C577 Spring 2022 Homework 2

Mohit Mayank – mayank@purdue.edu

PUID: 033744160

Note: I have used 4 late days for this assignment.

---

2 Conceptual Questions

- The feature for our Deep Maximum Entropy Markov Model consists of a pair of the current word and the previous tag, ie: $< w_i, t_{i-1} >$.
- For the first two options, the model consists of a Neural Network with a *Linear* input layer and a single *Linear* hidden layer. The model uses *RelU* as the activation function and *log_softmax* for the output.
- The third option has a bi-directional GRU as the input layer.
- The model uses **Cross Entropy Loss** for the loss function and **Adam** optimizer.

1. One of the hyperparameters used for tuning: **learning rate**.

   - The model was run for 100 epochs.
   - As we can see in Figure 2 and Figure 3, the model has really low accuracy for low values of learning rate. However for randomized embedding, the model performs the better only for lower values of learning rate.
   - For both train and test datasets, the F1 score increases and then dips. It can be explained as the model could be oscillating between local minima for higher learning rate and fails to reach the global minimum.
   - For validation dataset in Figure 2 and Figure 3, the F1 score increases with higher values of learning rate.
   - We chose the value where training F1 dips sharply while validation F1 increases. This value of 0.05 gave the optimal value of F1 for test (for contexualized embedding model) as can be seen in Figure 3.

