CS 615 - Deep Learning

Assignment 2 - Artificial Neural Networks
John Obuch

1 Theory

1. Another common activation function is the hyperbolic tangent function, *tanh*, which is defined as:

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{1}$$

What is $\frac{\partial tanh(x\theta)}{\partial \theta_j}$ (5pts)?

Observing the provided definition of $tanh(z)$ and we seek to solve $\frac{\partial tanh(x\theta)}{\partial \theta_j}$, we will write our hyperbolic tangent function as follows:

$$tanh(x\theta) = \frac{e^{x\theta} - e^{-x\theta}}{e^{x\theta} + e^{-x\theta}}$$

Taking the derivative and applying the quotient rule we have the following:

$$\frac{\partial tanh(x\theta)}{\partial \theta_j} = \frac{(e^{x\theta} + e^{-x\theta})(x_j e^{x\theta} + x_j e^{-x\theta}) - (e^{x\theta} - e^{-x\theta})(x_j e^{x\theta} - x_j e^{-x\theta})}{(e^{x\theta} + e^{-x\theta})^2}$$

$$= \frac{(x_j e^{2x\theta} + x_j + x_j + x_j e^{-2x\theta}) - (x_j e^{2x\theta} - x_j - x_j + x_j e^{-2x\theta})}{(e^{x\theta} + e^{-x\theta})^2}$$

$$= \frac{x_j e^{2x\theta} + 2x_j + x_j e^{-2x\theta} - x_j e^{2x\theta} + 2x_j - x_j e^{-2x\theta}}{(e^{x\theta} + e^{-x\theta})^2}$$

$$= \frac{4x_j}{(e^{x\theta} + e^{-x\theta})^2}$$

■

Approaching the solution in another form we see the following:

$$\frac{\partial tanh(x\theta)}{\partial \theta_j} = \frac{(e^{x\theta} + e^{-x\theta})(x_j e^{x\theta} + x_j e^{-x\theta}) - (e^{x\theta} - e^{-x\theta})(x_j e^{x\theta} - x_j e^{-x\theta})}{(e^{x\theta} + e^{-x\theta})^2}$$

$$= \frac{(e^{x\theta} + e^{-x\theta})x_j(e^{x\theta} + e^{-x\theta}) - (e^{x\theta} - e^{-x\theta})x_j(e^{x\theta} - e^{-x\theta})}{(e^{x\theta} + e^{-x\theta})^2}$$

$$= \frac{x_j(e^{x\theta} + e^{-x\theta})^2 - x_j(e^{x\theta} - e^{-x\theta})^2}{(e^{x\theta} + e^{-x\theta})^2}$$

$$= x_j[1 - \frac{(e^{x\theta} - e^{-x\theta})^2}{(e^{x\theta} + e^{-x\theta})^2}]$$

■

2. Given a network with a single hidden layer, a softmax activation function at the output, a linear activation at the hidden layer, and a cross-entropy objective function what is:

   (a) $\frac{\partial J}{\partial \theta_{jk}}$ (5pts)

   Let us first define our objective function and activation functions for us to reference:

   Cross Entropy Loss Objective Function:

   $$J = -\sum y_k \ln(\hat{y_k})$$

   Given that our typical distribution of y will be all zeros with just one location of probability $y_a = 1$, we can simplify the above objective function as follows:

   $$J = -\ln(\hat{y_a})$$

   Activation Function (Hidden Layer)

   $$g(z) = z$$

   Activation Function (Output Layer)

   $$g(z) = \frac{e^z}{\sum_{k=1}^{K} e^{z_k}}$$

   From here, let us perform back propagation to obtain $\frac{\partial J}{\partial \theta_{jk}}$. Recall that:

   $$\frac{\partial J}{\partial \theta_{jk}} = \frac{\partial J}{\partial g(net_{o_k})} \cdot \frac{\partial g(net_{o_k})}{\partial net_{o_k}} \cdot \frac{\partial net_{o_k}}{\partial \theta_{jk}}$$

   Given the above chained gradient formulation, we will proceed by computing it's parts. Recall that $net_{o_k} = h\theta_{:k}$. Therefore, we have that:

   $$\frac{\partial net_{o_k}}{\partial \theta_{jk}} = h_j$$

   .

