Ethan Leicht

Professor Wenpeng Yin

Machine Learning and AI - Section 1

2 Dec 2023

Newswire Classification with Recurrent Neural Network and Transformer

Introduction

I worked individually on this project. In this paper, I will describe how I implemented two deep learning systems for classifying newsires by topic, my results, and the lessons I learned during the process.

Task & dataset & preprocessing

The dataset I chose to use is the Reuters newswire classification dataset, which is available for download from the Keras library. A newswire is an electronically transmitted message providing up-to-the-minute news stories, financial market updates, and other information. The dataset consists of 11,228 newswires from Reuters, a news agency. The newswires are labeled over 46 topics. The task I set out to accomplish was classifying the newswires with their respective topic labels.

Each newswire is encoded as a list of word indexes, appearing in the order of the words in the newswire. These words are indexed by overall frequency in the dataset. For example, the integer “3” encodes the 3rd most frequent word in the data. As a convention, “0” encodes any unknown word.

Preprocessing the data was simple. I loaded the training and test data from Keras and set the vocabulary size to 1,000, meaning only the 1,000 most common words were kept in the dataset. I set the number of words to be considered from each newswire to 200. Training examples longer than 200 words were padded with zeros, and training examples shorter than 200 words were truncated. Finally, I sent the training labels through Keras’ to_categorical function to convert the class integers into a binary class matrix.

The implementation & architectures of two deep learning systems

For my two deep learning systems, I chose the recurrent neural network and the transformer.

I built the recurrent neural network, more specifically Long Short-Term Memory (LSTM), using the Keras library. LSTM is a neural network architecture which is designed to resolve the vanishing gradient problem faced by recurrent neural networks. Each LSTM unit consists of three gates: the forgetting gate, the input gate, and the output gate. LSTM is useful for natural language processing tasks with sequential data.

The first layer of my network is an embedding layer which converts the positive indexes of each training example into dense vectors of fixed size. I set the embedding size to 300. Next, the data passes through a layer of 120 LSTM units. Finally, the data passes through a dense layer of 46 units, one for each newswire topic. The activation function at this layer is softmax.

I built the transformer neural network using the Keras library. The transformer is an encoder-decoder model for machine translation in which the decoder uses an attention mechanism to extract the most relevant parts of a data sequence. My transformer uses Multi-Head Attention, which passes a sequence through an attention mechanism several times in parallel then concatenates and transforms the outputs into the expected dimension.

The first layer of my transformer is Keras' TokenAndPositionEmbedding layer, which sums the token and position embedding for each word in the input sequence. The data then passes through a transformer block consisting of one Multi-Head Attention layer with 4 heads, a dropout layer, a normalization layer, a 50-unit dense layer with ReLU activation function, another dropout layer, and a final normalization layer. After the transformer block, there is a 1-D global average pooling layer, which reduces the throughput of the network by averaging the sequential data. Finally, data passes through a dropout layer, a 50-unit dense layer with ReLU activation function, another dropout layer, and a final dense layer with 46-units for each class and a softmax activation function. All dropout layers use a dropout rate of 0.1 and were included to prevent overfitting.

The training details of two deep learning systems

Both systems were trained for 15 epochs with a batch size of 64. Each epoch included 140 training examples. One batch was set aside as a validation dataset to be evaluated during the training process. To decrease training time, both models were trained in a Tensorflow GPU environment using the NVIDIA GeForce RTX 3050 in my personal laptop.

For the LSTM network, each epoch took an average of 3 seconds. For the transformer, each epoch took an average of 10 seconds.

The results & observations & conclusions

Using the hyperparameters detailed above, I was able to achieve a testing accuracy of 76.0% with the LSTM network, meaning 76.0% of the newswires in the test data set were correctly classified by their topic. Using the hyperparameters detailed above, I was able to achieve a testing accuracy of 77.8% with the transformer.

In general, I had to tune the LSTM network's hyperparameters more than the transformer's to get satisfactory results. I discuss hyperparameter tuning in greater detail in the following section. In general, the Multi-Head Attention transformer seems to perform better than the LSTM network on this classification problem.

The challenges & obstacles you met and your solutions

The main challenge I met with both systems was finding the optimal set of hyperparameters for completing this task, specifically the network size, embedding size, vocabulary size, and number of training epochs.

With the LSTM network, the main lesson learned was that simplifying the model and data often yielded better results. Increasing data complexity did not yield significant improvement. For example, with a vocabulary size of 10,000 and a training example length of 500, testing accuracy was lower at 69.3% than it was with a vocabulary size of 1,000 and a training example length of 200, which yielded 76.0% testing accuracy. Likewise, increasing model complexity did not yield significant improvement, and sometimes made the model worse. For example, a network with 120 LSTM units yielded testing accuracy of 76.0%; increasing the number of LSTM units to 200 LSTM depreciated this metric to 75.6%. Including additional LSTM and dropout layers also did not improve performance.

With the transformer network, there were more hyperparameters to consider, as it is a more complex system. Adding heads to the Multi-Head Attention layer seemed to slightly improve performance until it made the model so computationally intensive that my device could not execute the training. Once again, there were diminishing returns for increasing model complexity. For example, with all other parameters held constant, a model with only 2 heads yielded 76.3% testing accuracy while 5 heads yielded 76.9%. As with the LSTM hyperparameter tuning, decreasing layer size, specifically of the dense layers, yielded better accuracy. I experimented with layers of 200 units and decreased this until landing around 50 units.

With both networks, I noticed validation accuracy plateaued after about eight training epochs. It quickly became clear that training for more than fifteen epochs was a waste of time and resources. Because each of these models are relatively complex, they are able to produce quality results with just a handful of training batches.

As a final note, I would like to acknowledge that I was essentially shooting in the dark to find appropriate hyperparameter values for this problem. It is quite possible that my architecture could have performed 10% better if I had a better understanding of how these values are intelligently determined in natural language processing applications.