Winning Ticket to the Movies: An Analysis of Text Classification With Sparse Networks

Samuel Munro
munrosl@rose-hulman.edu

Cade Parkhurst
parkhuca@rose-hulman.edu

Abstract

Sparse networks are defined as neural networks that have been "pruned" down. Pruning a model is when a percentage of connections and neurons get completely removed from the network. The end result is a fully trained network that has almost identical network performance to a dense network (one that has not been pruned). The benefit of removing connections and neurons from a network is increased memory and computational efficiency.

We challenge the efficiency versus accuracy in sparse networks when classifying movie Motion Picture Association of America (MPAA) ratings based on their synopses. Many applications of sparse networks deal with image classification and focus heavily on traditional, fully connected networks. We expand the topic to include a natural language application using techniques common to the field. This includes varying neural architectures that are designed for recognizing sequences, word embeddings, and tokenization. We believe pruning will be especially useful and needed in these networks. For example, a recurrent neural network (RNN) and its "unfolded" counterpart has many layers and neurons. Using network pruning as a supplement to traditional neural network training methods such as backpropagation and Backpropagation Through Time (a common technique for training RNNs), we can increase the efficiency of these networks while not significantly decreasing performance.

We will show that the use of sparse networks in natural language applications is not only feasible, but advantageous when large networks are involved, saving memory and computation energy. Most of the studied architectures perform just as well at 10% of their original size when compared to their dense counterparts.

Introduction

When looking at efficiency versus accuracy in the pruning process of neural networks, there are many factors to consider. This is a topic that has been covered in multiple prior works, most notably: "The Lottery Ticket Hypothesis: Finding Sparse Trainable Neural Networks" by Jonathan Frankle and Michael Carbin[1]. They describe a "winning ticket" subnetwork: a smaller network within a neural network that contains fewer parameters. They state that "dense, randomly-initialized, feed-forward networks" contain these winning tickets. We intend to use this method for finding winning tickets in the field of natural language processing with various neural network architectures including: recurrent neural networks (RNN), long short term memory networks (LSTM), gated recurrent units (GRU), convolutional neural networks (CNN), and fully connected neural network. Because pruning usually occurs during the training process, where after an amount of cycles the network "prunes" and removes a set percentage of connections based on their magnitude, we will train these networks to varying densities, from 100% full to 99% empty. We classify the MPAA film ratings of various movies based on a concise plot synopsis. This type of classification task is very similar to the task in "Prediction of a Movie's Success From Plot Summaries Using Deep Learning Models" by You Jin Kim among others [2], as well as in "Predicting the Genre and rating of a Movie based on its Synopsis" by Varshit Battu and others [3]. Both papers describe models that use the plot of a movie to classify different aspects of the movie, whether it be success or genre. Using what we have learned from these papers we

classify the MPAA rating of a film based on a movie's plot synopsis using sparse networks to further investigate the effects of pruning in text classification.

Model Creation

We show that models of various architectures can be pruned while maintaining their accuracy when classifying sentences. We studied six models and achieved various levels of success. We constructed our models using Google's open source deep learning Tensorflow library. Tensorflow allows for the quick prototyping and training of models, and its built-in tokenizer allows for defining datasets with integers that correspond to each individual word. This allows us to pass our plot summaries into the embedding layer that is included in all of the architectures used in this paper. The embedding layer learns the words and converts them into high dimensional vectors for the model to use. Figure 1, below, displays an example of the eight-dimensional vectors used in our fully connected model. After training, words with similar meanings should also have similar dimensional features such as magnitude and direction.

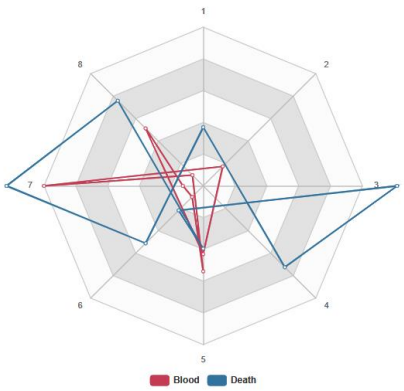

Figure 1: A radar plot of the embedding dimensions of two related words: "blood" and "death".

