CSE 4/574
Introduction to Machine Learning
Homework 4
Total Marks 60 (You can also earn a Bonus of 15)

Your Name: Karan Bali
Your email ID: kbali@buffalo.edu
Your UB Person ID: kbali

---

1. [8] What Is the Difference Between a Feedforward Neural Network and Recurrent Neural Network?

Ans.1. Feedforward neural networks are the most common type of neural networks where the flow of outputs calculated at the nodes of a hidden layer is propagated only in one direction (i.e. to the nodes of the next hidden layer). There are no connections between the nodes of the same level which forms some sort of cyclical structure. While Feedforward neural networks can work magically for most of the data patterns, it fails to capture the relationships between consequent inputs of the data sample. This is a great drawback when dealing with Time-series data.

On other hand, a Recurrent Neural Network is one where the flow of outputs calculated at the nodes of a hidden layer can be propagated in any direction (i.e. to the nodes of the next hidden layer or to the nodes of the same hidden layer). There can be connections between the nodes of the same level which forms some sort of cyclical structure. This also means that weights can be shared across time.

The key advantage of RNN is its ability to take the historical information(i.e. Calculated values of previous inputs) into its consideration while calculating the outputs of current input. This property is very significant when dealing with a Time-series data, where the consequent inputs are related to each other by a whole lot of factors. This level dependency is so severe in other forms of data. Thus, many of the applications of RNN like LSTM are commonly used in Natural language processing applications, where tokens in the input sentence have a great dependence on the preceding & succeeding tokens. There are various modifications to RNN that modify it to deal with longer relationships & shorter relationships as well. But, most importantly it's the possibility of flow of hidden layer outputs to same level layers that makes all the difference. Although this makes the RNN harder to learn & train in comparison to a simpler Feedforward neural network.

---

2. [7] What Are the Softmax and ReLU Functions? What is the use of leaky ReLU

function?

Ans.2.

Most of the data & complex relationships in nature happen to follow a non-linear transformation. Thus, while training our neural network, we need a way to induce this nonlinearity into our neural network. This can be done by using an activation function, which transforms the in-situ outputs of hidden layers into corresponding non-linear values.

There are many activation functions. Softmax & ReLU happens to be most common among them. Softmax activation function is commonly used for multiclass problems. It's a generalization of the sigmoid function for binary class problem to multiclass problem. Thus, a softmax activation function tries to divide(distribute) the probability among 'k' units of the hidden layer in proportion to their value(scale). Although softmax can be used in a hidden layer, it is commonly used in the end layer.

Rectified linear activation function (ReLU) is one the most efficient, common, popular & simple activation functions. Sigmoid & Tanh functions have a problem of saturations at higher impulse levels. Thus, They are more sensitive to middle values in comparison to extreme values. Thus, they try to squeeze the output values to 0 to 1 or -1 to 1 respectively. Due to above factors deep hidden layers are unable to learn faster. Thus,to counter Relu activation function, which is defined by y=max(0,x) can be used. It's more simpler & efficient in comparison to its alternatives. It also takes care of problems mentioned earlier, like sensitivity to higher impulses so that deeper layers can learn faster.

However, even ReLU has a drawback as it might knock off some of the weights during training due to large weight updates. Thus, once they update a weight largely, the same weight might not be updated again. This is due to the fact that gradient is 0 whenever the unit is not active or the learning happens in the negative direction. To counter this problem, a small slope can be provided in the negative direction to prevent the weight from dying during training. This slope can be chosen depending upon the objective, but it's usually very small. This ReLU function with a slope in negative direction is called as Leaky ReLU.

---

3. [8] What Is the Difference Between Batch Gradient Descent and Stochastic Gradient Descent?

Ans.3.

One of the most common techniques in machine learning is to define a cost(loss) function for the prospective prediction model on training samples & then the problem is transformed into finding the minima of that cost(loss) function w.r.t to the trained model. The weights of the trained model at the minima of the cost function are deemed as optimal weights for the current problem. There are many ways to find the minima of the cost function, however the most common method is to use the power of calculus (i.e. gradient of the function). The gradient of the function defines the slope of the function (i.e. the direction in which the cost seems to be decreasing the most.). Thus, gradient of the function helps us in finding the minima of the cost function.

But to compute the gradient, we need to process the information embedded in the training samples. Thus, the whole approach is to process & update the weights in the direction of gradient for each epoch. One epoch will update the weights once & so on the cycle will continue till the training is stopped. Each update is proportional to a Hyper-parameter of 'learning rate'.

The 2 methods (i.e. Batch Gradient Descent and Stochastic Gradient Descent) are the different ways of achieving the same goal (i.e. find the minima).

Techniques that use the whole data set at once are called "batch" methods. At each step the weight vector is moved in the direction of the greatest rate of decrease of the error function, and so this approach is known as gradient descent or steepest descent. Although such an approach might intuitively seem reasonable, in fact it turns out to be a poor algorithm especially w.r.t speed. Thus, Batch Gradient Descent is about processing & calculating the gradient for the whole data set in each epoch update. Although the update in each epoch is more stable in comparison to earlier updates, Batch Gradient Descent is way slower than it's alternative of Stochastic Gradient Descent.

