CS446: Machine Learning **Fall 2017**

Problem Set 5

Handed Out: Official Solution *Due: April* 17^{th}, 2016

1. [**Neural Networks - 50 points**]

(a) As specified in the notes,

$$\Delta w_{ij} = -R\frac{\partial E}{\partial w_{ij}}$$

where R is the learning rate and E is the squared error loss function. For a weight connected to an output node (i.e. node j is one of the output nodes), we apply the chain rule to get the following:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

$\frac{\partial E}{\partial o_j}$ and $\frac{\partial net_j}{\partial w_{ij}}$ are identical to the values found in the notes, since we are using the same loss; the partial derivative of the activation function is straightforward, since it is piecewise linear. One technicality arises due to the fact that it is not differentiable when the input equals 0; for our purposes, though, we can choose to set the derivative there equal to 0 or 1. Thus, we have the following:

$$\frac{\partial E}{\partial o_j} = -(t_j - o_j)$$

$$\frac{\partial o_j}{\partial net_j} = \begin{cases} 1, & \text{if } net_j > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\frac{\partial net_j}{\partial w_{ij}} = x_{ij}$$

where t_j is the correct output for node j, o_j is the output produced by node j, and x_{ij} is the output of node i (which feeds into node j). When w_{ij} is not connected to an output node (i.e. node j is not an output node), we have to consider all of the nodes that j influences; this gives us

$$\begin{aligned}
\frac{\partial E}{\partial w_{ij}} &= \frac{\partial E}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}} \\
&= \sum_{k\in\text{downstream}(j)} \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}} \\
&= \sum_{k\in\text{downstream}(j)} \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}
\end{aligned}$$

Where $\frac{\partial o_j}{\partial net_j}$ and $\frac{\partial net_j}{\partial w_{ij}}$ are the same as before, and

$$\frac{\partial E}{\partial net_k} = -\frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial net_k}$$
$$\frac{\partial net_k}{\partial o_j} = w_{jk}$$

(b) Here are correct implementations for the two incomplete functions:

```
def squared_loss_gradient (output_activations, y):
    return output_activations - y

def relu_derivative(z):
    derivatives = np.zeros(z.shape)
    for i, val in np.ndenumerate(z):
        if val > 0:
            derivatives[i] = 1

# OR
def relu_derivative(z):
    return 1.0 * (z > 0.0)
```

The exact best parameter setting for the neural network may vary. A correct solution has two learning curve graphs; here are examples of what those should look like: For the these runs on the circle data, the neural network (using tanh, batch size = 10, learning rate = 0.1, hidden layer width = 10) achieved an accuracy of 100% on the test data, while the perceptron achieved an accuracy of 49.5%. For these runs on the mnist data, the neural network (using the same parameter settings as before) achieved an accuracy of 96.875% on the test data, while the perceptron achieved an accuracy of 88.9%.

There are two primary conclusions one can arrive at based on this experiment:

- There are datasets and hypotheses on which a linear model will perform poorly but a neural network will perform well (as indicated by the circles results).
- Just because a hypothesis is "complex" doesn't mean a linear learner will perform poorly (as indicated by the mnist results).

2. **[Multi-class classification - 30 points]**

   (a) i. Number of classifiers

   - For the **One vs. All** scheme, we will have k classifiers.
   - For the **All vs. All** scheme, we will have $\binom{k}{2} = \frac{k(k-1)}{2}$ classifiers.

   ii. Number of examples used to learn each classifier

   - Each classifier in **One vs. All** scheme is learned over m examples – $\frac{m}{k}$ "positive" and $\left(m - \frac{m}{k}\right) = \frac{(k-1)m}{k}$ "negative".
   - In the **All vs. All** scheme, each classifier is learned over only $\frac{2m}{k}$ examples – $\frac{m}{k}$ "positive" and another $\frac{m}{k}$ "negative".

