hw2

Hongyi Guo 516030910306

March 14, 2019

1

Suppose we have N iterations in one episode, there are M layers in the network. Similar to the slides, the output of each neuron in a multilayer quadratic perceptron (MLQP) network is

$$x_{kj} = f(z_{kj}) \tag{1}$$

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

where both u_{kji} and v_{kji} are the weights connecting the ith unit in the layer $k-1$ to the jth unit in the layer k, b_{kj} is the bias of the jth unit in the layer k, N_k is the number of units in the k $(1 \leq k \leq M)$, and $f(.)$ is hte sigmoidal activation function.
The error signal at the output of neuron j at iteration n is defined by

$$e_j(n) = d_j(n) - x_{Mj}(n), j \in \{1, \ldots, N_M\} \tag{3}$$

The instantaneous value of the error energy for neuron j is defined by $e_j^2(n)/2$. The total instantaneous error energy ε_n for all the neurons in the output layer is therefore

$$\varepsilon(n) = \frac{1}{2}\sum_{j=1}^{N_M} e_j^2(n) \tag{4}$$

Then the average squared error energy is

$$\varepsilon_{av} = \frac{1}{N}\sum_{n=1}^{N} \varepsilon(n) \tag{5}$$

Define the local gradient for neuron j in layer k at iteration n as

$$\delta_{kj}(n) = -\frac{\partial \varepsilon(n)}{\partial z_{kj}(n)} \tag{6}$$

For the neurons in last layer M, we can directly compute their local gradients with chain rule of calculus.

$$\delta_{Mj}(n) = -\frac{\partial \varepsilon(n)}{\partial z_{Mj}(n)} = -\frac{\partial \varepsilon(n)}{\partial e_j(n)}\frac{\partial e_j(n)}{\partial x_{Mj}(n)}\frac{\partial x_{Mj}(n)}{\partial z_{Mj}(n)} \tag{7}$$

Differentiating both sides of Eq.(4) with respect to $e_j(n)$, we get

$$\frac{\partial \varepsilon_n}{\partial e_j(n)} = e_j(n) \tag{8}$$

Differentiating both sides of Eq.(3) with respect to $x_{Mj}(n)$, we get

$$\frac{\partial e_j(n)}{\partial x_{Mj}(n)} = -1 \tag{9}$$

Differentiating both sides of Eq.(1) with respect to $z_{kj}(n)$, we get

$$\frac{\partial x_{kj}(n)}{\partial z_{kj}(n)} = f'(z_{kj}(n)) \tag{10}$$

Insert Eq.(8), Eq.(9) and Eq.(10) into Eq.(7) and we get

$$\delta_{Mj}(n) = e_j(n) f'(z_{Mj}(n)) = (d_j(n) - x_{Mj}(n)) f'(z_{Mj}(n)) \tag{11}$$

For neuron i in layer $k \in \{1, 2, \ldots, M-1\}$, we compute the local gradient as

$$\delta_{ki}(n) = -\frac{\partial \varepsilon(n)}{\partial z_{ki}(n)} = -\sum_{j=1}^{N_{k+1}} \left[\frac{\partial \varepsilon(n)}{\partial z_{k+1,j}(n)} \frac{\partial z_{k+1,j}(n)}{\partial x_{ki}(n)} \right] \frac{\partial x_{ki}(n)}{\partial z_{ki}(n)} \tag{12}$$

Differentiating both sides of Eq.(2) with respect to $x_{k-1,i}(n)$, we get

$$\frac{\partial z_{kj}(n)}{\partial x_{k-1,i}(n)} = 2u_{kji} x_{k-1,i}(n) + v_{kji} \tag{13}$$

Insert Eq.(6), Eq.(13) and Eq.(10) into Eq.(12) and we get

$$\delta_{ki}(n) = \sum_{j=1}^{N_{k+1}} \left[\delta_{k+1,j}(n) (2u_{k+1,ji} x_{ki}(n) + v_{k+1,ji}) \right] f'(z_{ki}(n)) \tag{14}$$