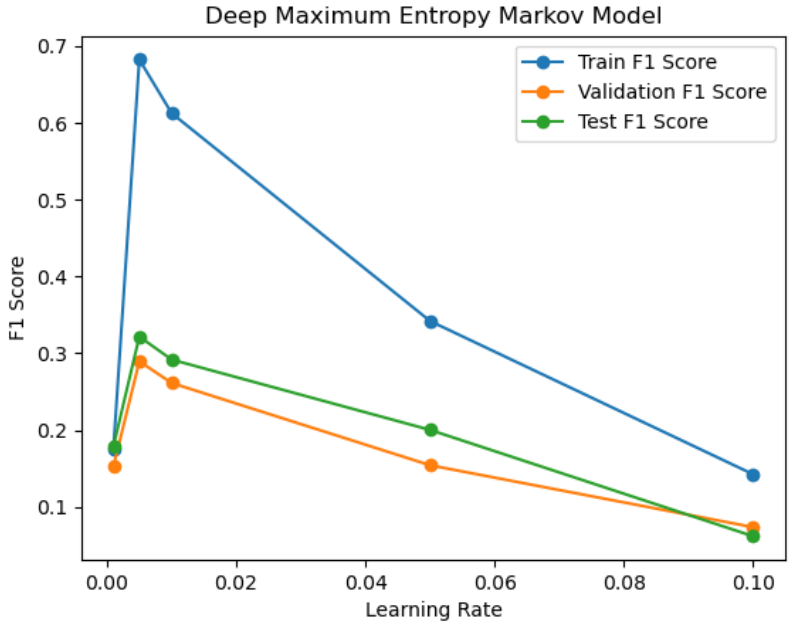

Figure 1: Randomized Embedding: F1 vs Learning Rate

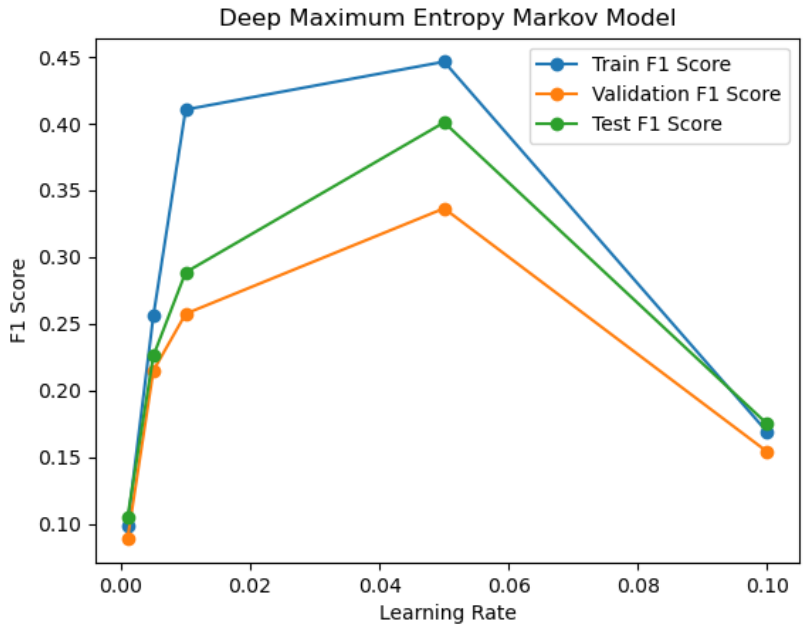

Figure 2: Word2vec Embedding: F1 vs Learning Rate

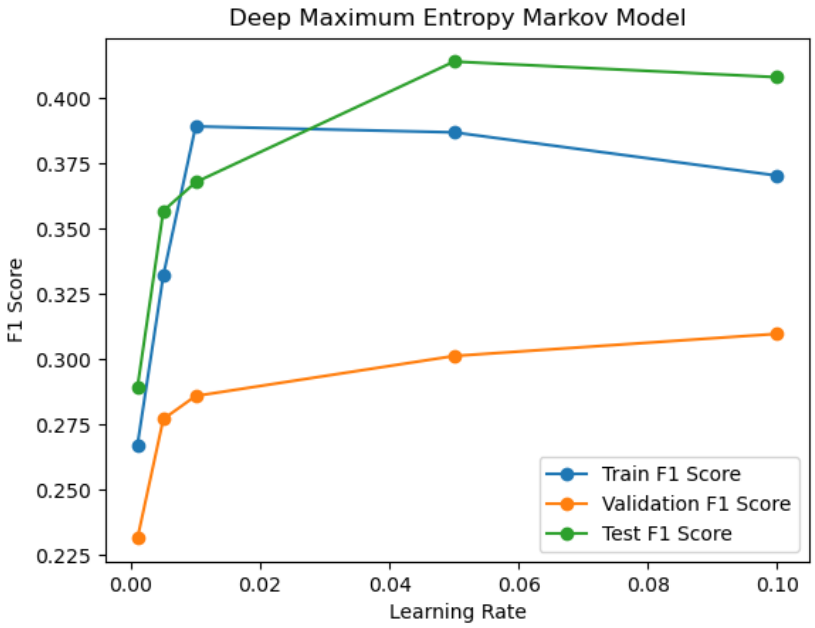

Figure 3: Contexualized Embedding: F1 vs Learning Rate

2. The following configuration gave the highest F1 score on test set:

Hyperparameters	**Value**
Option	3
Hidden layer neurons	128
Learning rate	0.01
Epochs	400

- As we can see in Figure 4, the F1 score increases for both validation and test and then starts to dip. However, for train the accuracy increases with higher epochs with a singular dip in between.
- This can be explained as the model generalizes better with higher number of epochs. Also, since the model uses GRU, it needs more iteration to generalize and learn the context better. However, eventually after 400 epochs, the F1 score drops as the model starts to overfit.

