Deep Neural Networks for Added Emphasis

Niraj Jayant
Microsoft Corporation
Sunnyvale, CA, USA
nirjay@stanford.edu

Jonathan Kotker
Microsoft Corporation
Sunnyvale, CA, USA
jkotker@stanford.edu

Abstract

In this project, we focus on the task of emphasis addition, as it relates to the slides of a PowerPoint presentation. We train a deep neural network on a database of publicly available presentations to determine which text on a slide can be emphasized, through either bolding, italicizing, or underlining. We explore how best to represent this data set as an input to the network, and conclude that a slide-level representation of the slide text produces better results than a paragraph-level representation of the slide text. Finally, we show that a recurrent neural network with bi-directional LSTM units and character-level embedding is able to achieve F_1-scores of 15% through 26% during training, and of 21.31% on a test data set.

1 Introduction

Microsoft PowerPoint is a popular presentation program, used widely in educational and office settings to display and disseminate information. The PowerPoint development team recently released the PowerPoint Designer feature [7], which examines a user-created slide and proffers suggestions to improve the design and the layout of the slide, among other things. To construct these suggestions, PowerPoint Designer leverages machine learning and natural language processing (NLP).

A common critique of PowerPoint presentations is colloquially referred to as 'death by PowerPoint' [11], whereby creators often fill their slides with large chunks of text. This renders the text ineffective for consumption by an audience that is attempting to simultaneously listen to the presenter *and* read the slides. In the spirit of PowerPoint Designer, this project uses machine learning to explore *emphasis addition* as a possible solution to 'death by PowerPoint'.

The problem of emphasis addition is to determine which elements of a slide should be emphasized, since each element could capture some important point of the slide. In this project, we focus on using NLP and deep learning to find those *words* of a slide that should be emphasized. For the purposes of this project, we consider a word as emphasized if it has been either bolded, italicized, or underlined.

2 Background and Related Work

2.1 Recurrent Neural Networks

In this project, we employed recurrent neural networks (RNNs), a family of deep neural networks that process sequences of inputs [9]. In a basic, 'vanilla' RNN, each hidden state $h^{(t)}$ is dependent on the previous hidden state $h^{(t-1)}$ and the sequence element at time t, $e^{(t)}$, according to the equation

$$h^{(t)} = \sigma(h^{(t-1)}W_h + e^{(t)}W_e + b_1),$$

where W_h and W_e are trainable matrices, and b_1 is a trainable bias vector. Finally, the prediction $\hat{y}^{(t)}$ at time t is given by

$$\hat{y}^{(t)} = \text{softmax}(h^{(t)}U + b_2),$$

where U is a trainable matrix and b_2 is a trainable bias vector. Figure 1 depicts the architecture of a vanilla recurrent neural network, as depicted in [9]. Our vanilla RNNs were trained using cross-entropy loss. Recurrent neural networks are especially relevant for this project, since we view the text on a slide as a sequence of tokens, with the emphasis of each token possibly dependent on the emphases of previous tokens.

2.2 Long Short-Term Memory Units

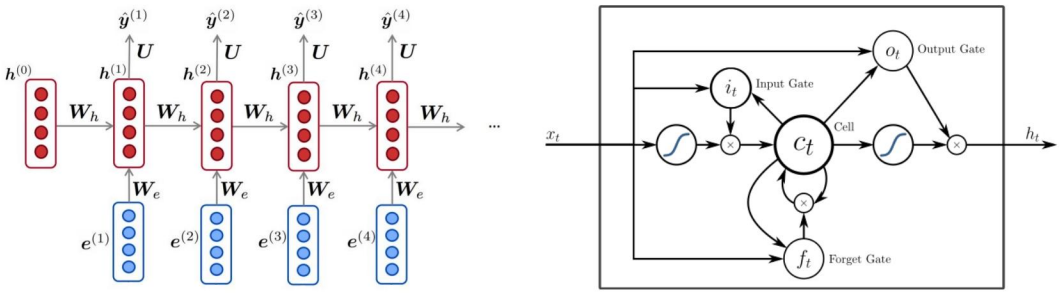


Figure 1: The architectures of a vanilla recurrent neutral network, as depicted in [9], and of a long short-term memory unit, as depicted in [13].

However, a vanilla RNN model often runs into the problems of either vanishing or exploding gradients, which render it unable to use information from more than a few timesteps into the past. To combat this, we can construct an RNN out of long short-term memory (LSTM) units [10], [13], which can maintain short-term memory over a long time. Each unit achieves this through the use of an input gate $i^{(t)}$, which controls its dependence on the input $x^{(t)}$, a forget gate $f^{(t)}$, which controls its dependence on previous inputs and units, and an output gate $o^{(t)}$, which controls the output of the unit. The equations that govern these gates are given by

$$
\begin{aligned}
i^{(t)} &= \sigma(x^{(t)}W_i + h^{(t-1)}U_i + b_i), \\
f^{(t)} &= \sigma(x^{(t)}W_f + h^{(t-1)}U_f + b_f), \\
o^{(t)} &= \sigma(x^{(t)}W_o + h^{(t-1)}U_o + b_o), \\
c^{(t)} &= f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tanh(x^{(t)}W_c + h^{(t-1)}U_c + b_c), \\
h^{(t)} &= o^{(t)} \circ \tanh(c^t).
\end{aligned}
$$

We extended this model further by using bi-directional LSTM units, which depend on both previous units and future units. This additional context was especially useful in deciding whether a group of words needed to be emphasized.

