CSE538 Natural Language Processing Fall 18

Prof. Niranjan Balasubramanian
Stony Brook University

Assignment 3
16th November, 2018

Name - Rushikesh Nalla
ID - 111971011

1 Number of Hidden Layers -

Default Configuration of parameters -

- max_iter = 1001
- batch_size = 10000
- hidden_size = 200
- embedding_size = 50
- learning_rate = 0.1
- n_Tokens = 48
- lam = 1e-8

I will be mentioning default parameters in the report where ever this set of parameters have been used for the implementation so as to avoid repeating the same information everywhere.

The paper has proposed only one hidden layer but I have explored two and three hidden layers networks. I have used the default set of parameters with some additional settings as mentioned in the table 1.

- A network with one hidden layer as mentioned in paper performed the best keeping in mind the increase in computation cost for deeper networks.
- Using the cube activation function for all 3 hidden layers performed the worst because of the substantial increase in parameters that needs to be trained. Increasing the training data and the number of iterations of training would improve the results.
- Changing the activation function to relu for second and third hidden layers improved the results significantly for 3 hidden layer network and helped in faster convergence.
- We can conclude that one hidden network is good enough for the given training data. The improvement in performance considering the increase in parameters and computation time is not significant.

Hidden Layer Configuration	Loss	UAS	UASnoPunc	LAS	LASnoPunc	UEM	UEMnoPunc	Root
Hidden Layers = 1 Layer 1 units = 200 Activation 1 = cube	0.351	70.486	73.557	66.512	69.179	10.705	11.647	57.411
Hidden Layers = 2 Layer 1 units = 200 Layer 2 units = 200 Activation 1 = cube Activation 2 = cube	0.433	67.023	70.400	62.252	65.319	7.882	8.294	49.823
Hidden Layers = 3 Layer 1 units = 200 Layer 2 units = 200 Layer 3 units = 200 Activation 1 = cube Activation 2 = cube Activation 3 = cube	4.510	16.043	15.887	0.266	0.299	0.588	0.588	2.941
Hidden Layers = 2 Layer 1 units = 200 Layer 2 units = 200 Activation 1 = cube Activation 2 = relu	0.336	68.222	71.454	64.067	67.088	8.294	8.588	48.823
Hidden Layers = 3 Layer 1 units = 200 Layer 2 units = 200 Layer 3 units = 200 Activation 1 = cube Activation 2 = relu Activation 3 = relu	0.330	72.350	75.170	68.379	70.816	11.647	12.823	62.117

Table 1: Different number of hidden layers

2 Different Activation Functions -

I have explored the cubic (mentioned in paper), sigmoid, tanh and relu activation functions with the default set of parameters and obtained the following results.

- The cubic non linear function works the best as it captures the interactions between words, tags and labels which is essential. Hence it was used in the paper.
- Relu and Tanh were the next best set of activation functions. It fails to capture the cross interactions between the input (words, tags and labels).
- Sigmoid was the worst of all because of the same reason as above, it fails to capture the cross interactions between the input (words, tags and labels).

Activation Function	Loss	UAS	UASnoPunc	LAS	LASnoPunc	UEM	UEMnoPunc	Root
Cube	0.351	70.486	73.557	66.512	69.179	10.705	11.647	57.411
Sigmoid	1.172	42.819	46.515	31.552	34.767	1.411	1.411	9.176
Tanh	0.562	56.906	60.213	50.903	54.007	3.941	3.941	33.470
ReLU	0.549	57.641	60.981	51.708	54.778	4.0	4.117	34.941

Table 2: Performance of different activation functions

3 Three separate Parallel Hidden Layers -

Having separate parallel hidden layers means that there should be no possible interaction between the word, POS and label embeddings. Due to the lack of such cross-connections it lead to bad results. This is obvious because these words, tags and labels depend on each other and help in transition predictions. I have used the default set of parameters for this experiment with a change in the size of the hidden layer. It was set to 300. 100 for each of word, pos and label embeddings which was concatenated.

Separate Parallel Hidden Layers (Separate Word, POS and Label Embeddings)							
Loss	**UAS**	**UASnoPunc**	**LAS**	**LASnoPunc**	**UEM**	**UEMnoPunc**	**Root**
0.767	42.463	44.978	32.664	34.392	2.647	2.764	16.235

Table 3: Separate Word, POS and Label Embeddings

4 Initial Random Configuration for weight matrices -

Initialized the weight matrices randomly first using the normal distribution and second using uniform distribution. The bias matrix was initialized with zeros (this was the best setting). Weights from normal distribution helped in faster convergence and hence gave better results as compared to weights from uniform distribution. Training for more number of iterations might improve the results of the latter. I have used the default set of parameters for this experiment.