   iii. Labeling a new example $\mathbf{x}$

   - For the **One vs. All** scheme, assume that the k classifiers correspond to k weight vectors, $\mathbf{w}_1, \mathbf{w}_2, \ldots \mathbf{w}_k$, and each classifier classifies an example $\mathbf{x}$ as `positive` or `negative` based on $sgn(\mathbf{w}_i \cdot \mathbf{x})$. We would ideally have only one classifier giving the label `positive` (i.e. $\mathbf{w}_i \cdot \mathbf{x} \geq 0$), In general though, we can choose the label that achieves the highest score, i.e. $y^* = \arg\max_i \mathbf{w}_i \cdot \mathbf{x}$.
   - For the **All vs. All** scheme, we have several options.
     One approach would be to apply all the $\binom{k}{2}$ classifiers on example $\mathbf{x}$ and let each classifier vote on the class label. The label with highest number of votes would be the winner.
     Another approach is to conduct a tournament between the labels.

   iv. Computational complexity

   - For the **One vs. All** scheme, we have k classifiers, each learned over m examples. So, the computational complexity is $O(mk)$.
   - For the **All vs. All** scheme, we have $\binom{k}{2}$ classifiers, each learned over $\frac{2m}{k}$ examples. So, the computational complexity is $O\left(\frac{2m}{k} \times \frac{k(k-1)}{2}\right) = O(mk)$.

   (b) Based on the analysis above, we see that both schemes are of the same order of complexity, $O(mk)$. So, either scheme is fine, when using simple Perceptron-style classifier.

   (c) Recall that a KernelPerceptron has the computational complexity of $O(m^2)$, where m is the number of examples used by the training algorithm. Note that this is different from the simple Perceptron (which is of order $O(m)$).

   This changes the analysis we did earlier.

- For the **One vs. All** scheme, we have k classifiers, each learned over m examples. So, the computational complexity of using KernelPerceptron is $O(m^2 k)$.
- For the **All vs. All** scheme, we have $\binom{k}{2}$ classifiers, each learned over $\frac{2m}{k}$ examples. So, the computational complexity of using KernelPerceptron is $O\left(\frac{4m^2}{k^2} \times \frac{k(k-1)}{2}\right) = O(m^2)$.

So, when using KernelPerceptron, we would prefer the **All vs. All** scheme over the **One vs. All** scheme.

(d)
- For the **One vs. All** scheme, we have k classifiers, each learned over m examples. So, the computational complexity of using the blackbox learning algorithm is $O(m^2 kd)$.
- For the **All vs. All** scheme, we have $\binom{k}{2}$ classifiers, each learned over $\frac{2m}{k}$ examples. So, the computational complexity of using blackbox learning algorithm is $O\left(d\frac{4m^2}{k^2} \times \frac{k(k-1)}{2}\right) = O(m^2 d)$.

(e)
- For the **One vs. All** scheme, we have k classifiers, each learned over m examples. So, the computational complexity of using the blackbox learning algorithm is $O(mkd^2)$.
- For the **All vs. All** scheme, we have $\binom{k}{2}$ classifiers, each learned over $\frac{2m}{k}$ examples. So, the computational complexity of using blackbox learning algorithm is $O\left(d^2\frac{4m}{k} \times \frac{k(k-1)}{2}\right) = O(d^2 mk)$.

(f)
- For the **Counting** scheme, we need to run each classifier once on the given example, which is $\frac{m(m-1)}{2}$. Time complexity is $O(m^2)$.
- For the **Knockout** scheme, each time, we will eliminate one class, and in order to pick the final winner, we need to run (m-1) classifier. Time complexity is $O(m)$.

3. **[Probability Review - 20 points]**

(a) i. Let X be a random variable to denote number of children in a family.

