0. Default

Following is the result obtained on default configuration. Performance seems decent and error was almost stopped decreasing after 1000 epochs.

CONFIGURATION :

Activation Function	: Cubic
number of Layers and neurons	: 1 layer of 200 neurons
learning rate	: 0.1
Epochs	: 1000

RESULT:

'Average loss at step ', 1000, ': ', 0.35844394236803057

UAS: 68.4846823043
UASnoPunc: 71.5763296219
LAS: 64.3791908667
LASnoPunc: 67.2299779574

UEM: 8.76470588235
UEMnoPunc: 9.41176470588
ROOT: 53.4117647059

1. Multiple layers

Increasing layers can improve performance. But if our model gets too complicated and we don't have enough data to learn all parameters then it will overfit on training data and it may not perform well on test data.

In 3 layers, it seems it's overfitting the train data and its performance on validation data is very less. So, next I tried two layers.

CONFIGURATION :

Activation Function	: tanh and cubic
number of Layers and neurons	: 3 layer of 50-70-50 neurons
learning rate	: 0.1
Epochs	: 4001

RESULT:

('Average loss at step ', 1000, ': ', 0.8395787787437439)
('\nTesting on dev set at step ', 1000)
UAS: 54.4208191041
UASnoPunc: 57.7007856214

LAS: 44.3029139766
LASnoPunc: 46.6342621376

UEM: 3.29411764706
UEMnoPunc: 3.52941176471
ROOT: 35.4117647059

Here, in first configuration of two layers it was not able to learn weights in 1000 epochs. So, In second attempt I tried 4000 iterations and in after that it gave really good accuracy in all aspects.

Configuration :

Activation Function	: tanh and cubic
number of Layers and neurons	: 2 layer of 50 neurons
learning rate	: 0.2
Epochs	: 1000

RESULT:

('\nTesting on dev set at step ', 1000)
UAS: 72.2461799237
UASnoPunc: 75.1201039959
LAS: 67.323079991
LASnoPunc: 69.632057876

UEM: 10.8235294118
UEMnoPunc: 11.7647058824
ROOT: 64.8823529412

Configuration :

Activation Function	: tanh and cubic
number of Layers and neurons	: 2 layer of 50 neurons
learning rate	: 0.2
Epochs	: 4000

RESULT:

('\nTesting on dev set at step ', 4000)
UAS: 81.6262432385
UASnoPunc: 83.6093370259
LAS: 78.4006780168
LASnoPunc: 79.9638275024

UEM: 21.7058823529
UEMnoPunc: 23.4117647059
ROOT: 79.7058823529

---

2. (a). Non-linear activation function

Adding non-linear activation function can improve results and Here are results of different activation functions.

CONFIGURATION :

Activation Function	: tanh
number of Layers and neurons	: 1 layer of 200 neurons
learning rate	: 0.1
Epochs	: 1000

RESULT:

('Average loss at step ', 1000, ': ', 0.839706027507782)

UAS: 48.7349502705
UASnoPunc: 51.6871078958
LAS: 39.9431662388
LASnoPunc: 42.4631210083

UEM: 1.94117647059
UEMnoPunc: 2.11764705882
ROOT: 27.5294117647

CONFIGURATION :

Activation Function	: sigmoid
number of Layers and neurons	: 1 layer of 200 neurons
learning rate	: 0.1
Epochs	: 1000

RESULT:

('Average loss at step ', 1000, ': ', 1.6952998840808868)

UAS: 16.2150709176
UASnoPunc: 16.3030577064
LAS: 8.12373806616
LASnoPunc: 9.11377380885

UEM: 0.647058823529
UEMnoPunc: 0.647058823529
ROOT: 2.58823529412

CONFIGURATION :

Activation Function	: ReLU
number of Layers and neurons	: 1 layer of 200 neurons
learning rate	: 0.1
Epochs	: 1000

RESULT:

relu, learning_rate = 0.1, nrn=200

('Average loss at step ', 900, ': ', 0.8956931811571122)
('Average loss at step ', 1000, ': ', 0.816359366774559)

UAS: 51.5766383329
UASnoPunc: 54.4622166959
LAS: 42.8970261984
LASnoPunc: 45.0404114622

UEM: 3.0
UEMnoPunc: 3.23529411765
ROOT: 36.2352941176

---

2. (a-2) Non-linear activation function with different learning rate and neurons

Observing change in loss with respect epoch reveals that for tanh, relu and sigmoid it's learning slowly and even after 1000 epochs, loss is decreasing. So next I tried higher learning rate and lower number of neurons. Doing so improved accuracy in most of the cases. Following are results for these configuration.

CONFIGURATION :

Activation Function	: tanh
number of Layers and neurons	: 1 layer of 50 neurons
learning rate	: 0.5
Epochs	: 1000

RESULT:

('Average loss at step ', 1000, ': ', 0.38621831327676776)
UAS: 71.8373756761
UASnoPunc: 74.8459842876
LAS: 66.7098736197
LASnoPunc: 69.179901656

UEM: 10.7647058824
UEMnoPunc: 11.2352941176
ROOT: 59.4705882353

CONFIGURATION :

Activation Function	: Sigmoid
number of Layers and neurons	: 1 layer of 50 neurons
learning rate	: 0.1
Epochs	: 1000

RESULT:

('\nTesting on dev set at step ', 800)

UAS: 16.2200563352
UASnoPunc: 16.2861018482
LAS: 6.36139292569
LASnoPunc: 7.21189170859

