Assessment 3 – Language Processing

Intro:

The aim of this assignment was to create a language processing program that would learn and predict sentiment and industry category from a list of reviews about a set of businesses. I will be using a LSTM recurrent Neural Network to process the text and splitting the data into 80/20 train/validation for training and assessing.

Language processing for sentiment is very important for businesses today as it allows decision makers to process large amounts of feedback quickly to gauge how their services are being received by their customers.

Data Description:

The first step in the analysis was looking at the data to understand how it is being presented. Opening the json file I see a rows of dictionaries with labels for sentiment and category and features as the review text. Sentiment from the reviews was labelled 0=positive and 1=negative and the industry category was labelled 0=Restaurants, 1=Shopping, 2=Home Services, 3=Health & Medical, 4=Automotive.

Pre-processing:

We will now need to do some pre-processing on the data to det it into the form required for our Neural Network model. The reviews will need to be split up into individual unique words, then they will be numericalized into positive integer values, and then a vectorized vocabulary will be created for each word.

Firstly, I use the torchtext tokeniser function to create the list of individual words. Then I did some further pre-processing by removing common stop words. These are words like ‘a’, ‘the’, ‘you’re’ ect that add to context to the sentiment of the sentence. For the sake of time and memory I used a pre-set list of stop-words/join-words. Following this the words would be indexed into positive integer values. This process is all put a data structure called a ‘Field’ which sets the datatype. We also run the Label data through the ‘Field’ structure as well. Next, we load the data from the json file, which takes in all the ‘Field’ pre-processing instructions. Then the data is formed into a tabular data structure.

We now need to ‘vectorize’ the words and build a vocabulary. This process creates n-dimensional vectors for each word based on pre-trained data. The pretrained vectors sit in a word-vector space and represent each word as a position in the space. Words of similar meaning will cluster together in the space while different words will be far apart. The tokenized words in our corpus will be represented in this space and the vectors form our vocabulary. I have used Glove word vectorisation with 50 dimensions to represent our vocabulary.

The model:

For our NN model we will be using an LSTN recurrent NN. The LSTM model is used because it has the ability to retain sequential information, and so will be able to recognize context from a sequence of words. It also alleviates the issues that are associated with the vanishing gradient problem which are inherent in deep NN’s.

The structure of our model will be as follows:

- Inputs received from vectorized-vocal-form of each sentence example
- Inputs fed into LSTM model
- The hidden outputs are run through fully connected linear model which the forms the output

Since we are feeding in the vector form of the vocabulary, I have left out the embedding layer. The embedding layer is the process that converts the tokens into vectorized words. So im using our pre-trained vectors (from Glove) as the inputs to the LSTM. Before I run the input through the LSTM I've declared empty hidden an cell variables as well as using the torch.utils function to pad out the sequences as they have varying length.

The model will output two tensors, the rating tensor and the category tensor. Rating tensor is of size [batch_size, 2] and the category tensor is of size [batch_size, 5].

Loss Function:

For the loss function I have chosen the Cross Entropy loss. I used the cross-entropy loss as our output is a set of probabilities and this loss function will be able to measure how far the output is from the true label. I have used this loss function across both outputs of our LSTM NN and then summed them together to produce a single loss output.

Data splitting:

To avoid overfitting, the model I have broken up the data into 70% training, 30% validation. This will allow for model assessment (after training) to be done on a set of unseen data. I also used the dropout method on the hidden layer at a probability of 0.5 to further avoid overfitting.

Optimiser:

I ran the model quit a few times using both the adam and sdg optimizers with different learning rates to assess which performed best. I found that the sdg optimiser with learning rate of 0.01 performed quite well in the instance, the adam optimizer would often plateau at certain points and not learn for a long time.

The epochs number I chose was 20, I found that it needed a decent level of epochs before it would reach a small loss value (<0.2). It took 15 epochs just for loss to go below 1.

References:

ml-cheatsheet.readthedocs.io. (n.d.). *Loss Functions — ML Glossary documentation*. [online] Available at: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html. [Accessed 07 Oct. 2021].

NS, A. (2020). *Multiclass Text Classification using LSTM in Pytorch*. [online] Medium. Available at: https://towardsdatascience.com/multiclass-text-classification-using-lstm-in-pytorch-eac56baed8df. [Accessed 07 Oct. 2021].

Educative: Interactive Courses for Software Developers. (n.d.). *Vocabulary - Natural Language Processing with Machine Learning*. [online] Available at: https://www.educative.io/courses/natural-language-processing-ml/N0Wr9zwpEmv. [Accessed 08 Oct. 2021].

Analytics Vidhya. (2021). *Sentiment classification using NLP With Text Analytics*. [online] Available at: https://www.analyticsvidhya.com/blog/2021/09/sentiment-classification-using-nlp-with-text-analytics/ [Accessed 08 Oct. 2021].