- Town A: Since every family has exactly one child (a uniform distribution), the expected value of number of children, $E[X] = 1$.
- Town B: The expected value of number of children is given as

$$E[X] = \sum_i i.P(X = i) = 1 \times \frac{1}{2} + 2 \times \left(\frac{1}{2}\right)^2 + ...$$

Notice that this is the same as finding the expected value of a geometric series with ratio $\lambda = 0.5$. We can show that the expected value of geometric series with parameter λ is 1. Hence, the expected number of λ children in a family in town B,

$$E[X] = \frac{1}{0.5} = 2.$$

This is, in fact, also easy to compute, if you dont know the formula by heart. There are multiple ways to prove it, and is left as an exercise.

ii. Let X be number of boy children and Y be number of girl children in a town.

- Town A: Let there be m families in town A. Since it is equally likely to have a boy child or a girl child, and each family has only one child, $E[X] = E[Y] = \frac{m}{2}$, and the boy to girl ratio is $E[X] = E[Y] = 1$.
- Town B: Let there be n families in town B. Since each family stops having children when a boy child is born and not earlier, there is a boy child in every family. So, $E[X] = n$. Let us compute $E[Y]$, the expected number of girl children in town B:

$$\begin{aligned}
E[Y] &= n\left[\sum_i i.P(X = i)\right] \\
&= n\left[1 \times \left(\frac{1}{2}\right)^2 + 2 \times \left(\frac{1}{2}\right)^3 + ...\right] \\
&= n
\end{aligned}$$

So, we see that for town B, $E[X] = E[Y] = n$. Hence the boy to girl ratio in town B is also $E[X] : E[Y] = 1 : 1$.

(b) i. This is Bayes theorem. From the chain rule, we have that $P(A, B) = P(A|B)P(B)$ and $P(A, B) = P(B|A)P(A)$. Therefore, $P(A|B)P(B) = P(B|A)P(A)$. Rearranging gives the desired equality.

ii.

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

This follows from repeated application of the chain rule:

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

(c)

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

(d) Using the facts that for any events A and B, (i.e., any subsets of the possible assignments of 0 and 1 to the variables X, Y , and Z) we have $P(A) = \sum_{(x,y,z)\in A} P(X=x, Y=y, Z=z)$ and $P(A|B) = P(A,B)/P(B)$, we have the following solutions

i. No.

$$
\begin{aligned}
P(X=0) &= 1/15 + 1/10 + 4/15 + 8/45 = 11/18, \\
P(Y=0) &= 1/15 + 1/15 + 4/15 + 2/15 = 8/15, \\
P(X=0, Y=0) &= 1/15 + 4//15
\end{aligned}
$$

we can see that:

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

Another way of looking at it is inspecting $P(X=0|Y=0)$:

$$P(X=0|Y=0) = \frac{P(X=0, Y=0)}{P(Y=0)} = \frac{1/15 + 4//15}{8/15} = 5/8$$

Since $P(X=0|Y=0)$ does not equal $P(X=0)$, X is not independent of Y.

ii. For all pairs $y, z \in \{0,1\}$, we need to check that $P(X=0|Y=y, Z=z) = P(X=0|Z=z)$. That the other probabilities are equal follows from the law of total probability. First we have

$$
\begin{aligned}
P(X=0|Y=0, Z=0) &= \frac{1/15}{1/15 + 1/15} = 1/2 \\
P(X=0|Y=1, Z=0) &= \frac{1/10}{1/10 + 1/10} = 1/2 \\
P(X=0|Y=0, Z=1) &= \frac{4/15}{4/15 + 2/15} = 2/3 \\
P(X=0|Y=1, Z=1) &= \frac{8/45}{8/45 + 4/45} = 2/3
\end{aligned}
$$

Also,

$$
\begin{aligned}
P(X=0|Z=0) &= \frac{1/15 + 1/10}{1/15 + 1/15 + 1/10 + 1/10} = 1/2 \\
P(X=0|Z=1) &= \frac{4/15 + 8/45}{2/15 + 4/15 + 8/45 + 4/45} = 2/3
\end{aligned}
$$

This shows that X is independent of Y given Z.

iii.

$$P(X=0|X+Y>0) = \frac{1/10 + 8/45}{1/15 + 1/10 + 1/10 + 2/15 + 4/45 + 8/45} = 5/12$$