UNIVERSITY OF CALIFORNIA
Los Angeles

Paying Attention to Product Reviews:
Sentiment Analysis with Additive, Multiplicative, and Local Attention Mechanisms

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Applied Statistics

by

Greg Eastman

2021

ABSTRACT OF THE THESIS

Paying Attention to Product Reviews:
Sentiment Analysis with Additive, Multiplicative, and Local Attention Mechanisms

by

Greg Eastman
Master of Applied Statistics
University of California, Los Angeles, 2021
Professor Yingnian Wu, Chair

Every business needs to understand its consumers' experiences to succeed, but it is impossible for a large company to read every review. By having a computer read customer responses and relay their preferences, businesses can respond to consumers with a more informed approach. We begin this work by reviewing text preprocessing strategies as well as previous solutions to sentiment tasks. Then we use Amazon tool review data to introduce the addition of an attention mechanism to a BiLSTM encoder-decoder format. Three attention strategies are tested: additive, multiplicative, and local. To experimentally investigate their performance, we compare the attention models to each other as well as two controls.

The thesis of Greg Eastman is approved.

Akram Almohalwas

Chad Hazlett

Frederic Paik Schoenberg

Yingnian Wu, Committee Chair

University of California, Los Angeles

2021

Table of Contents

1 Introduction **1**

2 Text Preprocessing **3**

2.1 Cleaning Text 3

2.2 Stop Word Removal 4

2.3 Lemmatization 4

2.4 Encoding 5

2.4.1 Vocabulary Size 5

2.4.2 Text Length 5

3 Traditional Sentiment Strategies **7**

3.1 LSTM 8

3.2 BiLSTM 12

4 The Attention Mechanisms **15**

4.1 Additive 15

4.2 Multiplicative 18

4.3 Local 19

5 Final Models **23**

5.1 BiLSTM 23

5.2 BiLSTM Encoder-Decoder 24

5.3 Additive Attention Encoder-Decoder 25

5.4 Multiplicative Attention Encoder-Decoder 27

5.5 Local Attention Encoder-Decoder 28

6 Experiment **29**

6.1 Data 29

6.2 Metrics 30

6.2.1 Accuracy 30

6.2.2 Precision 31

6.2.3 Recall 31

6.3 Results 32

7 Conclusion **34**

List of Figures

3.1 Recurrent Neural Network 8

3.2 Long Short-Term Memory 9

3.3 Bidirectional Long Short-Term Memory 13

4.1 Additive Attention 16

4.2 Multiplicative Attention 18

4.3 Local Attention 20

5.1 Basic BiLSTM Model 23

5.2 BiLSTM Encoder-Decoder Model 25

5.3 Additive Attention Model 26

5.4 Multiplicative and Local Attention Models 27

List of Tables

6.1 Results 32

1 Introduction

Businesses have immense amounts of consumer feedback thanks to advances in technology and connectivity, and to use all this data, machine learning has grown. Specifically, text data has been an untapped mine until roughly the last decade when neural language processing saw its boom [13]. Therefore, machine reading, and in particular, sentiment analysis has developed. This branch of machine learning decomposes large bodies of text into more digestible pieces. Furthermore, star ratings and written reviews have become intertwined, allowing for more supervised approaches. Because of this, neural networks, as a form of AI, have seen a lot of success in handling text data [18]. Many advances have come from trying to make computers mimic human thinking, and this work will do the same.

Long Short-Term Memory has been a popular analytical technique, especially for natural language processing tasks like sentiment analysis [23] and machine reading [5]. LSTMs allow the computer to remember the entire text sequentially instead of each part independently. The flow of words is read from beginning to end, in the same manner as a human reader. This allows information in the beginning of the sentence to aid in understanding the meaning of later terms as well as the overall corpus. Furthermore, a bidirectional call reads the text forwards and backwards. This is often used to make the models more accurate [19] because information in a sentence is not always conveyed linearly from start to finish. However, this strategy is not sufficient to mimic human reading, as only a subset of the text's words contains important information. To further close the gap between human and machine reading, attention mechanisms have grown in popularity.

Attention strategies mirror the way a person focuses more on the important words or areas of a sentence. This is not a new idea in language processing, as text cleaning strategies often remove stop words to boost performance. But focusing more on a few words that are essential to the problem is more complicated than a simple dictionary approach. Attention allows the

model to predict what parts of a sentence are most meaningful, and then gives those sections more weight in the final evaluation. This attentive next step in mimicking human reading has seen massive success [15] but still has room for improvement and exploration.

2 Text Preprocessing

When using text data, the computer cannot immediately understand English words, so it is necessary to preprocess the corpus. Each preprocessing step has a different purpose. Some remove non-essential characters, some transform words into a single root term, and some encode each word with a unique numeric identifier. Each step plays an essential role, but text preprocessing steps vary from corpus to corpus, so each part needs to be justified.

2.1 Cleaning Text

The first cleaning step we took was fixing contractions. We split the words into their two distinct parts so that the full meaning is preserved. Otherwise, the computer treats "don't" and "do not" as different cases. We did all this using a dictionary search for possible contracted words, then moved to the next step.

We stripped all punctuation from the corpus to make the text more consistent, the learning task quicker, and the model more stable. Networks that incorporate punctuation become extremely dependent on their usage. Omitted marks, as is common with reviews, or creative punctuation, will cause the model to give poor results [1]. Additionally, this work is limited by the hardware running the models, so speeding up training is important. Once the punctuation was removed, we dealt with non-traditional characters.

We removed all non-alphanumeric characters, as the data is in English from US amazon reviews, no other characters should hold essential meaning. Amazon does not allow emojis or other reactions in their reviews. The next preprocessing strategy we employed helped the computer read more efficiently.

2.2 Stop Word Removal

Not all words add meaning to a sentence, many simply add structure, so we have removed these superfluous terms. Stop words exist simply as determiners to mark nouns like “the”, or coordinating conjunctions like “but”, or prepositions like “in”. Although these terms help to form a language, they do not alter the text’s sentiment. Removing these words only takes a search using the NLTK package [17] and leads to faster training and better boost for the model [11][8]. Up to this point we have avoided interacting or cleaning any parts of the text that may hold meaning, but the next step breaks that trend.

2.3 Lemmatization

Lemmatization converts words into a root term so that the computer does not have to learn every form of every word. To give an example of this process, the sentences: ”I loved this product,” and ”I love this product” become ”I love this product”. Both have the same meaning and sentiment, all that differed was the tense. Since we do not care about tenses, converting to a single root helps the computer train on the data [21]. Unfortunately, lemmatization is not simple, because homophones, or words with the same spelling but multiple meanings, are common. Therefore, we used the NLTK package to tag the part of speech for every word in the text. Then, in combination with a dictionary search, we were able to figure out if a term has a base root. This process was repeated for each word in the corpus, and when a root was identified the term was converted accordingly. Lemmatization was the final major step in high-level text modification, the rest of the preprocessing helped the computer work with the data.

2.4 Encoding

The models used word level embeddings to read the corpus. In word level embeddings, each term, or string of characters separated by spaces on either end, had its own unique integer identifier. This was done with Keras's one- hot encoding command. With this method the model reads each word as a unique numeric value. The embedding is spelling specific, which is why the previous text preprocessing is used. Additionally, when encoding text there are two major hyper-parameters to consider. The first is the size of the computer's vocabulary.