In the following section we will briefly outline each of our models, describing layer sizes and functions.

Fully Connected

The classic fully connected neural network is a strong tool for many different tasks, sentence classification being one of them. Our fully connected model contains the following layers:

1. Embedding layer
2. Flatten
3. Dense (320)
4. Dense (64)
5. Dense (4) - Softmax

This model had two hundred-thousand trainable parameters. We had to include a flattening layer in between the embedding layer and dense network in order to convert the eight-dimensional vectors into a shape that can be understood by the fully connected dense network.

1-D Convolutional

Convolutional neural networks excel at image classification tasks. CNNs use a multitude of techniques such as pooling and filters in order to scan data for patterns. We attempted to use this idea to scan sections of each input sentence. Our model implementing a convolutional layer consisted of the following layers:

1. Embedding
2. 1D-Convolutional (5 filters)
3. Dropout (10%)
4. Dense (10)
5. Dense (4) - Softmax

This network contained almost nine hundred-thousand trainable parameters. We included a dropout layer in this model in order to help prevent overfitting by randomly excluding different connections.

Recurrent Neural Networks

Recurrent neural networks are very similar to feed forward networks, but the output of a layer is also fed back as an input for the next step in time. This allows for sequenced data to be used as input. This gives the network a form of memory, making it very useful for text classification. Tensorflow's "SimpleRNN" layer is a very basic implementation of a recurrent neural network. Our model contained the following layers:

1. Embedding
2. SimpleRNN
3. Dense (64)
4. Dense (4) - Softmax

This network also contained almost nine hundred-thousand trainable parameters. Overall this is a pretty straight forward model through which we are attempting to test the ability of the "SimpleRNN" layer.

Gated Recurrent Unit

The gated recurrent unit is a specialized type of recurrent network that uses extra logic gates in order to analyze the data. This allows it to remember aspects of input data for longer amounts of time, which is a common problem often found with RNNs. Our GRU model is very similar to our RNN model, only

replacing the "SimpleRNN" with a GRU layer. This network also contained almost nine hundred-thousand trainable parameters.

Long Short-Term Memory

Long short-term memory units (LSTMs) are another type of specialized RNN. LSTMs have even more gates and paths for data to pass through when compared to a GRU. The LSTM adds another output that is only used internally in-between time steps. We implemented two different models with the LSTM layers. Our first model was constructed as follows:

1. Embedding
2. LSTM (256, returns sequences)
3. LSTM (64)
4. Dense (64)
5. Dense (4) - Softmax

This first LSTM contained 1.2 million trainable parameters. Our second LSTM model added an LSTM that processed the data in reverse to gain even more information about the passage. This was accomplished by adding another LSTM layer of size two hundred fifty-six output dimensions that returns sequences and goes backwards in between the second and third layer of the previous model. This model contained 1.8 million trainable parameters.

Dataset and Data Pipeline

The main goal of this project was to classify movie MPAA ratings based on the movie's synopsis. The IMDB Top 250 Lists and 5000-plus IMDB records dataset compiled by the data.world (a popular dataset website) user TheMitchWorksPro contains two large CSV files of about five-thousand records each [4]. Of these records, about eight thousand were usable. Some needed to be removed because they had missing ratings, or only had a handful of training examples. For instance, there were only seventeen movies that were rated NC-17 in the dataset, which did not provide enough examples for training.

Then we passed all of the input sequences to a tokenizer, as mentioned earlier. This tokenizer converts a sentence such as [I, love, Machine, Learning] to a sequence of integers [1, 904, 345, 12]. This sequence is data that the embedding layer, which precedes every model, can accept. For our models, we chose to use eight embedding dimensions for the fully connected model, and sixty-four for all of the others. From here, the data is passed into one of the six models described above. We shuffled the dataset, and isolated five hundred records for testing.