   Given that we have a Softmax activation function (as outlined above for reference) we will compute $\frac{\partial g(net_{o_k})}{\partial net_{o_k}}$. Recall that our Softmax activation function will be defined as follows:

   $$g(net_{o_k}) = g(h \mid \theta_{:,j}) = \frac{e^{h\theta_{:,j}}}{\sum_{k=1}^{K} e^{h\theta_{:,k}}}$$

After some work, we see that we have two cases:

$$\frac{\partial g(net_{o_k})}{\partial net_{o_k}} = \left\{ \begin{array}{ll} -g(h \mid \theta_{:,a})g(h \mid \theta_{:,k}), & \text{if } k \neq a \\ (g(h \mid \theta_{:,k}) - g(h \mid \theta_{:,k})^2), & \text{if } k = a \end{array} \right\}$$

Now we compute the final part of our chained gradient, namely $\frac{\partial J}{\partial g(net_{o_k})}$. Since our objective function is the cross entropy loss, we see the following from leveraging the equations we initially defined above:

$$\frac{\partial J}{\partial g(net_{o_k})} = -\frac{1}{g(net_{o_k})} = -\frac{1}{\hat{y}} = -\frac{1}{g(h \mid \theta_{:,a})}$$

.

Combining the resulting chained gradients, we yield the following:

$$\frac{\partial J}{\partial \theta_{jk}} = -\frac{1}{g(h \mid \theta_{:,a})} \left\{ \begin{array}{ll} -g(h \mid \theta_{:,a})g(h \mid \theta_{:,k}), & \text{if } k \neq a \\ (g(h \mid \theta_{:,k}) - g(h \mid \theta_{:,k})^2), & \text{if } k = a \end{array} \right\} h_j$$

Simplifying our result we obtain the following:

$$\frac{\partial J}{\partial \theta_{jk}} = \left\{ \begin{array}{ll} g(h \mid \theta_{:,k}), & \text{if } k \neq a \\ (-1 + g(h \mid \theta_{:,k})), & \text{if } k = a \end{array} \right\} h_j$$

Since our second case condition contains target labels, (i.e. If we have a binary classifier such that $y \in \{0, 1\}$ then we can write the above as follows:

$$\frac{\partial J}{\partial \theta_{jk}} = (g(h \mid \theta_{:,k}) - y)h_j = (\hat{y_k} - y_k)h_j = h_j(\hat{y_k} - y_k)$$

■

(b) $\frac{\partial J}{\partial \beta_{ij}}$ (5pts)

Here, we seek to find β_{ij} which does not have a specific output k. Thus, we will need to take the derivative of each term of the summation:

$$\frac{\partial J}{\partial \beta_{ij}} = \sum_{k=1}^{K} \frac{\partial J_k}{\partial \beta_{ij}}$$

$$\frac{\partial J}{\partial \beta_{ij}} = \sum_{k=1}^{K} (\frac{\partial J_k}{\partial g(net_{o_k})} \cdot \frac{\partial g(net_{o_k})}{\partial net_{o_k}} \cdot \frac{\partial net_{o_k}}{\partial g(net_{h_j})} \cdot \frac{\partial g(net_{h_j})}{\partial net_{h_j}} \cdot \frac{\partial net_{h_j}}{\partial \beta_{ij}})$$

Computing each derivative one at a time, we get that:

$$\frac{\partial net_{h_j}}{\partial \beta_{ij}} = x_i$$

Since our activation function at the hidden layer is linear (e.g. $g(z) = z$) we see that:

$$\frac{\partial g(net_{h_j})}{\partial net_{h_j}} = 1$$

Now, taking the derivative of $net_{o_k} = h\theta_{:k}$ with respect to h we obtain the following:

$$\frac{\partial net_{o_k}}{\partial g(net_{h_j})} = \theta_{jk}$$

The last two terms we get for free since we computed them earlier. Combining these results together, we obtain the following:

$$\frac{\partial J}{\partial \beta_{ij}} = x_i \sum_{k=1}^{K} (\hat{y}_k - y_k)\theta_{jk}$$

■

2 Shallow Artificial Neural Networks

Chosen Hyper Parameters:

1. Initial Parameters: β and θ where the column vectors of both β and θ were initialized with uniformly random values between $[-1, 1]$ such that the resulting dimensions of the β matrix are (D, M) where D is the number of input nodes with respect to the training data set, and M is the number of hidden nodes. Similarly, the resulting dimensions of the θ matrix are (M, K) where M here is again the number of hidden nodes and K is the number class labels.
2. Number of Hidden Nodes: 1000
3. Learning Rate: $\eta = 0.01$
4. L2 Regularization amount: $\lambda = 0.5$ where $0 \leq \lambda \leq \infty$ is our blending factor. Recall that $L2 = \frac{1}{2}\theta^T\theta$ and we regularize our cost function by performing the operation $\lambda L2$ and subtracting it from the logarithmic cost function. Similarly, we penalize the gradients by subtracting off $\lambda\beta$ and $\lambda\theta$.
5. Change in Cost Threshold: log_thresh = 0.001
6. Random seed 1

Note: Training occurs until convergence criteria is met. Specifically, the algorithm will converge/terminate when the change in the log cost is less than or equal to log_thresh (i.e., While (chg_in_log > log_thresh), train the data).