2.4.1 Vocabulary Size

Vocabulary size is a hyper-parameter that tells the computer how many different words can be learned, we set this value at 1,200. The reason to even set a vocabulary size is that learning every word in the text is not necessarily desirable. When making a vocabulary the researcher wants to balance knowing enough words to draw meaningful conclusions while avoiding uncommon words. These rare terms pose an issue because they may be given an unduly large weight simply from the low sample size. There is also no standard to inform vocabulary size for hot encoding. Since my data has 60,000 reviews there are a lot of words, so we chose a vocabulary size of 1,200. This should avoid some rare words but also allow for a sizable dictionary. The last step in preprocessing was setting a unified length for each observation.

2.4.2 Text Length

Reviews do not have a set length and setting one is important to this work both for building and training the model, so we made each review 400 words long. Picking a text

size allows the data to have an established shape for embeddings. To make each text array a standard size, the integer 0 is encoded in all the areas where there are no words. If a review had more than 400 terms, which was rare, all words after the cutoff were removed. Besides making the data easier to handle, some individuals would put in large amounts of text. This makes the model forget what was said in the beginning of the review. Since people often give their opinion as the topic sentence of their review, having a better memory of the beginning is important. When it comes to choosing a length there is no set rule. Often the longest review is the upper limit, but some people write long reviews, especially as a joke. To account for that we limited the size of each review to 400 words. The overall sense of the review should be able to be gleaned from that amount, so any more is unnecessary. With the text preprocessing covered, the next section will go over the foundational models for supervised neural language processing.

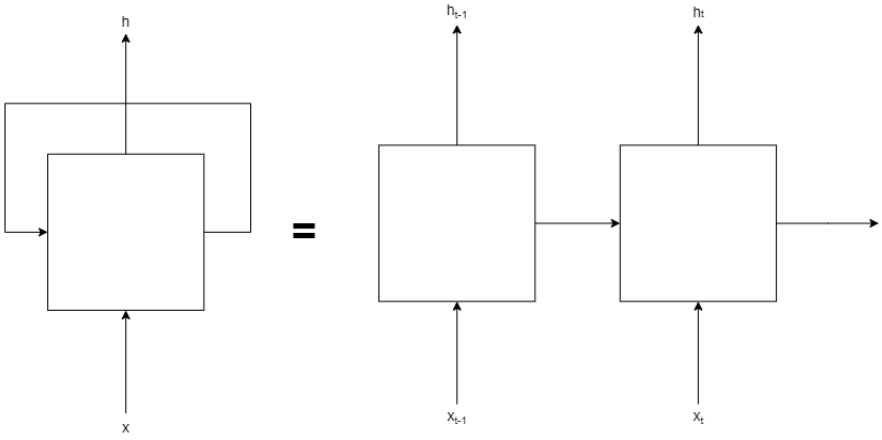
3 Traditional Sentiment Strategies

Two initial models emerged for machine reading in the early-mid 2010's [25]. The first was convolutional networks [10]. This model takes a small window over the data and summing the dot product of the weights in that view. This creates a single value for each window location, which alters the dimensionality of the input. This approach has several issues.

First, it loses information in the convolutional process because reshaping the data in small local groups can alter the meaning of the words in each step. It also does not remember the context of previous words because it only looks at the terms inside a window, instead of relating words by meaning to each other and the overall piece. Its popularity arose due to its success in machine vision [12]. It has an ability to comprehend spatial meanings due to the local window, but since text data has weak spatial relations, it does not work well in NLP. To make a model that is more tailored to the task, RNNs were introduced.

Early recurrent neural networks showed promising performance and formed a conceptual foundation for more modern techniques [18]. The core idea of this framework is that words in a corpus are not independent and are often intentionally related. To acknowledge this, RNNs use information from each word in the text to understand the meaning of later terms as well as the overall document. They do this by having a series of cells that take in both the inputs from the data, as well that of the previous node, to make a prediction. Within each cell the former timestep's output, and the current timestep's input, are both weighted and multiplied to get the new node's output. We visualize this process on the next page.

Figure 3.1: Recurrent Neural Network

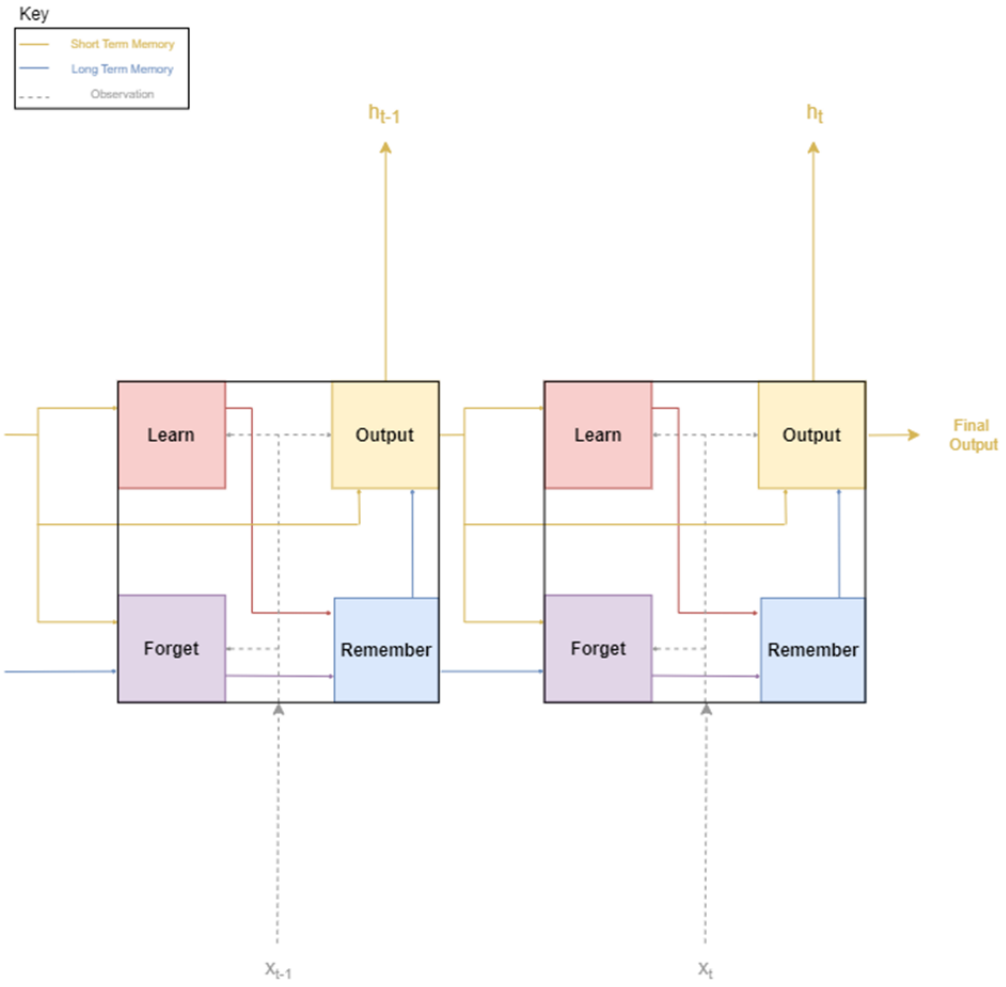


Despite their advantages RNNs have two major flaws, exploding and vanishing gradients. Exploding gradients occur when the weights update uncontrollably, making the model prone to large shifts in learning or prevent learning altogether. Vanishing gradients are the opposite issue, where weights do not update, preventing the model from training any further. Both issues are alleviated [20] by a special case of recurrent models called Long Short-Term Memory networks.

