Document Summarization (draft)

Jeffrey Ling

March 22, 2017

Abstract

Standard sequence-to-sequence (seq2seq) attention models have seen great success in NLP, but do not scale well to tasks with long sequences. We propose a novel coarse-to-fine attention method to reduce the number of computations necessary in standard attention. By organizing the source sequence into a 2-dimensional image, we hierarchically apply attention, using a coarse mechanism for the first layer to select a row sequence, and a finer mechanism for the second soft attention layer. While the computation for training standard seq2seq models scales linearly with source sequence length, our method is invariant to length and thus can scale arbitrarily.

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

Contents

Chapter 1

Introduction

1.1 Natural Language Processing

Natural language processing is a field with a variety of interesting structured prediction problems. The essential goal of NLP is to build a model of language so that computers can automatically process substantial quantities of text — a highly relevant problem in today's information age.

While humans have no trouble understanding and using language, even the simplest language tasks can be impossible for computers. Researchers developed the field of linguistics as they tried to build a formal model of how language works, and as the number of rules grew, computational methods for processing text soon became relevant. Today, NLP has transformed into its own subfield — some classical linguistic problems still remain relevant, such as part-of-speech tagging, parsing, and semantic modeling, but more computational approaches are also used to tackle problems like language modeling and machine translation.

It is informative to consider the history of machine translation. The first methods were rule-based linguistics systems, and the company SYSTRAN provided one of the first widely available translation tools.

Later, statistical methods for NLP became increasingly popular. By considering language generation as a probabilistic process, one can collect statistics over large corpuses of data to automatically deduce the parameters of the model. In machine translation, this idea was used to great effect in Brown et al. (1993), whose count-based methods form the core of state-of-the-art systems like Google Translate.

Recently, researchers found that deep learning works effectively for many NLP tasks. For the first time, neural networks were able learning structure and features from language almost completely from scratch (Collobert et al., 2011b). The success of neural methods has been adopted by Google Translate to build even better systems (Wu et al., 2016). Research in applying deep learning to NLP is ongoing.

1.2 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.

Some other work continued: Rumelhart et al. (1986) showed that the backpropagation algorithm could be used to efficiently train general neural networks, but this was before the era of GPUs and modern computing, and so the algorithm could not be practically used.

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 modern computing power and GPUs, which can train networks at 10 or 20 times the speed of standard CPUs. Today, deep models are used to successfully play Go (Silver et al., 2016), play Atari games from pixels (Mnih et al., 2015), and ......

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.

figure

The idea of learning representations of the input is highly general, and so it comes as no surprise that deep networks soon found applications 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.

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. State-of-the-art systems for many problems, including machine translation, now use deep learning as their core algorithm. 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.

1.3 Motivation

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

On one hand, systems with handcrafted features can perform extremely well if domain specific knowledge can be built in. On the other hand, a purely data-driven system can forgo most of these details, using highly general models for the task at hand.

Examples?

There are a few reasons why deep networks are desirable for NLP. First, they work remarkably well without any feature engineering, and are state-of-the-art in many existing tasks. Second, they are not mutually exclusive with standard feature extraction methods. Third, we find that trained models can discover latent structure in language automatically, which is an interesting phenomenon that may reveal more 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 supervision.

It is not yet clear, however, to what extent neural methods can replace models with hard-coded linguistic assumptions. We set out to understand this question by analyzing and extending a particular deep neural model for the task of document summarization.

1.4 Our problem

Text summarization is an important problem for compressing large bodies of natural text into a more easily digestable form. Document summarization is one of the most challenging formulations of this problem, where given a document with several sentences of text, the goal is to produce a coherent summary that captures most or all of its salient points.

To accomplish this, we use the most recent advances in deep learning to automatically produce summaries by training on a large dataset of given examples.

We can frame summarization as a supervised sequence-to-sequence task: given a news article (the source sequence of words), we desire to produce a summary (the target sequence). Existing methods in deep learning have been developed and proven to be highly effective for this kind of task, especially the sequence-to-sequence (seq2seq) model applied to machine translation (Sutskever et al., 2014; Bahdanau et al., 2014). Rush et al. (2015) uses the seq2seq model to summarize sentences into shorter headlines, but we consider the more general problem of arbitrarily long documents.

