CS446: Machine Learning **Spring 2017**

Problem Set 5

Hsiang-Wei Hwang *Handed In: April 4, 2017*

1. **[Neural Networks]**

   (a)

$$f(x) = max(0, x) \Rightarrow f(x)' = \begin{cases} 1 & \text{if } x > 0 \\ 0 & otherwise \end{cases}$$

For output node j at the last layer,

$$net_j = \sum_i w_{ij} x_i$$

$$\frac{\partial Err}{\partial w_{ij}} = \frac{\partial Err}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} = \begin{cases} -(t_j - o_j) \times 1 \times x_{ij} & \text{if } net_j > 0 \\ 0 & otherwise \end{cases}$$

$$\text{and } \delta_j = \begin{cases} (t_j - o_j) & \text{if } net_j > 0 \\ 0 & otherwise \end{cases}$$

$$\Rightarrow \Delta w_{ij} = R \delta_j x_{ij}$$

Back propagation rule at node j:

$x_{jk} = o_j = f(\sum w_{ij} x_{ij})$, output of node j is the input of node k.

$$\frac{\partial Err}{\partial w_{ij}} = \frac{\partial Err}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} = \sum_{k \in downstream(j)} \frac{\partial Err}{\partial net_k} \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

$$\Rightarrow \frac{\partial Err}{\partial net_k} = -\delta_k, \ \frac{\partial net_k}{\partial o_j} = w_{jk}, \ \frac{\partial o_j}{\partial net_j} = \begin{cases} 1 & \text{if } net_j > 0 \\ 0 & otherwise \end{cases}, \ \frac{\partial net_j}{\partial w_{ij}} = x_{ij}$$

$$\Rightarrow \frac{\partial Err}{\partial w_{ij}} = \begin{cases} (\sum_{k \in downstream(j)} -\delta_k w_{jk}) x_{ij} & \text{if } net_j > 0 \\ 0 & otherwise \end{cases},$$

$$\text{and } \delta_j = \begin{cases} \sum_{k \in downstream(j)} \delta_k w_{jk} & \text{if } net_j > 0 \\ 0 & otherwise \end{cases}$$

$$\Rightarrow \Delta w_{ij} = R \delta_j x_{ij}$$

(b) i. `squared_loss_gradient(output, label)`:
Gradient should be $-(t_j - o_j) = o_j - t_j$.

```
17  def squared_loss_gradient (output_activations, y):
18      #IMPLEMENT THIS!
19      return np.subtract(output_activations, y)
20  #endDef
```

`relu_derivative(z)`:
Relu derivative should be $\begin{cases} 1 & \text{if } z > 0 \\ 0 & otherwise \end{cases}$.

```
43  def relu_derivative (z):
44      #IMPLEMENT THIS!
45      return np.transpose(np.array([[ int(y > 0) for y in z]]))
46  #endDef
```

ii. For data set mnist, listed in descending order of accuracy :

batch_size	activation_function	learning_rate	hidden_layer_width	Accuracy
10	tanh	0.1	10	96.9783
10	tanh	0.1	50	96.9700
50	tanh	0.1	10	96.8949
10	tanh	0.01	10	96.7864
100	tanh	0.1	10	96.7697
100	relu	0.01	50	96.7279
50	relu	0.01	10	96.7196
100	relu	0.01	10	96.7112
50	relu	0.01	50	96.7029
50	tanh	0.1	50	96.5276
50	relu	0.1	50	96.4775
10	relu	0.01	50	96.4191
50	relu	0.1	10	96.3940
100	relu	0.1	10	96.3523
100	relu	0.1	50	96.3273
10	relu	0.01	10	96.3189
50	tanh	0.01	10	96.3189
100	tanh	0.1	50	96.2688
10	tanh	0.01	50	96.2521
100	tanh	0.01	50	96.2021
10	relu	0.1	50	96.1687
100	tanh	0.01	10	96.1603
50	tanh	0.01	50	96.1436
10	relu	0.1	10	95.9099

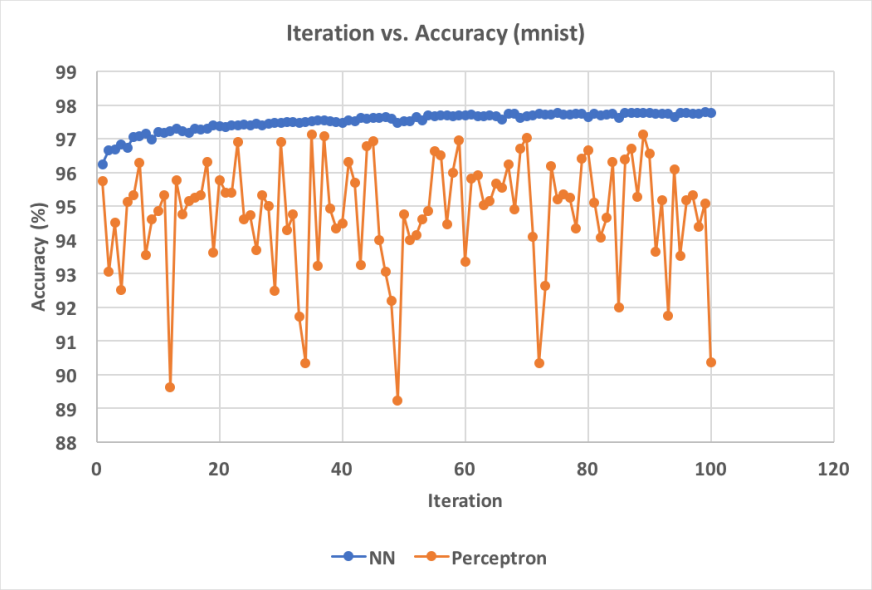
For data set circles, listed in descending order of accuracy :

