CS446: Machine Learning **Spring 2017**

Problem Set 5

Zhuo Li(zhuol2) *Handed In: April 10, 2017*

1. **Neural Networks**

(a) For each training example d, we know that we need to update weights on the opposite direction of the gradient (gradient descent). So we have

$$\Delta\omega_{ij} = -R\frac{\partial E_d}{\partial \omega_{ij}} \tag{1}$$

ω_{ij} can only influence the output through net_j, such that

$$\text{net}_j = \sum \omega_{ij} x_{ij} \tag{2}$$

where x_{ij} is from the previous layer of unit j. From (1)(2) we have

$$\frac{\partial E_d}{\partial \omega_{ij}} = \frac{\partial E_d}{\partial \text{net}_j}\frac{\partial \text{net}_j}{\partial \omega_{ij}} = \frac{\partial E_d}{\partial \text{net}_j} x_{ij} \tag{3}$$

$$\Delta\omega_{ij} = -R\frac{\partial E_d}{\partial \text{net}_j} x_{ij} \tag{4}$$

Then we consider two cases:

i. **Unit j is the output unit**. So net_j can only influence the rest of the network through o_j. So

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial \text{net}_j} \tag{5}$$

We also have

$$E_d = \frac{1}{2}\sum_{k\in K}(t_k - o_k)^2 \tag{6}$$

$$o_j = max(0, \text{net}_j) \tag{7}$$

Thus

$$\frac{\partial o_j}{\partial \text{net}_j} = \begin{cases} 0 & \text{if net}_j \le 0 \\ 1 & \text{otherwise} \end{cases} \tag{8}$$

Then

$$\begin{aligned} \frac{\partial E_d}{\partial \text{net}_j} &= \frac{1}{2}2(t_j - o_j)\frac{\partial (t_j - o_j)}{\partial \text{net}_j} \\ &= \begin{cases} 0 & \text{if net}_j \le 0 \\ -o_j(t_j - o_j) & \text{otherwise} \end{cases} \end{aligned} \tag{9}$$

So

$$\begin{aligned}\Delta\omega_{ij} &= -R\frac{\partial E_d}{\partial \text{net}_j}x_{ij} = R\delta_j x_{ij} \\ &= \begin{cases} 0 & \text{if net}_j \le 0 \\ R(t_j - o_j)o_j x_{ij} & \text{otherwise} \end{cases}\end{aligned} \tag{10}$$

where

$$\delta_j \equiv \begin{cases} 0 & \text{if net}_j \le 0 \\ (t_j - o_j)o_j & \text{otherwise} \end{cases} \tag{11}$$

ii. **Unit j is the hidden unit**. So net_j can influence the network only through downstream(j). Then

$$\begin{aligned}\frac{\partial E_d}{\partial \text{net}_j} &= \sum_{k\in\text{downstream}(j)} \frac{\partial E_d}{\partial \text{net}_k}\frac{\partial \text{net}_k}{\partial \text{net}_j} \\ &= \sum_{k\in\text{downstream}(j)} -\delta_k \frac{\partial \text{net}_k}{\partial \text{net}_j} \\ &= \sum_{k\in\text{downstream}(j)} -\delta_k \frac{\partial \text{net}_k}{\partial o_j}\frac{\partial o_j}{\partial \text{net}_j} \\ &= \sum_{k\in\text{downstream}(j)} -\delta_k \omega_{jk}\frac{\partial o_j}{\partial \text{net}_j}\end{aligned} \tag{12}$$

Rember what we have in (8)

$$\frac{\partial E_d}{\partial \text{net}_j} = \begin{cases} 0 & \text{if net}_j \le 0 \\ \sum_{k\in\text{downstream}(j)} -\delta_k\omega_{jk} & \text{otherwise} \end{cases} \tag{13}$$