Configuration of weight matrices	Loss	UAS	UASnoPunc	LAS	LASnoPunc	UEM	UEMnoPunc	Root
Normal Distribution	0.351	70.486	73.557	66.512	69.179	10.705	11.647	57.411
Uniform Distribution	33.484	48.879	51.783	39.544	41.327	3.117	3.294	34.647

Table 4: Distribution of initial random configuration of weight matrices

5 Fixing the Embeddings -

On fixing the word, POS and label embeddings (this is done by setting the parameter of tf.Variable (trainable=False) corresponding to self.embeddings) we observe very bad predictions because learning the embeddings is critical for the model to be able to predict the transitions correctly. It means that we are not updating/optimizing the embeddings during back propagation to minimize loss according to the training data. I have used the default set of parameters for this experiment.

Embeddings	Loss	UAS	UASnoPunc	LAS	LASnoPunc	UEM	UEMnoPunc	Root
Not Fixed	0.351	70.486	73.557	66.512	69.179	10.705	11.647	57.411
Fixed	1.61	23.261	24.752	12.151	13.030	0.764	0.764	5.058

Table 5: Fixing Word, POS and Label Embeddings

6 Gradient Clipping -

Gradient Clipping is a technique that is used to fix the problem of either vanishing or exploding gradients. Here, we face the problem of exploding gradients. Exploding gradients occurs while training the neural network due to multiplication of gradients having values > 1 when the network is updating the weight matrices during back-propagation. This results in very large updates to the network weights and the values may overflow and result in NaN values. We clip the gradient to some maximum value if they exceed a certain threshold. This way the gradients never explode and we have a stable neural network.

If gradient clipping is removed from the code, the loss value goes from 4.5 (approx value) at step 0 to NaN at step 100. This is due to the exploding gradients.

7 Different Configurations explored -

Exploring different settings of the parameters for obtaining the best results. Following are the observations -

- We get good results using the default setting of parameters but we can definitely improve by tuning the parameters.
- On increasing the max_iter we observe that the performance keeps improving. We should be careful with the increments to avoid over-fitting.
- Increasing the learning rate also helped in faster convergence and produced good results.
- So I decided to increase both max_iter and learning_rate and observed a substantial increase in performance.
- Increasing the max_iter to 5001 further improved the results (gave the best results).
- Increasing the hidden layer size improved the results but not by a big margin because the model has already learnt (as we have increased max_iter and learning_rate) what it has to from the training data.

Configuration	Loss	UAS	UASnoPunc	LAS	LASnoPunc	UEM	UEMnoPunc	Root
max_iter = 1001 batch_size = 10000 hidden_size = 200 embedding_size = 50 learning_rate = 0.1 n_Tokens = 48 lam = 1e-8	0.353	67.450	70.852	63.252	66.373	7.588	8.411	48.235
max_iter = 3001 batch_size = 10000 hidden_size = 200 embedding_size = 50 learning_rate = 0.1 n_Tokens = 48 lam = 1e-8	0.230	78.834	80.927	75.588	77.273	17.764	18.941	78.647
max_iter = 1001 batch_size = 10000 hidden_size = 200 embedding_size = 50 learning_rate = 0.3 n_Tokens = 48 lam = 1e-8	0.266	75.890	78.499	72.642	74.902	14.705	15.647	68.764
max_iter = 3001 batch_size = 10000 hidden_size = 200 embedding_size = 50 learning_rate = 0.3 n_Tokens = 48 lam = 1e-8	0.176	82.922	84.767	80.180	81.670	24.882	26.294	86.529
max_iter = 3001 batch_size = 10000 hidden_size = 300 embedding_size = 50 learning_rate = 0.3 n_Tokens = 48 lam = 1e-8	0.174	83.844	85.610	81.010	82.405	26.176	28.117	86.058
max_iter = 5001 batch_size = 10000 hidden_size = 300 embedding_size = 50 learning_rate = 0.3 n_Tokens = 48 lam = 1e-8	0.151	85.886	87.574	83.271	84.621	30.058	32.117	87.117

Table 6: Different Configurations

8 Best Configuration -

I got the best results using the below mentioned configuration and used it for the test data predictions.

Best Configuration	**Loss**	**UAS**	**UASnoPunc**	**LAS**	**LASnoPunc**	**UEM**	**UEMnoPunc**	**Root**
max_iter = 5001 batch_size = 10000 hidden_size = 300 embedding_size = 50 learning_rate = 0.3 n_Tokens = 48 lam = 1e-8	0.151	85.886	87.574	83.271	84.621	30.058	32.117	87.117

Table 7: Best Configuration