NLP CS577 Homework 2

Tunazzina Islam, islam32@purdue.edu

April 2020

Conceptual Questions

Our DMEMM implementation we have used three different architectures: Randomly Initialized embedding model, Pretrained Word2Vec model and Bi-LSTM model.

Q1

Hyper-parameter Tuning:

For Randomly Initialized embedding model, we use trigram feature. Let's assume w_i is the current word and t_i is the current tag. So for the context we use $< w_i, w_{i-1}, w_{i-2}, t_{i-1}, t_{i-2} >$. We have used following hyper-parameters:

no of epochs = 200

batch_size = 64

learning rate = 0.05

Optimizer = Adam

weight decay = 0.0001 (regularization parameter, we used L^2 regularization)

Word embeddings dimension size = 10

Tag embeddings dimension size = 10

We used 1 hidden layer Neural Network with dimension 50 * 100 * 4. For hidden layer, we used non-linear activation function ReLu. For the output layer, we used log_softmax. We use cross entropy loss for the loss function.

We will explain one of the hyper-parameters(learning rate) tuning procedure here with learning curve:

We randomly choose 80% train and 20% validation data from *twitter1_train.txt* file and we use *twitter1_test.txt* file for test data. We run those data with different learning rates = 0.01, 0.05, 0.1, 0.5 and plotted F1 score (after implementing Viterbi) (Fig. 1).

For learning rate = 0.05, we obtained highest F1-Score for validation and test data(validation: 0.155, test: 0.162). So we choose learning rate = 0.05.

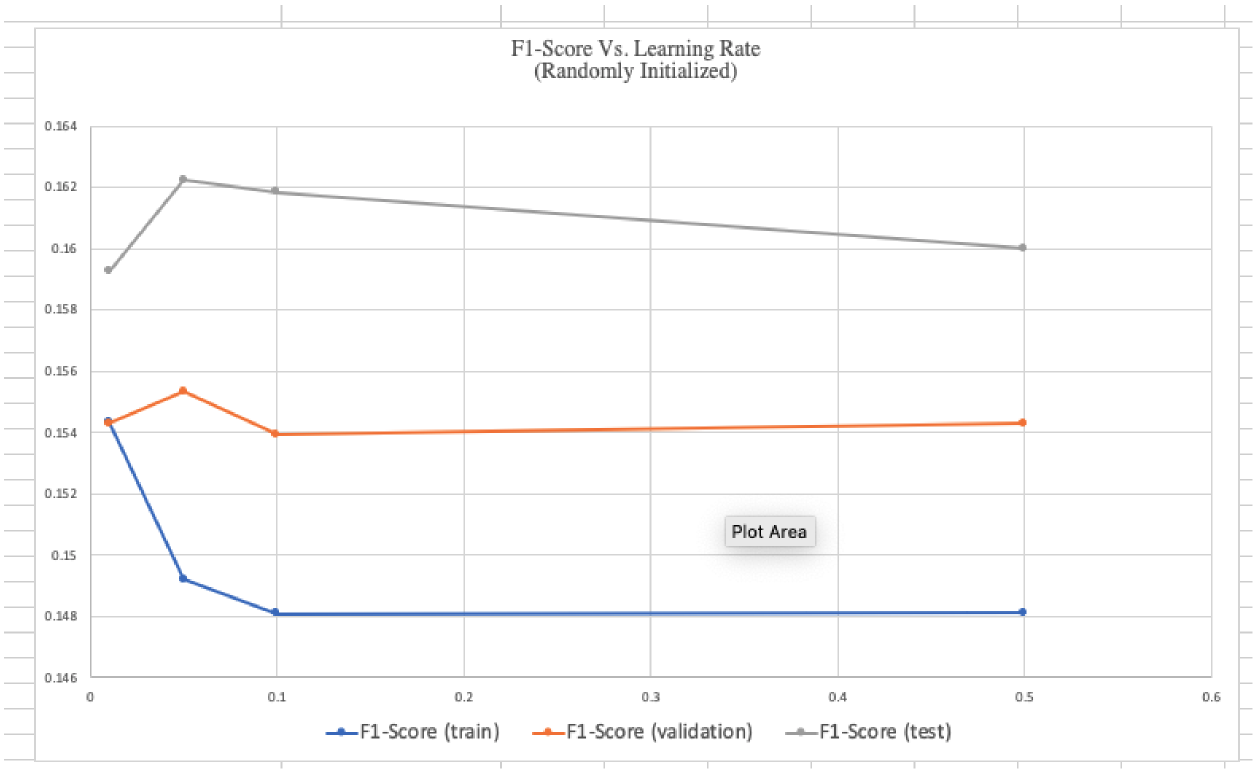

Figure 1: Learning curve F1-Score vs. Learning rate: Randomly Initialized Word Embedding

Q2

Best Neural Network:

In our work the best Neural Network model is the Pre-trained Word2Vec Model (Option 2). We achieved the highest **F1-Score = 0.36** using this model.