3.1 LSTM

Long Short-Term Memory networks use the base model of an RNN but change how each cell works. The original idea was developed in 1997 [9], but it has only been applied recently in machine reading [5] and sentiment analysis [23]. We build our work on previous research, so before going into attention layers we will break down how Long Short-Term Memory networks work. We start by visualizing the high-level architecture of an LSTM and then we will walk through the math.

Figure 3.2: Long Short-Term Memory



Each of the two large black boxes represents a single node, also called cell, in the LSTM. These store the memory of what was learned in the previous iteration, which helps inform the next node in the sequence. This recurrent learning happens until the last node gives the final output. To better understand this process, we will go through the givens of each input, weight, and bias.

W_n: Learning weights

b_n: Learning bias

W_i: Ignoring weights

b_i: Ignoring bias

W_f: Forgetting weights

b_f: Forgetting bias

W_o: Forgetting weights

b_o: Forgetting bias

x_t: Current input

$O_(t-1)$: Short term memory from previous timestep

$R_(t-1)$: Long term memory from previous timestep

The first gate we will explore is the learn gate, shown in red. This part of the cell takes in two inputs, the previous short-term memory, and the current observation. We train the weights and biases to learn the meaning of the input via backpropagation. We have the learn gate apply these functions using the weights and biases to relay their understanding of the information.

$$N_t = \sigma(W_n[O_(t-1), x_t] + b_n)$$

$$i_t = tanh(W_i[O_(t-1), x_t] + b_i)$$

Then we multiply the results to get the output of the gate.

$$M_t = N_t \times i_t$$

The next gate is the forget gate, we show in purple. This decides what information is important and forgets the rest. Mathematically, an output value near 0 means the information is not important and is going to be weighed out or forgotten. We first operate on the previous node's short-term memory and current input, which are denoted $h_(t-1)$ and x_t respectively. Then we have the gate use the last cell's long-term memory to factor in the previous iteration, this labeled as $l_(t-1)$. The process of mathematically forgetting the useless information is shown below.

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

$$Z_t = R_(t-1) \times f_t$$

We have the forget gate output Z_t, as well as the learned output M_t, passed to the remember gate, which we visualize in blue. The we have the gate perform a simple computation, adding the newly learned information with what was not forgotten. This simple process creates the long-term memory, and the formula shown below.

$$R_t = M_t + Z_t$$

We send R_t to both the next cell and the output gate that we show in yellow. We have this gate operate on the current input and previous short-term memory by weighing the data and squeezing it with a sigmoid function. Then we have it compress the long term-memory from this current cell with a tanh activation. The tanh and sigmoid functions from the last two steps compress all the information from both the long-term and short-term memory operations into a space bounded by 0 and 1. We finally multiply these two condensed values together. These steps get the intersection of the old short-term memory and the new long-term memory, creating the current short-term memory. We show the math for this process below.

$$O_t = \sigma(W_o[O_{t-1}, x_t] + b_o)$$

$$h_t = O_t \times tanh(R_t)$$

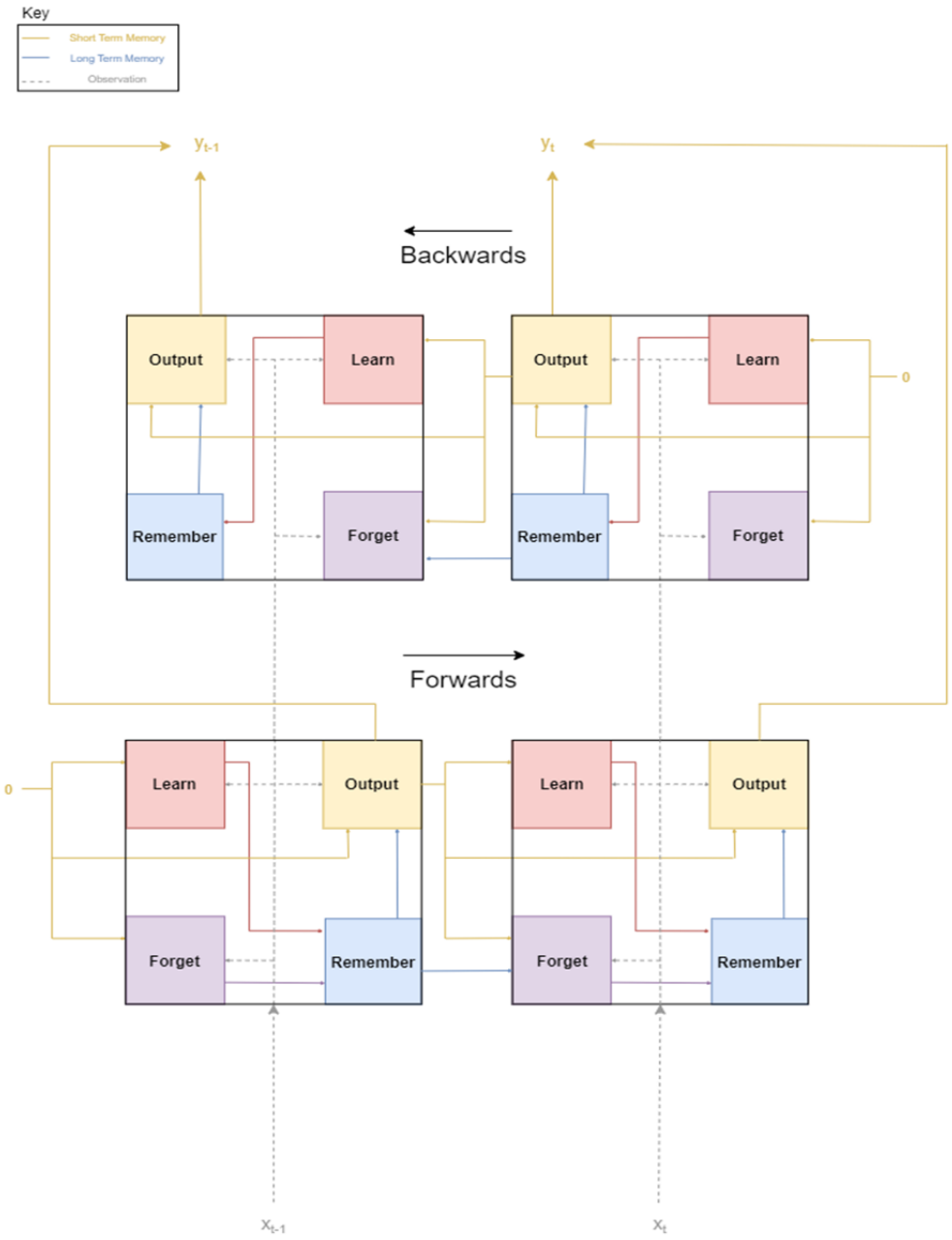
The above math represents the calculations of a single cell, but in practice, we do these operations for every node in the LSTM. Our model simply uses 0 as an initializer where the old long- and short-term memories would otherwise be. With a stronger understanding of how a simple LSTM functions, we will discuss its shortcomings.

