Document Summarization (draft)

Jeffrey Ling

March 28, 2017

Abstract

While humans are naturally able to produce high-level summaries upon reading paragraphs of text, computers still find such a task enormously difficult. Despite progress over the years, the general problem of document summarization remains intractable

Inspired by recent work in deep learning, we apply the sequence-to-sequence (seq2seq) model with attention to the summarization problem. While seq2seq models are successful in a variety of natural language processing tasks, the computation does not scale well to tasks with long sequences such as documents. To that end, we propose a novel coarse-to-fine attention method to reduce the computational complexity of the standard attention model.

We evaluate our model on the CNN/Dailymail document summarization dataset. Results...

Contents

Chapter 1

Introduction

1.1 Natural Language Processing

The field of natural language processing arises from a very simple question: how can we teach machines to read, speak, and understand the words that we use with such ease and fluency?

Such a question has been considered since the first computers were built. The classic Turing test, posed by Alan Turing in 1950, requires a machine to converse in a way that is indistinguishable from a human, and thus requires a fundamental grasp on how to properly use language. Although it was simple for Turing to conceptualize what a successful machine might look like, many have been stumped on how to actually construct such a system. Indeed, to this day, no machine has been able to fully pass the Turing test as it was originally posed.

While computers can now run computations at a rate that far exceeds human cognition, language tasks that we consider trivial still prove to be extraordinarily difficult for a machine to solve. Consider the problem of deciding words with multiple meanings such as "bass" (word sense disambiguation), or the problem of identifying to what or whom a certain pronoun refers (coreference). While humans reliably perform these functions on a daily basis, they are not at all easy for computers to handle.

However, the need for computers to understand language has never been greater. In today's information age, NLP grows increasingly important as the accumulation of free-form text begins to outpace the ability of humans to process it. In fields such as medicine, this can mean missed diagnoses; in law, wasted effort on irrelevant documents; in international relations, misinterpretations of foreign articles. Because natural language

is everywhere and used by everyone, the demand for text processing solutions remains as high as ever.

1.2 Methods in NLP

We have established that NLP is a difficult yet worthwhile undertaking. In this section, we investigate the general philosophy of tackling these hard problems.

The key property of language that makes it difficult for machines to handle is its discrete and combinatorial nature. When we form sentences, we can string together arbitrary words from our vocabulary, as long as we follow the rules of some highly structured grammar. In some sense, the essential difficulty in NLP lies in handling these complicated structured problems.

Linguistics attempts to answer this by building a formal theory of language. Indeed, many ideas from linguistics, including sentence parses, morphology, and semantics are invaluable in NLP for understanding the rules for how sentences and phrases can be put together. In order to solve a language problem, we might begin by enriching the surface form of text (the raw sentences and paragraphs) with the parse structure, parts of speech, coreferent entities, etc. We can then use this featurized form of language in whichever way we prefer.

Another mode of thought focuses on the role of statistics in language. A cursory examination of the distribution of English words reveals an interesting power law distribution known as Zipf's law (Figure 1.1). By drawing from ideas of information theory (Shannon, 1948), we can treat language as the result of some noisy probabilistic process, and we can reduce problems such as language modeling to learning parameters of some simple distributions.

Historically, there has been contention about the roles of linguistics and statistics in NLP. Certain practical problems seem to be better off without linguistic theory; as IBM researcher Frederick Jelinek famously said, "Every time I fire a linguist, the performance of the speech recognizer goes up." Indeed, even naive models of language perform well if given enough training data, as evidenced by the IBM models for machine translation (Brown et al., 1993). citation

Today, the linguistics-free approach has been taken to an extreme by deep learning systems. For the first time, neural networks are able to learn to perform language tasks in an end-to-end fashion (Collobert et al., 2011b), i.e. without the linguistic preprocessing

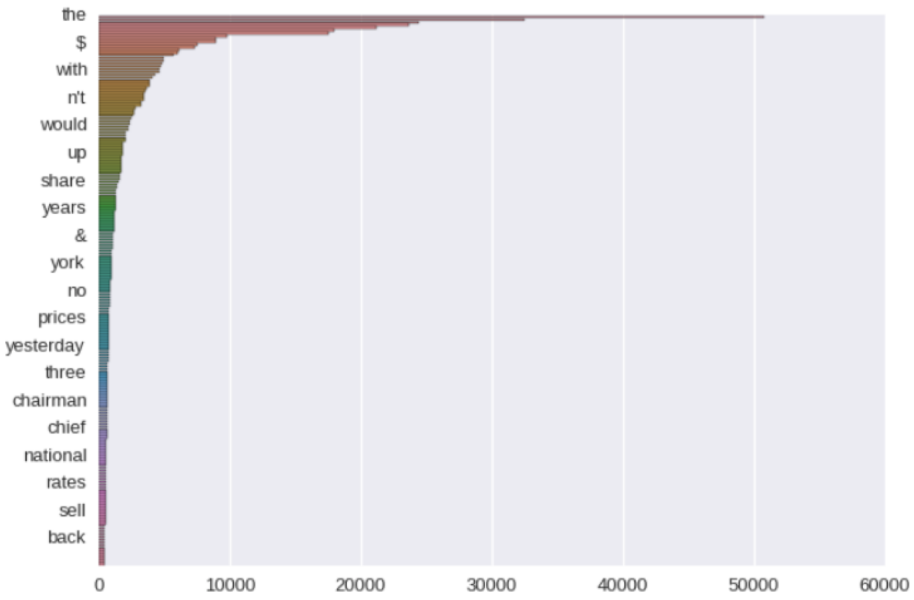


Figure 1.1: Zipf's law, indicating the relationship between English words and their frequency. The distribution approximately follows an inverse power law. Image from CS287.

that was once considered necessary. Neural methods have been adopted by user-facing systems like Google Translate with great success (Wu et al., 2016).

1.3 Automatic Summarization

In this thesis, we will consider the particular problem of *text summarization*. That is, given a document with several sentences of text, the goal is to produce a short summary that captures most or all of its salient points.

why is this important

Nenkova and McKeown (2011) give a comprehensive overview of the problem of summarization. In particular, they provide a taxonomy of methods that researchers have developed to tackle the task:

Extractive vs. abstractive Extractive summaries extract certain sentences or phrases from the document, while abstractive summaries take a more holistic view and can in practice be anything (similar to how humans produce summaries).

Extractive methods are by far more popular due to the simplicity of building an extraction algorithm. Abstractive methods are much more difficult, as we need to both

extract relevant content and also ensure grammaticality of the output (a difficult problem in itself).

Single- vs. multi-document Summarization was originally posed as the problem of producing a summary for a single document. However, with the onset of the Internet, there are often multiple documents on the same topic, and so a summarization system should be able to use all of them to produce a summary.

Interestingly, it has been noted that the multi-document problem is often easier due to redundant information across sources.

Keyword vs. headline Keyword summaries are allowed to simply be a bag-of-words of important keywords from the document, while headline summaries must form a coherent sentence.

Generic vs. query-focused Generic summaries make no assumptions about the reader and are meant to be generally informative, while query-focused summaries take into consideration a query and only return relevant information.

The contrast between these two methods highlights an important question in summarization: to what end are we summarizing documents? If we can answer this question, we can build systems to more accurately accomplish our desired tasks.

Most of this taxonomy is based upon insights from DUC (Document Understanding Conferences), a set of tasks released by NIST between 2001-2006 to promote research in summarization (Over et al., 2007).

The DUC tasks were fairly diverse and varied from year to year. DUC 2001 and 2002 asked for generic summaries of news articles, while DUC 2003 presented a multi-document problem. DUC 2004-2006 shifted to a more question-answering based approach, and many of the documents were accompanied by focused queries.

While DUC did not lead to any definitive answers on how best to summarize documents, some important empirical discoveries were noted. For the generic summarization tasks, it was found that the first sentence of news articles was a strong baseline that more sophisticated methods found hard to beat. Thus, the generic summary task was seen as ill-formed and not as interesting, leading to the more query-focused tasks in the later years.

DUC also inspired a lot of thinking on the best way to evaluate summaries. Evaluating a summary is inherently ambiguous, as even humans tend to disagree on what makes for a good summary. Finding a quantitative metric for evaluating summaries turns out to be one of the hardest parts of the problem. While a single most effective metric for summarization may not exist, DUC established several important criteria, including grammaticality, non-redundancy, and content coverage.

For extractive summaries, people have proposed simple metrics such as precision and recall on selected sentences. These naturally do not work too well since 1) not all sentences are equally informative, and 2) not all parts of a sentence are relevant.

When we don't have sentence labels as in the extractive case, evaluation is harder. One proposed metric is recall on elementary discourse units (EDUs), labeled clauses within a summary that ought to be captured. ROUGE (Lin, 2004), inspired by the BLEU metric for machine translation (Papineni et al., 2002), is a cheap and fast method based on n-gram precision, recall, and length considerations.

None of these methods directly address the grammaticality of the output. Aside from using human evaluation, meaningful metrics for summaries is still very much an open question (Toutanova et al., 2016).

Going forward, we will limit our scope to the single-document, headline, and generic summarization case. We will explore a few different algorithms in extractive and abstractive summarization, and connect these approaches with recent trends in deep learning.

1.4 Summarization Methods

One of the first treatises on automatically producing summaries was Luhn (1958), which considered the problem of producing abstracts for scientific articles. At the time, computers were still monolithic machines that ran on punch cards, so automating the summarization process was quite ahead of its time.

Luhn (1958) proposed a simple sentence-ranking method to produce summaries. The algorithm gives each sentence a score based on the occurrence of frequently appearing words. This is one of the first examples of an extractive summarization method.

Since then, a variety of approaches have been used to solve summarization. We highlight some notable work in both the extractive and abstractive frameworks.

1.4.1 Extractive

The most popular methods for document summarization have generally been extractive due to their simplicity.

There are two natural procedures for extractive summarization: one is to produce a ranking of sentences based on some scoring function, and two is to classify sentences as either in the summary or not. Luhn (1958) is an example of the first approach.

Carbonell and Goldstein (1998) extend on the scoring-ranking method with an information metric, MMR (maximum marginal relevance), that penalizes pairwise similarity between sentences. Their work is one of the first that attempt to reduce redundancy in the summary.

Kupiec et al. (1995) pioneer the second method by treating sentence selection as a classification problem. They obtain a training corpus and apply a naive Bayes approach to classify sentences.

Ranking and classifying algorithms became more sophisticated over time, and most of these methods were soon applied to extractive summarization.

Shen et al. (2004) models sentence extraction as a sequential classification problem, training a linear-chain conditional random field to find the best subset of sentences.

Svore et al. (2007) use a basic neural network as a sentence classifier.

Deep learning models have also been used to extract sentences.

Cao et al. (2015) use convolutional neural networks to extract features for each sentence, combining these with document-level features to produce scores.

Cheng and Lapata (2016) apply the encoder-decoder model using recurrent neural networks to produce labels for each sentence.

finish

1.4.2 Abstractive

While extraction has proven to be successful, the method is inherently limited in its ability to summarize. The more challenging method, and also the closest to what humans do, is *abstractive* summarization. Instead of strictly requiring that all words of the summary come from the source document, any coherent text is allowed.

Two methods used to produce abstractive summaries are sentence compression and sentence fusion. Compression removes less useful information from sentences, while fusion is harder and combines information from sentences.

Compression: Knight and Marcu (2002) employs a noisy channel model, similar to machine translation, to deduce the "most probable" compression, while Clarke and Lapata (2008) uses an integer linear program. Cohn and Lapata (2008) extend the tree-based methods to allow for insertions and substitutions during compression, whereas prior methods were purely deletion based. Zajic et al. (2004) successfully use a sentence compression algorithm along with an unsupervised topic model on the DUC 2004 task.