2.3 Named Entity Recognition

The task of *named entity recognition* is concerned with identifying the words in a sentence that refer to named entities, such as places and people. This task is similar to the task of emphasis addition, as the latter is concerned with identifying the words that should be emphasized. The third assignment of CS224N employed both a vanilla recurrent neural network and an RNN with gated recurrent units for this task, which served as inspirations for our approach: A gated recurrent unit, it must be noted, is a simpler version of an LSTM that does not have an output gate. However, in contrast to named entity recognition, emphasis addition uses only two labels: *emphasized* or *not emphasized*.

Since the two tasks are similar, this project also used the same metric as named entity recognition: the F_1-score, which is the harmonic mean of precision and recall [12]. However, unlike named entity recognition, we do not measure entity-level F_1, since there is no equivalent concept of an entity in the task of emphasis addition.

2.4 Word Embeddings

The inputs to the recurrent neural networks detailed in section 2.1 are *word embeddings*, which are representations of words that aim to capture the similarity between words. In particular, we used the GloVe word vectors, which capture co-occurrence statistics of words in a given corpus [8]. We used the GloVe word vectors that were trained on a corpus that comprised both data extracted from Wikipedia in 2014 and data from the fifth edition of the English Gigaword data set [3].

2.5 Character Embeddings

In our survey of the data set, we noticed that technical PowerPoint slides can have a lot of terms that do not have GloVe embeddings. However, these terms can prove useful in emphasis addition, as the first occurrence of a technical term is usually emphasized. For a more holistic representation of a word, we used *character embeddings* along with the GloVe word embeddings. We followed the approach used in [4], which used an RNN model with bi-directional LSTM units and multiple layers, whose inputs were word *and* character embeddings. We adapted the implementation from a sequence-tagging tutorial [1] to create and train our models. This model can learn which sequences of *characters* need emphasis. It can also distinguish between information that may be lost when using word embeddings, such as different cases and spellings, endowing it with more flexibility and learning power.

3 Approach

3.1 Data Set Acquisition

The first important step in this project was acquiring the data set over which we would train and test our models. The PowerPoint development team maintains a large, internal database of publicly available presentations to use for test cases, which allows for the extraction of our desired data set. The internal database has around 750,000 PowerPoint presentations, and these presentations comprise 12 million slides with text. We wanted to construct a pipeline that would be able to shred these presentations and turn them into the token sequences that our models would expect.

PowerPoint maintains a Visual Basic API that allows external developers to work with the object model that represents a presentation [6]. The `pywin32` (or 'Python for Win32') Python package wraps around this API: It allowed us to build a Python script that iterates over all of the slides in a presentation, then over all of the shapes in each slide, and finally over all of the *text runs* within each shape. A *text run* is defined to be a sequence of characters that have the same formatting: Accordingly, many slides can have several adjacent text runs. As a result, the Python script was also able to detect runs that were emphasized. Finally, in an attempt at data sanitization, the Python script also removed URLs and e-mail addresses.

3.2 Data Set Format

With the data set acquisition pipeline in place, we needed to settle upon a format for the data set.

3.2.1 Paragraph-Level Representation

In our first attempt, we split each slide at the paragraph level. Each paragraph comprised a series of text runs, and each token in a text run was labeled as either `1` if emphasized or `0` if not emphasized. Each paragraph was separated by a newline. We employed the Stanford Tokenizer from the Stanford Natural Language Processing Group [2] to extract the tokens from the text on a slide. Figure 2(a) shows an example slide from the internal database of publicly available PowerPoint presentations, and figure 2(b) shows a sampling of the resulting data.

With this initial representation, we ran an exploratory baseline model that used a vanilla RNN architecture. The architecture was given 50-dimensional embeddings and had a hidden state dimension of 300. The model used a learning rate of 0.001, a dropout rate of 0.1, and a maximum sequence (or paragraph) length of 308. We trained this model over 10 epochs with a batch size of 32 paragraphs.

With a preliminary data of about 250,000 tokens in the training set, 60,000 tokens in the validation set, and 60,000 tokens in the test set, the model was able to achieve a baseline F_1-score of 13%.

3.2.2 Slide-Level Representation

On examining some of the predictions from the model, we found that the model was losing too much context by not having sentences from the same slide associated in any way. Put differently, since the model was not provided the context of which slide a sentence came from, its ability to predict emphasis patterns was harmed. Figure 3 visualizes a good example of this. We reasoned that this was because generally, users who emphasize text tend to do so consistently over one slide and even across multiple slides.

With this in mind, we altered our data set representation to instead split each slide at the slide level: Each *slide* was now separated by a newline, instead of each paragraph. This representation gave the model 'mega-sentences' that would keep all of the tokens of a slide together, allowing the model to take the entire slide into consideration when training and testing. Figure 2(c) shows a sampling of the new data that was derived from the example slide of figure 2(a). Additionally, we augmented our RNN architecture to use LSTM units, as detailed in section 2.2.