For Pre-trained Word2Vec model, we use unigram feature. Let's assume w_i is the current word and t_i is the current tag. So for the context we use $< w_i, t_{i-1} >$. We have used following hyper-parameters:

no of epochs = 200

batch_size = 100

learning rate = 0.05

Optimizer = Adam

weight decay = 0.0001 (regularization parameter, we used L^2 regularization)

Word embeddings dimension size = 300

Tag embeddings dimension size = 4

We used 1 hidden layer Neural Network with dimension 304 * 150 * 4. For hidden layer, we used non-linear activation function ReLu. For the output layer, we used log_softmax. We use negative log likelihood for the loss function.

Fig. 2 shows the train, validation, and test set performance as the epochs increase and training progresses. We can see that F1-Score of train and validation data is increasing with number of epoch increases. But after certain epochs F1 score of training is decreasing and validation is increasing. To avoid overfitting, we use early stopping. For early stopping, we only keep the model with best test F1-score in memory. We stopped at 200 epoch and test F1-score is 0.36 .

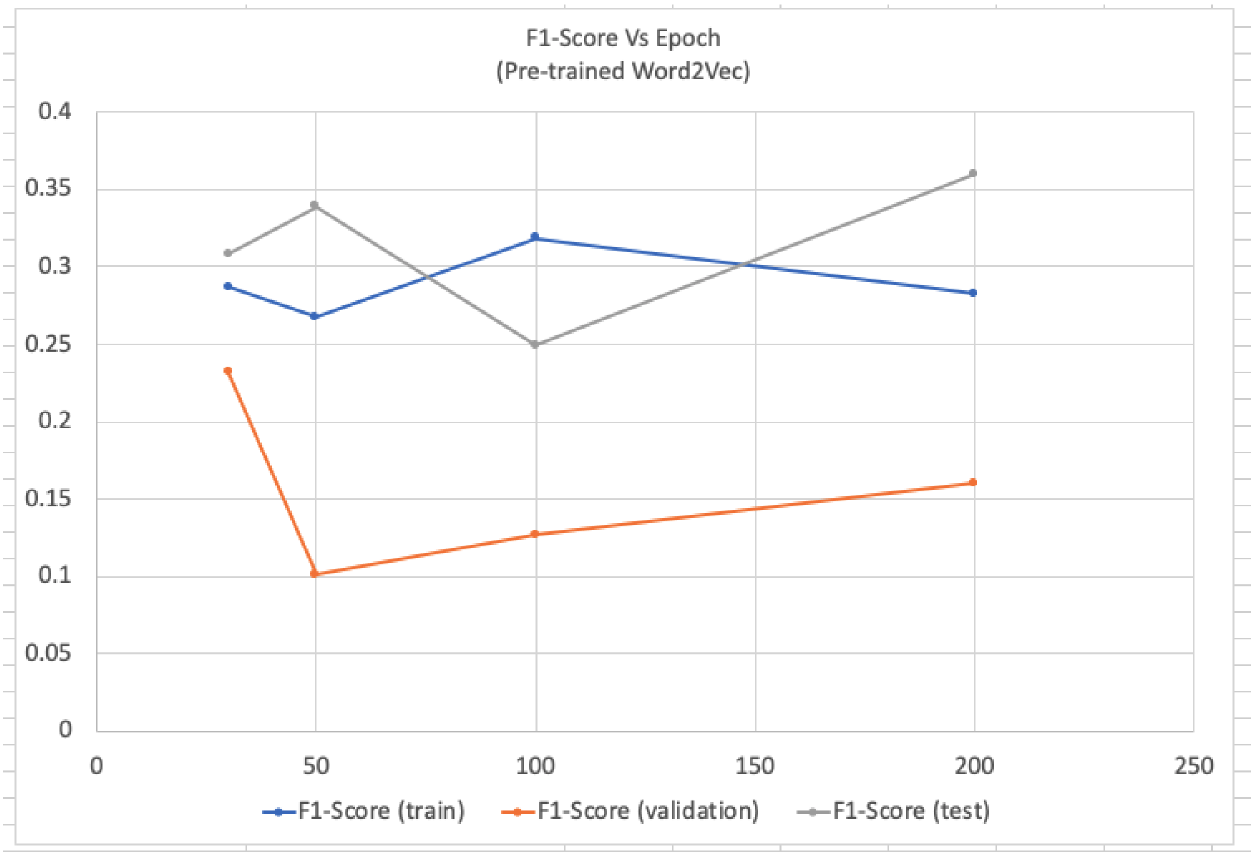

Figure 2: Learning curve F1-Score vs. Epoch: Pre-trained Word2Vec Embedding

Q3

Word Representations: (a)

We already discussed about the neural network architecture and hyper parameter of two models Randomly initialized model (please see Question 1) and Pre-trained Word2Vec Model (please see Question 2).