So

$$\begin{aligned}\Delta\omega_{ij} &= -R\frac{\partial E_d}{\partial \text{net}_j}x_{ij} = R\delta_j x_{ij} \\ &= \begin{cases} 0 & \text{if net}_j \le 0 \\ Rx_{ij}\sum_{k\in\text{downstream}(j)} \delta_k\omega_{jk} & \text{otherwise} \end{cases}\end{aligned} \tag{14}$$

where

$$\delta_j \equiv \begin{cases} 0 & \text{if net}_j \le 0 \\ \sum_{k\in\text{downstream}(j)} \delta_k\omega_{jk} & \text{otherwise} \end{cases} \tag{15}$$

(b) i. From (6) we have the gradient of squared loss with respect to output o_k is

$$\frac{\partial E_d}{\partial o_k} = (o_k - t_k)$$

So we can have the following codes:

```
def squared_loss_gradient (output_activations, y):
    #IMPLEMENT THIS
    (m, n) = output_activations.shape
    gradient = np.zeros((m, n))
    for i in range(m):
        for j in range(n):
            gradient[i, j] = output_activations[i, j] -
                y[i, j]
    return gradient
#endDef
```

And (8) is the formula for the derivative of the rectifier function, then

```
def relu_derivative (z):
    #IMPLEMENT THIS!
    (m, n) = z.shape
    derivative = np.zeros((m, n))
    for i in range(m):
        for j in range(n):
            if z[i, j] > 0:
                derivative[i, j] = 1
            else:
                derivative[i, j] = 0
    return derivative
#endDef
```

ii. We can run the *parameter_tuning.py*, and get the following results for **circles**:

```
==================== circles ====================
-- size = 10 , R = 0.1 , func = relu , node = 10 --
<Average Accuracy> 100.0
-- size = 10 , R = 0.1 , func = relu , node = 50 --
<Average Accuracy> 100.0
-- size = 10 , R = 0.1 , func = tanh , node = 10 --
<Average Accuracy> 100.0
-- size = 10 , R = 0.1 , func = tanh , node = 50 --
<Average Accuracy> 100.0
-- size = 10 , R = 0.01 , func = relu , node = 10 --
<Average Accuracy> 100.0
-- size = 10 , R = 0.01 , func = relu , node = 50 --
<Average Accuracy> 100.0
-- size = 10 , R = 0.01 , func = tanh , node = 10 --
<Average Accuracy> 52.7647714604
-- size = 10 , R = 0.01 , func = tanh , node = 50 --
<Average Accuracy> 50.3448001274
-- size = 50 , R = 0.1 , func = relu , node = 10 --
```

```
<Average Accuracy>  100.0
-- size = 50 , R = 0.1 , func = relu , node = 50 --
<Average Accuracy>  100.0
-- size = 50 , R = 0.1 , func = tanh , node = 10 --
<Average Accuracy>  66.0081223125
-- size = 50 , R = 0.1 , func = tanh , node = 50 --
<Average Accuracy>  57.1444497531
-- size = 50 , R = 0.01 , func = relu , node = 10 --
<Average Accuracy>  74.1025641026
-- size = 50 , R = 0.01 , func = relu , node = 50 --
<Average Accuracy>  99.6273291925
-- size = 50 , R = 0.01 , func = tanh , node = 10 --
<Average Accuracy>  51.9190953974
-- size = 50 , R = 0.01 , func = tanh , node = 50 --
<Average Accuracy>  51.2581621277
-- size = 100 , R = 0.1 , func = relu , node = 10 --
<Average Accuracy>  100.0
-- size = 100 , R = 0.1 , func = relu , node = 50 --
<Average Accuracy>  100.0
-- size = 100 , R = 0.1 , func = tanh , node = 10 --
<Average Accuracy>  52.0074852684
-- size = 100 , R = 0.1 , func = tanh , node = 50 --
<Average Accuracy>  51.8633540373
-- size = 100 , R = 0.01 , func = relu , node = 10 --
<Average Accuracy>  65.0947603122
-- size = 100 , R = 0.01 , func = relu , node = 50 --
<Average Accuracy>  74.7356266921
-- size = 100 , R = 0.01 , func = tanh , node = 10 --
<Average Accuracy>  52.668418538
-- size = 100 , R = 0.01 , func = tanh , node = 50 --
<Average Accuracy>  51.2501990763
```