One of the main challenges we had with the dataset was data imbalance. Over half of the training examples were R-rated, which biased the classifier towards that rating. To combat this, we set the models to weigh each training example differently based on that label's percent share of the total dataset.

$$weight = \frac{\left(\frac{1}{count_{class}}\right) * count_{total}}{strength}$$

Where *strength* is an arbitrary constant that adjusts the value that the classifier biases each class weight by. This ended up being a small optimization, bringing our test network accuracy from 94.7% to 98.8%, but was nice to have before we began testing the larger and more complex networks.

Finally, once data was input to the network, it was classified as a one-hot vector. For instance, movies that were R-rated received a data label of [1, 0, 0, 0]. This was necessary for the loss function we used: Cross Categorical Cross Entropy, which takes the output of softmax layers and uses it to calculate loss.

Network Pruning

To prune each network (listed above) we used Tensorflow's built-in optimization library, specifically the prune_low_magnitude() method. Each prune-compatible layer was wrapped with this method, and passed a set of pruning parameters:

- initial_sparsity
- final_sparsity
- begin_step
- end_step
- frequency

Our testing only modified the final_sparsity parameter, which is the sparsity that the model is pruned up to between the begin_step and end_step, which we set to the beginning and end of training for simplicity.

Due to our implementation strategy, the networks are pruned *by layer*, not as a whole. This coincides with the strategy used in Frankle and Carbin's paper, in which they prune each of their smaller network's layers at the same rate. They do, however, recommend global pruning for deep networks such as VGG-19 and RESNET-18 [1]. Future exploration of our topic could include testing the benefits and drawbacks of global pruning.

Results and Discussion

During testing, we measured and collected training metadata for each of the networks in their dense states and at eleven different sparsity levels: 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 99 percent. We hoped to compare how quickly each network converged when trained with the same dataset. Each network was then evaluated with an isolated test dataset, and the final accuracy of the network after 150 epochs of 20 examples each was plotted against the network's sparsity level. This totaled up to 72 total networks evaluated. To prevent the page from getting too filled up, we'll only display some highlighted results, and a link to the full set of results will be provided further down.

Highlighted Fully Connected Results

This model was one of the most consistent that we tested. As listed below, the 80% sparse network trains just as quickly as the dense network, but tapers off at 90% sparsity (see full results) and especially when pruned to 99% sparsity. This model far outperformed our expectations, because it has no way to understand context or the way words relate to each other, this model was expected to struggle with remembering important information.

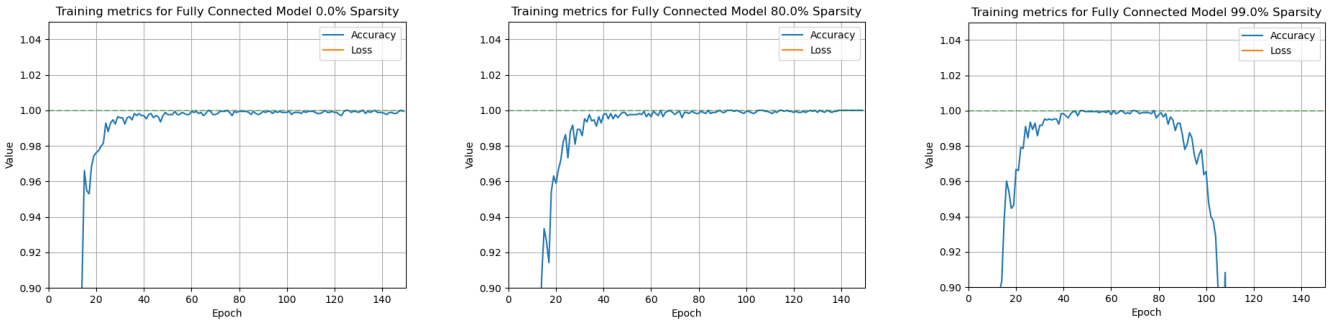

Figure 2: Accuracy of a Fully Connected Model at varying sparsity levels. There is not much distinction between 0% and 80% pruned, but at 99% pruned the accuracy falls off.

