Report - HW1

Name: Faizaan Charania
SBU ID: 111463646

Experiments:

1. Experiments with number of hidden layers

Fixed Parameters:

- Optimizer - SGD
- Weight initialization: random normal (mean: 0, std. dev: 0.08)

# hidden layers	Activation function	Epochs	Learning Rate	UAS (%)	UASnoPunc (%)	LAS (%)	LASNoPunc (%)
1	cube	1000	0.1	43.4	45.86	31.49	32.585
2	cube/cube	1000	0.1	34.89	36.97	23.58	24.87
2	cube/cube	1000	0.2	42.82	45.24	30.45	31.88
2	cube/cube	2000	0.2	64.14	66.19	54.63	55.47
3	cube/cube/cube	1000	0.1	16.46	16.28	2.92	3.31
3	relu/relu/relu	1000	0.2	50.56	53.33	40.05	41.84

As we already know, increasing the number of hidden layers increases the expressive power of our neural network, potentially increasing the achievable accuracy on the test set.

While that is true, we also know that more hidden layers lead to more parameters to train, which warranties a more suited learning rate/ more epochs or even a more powerful optimizer.

In our experiments we observe the phenomenons listed above. We get a much better accuracy with 2 hidden layers when we increase the learning rate and the number of epochs. However, training the network with 3 hidden layers proved to be very cumbersome with the limited number of epochs, and a simple optimizer like SGD.

Using ‘relu’ as the activation does offer some boost in accuracy.

It is important to note that, the ideal initialization of weights would be different for a network with one hidden layer and for another with multiple hidden layers. This becomes an important factor because we keep the number of epochs constant and use a simple SGD optimizer.

2.a Experiments with different activation functions

Fixed parameters:

- Epochs: 1000
- Optimizer: SGD
- # of hidden layers: 1
- Learning rate: 0.1
- Hidden layer size: 200

Activation Function	UAS (%)	UASnoPunc (%)	LAS (%)	LASNoPunc (%)	Stddev for normal weight initialization
cube	43.4	45.86	31.49	32.585	0.08
tanh	34.47	36.55	22.42	23.17	0.08
relu	30.37	32.43	19.07	20.39	0.08
sigmoid	19.82	20.68	7.43	8.28	0.08
sigmoid	33.76	35.81	21.3	22.32	0.8
cube	30.27	32.36	17.72	18.55	0.8

While all these activation functions add non-linearity to our neural network, it is observed that the initial weights of the hidden layer have a significant impact on the accuracy after 1000 epochs. The best result was observed when we use the cube activation function with weights initialized from a random normal distribution with mean =0 and stddev = 0.08.

A potential reason behind that could be the fact that the cube function captures the relations between the different words very well, because of its internal mathematical formula.

We also observe that sigmoid performs much better when the weight initialization is mean =0 and stddev = 0.8. This setting reduces the accuracy given by the cube function. Hence we conclude that a more comprehensive search would be required to find the best parameters for the model.

2.b Parallel hidden layers for different types of embeddings

Fixed parameters:

- Epochs: 1000
- Optimizer: SGD
- Activation function: Cube

Learning rate	UAS (%)	UASnoPunc (%)	LAS (%)	LASNoPunc (%)
0.1	48.24	50.79	35.55	36.6
0.2	52.82	55.67	42.595	24.78

For this current experiment, we have kept three different hidden layers of size 100 for each kinds of embeddings (word, pos and dep). These 3 hidden layers are then concatenated and connected to the output layer.

It is observed that this configuration gives us a better accuracy compared to when we use one bigger hidden layer that connects to all the embeddings. This intuitively makes sense because the information and trends that can be learnt from word embeddings need not be the same as those learnt from pos or dep embeddings. Giving all these embeddings separate hidden layers helps us achieve better results.

NOTE: 2 (c). Written after 2 (d).