Despite the name, Long Short-Term Memory models struggle with recalling information at the beginning of the chain. The long-term memory degrades, which is intentional, but the rate at which it happens can be faster than desired. Especially when faced with a large body of information. Simply, the larger the body of text the model must understand, the more it will struggle to do so. Granted, LSTMs suffer less from a vanishing gradient issue then the more basic form RNNs, but there is still room for improvement. Therefore, the Bidirectional Long Short-Term Memory layer was introduced.

3.2 BiLSTM

Bidirectionality is a simple concept that actively addresses the issue of forgetting early information. Many papers have used a bidirectional call as a single new addition to an LSTM [14] or in combination with other strategies [7]. BiLSTMs generate the result by running the layer forwards and backwards over the data. We have provided a visual of the workings of a Bidirectional Long Short-Term Memory on the next page.

Figure 3.3: Bidirectional Long Short-Term Memory



The above diagram looks almost twice as complicated as the last, but it is not. We just have the forward layer, or bottom couple of cells in the diagram, go through the inputs from the left to right using the same calculations as the LSTM from earlier. The backward

portion, or the upper pair of cells, are run from right to left. In other words, we have one read forward and the other backward. Then the short-term memories for both sets are paired to give the actual hidden states for the model. We have visualized this with the winding yellow arrows that meet to give the Y outputs.

This change from a unidirectional strategy to having a forwards and backwards layers makes the model more computationally intense but helps address LSTM's memory issues. Although, a BiLSTM alone can only perform so well. It does a better job remembering information but can struggle focusing on what matters most. Therefore, we look to advance this area of NLP by combining this above approach with another human reading strategy, attention mechanisms.

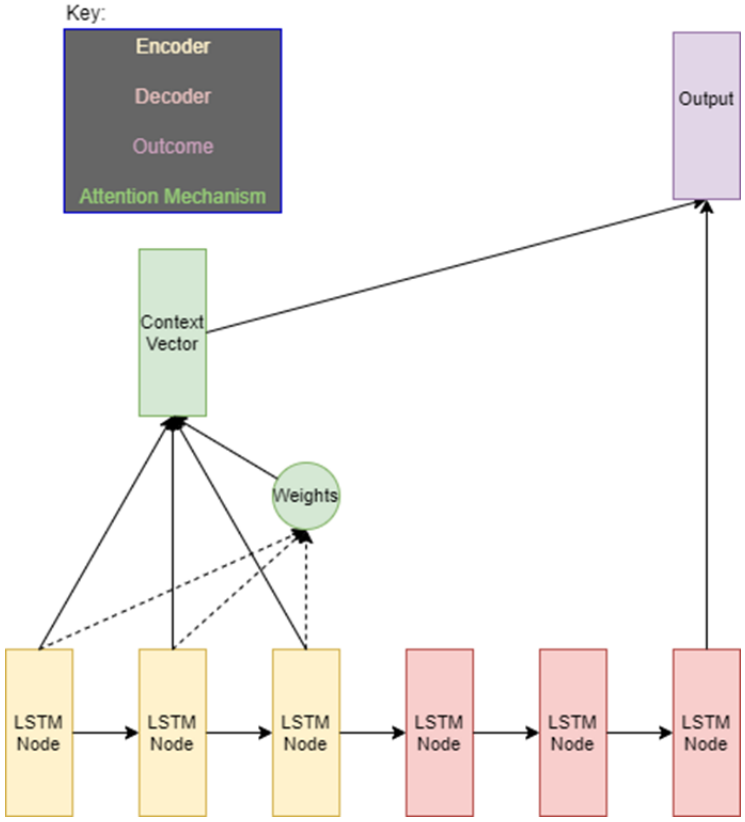
4 The Attention Mechanisms

Some words in a document are more important than others, and a good model should reflect this by looking for the more important terms. People exhibit this behavior when reading anything from a text message to a thesis. Even in proofreading, omitted words common to miss, because our minds will automatically fill in those gaps for us. This process can be seen in action as the last sentence omitted the word "are", but this likely did not make it significantly less comprehensible. Attention mechanisms were made to reflect this behavior [3], and the technique has seen a boom in popularity in the last few years [24] [16] [7] [14] [22]. Their value goes beyond machine reading, as machine vision has also made heavy use of these mechanisms [4]. When looking at images we tend to focus on specific parts more than others, hence the advantage of attention. In fact, entire architectures based on attention [22] have shown to be effective at a variety of tasks. Back to sentiment analysis, this work applies and compares three different attention strategies. The first technique is the original mechanism, additive attention, as formulated by Bahnadau [3].

4.1 Additive

Additive attention is a four-step process, we begin this by generating importance scores for the text. Then we use a softmax function to transfer the information into a domain between 0 and 1, giving the textual weights. Finally, we have the attention mechanism generate the weighted average importance of the terms, creating a context vector. We visualize the way the data flows through this process on the next page.

Figure 4.1: Additive Attention



With this visual aide we can see that the encoder feeds into the attention weights, to get the importance of the terms, as shown by the dashed lines. Then we have the encoder nodes and weights send their information to the context vector, denoted by the solid black lines leading to the green rectangle. Next, we see the vector and the decoder short-term memory join, forming the final output. Additive attention only uses the information from the encoder, shown as the yellow rectangles, to generate the context vector. To better understand the mathematical process of this high-level explanation, the givens are shown below.

h_t: Output from the last encoder node

h_s: The output from all BiLSTM encoder nodes

W_1: A fully connected layer

W_2: A fully connected layer

V: A fully connected layer

We first run the hidden states from every encoder node through a fully connected layer. Then we take the last node's short-term memory through a different FC layer. We add these two outputs together to complete the first step.

$$A = W_1(h_t) + W_2(h_s)$$

Next, we run A through a tanh function and then the last of the fully connected layers. This gives us the attention score, which is the last step before getting the weights.

$$S = score(h_t, h_s) = V(tanh(A))$$

Then we run the score through a softmax layer to bound the values between 0 and 1. This makes the weights which we show as an orange bubble in the diagram.

Finally, we multiply the weights by the output of each encoder cell and then summed to give the context vector.