Fusion: align parse trees and combine phrases that are similar finish

Durrett et al. (2016) apply an ILP approach, where they maximize a score based on textual and coreferent features with certain grammaticality and anaphora constraints.

Deep learning has also been successfully applied to abstractive summarization. Rush et al. (2015) propose a completely data-driven model for headline generation by training an end-to-end model. More recently, Nallapati et al. (2016) apply the same approach on full documents.

As with deep learning methods for extraction, these models require a large amount of supervised training data. One advantage of the abstractive model is that data in the document-abstract format is more easily obtainable than labeled extractive data.

1.4.3 In the Wild

Outside of the academic realm, summarization is an important problem in industry. One noteworthy summarization method is on Reddit[1]: in order to summarize long forum discussions, Reddit uses technology from Smmry[2].

Smmry's algorithm is a simple extractive method. It counts word occurrences, splits discussions by sentence, and ranks the sentences based on the sum of their word scores. This algorithm bears extraordinary similarity to Luhn (1958) — although a variety of work has been done since then, the simplest approaches turn out to be the most practical.

1.5 This Work

Inspired by advances in deep learning for NLP, we set out to build a deep model to abstractively generate summaries. Because deep learning is computationally expensive even for short lengths of text such as sentences, this creates a challenge for the general document summarization problem.

[1] `reddit.com`
[2] `smmry.com`

Hence, we will survey the literature for methods that alleviate the computation of training deep models. Along the way, we propose a new model architecture that extends the sequence-to-sequence model and attempts to reduce the computational complexity of the document summarization problem.

1.6 Outline

We provide an outline for the rest of this thesis.

In Chapter 2, we give a survey of deep learning and motivate its use in solving our problem. In Chapter 3, we provide the necessary background material for understanding our models and algorithms. In Chapter 4, we describe our training algorithm formally. In Chapter 5, we describe our models formally. In Chapter 6, we describe the experimental setup, including our dataset, baselines, and practical details for training our models. In Chapter 7, we show results, analyze the outputs of our models, and provide discussion. Finally, we conclude in Chapter 8.

Chapter 2

Related Work

2.1 Deep Learning

The history of neural networks dates back to the perceptron (Rosenblatt, 1958), a simple model that assumes data can be linearly separated. Due to this strict requirement, the machine learning community dismissed the idea as impractical, and research was sidelined for most of the 20th century.

Later, Rumelhart et al. (1986) showed that the backpropagation algorithm could be used to efficiently train general neural networks, reviving interest in the subject. However, this was before the era of GPUs and modern computing, and so the algorithm could not be put to practical use.

Recently, neural networks have made a resurgence. In the ImageNet image classification competition in 2012, Krizhevsky et al. (2012) won using deep convolutional neural networks (LeCun and Bengio, 1995), beating the competition by a significant margin. This led to a renewed wave of research, especially due to the advancement of GPU computing power, which can train networks at 10 to 20 times the speed of standard CPUs. Today, deep models are successfully used in image recognition (Farabet et al., 2013), speech recognition (Hinton et al., 2012), and Go playing (Silver et al., 2016), just to name a few.

In NLP, one of the first successful neural models was Bengio et al. (2003), which builds a neural language model using multi-layered perceptrons. Later, Collobert et al. (2011b) demonstrate that neural models can be used to train end-to-end models without any of the preprocessing that was once considered necessary.

Since the onset of deep learning, deep models have found their way into nearly every corner of NLP. Much of their success relies on the ubiquity of the *long short-term memory*

(LSTM) recurrent neural network (Hochreiter and Schmidhuber, 1997), a model used to both process and generate sequences of text. Several state-of-the-art algorithms are now based upon deep learning tools such as word vectors and LSTMs. While there is too much to cover here, Goldberg (2015) provides a concise summary of the models that have had the greatest impact on NLP.

2.2 Representation Learning

While neural networks are often treated as black box classifiers, Zeiler and Fergus (2014) show that the intermediate layers of deep convolutional networks contain abstracted qualities of the input, such as patterns, textures, and objects of the input. This suggests that neural networks are discovering features of the input and building generalized *representations* of their inputs.

The idea of learning representations of the input is quite general, and also applies in NLP. Mikolov and Dean (2013) show that by training a neural network on a Google News text corpus, the network learned to map words in the English language to a vector of real numbers known as *word embeddings*. These word embeddings are actually able to capture semantic properties of the words — for example, taking the vectors for *king*, *man*, and *woman*, we find that $v_{king} - v_{man} + v_{woman} \approx v_{queen}$, preserving the analogy that we usually make with text. Figure 2.1 shows some of the word2vec vectors when projected to two dimensional space.

Representation learning has become a central topic in NLP. Word vectors trained on a general text corpus can successfully be applied to almost any NLP task (Mikolov and Dean, 2013; Pennington et al., 2014). It remains to be seen whether it is possible to represent longer pieces of text, such as sentences, in a general way, and research in the area is active (Bowman et al., 2016).

2.3 Motivation

Why deep models? There are currently two general approaches to solving problems in NLP: one is to use as much linguistic theory as possible to reduce the problem, and another is to apply black box learners such as deep neural networks.

Deep models work fantastically well on many tasks, especially when it comes to lowering metrics. However, as big of a hammer as deep learning is, there must be nails

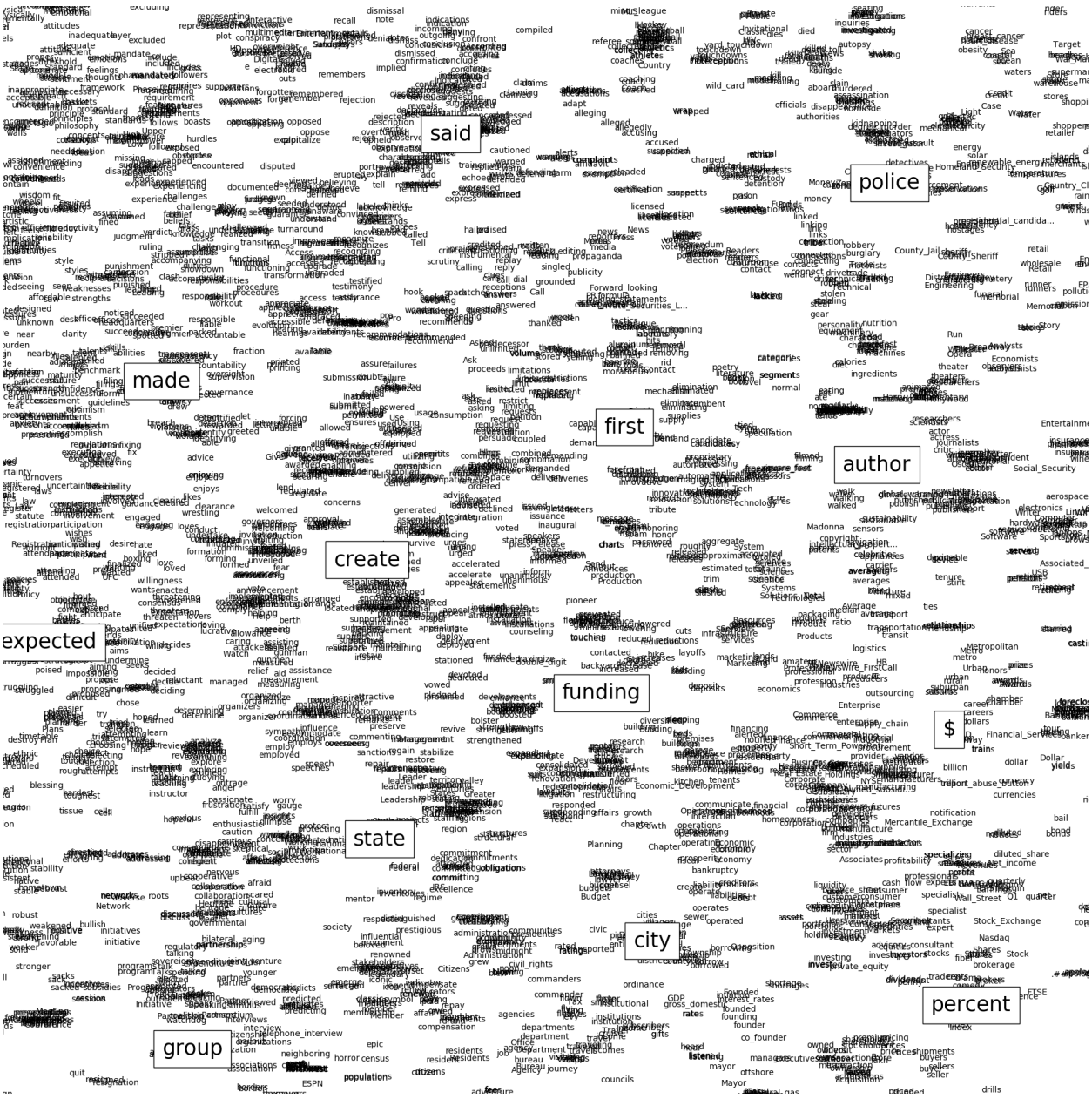


Figure 2.1: A visualization of word2vec word vectors (Mikolov and Dean, 2013), projected to two dimensional space (how?). Words with similar semantic meaning tend to cluster together in the space. Image from CS287.

available for the tool to hit. In order to further language understanding, datasets and tasks must be posed such that deep models can be applied; it is exactly in this domain that classical theory is still relevant. As Manning (2016) argues, although neural models have come to dominate NLP papers, there will always be a need for domain experts to prepare the field so that deep learning can succeed.

With this caveat, there are many worthwhile reasons to study deep networks in NLP. First, they work! In fact, they work remarkably well without any feature engineering, which tends to be one of the fussiest parts of building machine learning algorithms.

Second, they are not mutually exclusive with standard feature extraction methods, and so can augment classical methods.

Third, we find that trained models can discover latent structure in language automatically, which may reveal insights about how language is used. As an example, sequence-to-sequence models with attention (Bahdanau et al., 2014) learn the concept of a word alignment in translation without any direct supervision.

It is this third point upon which we base this thesis. Although they began as black-box optimizers, end-to-end deep models are slowly being dissected into more understandable parts. Our goal is to test the hypothesis that such parts are in fact interpretable and are functioning as we expect them to.

With this goal in mind, we attempt to interpretably extend the attention mechanism of existing sequence-to-sequence models, and we analyze its performance on the task of document summarization. In the next chapter, we provide the background for how we will do this.

needs work

Chapter 3

Background

In this chapter, we set up the relevant background ideas for our models. We describe the sequence-to-sequence attention model at a high level, then survey the literature for methods in reducing the computation cost of deep models. Finally, we introduce the framework of reinforcement learning, which will provide the foundation for one of these methods.

3.1 Sequence-to-Sequence Attention Models

Many NLP problems can be posed as follows: given an input sequence of tokens $x_1, \ldots, x_n \in \mathcal{V}$, we train a model to produce an output sequence $y_1, \ldots, y_m \in \mathcal{Y}$. We normally pose this as a probabilistic problem and model the conditional probabilities, so that we wish to find

$$\begin{aligned} &\arg\max_{y_1,\ldots,y_n \in \mathcal{Y}} p(y_1, \ldots, y_m | x_1, \ldots, x_n) \\ =&\arg\max_{y_1,\ldots,y_n} p(y_1|x_1, \ldots, x_n) p(y_2|y_1, x_1, \ldots, x_n) \cdots p(y_m|y_1, \ldots, y_{m-1}, x_1, \ldots, x_n) \end{aligned} \tag{3.1}$$

The sequence-to-sequence architecture (Sutskever et al., 2014), also known as the encoder-decoder architecture, neatly provides a solution to this framework. By encoding the input $x_1, \ldots, x_n$ into a fixed size vector which we call the *context vector*, we can compute the conditional probabilities and hence generate $y_1, \ldots, y_m$ conditioned on this context vector.