2.d Experiments for Best Configuration

Top 6 models

# of hidden layers	Activation function	Epochs	Learning Rate	Optimizer	UAS (%)	UASnoPunc (%)	LAS (%)	LASNoPunc (%)
3	cube/cube/cube	3750	0.001	adam	83.18	84.96	79.57	80.89
3	relu/relu/relu	3000	0.001	adam	82.88	84.67	79	80.41
Parallel + concat	cube/cube/cube	2000	0.002	adam	83.69	85.53	80.49	81.91
Parallel + concat	cube/cube/cube	2000	0.3	adagrad	78.84	80.68	73.76	74.9
Parallel + concat	relu/relu/relu	1000	0.003	adam	84.246	86.17	81.277	82.8
Parallel + concat	relu/relu/relu	2000	0.002	adam	83.11	85.05	80.18	81.74
Parallel + multiply	relu/relu/relu	500	0.003	adam	80.79	83.18	77.12	79.02

Note:

- Parallel + concat -> parallel hidden layers for embeddings and they are merged by concatenating them to form a bigger layer
- Parallel + multiply -> parallel hidden layers for embeddings and they are merged by pointwise multiplication of the elements of the layers

Best configuration complete result:

UAS (%)	UASnoPunc (%)	LAS (%)	LASnoPunc (%)	UEM (%)	UEMnoPunc (%)	ROOT (%)
84.246	86.17	81.277	82.8	27.82	29.88	83.0

We explored multiple configurations to find out the best performing model, the following observations were made in the process

1. When we train certain models from the earlier experiments for more number of epochs, they start showing much better results than previously reported for 1000 epochs.
2. If we change the optimizer from SGD to either Adagrad (used in the paper) or Adam, the loss reduces much faster and the accuracy increases at the same time.
3. In the last model ‘parallel + multiply’, we observe a very high accuracy at just 500 epochs (but saturates later), this tells us that how we merge the parallel hidden layers also makes an impact on our final result

2.c Effect of fixing word, POS and Dep Embeddings

- To fix the embeddings, we change the trainable attribute of the embeddings layer to false.
- Initially, POS and Dep embeddings were generated using randomly and were trained with the network, in this experiment, we make one-hot encoder labels to generate POS and Dep embeddings.
- Number of unique POS and Dep values are less than 50, therefore we can accomodate their one-hot encoded embeddings in our current embedding size of 50.

For this experiment we use the configuration of the best performing model and the result is as follows:

Embeddings	**UAS (%)**	**UASnoPunc (%)**	**LAS (%)**	**LASnoPunc (%)**	**UEM (%)**	**UEMnoPunc (%)**	**ROOT (%)**
Trained	84.246	86.17	81.277	82.8	27.82	29.88	83.0
Fixed	44.51	46.37	34.69	36.03	4.24	4.47	35.76

Because of how we initialize the embeddings, we have a very sparse embedding vector for POS tags and Dep labels. As a result of this, we don't have enough non-zero nodes in the network at any point in time, leading no gradient updates for weights of those nodes and hence hindering the training process.

From this experiment we conclude that, keeping a sparsely initialised embedding vector 'non-trainable' will affect the training process and hence the resulting accuracy of our model.

2.e Gradient Clipping

Gradient clipping is used to clip the gradient when it's absolute value grows too big. This problem is famously documented as the exploding gradient problem. Exploding gradients lead to very large updates to the weights of the network, making the training process unstable as the large updates could make the weights oscillate around the optimal value.

By adding gradient clipping to our network, we avoid the problem of exploding gradients by restricting the magnitude of our gradients, thereby keeping the weight updates in check and ensuring that we move towards the optimal value.

Effect on best model

The current proposed best model uses the Adam Optimizer with a learning rate of 0.003, since the learning rate is so small, we don't suffer from exploding gradients and weights becoming NaN as the effect of gradients on the weights is not that high.

Effect on default configuration model

Since in this configuration we use an SGD with a much higher learning rate, we get NaN values when the first result is displayed at epoch 100, the weights could've become NaN much before that.