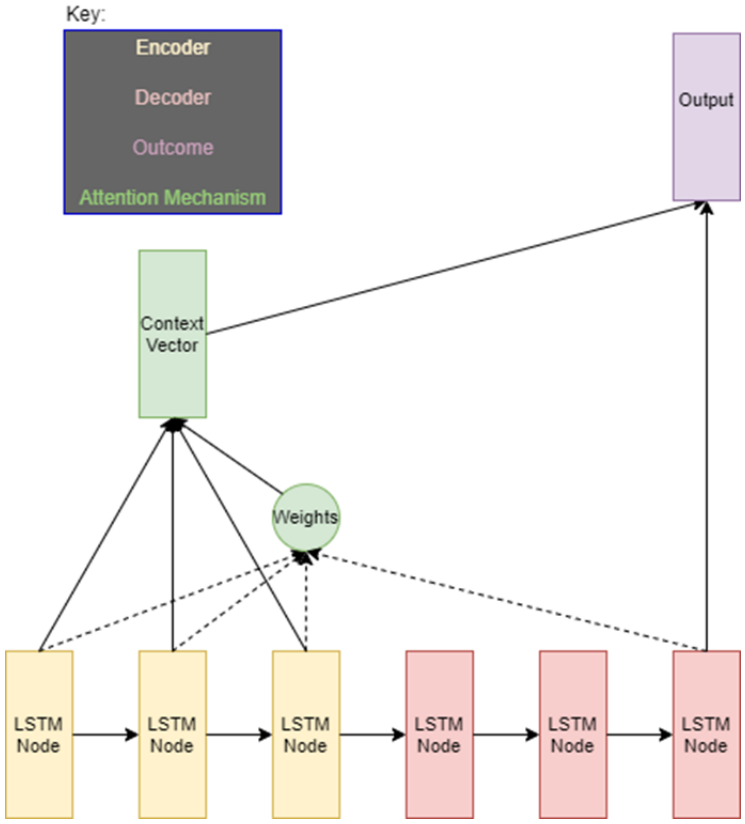
$$C_t = \sum \alpha_{ts} h_s$$

This method was the first and simplest of the attention mechanisms, as it does not involve the decoder at all in its calculations. The Multiplicative attention method differs in that aspect, as it uses the final output from the decoder, along with the encoder hidden states, to make the attention vector.

4.2 Multiplicative

Multiplicative attention, developed by Luong [16], differs from its additive counterpart by generating attention scores from a multiplicative process with the final decoder output. Although, the rest of the math is similar. To highlight the major changes between the methods we provide another graphic below.

Figure 4.2: Multiplicative Attention



In the first step we calculate the weights, but unlike the additive version there are different inputs. We show this above, as the final decoder node feeds into the weight calculation, which was not previously the case. After this change, we list the givens.

h_t: Output from the last decoder node

h_s: The output from all the nodes

W: A fully connected layer

The main difference between the two models is the calculation of the alignment score. For the multiplicative version we use a single dense layer and no activation function.

$$score(h_t, h_s) = h_t W h_s$$

In this layer h_t is different than before since it has changed from the encoder output to the decoder output. From this point on, we have the rest of the layer trace along with the additive version perfectly.

$$\alpha_{ts} = softmax(S) = \frac{exp(score(h_t, h_s))}{\sum_{s'=1}^{S} exp(score(h_t, h_{s'}))}$$

$$C_t = \sum \alpha_{ts} h_s$$

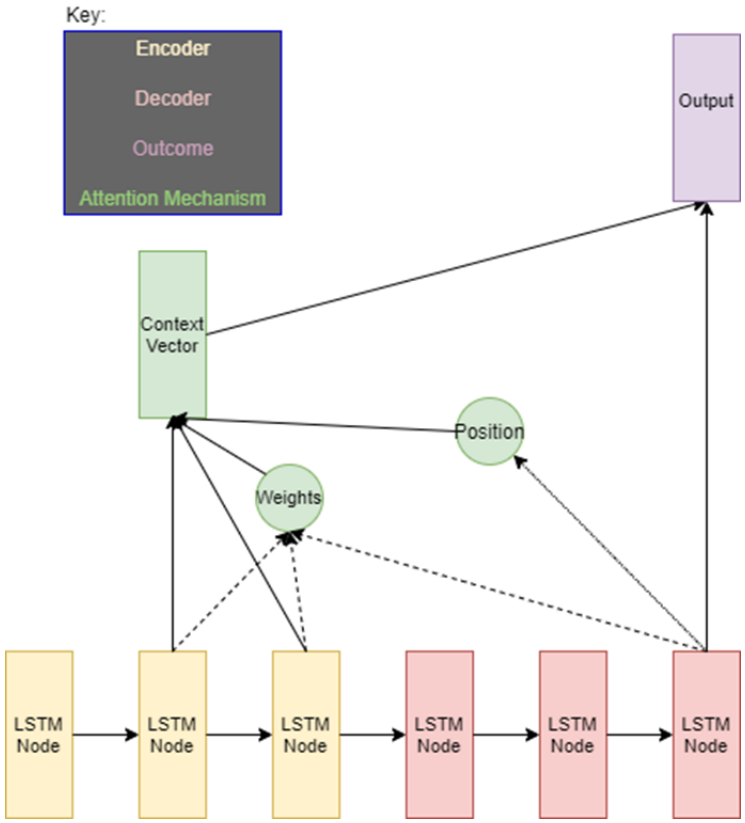
We complete the final calculation for multiplicative attention in the same way as with the additive version, combining the vector with the decoder output. We feed this into the last dense layer and give the text's sentiment. The multiplicative and additive versions are overall similar, but the next attention type is more divergent.

4.3 Local

Local attention differs from the previous two both mathematically and conceptually. In it, there is no reason to attempt paying attention to the entire text. Instead, we only look

at the parts that matter most and solely focus on them. This process is more involved, so we will once again start by showing a high-level graphic before going into the math.

Figure 4.3: Local Attention



There are two visual differences between this graphic and the previous, so we will start with the most apparent change. There is a new section of the attention mechanism, known as position, or aligned position. This finds the window of text that is most meaningful in the corpus. There are two different types of local alignments, and we use what is called predictive alignment. Essentially, the decoder output is used to forecast what should be factored into the context vector. This leads to the other major difference between this visual and the previous, not all the encoding nodes are factored in. To show how all of this happens, we will now walk through the math, once again starting with the givens.

h_t: Output from the last encoder node

h_s: The output from all the nodes

W_1: A fully connected layer

W_2: A fully connected layer

V: A fully connected layer

S: the length of the text

D: The width of the window (hyperparameter)

Despite the difference in givens, this model most resembles multiplicative attention. For the first step we take the decoder output and run it through a dense fully connected layer.

$$N = W_1(h_t)$$

Now we run N through a tanh function and another dense layer.

$$M = V(tanh(N))$$

Next, we take the sigmoid of M and multiply by the review length.

$$p_t = S \times sigmoid(M)$$

This gives us what we will need to position the window later, so that we can focus on the most meaningful part of the text. Specifically, p_t is the predicted position of the highest value section of text and is shown as the orange position bubble in the graphic. Next, we

calculate the attention score. This is a rather straightforward equation that is recognizable from the multiplicative version.

$$\alpha_t = softmax(h_t W_2 h_s)$$

Although, before multiplying and summing the attention vector by h_s, like in the previous models, there is another step. We use the normal distribution with $x = i$,$\mu = p_t$, and $\sigma = \frac{D}{2}$, to generate the windowed view of the encoder output. This is done via the function.

$$\alpha_t(s) = \alpha \times exp(\frac{-(i-p_t)^2}{2\sigma^2})$$

This normal distribution makes the viewing window for the attention mechanism. When multiplied by the output from the encoder, those areas not within the window are squeezed out as the model pays more attention to the distribution's center. With the viewing area formed, the next step is generating the context vector. We do this in the same fashion as the other attention layers, multiplying and summing the components.

$$C_t = \sum \alpha_t(s) h_s$$

This gives us the final context vector for the entire review, which like the others is combined with the decoder's short-term memory for the final output. With the Long Short-Term Memory and Attention layers covered we will go into more detail on each model in this paper.

5 Final Models

We compare five different models, each with increasing levels of complexity. Every network was built with Keras and TensorFlow in python. We utilized pre-constructed layers for portion excluding the attention mechanisms, which were custom coded. All models share the same learning rates, hidden dimensions, nodes, dropout rates, and optimizers. This should allow for better comparisons than if we had inconsistent hyper-parameters. The code for everything in this section can be found **here**. Next, we will go over the first pseudo-control, a basic BiLSTM.