Then we have the optimal parameters for **circles**:

```
[Optimal Batch Size] 10
[Optimal Learning Rate] 0.1
[Optimal Activation Function] relu
[Optimal Hidden Layer Width] 10
[Maximum Accuracy] 100.0%
```

And results for **mnist**:

```
==================== mnist ====================
-- size = 10 , R = 0.1 , func = relu , node = 10 --
<Average Accuracy>  95.9688541337
-- size = 10 , R = 0.1 , func = relu , node = 50 --
<Average Accuracy>  87.5582991689
-- size = 10 , R = 0.1 , func = tanh , node = 10 --
```

```
<Average Accuracy>  96.9621801688
-- size = 10 , R = 0.1 , func = tanh , node = 50  --
<Average Accuracy>  96.9038261068
-- size = 10 , R = 0.01 , func = relu , node = 10  --
<Average Accuracy>  96.6199395025
-- size = 10 , R = 0.01 , func = relu , node = 50  --
<Average Accuracy>  96.4864811173
-- size = 10 , R = 0.01 , func = tanh , node = 10  --
<Average Accuracy>  96.8787634501
-- size = 10 , R = 0.01 , func = tanh , node = 50  --
<Average Accuracy>  96.1944075876
-- size = 50 , R = 0.1 , func = relu , node = 10  --
<Average Accuracy>  96.2193238624
-- size = 50 , R = 0.1 , func = relu , node = 50  --
<Average Accuracy>  96.4613139022
-- size = 50 , R = 0.1 , func = tanh , node = 10  --
<Average Accuracy>  96.9455135544
-- size = 50 , R = 0.1 , func = tanh , node = 50  --
<Average Accuracy>  96.5115019505
-- size = 50 , R = 0.01 , func = relu , node = 10  --
<Average Accuracy>  96.6199604142
-- size = 50 , R = 0.01 , func = relu , node = 50  --
<Average Accuracy>  96.6700439041
-- size = 50 , R = 0.01 , func = tanh , node = 10  --
<Average Accuracy>  96.394689268
-- size = 50 , R = 0.01 , func = tanh , node = 50  --
<Average Accuracy>  96.261084501
-- size = 100 , R = 0.1 , func = relu , node = 10  --
<Average Accuracy>  96.3945847096
-- size = 100 , R = 0.1 , func = relu , node = 50  --
<Average Accuracy>  96.5949291251
-- size = 100 , R = 0.1 , func = tanh , node = 10  --
<Average Accuracy>  96.8787320826
-- size = 100 , R = 0.1 , func = tanh , node = 50  --
<Average Accuracy>  96.2945431999
-- size = 100 , R = 0.01 , func = relu , node = 10  --
<Average Accuracy>  96.7201378498
-- size = 100 , R = 0.01 , func = relu , node = 50  --
<Average Accuracy>  96.761856665
-- size = 100 , R = 0.01 , func = tanh , node = 10  --
<Average Accuracy>  96.2527930172
-- size = 100 , R = 0.01 , func = tanh , node = 50  --
<Average Accuracy>  96.2193865975
```

Then we have the optimal parameters for **circles**:

```
[Optimal Batch Size] 10
[Optimal Learning Rate] 0.1
[Optimal Activation Function] tanh
[Optimal Hidden Layer Width] 10
[Maximum Accuracy] 96.9621801688%
```

iii. Run the *plot_learning_curve.py*, we can get learning curves in Figure 1 and Figure 2.