The model has been used to great effect in a variety of NLP tasks, including machine translation (Sutskever et al., 2014; Bahdanau et al., 2014), question answering (Hermann

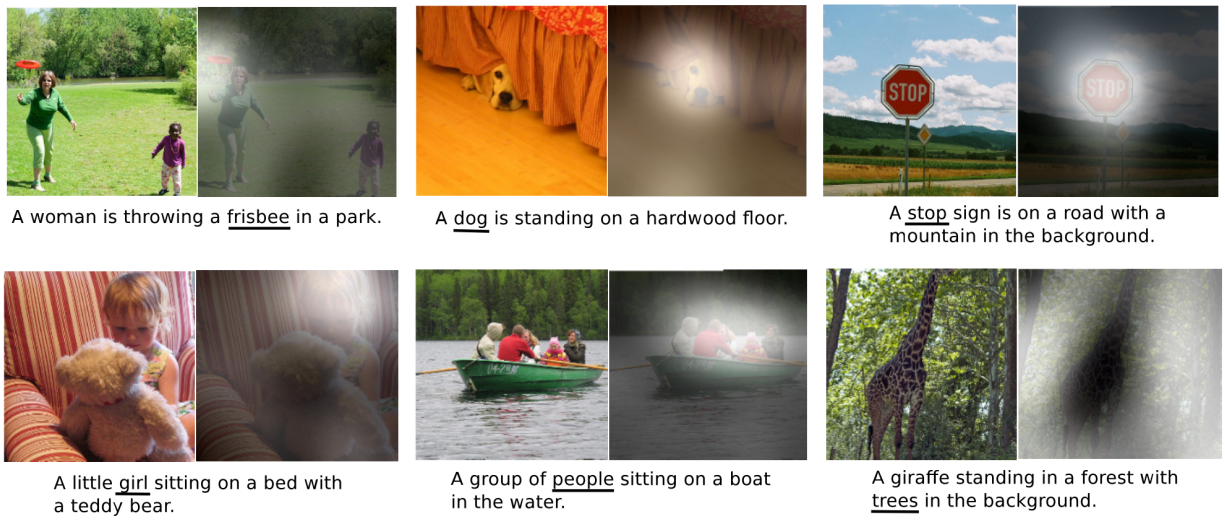


Figure 3.1: Attention of a caption generating neural model (Xu et al., 2015). The model learns to highlight the correct object when generating words.

et al., 2015), dialogue (Li et al., 2016a), caption generation (Xu et al., 2015), and in particular summarization (Rush et al., 2015).

A popular variant of sequence-to-sequence models are *attention* models (Bahdanau et al., 2014). The key idea is to keep an encoded representation of all parts of the input, attending to the relevant part each time we produce an output from the decoder. If we have a representation $\mathbf{h}_i \in \mathbb{R}^d$ for input token x_i, we model this attention process by computing weights $\alpha_1, \ldots, \alpha_n$ such that $\sum_{i=1}^n \alpha_i = 1$. The resulting context vector is then the weighted average $\sum_{i=1}^n \alpha_i \mathbf{h}_i$.

Xu et al. (2015) show how attention models can be used to "summarize" an image and produce a caption. By analyzing where in the image their models attend to when generating each word of the caption, i.e. where α_i is highest in the image, they qualitatively find that the model is essentially describing an object of the image. Figure 3.1 shows some examples.

While successful, existing seq2seq methods are limited by the length of source and target sequences. For a problem such as document summarization, the source sequence of length N requires $O(N)$ model computations to encode, where N could potentially be very large. However, it makes sense intuitively that not every word of the document will be necessary for generating a summary, and so we would like to reduce the amount of necessary computations over the source document.

Therefore, in order to scale seq2seq methods for this problem, we aim to prune down

the length of the source sequence in an intelligent way. The natural solution is to force the model to only use a subset of the input rather than naively encoding the entire input. We investigate some related work in this area.

3.2 Conditional Computation

Many techniques have been proposed in the literature to handle the problem of large inputs to deep neural networks.

The term "conditional computation" was coined by Bengio et al. (2013), where the idea is to compute a subset of units for a given network per input. This would have the advantage of being highly efficient, especially for networks that handle extremely large inputs as is common in vision and NLP.

Conditional computation in practice is usually implemented through the use of discrete units — these can serve as gates to select certain parts of the network for computation.

needs some reorganizing

Unfortunately, discrete variables cannot be backpropagated through as they are either not differentiable or produce zero gradient. Several techniques have been proposed to get around this problem.

Bengio et al. (2013) propose the *straight-through* estimator for binary stochastic gates. We simply sample from the stochastic gates in the forward step, and backpropagate the gradient as if we had not sampled. While this works empirically for simple binary gates, it is theoretically unjustified.

In general, we may want to sample from multinomial distributions, i.e. select one choice out of multiple choices. In deep networks, the softmax function $\text{softmax} : \mathbb{R}^m \to [0,1]^m$ is used to produce a normalized probability distribution out of any vector of reals. For $\mathbf{z} \in \mathbb{R}^m$,

$$\text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \tag{3.2}$$

Rae et al. (2016) use an approximate nearest neighbors approach for their "sparse access memory" model to train a large-scale neural Turing machine. Shazeer et al. (2017) introduce a mixture-of-experts model that selectively chooses a subset of "expert" networks at any given time during training. In the spirit of conditional computation, they only train K experts at a time using a sparse gating function.

Martins and Astudillo (2016) propose an alternative to softmax called the *sparsemax* function. For a given $\mathbf{z}$, sparsemax projects the point to the probability simplex. It turns out that this function has a useful gradient while having a sparse output. The downside is that we are not guaranteed to have a one-hot vector as we get from sampling the multinomial distribution.

Maddison et al. (2017) apply a smoothed version of the Gumbel max trick in order to approximate the sampling process. The Gumbel max trick is an alternative method for sampling multinomial random variables: by drawing i.i.d. uniform variables $U_i \sim \text{Unif}(0, 1)$, taking the argmax of $z_i - \log(-\log(U_i))$ gives us a sample with the correct probability. While this process is still discrete and has no gradient, Maddison et al. (2017) use a softmax instead of a max to obtain a smooth approximation that can be differentiated.

Reinforcement learning has also been proposed as an approach to handling discrete units in a network. Xu et al. (2015) suggest "hard" attention as one possible discrete selection mechanism. While standard "soft" attention averages the representations of where the model attends to, for hard attention we make a hard decision and choose only one location. To train such models, we can use reinforcement learning.

In the next section, we briefly overview reinforcement learning and explain how it can be applied to train the hard attention model.

3.3 Reinforcement Learning

Standard backpropagation training of neural networks assumes that the output is a deterministic and differentiable function of the input. Reinforcement learning, however, is a more general framework that makes no such assumptions.

The traditional setup of reinforcement learning (RL) assumes some agent is navigating an environment and earning rewards. We assume a state space $\mathcal{S}$, an action space $\mathcal{A}$, a reward function of state and action $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and a Markovian transition distribution $p(s'|s, a)$ for $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$.

We suppose that at time t, the agent is in state $s_t \in \mathcal{S}$, makes an action $a_t \in \mathcal{A}$, earns a reward $r_t = R(s_t, a_t)$, and transitions probabilistically to the next state s_{t+1}. Assuming the reward function is unknown, the agent wants to maximize total expected reward

$$\mathbb{E}_{s_t, r_t}[\sum_{t=0}^{\infty} \gamma^t r_t]$$

by finding an optimal action policy, i.e. finding the optimal policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ for states $s \in \mathcal{S}$. Here, γ is a time discount factor for the reward.

In general RL, we assume that we don't know the reward function $R(s, a)$ and we don't know the transition distribution $p(s_{t+1}|s_t, a_t)$. While the environment still gives us rewards for actions, finding the optimal policy requires predicting which states lead to those rewards. Since we don't know ahead of time which states are best, we must explore the state space to find what actions lead to the best rewards.

There are several methods for solving the general RL problem. Some are model-based, which attempt to model the transition distribution $p(s_{t+1}|s_t, a_t)$; others are model-free, which directly attempt to learn actions based on states. One such is Q-learning (Watkins, 1989), which estimates the reward for every pair $(s, a) \in \mathcal{S} \times \mathcal{A}$.

Another method is to learn a direct policy function $\pi(\cdot; \theta) : \mathcal{S} \rightarrow \mathcal{A}$, where π is parametrized by weights θ. We can train π to maximize expected reward through a gradient ascent method known as *policy gradient*, or the REINFORCE algorithm (Williams, 1992).

It turns out that REINFORCE can be used to train deep neural networks with stochastic units by a simple extension of the backpropagation algorithm. In the next chapter, we connect reinforcement learning and deep learning, and we derive the training algorithm for these deep networks.

Chapter 4

Algorithms

While reinforcement learning is an attractive framework for posing our models, the details of training become more complicated in the context of deep learning. In this chapter, we describe how both deep learning and reinforcement learning both fit into the rigorous model of stochastic computation graphs. We then give the corresponding training algorithm in detail.

4.1 Stochastic Computation Graphs

Neural networks with stochastic units are also known as *stochastic computation graphs*, as defined by Schulman et al. (2015). Formally, they define a stochastic computation graph (SCG) as a directed, acylic graph with three kinds of nodes:

1. Input nodes, including fixed network inputs and parameters.
2. Deterministic nodes, which are deterministic functions of their parents.
3. Stochastic nodes, which are random variables distributed conditionally on their parents.

We can formulate a training problem for SCGs by choosing certain terminal nodes and taking their sum as the objective function. If a node is stochastic, we take its expectation. That is, for a set of terminal nodes $\mathcal{C}$, we optimize $\sum_{c \in \mathcal{C}} \mathbb{E}[\hat{c}]$, where $\hat{c}$ denotes the random variable corresponding to node c.

Note that the SCG formulation captures both supervised deep learning and RL policy functions. Deep neural networks are just SCGs with all deterministic nodes, and the loss

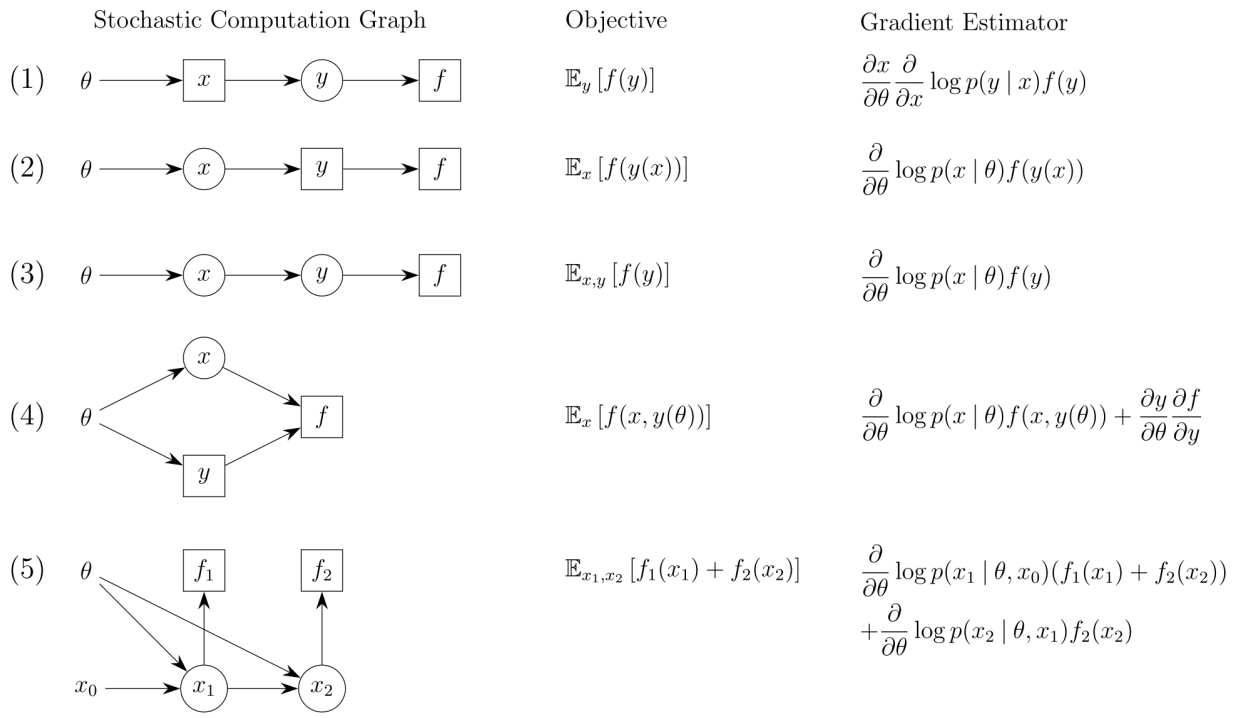


Figure 4.1: Simple examples of stochastic computation graphs (SCGs), along with the objective and corresponding gradient estimator. Nodes in circles are stochastic, nodes in squares are deterministic, and the rest are input nodes. Diagrams from Schulman et al. (2015).

function $\mathcal{L}$ of the task is the objective. The RL policy function is an SCG with actions at the stochastic nodes, and the expected reward $\mathbb{E}[r]$ is the objective. As we will see, SCGs will turn out to be a generalization of the REINFORCE algorithm.