5.1 BiLSTM

The BiLSTM has shown promise in many areas but is incomplete for natural language processing. Nonetheless, it is helpful for us to use as a first step for the versions to come. The model takes in a 3D tensor of shape [BATCH_SIZE, REVIEW_LENGTH, 1], then we run the input through a BiLSTM and a dense layer with a sigmoid activation function. The fully connected layer gives the us the binary output. We show this process below.

Figure 5.1: Basic BiLSTM Model

This is a simple sequential model that feeds directly from one layer to the next. This basic network may seem like it should work well, given what has been discussed involving long short-term memory layers, but it does not. The basic version is missing a way to encode the words so that their meanings may evolve and change as it learns. This is where it is

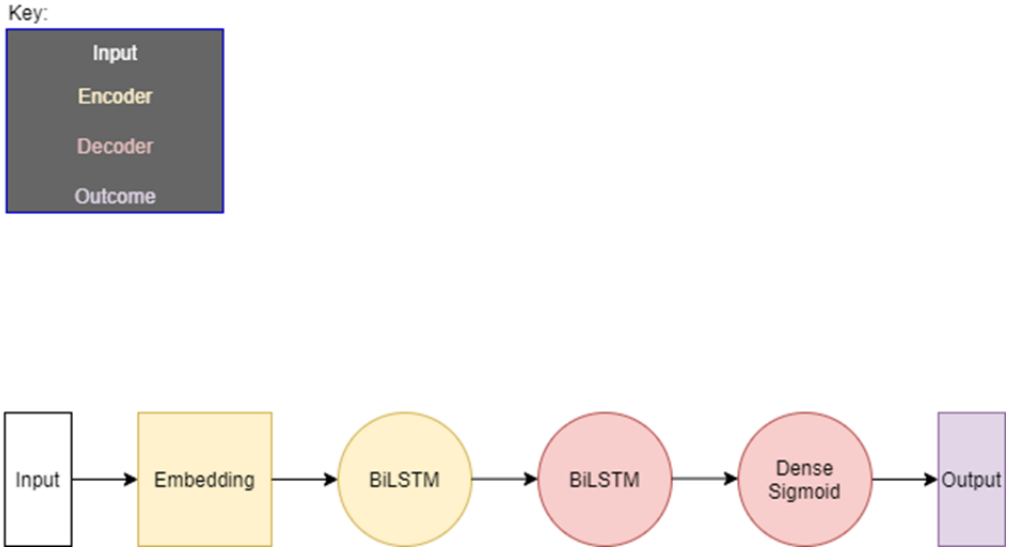
surpassed by the encoder-decoder model.

5.2 BiLSTM Encoder-Decoder

The encoder is a simple addition to the previous model, but with a substantial performance boost. Our new version changes the static encoded words into a learnable space. Specifically, the model takes in a 2D Tensor of size [BATCH_SIZE, REVIEW_LENGTH] which we then run through an embedding layer. This layer makes a space where similar words are mathematically closer to each other, and disparate words are farther away. This allows the data to be learned and understood more organically. Next, we have the embedding output go through a BiLSTM to help translate the terms. These layers form the encoder. As they are trained the embeddings become more accurate, and the BiLSTM returns a more understandable output. Additionally, this encoder can be removed and used in other models that are training on the same data. This encoding strategy allows for the generation of pretrained embeddings. These are useful if there are similar questions about new data, or if there is a subset to be further explored. The embeddings mean that researchers do not have to go through as much of the training process so that the model can learn the words in the data. Pretraining is beyond the scope of this paper but is nonetheless an important advantage of an encoder. The first half of our encoder-decoder model makes the words understandable; the second half uses them to answer the research question.

Decoders take the trained meanings from the encoder and use them to get a specific meaning, like a text's sentiment. This half of our model can take many shapes, but we just use a second BiLSTM and dense layer. This encoder-decoder format mirrors the style of previous work with RNNs [6]. With the encoder and decoder covered, we will tie them together with the diagram on the next page.

Figure 5.2: BiLSTM Encoder-Decoder Model

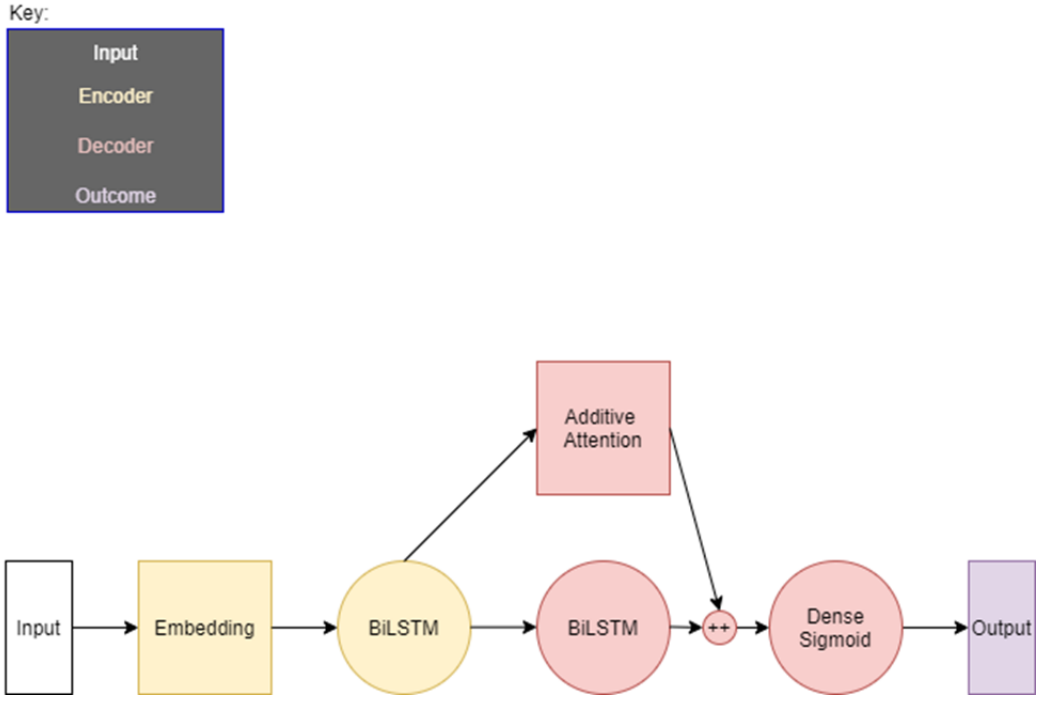


Our full model is simply a sequential path through the encoder and decoder. First, the network intakes the data and runs it through an embedding layer. Then, we pass the data through a BiLSTM which returns the result at each timestep, or for each node in the recurrent sequence, to the decoder BiLSTM. This then links to the final fully connected layer with a sigmoid activation function, getting the categorical output of "good" or "bad". The rest of the models will not follow this sequential format.

5.3 Additive Attention Encoder-Decoder

The additive attention model we use builds off the BiLSTM encoder-decoder but differs in the decoder. After sending the output through the encoder we pass it onto both the BiLSTM and the attention layer. We show the flow of this in the diagram on the next page.

