	7
Attention model	80.6	13	88.4	7	65.3	7
Generator (independent)	94.8	13	93.8	7	79.3	7
Generator (dependent)	96.3	14	95.1	7	80.2	7

Table 3.3: Precision of selected rationales for the first three aspects. The precision is evaluated based on whether the selected words are in the sentences describing the target aspect, based on the sentence-level annotations. Best training epochs are selected based on the objective value on the development set (no sentence annotation is used).

Rationale Selection To evaluate the supporting rationales for each aspect, we train the joint encoder-generator model on each de-correlated subset. We set the cardinality regularization λ_1 between values $\{2e-4, 3e-4, 4e-4\}$ so the extracted rationale texts are neither too long nor too short. For simplicity, we set $\lambda_2 = 2\lambda_1$ to encourage local coherency of the extraction.

For comparison we use the bigram SVM model and implement an attention-based neural network model. The SVM model successively extracts unigram or bigram (from the test reviews) with the highest feature. The attention-based model learns a normalized attention vector of the input tokens (using similarly the forward and backward RNNs), then the model averages over the encoder states accordingly to the attention, and feed the averaged vector to the output layer. Similar to the SVM model, the attention-based model can selects words based on their attention weights.

Table 3.3 presents the precision of the extracted rationales calculated based on sentence-level aspect annotations. The λ_1 regularization hyper-parameter is tuned so the two versions of our model extract similar number of words as rationales. The SVM and attention-based model are constrained similarly for comparison. Figure 3-5 further shows the precision when different amounts of text are extracted. Again, for our model this corresponds to changing the λ_1 regularization. As shown in the table and the figure, our encoder-generator networks extract text pieces describing the target aspect with high precision, ranging from 80% to 96% across the three aspects appearance, smell and palate. The SVM baseline performs poorly, achieving around 30% accuracy. The attention-based model achieves reasonable but worse performance than the rationale generator, suggesting the potential of directly modeling rationales as explicit extraction.

Figure 3-6 shows the learning curves of our model for the smell aspect. In the early training epochs, both the independent and (recurrent) dependent selection models fail to produce good rationales, getting low precision as a result. After a few epochs of exploration however, the models start to achieve high accuracy. We observe that the dependent version learns more quickly in general, but both versions obtain close results in the end.

Finally we conduct a qualitative case study on the extracted rationales. Figure 3-7 presents several reviews, with highlighted rationales predicted by the model. Our rationale generator identifies key phrases or adjectives that indicate the sentiment of a particular aspect.

3.4.2 Similar Text Retrieval on QA Forum

Dataset For our second application, we use the real-world AskUbuntu[4] dataset used in recent work [25, 67]. This set contains a set of 167k unique questions (each consisting a question title and a body) and 16k user-identified similar question pairs. Following previous work, this data is used to train the neural encoder that learns the vector representation of the input question, optimizing the cosine distance (i.e. cosine similarity) between similar questions against random non-similar ones. We use the "one-versus-all" hinge loss (i.e. positive versus other negatives) for the encoder, similar to [67]. During development and testing, the model is used to score 20 candidate questions given each query question, and a total of 400×20 query-candidate question pairs are annotated for evaluation[5].

[4] `askubuntu.com`

[5] `https://github.com/taolei87/askubuntu`

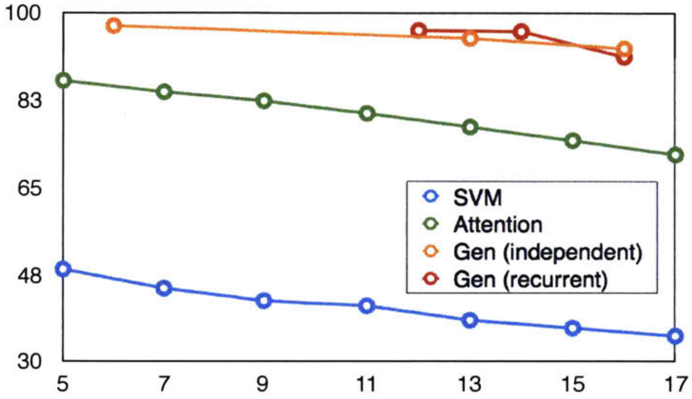


Figure 3-5: Precision (y-axis) when various percentages of text are extracted as rationales (x-axis) for the appearance aspect.

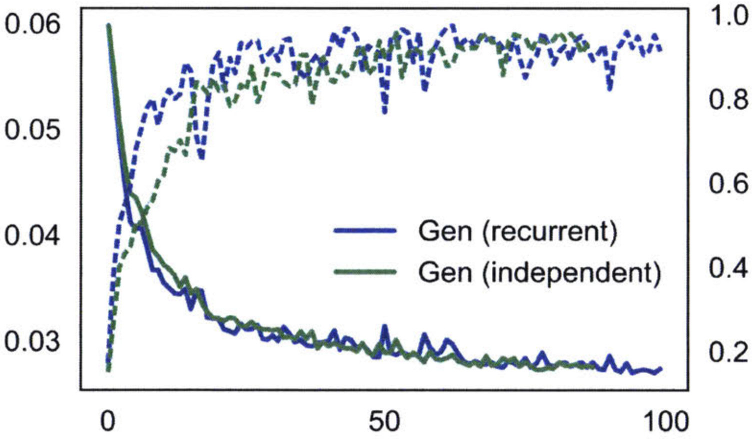
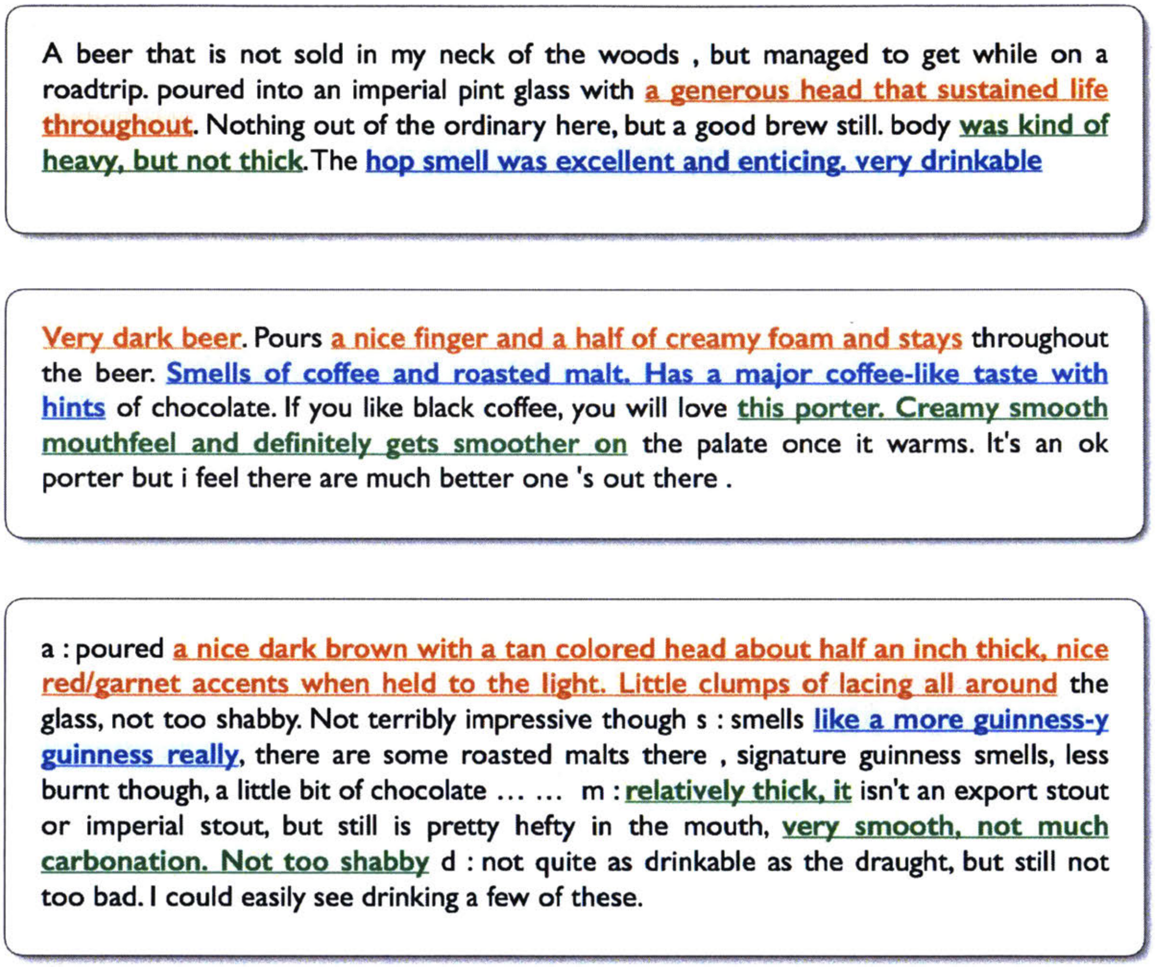