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 $O(N)$ seq2seq model computations. 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. In the problem of image captioning, Xu et al. (2015) show that when generating words of the caption, the model places attention on the relevant part of the image. Inspired by this idea, we introduce the coarse-to-fine attention mechanism that sparsely selects subsets of the source document for processing.

We provide an outline for the rest of this thesis. In section ? we cover related work in summarization and deep learning. In section ? we give the necessary background for our models. In section ? we describe our models in detail. In section ? we show results and discussion.

Chapter 2

Related Work

2.1 Automatic Summarization

Nenkova and McKeown (2011) give a detailed overview of the problem of summarization. In particular, they provide a taxonomy of methods that researchers have used 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).

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 and search engines, 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.

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 have no assumptions on 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 more precisely, we will be better able to build systems to accomplish our desired tasks.

Before elaborating on specific methods, it is important to highlight these differences to understand how people thought about the problem. For example, extractive methods are by far more popular due to the simplicity of building an extraction algorithm, while abstractive methods have been difficult as we do not have a fully working model of language generation.

Based on this taxonomy, the deep model we set out to build would be classified as: abstractive, single-document, headline, and generic. While there is value to exploring other branches of this taxonomy, this combination of categories is the simplest for a deep model to handle.

In the rest of this thesis, we will limit the scope of the summarization problem to the single-document, headline, and generic summarization problem. In the next section, we give a brief overview of some of the relevant methods used.

2.2 Methods

2.2.1 Classical

One of the first considerations of automatically producing summaries was Luhn (1958), which pioneered work in the field. At the time, computers still ran on punch cards, so automating the summarization process was no easy feat.

Luhn (1958) proposed a 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 framework.

Extractive The most popular methods for document summarization have generally been extractive due to their simplicity. One natural procedure for an extractive summarization is to score sentences based on some relevance metric and return the highest scoring sentences.

Some examples are Carbonell and Goldstein (1998), which uses a simple information metric for ranking sentences, and Svore et al. (2007), which uses a basic neural network for the same purpose.

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

details

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

2.2.2 Deep Learning

With the onset of deep learning, learning an end-to-end abstractive model for summarization has become more feasible. Rush et al. (2015) propose a data-driven, completely abstractive model for summarizing short sentences by training a sequence-to-sequence model with attention. More recent work in deep learning has been done for both extractive () and abstractive (Nallapati et al., 2016) methods that scale the models to full documents, demonstrating the feasibility of end-to-end models.

These new models require a large amount of supervised training data, which previously was not available. However, thanks to the Internet, the web can easily be scraped to produce large-scale annotated datasets to train our deep models.

2.3 Datasets

To standardize the summarization task, NIST released data for DUC (Document Understanding Conferences) between 2001-2007 (Over et al., 2007). The DUC tasks involved producing summaries for both single- and multiple-document sets of news articles. DUC 2001 and 2002 ask for general summaries of these articles documents and summaries, while DUC 2003-2006 also evaluate summaries based on their usefulness for certain question-answer tasks.

DUC overall was not particularly impactful. For many of the news summary tasks, it was found that taking the first sentence of each article was a strong baseline that more sophisticated methods found hard to beat. However, DUC inspired a lot of thinking on the best way to evaluate summaries (see below).

Recent datasets are much larger thanks to the power of scraping the Internet. Hermann et al. (2015) released the CNN/Dailymail for question answering, which can also be adapted for summarization. Each document is a news article accompanied by three or four “highlight” bullet points, which we can treat as the summaries. We will use the CNN/Dailymail dataset to train our deep models.

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). 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.

2.4 Evaluation

Evaluating a good summary is inherently ambiguous, and probably one of the hardest parts of the problem.

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.

The DUC conferences really pushed forward understanding on evaluation. While a single most effective metric for summarization may not exist, DUC established several important criteria, including grammaticality, non-redundancy, and content coverage.

In response, people came up with recall on elementary discourse units (EDUs), based on clauses within a summary that ought to be captured. ROUGE Lin (2004) is cheap and fast. Pyramid method is a complicated human evaluation method based on summary content units (SCUs).

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). In our work, we settle for ROUGE due to its cheapness and ease of use in evaluating our models.

2.5 In the wild

Summarization is an important real-world problem due to the explosion of available data. Thus, there are many practical methods that have been developed and deployed in real-world settings. One example is on Reddit[1]: in order to summarize long forum discussions, Reddit uses technology from SMMRY[2].

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

[1] reddit.com

[2] smmry.com

Chapter 3

Background

In this chapter, we set up the relevant background ideas for our models.