Highlighted 1-D Convolutional Results

The model was unable to train to over 50% accuracy, regardless of the sparsity. If the input data had been formatted differently, we believe that this network could have performed better. For this specific application, however, a one-dimensional CNN is not viable.

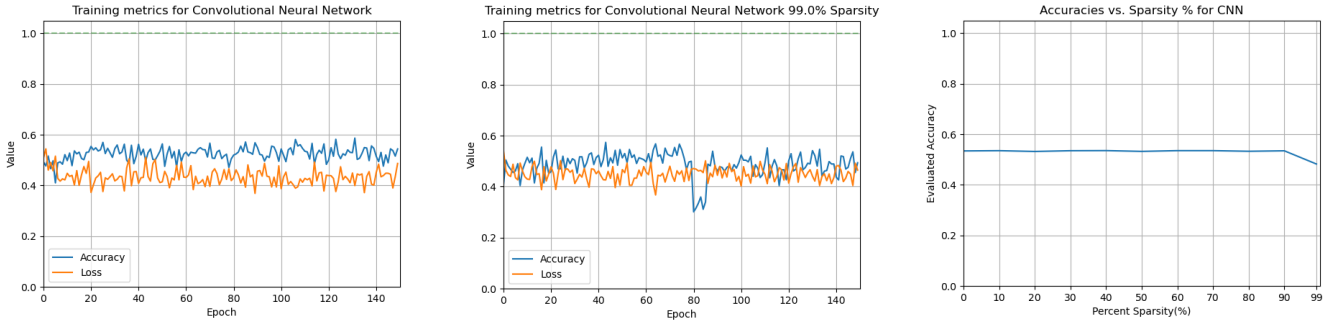

Figure 3: Accuracy of a one-Dimensional CNN-based neural network at varying sparsity levels. This never achieved good results.

Highlighted Recurrent Neural Network Results

Again, this network was extremely consistent. It was able to train up to >99% accuracy in a similar number of epochs as LSTM v2 (see below), but with much fewer parameters. Even at 90% sparsity, it still performed just as well as the dense model. The two were so close that a change in scale on the y-axis of the plots was needed to represent the differences between the sparsity levels.

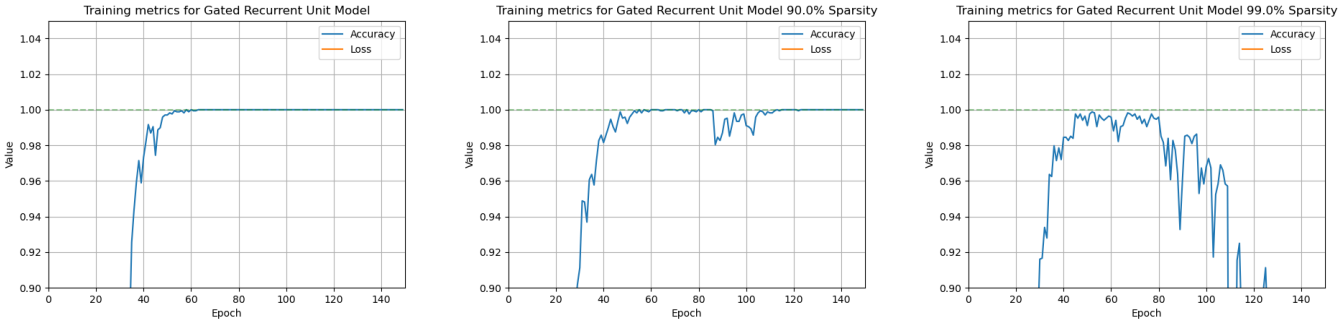

Figure 4: Accuracy of a Simple RNN at varying sparsity levels. As the leftmost and middle figures suggest, a Simple RNN can be pruned to very low levels and still maintain exceptional accuracy.

Highlighted Gated Recurrent Unit Results

This model was also extremely successful. The dense model trained up to >99% accuracy within only 40 epochs. The sparse models slowed down a bit, but this is still an excellent model for pruning. Something interesting that we observed was that the first 20 epochs were inconsistent; this occurred at every sparsity level.

