Pennsylvania State University

Final Project Report

Aziza Kamanzi, Kofwana Lawson, Ailish Quinones, Noah Schiro

CMPSC 448: Machine Learning and Algorithmic AI

Professor Wenpeng Yin

December 2, 2023

Introduction

For this final project, we were asked to create a classification model using two out of the three Deep Learning (DL) systems options, which were Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformer-based systems. We decided to build a sentiment analysis system using RNNs and transformers. Sentiment analysis, also known as opinion mining, is an approach to natural language processing that identifies the emotional meaning behind a body of text (Barney, 2023). Sentiment analysis is highly beneficial for companies, offering insights into customer opinions regarding their products, facilitating market research, understanding political opinions and social issues, and aiding potential fraud activities. With the exponential growth of social media, we consider sentiment analysis as an essential tool for guiding decision making processes within large corporations. Sentiment analysis is a great skill to develop, hence our motivation in learning to apply different techniques learned in class to refine our skills in this specific topic.

From the Deep Learning systems we learned, we decided that RNNs and Transformer-based systems were the best to implement our project. As learned in class, RNNs are extremely useful for processing sequential data, making them suitable for sentiment analysis on a sequence of text. Concurrently, we chose a Transformer system due to their capability to capture contextual information. Given that Convolutional Neural Networks are traditionally associated with image processing, we opted for the other two DL systems. Since our data was textual information, we were interested in comparing RNN and Transformer based systems to discern their strengths and weaknesses in the context of sentiment analysis.

Dataset and Preprocessing

Dataset

All of our data came from Kaggle, where we found a data set with 1.6 million tweets ready to use for our project. The decision to utilize Twitter tweets for sentiment analysis in the development of Transformer and RNN models is grounded in the dataset's diverse sentiment labels, encompassing negative, neutral, and positive sentiments denoted by values 0, 2, and 4 in the "target" column (though the dataset only contains targets of 0 and 4). This diversity enables the models to discern various sentiment intensities. Additionally, the dataset comprises real-world, user-generated data, offering an authentic representation of language use and sentiment expression.

The inherent conciseness of Twitter tweets, owing to the platform's character limit, poses a challenge for models to handle short and varied text lengths effectively. Timestamp information in the "date" column allows for exploring potential temporal trends in sentiment, while details such as the query information in the "flag" column and user information in the "user" column provide context for sentiment analysis within specific topics or user patterns. The textual content in the "text" column, reflecting the colloquial nature of Twitter language, serves as the primary input for sentiment analysis models. Overall, the Twitter dataset chosen provides a rich, realistic, and challenging source of data for training and evaluating sentiment analysis models, assessing their performance across varied sentiments and linguistic nuances.

Preprocessing for the RNN model

Our preprocessing for the RNN model started with collecting and encoding the data. Once we found our dataset, we first used the pandas library to read the CSV file. After the CSV was read by Pandas, target labels were encoded using LabelEncoder, which we used to normalize the labels. After reading the data, we went on to use Pytorch and torch libraries to continue processing the text data. It’s important to note that not every aspect of this data set was used for our project. Our data was modified and adjusted using Pytorch libraries to fit the needs of our project.

Pytorch has many sister libraries that accompany its main library, torch. For this particular project, we were interested in the torchtext library which provides useful tools for text processing. The steps taken using torchtext to process our data were the following:

1. We used the tokenizer for basic English to convert a string into a list of strings, each of which consisted of one token.

2. We created a vocabulary from our dataset, which maps each possible token to a unique number.
3. We added a special token and vocabulary number for unknown tokens.

The special tokens were necessary for our project given the fact that our data included names, URLs, and other basic words that are not present in basic English but are very prominent in the Twitter platform.

Preprocessing for the Transformer model

In the case of the transformer, we had to preprocess the text in a somewhat specialized way. We began by splitting the dataset into a training and validation set. Next, we converted our data object to a "PyTorch Dataset" type of a custom variety. This particular dataset implementation, when asked for an individual datapoint, will:

1. Tokenize the text.
2. Pad the text until a certain context length is met, or truncate the text until that context length is met. In this case we chose 50 tokens as our context length since the average tweet length (in tokens) was 16.5 and the maximum tweet token length was 229.
3. Convert the list of tokens to a list of integers with our vocabulary.
4. Convert this list to a Pytorch tensor.
5. Divide the label by 4. The original dataset has negative sentiment as 0 and positive as 4. Dividing by 4 maps these labels to 0 and 1 respectively.
6. Return a tuple of the label and our tensor representing our text.