3.1 Sequence-to-Sequence Attention Models

The sequence-to-sequence architecture (Sutskever et al., 2014), also known as the encoder-decoder architecture, forms the backbone of many successful models in NLP. 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. These models have 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 et al., 2015), dialogue (Li et al., 2016a), caption generation (Xu et al., 2015), and in particular summarization (Rush et al., 2015). In particular, these works show that learning an accurate attention function is important for good performance.

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, they qualitatively find that the model is essentially describing that region of the image. Figure **??** shows some examples.

We can leverage the same idea for text summarization, assuming we have a suitable representation of our input document. The simplest method for doing so would be to run an LSTM over the document.

However, the attention step becomes computationally difficult — 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. 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 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. Thus, we run a low-level LSTM encoder on the words of each sentence for a fine-grained representation of the text, and a simpler encoder model (e.g. bag

[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. Similar ideas have also been called hierarchical attention (Nallapati et al., 2016).

of words) for coarse-grained sentence representations. 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 coarse and fine encodings to develop a hierarchical autoencoder for representing paragraphs of text.

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. This idea runs into a very serious problem, however: by posing the attention as a discrete selection process, the neural network loses the key property of being differentiable, and so standard backpropagation no longer applies.

Xu et al. (2015) suggest “hard” attention as one possible solution to the discrete selection problem. While standard “soft” attention actually averages the representations of where the model attends to, for hard attention we make a hard decision and choose only one location. Such models can be trained using reinforcement learning. Before we elaborate on this method, we survey some other methods invented to overcome this discrete attention problem.

3.2 Conditional Computation

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

make this section more coherent

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 much more efficient, especially for networks that handle extremely large inputs as is common in vision and NLP.

Several ideas have been proposed to conditionally compute on large inputs. 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) introduces 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.

3.2.1 Discrete units

One particular class of methods to perform conditional computation relies on the use of discrete units within our deep network. Discrete units can be produced using stochastic gates to select certain parts of the network for computation. Unfortunately, discrete variables cannot be backpropagated through as they are not differentiable. 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 for simple binary gates, it is theoretically unjustified.

In general, we usually want to sample from multinomial distributions given by the output of a softmax function. The function softmax : $\mathbb{R}^m \to [0,1]^m$ produces a normalized probability distribution, where for $\boldsymbol{z} \in \mathbb{R}^m$,

$$\text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \tag{3.1}$$

Martins and Astudillo (2016) propose an alternative to softmax called the *sparsemax* function. For a given $\boldsymbol{z}$, 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. In this thesis, we focus on the hard attention method of Xu et al. (2015). In the next section, we explain reinforcement learning and how it can be used 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 transition distribution $p(s_{t+1}|s_t, a_t)$ (so that our environment is Markovian).

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. Here, γ is a time discount factor.

3.3.1 Methods for training RL agents

A variety of methods have been invented for solving the RL problem, i.e. finding the optimal policy $\pi(s)$ for a given state s.

When the transition distribution $p(s_{t+1}|s_t, a_t)$ and reward function $R(s, a)$ is known, our problem is also known as a Markov decision process (MDP). We can compute an optimal value function $Q(s, a)$ that represents the estimated reward from taking action a in state s:

$$Q(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V(s') \tag{3.2}$$

$$V(s) = \max_{a \in \mathcal{A}} Q(s, a) \tag{3.3}$$

for every $s \in \mathcal{S}, a \in \mathcal{A}$. These equations are known as the Bellman equations, and finding an optimal policy then amounts to solving this system for π. This can be done by treating the system as a fixed point problem and using iterative methods.

When we don't know the transition distribution, we can estimate it by standard maximum-likelihood methods and exploration of the state space. The same iteration techniques then apply.

When we don't know the reward function *and* we don't know the transition distribution, things become trickier. While the environment still gives us rewards for actions, finding the optimal policy requires knowing which states lead to those rewards. Since we don't know ahead of time which states are best, we are forced to extensively explore the state space to find what actions lead to the best rewards.

There are two methods for approaching the general RL problem. First, model-based approaches attempt to estimate the transition distribution $p(s_{t+1}|s_t, a_t)$ and apply the Bellman equations to find the optimal policy. Second, model-free approaches forgo the transitions and simply learn what action is best in a given state.

3.3.2 Model-free approaches

We consider the model-free approach in more detail.