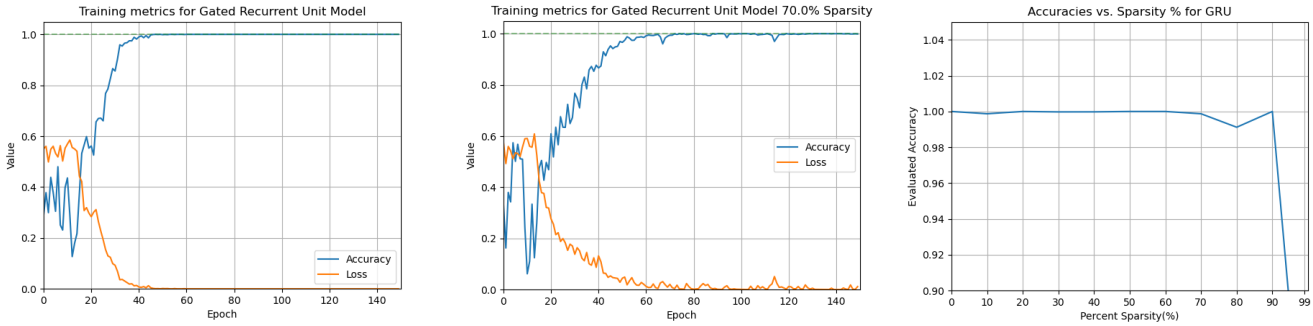

Figure 5: Accuracy of a GRU-based model at varying sparsity levels. This model demonstrates the increased training time caused by pruning between the 0% and 70% sparsity levels, where the sparser model needs about 20 more epochs to be as accurate as its dense counterpart

Highlighted LSTM v1 Results

This model was successful at classifying movies but was inconsistent and volatile at seemingly random sparsity levels. It would occasionally not converge, even at low sparsity levels. When it did converge, however, it trained quicker than the Gated Recurrent Unit, even at higher sparsity levels.

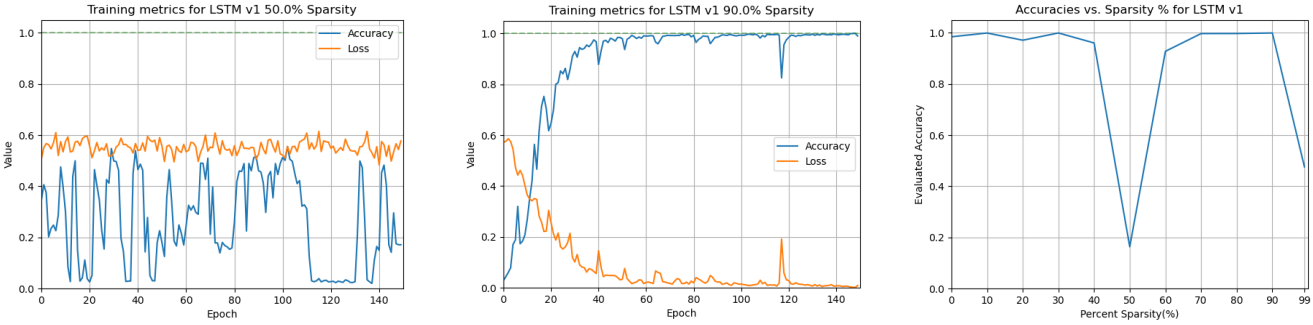

Figure 6: Accuracy of LSTM v1 (details above) at varying sparsity levels. The leftmost plot shows the volatility of the model. Once converged, however, this model performs well at high sparsity levels, as demonstrated by the middle plot.

Highlighted LSTM v2 Results

This model was very consistent overall. This is most likely due to the addition of a backwards LSTM that can better process the input sentences and how each word relates to the rest of the sentence. It performed much better than most of the other tested models, even at 99% sparsity. We believe that this is due to the large number of parameters that the model started off with.

Figure 7: Accuracy of LSTM v2 (details above) at varying sparsity levels. This model performed the best, most likely due to its high number of parameters.

