CS7327-033-M01 Assignment 1

Liong Khai Jiet (120033990010)

April 7, 2021

Problem 1.

Given that the output of neuron in a multilayer perception is:

$$x_{kj} = f\left(\sum_{i=1}^{N_{k-1}} (u_{kji}x_{k-1,i}^2 + v_{kji}x_{k-1,i}) + b_{kj}\right) \tag{1}$$

where f is the sigmoid activation, given by the equation

$$f(x) = \frac{1}{1+\exp(-x)}$$

and *net* is used to defined the local gradient in unit j as follows:

$$net_j = \sum_{i=1}^{N_{k-1}} (u_{kji}x_{k-1,i}^2 + v_{kji}x_{k-1,i}) + b_{kj}$$

1. Before deriving the online learning and batch learning algorithm, we need the following definitions:

   (a) The instantaneous error at unit j is defined as:

   $$\varepsilon_j(n) = \frac{1}{2}e_j^2(n) \tag{2}$$

   (b) The total neural network error can be defined as:

   $$\varepsilon(n) = \sum_{j\in p}\varepsilon_j(n) = \frac{1}{2}\sum_{j\in p}e^2(n) \tag{3}$$

   where p is the number of output units.

   (c) If the training data has N samples, the average error can be defined as:

   $$\varepsilon_{av}(n) = \frac{1}{N}\sum_{n=1}^{N}\varepsilon(n) = \frac{1}{2N}\sum_{n=1}^{N}\sum_{j\in p}e_j^2(n) \tag{4}$$

2. The following are the derivation of the back-propagation algorithm for

2.1 Online Learning

We need to compute the instantaneous error ε w.r.t to each the weights u, v and bias b of the neuron unit in layer K, using the chain rule, we can express the local gradient as

(a) For weight u

$$\frac{\partial \varepsilon(n)}{\partial u_{ji}(n)} = \frac{\partial \varepsilon(n)}{\partial e_{kj}(n)} \frac{\partial e_{kj}(n)}{\partial f_j(n)} \frac{\partial f_j(n)}{\partial net_j(n)} \frac{\partial net_j(n)}{\partial u_{ji}(n)} \tag{5}$$

(b) for weight v

$$\frac{\partial \varepsilon(n)}{\partial v_{ji}(n)} = \frac{\partial \varepsilon(n)}{\partial e_{kj}(n)} \frac{\partial e_{kj}(n)}{\partial f_j(n)} \frac{\partial f_j(n)}{\partial net_j(n)} \frac{\partial net_j(n)}{\partial v_{ji}(n)} \tag{6}$$

(c) for bias b

$$\frac{\partial \varepsilon(n)}{\partial b_{ji}(n)} = \frac{\partial \varepsilon(n)}{\partial e_{kj}(n)} \frac{\partial e_{kj}(n)}{\partial f_j(n)} \frac{\partial f_j(n)}{\partial net_j(n)} \frac{\partial net_j(n)}{\partial b_{ji}(n)} \tag{7}$$

Because the first few chains of the equation are similar, so we can first compute the partial derivative $\frac{\partial \varepsilon(n)}{\partial net_j(n)}$ for the equation 5, 6 and 7.

(a) Differentiating $\varepsilon(n)$ w.r.t $e_{kj}(n)$ in equation 2, we get the first chain in equation:

$$\frac{\partial \varepsilon(n)}{\partial e_{kj}(n)} = e_{kj}(n) \tag{8}$$

(b) Taking the derivative of error signal $e_j(n)$ w.r.t. $f_j(n)$, we get the second chain in the equation:

$$e_j(n) = \hat{y}_j - f_j(n) \frac{\partial e_j(n)}{\partial f_j(n)} = -1 \tag{9}$$

(c) Taking the derivative of the activation function in equation , we get the third chain:

$$\frac{\partial f_j}{\partial net_j} = \frac{1}{1 + \exp(-net_j)} \cdot \frac{-\exp(-net_j)}{1 + \exp(-net_j)} \tag{10}$$

$$= f(net_j)(1 - f(net_j)) \tag{11}$$

However, for convenience, the derivative of the activation function is defined as $f'(\cdot)$

(d) Combining the steps above, we get the formula of local gradient as follows:

$$\delta_{kj}(n) = \frac{\partial \varepsilon(n)}{\partial net_j(n)} = -e_j(n) f'_j(net_j(n)) \tag{12}$$