In conclusion, the local gradient of neuron j in layer k can be computed as

$$\delta_{kj}(n) = \begin{cases} (d_j(n) - x_{kj}(n)) f'(z_{kj}(n)) & k = M \\ \sum_{i=1}^{N_{k+1}} \left[\delta_{k+1,i}(n) (2u_{k+1,ij} x_{kj}(n) + v_{k+1,ij}) \right] f'(z_{kj}(n)) & k \in \{1, \ldots, M-1\} \end{cases} \tag{15}$$

where

$$f'(x) = sigmoid'(x) = \frac{e^{-x}}{(1 + e^{-x})^2}$$

Finally, we can compute the gradients for u_{kji}, v_{kji} and b_{kj}.
For sequential model (online mode), we update the network in every step, $\mathcal{L}_{\text{seq}} = \varepsilon(n)$, we can compute the gradients for neuron j in layer $k \in \{1, \ldots, M\}$ as

$$\nabla_{u_{kji}} \mathcal{L}_{\text{seq}} = \frac{\partial \varepsilon(n)}{\partial u_{kji}} = \frac{\partial \varepsilon(n)}{\partial z_{kj}(n)} \frac{\partial z_{kj}(n)}{\partial u_{kji}} = \delta_{kj}(n) x_{k-1,i}^2 \tag{16}$$

$$\nabla_{v_{kji}}\mathcal{L}_{\text{seq}} = \frac{\partial\varepsilon(n)}{\partial v_{kji}} = \frac{\partial\varepsilon(n)}{\partial z_{kj}(n)}\frac{\partial z_{kj}(n)}{\partial v_{kji}} = \delta_{kj}(n)x_{k-1,i} \tag{17}$$

$$\nabla_{b_{kj}}\mathcal{L}_{\text{seq}} = \frac{\partial\varepsilon(n)}{\partial b_{kj}} = \frac{\partial\varepsilon(n)}{\partial z_{kj}(n)}\frac{\partial z_{kj}(n)}{\partial b_{kj}} = \delta_{kj}(n) \tag{18}$$

For batch mode, the cost function is $\mathcal{L}_{\text{batch}} = \varepsilon_{av}$, we can compute the gradients for neuron j in layer $k \in \{1, \ldots, M\}$ as

$$\nabla_{u_{kji}}\mathcal{L}_{\text{batch}} = \frac{\partial\varepsilon_{av}}{\partial u_{kji}} = \frac{\partial\frac{1}{N}\sum_{n=1}^{N}\varepsilon(n)}{\partial u_{kji}} = \frac{1}{N}\sum_{n=1}^{N}\delta_{kj}(n)x_{k-1,i}^2 \tag{19}$$

$$\nabla_{v_{kji}}\mathcal{L}_{\text{batch}} = \frac{\partial\varepsilon_{av}}{\partial v_{kji}} = \frac{\partial\frac{1}{N}\sum_{n=1}^{N}\varepsilon(n)}{\partial v_{kji}} = \frac{1}{N}\sum_{n=1}^{N}\delta_{kj}(n)x_{k-1,i} \tag{20}$$

$$\nabla_{b_{kj}}\mathcal{L}_{\text{batch}} = \frac{\partial\varepsilon_{av}}{\partial b_{kj}} = \frac{\partial\frac{1}{N}\sum_{n=1}^{N}\varepsilon(n)}{\partial b_{kj}} = \frac{1}{N}\sum_{n=1}^{N}\delta_{kj}(n) \tag{21}$$

2

My tensorflow implementation is as in `two_spiral.py`. In my implementation, the model is considered converged when the loss is less than 0.001. To accelerate converging, labels are transformed from $\{0, 1\}$ to $\{0.1, 0.9\}$.
I find that using normal initializer for weights makes converging faster than uniform initializer.

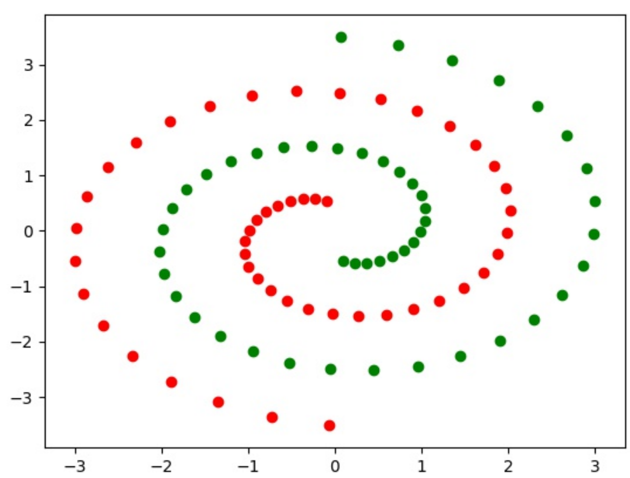

Figure 1: Results on test set. Red dots represent positive samples and green dots represent negative samples.

3

The number of iterations until convergence, time passed till convergence and response plots under different learning rates (0.1, 0.01, 0.001) are as following.

lr	0.1	0.01	0.001
#Iterations	600	7400	66800
Time(seconds)	17.787	214.149	1935.307
Response plots