Figure 1: Learning Curve for Circles

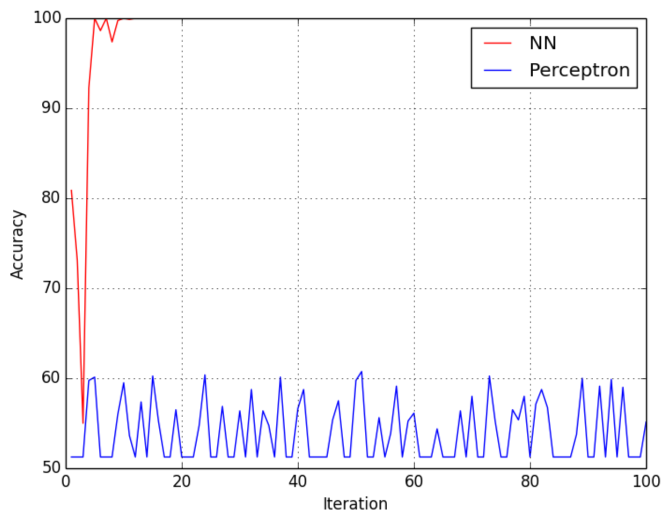
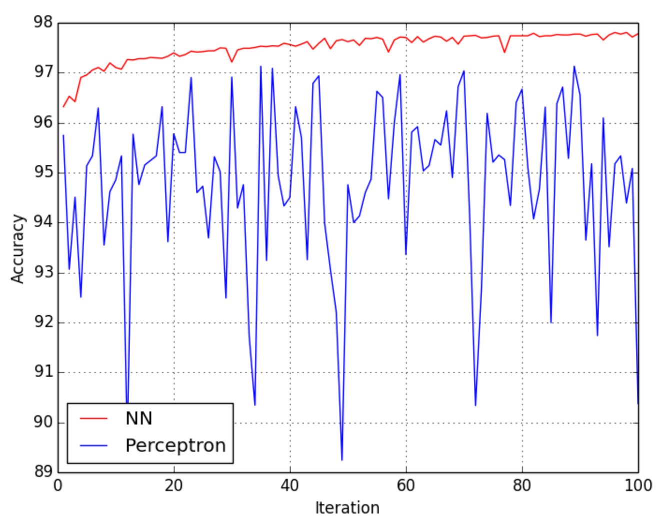


Figure 2: Learning Curve for MNIST



- In Figure 1, we can find that the performance of Perceptron is terrible with accuracy between 50% and 60%. This is because the circle dataset is not linearly separable, while Perceptron can only learn linear functions. Neural network performs much better and reach nearly 100% accuracy after a few iterations.

- In Figure 2, Perceptron can learn quite well with the average accuracy around 95%, but it is still not as good as neural network. In this case, we can also find that it takes more iterations for neural networks converging.

2. **Multi-class classification**

(a) i.
- **OvA**: We learn k classifiers.
- **AvA**: We learn $\binom{k}{2}$ classifiers.

ii.
- **OvA**: We use m examples to learn each classifier.
- **AvA**: Wwe use $\frac{2m}{k}$ examples to learn each classifier.

iii.
- **OvA**: We use the winner takes all (WTA) strategy, such that $f(x) = argmax_i \omega_i^T x$. The "score" $\omega_i^T x$ can be thought of as the probability that x has label i.
- **AvA**: We have two options to make a decision. One is to classify example x to take label i if i wins on x more oftern than any $j = 1, ..., k$. Alternatively, we can do a tournament. Starting with $n/2$ pairs, continue with the winners and go down iteratively.

iv.
- **OvA**: We need to train $O(k)$ (linear) number of classifiers.
- **AvA**: We need to train $O(k^2)$ (quadratic) number of classifiers.

(b) I prefer **OvA** than **AvA**. Because we have better computational complexity of **OvA** over **AvA**. **OvA** has more examples to learn than **AvA** as well.

(c) The analysis above doesn't hold for **Kernel Perceptron**. In this case, we need dual representations, **AvA** has smaller learning problems in terms of number of examples, thus being preferable when ran in dual. And it is also more expressive. So I prefer **AvA** in this case.

(d)
- **OvA**: We have m examples supplied for each classifier and k classifiers to learn. So the overall complexity is

$$O(dm^2)k = O(kdm^2)$$

- **AvA**: We have $\frac{m}{k}$ examples supplied for each classifier and $\binom{k}{2}$ classifiers to learn. So the overall complexity is

$$O(d(\frac{m}{k})^2)\binom{k}{2} = O(\frac{dm^2}{k^2})O(k^2) = O(dm^2)$$

.

So **AvA** is the most efficient.