(e) For the last chain, we get the derivatives of the weights and bias respectively:

$$\frac{\partial net_j}{\partial u_{ji}} = x^2_{k-1,i}(n) \tag{13}$$

$$\frac{\partial net_j}{\partial v_{ji}} = x_{k-1,i}(n) \tag{14}$$

$$\frac{\partial net_j}{\partial b_{ji}} = 1 \tag{15}$$

(f) Then, using the chain rule with equation 12, 13, 14 and 15 respectively. We can compute:

$$\frac{\partial \varepsilon(n)}{\partial u_{ji}(n)} = -e_j(n) f'_j(net_j(n)) x^2_{k-1,i}(n) \tag{16}$$

$$\frac{\partial \varepsilon(n)}{\partial v_{ji}(n)} = -e_j(n) f'_j(net_j(n)) x_{k-1,i}(n) \tag{17}$$

$$\frac{\partial \varepsilon(n)}{\partial b_{ji}(n)} = -e_j(n) f'_j(net_j(n)) \tag{18}$$

3. So, the weight correction method can be defined as:

$$\Delta u_{ji}(n) = \alpha \frac{\partial \varepsilon(n)}{\partial u_{ji}(n)} = -\alpha e_j(n) f'_j(net_j(n)) x^2_{k-1,i}(n) \tag{19}$$

$$= \alpha \delta_{kj}(n) x^2_{k-1,i}(n) \tag{20}$$

$$\Delta v_{ji}(n) = \alpha \frac{\partial \varepsilon(n)}{\partial v_{ji}(n)} = -\alpha e_j(n) f'_j(net_j(n)) x_{k-1,i}(n) \tag{21}$$

$$= \alpha \delta_{kj}(n) x_{k-1,i}(n) \tag{22}$$

$$\Delta b_{ji}(n) = \alpha \frac{\partial \varepsilon(n)}{\partial b_{ji}(n)} = -\alpha e_j(n) f'_j(net_j(n)) \tag{23}$$

$$= \alpha \delta_{kj} \tag{24}$$

where α is the learning rate and the negative sign is the direction of gradient descent.

4. (a) The above formula is for neuron j as a output unit.

(b) If the neuron j is a hidden unit, we would need to change the formula of local gradient in equation 12 as follows:

$$\delta_{kj}(n) = \frac{\partial \varepsilon(n)}{\partial f_{kj}(n)} \frac{\partial f_{kj}(n)}{\partial net_j(n)} = -\frac{\partial \varepsilon(n)}{\partial f_{kj}(n)} f'_j(net_j(n)) \tag{25}$$

$$\varepsilon(n) = \frac{1}{2} \sum_{1 \le i \le N_{k+1}} e^2_{k+1,i}(n) \tag{26}$$

Then taking the derivative of equation 26 w.r.t $x_{kj}(n)$, we get:

$$\frac{\partial \varepsilon(n)}{\partial x_{kj}(n)} = \sum_i e_{k+1,i}(n) \frac{\partial e_{k+1,i}(n)}{\partial x_{kj}(n)} \tag{27}$$

where $e_{k+1,i}(n) = \hat{y}_{k+1,i}(n) - f(net_{k+1,i}(n))$, which yields:

$$\frac{\partial e_{k+1,i}(n)}{\partial net_{k+1,i}(n)} = -f'(net_{k+1,i}(n)) \tag{28}$$

Taking the derivative of $net_{k+1,i}(n)$ w.r.t. $x_{kj}(n)$, we get:

$$\frac{\partial net_{k+1,i}(n)}{\partial x_{kj}(n)} = 2u_{k+1,ij}x_{kj}(n) + v_{k+1,ij} \tag{29}$$

Inserting equation 28, 29 into 25, we get:

$$\frac{\partial \varepsilon(n)}{\partial x_{kj}(n)} = -\sum_i e_{k+1,h}(n)f'(net_{k+1,h}(n))(2u_{k+1,ij}x_{kj}(n) + v_{k+1,ij}) \tag{30}$$

$$= -\sum_i \delta_{k+1,h}(n)(2u_{k+1,ij}x_{kj}(n) + v_{k+1,ij}) \tag{31}$$

Finally, inserting equation 31 into equation 25, we get:

$$\delta_{kj}(n) = f'(net_{kj}(n))\sum_i \delta_{k+1,i}(n)(2u_{k+1,ij}x_{kj}(n) + v_{k+1,ij}) \tag{32}$$

for the weights u, v and bias b.

