CSA 250: Deep Learning Project II Report

Modukuri Sumanth Kumar

M.Tech AI sumanthkumar@iisc.ac.in S/RNo.: 17025

Tools Used:

- Python 3
- Tensorflow 2.x
- Keras
- Nltk
- Sklearn
- Pickle
- Glove
- Pytorch
- Transformers

Model Comparison:

- What made you choose your current architecture? and How many different architectures have you considered?

  Tf-Idf: The Tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Since it is not using the properties of the sentences and only considering the frequency of words, it is not a good option for Natural Language Inference.

Using this architecture I got a test accuracy of **64 %**.

RNN: Since the data to be processed is temporally related it makes a lot of sense to use RNNs to solve the problem. As it stores the information for current feature as well neighboring features for prediction. A RNN maintains a memory based on history information, which enables the model to predict the current output conditioned on long distance features. I have tried a few different variations of trivial RNNs and optimized them by tuning their hyperparameters using random search given the limited resources, I was able to get a maximum test accuracy of **70 %**. Since it faces the vanishing gradient problem, it is a little unstable.

GRU: Given that it is a variant of RNN, it would be suitable for this task. It also has property to store more information, which would help in classifying the sentences properly. It also does not face the vanishing gradient problem so is also stable enough. To solve the vanishing gradient problem of a standard RNN, GRU uses update gate and reset gate. These are two vectors which decide what information should be passed to the output. It can be trained to keep information from long ago, without washing it through time or remove information which is irrelevant to the prediction. Hence it performs decently for this problem. I have tried a few different variations and optimized them using random search given the limited resources, I was able to get a maximum test accuracy of **74.2 %**.

LSTM: It is another of RNNs variation which works well for any sequential processing task in which we suspect that a hierarchical decomposition may exist, but do not know in advance what this decomposition is. Since Sentences have some form of hierarchical structure due to the grammar, we get better performances using this. Here we can upgrade the performance by using Bi-directional LSTMs. The basic idea of bidirectional recurrent neural nets is to present each training sequence forwards and backwards to two separate recurrent nets, both of which are connected to the same output layer. Since it processes data from both directions, it gets future conexts as well, & hence performs better. After hyperparameter tuning using random search it gave around **78 %**. I could have increased the performance by interoducing an ATTENTION layer but did not get the time to do so.

BERT: BERT (Bidirectional Encoder Representations from Transformers) is a recent paper published by researchers at Google AI Language. It has caused a stir in the Machine Learning community by presenting state-of-the-art results in a wide variety of NLP tasks. BERT's key technical innovation is applying the bidirectional training of Transformer, a popular attention model, to language modelling. This is in contrast to previous efforts which looked at a text sequence either from left to right or combined left-to-right and right-to-left training (like the above discussed models). Since it is a pre-trained model, it has almost all the features needed to solve the problem. I used the Classifier variation of BERT which takes 2 sentences as input and provides a class as an output. So I combined the two sentences and trained the model & got the best performing model. After tuning the hyper-parameters I managed to get test accuracy of **90.5 %**.

- How did you arrive at your choice of hyperparameters - number of neurons/layers, activation function, learning rate etc.?

    - Number of layers:

      Result: **6-Layer Models** (for RNN, GRU & LSTM)
      I had studied about others who attempted using these models and found that this architecture gave the best results & increasing it further was not improving the performance.

- How did you validate your model?

  I used 20% validation set when training the DNN models (except BERT). Since we had a lot of data, it was good to allocate 20%.

<u>Key Points:</u>

- When using the Tf-Idf, I had to store the Tf-Idf vectorizer after fitting on the corpus data during training as we had to use the same during testing.
- When training the DNN models (except BERT) I had to use the same embedding matrix that I used while training during the testing phase. One can think of it as if I am also learning the Embedding Matrix during the training phase along with the model.
- GloVe embedding helped a lot by providing a similarity index among words. It improved the performance by quite a bit.
- The words which are new in the testing dataset are assigned the same embedding (unknown) as it was not trained on that data.
- Lemmitization removed the endings in the words and converted them into unrecognizable words, hence when used GloVe embedding, it did not embed it to its original word but embedded all of them to a specific embedding(unknown). Surprisingly, it did not affect the performance that much.
- Did try hyperparameter tuning using random search as the computing resources were not sufficient. Tried using the Clserv but it was not working properly, so had to implement all on Google Colab.

Observation:

BERT is the new big thing in the NLP department. LSTMs are relatively old but still effective with newer concepts like Attention. Tf-Idf is the most basic way to implement but is not effective in complex problems.