Q-learning One technique is to model the Q-function $Q(s, a)$ that estimates the total reward of action a in state s. To learn the Q-function, we predefine some policy based on our current estimates of the Q-function, and we make learning updates as

$$Q(s,a) \leftarrow Q(s,a) + \eta \left(r + \gamma \max_{a' \in \mathcal{A}} Q(s', a') - Q(s,a) \right) \tag{3.4}$$

upon taking action a in state s, where η is a learning rate, r is the reward received, and s' is the state we transition into. This is known as the *Q-learning* update rule. An alternative update rule takes into account the action a' we would take next in state s':

$$Q(s,a) \leftarrow Q(s,a) + \eta \left(r + \gamma Q(s', a') - Q(s,a) \right) \tag{3.5}$$

This is known as the SARSA update rule.

In both cases, if we have a large state space, we may not want to record $Q(s, a)$ for every pair of s, a. We can instead parametrize $Q(s, a)$ with weights w, and maximizing reward using backpropagation gives us the update rule

$$w \leftarrow w + \eta \left(r + \gamma \max_{a' \in \mathcal{A}} Q(s', a') - Q(s,a) \right) \frac{\partial Q(s,a)}{\partial w} \tag{3.6}$$

Policy gradient An alternative to modeling the Q-function is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ directly. We parametrize π with weights w, and assume that at training time, we have a probability distribution $p(a|s; w)$ over the actions that we sample from. Then assuming we get reward r, our update is

$$w \leftarrow w + \eta \mathbb{E}_a \left[r \cdot \frac{\partial \log p(a|s;w)}{\partial w} \right] \tag{3.7}$$

This is known as the policy gradient update or REINFORCE algorithm (Williams, 1992). We give the derivation in the next section.

If we have a full trajectory of states s_t and rewards r_t, then we can assume our action a_t at time t led to future discounted reward $R_t = \sum_{k=t}^{\infty} \gamma^{k-t} r_k$. We then have the update rule

$$w^{(t+1)} \leftarrow w^{(t)} + \eta \mathbb{E}_{a_t} \left[R_t \cdot \frac{\partial \log p(a_t | s_t; w)}{\partial w} \right] \tag{3.8}$$

3.4 Deep reinforcement learning

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).

Aside from these classical applications of RL, the RL framework can be applied to train deep neural networks with stochastic units. We cast our learning problem with the neural network as an agent with a parametrized policy function $\pi : \mathcal{S} \to \mathcal{A}$, where the states $\mathcal{S}$ are the inputs and the actions $\mathcal{A}$ are the possible outputs of each stochastic unit. The rewards are then negative loss (e.g. from regression or classification).

More concretely, consider a sequential classification problem such as summarization, and consider the sequence-to-sequence model with attention. At each time step t of the decoder, our state s_t is given by the LSTM state and the encoded context, and our action a_t is where in the encoder context to attend to. The reward r_t is then the log-likelihood of predicting the gold word (according to some supervised dataset).

diagram

To obtain gradients for backpropagation, we can apply policy gradient methods for the nondifferentiable stochastic units. Because our network has both differentiable and nondifferentiable weights on the pathways to the output, we can consider it as a *stochastic computation graph* (Schulman et al., 2015).

Suppose that, given input x, our network has stochastic units drawn from some parametrized distribution $z \sim p(z|x;\phi)$. Then, a parametrized deterministic scalar function $f(z;\theta)$ gives us the output, and we want to maximize the expectation $\mathbb{E}_{z \sim p(z|x;\phi)}[f(z;\theta)]$.

To train the weights directly connected to the loss function (i.e. θ) we backpropagate the standard gradient:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial}{\partial \theta} \mathbb{E}_{z \sim p(z|x;\phi)}[f(z;\theta)] = \mathbb{E}_{z \sim p(z|x;\phi)} \left[\frac{\partial f(z;\theta)}{\partial \theta} \right] \tag{3.9}$$

To train weights that are not directly connected, but precede an intermediate stochastic unit (i.e.