Figure 4.1 shows examples of simple SCGs, as well as the estimator of the objective gradient for the parameter θ. Using the estimated gradient, we optimize the objective function using gradient descent.

For these special cases, notice that for all stochastic nodes, the gradient estimator includes a term of the form $\partial \log p(-|-)/\partial\theta$. On the other hand, deterministic nodes lead to derivatives as in the chain rule from backpropagation.

Next, we derive the gradient of the objective with respect to parameter input nodes in the general case.

We define for nodes w, v the relation $w \prec v$ (pronounced w influences v) if there exists a path from w to v in the SCG. We also say $w \prec^D v$ (w deterministically influences v) if there exists a path that consists of only deterministic nodes.

Given an SCG, let $\mathcal{S}$ be the set of stochastic nodes, $\mathcal{C}$ the set of cost nodes that give the

objective. For a given node v, let $\text{par}(v)$ denote its parents.

For a given parameter θ, we have the following:

Theorem 1. *Assume differentiability of all functions. Then the gradient of the objective with respect to θ is*

$$\frac{\partial}{\partial\theta}\mathbb{E}\left[\sum_{c\in\mathcal{C}}\hat{c}\right] = \mathbb{E}\left[\sum_{c\in\mathcal{C}}\hat{c}\sum_{v\in\mathcal{S},\ \theta\prec^D v,\ v\prec c}\frac{\partial}{\partial\theta}\log p(v|\,\text{par}(v);\theta) + \sum_{c\in\mathcal{C},\ \theta\prec^D c}\frac{\partial}{\partial\theta}\hat{c}\right] \tag{4.1}$$

$$= \mathbb{E}\left[\sum_{v\in\mathcal{S},\ \theta\prec^D v}\left(\frac{\partial}{\partial\theta}\log p(v|\,\text{par}(v);\theta)\right)\hat{C}_v + \sum_{c\in\mathcal{C},\ \theta\prec^D c}\frac{\partial}{\partial\theta}\hat{c}\right] \tag{4.2}$$

where $\hat{C}_v = \sum_{c\in\mathcal{C},\ v\prec c}\hat{c}$ is a random variable of the cost that stochastic node v influences.

Before we give the proof, we note the intuition. The first term represents the gradient from the stochastic nodes, where we multiply the gradient of the log probability with the cost. The second term is the standard gradient we obtain from backpropagation.

Proof. Due to linearity of expectation, we only need consider a single node $c \in \mathcal{C}$. Let $\mathcal{L} = \mathbb{E}[\hat{c}]$ be the loss function.

We have stochastic nodes v such that $\theta \prec^D v$ and $v \prec c$. Let this set be $\mathcal{S}_\theta$, and denote the joint random variable of this set as $\hat{\mathbf{v}}$. Let $c(\hat{\mathbf{v}};\theta)$ explicitly denote $\hat{c}$.

We compute the gradient:

$$\begin{aligned}
\frac{\partial}{\partial\theta}\mathbb{E}[c] &= \frac{\partial}{\partial\theta}\sum_{\hat{\mathbf{v}}} p(\hat{\mathbf{v}}|\,\text{par}(\mathbf{v});\theta)\cdot c(\hat{\mathbf{v}};\theta) \\
&= \sum_{\hat{\mathbf{v}}}\frac{\partial p(\hat{\mathbf{v}}|\,\text{par}(\mathbf{v});\theta)}{\partial\theta}\cdot c(\hat{\mathbf{v}};\theta) + p(\hat{\mathbf{v}}|\,\text{par}(\mathbf{v});\theta)\cdot\frac{\partial}{\partial\theta}c(\hat{\mathbf{v}};\theta) \\
&= \sum_{\hat{\mathbf{v}}}\frac{\partial p(\hat{\mathbf{v}}|\,\text{par}(\mathbf{v});\theta)}{\partial\theta}\cdot c(\hat{\mathbf{v}};\theta) + \mathbb{E}_{\mathbf{v}}\left[\frac{\partial}{\partial\theta}c(\hat{\mathbf{v}};\theta)\right]
\end{aligned}$$

Note that the second term gives the standard backpropagation gradient. We can rewrite

the first term:

$$\begin{aligned}
\sum_{\hat{\mathbf{v}}} \frac{\partial p(\hat{\mathbf{v}}|\operatorname{par}(\mathbf{v});\theta)}{\partial\theta} \cdot c(\hat{\mathbf{v}};\theta) &= \sum_{\hat{\mathbf{v}}} p(\hat{\mathbf{v}}|\operatorname{par}(\mathbf{v});\theta) \frac{1}{p(\hat{\mathbf{v}}|\operatorname{par}(\mathbf{v});\theta)} \frac{\partial p(\hat{\mathbf{v}}|\operatorname{par}(\mathbf{v});\theta)}{\partial\theta} \cdot c(\hat{\mathbf{v}};\theta) \\
&= \sum_{\hat{\mathbf{v}}} p(\hat{\mathbf{v}}|\operatorname{par}(\mathbf{v});\theta) \frac{\partial \log p(\hat{\mathbf{v}}|\operatorname{par}(\mathbf{v});\theta)}{\partial\theta} \cdot c(\hat{\mathbf{v}};\theta) \\
&= \mathbb{E}_{\hat{\mathbf{v}}\sim p(\hat{\mathbf{v}}|\operatorname{par}(\mathbf{v});\theta)} \left[\frac{\partial \log p(\hat{\mathbf{v}}|\operatorname{par}(\mathbf{v});\theta)}{\partial\theta} \cdot c(\hat{\mathbf{v}};\theta) \right] \\
&= \mathbb{E}\left[\widehat{c} \cdot \sum_{v\in\mathcal{S}_\theta} \frac{\partial \log p(\hat{v}|\operatorname{par}(v);\theta)}{\partial\theta} \right]
\end{aligned}$$

where the last equality follows from separating the log joint probability term into its conditional marginals.

We have thus shown Equation 4.1. Equation 4.2 follows directly by rearranging the order of summations, noting which nodes v influence a certain c.

□

Because the expectation in both gradient formulae are intractable in the general case, we use a single Monte Carlo sample. In practice, however, the variance of the Monte Carlo gradient can be very high.

One of the simplest ways to reduce the variance of the gradient estimator is to introduce a baseline scalar $b \approx \mathbb{E}[\hat{c}]$ which we subtract from each cost term. That is, we have:

Theorem 2. *The gradient from Theorem 1 can also be written as*

$$\frac{\partial}{\partial\theta}\mathbb{E}\left[\sum_{c\in\mathcal{C}} \hat{c}\right] = \mathbb{E}\left[\sum_{c\in\mathcal{C}} (\hat{c} - b_c) \sum_{v\in\mathcal{S},\ \theta\prec^D v,\ v\prec c} \frac{\partial}{\partial\theta} \log p(v|\operatorname{par}(v);\theta) + \sum_{c\in\mathcal{C},\ \theta\prec^D c} \frac{\partial}{\partial\theta}\hat{c}\right] \quad (4.3)$$

where b_c is a scalar not influenced by any of the v in the summation.

Proof. The proof boils down to a trick with the log probability. Considering a single c, by linearity it suffices to show that

$$\mathbb{E}\left[\frac{\partial}{\partial\theta} \log p(v|\operatorname{par}(v);\theta)\right] = 0, \qquad \forall v \in \mathcal{S}_\theta$$

This is true since the b_c term is constant with respect to the expectation (by our assumption that no v influences it).

But this follows since

$$\begin{aligned}\mathbb{E}\left[\frac{\partial}{\partial\theta}\log p(\hat{v}|\operatorname{par}(v);\theta)\right] &= \sum_{\hat{v}} p(\hat{v}|\operatorname{par}(v);\theta)\frac{\partial \log p(\hat{v}|\operatorname{par}(v);\theta)}{\partial\theta} \\ &= \sum_{\hat{v}} p(\hat{v}|\operatorname{par}(v);\theta)\frac{1}{p(\hat{v}|\operatorname{par}(v);\theta)}\frac{\partial p(\hat{v}|\operatorname{par}(v);\theta)}{\partial\theta} \\ &= \frac{\partial}{\partial\theta}\sum_{\hat{v}} p(\hat{v}|\operatorname{par}(v);\theta) = \frac{\partial}{\partial\theta}[1] = 0\end{aligned}$$

Thus, the gradient with baseline is unbiased. □

Including a baseline is proven to reduce the variance of the estimator (Weaver and Tao, 2001). There are several different methods for producing baselines, such as taking the average of all previously seen cost terms, and we will not cover all of them here.

4.1.1 Training

In order to compute the gradients of SCGs in practice, we make a slight tweak to the backpropagation algorithm.

We first perform a forward pass of the SCG to obtain all values and samples for each node. In the backward pass, gradients for nodes that are directly connected to the cost nodes can be computed with usual backpropagation. For stochastic nodes, we first compute the costs $\hat{C}_v$ for node v in Equation 4.2. Broadcasting these values to the corresponding nodes, we can then continue backpropagation from the stochastic nodes with the gradient of the log probability.

Finally, if g_θ is the resulting gradient for parameter θ, our gradient descent update is

$$\theta \leftarrow \theta - \eta g_\theta \tag{4.4}$$

where η is a learning rate.

Algorithm 1 gives the complete algorithm for gradient descent on SCGs.

While SCGs are a very useful and general framework, the language of reinforcement learning is more intuitive and easy to understand.

We will be most interested in the sequential decision problem of RL. That is, suppose we have a trajectory of states s_t and we make actions a_t at time t based on some parameterized policy function. Each action leads to a reward r_t, and influences total future reward $\sum_{s=t}^{T} r_s$ where $T < \infty$ is the time horizon.