2.2 Batch Learning

Batch learning has similar steps for derivation except for:

1. Loss function needs to be modified to use the average error rate of N training samples as follows:

$$\varepsilon_{av}(N) = \frac{1}{N}\sum_{n=1}^{N} E(n) = \frac{1}{2N}\sum_{n=1}^{N}\sum_{j\in C} e_j^2(n) \tag{33}$$

followed by the update to the weights correction method as follows:

$$\Delta w_{kji} = \alpha\frac{\partial \varepsilon_{av}(N)}{\partial w_{kji}} = -\frac{\alpha}{N}\sum_{n=1}^{N} e_{kj}(n)\frac{\partial e_{kj}(n)}{\partial w_{kji}} \tag{34}$$

So, the weight correction formula for u, v and b are as follows:

$$\Delta u_{kji} = \alpha\frac{1}{N}\sum_{n=1}^{N}\delta_{kj}(n)x_{k-1,i}^2(n) \tag{35}$$

$$\Delta v_{kji} = \alpha\frac{1}{N}\sum_{n=1}^{N}\delta_{kj}(n)x_{k-1,i}(n) \tag{36}$$

$$\Delta b_{ji} = \alpha\frac{1}{N}\sum_{n=1}^{N}\delta_{ji}(n) \tag{37}$$

Problem 2.

The purpose of this experiment is to add quadratic terms to the vanilla Multi layer Perceptron to improve its performance and generalization ability.

2.3 Setting

The experiment setting is initially run with the 12 unit as specified in this study [1]. After experimenting with higher number of hidden units, 50 units shows better performance without increasing the training times.

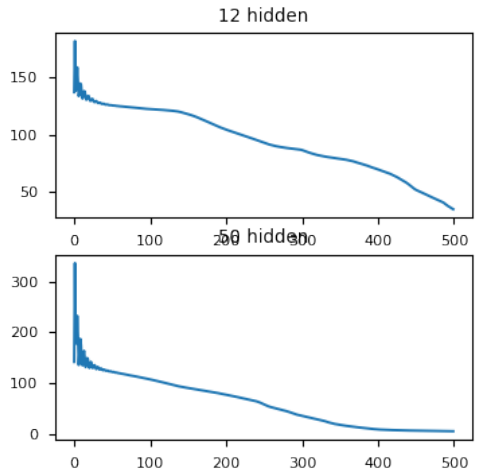

Fig. 1. Training loss of 12 hidden units and 50 hidden units model

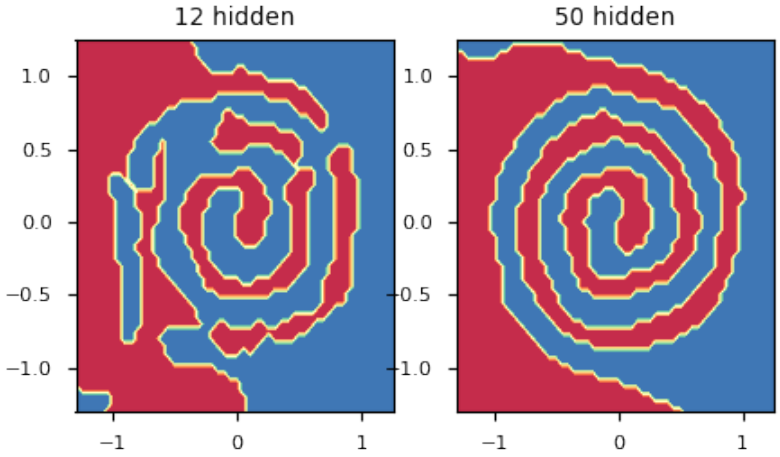

Fig. 2. Decision boundary of 12 hidden units and 50 hidden units model

Besides, the weight of hidden units are randomly initialized with the the normal distribution $\mathcal{N}(\mu = -15, \sigma = 20)$. Adam optimizer with weight decay rate (1e-5) is used in this experiment.

The training data is normalized to have $\mu = 0$ and $\sigma = 0.5$ during the data preprocessing process.

2.4 Result

The experiment is run with the skorch and pytorch framework for 500 epochs. The cross validation results and best model results are tabulated as follows: The experiment results showed that model

	Model		
	A	B	C
Learning Rate	0.001	0.1	3
epoch	2000	400	2000
training loss	0.4	0.01	0.3
best train score (%)	0.804	1	0.742
cv mean train score (%)	56	100	67
train time (ms)	625	521	518

A (α=0.001) fails to converge after 500 epochs. In the other hand, model C shows the learning rate is too high as the training error oscillates between the 1 and 50 epochs. Model B shows overall smooth curve in the learning rate and the decision boundary shows the quadratic mlqp manage to approximate the function to solve the two spiral problem.

