Natural Language Processing - Home Work 3

CSE 538 Fall '19

Prof. Niranjan Balasubramaniam

Submission By:

Narayan Acharya

112734365

Contents

1 Model Implementation

1. **Model Architecture:**

   Our model is a neural network for dependency parsing. We try to emulate the dependency parser as mentioned in the paper [1]. In accordance with the model architecture in the paper we build a 2 layer architecture.

   For the first layer, we have *hidden_dim* number of units. The first layer is followed by an activation function. The paper mentions the comparison of various possible activation functions, namely, cubic, hyperbolic tangent and sigmoid. We try each as part of our experiments and analyse the results from each. The next layer contains *num_transitions* number of units, with each unit corresponding to the probability of it being the next transition in constructing the parse tree. We get the probabilities by applying a softmax function to the second layer.

   The first layer has two variables that we intend to learn, the weight matrix W^1 and bias b^1, while the second layer has a single weight matrix W^2. Given we have to learn these, it is critical to come up with the good initial values for these weights and biases. We use Xavier initialization for both the weights, W^1 and W^2, and we start with zeros for the bias b^1 in layer 1. We also learn the embeddings as part of our experiments. In cases where we do wish to learn the embeddings as well, we initialize them between -0.01 and 0.01 using a uniform random initializer.

   NOTE: The paper denotes weights for layer 1 using 3 separate matrices, one for each - words, Part of Speech (PoS) tags and labels. We can get away with just 1 weight matrix, W^1 in our case, by simply concatenating the columns of the 3 matrices. In order for this to work, we also need to make smarter ways of reshaping our input matrices coming into the model. This is to ensure that the matrix calculations we do end up having the same effect as if the single weight matrix were 3 separate matrices like we have in the paper.

2. **Model Forward Propagation:**

   The input to the first layer is a matrix of embeddings of the tokens. We reshape these inputs into a long one dimensional vector as to conform to the requirements of the input to the first layer of our model.

   Forward propagation is greatly simplified due to the use of single weight matrix in layer 1. We multiply the inputs coming in with W^1 and then add the bias b^1 to the result. Next we pass the computed value through our activation function to get the output of the first layer of our architecture. The default activation function is cubic, as suggested in the paper. The output of the first layer is then passed through to the second layer. The output of the second layer is just multiplying the input with weight matrix W^2.

3. **Loss Calculation:**

   When we are training the model, we also need to compute the loss for back propagation which will update our trainable variables, W^1, b^1, W^2 and our embeddings when they are trainable. The loss we calculate has 2 components, the cross entropy loss and a regularization term, which are added to compute the final loss.

   (a) Cross Entropy Loss: As mentioned in the paper, we do **not** calculate the cross entropy loss for all the logits from layer 2, but only for the feasible transitions. Thus, we mask out the transitions with labels < 0 and compute the softmax for all the other transitions. We calculate the "stable softmax" instead of the regular softmax to protect against underflowing and overflowing problems when we encounter unusually small or large values in our logits. We also add a small noise (e^{-9}) while taking the log of softmax while calculating the cross entropy loss. We do this to avoid taking a log of 0 when the softmax is 0. Finally, we take the mean value of the cross entropy loss over the entire batch.

   (b) Regularization: The regularization term consists of L2 loss for our trainable variables. We include the embeddings as part of the regularization term only when they are trainable. We multiply the sum of the L2 loss for our trainable variables with the regularization parameter (λ) to compute the regularization.

4. **Model Features:**

   The features to our model are extracted from the state of the configuration after various transitions are applied. The configuration is representative of the stack, the buffer and set of dependency arcs. The transitions are carried out based on the rules as mentioned in [1] based on the possible transitions as mentioned in [2]. We have 3 different transition options at any given time, LEFT-ARC, RIGHT-ARC or SHIFT but we can apply these only if certain conditions are met. For. eg. We cannot add a LEFT-ARC of RIGHT-ARC if there are less than 2 elements in the stack and we cannot SHIFT something from the buffer to the stack if the buffer is empty.