Algorithm 1 Gradient Descent for SCGs

Forward pass through SCG
for all $c \in \mathcal{C}$ **do**
  $\hat{c} \leftarrow \hat{c} - b_c$ ▷ Subtract baselines from costs
end for
for all $v \in$ SCG **do**
  $\mathbf{g}_v = \begin{cases} 1 & \text{if } v \in \mathcal{C} \\ 0 & \text{else} \end{cases}$
  Compute $\hat{C}_v \leftarrow \sum_{c \in \mathcal{C},\, v \prec c} \hat{c}$ ▷ Aggregated costs
end for
for v in REVERSETOPOLOGICALORDER(SCG) **do**
  for $w \in \mathrm{par}(v)$ **do**
    if not ISSTOCHASTIC(w) **then**
      if ISSTOCHASTIC(v) **then**
        $\mathbf{g}_w$ += $\hat{C}_w \cdot \frac{\partial}{\partial w} \log p(v | \mathrm{par}(v))$
      else
        $\mathbf{g}_w$ += $(\frac{\partial v}{\partial w})^\top \mathbf{g}_v$
      end if
    end if
  end for
end for
for all parameters θ **do**
  $\theta \leftarrow \theta - \eta g_\theta$ ▷ Gradient descent
end for

Note that we can treat our actions a_t for $t = 0, \ldots, T$ as stochastic nodes in an SCG, and the costs $\hat{c}$ are the rewards we receive for each t. Then instead of computing $\hat{C}_{a_t} = \sum_{s=t}^{T} r_s$, we can scale later rewards with a discount factor γ, giving

$$\hat{C}_{a_t} = \sum_{s=t}^{T} \gamma^{s-t} r_s \tag{4.5}$$

which fits neatly in with Algorithm 1. We will use this RL setup to discuss our models, and our algorithms are confidently backed up by the theory of SCGs.

diagram?

4.2 Reinforcement Learning in NLP

needs some organizing

Even before the concept of SCGs were formalized, reinforcement learning and deep learning have successfully been combined to play Go (Silver et al., 2016), control robots (Levine et al., 2016), and play Atari games from pixels (Mnih et al., 2015). In computer vision, policy gradient methods have been fairly successful on several tasks (Mnih et al., 2014; Ba et al., 2015; Xu et al., 2015).

In NLP, reinforcement learning research is still preliminary, but has been attempted with varying degrees of success so far. Ranzato et al. (2015) apply RL to improve sequential estimation by e.g. training on BLEU rewards for machine translation. Narasimhan et al. (2015) use RL play text-based games. Li et al. (2016b) use RL to train an end-to-end dialogue system. Narasimhan et al. (2016) use RL to improve information extraction on scarce data. Yogatama et al. (2017) use RL along with an architecture called a tree LSTM that produces a tree-structured latent variable on a sentence using SHIFT and REDUCE actions. They show that the tree LSTM can produce reasonable parse structures by solely using RL methods.

Inspired by these recent works, we are interested to see if reinforcement learning can be applied to the document summarization problem. We will introduce stochastic nodes into a deep learning model and investigate the feasibility of using policy gradient methods for conditional computation.

In the next chapter we describe our models in detail.

Chapter 5

Models

In this chapter, we describe our models. We begin by introducing the standard sequence-to-sequence attention model, then describe our extensions of the basic architecture.

5.1 Sequence-to-sequence (seq2seq)

We first describe the neural network architecture of the seq2seq models, also known as encoder-decoder models (Bahdanau et al., 2014).

In the seq2seq model, an *encoder* recurrent neural network (RNN) reads the source sequence as input to produce the *context*, and a *decoder* RNN generates the output sequence using the context as input. One popular RNN choice is the long-short term memory (LSTM) network (Hochreiter and Schmidhuber, 1997).

More formally, suppose we have a vocabulary $\mathcal{V}$. A given input sentence $w_1, \ldots, w_n \in \mathcal{V}$ is transformed into a sequence of vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^{d_{in}}$ through a word embedding matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{V}| \times d_{in}}$ as $\mathbf{x}_t = \mathbf{E}w_t$.

An RNN is given by a parametrizable function f_{enc} and a hidden state $\mathbf{h}_t \in \mathbb{R}^{d_{hid}}$ at each time step t with $\mathbf{h}_t = f_{enc}(\mathbf{x}_t, \mathbf{h}_{t-1})$. For the LSTM, we keep an auxiliary state $\mathbf{c}_t$

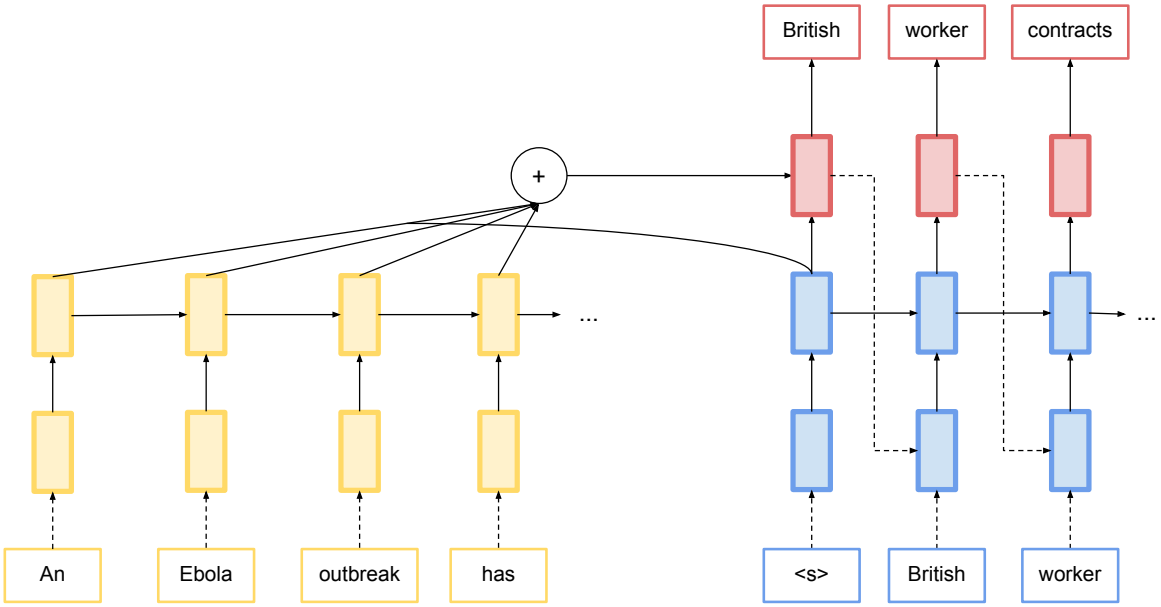


Figure 5.1: Model architecture for sequence-to-sequence with attention, or MODEL 0. *The yellow hidden states are the encoder, the blue hidden states are the decoder, and the red hidden states are the generator. The decoder hidden state at each time step determines the attention weights, which we use to average the encoder hidden states to produce a context vector. The result feeds into the generator.*

along with $\mathbf{h}_t$, and we compute f_{enc} as

$$\mathbf{f}_t = \sigma(\mathbf{W}^f \mathbf{x}_t + \mathbf{U}^f \mathbf{h}_t + b_f) \tag{5.1}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}^i \mathbf{x}_t + \mathbf{U}^i \mathbf{h}_t + b_i) \tag{5.2}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}^o \mathbf{x}_t + \mathbf{U}^o \mathbf{h}_t + b_o) \tag{5.3}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}^c \mathbf{x}_t + \mathbf{U}^c \mathbf{h}_{t-1} + b_c) \tag{5.4}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \tag{5.5}$$

where $\mathbf{W}, \mathbf{U}, b$ are learned parameters, and $\odot$ is the elementwise product. Intuitively, $\mathbf{c}_t$ is the memory cell, $\mathbf{f}_t$ is the forget gate, $\mathbf{i}_t$ is the input gate, $\mathbf{h}_t$ is the output cell, and $\mathbf{o}_t$ is the output gate.

LSTMs can be stacked on top of one another by treating the outputs $\mathbf{h}_t$ of one LSTM as the inputs to another LSTM.

In stacked LSTMs, we will take the top sequence of hidden states $\mathbf{h}_1, \ldots, \mathbf{h}_n$ to form a

single context vector.

The decoder is another RNN f_{dec} that generates output words $y_t \in \mathcal{V}$. It keeps hidden state $\mathbf{h}_t^{dec} \in \mathbb{R}^{d_{hid}}$ as $\mathbf{h}_t^{dec} = f_{dec}(y_{t-1}, \mathbf{h}_{t-1}^{dec})$ similar to the encoder RNN. A context vector is produced at each time step using an attention function a that takes the encoded hidden states $[\mathbf{h}_1, \ldots, \mathbf{h}_n]$ and the current decoder hidden state $\mathbf{h}_t^{dec}$ and produces the context $\mathbf{c}_t \in \mathbb{R}^{d_{ctx}}$:

$$\mathbf{c}_t = a([\mathbf{h}_1, \ldots, \mathbf{h}_n], \mathbf{h}_t^{dec}) \tag{5.6}$$

As in Luong et al. (2015), it is helpful to feed the context vector at time $t-1$ back into the decoder RNN at time t, i.e. $\mathbf{h}_t^{dec} = f_{dec}([y_{t-1}, \mathbf{c}_{t-1}], \mathbf{h}_{t-1}^{dec})$.

Finally, a linear projection produces a distribution over output words $y_t \in \mathcal{V}$:

$$p(y_t | y_{t-1}, \ldots, y_1, [h_1, \ldots, h_n]) = \mathbf{W}^{out}\mathbf{c}_t + b^{out} \tag{5.7}$$

Given document-summary pairs $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N$ for training, the models are trained to maximize the log probability of getting the sequences in the dataset correct, i.e. minimize the negative log-likelihood (NLL):

$$\begin{aligned}\mathcal{L}(\theta) &= -\sum_{i=1}^{N} \log p(\mathbf{y}^{(i)} | \mathbf{x}^{(i)}; \theta) \\ &= -\sum_{i=1}^{N} \sum_{t=1}^{T-1} \log p(y_{t+1}^{(i)} | \mathbf{x}^{(i)}, y_t^{(i)}, \ldots, y_1^{(i)}; \theta)\end{aligned}$$

As the model is fully differentiable with respect to its parameters, we can train it end-to-end with stochastic gradient descent and the backpropagation algorithm.

We note that we have great flexibility in how our attention function $a(\cdot)$ combines the encoder context and the current decoder hidden state. In the next few sections, we explain standard choices for $a(\cdot)$ as well as our proposed model of *coarse-to-fine attention*.

5.2 Model 0: Standard Attention

In Bahdanau et al. (2014), the function $a(\cdot)$ is implemented with an *attention network*. We compute attention weights for each encoder hidden state h_i as follows:

$$\beta_{t,i} = \mathbf{h}_i^\top \mathbf{W}^{attn} \mathbf{h}_t^{dec} \quad \forall i = 1, \ldots, n \tag{5.8}$$

$$\boldsymbol{\alpha}_t = \text{softmax}(\boldsymbol{\beta}_t) \tag{5.9}$$

$$\widetilde{\mathbf{c}}_t = \sum_{i=1}^{n} \alpha_{t,i} \mathbf{h}_i \tag{5.10}$$

The idea behind attention is to select the most relevant words of the source (by assigning higher attention weights) when generating output word y_t at time t.

The softmax function, defined as

$$\text{softmax}([\beta_1, \ldots, \beta_n])_i = \frac{\exp(\beta_i)}{\sum_{j=1}^{n} \exp(\beta_j)} \tag{5.11}$$

normalizes the α_i to sum to 1 over the source sentence words. This gives us a notion of probability distribution over the encoder words — we can therefore write $\mathbf{c}_t$ as the expectation $\mathbb{E}_\alpha[\mathbf{h}]$, where we pick $\mathbf{h}_i$ with probability α_i.

Our final context vector is then

$$\mathbf{c}_t = \tanh(\mathbf{W}^2[\widetilde{\mathbf{c}}_t, \mathbf{h}_t^{dec}]) \tag{5.12}$$

for $\mathbf{W}^2 \in \mathbb{R}^{2d_{hid} \times d_{ctx}}$ a learned matrix.

Going forward, we call this instantiation of the attention function MODEL 0.