Figure 3-6: Learning curves of the optimized cost function on the development set and the precision of rationales on the test set. The smell (aroma) aspect is the target aspect.



A beer that is not sold in my neck of the woods , but managed to get while on a roadtrip. poured into an imperial pint glass with a generous head that sustained life throughout. Nothing out of the ordinary here, but a good brew still. body was kind of heavy, but not thick.The hop smell was excellent and enticing. very drinkable

Very dark beer. Pours a nice finger and a half of creamy foam and stays throughout the beer. Smells of coffee and roasted malt. Has a major coffee-like taste with hints of chocolate. If you like black coffee, you will love this porter. Creamy smooth mouthfeel and definitely gets smoother on the palate once it warms. It's an ok porter but i feel there are much better one 's out there .

a : poured a nice dark brown with a tan colored head about half an inch thick, nice red/garnet accents when held to the light. Little clumps of lacing all around the glass, not too shabby. Not terribly impressive though s : smells like a more guinness-y guinness really, there are some roasted malts there , signature guinness smells, less burnt though, a little bit of chocolate ... ... m : relatively thick, it isn't an export stout or imperial stout, but still is pretty hefty in the mouth, very smooth, not much carbonation. Not too shabby d : not quite as drinkable as the draught, but still not too bad. I could easily see drinking a few of these.

Figure 3-7: Examples of extracted rationales indicating the sentiments of various aspects. The extracted texts for appearance, smell and palate are shown in orange, blue and green color respectively. The last example is shortened for space.

Task/Evaluation Setup The question descriptions are often long and fraught with irrelevant details. In this set-up, a fraction of the original question text should be sufficient to represent its content, and be used for retrieving similar questions. Therefore, we will evaluate rationales based on the accuracy of the question retrieval task, assuming that better rationales achieve higher performance. To put this performance in context, we also report the accuracy when full body of a question is used, as well as titles alone. The latter constitutes an upper bound on the model performance as in this dataset titles provide short, informative summaries of the question content. We evaluate the rationales using the mean average precision (MAP) of retrieval.

Results Table 3.4 presents the results of our rationale model. We explore a range of hyper-parameter values[6]. We include two runs for each version. The first one achieves the highest MAP on the development set, The second run is selected to compare the models when they use roughly 10% of question text (7 words on average). We also show the results of different runs in Figure 3-8. The rationales achieve the MAP up to 56.5%, getting close to using the titles. The models also outperform the baseline of using the noisy question bodies, indicating the the models' capacity of extracting short but important fragments.

Figure 3-9 shows the rationales for several questions in the AskUbuntu domain, using the recurrent version with around 10% extraction. Interestingly, the model does not always select words from the question title. The reasons are that the question body can contain the same or even complementary information useful for retrieval. Indeed, some rationale fragments shown in the figure are error messages, which are typically not in the titles but very useful to identify similar questions.

[6] $\lambda_1 \in \{.008, .01, .012, .015\}$, $\lambda_2 = \{0, \lambda_1, 2\lambda_1\}$, dropout $\in \{0.1, 0.2\}$

	MAP (dev)	MAP (test)	%words
Full title	56.5	60.0	10.1
Full body	54.2	53.0	89.9
Independent	55.7	53.6	9.7
	56.3	52.6	19.7
Dependent	56.1	54.6	11.6
	56.5	55.6	32.8

Table 3.4: Comparison between rationale models (middle and bottom rows) and the baselines using full title or body (top row).