Figure 5.3: Additive Attention Model

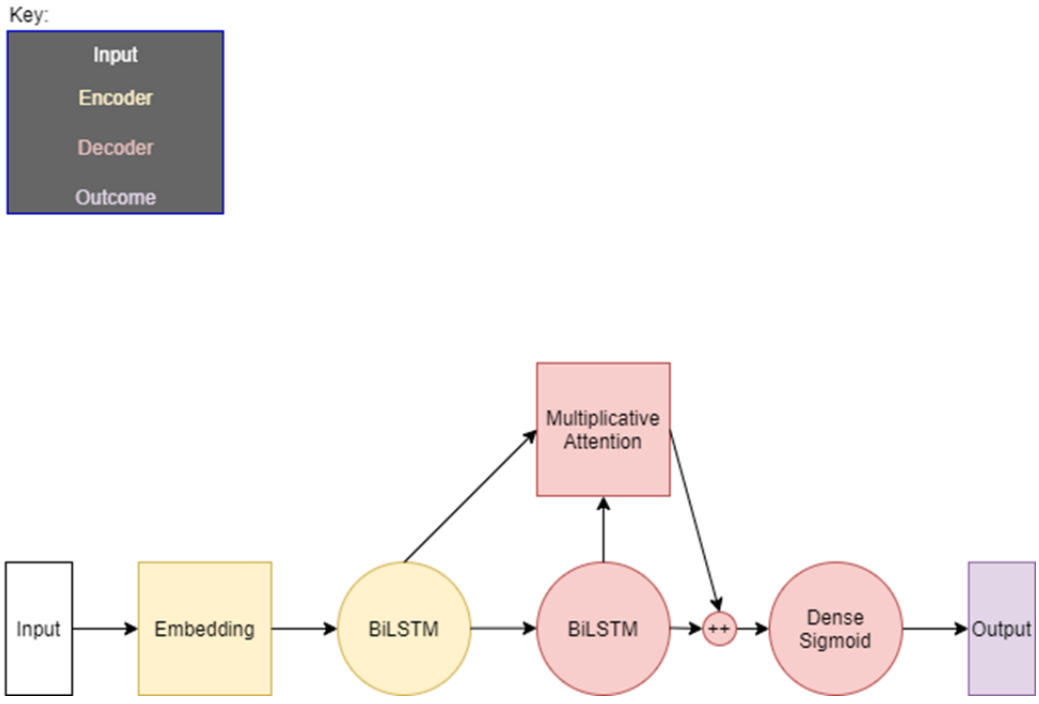


Unlike the previous models the additive version is not sequential, one layer does not just feed into the next. Instead, we have the BiLSTM layer go onto multiple layers which eventually link together. We have the attention mechanism and the decoder BiLSTM both take in the encoder hidden states. Then, concatenate their outputs and pass them onto the final dense layer. Although there are many ways that the final outputs can be combined, like dot products [16], we take after Bahdanau [3] and use concatenation. Finally, we pass this output through a dense sigmoid layer to get the binary result. With these three architectures covered we will move onto multiplicative attention.

5.4 Multiplicative Attention Encoder-Decoder

This is a non-sequential model that has two major differences from the previous that stem just from the swapping of additive to multiplicative attention. The differences within the attention layers have been explored, but these changes also require an alteration in our overall architecture. We visualize this below.

Figure 5.4: Multiplicative and Local Attention Models



The major difference is that both the encoder and decoder BiLSTMs feed into the attention mechanism. Previously, we only had the encoder pass its output to the attention layer. Otherwise, the model architectures are identical. We still concatenate the attention and decoder BiLSTM outputs before passing them onto the dense layer. This multiplicative model has a nearly identical architecture to the last one.

5.5 Local Attention Encoder-Decoder

The local and multiplicative models differ only in the attention mechanism. Despite the local version having the most complicated attention layer, the construction is no more involved than the last. The inputs and outputs from the attention mechanisms come and go from the same layers. The diagram of what sections feed into the others is the same as the previous multiplicative graphic. Therefore, with all the models and their constructions covered, we will move onto the experiment.

6 Experiment

We have presented the math and reasoning for the five different models, but now it is worth discussing why each is necessary for this experiment. Every architecture builds up to the next. Comparing each iteration with the previous highlights the performance change between strategies. Overall, this process of stepwise model building should allow for more robust results. The BiLSTM and encoder-decoder models will act as a baseline comparison group for the attention mechanisms. Additionally, we will contrast each attention strategy with the others. Although, before performing any analysis the data and metrics need to be discussed.

6.1 Data

We use Amazon product reviews of tools from the last decade. These 150,000 reviews were downloaded from the Amazon Review Dataset [2] directly off the Athena console. The data was wildly imbalanced. Most people do not have negative things to say. In general reviews had 5 stars and were overwhelmingly positive. This is great for consumers but bad for data science. If the dataset remained as it was, the imbalanced outcome would simply cause the model to guess positive more often. Before addressing this issue, we had to choose what ratings constitute "good" and "bad". We decided any review that is 3 stars or less is "bad". This choice was influenced by the fact that most reviews are close to 5 stars. This implies that despite 3 being the center value, it is not perceived as an average experience. After the star ratings were converted to a sentiment binary, we downsampled the positive reviews. This gave an equal number of good and bad observations. Therefore, we were left with roughly 60,000 samples to run on. This new dataset was then split into training and testing groups. A fifth of the data, or 11,800 reviews, were reserved for testing. This set

will be used later to evaluate the performance of the models, since metrics created on the training data would only demonstrate the power of overfitting. With the background for the data covered, we will discuss how the models will be evaluated.

6.2 Metrics

With categorical data there is a wide variety of ways to measure performance, and they are often tailored to the question. For example, if a model is trying to detect a disease, it may be important to limit the number of false negatives, even when severely increasing false positives. This would make sure that patients do not go undiagnosed. We have no specific use case for this work, and since the purpose of this paper is to evaluate the performance of the models overall, we have selected three of the most common metrics. Each serves a different purpose and should therefore give a clearer picture of the models' true performances. The most common metric for this type of analysis will be the first we inspect, accuracy.

6.2.1 Accuracy

Accuracy is a metric that only cares about how often the model is right, regardless of how it guesses. Applying it to this work, accuracy is how often our model classifies a positive review as positive and a negative as negative. The calculation is displayed below.

$$\text{Accuracy} = \frac{\text{True Positives+True Negatives}}{\text{Total Observations}}$$

This is usually used as the baseline statistic for model evaluation and is the most well-known. The next statistic is another important value but is not quite as popular.

6.2.2 Precision

Precision is a good measure of how often a model guesses positive correctly. To put the metric in the context of this work, precision tells us the proportion of times the network is correct when predicting a product is liked. The formula is given below.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives+False Positives}}$$

This is often used when it is important that a positive prediction be correct. This could matter if there is an expensive action to undertake from a positive guess, like replacing a factory's equipment. The next metric is similar but with a distinct use case.

6.2.3 Recall

Recall is a measure of how often a model predicts positive when fed data from a positive outcome. Whereas precision is how often when predicting a positive the value is positive, this metric turns that around. In the context of this work, recall indicates how often the model guesses a review is good when the rating sentiment is positive. The formula is shown below.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives+False Negatives}}$$

This type of metric is important when a positive value needs to be predicted correctly. An example could be predicting if a tumor is malignant. Recall ensures as few of people who have tumors are misdiagnosed as possible, not that the greatest number of diagnoses are correct. Now with the three major metrics chosen, it is time to go into how the models performed.