ϕ), we backpropagate the weighted policy gradient:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \phi} &= \frac{\partial}{\partial \phi} \sum_z p(z|x;\phi) f(z;\theta) \\
&= \sum_z \frac{\partial p(z|x;\phi)}{\partial \phi} \cdot f(z;\theta) \\
&= \sum_z p(z|x;\phi) \frac{1}{p(z|x;\phi)} \frac{\partial p(z|x;\phi)}{\partial \phi} f(z;\theta) \\
&= \sum_z p(z|x;\phi) \frac{\partial \log p(z|x;\phi)}{\partial \phi} f(z;\theta) \\
\frac{\partial \mathcal{L}}{\partial \phi} &= \mathbb{E}_{z\sim p(z|x;\phi)} \left[\frac{\partial \log p(z|x;\phi)}{\partial \phi} f(z;\theta) \right] \qquad (3.10)
\end{aligned}$$

For a given x, we perform Monte Carlo sampling to obtain the gradients, e.g.

$$\frac{\partial \mathcal{L}}{\partial \phi} \approx \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \log p(z_i|x;\phi)}{\partial \phi} f(z_i;\theta) \qquad (3.11)$$

where $z_i \sim p(z|x;\phi)$ i.i.d. for $i = 1, \ldots, N$. In practice, N is set to 1 for computational reasons.

Note that this generalizes the REINFORCE algorithm in equation 3.6, where $f(z;\theta)$ is the reward.

3.4.1 Baseline reward

In practice, 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 reward b which we subtract from our reward. This works because

$$\mathbb{E}_z \left[(f(z;\theta) - b) \frac{\partial \log p(z|x;\phi)}{\partial \phi} \right] = \mathbb{E}_z \left[f(z;\theta) \frac{\partial \log p(z|x;\phi)}{\partial \phi} \right] - b\mathbb{E}_z \left[\frac{\partial \log p(z|x;\phi)}{\partial \phi} \right] \qquad (3.12)$$

The second term vanishes as

$$\begin{aligned}
\mathbb{E}_z \left[\frac{\partial \log p(z|x;\phi)}{\partial \phi} \right] &= \sum_z p(z|x;\phi) \frac{\partial \log p(z|x;\phi)}{\partial \phi} = \sum_z p(z|x;\phi) \frac{1}{p(z|x;\phi)} \frac{\partial p(z|x;\phi)}{\partial \phi} \\
&= \frac{\partial}{\partial \phi} \sum_z p(z|x;\phi) = \frac{\partial}{\partial \phi}[1] = 0
\end{aligned}$$

Thus, the gradient with baseline is unbiased.

Including a baseline is proven to reduce the variance of the estimator (Weaver and Tao, 2001). Intuitively, we can think of it as an estimate of the value function $V(s)$ for a state s. There are a few ways to produce a baseline for a given reward, which we describe in section ????.

Our policy gradient update from equation 3.8 can therefore be summarized as

$$w^{(t+1)} \leftarrow w^{(t)} + \eta \mathbb{E}_{a_t} \left[(R_t - b) \frac{\partial \log p(a_t|s_t;w)}{\partial w} \right] \qquad (3.13)$$

3.4.2 Deep RL in NLP

In computer vision, deep reinforcement learning has been applied with some success (Mnih et al., 2014; Ba et al., 2015; Xu et al., 2015).

In NLP, RL 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) uses RL play text-based games. Li et al. (2016b) uses RL to train an end-to-end dialogue system. Narasimhan et al. (2016) uses RL to improve information extraction on scarce data. Yogatama et al. (2017) uses 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 ideas, we handle the document summarization problem by (1) implementing conditional computation using coarse-to-fine attention at the sentence level, and (2) including reinforcement learning for a sparse coarse attention layer.

In the next chapter we describe our models in detail.

Chapter 4

Models

4.1 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).

4.1.1 Recurrent encoder and decoder

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 $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^{d_{in}}$ through a word embedding matrix $\boldsymbol{E} \in \mathbb{R}^{|\mathcal{V}| \times d_{in}}$ as $\boldsymbol{x}_t = \boldsymbol{E} w_t$.