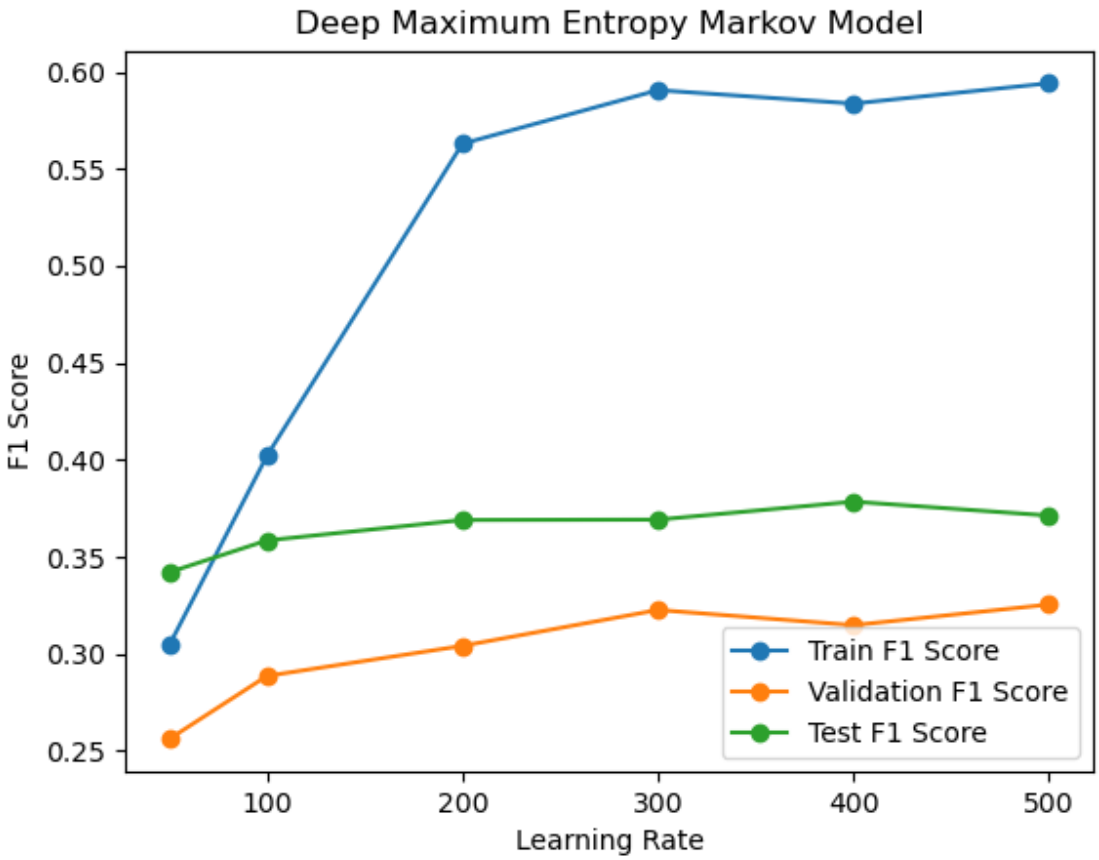

Figure 4: Contexualized Embedding: F1 vs Epochs

3. The graphs for the three models are below. Out of the three, the model using contexualized embedding with bi-directional GRU performed the best with an F1 score of 0.37.

   (a)
   - For the first two models (randomized, refer Figure 5 and word2vec, refer Figure 6), the F1 score for the models increase with the epoch till 200 epochs after which it dips. This can be explained as the model is able to learn with fewer epochs and then starts to overfit. Also, it was observed that due to the skew in tags, the model starts predicting only O when trained for more than 200 epochs.
   - For the contexualized embedding model with bi-directional GRU (refer Figure 7), the model's performance increases consistently until about 500 epochs. Since there are many more weight vectors, this model require more number of epochs for generalizing. Also, the model outperforms others since RNN is much more efficient for sequential models as it is able to learn relevant features while forgetting the irrelevant ones.

(b) I would use the model with contexualized embedding since it performs better and is well suited for the problem at hand, that is, a sequential dataset.

- Randomized Embedding
  - Pros:
    - Simple model, faster to train as there are fewer number of weights to learn.
  - Cons:
    - No context, thus the model fails to learn the relevance between different words in a sentence.