6.3 Results

Before we look at the results it is worth noting that there are five different models and three different architectures. These will all be discussed when analyzing the results table provided below.

Table 6.1: Results

Model	Accuracy	Precision	Recall
Basic BiLSTM	.569	.594	.440
BiLSTM Encoder-Decoder	.826	.804	.794
Additive Attention	.842	.856	.824
Multiplicative Attention	.840	.830	.855
Local Attention	.853	.849	.859

From this table it is visible that the attention architectures outperformed their counterparts. Every attention version had a higher accuracy, precision, and recall. The local attention model appears to be the best. The only exception is precision, where it was outperformed by the additive version. With the most apparent takeaways covered, it is time to go over the results of each model.

The basic BiLSTM showed poor performance. This was expected, as it does not embed or encode the text properly. This baseline shows the need for improving sentiment analysis techniques. There is not much more to discuss as the BiLSTM was included for the purpose of showing the development of more modern models, and to give a very low-level comparison group. The next model is a stronger, more modern, and more representative counterfactual to attention-based techniques.

The bidirectional long short-term memory encoder-decoder performed well in analyzing the sentiment of amazon reviews. It classified nearly 30% better than the basic version in all

metrics. Furthermore, it gave roughly an 80% success rate for every metric, meaning that it is strong on its own, and will make a better comparison group for the attention models.

Additive attention outperformed the simple BiLSTM and was the most precise, but the multiplicative version was not far behind. The additive model was only 1.1 percentage points less accurate than its local counterpart and had a competitive recall. This is not too surprising as all the mechanisms are essentially doing the same thing and should perform similarly. With a .002 difference in accuracy, both simpler attention mechanisms are roughly equivalent and are better than the BiLSTM. There is one architecture left to be discussed, and it is the strongest of them all.

The local attention model performed best. It was over 3% more accurate than the BiLSTM, and about 1% better than the other attention models. Its recall tells roughly the same story. Although the precision was worse than the additive model, it is still the strongest of the three. Also, this is the most complicated model, making it take slightly longer to train. Overall, attention-based models seem to perform better, but by only a few percentage points.

7 Conclusion

We presented the reasons for the importance of sentiment analysis in the modern world. The implementation of natural language processing can have a major impact on both corporations and people. Being able to efficiently analyze mass media information can help us better understand the needs, wants, and feelings of those around us. Corporations can adjust pricing and focus on bringing their goods to communities in need. The government can get an idea of what is most important to voters and what issues need to be addressed. We as individuals may be able to see that those who write inflammatory remarks are in the minority, despite their voices sounding so loud. There are more possibilities for how our society can benefit from NLP than can be simply listed, and the tools for machine reading are still growing.

The modeling approaches we take are in line with modern implementations of supervised natural language processing. Additionally, previous works using LSTMs, BiLSTMs, and encoder-decoders were all explored. With this foundation set, we introduced attention mechanisms. We explained these strategies both conceptually and mathematically. Finally, we created and tested five different models.

To compare the results, each network had identical hyper-parameters, training data, and testing data. The findings showed that the attention models performed superiorly to their counterparts, albeit only by a small margin. Every attention layer was around 2-3% more accurate than the BiLSTM encoder-decoder model. Additionally, the attention architectures generally had better precision and recall as well. Within the attention mechanisms, local had the best results, but took the longest to train. Therefore, we conclude that attention does allow for a small but meaningful performance improvement over non-attentive methodology. Although future work could explore these results further in several ways.

Looking at different tasks, data, and more attention heavy models could help to explore

the full advantages of this new technique. This model was only built and tested on one dataset for one NLP task. Future work could look at other tasks, such as machine translation, to see if the results hold or are bolstered in those areas. Additionally, this paper did not compare some other transformer architectures like discussed in Attention is All You Need [22], where the models are almost exclusively built on attention mechanisms. These are all beyond the scope of this work but are worth looking into in the future.

Bibliography

[1] Basemah Alshemali and Jugal Kalita. "Improving the reliability of deep neural networks in NLP: A review". In: *Knowledge-Based Systems* 191 (2020), p. 105210.

[2] Amazon. *Amazon Customer Reviews Dataset*. URL: `https://s3.amazonaws.com/amazon-reviews-pds/readme.html`.

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

[4] Marco Cannici et al. "Attention mechanisms for object recognition with event-based cameras". In: *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2019, pp. 1127–1136.

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

[6] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[7] Andrea Galassi, Marco Lippi, and Paolo Torroni. "Attention in natural language processing". In: *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[8] Martin Gerlach, Hanyu Shi, and Lu A Nunes Amaral. "A universal information theoretic approach to the identification of stopwords". In: *Nature Machine Intelligence* 1.12 (2019), pp. 606–612.

[9] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[10] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. "A convolutional neural network for modelling sentences". In: *arXiv preprint arXiv:1404.2188* (2014).

[11] Subbu Kannan et al. "Preprocessing techniques for text mining". In: *International Journal of Computer Science & Communication Networks* 5.1 (2014), pp. 7–16.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[13] Ela Kumar. *Natural language processing*. IK International Pvt Ltd, 2011.

[14] Jiangming Liu and Yue Zhang. "Attention modeling for targeted sentiment". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. 2017, pp. 572–577.

[15] Yang Liu and Sujian Li. "Recognizing implicit discourse relations via repeated reading: Neural networks with multi-level attention". In: *arXiv preprint arXiv:1609.06380* (2016).

[16] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. "Effective approaches to attention-based neural machine translation". In: *arXiv preprint arXiv:1508.04025* (2015).

[17] Usman Malik. *Removing Stop Words from Strings in Python*. 2021. URL: https://stackabuse.com/removing-stop-words-from-strings-in-python/#:~:text=NLTK%20supports%20stop%20word%20removal,stop%20words%20provided%20by%20NLTK.

[18] Tomáš Mikolov et al. "Recurrent neural network based language model". In: *Eleventh annual conference of the international speech communication association*. 2010.

[19] Amr Mousa and Björn Schuller. "Contextual bidirectional long short-term memory recurrent neural network language models: A generative approach to sentiment analysis". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. 2017, pp. 1023–1032.

[20] Apeksha Shewalkar. "Performance evaluation of deep neural networks applied to speech recognition: RNN, LSTM and GRU". In: *Journal of Artificial Intelligence and Soft Computing Research* 9.4 (2019), pp. 235–245.

[21] Michal Toman, Roman Tesar, and Karel Jezek. "Influence of word normalization on text classification". In: *Proceedings of InSciT* 4 (2006), pp. 354–358.

[22] Ashish Vaswani et al. "Attention is all you need". In: *arXiv preprint arXiv:1706.03762* (2017).

[23] Jin Wang et al. "Dimensional sentiment analysis using a regional CNN-LSTM model". In: *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers)*. 2016, pp. 225–230.

[24] Yequan Wang et al. "Attention-based LSTM for aspect-level sentiment classification". In: *Proceedings of the 2016 conference on empirical methods in natural language processing*. 2016, pp. 606–615.

[25] Wenpeng Yin et al. "Comparative study of CNN and RNN for natural language processing". In: *arXiv preprint arXiv:1702.01923* (2017).