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

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}^f \boldsymbol{x}_t + \boldsymbol{U}^f \boldsymbol{h}_t + b_f) \tag{4.1}$$

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}^i \boldsymbol{x}_t + \boldsymbol{U}^i \boldsymbol{h}_t + b_i) \tag{4.2}$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}^o \boldsymbol{x}_t + \boldsymbol{U}^o \boldsymbol{h}_t + b_o) \tag{4.3}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tanh(\boldsymbol{W}^c \boldsymbol{x}_t + \boldsymbol{U}^c \boldsymbol{h}_{t-1} + b_c) \tag{4.4}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t) \tag{4.5}$$

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

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

In stacked LSTMs, we will take the top sequence of hidden states $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n$ 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 $\boldsymbol{h}_t^{dec} \in \mathbb{R}^{d_{hid}}$ as $\boldsymbol{h}_t^{dec} = f_{dec}(y_{t-1}, \boldsymbol{h}_{t-1}^{dec})$ 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 $[\boldsymbol{h}_1, \dots, \boldsymbol{h}_n]$ and the current decoder hidden state $\boldsymbol{h}_t^{dec}$ and produces the context $\boldsymbol{c}_t \in \mathbb{R}^{d_{ctx}}$:

$$\boldsymbol{c}_t = a([\boldsymbol{h}_1, \dots, \boldsymbol{h}_n], \boldsymbol{h}_t^{dec}) \tag{4.6}$$

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. $\boldsymbol{h}_t^{dec} = f_{dec}([y_{t-1}, \boldsymbol{c}_t], \boldsymbol{h}_{t-1}^{dec})$.

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

$$p(y_t | y_{t-1}, \dots, y_1, [h_1, \dots, h_n]) = \boldsymbol{W}^{out} \boldsymbol{c}_t + b^{out} \tag{4.7}$$

The models are trained to maximize the log probability of getting the sequences in the dataset correct. 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 combines the encoder context and the current decoder hidden state. In the next few sections, we explain standard choices for a as well as our proposed coarse-to-fine attention model.

4.2 Model 0: Standard Attention

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

$$\beta_{t,i} = \boldsymbol{h}_i^\top \boldsymbol{W}^{attn} \boldsymbol{h}_t^{dec} \quad \forall i = 1, \dots, n \tag{4.8}$$

$$\boldsymbol{\alpha}_t = \text{softmax}(\boldsymbol{\beta}_t) \tag{4.9}$$

$$\widetilde{\boldsymbol{c}}_t = \sum_{i=1}^{n} \alpha_{t,i} \boldsymbol{h}_i \tag{4.10}$$

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

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

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 $\boldsymbol{c}_t$ as the expectation $\mathbb{E}_\alpha[\boldsymbol{h}]$, where we pick $\boldsymbol{h}_i$ with probability α_i.

Our final context vector is then

$$\boldsymbol{c}_t = \tanh(\boldsymbol{W}^2[\widetilde{\boldsymbol{c}}_t, \boldsymbol{h}_t^{dec}]) \tag{4.11}$$

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

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

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

For a large source input like a document, it may be computationally inefficient to run an RNN over the entire source. Instead, we can consider organizing the document into distinct sentences and select one sentence to attend to at a time.

Specifically, suppose we have sentences $s_1, \ldots, s_m$ with words $w_{i,1}, \ldots, w_{i,n_i}$ for sentence s_i, so that we can apply an RNN to each separately to get corresponding hidden states $\boldsymbol{h}_{i,j}$ for $i = 1, \ldots, m$ and $j = 1, \ldots, n_i$. 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 $h_{i,j}$ 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 $\alpha_1^s, \ldots, \alpha_m^s$ for each sentence, and then for sentence s_i, we have another set of weights $\alpha_{i,1}^w, \ldots, \alpha_{i,n_i}^w$. Our final attention weight on word $w_{i,j}$ is then $\alpha_{i,j} = \alpha_i^s \cdot \alpha_{i,j}^w$.

In order to compute the sentence attention weights α_i^s, 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 $\boldsymbol{h}_i^s \in \mathbb{R}^{d_{sent}}$.

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

$$\boldsymbol{h}_i^s = \sum_{j=1}^{n_i} \boldsymbol{E} w_{i,j} \tag{4.12}$$

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

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

Our final output is given by

$$\boldsymbol{h}^s = \max - \text{over} - \text{time}(\tanh(\boldsymbol{u} + b^{conv})) \tag{4.13}$$

where $\max - \text{over} - \text{time}$ takes the maximum along the j-indexing dimension.

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 $\boldsymbol{h}_{i,j}$.

We call this method of attention MODEL 2.

4.4 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 $\widetilde{\boldsymbol{c}}$ as an expectation over $\boldsymbol{\alpha}$ (i.e. $\boldsymbol{c} = \sum_{i=1}^{n} \alpha_i \boldsymbol{h}_i$), we can sample from the probability distribution $\boldsymbol{\alpha}$ to obtain a single state $\boldsymbol{h}_i$, and we set $\widetilde{\boldsymbol{c}} = h_i$ 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 $\alpha_{i,1}^w, \ldots, \widetilde{\alpha}_{i,n_i}^w$ 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 turn to policy gradient methods from the reinforcement learning literature.

