CSA 250 : Deep Learning Project III Report

Abhishek Kumar (15648)

May 15, 2020

Project III : Problem Statement

Natural Language Entailment : Given two natural language sentences, a premise P and a hypothesis H, the textual entailment task – also known as natural language inference (NLI) – consists of determining whether the premise entails, contradicts, or is neutral with respect to the given hypothesis (MacCartney and Manning 2009).
Teaching machines how to read, reason, and answer questions over natural language questions is a long-standing area of research; doing this well has been a very important mission of both the NLP and AI communities. In this project we are trying to solve the natural language inference using deep learning methods. But before using these texts as input for our deep-learning model we need to convert it into numbers or vectors. We are comparing the performance of two model :
(1) Tf-Idf feature method with logistic regression
(2) Deep learning model using LSTM or pre-trained model such as BERT, XLNet

Dataset

Stanford Natural Language Inference (SNLI) The SNLI corpus (version 1.0) is a collection of 570k human-written English sentence pairs manually labeled for balanced classification with the labels entailment, contradiction, and neutral, supporting the task of natural language inference (NLI), also known as recognizing textual entailment (RTE). This dataset is provided by the Stanford NLP community.

Dataset Preprocessing
We know that input to deep learning models are the tensors and these tensors are made up of numbers but an english sentence contains word where each word is a collection of alphabets i.e., we cannot directly feed these texts to our deep learning model for training. Hence, these corpus needs pre-processing before it can be used. We followed following steps to preprocess the corpus:

1. **Removal of STOPWORDS** : removed words which are used frequently in a sentence but give little information about the sentence. Some examples are : 'is', 'he', 'the', 'are' etc.
2. **Removal of punctuation and special characters** : similar to stop-words, these words hold little or no information.
3. **Removal of white-spaces and alphanumeric characters :** The idea behind removing these words is similar to step 1 and 2. It is very crucial to remove these stopwords and special characters because they occur very frequent in a sentence and it can make the model's learning very difficult/skewed. Model may give more importance to these because there frequencies is very high.

Model : Multi-Layer Neural Network

• **Initial Model Architecure** :
No. of hidden layers : 4
Layer 1 : 1024 neurons
Layer 2 : 512 neurons
Layer 3 : 64 neurons
layer 4 : 20 neurons and it is connected to output layer consisting of 10 neurons.

Initial Hyperparameters :
Loss Function : Cross-Entropy Loss Optimizer : SGD with learning rate 0.01 and momentum 0.9.
Epochs : 100 epochs (Until validation error started increasing)
Batch Size : 1024
Initial Results:
Accuracy :88.33% and Test loss : 0.001641760636420467

The main idea to start with this architecture is that the input feature is 784 dimensional and we wanted to project it to higher dimensional such that the classes can be classified linearly. Subsequently I reduced it to the number of classes i.e.,10. • **Final Model Architecure** :
No. of hidden layers : 4
Layer 1 : 1024 neurons
Layer 2 : Dropout layer with p = 0.5 Layer 3 : 512 neurons
Layer 4 : 128 neurons and it is connected to output layer consisting of 10 neurons.

Final Hyperparameters :
Loss Function : Cross-Entropy Loss Optimizer : Adam with learning rate 0.01, $\beta_1 = 0.9$ and $\beta_2 = 0.98$
Epochs : 300 epochs (Until validation error started increasing)
Batch Size : 1024
Final Results:
Accuracy :89.3% and Test loss : 0.0002153349013776826

Figure 1: MLP Training Loss

I added a dropout layer after the first layer to add regularization and also to make the model more robust. It will ask the model to put more focus on the important features. I dropped the layer with 64 neurons s.t. the no. of parameters of the model should not be bigger. I experimented with different optimizer and I found that Adam in this case was working more efficient compared to SGD.

Model : Convolutional Neural Network

• **Initial Model Architecure** :
Standard Le-Net architecture :
First Conv layer : Conv2d(1, 32, kernel size=(5, 5), stride=(1, 1), padding=(2, 2))
Second Conv layer : Conv2d(32, 64, kernel size=(5, 5), stride=(1, 1), padding=(2, 2))
Fully connected layer: Linear(input features=3136, output features=20, bias=True)
Output Layer: Linear(input features=20, output features=10, bias=True)

Initial Hyperparameters :
Loss Function : Cross-Entropy Loss Optimizer : SGD with learning rate 0.0001 and momentum 0.9.
Epochs : 100 epochs (Until validation error started increasing)
Batch Size : 64
Initial Results:

Accuracy :88.82% and Test loss : 0.0009086105575193237

confusion_linear.png

Since the image size is very small as well as it does not contain much complex structure so its better to start with a simple architecture rather than going for a biffar and complex architecture with Lot parameter.

- **Final Model Architecure** :

First Conv layer : Conv2d(1, 32, kernel size=(3, 3), stride=(1, 1), padding=(1, 1)) Second Conv layer): Conv2d(32, 64, kernel size=(5, 5), stride=(1, 1), padding=(2, 2)) Third : Conv2d(64, 128, kernel size=(7, 7), stride=(1, 1), padding=(3, 3)) Fully connected layer : Linear(input features=6272, output features=100, bias=True) Dropout Layer : Dropout(p=0.25, inplace=False) Output layer : Linear(input features=100, output features=10, bias=True)

Final Hyperparameters :
Loss Function : Cross-Entropy Loss Optimizer : SGD with learning rate 0.01 and momentum 0.9.
Epochs : 300 epochs (Until validation error started increasing)
Batch Size : 2048
Final Results:
Accuracy :92.11% and Test loss : 0.0025724479497558402

Figure 2: Conv Training Loss

confusion_conv.png

I added a dropout layer after the first layer to add regularization and also to make the model more robust. It will ask the model to put more focus on the important features. I dropped the layer with 64 neurons s.t. the no. of parameters of the model should not be bigger. I experimented with different optimizer and I found that Adam in this case was working more efficient compared to SGD.