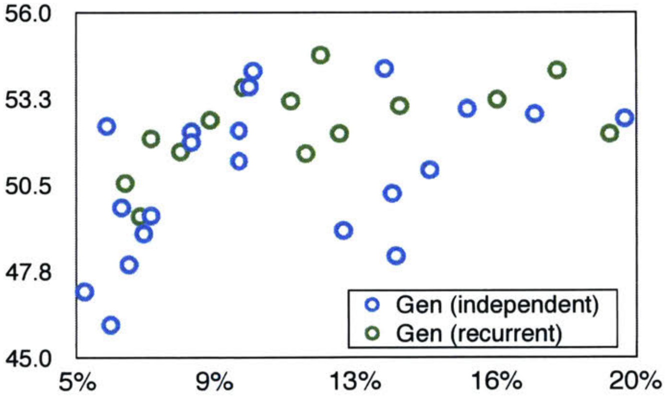
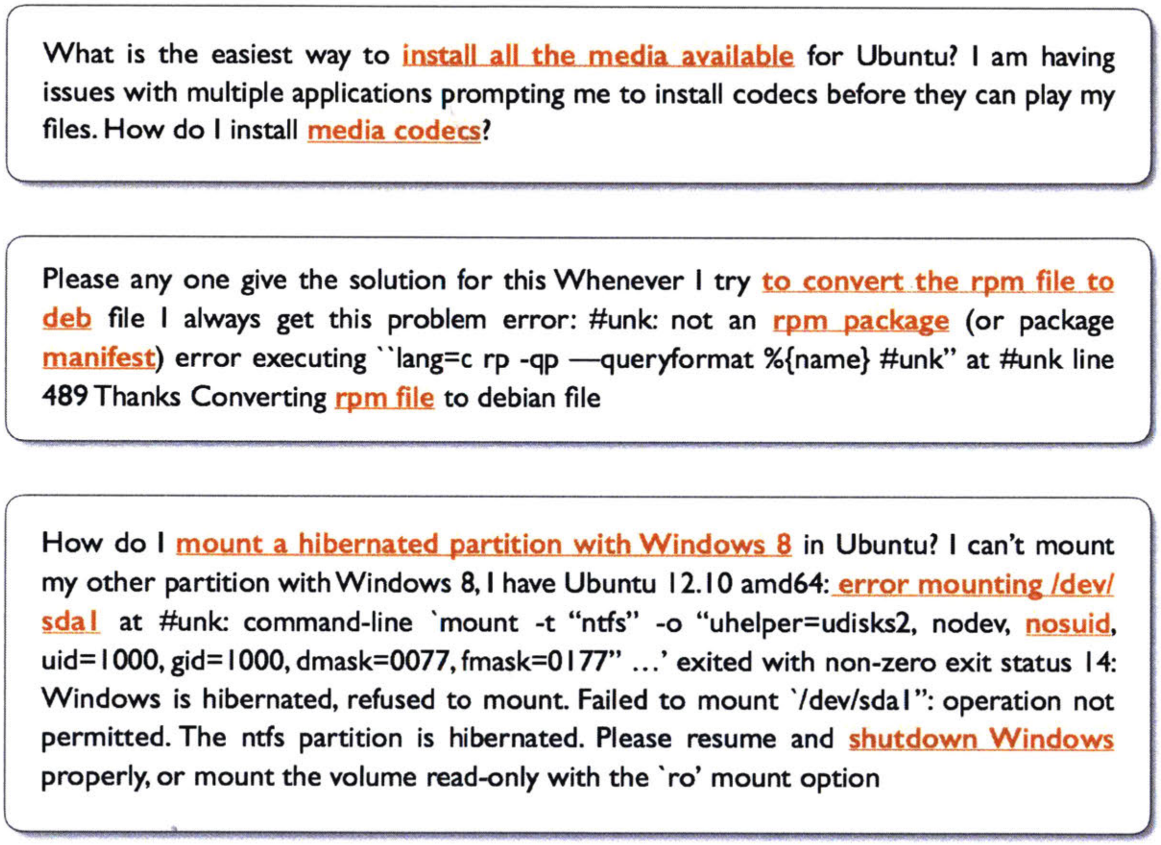


Figure 3-8: Retrieval MAP on the test set when various percentages of the texts are chosen as rationales. Data points correspond to models trained with different hyper-parameters.



What is the easiest way to install all the media available for Ubuntu? I am having issues with multiple applications prompting me to install codecs before they can play my files. How do I install media codecs?

Please any one give the solution for this Whenever I try to convert the rpm file to deb file I always get this problem error: #unk: not an rpm package (or package manifest) error executing ``lang=c rp -qp —queryformat %{name} #unk" at #unk line 489 Thanks Converting rpm file to debian file

How do I mount a hibernated partition with Windows 8 in Ubuntu? I can't mount my other partition with Windows 8, I have Ubuntu 12.10 amd64: error mounting /dev/sda1 at #unk: command-line `mount -t "ntfs" -o "uhelper=udisks2, nodev, nosuid, uid=1000, gid=1000, dmask=0077, fmask=0177" ...' exited with non-zero exit status 14: Windows is hibernated, refused to mount. Failed to mount `/dev/sda1": operation not permitted. The ntfs partition is hibernated. Please resume and shutdown Windows properly, or mount the volume read-only with the `ro' mount option

Figure 3-9: Examples of extracted rationales of questions in the AskUbuntu domain.

3.5 Conclusions and Discussions

In this chapter, we proposed a novel modular neural framework to automatically generate concise yet sufficient text fragments to justify predictions made by neural networks. We demonstrated that our encoder-generator framework, trained in an end-to-end manner, gives rise to quality rationales in the absence of any explicit rationale annotations. The approach could be modified or extended in various ways to other applications or types of data.

Choices of enc(·) and gen(·). The encoder and generator can be realized in numerous ways without changing the broader algorithm. For instance, we could use a convolutional network [55, 50], deep averaging network [45, 47] or a boosting classifier as the encoder. When rationales can be expected to conform to repeated stereotypical patterns in the text, a simpler encoder consistent with this bias can work better. We emphasize that, in this paper, rationales are flexible explanations that may vary substantially from instance to another. On the generator side, many additional constraints could be imposed to further guide acceptable rationales.

Dealing with Search Space. Our training method employs a REINFORCE-style algorithm [109] where the gradient with respect to the parameters is estimated by sampling possible rationales. Additional constraints on the generator output can be helpful in alleviating problems of exploring potentially a large space of possible rationales in terms of their interaction with the encoder. We could also apply variance reduction techniques to increase stability of stochastic training (cf. [108, 79, 3, 111]).

4
Conclusions

In this thesis, we have proposed several neural components as well as theoretical and empirical studies for interpretable deep learning in the context of NLP. In particular, we have studied a class of neural operation (or layer) for sequence modeling. Broadly speaking, this operation is a generalization of convolutional architectures and recurrent architectures (with gating) which have been shown to work well for NLP. We argue that the effectiveness of such class for NLP can be seen and justified by relating it to sequence kernels – a natural class of functions for measuring sequence similarity. We hope a deeper understanding would lead to empirical success. Indeed, we have achieved state-of-the-art or competitive results in various NLP applications by exploring neural architectures guided by the theoretical justifications.

Furthermore, we have presented a framework for generating human-readable rationales, e.g. text pieces in the context of NLP, to justify model's prediction. For tasks such as classification, the model framework can learn rationales jointly with classification predictions with only classification label annotations. We have derived a REINFORCE-style learning algorithm and successfully demonstrated its effectiveness in several NLP applications. The proposed framework opens a way for ordinary users to verify a neural model's decision and to communicate with the model.

Future Work

Several paths of research arises from the work presented in this thesis. We briefly discuss a few of them below.

- **Structured Components beyond Sequence** The neural operation in Chapter 2 is designed for sequential data. In applications such as parsing, translation or reasoning, it may be necessary to investigate components and operations that encode more complicated structures, such as trees or graphs. We have presented a way for encoding efficient sequence-based computation into the neural structures, and hope to shed some light on this direction.

- **Rationale Framework Extensions** The rationale framework is presented in the context of text classification. At a high-level, the framework can be applied in other scenarios, such as image classification and object detection to provide supporting image regions without explicit human annotations during training. Another challenging problem is to model interactions of rationales – when the decision is made by reasoning between two or several input pieces. On the technical aspect, the learning is driven by sampling / exploring the rationale space, and hence it would be also necessary to explore learning techniques such as variance reduction and model distillation to improve learning stability.