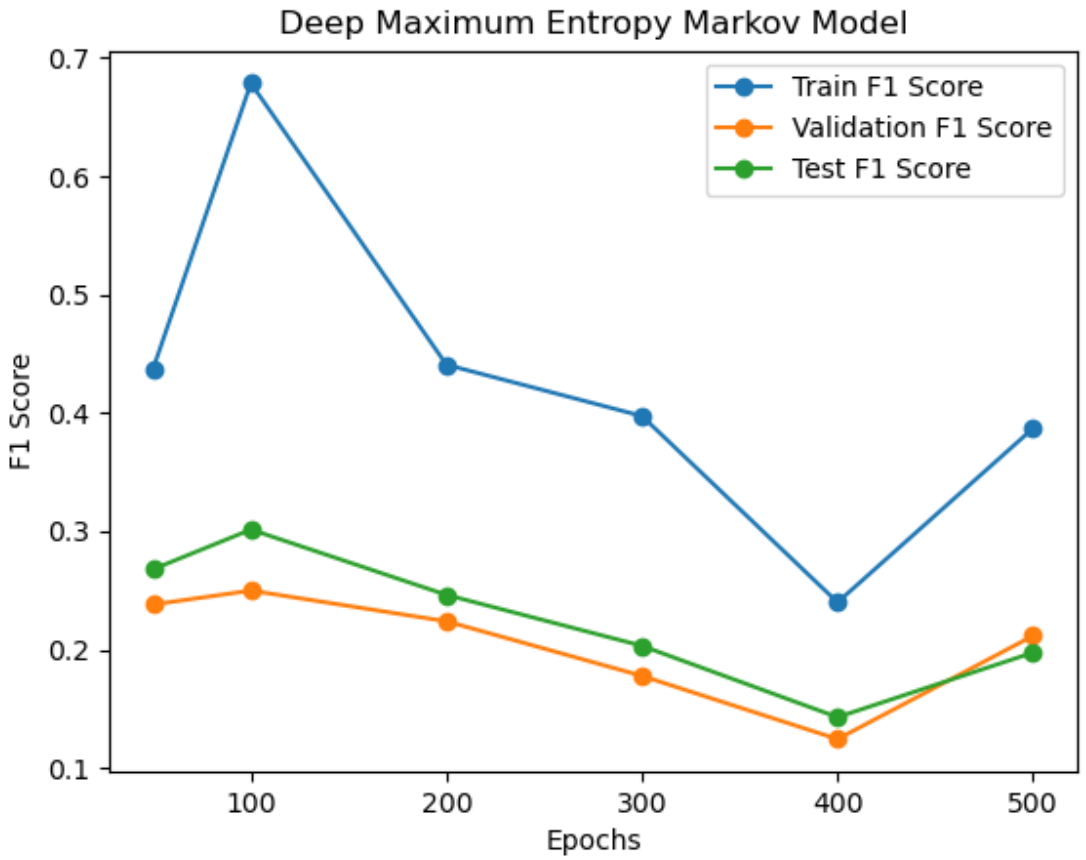

Figure 5: Randomized Embedding: F1 vs Epochs

- Word2vec Embedding
  - Pros:
    - Model has context between similar words, thus it is able to learn if *terrible* means a negative sentiment, then *awful* does too since the vector distance between the two words is small.
    - Still faster to train since there are same number of weights to learn.
  - Cons:
    - Model does not have context between different words in a sentence.