For Bi-LSTM model, we use Pre-trained Word2Vec as word representation. We provided the representation in our DMEMM model and we use Bidirectional LSTM there. We achieved F1-Score 0.24 for Bi-LSTM model.

We have used following hyper-parameters for Bi-LSTM:

no of epochs = 100

batch_size = 100

learning rate = 0.05

Optimizer = Adam

weight decay = 0.0001 (regularization parameter, we used L^2 regularization)

Word embeddings dimension size = 300

Tag embeddings dimension size = 4

We used 1 layer Bi-LSTM with dimension 304 * 150 * 4. For the output layer, we used log_softmax. We use negative log likelihood for the loss function.

Fig. 2 shows the learning curve (F1-Score vs. Epoch) for Pre-trained Word2Vec Embedding. F1-score = 0.36

Fig. 3 shows the learning curve (F1-Score vs. Epoch) for Bi-LSTM model. F1-score = 0.24

Fig. 4 shows the learning curve (F1-Score vs. Epoch) for Randomly Initialized Word Embedding. F1-score = 0.16

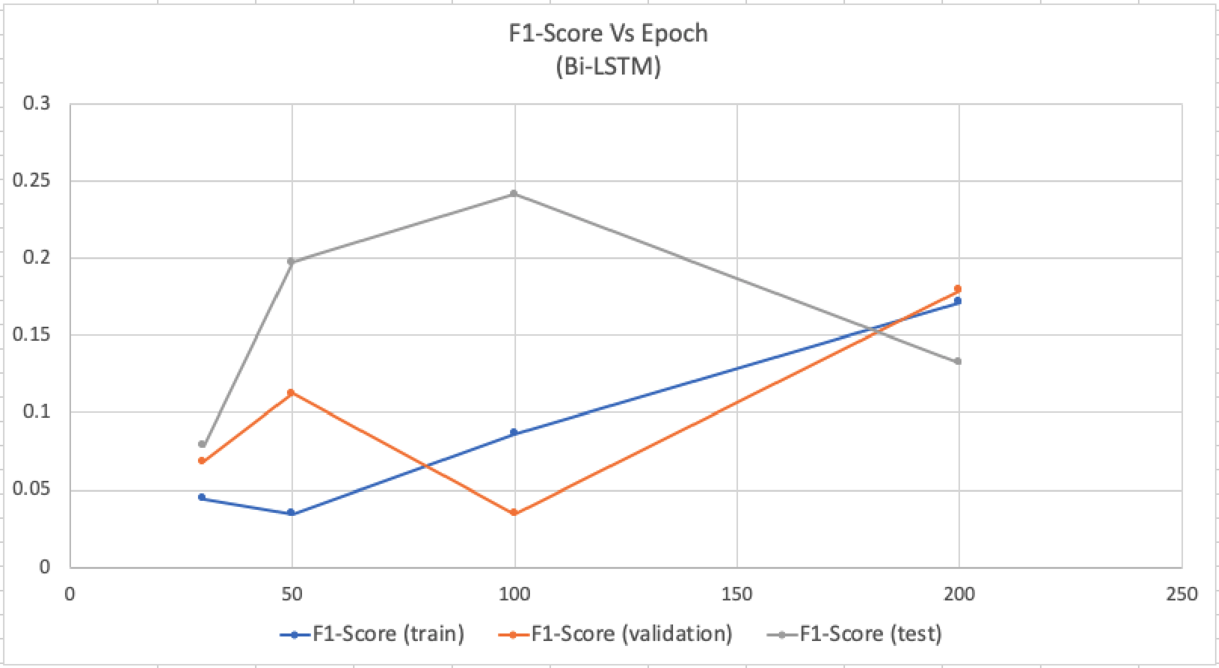

Figure 3: Learning curve F1-Score vs. Epoch: Bi-LSTM model

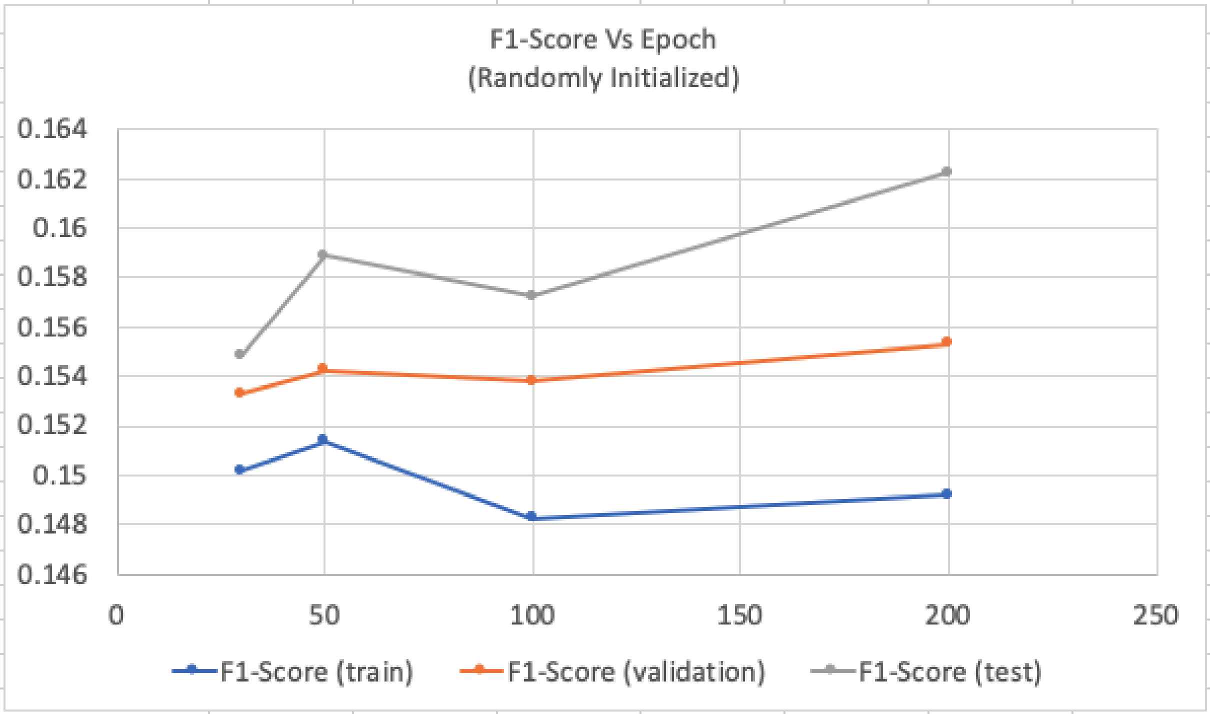

Figure 4: Learning curve F1-Score vs. Epoch: Randomly Initialized Word Embedding

Word Representations: (b)

According to the implementation, Pre-trained Word2Vec Embedding model has highest F1-score (0.36) than the other two models (Randomized : F1-score = 0.16, Bi-LSTM: F1-score = 0.24). Followings are the details result of those models:

1. Randomly Intitialized embedding: Precision = 0.089, Recall = 0.58
2. Pre-trained Word2Vec embedding: Precision = 0.53, Recall = 0.27
3. Bi-LSTM: Precision = 0.22, Recall = 0.27

Here we can see, Pre-trained has high precision score than others. Bi-LSTM has a blanced precision and recall, Randomized has the highest recall score. Precision is a measure of exactness or quality. Recall is a measure of completeness or quantity. This is more or less application oriented.

