Sequential CRF for NER

Nidhi Kadkol
University of Texas at Austin
nidhikadkol@gmail.com

1 Introduction

Hidden Markov Models (HMMs) are generative models with a locally normalized form. They do reasonably well in Named Entity Recognition (NER) tasks. Conditional Random Fields (CRFs) are discriminative models with a globally normalized form. Their performance is much better than that of HMMs on the same NER task. In this project we implement the decode function for Viterbi algorithm for HMMs and extend that to CRFs. We also experiment with Beam Search.

2 Conditional Random Fields

We implement Conditional Random Fields to predict the tag of a word from a sentence. We use context features as well as structural features of the word itself. We also have to impose sequential constraints drawing inferences from the model. This is because we want the predicted sequence of tags to be sensible, for example an I-PER tag cannot follow a B-LOC tag. When training the model, we use the transition probabilities from the HMM model that are obtained by empirical calculations on the training dataset. We then pass these probabilities to our CRF Model (modifying the template of the class by doing so), and during the inference of CRF, before we evaluate the sentence passed in we set the values of the invalid transitions to `-np.inf`. While we get an F1 score of 76.89 with the Hidden Markov Model, we immediately see a huge jump when we switch to CRFs at the end of the first epoch itself getting 83.75 on the dev set. Running the model for 20 epochs gives us the highest F1 score of **88.73** on the development set at the 20th epoch, followed by 88.69 for the 9th, 20th, and 21st epoch. We did not run for further epochs due to time and computing power constraints.

3 Beam Search

Beam Search is an approximate search strategy that does not pick the best score over the entire list of tags, but over the top few scores in the list. This allows us to not have to scan through all the tags at every step and saves time when we have a large number of objects. We experimented with Beam Search in this project with respect to accuracy and efficiency. We did not see a major improvement in the time taken to run the model by using beam search. This is most probably because we use beam search in the decode phase where its complexity depends on the number of tags. Our total number of tags (9 tags) is small to begin with, so reducing that to 2 or 3 would not make a major difference in the program's running time.

4 Experimental Results

4.1 Optimizer Choice

The first run of CRF Models without using Beam search and with the Stochastic Gradient Descent (SGD) optimizer gives us a starting F1 score of 71.71. This takes 230 seconds to train for one epoch and then run on the dev and test sets. When we switch to UnregularizedAdagrad, we get an F1 score of 83.75 at the end of the first epoch, taking 263 seconds to train for one epoch and then run on the

dev and test sets. This is consistent with what we expect, as SGD will converge slowly due to noisy gradient updates as it does not taken into account the history of the gradients every time a gradient update is made. Adagrad has a running counter of the first order moments of these gradients, so it stabilizes the general direction of gradient update. Thus, even though each update of Adagrad requires more computation and slightly more memory than SGD, it will converge in fewer epochs and give us better performance in terms of F1 socre. This is also reflected in Figure 1. Keeping this in mind, we switch to Adagrad and perform the rest of our experiments using this optimizer. With Adagrad, the model converges to its high values of 88 within the first few epochs and then takes about 10 epochs to go from 88.60 to 88.73.

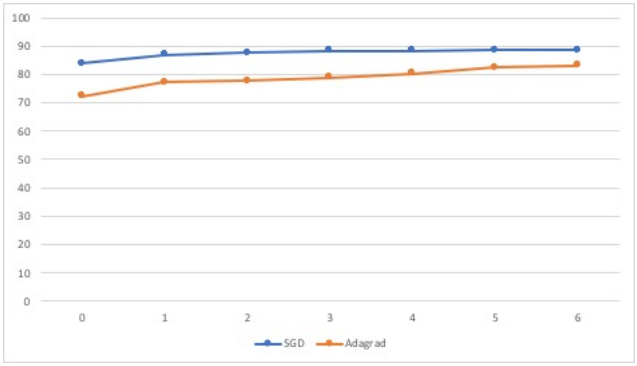

Figure 1: Graph of F1 scores versus epoch number on development set for SGD and Unregularized Adagrad

4.2 Beam Search

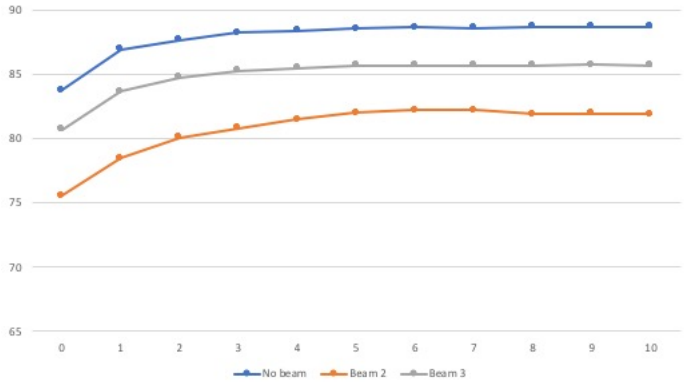

Figure 2: Graph of F1 scores versus epoch number on development set

We implement beam search with beam sizes of 2 and 3. Beams with size 2 give a significantly worse performance, starting with an F1 score of **75.57** on the development set and reaching **81.93** by the 10th epoch. This is expected, since with a beam of size 2 we are discarding all but the top two scoring tags at a given time step. The discarded tags might have been included in the final predicted tag sequence, as the marginal probabilities in subsequent timesteps which included these tags in the sequence could have been high. By discarding these tags, we are doing an approximation of greedy search over all the tags in the previous timestep, thus the accuracy of our model would be affected slightly. We also run the experiment with a beam of size 3, and this gives us a significantly better performance, with an F1 score of **80.68** on the development set at the end of the first epoch, and reaching **85.66** by the 10th epoch. Keeping track of the top 3 scoring previous tags is a much better approximation of the greedy search than keeping track of the top 2. We see from Figure 2 that in all three cases a stable value of the F1 score is reached by approximately the 8th epoch, but this F1 value differs for each choice.