5.3 Model 1 and 2: Coarse-to-Fine Soft Attention

We can also leverage sequence-to-sequence for text summarization. However, the attention step becomes computationally expensive — for each word we generate, we need to compare it to every word of the document in order to determine which part to attend to. Intuitively, we should only need to attend to a few important sentences of the document instead of all of it. Therefore, we propose a hierarchical method of attending to the document by first attending to sentences, then to the words within sentences. We call this

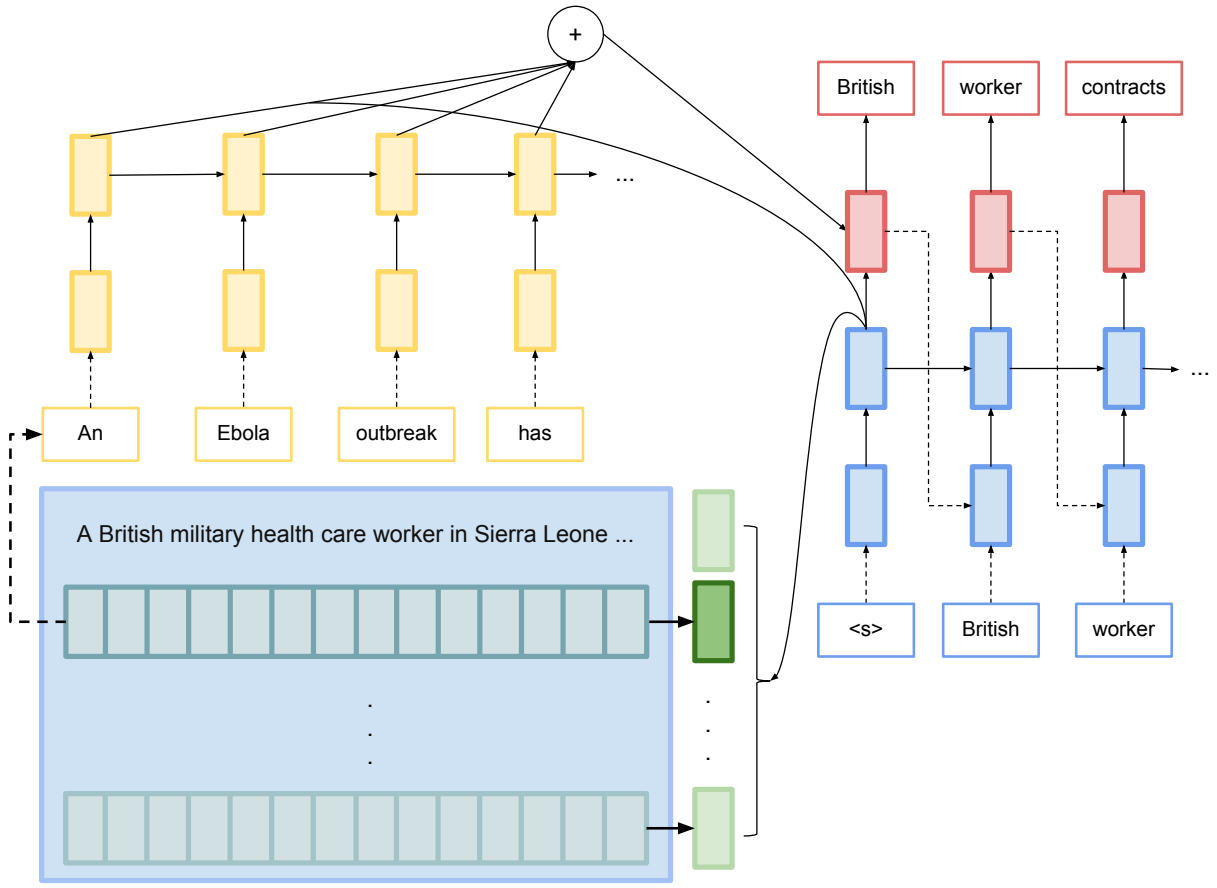


Figure 5.2: Model architecture for sequence-to-sequence with coarse-to-fine attention. The left side is the encoder that reads the document, and the right side is the decoder that produces the output sequence. On the encoder side, the green hidden states (sentence-level) are used for the coarse attention weights, while the yellow hidden states (word-level) are used for the fine attention weights. The context vector is then produced by averaging the word-level states. In MODEL 2*, we average over the coarse attention weights, thus requiring computation of all word-level hidden states. In* MODEL 3*, we make a hard decision for which sentence to use, and so we only need to compute word-level hidden states for one sentence.*

method *coarse-to-fine attention* [1].

To be able to attend to both sentences and words in a hierarchical manner, we need to construct encodings of the document at both levels. For the coarse-grained sentence representations, we use a simple encoding model (e.g. bag of words), and for the fine-grained representations, we run an LSTM encoder on the words of a sentence. Sukhbaatar et al. (2015) demonstrate how coarse representations can be useful by using memory networks to access information for simple question-answering tasks. Li et al. (2015) use the idea of a hierarchical representation of text to develop a two-layer LSTM autoencoder for paragraph representation.

Therefore, if we can make our model first use coarse attention to choose sentences, then use fine attention to choose words only from that sentence, then we avoid the computational cost of searching over the entire document.

Specifically, suppose we have sentences $s_1, \ldots, s_m$ with words $w_{i,1}, \ldots, w_{i,n_i}$ for sentence s_i. We apply an RNN to each sentence separately to get corresponding hidden states $\mathbf{h}_{i,j}$ for $i = 1, \ldots, m$ and $j = 1, \ldots, n_i$, so that

$$\mathbf{h}_{i,j} = \text{RNN}(\mathbf{h}_{i,j-1}, w_{i,j}) \tag{5.13}$$

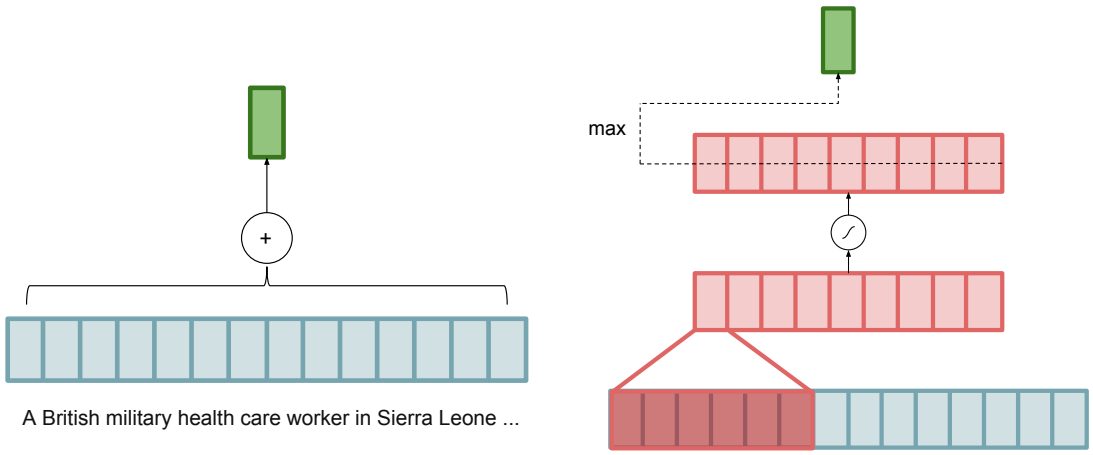
For attention, we then consider two options.

Model 1 We can follow MODEL 0 and compute attention weights $\alpha_{i,j}$ for each hidden state $\mathbf{h}_{i,j}$ by normalizing over all states. We call this MODEL 1.

Model 2 Alternatively, rather than taking attention over the entire document, we can instead have a two-layered hierarchical attention mechanism: first, we have weights $\alpha^s_1, \ldots, \alpha^s_m$ for each sentence, and then for sentence s_i, we have another set of weights $\alpha^w_{i,1}, \ldots, \alpha^w_{i,n_i}$. Our final attention weight on word $w_{i,j}$ is then $\alpha_{i,j} = \alpha^s_i \cdot \alpha^w_{i,j}$.

In order to compute the sentence attention weights α^s_i, we need to produce representations of each sentence; i.e., given the words $w_{i,1}, \ldots, w_{i,n_i}$ of the sentence, we produce a vector representation $\mathbf{h}^s_i \in \mathbb{R}^{d_{sent}}$.

[1] The term coarse-to-fine attention has previously been introduced in the literature (Mei et al., 2016). However, their idea is different: they use coarse attention to reweight the fine attention computed over the entire input. This idea has also been called hierarchical attention (Nallapati et al., 2016).



(a) Bag-of-words

(b) Convolution

Figure 5.3: Options for producing sentence representations from the word embeddings. We can either use a bag of words by summing the embeddings, or apply convolutions and max-over-time pooling.

Our first option is bag of words: we simply take the representation as

$$\mathbf{h}_i^s = \sum_{j=1}^{n_i} \mathbf{E} w_{i,j} \tag{5.14}$$

i.e. the sum of the word embeddings, where $\mathbf{E}$ is another embedding matrix.

Alternatively, we can use a convolutional method: as in Kim (2014), we perform a convolution over each window of words in the sentence using a fixed kernel width. We use max-over-time pooling to obtain a fixed-dimensional sentence representation in $\mathbb{R}^{d_f}$ where d_f is the number of filters.

Explicitly, fix a sentence and suppose we have word vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ with $\mathbf{x}_j = \mathbf{E} w_j \in \mathbb{R}^{d_{in}}$, and suppose we have kernel width k and convolution weights $\mathbf{W}^{conv} \in \mathbb{R}^{d_f \times d_{in} k}$ where d_f is the number of filters. Then applying the convolution $\mathbf{W}^{conv} * [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ gives result $\mathbf{u} = [\mathbf{u}_1, \ldots, \mathbf{u}_{n-k+1}]$ with jth element

$$\mathbf{u}_j = \mathbf{W}^{conv} \cdot [\mathbf{x}_j, \mathbf{x}_{j+1}, \ldots, \mathbf{x}_{j+k-1}] + \mathbf{b}^{conv} \in \mathbb{R}^{d_f}$$

Our final output is given by

$$\mathbf{h}^s = \max_j(\tanh(\mathbf{u}_j)) \tag{5.15}$$

where the max-over-time takes the maximum along the word indexing dimension. See Figure 5.3 for diagrams of the sentence representation models.

Thus, using the sentence representations, we can compute attention over the sentences. For the words in each sentence, we run an LSTM over each sentence separately, and create attention weights over each sentence in the same way as MODEL 0. Using attention on word $w_{i,j}$ as $\alpha_{i,j} = \alpha_i^s \cdot \alpha_{i,j}^w$, we can proceed exactly as in MODEL 0 by computing the weighted average over hidden states $\mathbf{h}_{i,j}$.

We call this method of attention MODEL 2.

5.4 Model 3: Coarse-to-Fine Sparse Attention

With the previous models, we are required to compute hidden states over all words and sentences in the document, so that if we have M sentences and N words per sentence, the computational complexity is $O(MN)$ for each attention step.

However, if we are able to perform conditional computation and only compute on M^+ of the sentences, we can reduce the complexity to $O(M^+N)$. If we are able to make M^+ constant or even 1, this would give significant benefits to our overall complexity.

In our experiments, we will apply stochastic sampling to the attention distribution $\boldsymbol{\alpha}$ in the spirit of Xu et al. (2015) and "hard attention". Specifically, rather than computing the context $\widetilde{\mathbf{c}}$ as an expectation over $\boldsymbol{\alpha}$ (i.e. $\mathbf{c} = \sum_{i=1}^n \alpha_i \mathbf{h}_i$), we can sample from the probability distribution $\boldsymbol{\alpha}$ to obtain a single state $\mathbf{h}_i$, and we set $\widetilde{\mathbf{c}} = h_i$ as the sampled hidden state.