A

Derivations and Proofs

A.1 Derivation of the Dynamic Programming Method

Consider the feature extraction operations defined in Table 2.1 of Section 2.3. We derive the dynamic programming method for computing $\mathbf{c}_n[t]$ and $\mathbf{z}_n[t]$. As a running example, let's use the multiplicative, un-normalized version,

$$\mathbf{c}_n[t] = \sum_{1 \le i_1 < i_2 < \cdots < i_n \le t} \left(\mathbf{W}^{(1)}\mathbf{x}_{i_1} \odot \cdots \odot \mathbf{W}^{(n)}\mathbf{x}_{i_n}\right) \cdot \lambda^{t-i_1-n+1}$$

$$\mathbf{z}_n[t] = \sum_{1 \le i_1 < i_2 < \cdots < i_n = t} \left(\mathbf{W}^{(1)}\mathbf{x}_{i_1} \odot \cdots \odot \mathbf{W}^{(n)}\mathbf{x}_{i_n}\right) \cdot \lambda^{t-i_1-n+1}$$

The corresponding DP equations are,

$$n = 1: \qquad \mathbf{c}_1[t] = \lambda \cdot \mathbf{c}_1[t-1] + \mathbf{W}^{(1)}\mathbf{x}_t$$

$$\mathbf{z}_1[t] = \mathbf{W}^{(1)}\mathbf{x}_t$$

$$n > 1: \qquad \mathbf{c}_n[t] = \lambda \cdot \mathbf{c}_n[t-1] + \left(\mathbf{c}_{n-1}[t-1] \odot \mathbf{W}^{(n)}\mathbf{x}_t\right)$$

$$\mathbf{z}_n[t] = \mathbf{c}_{n-1}[t-1] \odot \mathbf{W}^{(n)}\mathbf{x}_t$$

We now prove the equivalence by induction.

Proof: When $n = 1$, $\mathbf{z}_1[t] = \mathbf{W}^{(1)}\mathbf{x}_t$ by definition. For $\mathbf{c}_1[t]$ we have,

$$\begin{aligned}
\mathbf{c}_1[t] &= \sum_{1\leq i_1 \leq t} \left(\mathbf{W}^{(1)}\mathbf{x}_{i_1}\right) \cdot \lambda^{t-i_1} \\
&= \underbrace{\left(\sum_{1\leq i_1\leq t-1} \left(\mathbf{W}^{(1)}\mathbf{x}_{i_1}\right) \cdot \lambda^{t-1-i_1} \right)}_{\mathbf{c}_1[t-1]} \cdot\lambda \; + \; \mathbf{W}^{(1)}\mathbf{x}_t \\
&= \mathbf{c}_1[t-1] \cdot \lambda \; + \; \mathbf{W}^{(1)}\mathbf{x}_t
\end{aligned}$$

When $n = k > 1$, suppose the DP equations hold for $1, \cdots, k-1$, we have,

$$\begin{aligned}
\mathbf{z}_n[t] &= \sum_{1\leq i_1<\cdots<i_n=t} \left(\mathbf{W}^{(1)}\mathbf{x}_{i_1} \odot \cdots \odot \mathbf{W}^{(n)}\mathbf{x}_{i_n}\right) \cdot \lambda^{t-i_1-n+1} \\
&= \underbrace{\left(\sum_{1\leq i_1<\cdots<i_{n-1}<=t-1} \left(\mathbf{W}^{(1)}\mathbf{x}_{i_1} \odot \cdots \odot \mathbf{W}^{(n-1)}\mathbf{x}_{i_{n-1}}\right) \cdot \lambda^{t-i_1-n+1} \right)}_{\mathbf{c}_{n-1}[t-1]} \odot \mathbf{W}^{(n)}\mathbf{x}_t \\
&= \mathbf{c}_{n-1}[t-1] \odot \mathbf{W}^{(n)}\mathbf{x}_t
\end{aligned}$$

$$\begin{aligned}
\mathbf{c}_n[t] &= \sum_{1\leq i_1<i_2<\cdots<i_n\leq t} \left(\mathbf{W}^{(1)}\mathbf{x}_{i_1} \odot \cdots \odot \mathbf{W}^{(n)}\mathbf{x}_{i_n}\right) \cdot \lambda^{t-i_1-n+1} \\
&= \underbrace{\left(\sum_{1\leq i_1<i_2<\cdots<i_n\leq t-1} \left(\mathbf{W}^{(1)}\mathbf{x}_{i_1} \odot \cdots \odot \mathbf{W}^{(n)}\mathbf{x}_{i_n}\right) \cdot \lambda^{t-1-i_1-n+1} \right)}_{\text{when } i_n<t:\;\; \mathbf{c}_n[t-1]} \cdot\lambda \\
&\quad + \underbrace{\left(\sum_{1\leq i_1<i_2<\cdots<i_n=t} \left(\mathbf{W}^{(1)}\mathbf{x}_{i_1} \odot \cdots \odot \mathbf{W}^{(n)}\mathbf{x}_{i_n}\right) \cdot \lambda^{t-1-i_1-n+1} \right)}_{\text{when } i_n=t:\;\; \mathbf{z}_n[t]} \\
&= c_n[t-1] \cdot \lambda \; + \; \mathbf{z}_n[t] \\
&= c_n[t-1] \cdot \lambda \; + \; \left(\mathbf{c}_{n-1}[t-1] \odot \mathbf{W}^{(n)}\mathbf{x}_t\right)
\end{aligned}$$

DP equations for other versions of RCNN can be derived exactly the same way. □

A.2 Proof of Theorem 1

Theorem 1 states that each version of RCNN (Table 2.2) computes an underlying variant of string kernel (Table 2.3). Again, in this proof we use the multiplicative and additive version as the running example. The proof sketch applies to other variants as well.