Finally, we performed a quick analysis of our data set to determine the maximum length of any input sequence, and the maximum length of any token: The former would dictate the number of hidden LSTM units in the RNN, while the latter would dictate how many character embeddings had to be concatenated to each word embedding. The charts in figures 4 and 5 are histograms of slide-level 'mega-sentence' lengths and word lengths in our data set. From these histograms, we elected to use 200 as the maximum length of an input sequence, and 15 as the maximum length of a token in the sequence: Both values represented the 99th percentile of values in their respective categories. Any slide or token with lengths that exceeded these thresholds was truncated.

The F_1-score of the new model rose into the 17%-18% range. Since we saw improvements from the baseline model, we continued to invest in the model: We moved from using LSTM units to bi-directional LSTM units, and also added multiple layers. These changes to the complexity gave us incremental improvements, but no change vastly affected our F_1-score. As a result, we re-examined our data set and the source slides to determine if there were any other issues with our data set, or if there was any other information we may be losing.

3.2.3 Slide-Level Representation with Placeholder Types

By default, PowerPoint slides have placeholders, which are special shapes with dotted borders where users can add content [5]. These placeholders are of twenty-one types, which range from titles and subtitles, to text and media. Figure 6 shows a PowerPoint slide that has a title placeholder, where a user can add a title, and a content placeholder, where a user can add any content of any type.

We noticed that the data set acquistion hitherto did not consider the type of the placeholder where the text is actually located, which is important since, for example, text in titles tend to be differently emphasized than text in a text placeholder. We thus ran a final round of data set acquistion, where we included information about the placeholder type along with each paragraph of a slide. Figure 2(d) shows a sampling of the new data set that was derived from the example slide of figure 2(a).

When constructing the inputs to our RNNs, we encode this placeholder type as a 22-dimensional one-hot vector: The extra dimension accounts for text that did not live in a placeholder, but lived in a regular shape. This one-hot vector was then concatenated to the word embedding and the character embeddings for each word, the result of which was our final representation of a word. Figure 7 depicts the final architecture of the model that we used in our subsequent experiments.

4 Experiments

4.1 Human-Level Performance

Before proceeding with our experiments on the final model, we needed to determine how a human would perform on the task of emphasis addition. In particular, we wanted to estimate a range of F_1-scores obtained when a human was asked to examine the text of a slide, devoid of any emphasis, and

to determine which tokens needed to be emphasized. The two authors of this paper were provided with 29 such slides, and obtained F_1-scores of 53.05% and 55.19%. From this, we hypothesized that F_1-scores around 50% could be considered 'human-level' performance, though we would need more human-labeled inputs to get more accurate scores.

4.2 Hyperparameter Search

With our final model architecture in place, we set out to discover how tweaking the hyperparameters of this architecture would change its performance, and also to determine the final set of hyperparameters that we would use to understand the performance of the model on a test data set. For the experiments that follow, the training data set had 2,127,484 labeled tokens, while the validation data set had 531,887 labeled tokens. We also tweaked one hyperparameter a time, of a default model that had one layer, a hidden state size of 50, a learning rate of 0.001, and a dropout rate of 0.5, and that used a batch size of 32 and was trained for 10 epochs.

4.2.1 Batch Size

Figure 8 demonstrates how the accuracy, training loss, and F_1-score of the model changed during ten training epochs with a change in the size of batches fed to the model. The larger the batch size, the better the model performed in all three metrics: As a result, we chose 128 as the batch size in the final set of hyperparameters.

4.2.2 Hidden State Size

Figure 9 demonstrates how the accuracy, training loss, and F_1-score of the model changed during ten training epochs with a change in the size of the hidden states, or of the dimensions of the LSTM units. Though the model with a hidden state size of 100 showed a dip in performance at epoch 5, it eventually outperformed the model with a hidden state size of 50: We thus chose 100 as the hidden state size in the final set of hyperparameters.

4.2.3 Number of Layers

Figure 10 demonstrates how the accuracy, training loss, and F_1-score of the model changed during ten training epochs with a change in the number of layers in the network. The one-layered model appeared to lose accuracy with more epochs. Additionally, this model and the two-layered and five-layered models seemed to be overfitting based on the F_1-score metric, which went down significantly with more epochs. The three-layered model, however, remained steady in all three metrics: Accordingly, we chose 3 as the number of layers in the final set of hyperparameters.

4.2.4 Epochs

Table 1 shows the best accuracy and F_1-score of the default model during its training, when trained for different epochs. We observed that the longer the model was trained for, the better its performance, which informed our choice of 50 epochs in the final set of hyperparamters.

Epochs	**Accuracy**	**F_1-Score**
10	77.11	25.20%
20	77.39	25.36%
50	77.39	26.32%

Table 1: The best values for the accuracy and F_1-score of the default model, when trained for different epochs.

4.2.5 Learning Rate

Figure 11 demonstrates how the accuracy, training loss, and F_1-score of the model changed during ten training epochs with a change in the learning rate of the model. While the model with a learning rate of 0.5 appeared to keep steady in the metrics of accuracy and F_1-score as training progressed, we were unable to obtain information on the training loss, which was logged as `NaN`. Since the

model with a learning rate of 0.1 was able to show a similar performance, *and* we had the training loss information for this model, the final set of hyperparameters included 0.1 as the learning rate.

4.2.6 Dropout Rate

Figure 12 demonstrates how the accuracy, training loss, and F_1-score of the model changed during ten training epochs with a change in the dropout rate. The three metrics of models with all three dropout rates exhibited similar behaviors: In particular, all three dropout rates showed some degree of overfitting, as depicted by their F_1-scores. Since their performances were so similar, we chose the middle value of 0.5 as the dropout rate in the final set of hyperparameters.