4.4.1 Practical considerations

Recall the policy gradient update from section ??? at time t of the decoder:

$$w \leftarrow w + R_t \frac{\partial \log p(\boldsymbol{\alpha}_t | y_1, \ldots, y_{t-1})}{\partial w} \tag{4.14}$$

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 $R_t = \sum_{s=t}^{n} \gamma^{n-s} r_s$ at time t, where $r_t = \log p(y_t | y_1, \ldots, y_{t-1})$ is the single step reward.

While this gradient is theoretically unbiased, because the gradient is sampled through a stochastic process, it tends to have high variance in practice. Thus, we implement a few variance reduction techniques to stabilize training.

Baseline reward As described in section ???, we subtract a baseline from the reward to reduce variance while keeping the gradient unbiased.

In practice, we keep a separate baseline b_t for each decoder time step t. Then our total discounted reward at time t becomes

$$R_t = \sum_{s=t}^{n} \gamma^{n-s} (r_s - b_s) \tag{4.15}$$

Our policy gradient update then becomes

$$w \leftarrow w + R_t \frac{\partial \log p(\boldsymbol{\alpha}_t | y_1, \ldots, y_{t-1})}{\partial w} \tag{4.16}$$

There are a few methods for producing the baseline. We follow Xu et al. (2015) and keep an exponentially moving average of the reward for each time step t:

$$b_t \leftarrow b_t + \beta(r_t - b_t) \tag{4.17}$$

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 from the RNN state in RL tasks (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 RNN state and the reward.

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.

Chapter 5

Experiments

5.1 Data

5.1.1 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 5.1 lists statistics for the CNN/Dailymail dataset.

Dataset	CNN	Dailymail
Train size	90267	196962
Valid size	1221	12149
Average words per doc	794	832
Average sents per doc	21	29
Average words per sent	36	27
Average words per summary	13	15

Table 5.1: Statistics for CNN/Dailymail data.

examples of data

5.2 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 important for summarizing the document. Due to memory issues, this is a problem we still have to resolve.

5.3 Models

5.3.1 Baselines

For a baseline, we take the first 15 words of the document (chosen as the average length of a sentence in the training dataset). We call this FIRST.

Others...

5.3.2 Our models

We ran experiments with Models 0 to 3 as described above. Model 0 serves as the baseline.

- 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.

5.4 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.[1]

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[2]. We ran our experiments on a GPU (specs?).

In the next chapter we show results.

[1] We tried more complicated 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.

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

Chapter 6

Results

6.1 Evaluation

We report metrics for best validation perplexity and ROUGE scores (Lin, 2004). We use the ROUGE balanced F-scores with ROUGE-1 (unigrams), ROUGE-2 (bigrams), and ROUGE-L (longest common substring). We chose F-score since recall is biased towards longer predicted sentences.

To generate summaries for evaluation, we run beam search with a beam size of 5.

Model	PPL	ROUGE-1	ROUGE-2	ROUGE-L
FIRST	-	23.1	9.8	20.5
BERKELEY				
MODEL 0	13.6			
MODEL 1				
MODEL 2 CONV	15.8			
MODEL 2 SPARSEMAX				
MODEL 3 HARD ONLY				
MODEL 3 SEMI SAMPLING				
MODEL 2 BOW	16.4			
MODEL 2 LSTM				
MODEL 2 5X80 CONV				
MODEL 2 2X200 CONV				

Table 6.1: Summarization results for CNN/Dailymail.

6.2 Analysis

See Table 6.1 for summary results.

We investigate the entropy of the sentence attention on the validation set in Table 6.2.

We see hard attention takes longer to converge.

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

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

Table 6.2: Entropy over sentence attention. All numbers are for We computed MODEL 0 *and* MODEL 1 *entropy by summing over each row.*

be due to ??? See Appendix A for more visualizations.

We show some predicted summaries from the model.

Chapter 7

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, 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`.

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`.

J Carbonell and J Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '98, pages 335–336, New York, NY, USA, 1998. ACM. ISBN 1581130155. 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.

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`.

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`.

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.

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.

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.

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=Learn`.

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`.

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.

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