In our case, we take MODEL 2 and apply hard attention at the sentence level, but keep the word level attention per sentence as is. That is, we sample from the attention weights $\alpha_1^s, \ldots, \alpha_m^s$ to obtain a one-hot encoding for the sentence attention, and apply the same multiplication with this one-hot vector on the word-level attention weights $\alpha_{i,1}^w, \ldots, \alpha_{i,n_i}^w$ for all $i = 1, \ldots, m$. We call this MODEL 3.

Because the hard attention model loses the property of being end-to-end differentiable, we use reinforcement learning to train our network.

5.4.1 Practical Considerations

Our hard attention model is now a stochastic computation graph, and thus Algorithm 1 from Section 4.1.1 applies.

Here, we have an RL agent where the state s_t is the LSTM decoder state at time t, and actions a_t are the hard attention decisions. Since samples from $\boldsymbol{\alpha}_t$ at time t of the RNN decoder can also affect future rewards, we use a discount factor of $\gamma = 0.5$, so that the reward is $\hat{C}_t = \sum_{s=t}^{n} \gamma^{n-s} r_s$ at time t, where $r_t = \log p(y_t | y_1, \ldots, y_{t-1}, \mathbf{x})$ is the single step reward.

To calculate the baselines for variance reduction, we store a constant b_t for each decoder time step t. We follow Xu et al. (2015) and keep an exponentially moving average of the reward for each time step t:

$$b_t \leftarrow b_t + \beta(r_t - b_t) \tag{5.16}$$

where r_t is the average minibatch reward and β is a learning rate (set to 0.1).

While several papers suggest using a learned baseline (e.g. Mnih et al., 2014; Ranzato et al., 2015), we have not found this to be effective. In our experiments, we found that attempting to learn the baseline failed to converge, most likely because there is not enough correlation between the reward and the hidden states preceding the attention layer.

In addition to including a baseline, we also scale the rewards by a tuned hyperparameter — we found that scaling helped to stabilize training. We empirically set this scale to 0.3.

Randomly training using soft attention Xu et al. (2015) explain that training hard attention with REINFORCE has very high variance, even when including a baseline. Thus, for every minibatch of training, they randomly use soft attention instead of hard attention with some probability (they use 0.5). The backpropagated gradient is then the standard soft attention gradient instead of the REINFORCE gradient.

While this method helps stabilize training, it's aesthetically not very pleasing. Our goal in coarse-to-fine attention is to reduce computation over the encoded hidden states, while this method requires that we perform the full amount of computation as soft attention for a random subset of minibatches. However, we include this training method in our experiments to test how feasible hard attention is.

Multiple samples From our initial experiments with MODEL 3, we found that taking a single sample was not very effective. However, we discovered that sampling multiple times from the distribution α improves performance.

To be precise, we sample based on the multinomial distribution $\text{Mult}(k_{mul}, \{\alpha_i\}_{i=1}^{n})$ to produce the sentence-level attention vector $\boldsymbol{\alpha}_t$ at time t, with $\alpha_i = x_i / k_{mul}$, where x_i is the ith entry of the multinomial sample. k_{mul} is a hyperparameter which can be tuned (standard hard attention is $k_{mul} = 1$).

Intuitively, k_{mul} is the number of sentences we select to produce the context. With higher k_{mul}, the hard attention model more closely approximates the soft attention model, and hence leads to better performance. This, however, incurs a cost in computational complexity.

5.5 Beam Search

Once we have our trained model, we use beam search at test time to produce the output summaries.

In beam search, we keep a beam of

Algorithm 2 Beam Search

hi

Chapter 6

Experiments

6.1 Data

6.1.1 CNN/Dailymail

Experiments were performed on the CNN/Dailymail dataset from Hermann et al. (2015). While the dataset was created for a question-answering task, the dataset format is suited for summary. Each data point is a news document accompanied by up to 4 "highlights", and we take the first of these as our target summary.

Train, validation, and test splits are provided along with document tokenization and sentence splitting. We do additional preprocessing by replacing all numbers with # and appending end of sentence tokens to each sentence. We limit our vocabulary size to 50000 most frequent words, replacing the rest with `<unk>` tokens. We dropped the documents which had an empty source (which came from photo articles).

Table 6.1 lists statistics for the CNN/Dailymail dataset. Figure 6.1 shows examples source and target pairs from the dataset.

In the context of these new datasets, the summarization task has not yet been fully standardized. Research in the area is still largely preliminary, with only a few papers reporting results (Nallapati et al., 2016, e.g.). While CNN/Dailymail may not be the most suitable dataset for the task due to its noisiness (Chen et al., 2016), a better alternative is yet to exist.

Figure 6.1: Examples of source and target for the CNN/Dailymail dataset. The source is a news article, and the target is one of the given highlights.

Dataset	CNN	Dailymail	Combined
Train size	90266	196961	287227
Valid size	1220	12148	13368
Test size	1093	10397	11490
Avg. # words per doc.	794	832	
Avg. # sent. per doc.	21	29	
Avg. # words per sent.	36	27	
Avg. # words per summary	13	15	

Table 6.1: Statistics for CNN/Dailymail data.

6.2 Implementation details

A few implementation details were necessary to make minibatch training possible. First, instead of taking attention over each individual sentence, we arrange the first 400 words of the document into a 10 by 40 image, and take each row to be a sentence.

Second, we pad short documents to the maximum length with a special padding word, and allow the model to attend to it. However, we zero out word embeddings for the padding states and also zero out their corresponding LSTM states. We found in practice that very little of the attention ended up on the padding words.

Ideally, we would prefer to not truncate documents, especially since later context can be useful for summarizing the document. Due to memory issues, this is a problem we still have to resolve.

6.3 Models

Baselines For a baseline, we take the first sentence of the document (chosen as the average length of a sentence in the training dataset). We call this FIRST.

We also consider the feature-based document summarizer of Durrett et al. (2016), which uses ILP methods to compress extracted sentences. We apply the code [1] directly on the test set without retraining the system. Their system requires that the documents are preprocessed in CONLL format, so we use the Berkeley coreference system[2] with the coreference and NER settings. We call this BERKELEY.

[1] https://github.com/gregdurrett/berkeley-doc-summarizer
[2] https://github.com/gregdurrett/berkeley-entity

Our models We ran experiments with Models 0 to 3 as described above.

- MODEL 0: Soft attention.
- MODEL 1: Organized by sentence, soft attention over all.
- MODEL 2: Hierarchical LSTM, coarse-to-fine with soft attention.
- MODEL 3: Hierarchical LSTM, coarse-to-fine with hard attention over sentences.

We also

6.4 Training

We train with minibatch stochastic gradient descent (SGD) with batch size 20 for 20 epochs, renormalizing gradients below norm 5. We initialize the learning rate to 0.1 for the sentence encoder and 1 for the rest of the model, and begin decaying it by a factor of 0.5 each epoch after the validation perplexity stops decreasing.[3]

We use 2 layer LSTMs with 500 hidden units, and we initialize word embeddings with 300-dimensional word2vec embeddings (Mikolov and Dean, 2013). For convolutional layers, we use a kernel width of 6 and 600 filters.

Our models are implemented using Torch (Collobert et al., 2011a) based on a past version of Harvard's OpenNMT system[4]. We ran our experiments on a GPU (specs?). The models take between 2-2.5 hours to train per epoch.

All of our code is available open source[5].

In the next chapter we show results.

[3]We tried more complicated SGD optimization methods such as Adagrad (Duchi et al., 2011) and Adam (Kingma and Ba, 2015), but found that they did not perform as well. This could be due to gradient norms that are too large.

[4]https://github.com/harvardnlp/seq2seq-attn

[5]https://github.com/jeffreyling/seq2seq-hard

Chapter 7

Results

7.1 Evaluation

We report metrics for perplexity and ROUGE scores (Lin, 2004) on the test set.

ROUGE-n computes n-gram overlap between a gold summary and a predicted summary, and ROUGE-L computes the longest common subsequence. We use ROUGE balanced F-scores and report numbers for ROUGE-1 (unigrams), ROUGE-2 (bigrams), and ROUGE-L. While ROUGE traditionally uses the recall metric, we choose F-score since recall is biased towards longer predicted sentences.

With multiple gold summaries in the CNN/Dailymail highlights, we choose to take the max ROUGE score over the gold summaries for a predicted summary, as our models are trained to produce a single sentence. The final metric is then the average over all test data points.

7.2 Analysis

See Table 7.1 for summary results.

We see hard attention takes longer to converge.

We investigate the entropy of the sentence attention on the validation set in Table 7.2.

We see that the attention is very spread out over sentences. In particular, we notice that the word-level attention focuses on specific stop words for all decoder time steps. We posit this may be due to ??? See Appendix A for more visualizations.

We show some predicted summaries from the model.

fill in numbers, plot training curves, attention heatmaps, predicted sum-

Model	PPL	ROUGE-1	ROUGE-2	ROUGE-L
FIRST	-	32.3	15.5	27.4
BERKELEY	-			
MODEL 0	13.9			
MODEL 1				
MODEL 2 CONV	15.8			
MODEL 3 CONV	32.3			
MODEL 2 BOW	16.4			
MODEL 2 LSTM	15.5			
MODEL 2 CONV +POS	15.3			
MODEL 2 5X80 CONV	14.8			
MODEL 2 2X200 CONV	14.2			
MODEL 3 CONV +POS	25.0			
MODEL 3 CONV +MULTI2	25.0			
MODEL 3 CONV +POS +MULTI2	25.0			
MODEL 3 CONV +MULTI3				

Table 7.1: Summarization results for CNN/Dailymail on the test set.

7.3 Discussion

Coarse-to-fine might work but it hurts performance. For next steps, we would want to scale to even larger datasets and implement hard attention.

Model	Entropy
MODEL 0	
MODEL 1	
MODEL 2 CONV	
MODEL 2 BOW	
MODEL 2 CONV +ENTROPY	
MODEL 2 CONV +POS	

Table 7.2: Entropy over sentence attention. We computed MODEL 0 *and* MODEL 1 *entropy by summing over each row.*

Chapter 8

Conclusion

Bibliography

Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple Object Recognition with Visual Attention. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation By Jointly Learning To Align and Translate. *Iclr 2015*, pages 1–15, 2014. ISSN 0147-006X. doi: 10.1146/annurev.neuro.26.041002.131047. URL http://arxiv.org/abs/1409.0473v3.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A Neural Probabilistic Language Model. *The Journal of Machine Learning Research*, 3:1137–1155, 2003. ISSN 15324435. doi: 10.1162/153244303322533223.

Yoshua Bengio, Nicholas Léonard, and Aaron C Courville. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *CoRR*, abs/1308.3, 2013. URL http://arxiv.org/abs/1308.3432.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. Generating Sentences from a Continuous Space. *Iclr*, pages 1–13, 2016. URL http://arxiv.org/abs/1511.06349.

Pf Brown, Vjd Pietra, S Pietra, and R Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311, 1993. ISSN 08912017. doi: 10.1080/08839514.2011.559906. URL http://www.aclweb.org/anthology/J93-2003.

Ziqiang Cao, Furu Wei, Sujian Li, Wenjie Li, Ming Zhou, and Houfeng Wang. Learning Summary Prior Representation for Extractive Summarization. *Proceedings ACL 2015*, pages 829–833, 2015.

J Carbonell and J Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 335–336, 1998. ISSN 01635840 (ISSN). doi: 10.1145/290941.291025. URL papers2://publication/uuid/1FA33AEC-2C9E-4149-B740-02A7C6C24B93.

Danqi Chen, Jason Bolton, and Christopher D Manning. A Thorough Examination of the CNN / Daily Mail Reading Comprehension Task. *Acl 2016*, pages 2358–2367, 2016.

Jianpeng Cheng and Mirella Lapata. Neural Summarization by Extracting Sentences and Words. *Arxiv*, pages 484–494, 2016. URL http://arxiv.org/abs/1603.07252.

