CSE 447 NLP Assignment 4 Report

Amol Sharma

10th March 2019

1 Neural Structured Perceptron

1.1 Recurrence Derivation

$$\heartsuit_j(y_j) = \max_{y_1,...,y_{j-1}} \sum_{i=1}^{j} s(\mathbf{x}, i, y_{i-1}, y_i)$$
$$\heartsuit_j(y_j) = \max_{y_1,...,y_{j-1}} \left(\sum_{i=1}^{j-1} s(\mathbf{x}, i, y_{i-1}, y_i)\right) + s(\mathbf{x}, j, y_{j-1}, y_j)$$
$$\heartsuit_j(y_j) = \max_{y_1,...,y_{j-1}} s(\mathbf{x}, j, y_{j-1}, y_j) + \heartsuit_{j-1}(y_{j-1})$$

1.2 Algorithm

Input: $s(\mathbf{x}, j, y_{j-1}, y_j)$ for each $j \in \langle 1, ..., n\rangle, \forall y_{j-1}, \forall y_j$ where y_{j-1} and y_j are labels for POS
Output: $\hat{\mathbf{y}}$: the most likely sequence of parts of speech

1. **Base case:** $\heartsuit_1(y_1) = s(\mathbf{x}, 1, y_0, y_1) = s(\mathbf{x}, 1, \text{START}, y_1)$
2. For $j \in \langle 2, ..., n-1\rangle$:
   $\heartsuit_i(y_i) = \max_{y_{i-1}} s(\mathbf{x}, i, y_{i-1}, y_i) + \heartsuit_{i-1}(y_{i-1})$
3. Special case at end to include stop:
   $\heartsuit_n(y_n) = s(\mathbf{x}, n+1, y_n, \text{STOP}) + \max_{y_{n-1}} s(\mathbf{x}, n, y_{n-1}, y_n) + \heartsuit_{n-1}(y_{n-1})$
4. $\hat{y} \leftarrow \text{argmax}_y \heartsuit_n(y)$
5. For $j \in \langle n-1, ..., 1\rangle$:
   $\hat{y_j} \leftarrow \text{argmax}_{y_{j+1}} s(\mathbf{x}, j+1, y_j, \hat{y}_{j+1}) + \heartsuit_j(y_j)$

1.3 Viterbi

The above algorithm is very similar to the Viterbi algorithm. The time complexity of the algorithm is the same: $|\mathcal{L}|^2 n$. There are $|\mathcal{L}|n$ boxes that we need to go through, and in each box, we look at $|\mathcal{L}|$ of the previous tags.

1.4 Neural Model

We defined s to be the sum of the unary and binary log-potentials, where the unary log-potential depends on $\mathbf{x}$ and y_i and the binary log-potential is a function of y_{i-1} and y_i. Since we are using the Viterbi algorithm, the neural model has to be able to calculate s at each point/token in the sequence and be able to propagate our loss backwards. An RNN or LSTM makes sense for this application. Therefore, we can have ϕ_u modelled by an RNN that maps the input $(\mathbf{x}, y_i)$ to $\mathbb{R}$ and gives us the probabilities of each token and label set at all of the indices in a sequence. Whereas, ϕ_b would be a learnable 2 dimensional tensor that would give us the probability of the current label given the previous one. Combining these two, we can get s as specified.

2 Implementation in AllenNLP

Table 1: Grid Search for Hyperparameters

Encoder	Bat. Sz.	Opt.	LR	H. Lyr. Sz.	Num Lyr.	Elmo	Train. Acc.	Val. Acc.
stacked_bi.._lstm	128	adagrad	0.01	256	2	N	0.9980	0.9113
stacked_bi.._lstm	128	adagrad	0.01	128	2	Y	0.9899	0.9565
stacked_bi.._lstm	128	adagrad	0.01	512	2	Y	0.9899	0.9622
stacked_bi.._lstm	64	adagrad	0.02	512	2	Y	0.9577	0.9356
stacked_bi.._lstm	64	adagrad	0.02	128	2	Y	0.9382	0.9029

3 Kaggle

To beat the random and basic baseline, I did not change the A3 config apart from the fields required to make it work. My tuning procedure afterwards to improve accuracy was to do a grid search over the hyperparameters. I stuck with the encoder type and optimizer as they had worked well in both A3 and in these tests.

4 Bonus

Although my model's accuracy isn't as high as many of my peers', I do think that my highest validation accuracy of 0.96 and public test accuracy of 0.91 is somewhat successful. To get to this accuracy from my basic model, the main change was using elmo embeddings. I think that the metrics would have been even higher if I had used a larger number of hidden dimensions or the larger embeddings from elmo: I had to use the smallest possible size as the servers were frequently running out of memory. Having more information/features in those larger embeddings would have proved especially useful for unseen data.