(e)
- **OvA**: We have m examples supplied for each classifier and k classifiers to learn. So the overall complexity is

$$O(d^2m)k = O(kd^2m)$$

- **AvA**: We have $\frac{m}{k}$ examples supplied for each classifier and $\binom{k}{2}$ classifiers to learn. So the overall complexity is

$$O(d^2(\frac{m}{k}))\binom{k}{2} = O(\frac{d^2m}{k})O(k^2) = O(kd^2m)$$

.

So **OvA** and **AvA** are in the same order of time complexity. We cannot say which one is the most efficient.

(f)
- **Counting**: For each example, we need first predict with all the classifiers, which take $m(m-1)/2$, i.e. $O(m^2)$, time complexity. Then we do a majority vote, which takes $O(1)$ time. So the overall time complexity is $O(m^2)$.
- **Knockout**: For each round, we knocked out half of the classifiers. So the overall time complexity is $O(logm)$.

3. **Probability Review**

(a) i. Let X be the random variable donating the number of children in a family.

- Town A: Since each family has just one child. So

$$E[X] = 1$$

- Town B: We have

$$\begin{aligned} E[X] &= \sum_{k}^{\infty} k \times P(X = k) \\ &= 1 \times \frac{1}{2} + 2 \times (\frac{1}{2})^2 + 3 \times (\frac{1}{2})^3 + ... \end{aligned} \tag{1}$$

To compute this, we multiply both sides of (1) by 2.

$$2E[X] = 1 + 2 \times (\frac{1}{2}) + 3 \times (\frac{1}{2})^2 + ... \tag{2}$$

Subtract both sides of (2) with corresponding sides of (1).

$$\begin{aligned} E[X] &= 1 + \frac{1}{2} + (\frac{1}{2})^2 + ... \\ &= \lim_{k\to\infty} \frac{1 \times (1 - (\frac{1}{2})^k)}{1 - \frac{1}{2}} \\ &= 2 \end{aligned} \tag{3}$$

ii. Let X, Y be the random variable donating the numbers of boys and girls, respectively, in the town at the end of one generation. And there are N_A families in Town A and N_B in Town B.

- Town A: Since each family has just one child with equal possibility to be a boy or a girl. So
$$E[X] = E[Y] = N_A/2$$
Then
$$\frac{E[X]}{E[Y]} = \frac{N_A/2}{N_A/2} = 1$$
- Town B: Each family will have exactly one boy. So
$$E[X] = N_B$$
For girls
$$\begin{aligned} E[Y] &= N_B \\ &= N_B\big(\sum_{k=0}^{\infty} k \times P(Z=k)\big) \\ &= N_B\big(1 \times (\frac{1}{2})^2 + 2 \times (\frac{1}{2})^3 + 3 \times (\frac{1}{2})^4 + ...\big) \end{aligned} \tag{1}$$
where Z is the random variable donating number of girls in each family. Use the same method as in (i)
$$2E[Y] = N_B\big(1 \times \frac{1}{2} + 2 \times (\frac{1}{2})^2 + 3 \times (\frac{1}{2})^3 + ...\big) \tag{2}$$
From (1)(2), we have
$$\begin{aligned} E[Y] &= N_B\big(\frac{1}{2} + (\frac{1}{2})^2 + (\frac{1}{2})^3 + ...\big) \\ &= N_B \lim_{k\to\infty} \frac{\frac{1}{2}\big(1-(\frac{1}{2})^k\big)}{1-\frac{1}{2}} \\ &= N_B \end{aligned} \tag{3}$$
Then
$$\frac{E[X]}{E[Y]} = \frac{N_B}{N_B} = 1$$

(b) i. Proof: From the chain rule, we have
$$P(A,B) = P(A|B)P(B) \tag{1}$$
$$P(A,B) = P(B|A)P(A) \tag{2}$$
So from (1)(2) we will have
$$P(A|B)P(B) = P(B|A)P(A)$$
Then
$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

ii. From the chain rule, we can rewrite as

$$P(A,B,C) = P(A|B,C)P(B,C) = P(A|B,C)P(B|C)P(C)$$

(c) Proof: According to the definition of expectation:

$$\begin{aligned} E[X] &= 1 \times P(X=1) + 0 \times P(X=0) \\ &= 1 \times P(A) \\ &= P(A) \end{aligned}$$