Testing Accuracy:

Tux: Acc = 0.9047619

What follows are the resulting convergence graphs and confusion matrix:

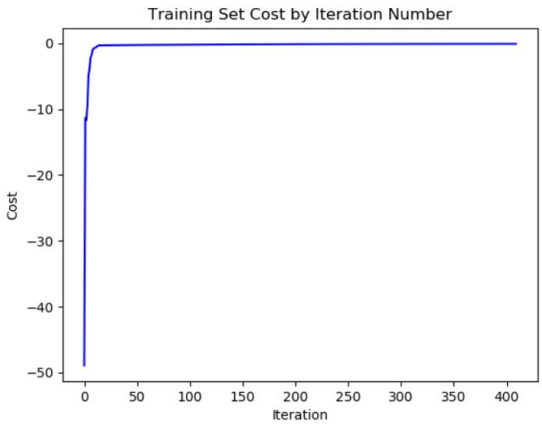

Figure 1: Average log Likelihood VS Iteration Number (Train)

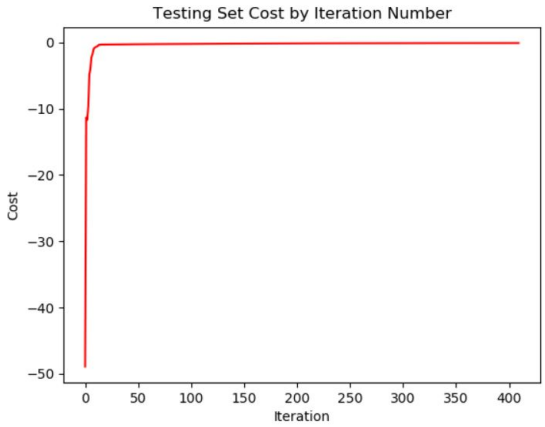

Figure 2: Average log Likelihood VS Iteration Number (Test)

```
TEST: CONFUSION MATRIX

[[3. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 3. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 3. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 3. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 3. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 3. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 2. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 2. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 3. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 3. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 3. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 3. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 2.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 2.]]
```

Figure 3: Confusion Matrix (Test) - Columns and Rows represent True Class and Predicted Class respectively.

3 Multi-Layer ANN

Chosen Hyper Parameters:

1. Initial Parameters: β and θ where the column vectors of both β and θ were initialized with uniformly random values between $[-1, 1]$ such that the resulting dimensions of the β matrix are (D, M) where D is the number of input nodes with respect to the training data set at each layer, and M is the number of hidden nodes. Similarly, the resulting dimensions of the θ matrix are (M, K) where M here is again the number of hidden nodes at the final layer and K is the number class labels.

2. Number of Hidden Nodes at each layer: *See network configuration table below*

3. Learning Rate: $\eta = 0.01$

4. L2 Regularization amount: $\lambda = 0.001$ where $0 \leq \lambda \leq \infty$ is our blending factor. Recall that $L2 = \frac{1}{2}\theta^T\theta$ and we regularize our cost function by performing the operation $\lambda L2$ and subtracting

it from the logarithmic cost function. Similarly, we penalize the gradients by subtracting off $\lambda\beta$ and $\lambda\theta$.

5. Change in Cost Threshold: log_thresh = 0.0001

6. Random seed 1

Note: Training occurs until convergence criteria is met. Specifically, the algorithm will converge/terminate when the change in the log cost is less than or equal to log_thresh (i.e., While (chg_in_log > log_thresh), train the data).

Network Configuration Table			
Configuration	**Iterations**	**Train Accuracy**	**Test Accuracy**
100, 30, 20	7717	1	0.95238
245, 100, 50	5255	1	0.95238
300, 217, 84	5043	1	0.97619
600, 358, 111	2067	1	0.92857

Figure 4: Network Configuration Table