4.3 Test Set Results

We trained the final model with the set of hyperparameters that were determined by the preceding experiments, and then used this model to make predictions on a test set of 664,864 labeled tokens. The model was able to achieve an F_1-score of 21.31% on the test set, in both of the two separate runs where we used 50-dimensional and 300-dimensional GloVe word embeddings. While a decent score, and a significant improvement over the F_1-score obtained by the baseline model, this unfortunately does not reach the 'human-level' performance determined in section 4.1.

5 Conclusions

5.1 Main Takeaways

Through our experiments in both the construction of the final model and the determination of the hyperparameters of this model, we conclude that a recurrent neural network with bi-directional LSTM units was able to achieve a respectable performance on the task of emphasis addition. We also discovered that the structure of the data set proved to be important in achieving better performance, with a slide-level representation of each slide showing an improvement over a paragraph-level representation; additionally, more features proved useful, with the placeholder type information also augmenting the performance.

5.2 Future Work

There remain several avenues of future work for this project:

1. One of the biggest problems that we faced was the cleanliness and the consistency of the source data. The task of emphasis addition is a very subjective task, with different users considering different elements of a slide important. This is in contrast to the ostensibly similar task of named entity recognition, which is not subjective and has correct answers. When we went through some of the slides in our source data set, we would often see slides where *all* of the text was emphasized, which could have proved harmful to our model's understanding of the problem. To alleviate these issues, we could:
   (a) Remove slides where the number of emphasized words exceeds a certain threshold;
   (b) Remove slides that are not written in the language under consideration: Even though we had queried the database for English presentations as a starter, we discovered that there were a few German presentations, which points to an error in labeling;
   (c) Narrow down the original problem: It may prove easier to train a model to, for example, emphasize certain numbers, such as percentages and temperatures, or to emphasize the first instance of a word in a slide that defines the word, or to emphasize the words that summarize the steps of a process; or
   (d) Use a set of 'good' examples of presentations, and synthesize new presentations from this set, to provide more training data to the model.
2. Another problem was the metric that we chose to understand the performance of our model: We chose the F_1-score because it was easy to calculate automatically, and also because it was used by the similar task of named entity recognition. However, given how imperfect the training data set is, the F_1-score may not be the best representation of how well the model

is performing. We would need to find a new metric that is easy to calculate automatically, but that also handles the inherent messiness and subjectivity of the task.

One possible metric is the *keep rate*, which is used by PowerPoint Designer to understand how well its models are performing: It is the percentage of slides suggested to a user that were eventually kept by the user in their presentation. If the inclusion of an emphasis addition feature to PowerPoint Designer improves its keep rate, that would count towards the success of the model. Additionally, if the user decides *not* to keep the suggested slide, then that sends another useful signal to the model.

3. Finally, we had moved from representing slides at the paragraph level to representing them at the slide level. Perhaps the model can learn more about how users prefer to emphasize by considering the *entire* presentation at a time.

Acknowledgments

We would like to thank Daniel Cheung, Derek Johnson, Sagar Patel, and Priyanka Sinha at Microsoft PowerPoint for helping us shape the idea for this project, and for providing guidance on approaches and available tools that helped us stand up the data set acquisition. We would also like to thank other PowerPoint employees Gus Logsdon and Smarth Behl, and the Natural Language and Experiences (NLX) team at Microsoft, for various pieces of great advice and prototyping help. Finally, we would like to heartily thank the staff of CS224N for a wonderful, fun, and engaging class, and especially Tim Shi, Jay Whang, and our mentor Kevin Clark, for their insights into our project.

References

[1] Guillaume Genthial. Sequence Tagging with Tensorflow. https://guillaumegenthial.github.io/sequence-tagging-with-tensorflow.html, April 2017.

[2] The Stanford Natural Language Processing Group. Stanford Tokenizer. https://nlp.stanford.edu/software/tokenizer.shtml, 2015.

[3] Christopher D. Manning Jeffrey Pennington, Richard Socher. GloVe: Global Vectors for Word Representation. https://nlp.stanford.edu/projects/glove/, August 2014.

[4] Xuezhe Ma and Eduard H. Hovy. End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. *CoRR*, abs/1603.01354, 2016.

[5] Microsoft. Add a placeholder to a layout. https://support.office.com/en-us/article/Add-a-placeholder-to-a-layout-a8d93d28-66cb-43fd-9f9d-e12d0a7a1f06, 2016.

[6] Microsoft. PowerPoint VBA reference. https://msdn.microsoft.com/en-us/vba/vba-powerpoint, June 2017.

[7] Microsoft. The evolution of PowerPoint – introducing Designer and Morph. https://blogs.office.com/en-us/2015/11/13/the-evolution-of-powerpoint-introducing-designer-and-morph/, November 2015.