batch_size	activation_function	learning_rate	hidden_layer_width	Accuracy
10	tanh	0.1	10	100
10	tanh	0.1	50	100
10	relu	0.1	10	100
10	relu	0.1	50	100
10	relu	0.01	10	100
10	relu	0.01	50	100
50	relu	0.1	10	100
50	relu	0.1	50	100
100	relu	0.1	10	100
100	relu	0.1	50	100
50	relu	0.01	50	95.25
100	relu	0.01	50	80.375
50	relu	0.01	10	64.125
50	tanh	0.1	50	61.375
100	relu	0.01	10	60.5
50	tanh	0.1	10	55
10	tanh	0.01	10	51.625
10	tanh	0.01	50	51.125
100	tanh	0.01	10	51
100	tanh	0.01	50	50.25
50	tanh	0.01	10	49.875
50	tanh	0.01	50	49.75
100	tanh	0.1	10	49.375
100	tanh	0.1	50	49.375

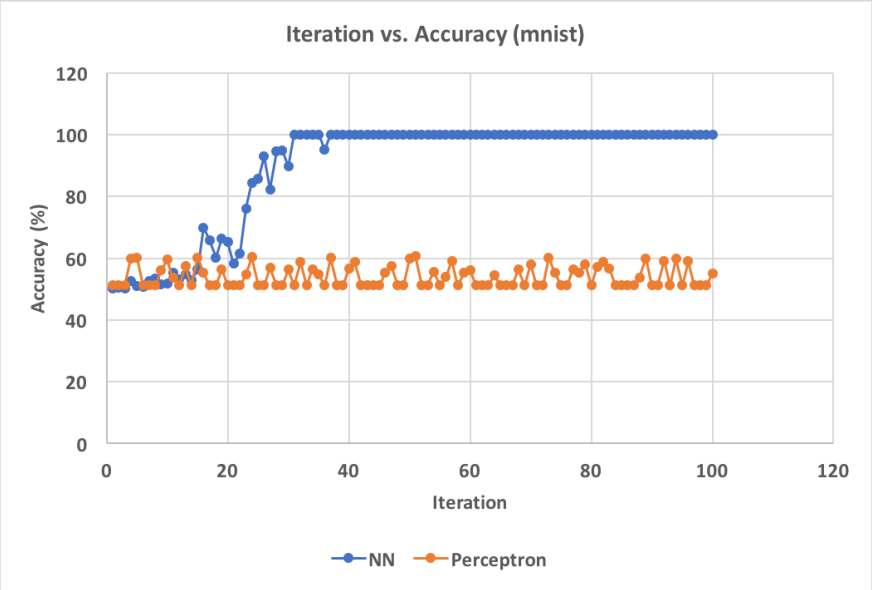
Thus, I will choose parameters as the following table.

batch_size	activation_function	learning_rate	hidden_layer_width
10	tanh	0.1	10

iii. Iteration vs. Accuracy in mnist data set for neural network and perceptron:



Iteration vs. Accuracy in circles data set for neural network and perceptron:



Accuracy(%) for both algorithm in different data sets:

mnist dataset		circles dataset	
Neural Network	Perceptron	Neural Network	Perceptron
96.7238	88.8609	100	49.5

2. [**Multi-class classification**]

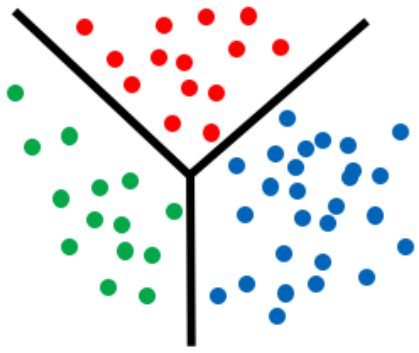
   (a) i. For one vs. all, there are **k** classifiers because each label will generate one classifier and there are k labels.
   For all vs. all, there are **k(k-1)/2** classifiers because each pair of labels will generate one classifier and there are $C_2^k = k(k-1)/2$ pairs of labels.

   ii. For one vs. all, number of positive examples is m/k and number of negative examples is m(k-1)/k for each classifier. Total number of examples used to learn is **m**.
   For all vs. all, number of positive examples is m/k and number of negative examples is m/k for each classifier. Total number of examples used to learn is **2m/k**.

   iii. For one vs. all, there are k classifiers corresponding to k labels. The input example will be labeled i if the i_{th} classifier returns positive. For all vs. all, every pair, $\langle i, j \rangle$, has its own classifier. If the example is labeled by all classifiers $\langle i, j \rangle, j \in [1, 2, ..., k], j \neq i$ as i, the example is thought to be in i.

   iv. For one vs. all, there are k classifiers with m training examples for each, so $O(km)$.
   For all vs. all, there are k(k-1)/2 classifiers with 2m/k training examples for each, so $O(k(k-1)/2 \times 2m/k) = O(km)$.