(d) i. No.
We can calculate

$$P(X=0) = 1/15 + 1/10 + 4/15 + 8/45 = 11/18$$

$$P(Y=0) = 1/15 + 1/15 + 4/15 + 2/15 = 8/15$$

And

$$P(X=0, Y=0) = 1/15 + 4/15 = 1/3$$

Then

$$P(X=0, Y=0) \neq P(X=0)P(Y=0)$$

So X is not independent of Y.

ii. Yes. We need to prove for any $x, y, z \in \{0, 1\}$:

$$P(X=x, Y=y|Z=z) = P(X=x|Z=z)P(Y=y|Z=z)$$

We have:

$$P(Z=0) = 1/15 + 1/15 + 1/10 + 1/0 = 1/3P(Z=1) = 1 - P(Z=0) = 2/3 \tag{1}$$

We have:

$$P(X=0|Z=0) = \frac{1/15 + 1/10}{1/3} = 1/2 \tag{2}$$

$$P(X=1|Z=0) = \frac{1/15 + 1/10}{1/3} = 1/2 \tag{3}$$

$$P(Y=0|Z=0) = \frac{1/15 + 1/15}{1/3} = 2/5 \tag{4}$$

$$P(Y=1|Z=0) = \frac{1/10 + 1/10}{1/3} = 3/5 \tag{5}$$

$$P(X=0|Z=1) = \frac{4/15 + 8/45}{2/3} = 2/3 \tag{6}$$

$$P(X=1|Z=1) = \frac{2/15 + 4/45}{2/3} = 1/3 \tag{7}$$

$$P(Y=0|Z=1) = \frac{4/15+2/15}{2/3} = 3/5 \tag{8}$$

$$P(Y=1|Z=1) = \frac{8/45+4/45}{2/3} = 2/5 \tag{9}$$

And:

$$\begin{aligned} P(X=0,Y=0|Z=0) &= \frac{1/15}{1/3} = 1/5 \\ &= P(X=0|Z=0)P(Y=0|Z=0) \end{aligned} \tag{10}$$

$$\begin{aligned} P(X=0,Y=1|Z=0) &= \frac{1/10}{1/3} = 3/10 \\ &= P(X=0|Z=0)P(Y=1|Z=0) \end{aligned} \tag{11}$$

$$\begin{aligned} P(X=1,Y=0|Z=0) &= \frac{1/15}{1/3} = 1/5 \\ &= P(X=1|Z=0)P(Y=0|Z=0) \end{aligned} \tag{12}$$

$$\begin{aligned} P(X=1,Y=1|Z=0) &= \frac{1/10}{1/3} = 3/10 \\ &= P(X=1|Z=0)P(Y=1|Z=0) \end{aligned} \tag{13}$$

$$\begin{aligned} P(X=0,Y=0|Z=1) &= \frac{4/15}{2/3} = 2/5 \\ &= P(X=0|Z=1)P(Y=0|Z=1) \end{aligned} \tag{14}$$

$$\begin{aligned} P(X=0,Y=1|Z=1) &= \frac{8/45}{2/3} = 4/15 \\ &= P(X=0|Z=1)P(Y=1|Z=1) \end{aligned} \tag{15}$$

$$\begin{aligned} P(X=1,Y=0|Z=1) &= \frac{2/15}{2/3} = 1/5 \\ &= P(X=1|Z=1)P(Y=0|Z=1) \end{aligned} \tag{16}$$

$$\begin{aligned} P(X=1,Y=1|Z=1) &= \frac{4/45}{2/3} = 2/15 \\ &= P(X=1|Z=1)P(Y=1|Z=1) \end{aligned} \tag{17}$$

So X is conditionally independent of Y given Z.

iii.

$$\begin{aligned} P(X=0|X+Y>0) &= \frac{P(X=0, X+Y>0}{P(X+Y>0)} \\ &= \frac{1/10+8/45}{1-1/15-4/15} \\ &= 5/12 \end{aligned}$$