We first generalize the kernel definition in Eq.(2.7) to the case of any n-gram. For any integer $n > 0$, the underlying mapping of the string kernel is defined as,

$$\phi_n(\mathbf{x}) \;=\; \sum_{1 \le i_1 < \cdots < i_n \le |\mathbf{x}|} \lambda^{|x| - i_1 - n + 1} \mathbf{x}_{i_1} \otimes \mathbf{x}_{i_2} \otimes \cdots \otimes \mathbf{x}_{i_n}$$

Now recall that the n-gram RCNN computes its states as

$$\mathbf{c}_n[t] \;=\; \lambda \cdot \mathbf{c}_n[t-1] + \left(\mathbf{c}_{n-1}[t-1] \odot \mathbf{W}^{(n)} \mathbf{x}_t\right)$$

From the derivation of the dynamic programming algorithm in A.1, we know that this is equivalent to summing over all n-grams within the first t tokens $\mathbf{x}_{1:t} = \{\mathbf{x}_1, \cdots, \mathbf{x}_t\}$,

$$\mathbf{c}_n[t] \;=\; \sum_{1 \le i_1 < i_2 < \cdots < i_n \le t} \left(\mathbf{W}^{(1)} \mathbf{x}_{i_1} \odot \cdots \odot \mathbf{W}^{(n)} \mathbf{x}_{i_n}\right) \cdot \lambda^{t - i_1 - n + 1}$$

In particular, the value of the i-th entry, $\mathbf{c}_n[t][i]$, is equal to,

$$\begin{aligned} \mathbf{c}_n[t][i] \;&=\; \sum_{1 \le i_1 < i_2 < \cdots < i_n \le t} \underbrace{\left\langle \mathbf{w}_i^{(1)}, \mathbf{x}_{i_1} \right\rangle \cdots \left\langle \mathbf{w}_i^{(n)}, \mathbf{x}_{i_n} \right\rangle}_{\left\langle \mathbf{x}_{i_1} \otimes \cdots \otimes \mathbf{x}_{i_n}, \mathbf{w}_i^{(1)} \otimes \cdots \otimes \mathbf{w}_i^{(n)} \right\rangle} \cdot \lambda^{t - i_1 - n + 1} \qquad \text{(A.1)} \\ &=\; \left\langle \sum_{1 \le i_1 < i_2 < \cdots < i_n \le t} \lambda^{t - i_1 - n + 1} \mathbf{x}_{i_1} \otimes \cdots \otimes \mathbf{x}_{i_n},\; \mathbf{w}_i^{(1)} \otimes \cdots \otimes \mathbf{w}_i^{(n)} \right\rangle \\ &=\; \langle \phi_n(\mathbf{x}_{1:t}), \phi_n(\mathbf{w}_{i,n}) \rangle \end{aligned}$$

where $\mathbf{w}_i^{(k)}$ represents the i-th row of matrix $\mathbf{W}^{(k)}$ and $\mathbf{w}_{i,n} = \{\mathbf{w}_i^{(1)}, \cdots, \mathbf{w}_i^{(n)}\}$. This completes the proof since $\mathcal{K}_n(\mathbf{x}_{1:t}, \mathbf{w}_{i,n}) = \langle \phi_n(\mathbf{x}_{1:t}), \phi_n(\mathbf{w}_{i,n}) \rangle$ by definition. □

A.3 Proof of Theorem 3

We first review necessary concepts and notations for the ease of reading. Similar to the proof in A.2, the generalized string kernel $\mathcal{K}^{(l)}$ and $\mathcal{K}_\sigma^{(l)}$ in Eq.(2.8) can be defined with the underlying mappings,

$$\begin{aligned}
\phi^{(l)}(\mathbf{x}) &= \sum_{1 \le i_1 < \cdots < i_n \le |\mathbf{x}|} \lambda^{|x| - i_1 - n + 1} \, \phi_\sigma^{(l-1)}(\mathbf{x}_{1:i_1}) \otimes \phi_\sigma^{(l-1)}(\mathbf{x}_{1:i_2}) \otimes \cdots \otimes \phi_\sigma^{(l-1)}(\mathbf{x}_{1:i_1}) \\
\phi_\sigma^{(l)}(\mathbf{x}) &= \phi_\sigma(\phi^{(l)}(\mathbf{x}))
\end{aligned}$$

where $\phi_\sigma()$ is the underlying mapping of a kernel function whose reproducing kernel Hilbert space (RKHS) contains the non-linear activation $\sigma()$ used in the RCNN layer. Here $\mathcal{K}^{(l)}()$ is the "pre-activation kernel" and $\mathcal{K}_\sigma^{(l)}()$ is the "post-activation kernel".

To show that the values of RCNN states $\mathbf{c}^{(l)}[t]$ is contained in the RKHS of $\mathcal{K}^{(l)}()$ and that of $\mathbf{h}^{(l)}[t]$ is contained in the RKHS of $\mathcal{K}_\sigma^{(l)}()$, we re-state the claim in the following way,

Theorem 4. *Given a deep n-gram RCNN model with non-linear activation $\sigma()$. Assuming $\sigma()$ lies in the RKHS of a kernel function with underlying mapping $\phi_\sigma()$, then*

(i) $\mathbf{c}^{(l)}[t]$ *lies in the RKHS of kernel $\mathcal{K}^{(l)}()$ in the sense that*

$$\mathbf{c}_j^{(l)}[t][i] = \left\langle \phi^{(l)}(\mathbf{x}_{1:t}), \psi_{i,j}^{(l)} \right\rangle$$

for any internal state $\mathbf{c}_j^{(l)}[t]$ ($1 \le j \le n$) of the l-th layer, where $\psi_{i,j}^{(l)}$ is a mapping constructed from the parameters of the network;

(ii) $\mathbf{h}^{(l)}[t]$ *lies in the RKHS of kernel $\mathcal{K}_\sigma^{(l)}()$ as a corollary in the sense that*

$$\begin{aligned}
\mathbf{h}^{(l)}[t][i] &= \sigma(\mathbf{c}_n^{(l)}[t][i]) \\
&= \sigma\left(\left\langle \phi^{(l)}(\mathbf{x}_{1:t}), \psi_{i,j}^{(l)} \right\rangle\right) && \textit{(based on (i))} \\
&= \left\langle \phi_\sigma(\phi^{(l)}(\mathbf{x}_{1:t})), \psi_\sigma(\psi_{i,j}^{(l)}) \right\rangle && \textit{(Lemma 1)}
\end{aligned}$$

and we denote $\psi_\sigma(\psi_{i,j}^{(l)})$ as $\psi_{\sigma,i,j}^{(l)}$ for short.

Proof: We prove by induction on l. When $l = 1$, the proof of Theorem 1 already shows that $\mathbf{c}_j^{(1)}[t][i] = \left\langle \phi_j^{(1)}(\mathbf{x}_{1:t}), \phi_j^{(1)}(\mathbf{w}_{i,j}) \right\rangle$ in a one-layer RCNN. Simply let $\psi_{i,j}^{(1)} = \phi_j^{(1)}(\mathbf{w}_{i,j})$ completes the proof for the case of $l = 1$.