For this application, we would prefer to choose either Pre-trained Word2Vec or Bi-LSTM.

For Bi-LSTM, we don't have to worry about feature extraction as the hidden state and gating mechanism of the LSTM allows it to remember what it has seen in the past and utilize the appropriate information when making the current prediction. It can capture relevant context. For the word representation we have used Pre-trained word embedding and it might have already capture few context.

For Pre-trained Word2Vec, we need to define the feature to capture the context. Here, we used unigram feature. We tried with bigram but it could not improve the performance.

For randomly initialized model, we used trigram feature to capture the context. This model has high recall value that means it returned most of the relevant results. But it has low precision. We tried this model with bigram and unigram but it could not improve the F1-score.

Q4

If the model is only allowed to use the embedding of current word when making the prediction, it will not perform well as DMEMM because we don't have knowledge about previous tag. We will not be able to use Viterbi algorithm here. We implemented a Neural Network model to predict the current tag only based on current word. We used Pre-trained Word2Vec with same architecture like before for epoch = 30 but without including Viterbi. We got F1-score = 0.33. As it's higher than our randomly initialized trigram featured Viterbi model and Bi-LSTM. We think for this application we could use embedding of current word and then use Pre-trained Word2Vec model with Bi-LSTM. Bi-LSTM will capture the contexts.