   (b) All vs. all.
   According to the analysis above, both method achieve the same computational complexity. However, it is possible that sets between all pairs of labels are linear separable but not separable when considering the set of one label to the union set of all other labels. If that happens, all-vs-all method can classify the examples well, but one-vs-one method cannot. For example, there are 3 sets of examples in 3 labels shown in figure. Using one-vs-all method is impossible to find a linear classifier telling the set in red. On the other hand, using all-vs-all method, all pairs of sets are linear separable generating corresponding classifiers telling each pair of sets. The examples can be classified correctly. Thus, all vs. all is a better choice.



(c) A KernelPerceptron can extend the space of the examples into another space in higher order and possibly separate the examples in the new space. Considering the same linear separability issue in the higher order space as we had in (b), all-vs-all method will outperform one-vs-one method. Besides, kernel functions should work in dual mode because most of kernel functions cannot find a transfer function from original space to the higher order space, but can calculate the inner product of two vectors. That means the computational complexity of these two methods when classifying is proportional to the training examples and the number of the classifiers. Thus, the computational complexity for one-vs-all method is $O(mk)$ and the computational complexity for all-vs-all method is $O(k(k-1)/2 \times 2m/k) = O(km)$. Thus, after considering performance and complexity, I still choose all vs. all.

(d) For one vs. all, there are k classifiers with m training examples to learn. The computational complexity is $O(kdm^2)$. For all vs. all, there are k(k-1)/2 classifiers with 2m/k training examples to learn. The computational complexity is $O(\frac{k(k-1)}{2}d(\frac{2m}{k})^2) = O(dm^2)$. All vs. all is most efficient.

(e) For one vs. all, there are k classifiers with m training examples to learn. The computational complexity is $O(kd^2m)$. For all vs. all, there are k(k-1)/2 classifiers with 2m/k training examples to learn. The computational complexity is $O(\frac{k(k-1)}{2}d^2(\frac{2m}{k})) = O(kd^2m)$. Both are the same in efficiency.

(f) It takes $O(d)$ to calculate $w_i^T x$. For **Counting**, it takes $O(d \times k(k-1)/2) = O(dk^2)$ because all k(k - 1)/2) classifiers should be calculated. For **Knockout**, after (k-1) comparisons, there will be only one label left which means the end of the classification. Thus, total number of comparisons is k-1. It takes $O(d \times (k-1)) = O(dk)$.

3. [**Probability Review**]

(a) i. Expected number of children in a family in town A is **1** because each family has one child no matter a boy or a girl.
Expected number of children in a family in town B:

$$f(i) = (1-p)^{n_i-1}p$$

where p is the probability of having a boy child, n_i is the number of children in family i.

$$L(f) = \prod_i (1-p)^{n_i-1}p, l(f) = \sum_i log(p) + (n_i - 1)log(1-p)$$

To calculate maximum likelihood, we calculate derivative of it to be 0.

$$\frac{\partial l(f)}{\partial p} = \sum_i \frac{1}{p} - \frac{(n_i-1)}{(1-p)} = \frac{N}{p} - \frac{\sum_i n_i - N}{(1-p)} = 0$$

$$N(1-p) - (\sum_i n_i - N)p = 0 \Rightarrow N = p\sum_i n_i \Rightarrow \frac{\sum_i n_i}{N} = \frac{1}{p} = 2$$

where N is total families. Thus, expected number of children in a family in B is **2**.

ii. The boy to girl ratio in town A is **1** because boy and girl have the same born rate.
The boy to girl ratio in town B:
The number of girls is $\sum(n_i - 1)$, and the number of boys is N. In previous calculation, we know $\frac{\sum_i n_i}{N} = \frac{1}{p} = 2$. Thus,

$$\sum_i (n_i - 1) = \sum_i n_i - N = N$$

The boy to girl ratio in town B is **1**.

(b) i.

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A \cap B)P(A)}{P(A)P(B)}$$

$$\because P(A \cap B) = P(B \cap A) \Rightarrow P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

ii.

$$\because P(A|B) = \frac{P(A \cap B)}{P(B)}, P(B|C) = \frac{P(B \cap C)}{P(C)} \Rightarrow P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

iii.

$$\because P(B, C) = P(B|C)P(C), P(C) = \frac{P(C|A)}{P(A|C)}P(A)$$

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

$$= P(A|B, C)P(B|C)\frac{P(C|A)}{P(A|C)}P(A)$$