Suppose the lemma holds for $l = 1, \cdots, k$, we now prove the case of $l = k + 1$. Similar to Eq.(A.1) in the proof of Theorem 1, the value of $\mathbf{c}_j^{(k+1)}[t][i]$ equals to

$$\mathbf{c}_j^{(k+1)}[t][i] \; = \sum_{1 \leq i_1 < \cdots < i_j \leq t} \left\langle \mathbf{w}_i^{(1)}, \mathbf{h}^{(k)}[i_1] \right\rangle \cdots \left\langle \mathbf{w}_i^{(j)}, \mathbf{h}^{(k)}[i_j] \right\rangle \cdot \lambda^{t-i_1-j+1} \qquad \text{(A.2)}$$

where $\mathbf{w}_i^{(j)}$ is the i-th row of the parameter matrix $\mathbf{W}^{(j)}$ of the l-th layer. Note $\mathbf{h}^{(k)}[t][i] = \left\langle \phi_\sigma^{(k)}(\mathbf{x}_{1:t}), \psi_{\sigma,i,n}^{(k)} \right\rangle$, we construct a matrix $\mathbf{M}$ by stacking all $\{\psi_{\sigma,i,n}^{(k)}\}_i$ as the row vectors.[1] We then can rewrite $\mathbf{h}^{(k)}[t] = \mathbf{M}\,\phi_\sigma^{(k)}(\mathbf{x}_{1:t})$. Plugging this into Eq (A.2), we get

$$\begin{aligned}
\mathbf{c}_j^{(k+1)}[t][i] \; &= \sum_{1 \leq i_1 < \cdots < i_j \leq t} \left\langle \mathbf{w}_i^{(1)}, \mathbf{M}\,\phi_\sigma^{(k)}(\mathbf{x}_{1:i_1}) \right\rangle \cdots \left\langle \mathbf{w}_i^{(j)}, \mathbf{M}\,\phi_\sigma^{(k)}(\mathbf{x}_{1:i_j}) \right\rangle \cdot \lambda^{t-i_1-j+1} \\
&= \sum_{1 \leq i_1 < \cdots < i_j \leq t} \left\langle \underbrace{\mathbf{M}^\top \mathbf{w}_i^{(1)}}_{\mathbf{u}_{i,1}}, \phi_\sigma^{(k)}(\mathbf{x}_{1:i_1}) \right\rangle \cdots \left\langle \underbrace{\mathbf{M}^\top \mathbf{w}_i^{(j)}}_{\mathbf{u}_{i,j}}, \phi_\sigma^{(k)}(\mathbf{x}_{1:i_j}) \right\rangle \cdot \lambda^{t-i_1-j+1} \\
&= \sum_{1 \leq i_1 < \cdots < i_j \leq t} \langle \mathbf{u}_{i,1}, \phi_\sigma^{(k)}(\mathbf{x}_{1:i_1}) \rangle \cdots \langle \mathbf{u}_{i,j}, \phi_\sigma^{(k)}(\mathbf{x}_{1:i_j}) \rangle \cdot \lambda^{t-i_1-j+1} \\
&= \left\langle \underbrace{\sum_{1 \leq i_1 < \cdots < i_j \leq t} \lambda^{t-i_1-j+1}\, \phi_\sigma^{(k)}(\mathbf{x}_{i_1}) \otimes \cdots \otimes \phi_\sigma^{(k)}(\mathbf{x}_{i_j})}_{\phi^{(k+1)}(\mathbf{x}_{1:t})},\; \mathbf{u}_{i,1} \otimes \cdots \otimes \mathbf{u}_{i,j} \right\rangle \\
&= \langle \phi^{(k+1)}(\mathbf{x}_{1:t}),\; \mathbf{u}_{i,1} \otimes \cdots \otimes \mathbf{u}_{i,j} \rangle
\end{aligned}$$

Define $\psi_{i,j}^{(k+1)} = \mathbf{u}_{i,1} \otimes \cdots \otimes \mathbf{u}_{i,j}$ completes the proof for the case of $l = k + 1$. □

[1] Note in practice the mappings $\phi_\sigma^{(k)}$ and $\psi_\sigma^{(k)}$ may have infinite dimension because the underlying mapping for the non-linear activation $\phi_\sigma()$ can have infinite dimension (See Zhang et al. [120, 121]). The proof still apply since the dimensions are still countable and the vectors have finite norm (easy to show this by assuming the input $\mathbf{x}_i$ and parameter $\mathbf{W}$ are bounded.

B
Experimental Details

B.1 Language Modeling on PTB

Initialization We initialize all parameter matrices from a uniform distribution $\mathrm{U}\left[-\sqrt{\frac{3}{d_i}}, +\sqrt{\frac{3}{d_i}}\right]$, where d_i is the input dimension. This means each row vector has zero mean and unit variance, as suggested by recent work [32, 36]. The bias term is initialized to 0.

Dropout We apply the standard dropout [41] on the input word embeddings and from the last recurrent layer to the output softmax layer. No dropout is applied within or between recurrent layers. For the small networks with 5m parameters, we use a dropout probability of 0.5. For the large networks with 20m parameters, we use a probability of 0.75.

Optimization We use the default hyperparemeters for Adam optimizer and an initial learning rate of 0.001. We train 30 epochs and for larger networks we halve the learning rate and revert the parameter values (to the version before this epoch) if the development perplexity doesn't improve.

For SGD optimizer, our setting largely follows recent work of state-of-the-art models. The initial learning rate is 1, and the gradient $L2$ norm is clipped to 5. The learning rate decay is set to 0.9, and we start to decrease the learning rate after 10 epochs for small networks and after 20 epochs for large networks. We train a maximum of 50 epochs for SGD optimizer.

Model Configuration To achieve a performance competitive to the state-of-the-art, we perform a hyperparameter search on the various configuration options such as the number of layers (1 to 3 layers), the activation function (tanh or identity). For the model using the adaptive decay, the best configuration uses 3 layers, an n-gram order $n = 1$ within each layer and a highway connection between each recurrent layer. tanh activation is not used.[1] The recurrent unit of each layer is quite simple,

$$\begin{aligned}\mathbf{c}[t] &= \lambda_t \odot \mathbf{c}[t-1] + (1-\lambda_t) \odot (\mathbf{W}\mathbf{x}_t) \\ \mathbf{h}[t] &= \mathbf{f}_t \odot \mathbf{c}[t] + (1-\mathbf{f}_t) \odot \mathbf{x}_t\end{aligned}$$

where λ_t is the adaptive decay given by $\sigma(\mathbf{U}\mathbf{x}_t + \mathbf{b})$ or $\sigma(\mathbf{U}\mathbf{x}_t + \mathbf{V}\mathbf{h}[t-1] + \mathbf{b})$, and $\mathbf{f}_t = \sigma(\mathbf{U}'\mathbf{x}_t + \mathbf{b}')$ is the forget gate of the highway connection.

For the model using the constant decay, a 2-gram multiplicative mapping performs better,

$$\begin{aligned}\mathbf{c}_1[t] &= \lambda_t \odot \mathbf{c}_1[t-1] + (1-\lambda_t) \odot (\mathbf{W}^{(1)}\mathbf{x}_t) \\ \mathbf{c}_2[t] &= \lambda_t \odot \mathbf{c}_2[t-1] + (1-\lambda_t) \odot (\mathbf{c}_1[t-1] \odot \mathbf{W}^{(2)}\mathbf{x}_t) \\ \mathbf{h}[t] &= \mathbf{f}_t \odot (\mathbf{c}_1[t] + \mathbf{c}_2[t]) + (1-\mathbf{f}_t) \odot \mathbf{x}_t\end{aligned}$$

where both 1-gram and 2-gram representations $\mathbf{c}_1[t]$ and $\mathbf{c}_2[t]$ are added in the final representation.