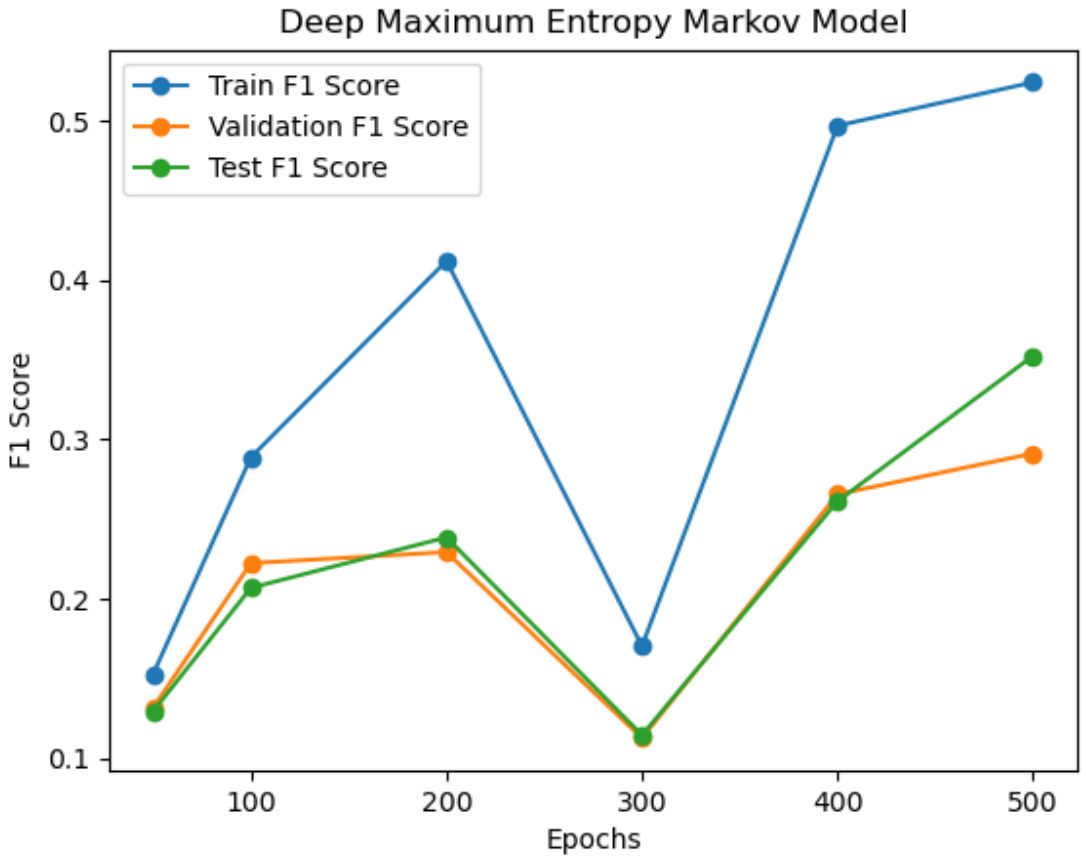

Figure 6: Word2vec Embedding: F1 vs Epochs

- Contexualized Embedding
  - Pros:
    - Since the input is a word2vec embedding, model has context between similar words as above.
    - Model learns the relation between the words in a sentence since we are training an RNN model which is one of the most effective model for sequential data.
  - Cons:
    - Model is slow to train since there are lots of weights and parameters to learn, thus it takes a higher number of epochs.

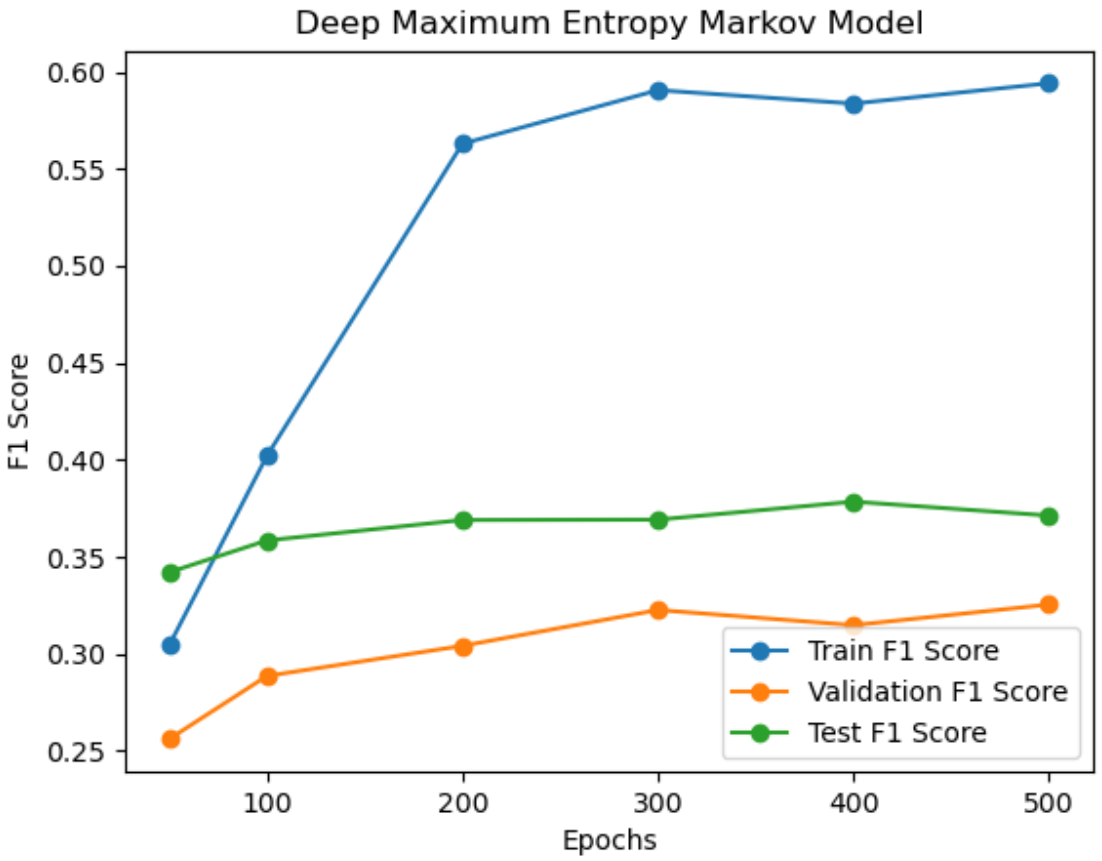

Figure 7: Contexualized Embedding: F1 vs Epochs

4. The model will not be able to perform well if it is only allowed to use the embedding of the current word. A Markov Model performs well for NER tasks because it uses both previous tag and the current word, as this allows it to have more context to predict the current tag. However, if we don't have the previous tag, we miss out on one of the features which would evaluate the probability $P(t_i|t_{i-1})$. We would also not be able to use Viterbi directly to find the best path for the predicted tags.

   That being said, even with just the current word, the model with contextualized embedding should be able to learn the context of the sentence and predict the tags with an average accuracy, however not as well as the Markov Model.