James Clarke and Mirella Lapata. Global inference for sentence compression: An integer linear programming approach. *Journal of Artificial Intelligence Research*, 31:399–429, 2008.

Trevor Cohn and Mirella Lapata. Sentence compression beyond word deletion. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 137–144. Association for Computational Linguistics, 2008.

Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. *BigLearn, NIPS Workshop*, pages 1–6, 2011a. URL http://infoscience.epfl.ch/record/192376/files/Collobert{_}NIPSWORKSHOP{_}2011.pdf.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011b. ISSN 0891-2017. doi: 10.1.1.231.4614.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011. ISSN 15324435. doi: 10.1109/CDC.2012.6426698. URL http://jmlr.org/papers/v12/duchi11a.html.

Greg Durrett, Taylor Berg-Kirkpatrick, and Dan Klein. Learning-Based Single-Document Summarization with Compression and Anaphoricity Constraints. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1998–2008, 2016. URL http://www.aclweb.org/anthology/P16-1188.

Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning Hierarchical Features for Scence Labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–15, 2013. ISSN 01628828. doi: 10.1109/TPAMI.2012.231. URL http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=6338939.

Yoav Goldberg. A primer on neural network models for natural language processing. *arXiv preprint arXiv:1510.00726*, 2015.

KM Hermann, T Kocisky, and E Grefenstette. Teaching machines to read and comprehend. *Advances in Neural*, pages 1–9, 2015. URL http://papers.nips.cc/paper/5945-teaching-machines-to-read-and-comprehend.

Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Others. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012. ISSN 1053-5888. doi: 10.1109/MSP.2012.2205597. URL http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=6296526{%}5Cnhttp://ieeexplore.ieee.org/xpl/login.jsp?reload=true{&}tp={&}arnumber=6296526{&}url=http{%}3A{%}2F{%}2Fieeexplore.ieee.org{%}2Fxpls{%}2Fabs{_}all.jsp{%}3Farnumber{%}3D6296526{%}5Cnhttp://www.isip.piconepress.com/courses/t.

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

Yoon Kim. Convolutional Neural Networks for Sentence Classification. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, pages 1746–1751, 2014. ISSN 10709908. doi: 10.1109/LSP.2014.2325781. URL http://emnlp2014.org/papers/pdf/EMNLP2014181.pdf.

Diederik P. Kingma and Jimmy Lei Ba. Adam: a Method for Stochastic Optimization. *International Conference on Learning Representations 2015*, pages 1–15, 2015.

Kevin Knight and Daniel Marcu. Summarization beyond sentence extraction: A probabilistic approach to sentence compression. *Artificial Intelligence*, 139(1):91–107, 2002.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, pages 1–9, 2012. ISSN 10495258. doi: http://dx.doi.org/10.1016/j.protcy.2014.09.007.

Julian Kupiec, Jan Pedersen, and Francine Chen. A Trainable Document Summarizer. *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 68–73, 1995. ISSN 01635840. doi: 10.1145/215206.215333. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.1161.

Y LeCun and Y Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(April 2016):255–258, 1995. ISSN 1098-7576. doi: 10.1109/IJCNN.2004.1381049. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.32.9297{&}rep=rep1{&}type=pdf.

Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection. *ISER*, pages 1–1, 2016. ISSN 00032999. doi: 10.1145/2835776.2835844. URL http://0-doi.acm.org.cisne.sim.ucm.es/10.1145/2835776.2835844{%}5Cnhttp://0-dl.acm.org.cisne.sim.ucm.es/ft{_}gateway.cfm?id=2835844{&}type=pdf{%}5Cnhttp://arxiv.org/abs/1603.02199.

Jiwei Li, Minh-Thang Luong, and Dan Jurafsky. A Hierarchical Neural Autoencoder for Paragraphs and Documents. *CoRR*, abs/1506.0, 2015. URL http://arxiv.org/abs/1506.01057.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A Persona-Based Neural Conversation Model. *arXiv preprint arXiv:1603.06155*, 2016a.

Jiwei Li, Will Monroe, Alan Ritter, and Dan Jurafsky. Deep Reinforcement Learning for Dialogue Generation. *arXiv*, 2(2):1192–1202, 2016b. URL http://arxiv.org/abs/1606.01541.

Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out: Proceedings of the ACL-04 workshop*, volume 8. Barcelona, Spain, 2004.

Hans Peter Luhn. The automatic creation of literature abstracts. *IBM Journal of research and development*, 2(2):159–165, 1958.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. *Emnlp*, (September):11, 2015. ISSN 10495258. doi: 10.18653/v1/D15-1166. URL http://arxiv.org/abs/1508.04025.

Chris J Maddison, Andriy Mnih, Yee Whye Teh, United Kingdom, and United Kingdom. the Concrete Distribution: a Continuous Relaxation of Discrete Random Variables. *ICLR*, pages 1–17, 2017.

Christopher D. Manning. Computational Linguistics and Deep Learning, 2016. URL http://www.mitpressjournals.org/doi/10.1162/COLI{_}a{_}00239.

André F. T. Martins and Ramón Fernandez Astudillo. From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification. *Proceedings of The 33rd International Conference on Machine Learning*, pages 1614–1623, 2016. URL http://arxiv.org/abs/1602.02068.

Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. What to talk about and how? Selective Generation using LSTMs with Coarse-to-Fine Alignment. *Proceedings of NAACL-HLT*, pages 1–11, 2016.

T Mikolov and J Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 2013.

Volodymyr Mnih, Nicolas Heess, Alex Graves, and koray Kavukcuoglu. Recurrent models of visual attention. *Advances in Neural Information Processing Systems*, pages 2204—-2212, 2014.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei a Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. ISSN 0028-0836. doi: 10.1038/nature14236. URL http://dx.doi.org/10.1038/nature14236.

Ramesh Nallapati, Bowen Zhou, Cicero Nogueira dos Santos, Caglar Gulcehre, and Bing Xiang. Abstractive Text Summarization Using Sequence-to-Sequence RNNs and Beyond. *Proceedings of CoNLL*, abs/1602.0:280–290, 2016. URL http://arxiv.org/abs/1602.06023.

Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. Language Understanding for Text-based Games Using Deep Reinforcement Learning. *Emnlp2015*, (September):10, 2015. doi: 10.18653/v1/D15-1001. URL http://arxiv.org/abs/1506.08941.

Karthik Narasimhan, Adam Yala, and Regina Barzilay. Improving Information Extraction by Acquiring External Evidence with Reinforcement Learning. *Emnlp*, pages 2355–2365, 2016.

Ani Nenkova and Kathleen McKeown. Automatic Summarization. *Foundations and Trends® in Information Retrieval*, 5(3):235–422, 2011. ISSN 1554-0669. doi: 10.1561/1500000015. URL http://www.nowpublishers.com/product.aspx?product=INR{&}doi=1500000015.

Paul Over, Hoa Dang, and Donna Harman. DUC in context. *Information Processing & Management*, 43(6):1506–1520, 2007. ISSN 03064573. doi: 10.1016/j.ipm.2007.01.019.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-jing Zhu. BLEU: a Method for Automatic Evaluation of Machine Translation. *Computational Linguistics*, (July):311–318, 2002.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, 2014. ISSN 10495258. doi: 10.3115/v1/D14-1162.

Jack Rae, Jonathan J Hunt, Ivo Danihelka, Timothy Harley, Andrew W Senior, Gregory Wayne, Alex Graves, and Tim Lillicrap. Scaling Memory-Augmented Neural Networks with Sparse Reads and Writes. In D D Lee, M Sugiyama, U V Luxburg, I Guyon, and R Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3621–3629. Curran Associates, Inc., 2016.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence Level Training with Recurrent Neural Networks. *CoRR*, abs/1511.0:1–15, 2015. doi: 10.1371/journal.pcbi.1005055. URL http://arxiv.org/abs/1511.06732.

F. Rosenblatt. A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. ISSN 1939-1471(Electronic);0033-295X(Print). doi: 10.1037/h0042519.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. ISSN 0028-0836. doi: 10.1038/323533a0. URL http://books.google.com/books?hl=en{&}lr={&}id=FJblV{_}iOPjIC{&}oi=fnd{&}pg=PA213{&}dq=Learning+representations+by+back-propagating+errors{&}ots=zZDj2mGYVQ{&}sig=mcyEACaE{_}ZB4FB4xsoTgXgcbE2g{%}5Cnhttp://books.google.com/books?hl=en{&}lr={&}id=FJblV{_}iOPjIC{&}oi=fnd{&}pg=PA213{&}dq=Lea.

Alexander M Rush, Sumit Chopra, and Jason Weston. A Neural Attention Model for Abstractive Sentence Summarization. *In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015. ISSN 19909772. doi: 10.1162/153244303322533223. URL http://arxiv.org/abs/1509.00685.

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient Estimation Using Stochastic Computation Graphs. *NIPS*, pages 1–13, 2015. ISSN 10495258. URL http://arxiv.org/abs/1506.05254.

Claude E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(July 1948):379–423, 1948.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously Large Neural Networks: the Sparsely-Gated Mixture-of-Experts Layer. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

Dou Shen, Jian-tao Sun, Hua Li, Qiang Yang, and Zheng Chen. Document Summarization using Conditional Random Fields. *Science*, 7:2862–2867, 2004. ISSN 10450823. URL http://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intitle:Document+Summarization+using+Conditional+Random+Fields{#}0.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search.

Nature, 529(7587):484–489, 2016. ISSN 0028-0836. doi: 10.1038/nature16961. URL http://dx.doi.org/10.1038/nature16961.

S Sukhbaatar, J Weston, and R Fergus. End-to-end memory networks. *Nips*, pages 1–9, 2015. URL http://papers.nips.cc/paper/5846-end-to-end-memory-networks.

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

Krysta Marie Svore, Lucy Vanderwende, Christopher J C Burges, K Svore Vanderwende, L., and Burges, C., and K Svore Vanderwende, L., and Burges, C. Enhancing single-document summarization by combining RankNet and third-party sources. *In Proceedings of the Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2007.

Kristina Toutanova, Ke M Tran, and Saleema Amershi. A Dataset and Evaluation Metrics for Abstractive Compression of Sentences and Short Paragraphs. In *EMNLP*, nov 2016.

Christopher JCH Watkins. Learning From Delayed Rewards. 1989.

L Weaver and N Tao. The optimal reward baseline for gradient-based reinforcement learning. *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 538–545, 2001. doi: 10.1.1.8.8533.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *ArXiv e-prints*, pages 1–23, 2016. URL http://arxiv.org/abs/1609.08144.

Kelvin Xu, Jimmy Lei Ba Ryan Kiros, Kyunghyun Cho Aaron Courville, Ruslan Salakhutdinov Richard S. Zemel Yoshua Bengio, Jimmy Ba, Ryan Kiros, Kyunghyun Cho,

Aaron C Courville, Ruslan Salakhutdinov, Richard S Zemel, and Yoshua Bengio. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. *ICML*, 14: 77—-81, 2015. ISSN 19410093. doi: 10.1109/72.279181. URL http://arxiv.org/abs/1502.03044.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. Learning to Compose Words into Sentences with Reinforcement Learning. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017. URL https://arxiv.org/pdf/1611.09100v1.pdf.

David Zajic, Bonnie Dorr, and Richard Schwartz. Bbn/umd at duc-2004: Topiary. In *Proceedings of the HLT-NAACL 2004 Document Understanding Workshop, Boston*, pages 112–119, 2004.

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8689 LNCS(PART 1): 818–833, 2014. ISSN 16113349. doi: 10.1007/978-3-319-10590-1_53. URL http://link.springer.com/10.1007/978-3-319-10590-1{_}53{%}5Cnhttp://arxiv.org/abs/1311.2901{%}5Cnpapers3://publication/uuid/44feb4b1-873a-4443-8baa-1730ecd16291.

Appendix A

Attention Visualizations

hi