[8] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

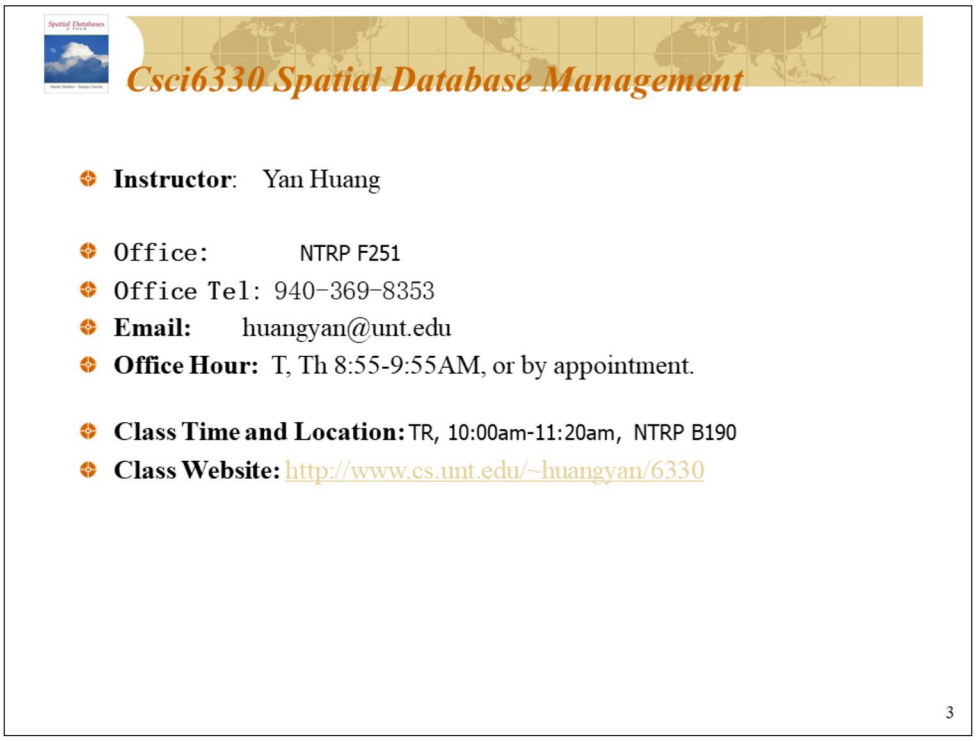
[9] Abigail See. Lecture 8: Recurrent Neural Networks and Language Models. http://web.stanford.edu/class/cs224n/lectures/lecture8.pdf, February 2018.

[10] Richard Socher. Lecture 9: Vanishing Gradients and Fancy RNNs (LSTMs and GRUs). http://web.stanford.edu/class/cs224n/lectures/lecture9.pdf, February 2018.

[11] WhatIs.com. death by PowerPoint. http://whatis.techtarget.com/definition/death-by-PowerPoint, August 2013.

[12] Wikipedia. F1 score. https://en.wikipedia.org/wiki/F1_score, 2018.

[13] Wikipedia. Long short-term memory. https://en.wikipedia.org/wiki/Long_short-term_memory, March 2018.




```
Csci6330    1
Spatial 1
Database    1
Management  1

Instructor  1
:   0
Yan 0
Huang   0

Office  1
:   1
NTRP    0
F251    0

Office  1
Tel 1
:   0
940-369-8353    0

Email   1
:   1

Office  1
Hour    1
:   1
T   0
,   0
Th  0
8:55-9:55   0
AM  0
,   0
or  0
```

```
Csci6330    1
Spatial 1
Database    1
Management  1
Instructor  1
:   0
Yan 0
Huang   0
Office  1
:   1
NTRP    0
F251    0
Office  1
Tel 1
:   0
940-369-8353    0
Email   1
:   1
Office  1
Hour    1
:   1
T   0
,   0
Th  0
8:55-9:55   0
AM  0
,   0
or  0
```

```
3
Csci6330    1
Spatial 1
Database    1
Management  1
4
Instructor  1
:   0
Yan 0
Huang   0
Office  1
:   1
NTRP    0
F251    0
Office  1
Tel 1
:   0
940-369-8353    0
Email   1
:   1
Office  1
Hour    1
:   1
T   0
,   0
Th  0
8:55-9:55   0
AM  0
,   0
or  0
```

Figure 2: (a) An example slide from the internal database of publicly available PowerPoint presentations. (b) *Paragraph-level representation*: Each paragraph of text runs is separated by two newlines, and each text run token is on one line. A token is labeled with either `1` if emphasized in the original slide, or `0` otherwise. Note that the email address has been stripped out in an attempt at data sanitization. (c) *Slide-level representation*: Each *slide* is now represented as one 'mega-sentence', and each mega-sentence is separated by two newlines. (d) *Slide-level representation with placeholder types*: Note the new tokens `3` and `4`: These are the types of the placeholders where the subsequent tokens originated. Here, `3` and `4` indicate a title placeholder and a content placeholder respectively.

*33 . Protection **from** drugs*

34 . Sexual exploitation

*35 . **Child trafficking***

Figure 3: A visualization of an exemplar result where the model was unable to properly predict emphasis patterns. Italicized words indicate words that were emphasized in the original slide; bolded words indicate words that the model predicts need to be emphasized: Ideally, in this visualization, words are either in normal typeface, or in *both* bold and italics. Note here that even though the three bullet points were *on the same slide*, *and* have been consistently emphasized, the model unevenly predicted the words that were emphasized. We reasoned that this may have been because if the model had encountered a similar slide during training, it was trained over one bullet point on the slide at a time, instead of over the whole slide at once.