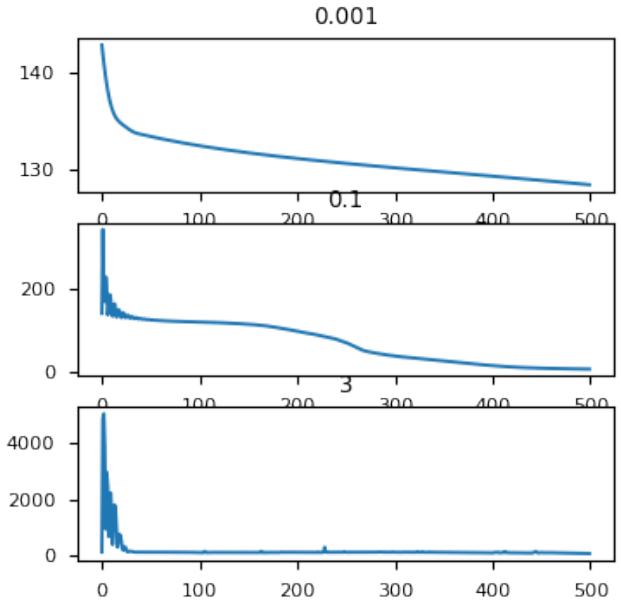

Fig. 3. Comparison of learning rate and training loss

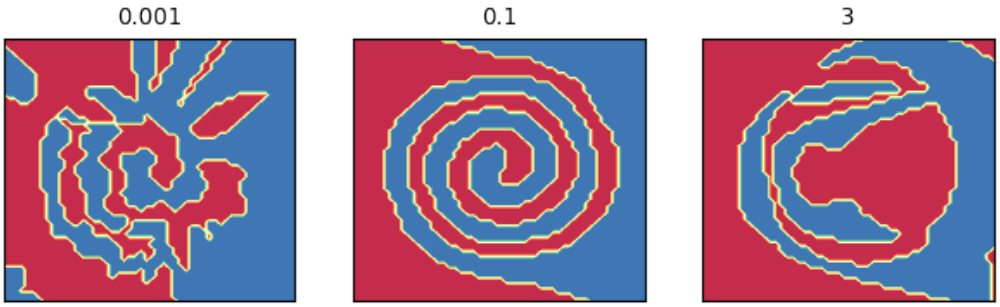

Fig. 4. Comparison of learning rate and decision boundaries

Problem 3.

3.5 Random partition

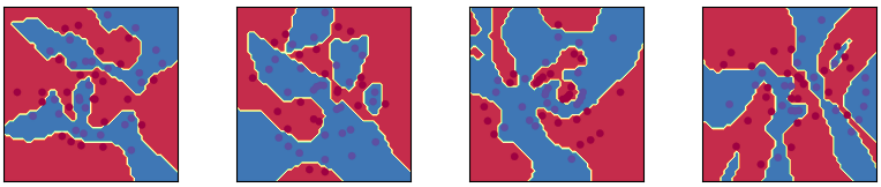

Fig. 5. Randomly split dataset (4 partition)

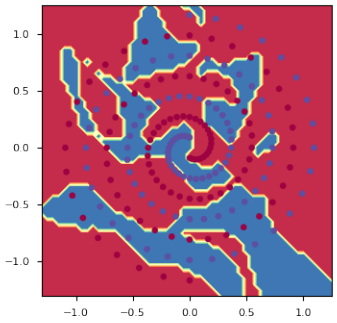

Fig. 6. Min max modular neural network decision boundary

3.6 Partition with prior knowledge

The spiral are divided according to the xy-coordinates. For example, point(1,1) is at the top right quadrant, point(-1,-1) is at the bottom left quadrant, point(1,-1) is at the bottom right quadrant and point(-1,1) is at the top left quadrant.

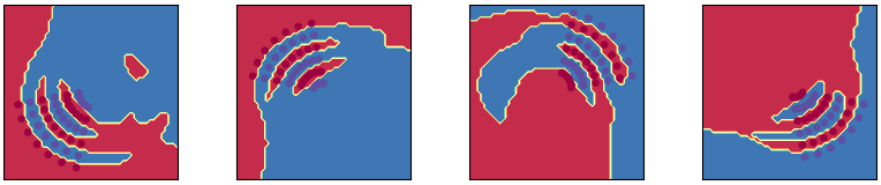

Fig. 7. Min max modular neural network decision boundary

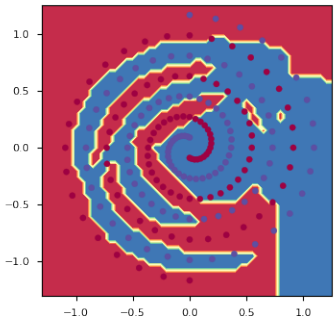

Fig. 8. Min max modular neural network decision boundary

References

[1] BAO-LiANG Lu, Yan Bai, Hajime Kita, and Yoshikazu Nishikawa. An efficient multilayer quadratic perceptron for pattern classification and function approximation. In *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*, volume 2, pages 1385–1388. IEEE, 1993.