   The features are from the state of the stack, buffer and parse tree at that instance of the parsing of the sentence. What we wish to predict is the next transition of all the possible transitions at that instance. This will help with generating a better dependency parse tree as we make the best possible guess for adding an arc to the parse at any given state. We extract certain features, 48 in total, as mentioned [1]. 18 of these are for the words, another 18 for the PoS tags and 12 features for dependency arcs.

Narayan Acharya
112734365

Exp	Activation	Pre-Trained Embeddings	Loss	UAS	UAS^{-}	LAS	LAS^{-}	UEM	UEM^{-}	ROOT
1	Cubic	YES	0.10	88.00	**89.63**	85.39	86.67	34.06	36.65	90.12
2	Tanh	YES	0.13	87.13	88.80	84.74	86.10	33.12	35.88	88.06
3	Sigmoid	YES	0.17	85.92	87.70	83.47	84.93	30.24	32.41	87.06
4	Cubic	NO	0.09	85.02	86.88	82.62	84.17	27.76	30.47	82.71
5	Cubic	YES*	0.14	84.92	86.73	82.17	83.65	29.59	31.83	86.18

Table 1: Experiment Results

2 Experiments

1. The results in 1 are for the 5 experiments carried out. The numbers reported are numbers after the 5 epochs run for each experiment. Below are the meanings of some of the relevant columns [3] [4].

   (a) Activation: The activation function used in layer 1 of our model.

   (b) Pre-Trained Embeddings: Whether pre-trained embeddings were used for training. NOTE: The 5th experiment did use pre-trained embeddings but they were not trainable.

   (c) Loss: The average training loss for that epoch.

   (d) UAS: Unlabeled Attachment Score - Percentage of words that get the correct head.

   (e) UAS^{-}: Unlabeled Attachment Score (Without Punctuation) - Percentage of words that get the correct head. Here we disregard punctuation tokens.

   (f) LAS: Labeled Attachment Score - Percentage of words that get the correct head and the label.

   (g) LAS^{-}: Unlabeled Attachment Score (Without Punctuation) - Percentage of words that get the correct head and the label. Here we disregard punctuation tokens.

   (h) UEM: Unlabeled Attachment Score with Exact Match - Percentage of exact match dependency parse trees.

   (i) UEM^{-}: Unlabeled Attachment Score with Exact Match (Without Punctuation)- Percentage of exact match dependency parse trees. Here we disregard punctuation tokens.

2. The UAS^{-} score of 89.63% is very close to the numbers reported in the paper for the cubic activation function. [1]. **NOTE:** The paper's mention of UAS/LAS is equivalent to UAS^{-}/LAS^{-} over here as all numbers reported for UAS/LAS are without punctuation in the paper.

3. Cubic activation function does out-perform hyperbolic tangent and sigmoid as can be seen from the results above. Cubic beats both these activations by almost 1% and 2% respectively when it comes to UAS^{-}.

4. Use of pre-trained embeddings is also a positive factor. Experiment 4, the one without the pre-trained embeddings performed poorly compared to the experiments where pre-trained

embeddings were used with any of the three activation functions. It only managed to get an accuracy of 86.88% for UAS^{-}.

5. For the last experiment, we use pre-trained embeddings but do not train/learn them over the course of the training. This model managed a accuracy slightly poor than the model without the pre-trained embeddings for the metrics for head and labels but somehow did well on the exact tree matches and ROOT scores.

References

[1] [Chen and Manning, 2014] Chen, D. and Manning, C. (2014). A fast and accurate dependency parser using neural networks. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 740-750.,
https://nlp.stanford.edu/pubs/emnlp2014-depparser.pdf

[2] [Nivre, 2004] Nivre, J. (2004). Incrementality in deterministic dependency parsing. In Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together, pages 50-57. Association for Computational Linguistics.,
https://pdfs.semanticscholar.org/0329/06f0d86fd6bbf7512d3fffd06fcb83593012.pdf

[3] NLP Dependency Parsing Lecture Slides by Joakim Nivre,
https://cl.lingfil.uu.se/~nivre/master/NLP-DepParsing.pdf

[4] Transition-based dependency parsing Lecture Slides by Sara Stymne
https://cl.lingfil.uu.se/~sara/kurser/5LN455-2014/lectures/5LN455-F8.pdf