UEM: 0.588235294118
UEMnoPunc: 0.588235294118
ROOT: 2.58823529412

CONFIGURATION :

Activation Function	: sigmoid
number of Layers and neurons	: 1 layer of 50 neurons
learning rate	: 0.5
Epochs	: 1000

RESULT:

('Average loss at step ', 900, ': ', 0.9099038666486741)
('Average loss at step ', 1000, ': ', 0.8398909497261048)
('\nTesting on dev set at step ', 1000)
UAS: 54.45073161
UASnoPunc: 57.32775674
LAS: 44.4101004562
LASnoPunc: 46.3403605946

UEM: 3.05882352941
UEMnoPunc: 3.17647058824
ROOT: 39.5294117647

CONFIGURATION :

Activation Function	: sigmoid
number of Layers and neurons	: 1 layer of 200 neurons
learning rate	: 0.5
Epochs	: 1000

RESULT:

sigmoid, learning_rate = 0.5, nrn=200
UAS: 54.6277139367
UASnoPunc: 57.4803594642
LAS: 44.1932347882
LASnoPunc: 46.1058045555

UEM: 3.29411764706
UEMnoPunc: 3.52941176471
ROOT: 39.2352941176

CONFIGURATION :

Activation Function	: ReLU
number of Layers and neurons	: 1 layer of 200 neurons
learning rate	: 0.5
Epochs	: 1000

RESULT:

relu, learning_rate = 0.5, nrn=200
('\nTesting on dev set at step ', 1000)
UAS: 71.9994017499
UASnoPunc: 75.1540157124
LAS: 68.1780791186
LASnoPunc: 70.951788843

UEM: 11.1764705882
UEMnoPunc: 11.8823529412
ROOT: 57.1176470588

---

2. (b) parallel hidden layers

For this task, I created 3 separate hidden layer and applied activation function separately. After that, I merged them before calculation loss. After increasing learning rate and changing activation function I was able to get decent accuracy. But even after 1000 iterations, it was leaning weights without overfitting. So, I ran 4000 iterations. Details of this execution is in 2 (c).

CONFIGURATION :

Activation Function	: ReLU
number of Layers and neurons	: 1 layer of 200 neurons
learning rate	: 0.2
Epochs	: 1000

RESULT:

('Average loss at step ', 1000, ': ', 0.3350693023204803)
('\nTesting on dev set at step ', 1000)
UAS: 73.7692250168
UASnoPunc: 76.6715650257
LAS: 69.7858763118
LASnoPunc: 72.3026055502

UEM: 12.7058823529
UEMnoPunc: 13.6470588235
ROOT: 67.6470588235

2. (c) Fixing Word, POS and Dep Embeddings

In the paper, word, POS and dependency embeddings are trainable and changing it to non-trainable can decrease the accuracy. Following is the performance achieved on such configuration.

It was learning slowly. So, I had to increase learning rate but still its performance didn't improve much.

CONFIGURATION :

Activation Function	: ReLU
number of Layers and neurons	: 1 layer of 200 neurons
learning rate	: 0.5
Epochs	: 1000

RESULT:

('Average loss at step ', 1000, ': ', 1.5756002640724183)
('\nTesting on dev set at step ', 1000)
UAS: 23.1298452028
UASnoPunc: 24.5803425083
LAS: 12.3937482863
LASnoPunc: 13.4742553552

UEM: 0.647058823529
UEMnoPunc: 0.647058823529
ROOT: 7.47058823529

2. (d) Best configuration

I tried several different configuration including all mentioned in this report. Following two configuration gave best performance.

Configuration :

Activation Function	: tanh
number of Layers and neurons	: 3 parallel layer of 200 neurons in each
learning rate	: 0.2
Epochs	: 4000

RESULT:

('Average loss at step ', 4000, ': ', 0.19793718069791794)
('\nTesting on dev set at step ', 4000)
UAS: 82.5211257073
UASnoPunc: 84.3610467416
LAS: 79.7990876686
LASnoPunc: 81.3089922568

UEM: 22.9411764706
UEMnoPunc: 24.7058823529
ROOT: 81.5882352941

I'm mentioning following configuration because it's results is quite close to the previous one even though its configuration is totally different.

Configuration :

Activation Function	: tanh and cubic
number of Layers and neurons	: 2 layer of 50 neurons
learning rate	: 0.2
Epochs	: 4000

RESULT:

('\nTesting on dev set at step ', 4000)
UAS: 81.6262432385
UASnoPunc: 83.6093370259
LAS: 78.4006780168
LASnoPunc: 79.9638275024

UEM: 21.7058823529
UEMnoPunc: 23.4117647059
ROOT: 79.7058823529

---

2. (e) Gradient Clipping

We use gradient clipping generally to deal with vanishing gradient and exploding gradient. In the field of NLP, where neural network has to deal with long sequence with dependency, exploding gradient and vanish gradient os quite common.

Result of training Neural network without gradient clipping is dependent on initial weights set. When I set weights with high stddev, it was giving NaN error. But after changing configuration I was able to get following result. Its accuracy is decreased compared to the result with gradient clipping.

CONFIGURATION :

Activation Function	: ReLU
number of Layers and neurons	: 1 layer of 200 neurons
learning rate	: 0.5
Epochs	: 1000

RESULT:

Without clipped gradient
UAS: 71.8149412967
UASnoPunc: 75.0268467756
LAS: 68.0509509684
LASnoPunc: 70.9235290793

UEM: 10.7058823529
UEMnoPunc: 11.1176470588
ROOT: 57.1764705882