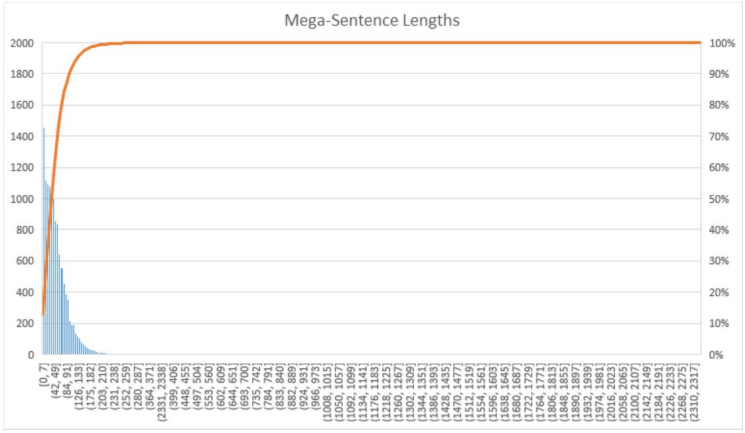


Figure 4: A histogram of slide-level mega-sentence lengths: The 99th percentile of these lengths is 196, and thus we elected to use 200 as the maximum length of an input sequence to the RNN.

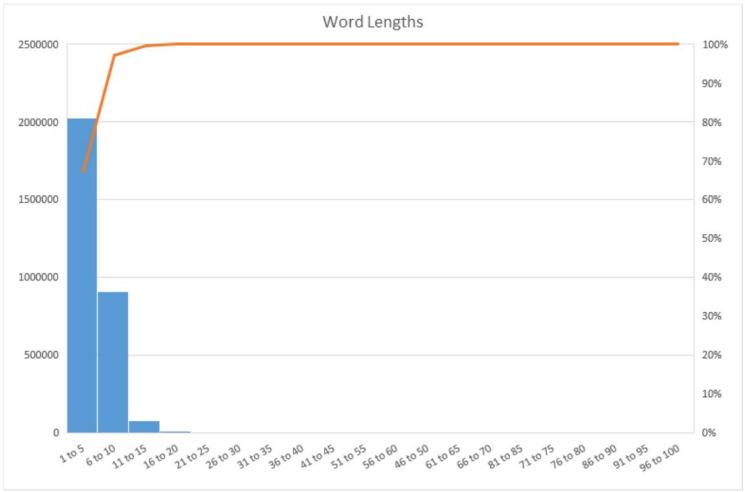


Figure 5: A histogram of word lengths: The 99th percentile of these lengths is 15, which is the value we elected to use as the maximum length of any token in an input sequence.

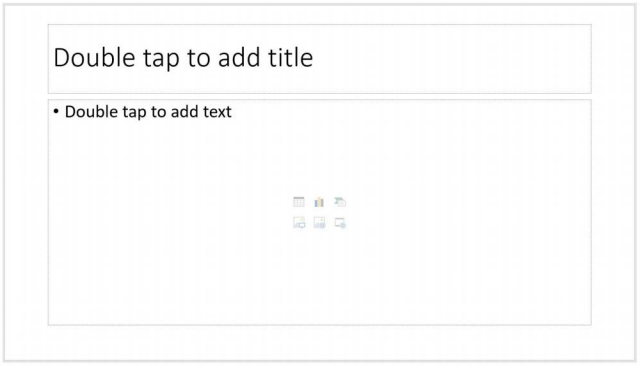


Figure 6: A default PowerPoint slide, with a title placeholder and a content placeholder.

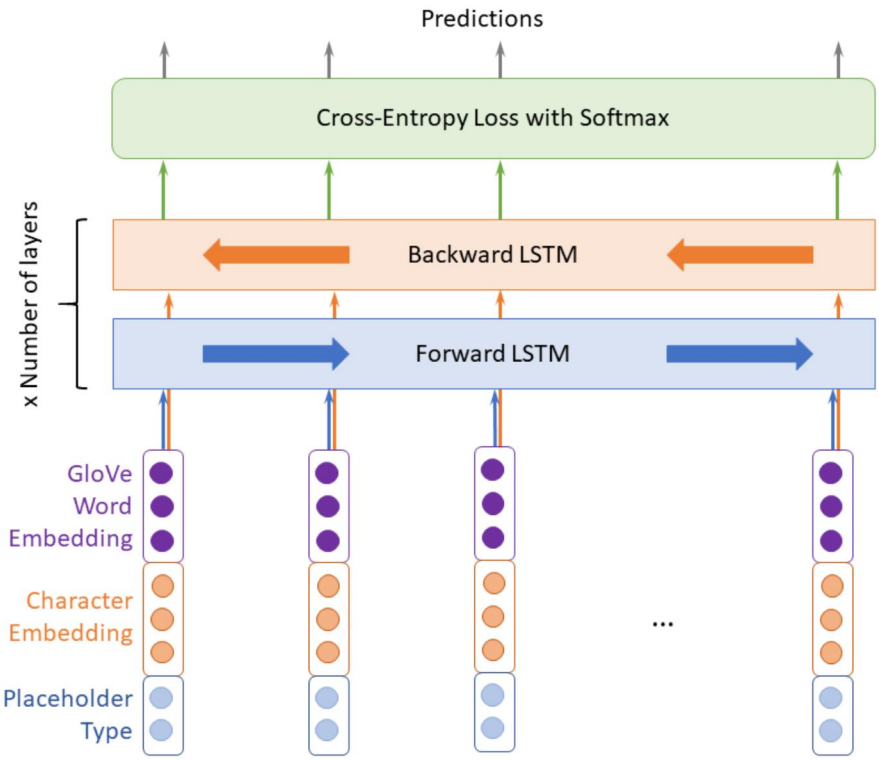


Figure 7: The final architecture of the model used in subsequent experiments.