Conclusion

When selecting the correct model architecture and structure, applying pruning techniques to text classification networks is an excellent strategy for optimizing memory usage and computation time. We have shown that sparse networks can match the performance of their dense counterparts when performing this classification task. Some can even be pruned up to 90% with no significant performance penalty, such as the LSTM v2 and the Simple RNN. While this could be explained by the fact that some of these networks started off with a high amount of parameters, we believe that our results more closely support a conjecture made in “The Lottery Ticket Hypothesis” where the authors conclude that an overparameterized network is easier to train because they have more combinations of subnetworks that have the potential to be winning tickets [1]. Our results support this conjecture; networks with high numbers of parameters consistently trained to >99% accuracy at almost every sparsity level. There is a limit, of course, to how much a network can be pruned before it loses its high dimensional understanding of a topic. We attempted to demonstrate this using the 99% sparsity tests. Our data suggests that the limit for each network is different and depends on its architecture and structure.

Future Work

There are a few extensions to our work that could be made, given more time. Network pruning is an expansive topic, and this project only scratched the surface of the many avenues that can be explored. Listed are a few potential topics that would be great extensions of our work and would answer detailed questions related to text classification and network pruning.

Pruning Parameters

In this project, we only explored the effects of changing the final_sparsity parameter. There are four more parameters in this specific implementation of network pruning alone, opening the door to experiment with combinations of these parameters and how they affect the effectiveness of pruned networks. Adjusting these parameters could reveal better pruning strategies that start pruning later in the training process or pruning a larger percentage of weights less often, for instance.

Dropout Layers

Dropout layers function by temporarily disabling a portion of the total network every training cycle. These layers have been commonly shown to help prevent overfitting because they do not allow an entire network to learn the same task at once. Of course, this definition can be rephrased to state that dropout layers survey a random subnetwork every training cycle, which closely relates to network pruning. Studying this relation would be an excellent extension of our work.

Reinitialization Testing

Conclusions made in the "Lottery Ticket Hypothesis" are largely based on results gathered from reinitialization testing, a technique in which a sparse network (winning ticket) is randomly reinitialized to random values and retrained. These new networks often show inferior performance when compared to the winning ticket. This supports the hypothesis because it shows that the initial random initialization of a winning ticket benefits its performance [1]. We believe that running similar reinitialization experiments with our project setup could provide interesting results and potentially further support the lottery ticket hypothesis. Further testing of winning tickets specifically has already been conducted in another scope in a recent paper from 2019 [5].

Appendix

Source code: https://github.com/HaveANiceDay33/NLPTesting/blob/master/MPAARatings.py

Full result sets:

1. Zoomed Graphs: https://github.com/HaveANiceDay33/NLPTesting/tree/master/GraphsZoomed
2. Not Zoomed Graphs: https://github.com/HaveANiceDay33/NLPTesting/tree/master/GraphsNotZoomed

Dataset: https://github.com/HaveANiceDay33/NLPTesting/tree/master/Data

References

[1] J Frankle and Carbin, 2019, “The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks”, In International Conference on Learning Representations (ICLR), volume abs/1803.03635.

[2] Y Kim, J Lee, Y Cheong, 2019, “Prediction of a Movie’s Success From Plot Summaries Using Deep Learning Models”, In *Proceedings of the Second Storytelling Workshop,* W19-3414.

[3] V Battu, V Batchu, R R Reddy, M K Reddy, and R Mamidi, 2018, “Predicting the Genre and Rating of a Movie Based on its Synopsis”, In 32nd Pacific Asia Conference on Language, Information and Computation, Y18-100.

[4] TheMitchWorksPro, 2017, “IMDB Top 250 Lists and 5000 plus IMDB records”, https://data.world/studentoflife/imdb-top-250-lists-and-5000-or-so-data-records.

[5] H Zhou, J Lan, R Liu, and J Yosinski, 2019, “Deconstructing Lottery Ticket: Zeros, Signs, and the Supermask”, In 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), volume abs/1905.01067.