These steps are handled in the PyTorch dataset object and are processed each time we prepare a batch of data to pass along to the model for inference.

Implementation and Architectures

RNN Implementation and Architecture

For our sentiment analysis implementation, we constructed it with a Recurrent Neural Network. Our RNN features essential components such as an embedding layer, an RNN layer, and a linear layer. The RNN model is designed to handle packed sequences, enabling the efficient processing of input data with varying lengths. The architecture is as follows:

1. An embedding layer to convert token indices to dense vector representations. This layer helps the model to learn contextual information by capturing relationships between words.
2. A one layer RNN to process the embedded sequences. This layer was used to process sequential data which was particularly useful when dealing with different sentence lengths.
3. A fully connected layer for classification using the output from the last time step. This layer is the final layer that produces the output for classification.

Transformer Implementation and Architecture

The transformer model utilizes a very simple classification implementation. The architecture is as follows:

1. An embedding layer, which converts the vocabulary integer into a 264 dimensional tensor. With our vocabulary size, we felt that 256 dimensions was sufficient to capture the complexity of the dataset.
2. A positional encoding, which follows the standard implementation seen in transformer models.
3. A transformer module, which comes from PyTorch's library. For this, we are using 12 attention heads, only 1 encoding and decoding layer, a feedforward dimensionality of 512, and a dropout of 0.1.
4. The output of the transformer module is then passed through global average pooling.
5. Finally, we pass through a multi-layer perceptron classification network. This takes the output of the pooling and casts it to a 64 dimensional hidden layer, performs element-wise relu, and then classified into two output nodes which have softmax applied to them. Each of these output nodes can be thought of as the network's prediction for the data with one of the nodes representing the positive class and one node corresponding to the negative class.

This structure is certainly not novel and is completed based on the “Attention is all you need” paper (Vaswani et al., 2017).

Model Training

RNN training

The training process of the RNN model was possible through the usage of the cross-entropy loss function and the Adam optimizer. Before using these models, we split the data into training and validations sets, agreeing on a split ratio of 0.8. Using these two optimizer models ensured that the parameter’s in the RNN model were being processed effectively during training iterations. The training loop operated by drawing on the efficiency of a DataLoader to handle the batched data. The input sequences from the one layer RNN are fed into the model and the sentiment labels are computed using cross-entropy loss. After that, we used the Adam optimizer to adjust the weights and biases of the model. The usage of these two techniques enhanced the ability to predict sentiment analysis across different inputs from our dataset.

Transformer training

The transformer was trained in a typical train test loop for each epoch. We decided on these basic hyperparameters for the model:

1. Batch Size = 64.
2. Learning rate = 10^{-3}.
3. Epochs = 1.
4. Optimizer = Adam.
5. Loss function = Cross entropy Loss.

As a special note about the training procedure, many measures were taken for the sake of optimization. Most importantly, hardware acceleration was used, specifically a NVIDIA 3060 GPU was employed for training this model. All of the typical optimizations that come with GPU training were also used including data pinning and increasing the number of workers in the data loader object. Next, we observed that the 3060 GPU utilizes an Ampere architecture, which has tensor cores. These cores can see a 2-3x speedup if we choose to use automatic mixed precision (AMP) optimization. As such, we modified the training script to include this and indeed see a reduction in training time and memory usage. In theory, one could utilize this increase in free memory to increase the batch size. This would lead to possibly even quicker

convergence with the model, however we left this as an exercise for the future, as our model was already converging quite quickly.

Results, Observations, and Conclusions

The transformer achieved an accuracy of 80.7% on our test set. We were initially dismayed to see this, but this led to us finding something quite interesting about the dataset. We found that many tweets which have neutral or ambiguous sentiment were labeled as positive or negative, as these are the only two options for labels. Upon further investigation, we found that the author of this dataset was only able to achieve 79.1% accuracy on their own dataset (this was performed with an LSTM model in the keras library). As a result, we concluded that the quality of the data was hurting the potential for the model to achieve a near-perfect score and that even a human would not be able to perform much better than our model. We now find 80.7% to be an excellent score given the issues with the dataset.