Figure 8: The accuracy, training loss, and F_1-score of models with different batch sizes in the ten training epochs. The accuracy and F_1-score were calculated on a validation data set; the training loss was calculated on a training data set. The batch size of 128 performed the best.

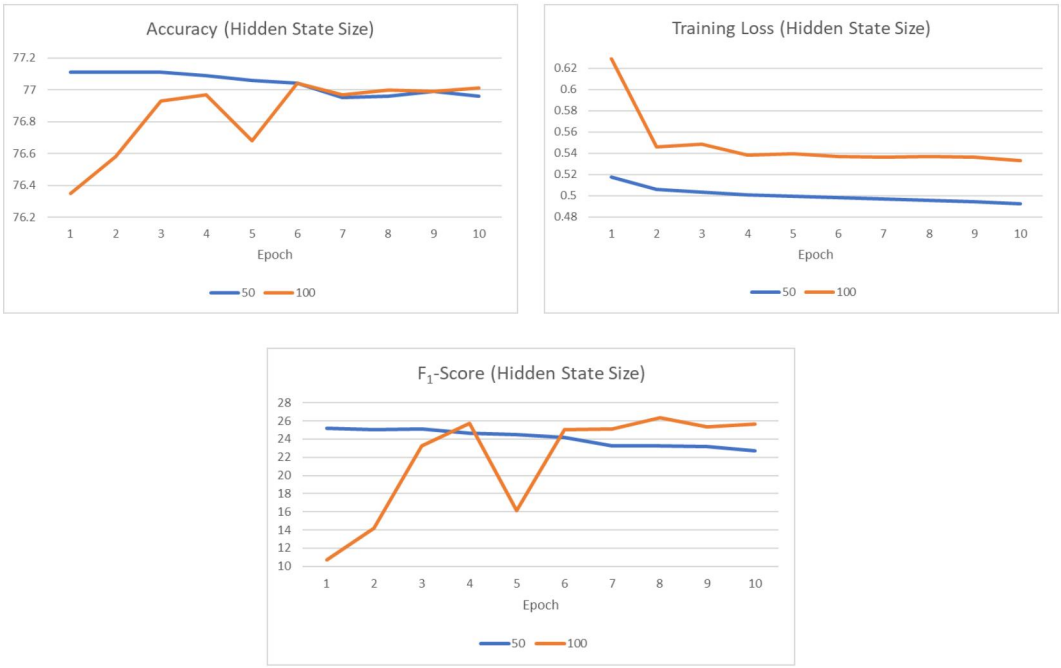


Figure 9: The accuracy, training loss, and F_1-score of models with different hidden state sizes in the ten training epochs. The accuracy and F_1-score were calculated on a validation data set; the training loss was calculated on a training data set. The hidden state size of 100 performed better.

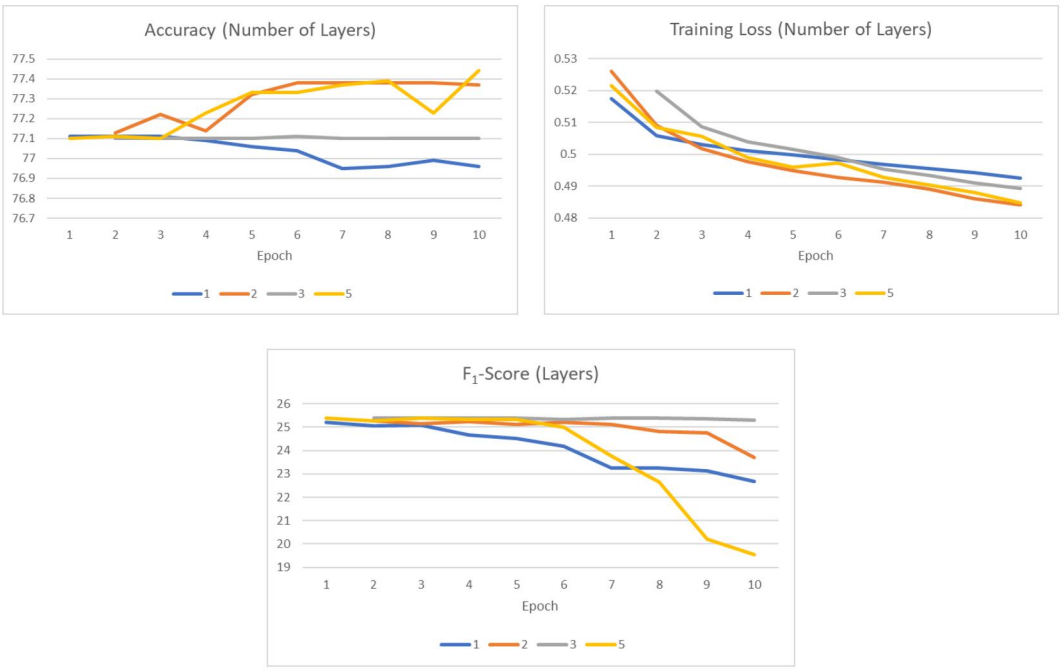


Figure 10: The accuracy, training loss, and F_1-score of models with different numbers of layers in the ten training epochs. The accuracy and F_1-score were calculated on a validation data set; the training loss was calculated on a training data set. The three-layered model performed the best.