[1] Using tanh gives very close performance.

B.2 Sentiment Classification

Initialization The parameter matrices are initialized from a uniform distribution $\mathrm{U}\left[-\sqrt{\frac{3}{d_i}}, +\sqrt{\frac{3}{d_i}}\right]$, similar to the models for language modeling. The word vectors are pre-trained and are fixed for simplicity.[2]

Model Configuration Following our prior work [66], we use 3 recurrent layers with ReLU activation and the hidden dimension $d = 200$ for each layer. We use the 2-gram multiplicative version of RCNN. The output hidden states $\mathbf{h}[t]$ of each layer are averaged across t. We concatenate the averaged vectors from the 3 recurrent layers, and use the concatenation as the input of the final softmax layer.

Optimization We use the default hyperparemeters for Adam optimizer and an initial learning rate of 0.001. We train a maximum of 50 epochs and after each epoch we decrease the learning rate by a factor of 0.95. We apply a dropout probability of 0.35 on the input and output of each layer. The L2 regularization coefficient (i.e. weight decay) is set to $1e - 6$.

[2] It may be helpful to fine tune the word vectors for the classification task.

Bibliography

[1] Senjian An, Farid Boussaid, and Mohammed Bennamoun. How can deep rectifier networks achieve linear separability and preserve distances. In *Proceedings of the 32nd International Conference on Machine Learning (ICML).*

[2] Senjian An, Munawar Hayat, Salman H Khan, Mohammed Bennamoun, Farid Boussaid, and Ferdous Sohel. Contractive rectifier networks for nonlinear maximum margin classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2515–2523, 2015.

[3] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple object recognition with visual attention. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

[5] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[6] Daniel Bär, Torsten Zesch, and Iryna Gurevych. Dkpro similarity: An open source framework for text similarity. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 121–126, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P13-4021`.

[7] Justin Bayer, Daan Wierstra, Julian Togelius, and Jürgen Schmidhuber. Evolving memory cell structures for sequence learning. In *International Conference on Artificial Neural Networks*, pages 755–764, 2009.

[8] Yoshua Bengio, Rėjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155, 2003.

[9] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2015.

[10] Danqi Chen and Christopher D Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, 2014.

[11] Kan Chen, Jiang Wang, Liang-Chieh Chen, Haoyuan Gao, Wei Xu, and Ram Nevatia. Abc-cnn: An attention based convolutional neural network for visual question answering. *arXiv preprint arXiv:1511.05960*, 2015.

[12] Jianpeng Cheng, Li Dong, and title = Long Short-Term Memory Networks for Machine Reading journal = EMNLP year = 2016 pages = 551–562 Lapata, Mirella.

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

[14] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[15] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. *arXiv preprint arXiv:1412.0233*, 2014.

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

[17] Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. In *Proceedings of the 29th Conference on Learning Theory COLT*, pages 698–728, 2016.

[18] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *International Conference on Machine Learning, ICML*, 2008.

[19] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

[20] Mark W Craven and Jude W Shavlik. Extracting tree-structured representations of trained networks. In *Advances in neural information processing systems (NIPS)*, 1996.

[21] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems*, pages 3061–3069, 2015.

[22] Yann Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, 2014.

[23] Olivier Delalleau and Yoshua Bengio. Shallow vs. deep sum-product networks. In *Advances in Neural Information Processing Systems*, pages 666–674, 2011.

[24] Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. Fast and robust neural network joint models for statistical machine translation. In *52nd Annual Meeting of the Association for Computational Linguistics*, 2014.

[25] Cicero dos Santos, Luciano Barbosa, Dasha Bogdanova, and Bianca Zadrozny. Learning hybrid representations to retrieve semantically equivalent questions. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 694–699, Beijing, China, July 2015. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P15-2114`.

[26] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[27] A Evgeniou and Massimiliano Pontil. Multi-task feature learning. In *Advances in neural information processing systems: Proceedings of the 2006 conference*. The MIT Press, 2007.

[28] Manaal Faruqui, Jesse Dodge, Sujay K. Jauhar, Chris Dyer, Eduard Hovy, and Noah A. Smith. Retrofitting word vectors to semantic lexicons. In *Proceedings of NAACL*, 2015.

[29] Manaal Faruqui, Yulia Tsvetkov, Dani Yogatama, Chris Dyer, and Noah A. Smith. Sparse overcomplete word vector representations. In *Proceedings of ACL*, 2015.

[30] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems 29 (NIPS)*, 2016.

[31] Jianfeng Gao, Patrick Pantel, Michael Gamon, Xiaodong He, Li Deng, and Yelong Shen. Modeling interestingness with deep neural networks. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 2014.

[32] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[33] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems.*

[34] Moritz Hardt and Tengyu Ma. Identity matters in deep learning. *CoRR*, abs/1611.04231, 2016.

[35] Tamir Hazan and Tommi Jaakkola. Steps toward deep kernel methods from infinite neural networks. *arXiv preprint arXiv:1508.05133*, 2015.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[37] Aurélie Herbelot and Eva Maria Vecchi. Building a shared world: mapping distributional to model-theoretic semantic spaces. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing.* Association for Computational Linguistics, 2015.

[38] Karl Moritz Hermann and Phil Blunsom. The role of syntax in vector space models of compositional semantics. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, 2013.

[39] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1684–1692, 2015.

[40] Michiel Hermans and Benjamin Schrauwen. Training and analysing deep recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 190–198, 2013.

[41] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

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

[43] Ozan Irsoy and Claire Cardie. Deep recursive neural networks for compositionality in language. In *Advances in Neural Information Processing Systems*, 2014.

[44] Mohit Iyyer, Jordan Boyd-Graber, Leonardo Claudino, Richard Socher, and Hal Daumé III. A neural network for factoid question answering over paragraphs. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 633–644, Doha, Qatar, October 2014. URL `http://www.aclweb.org/anthology/D14-1070`.

[45] Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2015.

[46] T. Joachims. Optimizing search engines using clickthrough data. In *ACM SIGKDD KDD*, 2002.

[47] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[48] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. *Journal of Machine Learning Research*, 2015.

[49] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP 2013)*, pages 1700–1709, 2013.

[50] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of the 52th Annual Meeting of the Association for Computational Linguistics*, 2014.

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

[52] Dimitri Kartsaklis, Mehrnoosh Sadrzadeh, and Stephen Pulman. A unified sentence space for categorical distributional-compositional semantics: Theory and experiments. In *In Proceedings of COLING: Posters*, 2012.

[53] Kenji Kawaguchi. Deep learning without poor local minima. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[54] B Kim, JA Shah, and F Doshi-Velez. Mind the gap: A generative approach to interpretable feature selection and extraction. In *Advances in Neural Information Processing Systems*, 2015.

[55] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 2014. URL `http://arxiv.org/abs/1408.5882`.

[56] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[57] Diederik P Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representation*, 2015.