The keen eye may have noticed that we only have one epoch of training for the transformer model. We noticed that our model was overfitting for any number of epochs greater than 1. Even with one epoch, we observed that the model converges to a stable loss and accuracy after seeing just 20% of the dataset. We have two explanations for why this might be the case. First, the dataset is very large with 1.6 million examples. It's quite possible that with this scale of data, classifying something as simple as positive or negative sentiment is not very difficult. The second explanation is that the transformer is much too powerful for this type of problem. The model has enough complexity that even small adjustments to all of the weights in the model can lead to hasty convergence. It's possible that these two hypotheses interact with each other and the data is simple while the model is complex, leading to the model "solving" the problem very quickly.

The RNN model achieved an accuracy of 76.6% on our test set, which also initially elicited a sense of disappointment. However, after reflecting on the previously discussed challenges with binary sentiment classification, we now recognize how remarkable this performance is. Acknowledging these constraints motivates us to consider further adjustments to the dataset labeling process or to explore different labeling techniques that might better reflect the nuanced nature of social media sentiment. These findings highlight the importance of continuously refining datasets in natural language processing tasks, as well as contextualizing results within the broader understanding of the potential dataset limitations.

Challenges and Obstacles

With the transformer, vanishing gradients plagued the project for a very long time. We had a lot of trouble identifying what was wrong with our training script. We also knew that this wasn't a problem with the model architecture itself, because transformers have many measures in place to reduce vanishing gradients, even when they are very deep. These measures include residual connections, relu activations, and layer normalization. We learned after much trial and error that the model would only learn and decrease its loss when the number of encoder/decoder layers was 1. Anything more, and the model would run into vanishing gradients. We suspect again that this is because our data is too simple and the model is already sufficiently complex with just one layer. This does indeed happen to be the case since we get such a high accuracy after just 20% of one epoch. We found that making a model much more complex than the problem set it was trying to solve can also lead to vanishing gradients. Our solution to this was to simply leave the model at one encoder/decoder layer.

With the RNN model, we also ran into vanishing and exploding gradients. This problem is inherent in RNNs, particularly when dealing with long sequences of text. As the model backpropagates through the input sequence, gradients can become extremely small or large, hindering the learning process. To address this, techniques such as gradient clipping or more advanced architectures like LSTM (Long Short-Term Memory) cells are often employed. Another challenge we encountered was handling sequential dependencies and capturing long-range dependencies in the data. Traditional RNNs struggle with retaining information from earlier time steps, leading to difficulties in understanding context and relationships within the text. Additionally, optimizing hyperparameters, such as the learning rate, batch size, and network architecture, is crucial for achieving optimal performance. Balancing model complexity with computational efficiency is yet another obstacle, especially when working with large datasets. Despite these challenges, successfully developing an RNN model involves a combination of architectural choices, careful parameter tuning, and addressing numerical stability issues to create a robust and accurate text processing solution.

Overall, while the development of the RNN model involves addressing inherent challenges, its potential to understand and analyze sequential data, such as textual information, is substantial. The utilization of techniques like embedding layers and LSTM cells helps mitigate issues like vanishing gradients, allowing the model to capture meaningful patterns in the data. However, it is essential to acknowledge the trade-offs between model complexity and computational efficiency, as well as the need for extensive hyperparameter tuning. Furthermore, the success of the RNN model is contingent on the specific

characteristics of the dataset and the nature of the task at hand. Despite these challenges, the versatility of RNNs in handling sequential data positions them as valuable tools for tasks like sentiment analysis, language modeling, and more. Continuous research and advancements in the field of recurrent neural networks contribute to overcoming limitations and enhancing their capabilities, fostering ongoing improvements in text processing and understanding.

Works Cited

Barney, N. (2023, March 10). *What is sentiment analysis (opinion mining)?*. Business Analytics. https://www.techtarget.com/searchbusinessanalytics/definition/opinion-mining-sentiment-mining.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017, June 12). *Attention Is All You Need.* ArXiv.org. https://arxiv.org/abs/1706.03762