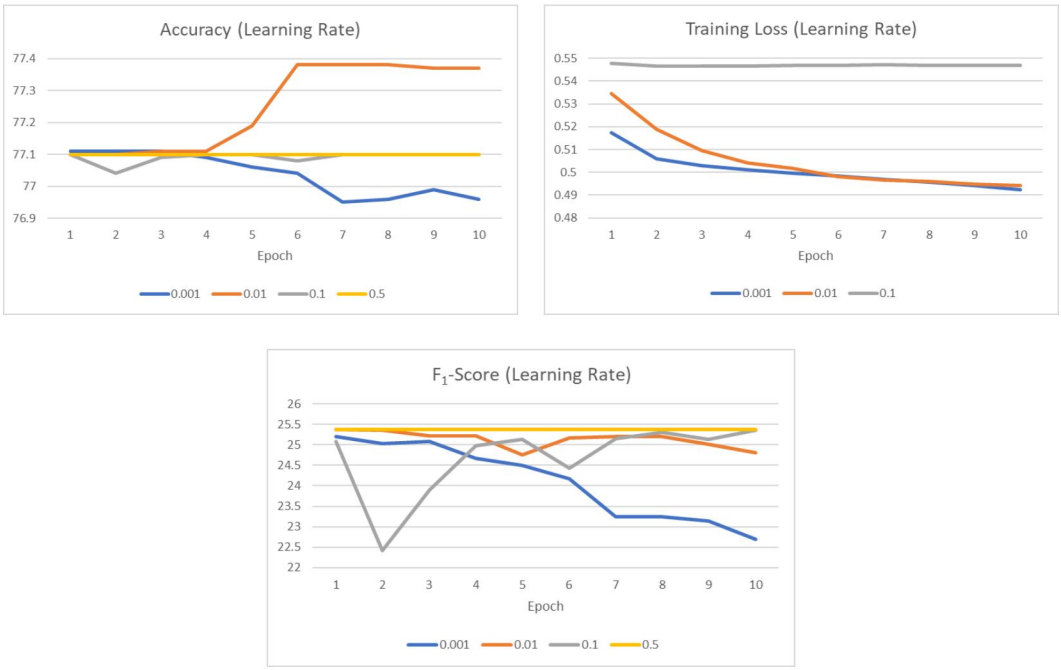


Figure 11: The accuracy, training loss, and F_1-score of models with different learning rates in the ten training epochs. The accuracy and F_1-score were calculated on a validation data set; the training loss was calculated on a training data set. We were unfortunately unable to record the training loss for the learning rate of 0.5, resulting in our choice of 0.1 as the learning rate.

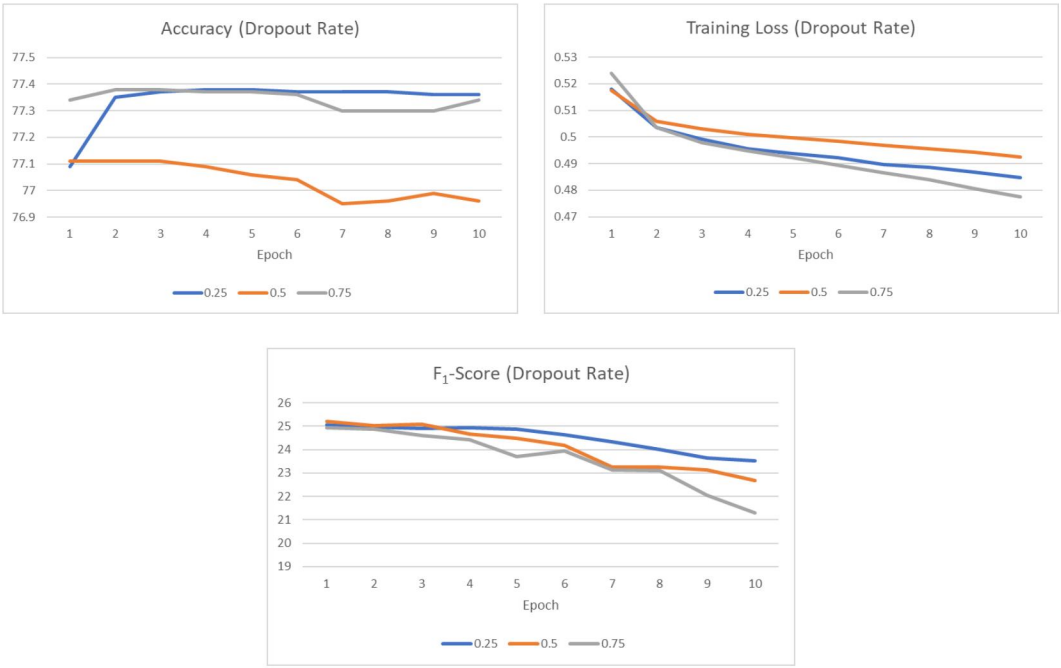


Figure 12: The accuracy, training loss, and F_1-score of models with different dropout rates in the ten training epochs. The accuracy and F_1-score were calculated on a validation data set; the training loss was calculated on a training data set. We chose a dropout rate of 0.5, because of the similar performances of the three dropout rates.