[58] Andreas Küchler and Christoph Goller. Inductive learning in symbolic domains using structure-driven recurrent neural networks. In *KI-96: Advances in Artificial Intelligence*, pages 183–197. 1996.

[59] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, Ishaan Gulrajani James Bradbury, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. 2016.

[60] Phong Le and Willem Zuidema. Compositional distributional semantics with long short term memory. In *Proceedings of Joint Conference on Lexical and Computational Semantics (*SEM)*, 2015.

[61] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1188–1196, 2014.

[62] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[63] Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 1999.

[64] T. Lei, R. Barzilay, and T. Jaakkola. Rationalizing neural predictions. In *Empirical Methods in Natural Language Processing*, 2016.

[65] Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. Low-rank tensors for scoring dependency structures. In *Proceedings of the 52th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2014.

[66] Tao Lei, Regina Barzilay, and Tommi Jaakkola. Molding cnns for text: non-linear, non-consecutive convolutions. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

[67] Tao Lei, Hrishikesh Joshi, Regina Barzilay, Tommi Jaakkola, Katerina Tymoshenko, Alessandro Moschitti, and Lluís Màrquez. Semi-supervised question retrieval with gated convolutions. In *Proceedings of the 2016 Conference of*

the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL), 2016.

[68] Benjamin Letham, Cynthia Rudin, Tyler H. McCormick, and David Madigan. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. *Annals of Applied Statistics*, 9(3):1350–1371, 2015.

[69] Jiwei Li, Minh-Thang Luong, Dan Jurafsky, and Eudard Hovy. When are tree structures necessary for deep learning of representations? 2015.

[70] Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. Visualizing and understanding neural models in nlp. In *Proceedings of NAACL*, 2016.

[71] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2(Feb):419–444, 2002.

[72] Iain J Marshall, Joël Kuiper, and Byron C Wallace. Robotreviewer: evaluation of a system for automatically assessing bias in clinical trials. *Journal of the American Medical Informatics Association*, 2015.

[73] André F. T. Martins and Ramón Fernandez Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. *CoRR*, abs/1602.02068, 2016.

[74] Julian McAuley, Jure Leskovec, and Dan Jurafsky. Learning attitudes and attributes from multi-aspect reviews. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 1020–1025. IEEE, 2012.

[75] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[76] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pages 1045–1048, 2010.

[77] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, 2013.

[78] Jeff Mitchell and Mirella Lapata. Vector-based models of semantic composition. In *ACL*, pages 236–244, 2008.

[79] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[80] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932, 2014.

[81] Thien Huu Nguyen and Ralph Grishman. Modeling skip-grams for event detection with convolutional neural networks. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

[82] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of 30th International Conference on Machine Learning (ICML)*, 2013.

[83] Razvan Pascanu, Guido Montúfar, and Yoshua Bengio. On the number of inference regions of deep feed forward networks with piece-wise linear activations. *CoRR*, abs/1312.6098, 2013.

[84] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. volume 12, 2014.

[85] Jordan B Pollack. Recursive distributed representations. *Artificial Intelligence*, 46:77–105, 1990.

[86] Ofir Press and Lior Wolf. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

[87] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?": Explaining the predictions of any classifier. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016.

[88] Stephen Robertson and Hugo Zaragoza. *The probabilistic relevance framework: BM25 and beyond.* Now Publishers Inc, 2009.

[89] Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiský, and Phil Blunsom. Reasoning about entailment with neural attention. In *International Conference on Learning Representations*, 2016.

[90] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

[91] Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015.

[92] Shai Shalev-Shwartz, Ohad Shamir, and Karthik Sridharan. Learning kernel-based halfspaces with the 0-1 loss. *SIAM Journal on Computing*, 40(6):1623–1646, 2011.

[93] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the companion publication of the 23rd international conference on World wide web companion*, pages 373–374. International World Wide Web Conferences Steering Committee, 2014.

[94] Richard Socher, Cliff C. Lin, Andrew Y. Ng, and Christopher D. Manning. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, 2011.

[95] Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 151–161. Association for Computational Linguistics, 2011.

[96] Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. Parsing With Compositional Vector Grammars. In *ACL*. 2013.

[97] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, October 2013.

[98] Richard Socher, Andrej Karpathy, Quoc V Le, Christopher D Manning, and Andrew Y Ng. Grounded compositional semantics for finding and describing images with sentences. *Transactions of the Association for Computational Linguistics*, 2:207–218, 2014.

[99] Nathan Srebro and Tommi Jaakkola. Weighted low-rank approximations. In *ICML*, 2003.

[100] Nathan Srebro, Jason Rennie, and Tommi S Jaakkola. Maximum-margin matrix factorization. In *Advances in neural information processing systems*, 2004.

[101] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385, 2015.

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

[103] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53th Annual Meeting of the Association for Computational Linguistics*, 2015.

[104] Aviv Tamar, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[105] Ming Tan, Bing Xiang, and Bowen Zhou. Lstm-based deep learning models for non-factoid answer selection. *arXiv preprint arXiv:1511.04108*, 2015.

[106] Sebastian Thrun. Extracting rules from artificial neural networks with distributed representations. In *Advances in neural information processing systems (NIPS)*, 1995.

[107] Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10. Association for Computational Linguistics, 2010.

[108] Lex Weaver and Nigel Tao. The optimal reward baseline for gradient-based reinforcement learning. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, 2001.

[109] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992.

[110] Huijuan Xu and Kate Saenko. Ask, attend and answer: Exploring question-guided spatial attention for visual question answering. *arXiv preprint arXiv:1511.05234*, 2015.

[111] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015.

[112] Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, and Alex Smola. Stacked attention networks for image question answering. *arXiv preprint arXiv:1511.02274*, 2015.

[113] Dauphin Yann, Harm de Vries, Junyoung Chung, and Yoshua Bengio. Equilibrated adaptive learning rates for non-convex optimization. *arXiv:1502.04390*, 2015.

[114] Wen-tau Yih, Xiaodong He, and Christopher Meek. Semantic parsing for single-relation question answering. In *Proceedings of ACL*, 2014.

[115] Omar Zaidan, Jason Eisner, and Christine D. Piatko. Using "annotator rationales" to improve machine learning for text categorization. In *Proceedings of Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 260–267, 2007.

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

[117] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[118] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, pages 649–657, 2015.

[119] Ye Zhang, Iain James Marshall, and Byron C. Wallace. Rationale-augmented convolutional neural networks for text classification. *CoRR*, abs/1605.04469, 2016.

[120] Yuchen Zhang, Jason D. Lee, and Michael I. Jordan. ℓ_1-regularized neural networks are improperly learnable in polynomial time. In *Proceedings of the 33nd International Conference on Machine Learning*, 2016.

[121] Yuchen Zhang, Percy Liang, and Martin J. Wainwright. Convexified convolutional neural networks. *CoRR*, abs/1609.01000, 2016.

[122] Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent Highway Networks. *arXiv preprint arXiv:1607.03474*, 2016.

[123] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.