There is, however, an on-line version of gradient descent that has proved useful
In practice for training neural networks on large data sets, it's Stochastic Gradient Descent. Stochastic Gradient Descent, also known as sequential gradient descent or stochastic gradient descent, makes an update to the weight vector based on one data point at a time, so that

$$E(w) = \sum_{n=1}^{N} E_n(w)$$

$$W_{t+1} = W_t - \eta \bigtriangledown E_n(W_t)$$

This update is repeated by cycling through the data either in sequence or by selecting points at random with replacement. There are of course intermediate scenarios in which the updates are based on batches of data points. One advantage of the Stochastic Gradient Descent method compared to batch Gradient Descent is that the former handle redundancy in the data much more efficiently. SGD is computationally a whole lot faster. However the updates in SDG are more noisy in comparison to Batch Gradient Descent. However this shortcoming can be overcome by leveraging the faster performance of SDG to perform more iterations with the same resources & time.

4. [10] How Are Weights Initialized in a Network? In a neural network, what if all the weights are initialized with the same value?

Ans.4.
Weights are the most important component of a trained model. You just need to know the values of the weights of the trained model in order to implement the same model somewhere else. However, to train the model you need to initialize the weights to some initial values. This is known as weights initialization. Weights initialization serves as the starting point for the optimization algorithm.
Like in a race a starting point defines how fast you reach your goal, similarly a good way to initialize the weights is critically important in finding optimal weights earlier.

There are many ways to initialize weights. Most common method among them is ‘random’ initialization of weights to some random values. There is also a more advanced initialization method like Xavier initialization that tries to match the variance of weights & input.

The scenario if all weights are initialized to the same value will result in a failure of the network to learn anything. This will happen as the new weight after update depends upon the previous weight and the error signal received during backpropagation. Because all the weights are similar & error received during backpropagation is also same (layer-wise), the corresponding updates will result in new weights with the same value. Thus over repeated epochs, learned weights will always remain the same, hence unable to learn.

---

5. [7] In a CNN, if the input size 5×5 and the filter size is 7×7, then what would be the size of the output?

Ans.5.
The size of the post-CNN output is given by the following formula: $O = [(I + 2p - k)/s] + 1$

Where ‘I’ is the input size, ‘p’ is the padding, ‘k’ is the kernel size & ‘O’ is the output size.

By putting these values in the formula, we get: $O = [(5 + 0 - 7)/1] + 1 = -1$
Hence, as $O = -1$,which is not possible.

The mentioned CNN operation of 7x7 filter on an input of 5x5 is not possible as the calculated O is negative.

---

6. [10] Show that maximizing likelihood for a multiclass neural network model in which the network outputs have interpretation

$$y_k(x, w) = p(t_k = 1|x)$$

is equivalent to minimization of cross entropy error function.

Ans.6.

For 'k' separate binary classification, we know that conditional distribution:

$$p(\mathbf{t} \mid \mathbf{x}, \mathbf{w}) = \prod_{k=1}^{K} y_k(\mathbf{x}, \mathbf{w})^{t_k} \left[1 - y_k(\mathbf{x}, \mathbf{w})\right]^{1-t_k}$$

However, for given assumptions & settings of $y_k(x, w) = p(t_k = 1|x)$, the previous equation becomes:

$$p(\mathbf{t} \mid \mathbf{x}, \mathbf{w}) = \prod_{k=1}^{K} y_k(\mathbf{x}, \mathbf{w})^{t_k}$$

For 'N' samples points, the equation becomes:

$$p(\mathbf{t} \mid \mathbf{x}, \mathbf{w}) = \prod_{n=1}^{N} \prod_{k=1}^{K} y_k(\mathbf{x}, \mathbf{w})^{t_k}$$

Using previous result, & taking the negative logarithm of the corresponding likelihood function then gives the following cross-entropy error function:

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{kn} \ln y_k\left(\mathbf{x}_n, \mathbf{w}\right)$$

Hence, maximizing likelihood for a multiclass neural network model in which the network outputs have interpretation $y_k(x, w) = p(t_k = 1|x)$ is equivalent to minimization of cross entropy error function.

---

7. [10] Show that as a consequence of the symmetry of the Hessian matrix H, the number of independent elements in the quadratic error function below (which is the Taylor series expansion of the error term $E(\mathrm{w})$) around a point w_0 is given by $W(W + 3)/2$. $E(\mathrm{w}) \approx E(\mathrm{w}_0) + (\mathrm{w} - \mathrm{w}_0)^T\mathrm{b} + 1/2(\mathrm{w} - \mathrm{w}_0)^T\mathrm{H}(\mathrm{w} - \mathrm{w}_0)$

Ans.7.
Given $E(\mathrm{w}) \approx E(\mathrm{w}_0) + (\mathrm{w} - \mathrm{w}_0)^T\mathrm{b} + 1/2(\mathrm{w} - \mathrm{w}_0)^T\mathrm{H}(\mathrm{w} - \mathrm{w}_0)$

Using concepts & formulas of permutations & combinations, we can easily get the number of parameters of a Symmetric Hessian matrix. It forms a pattern of summation of integers from 1 to W, where W is the dimension. Remember the matrix is symmetric. Thus, a symmetric Hessian matrix should have W(W +1)/2 parameters. Adding W additional parameters of 'bias' to it, will get us the total number of parameters as W(W+1)/2 + W = W(W+3)/2

In the quadratic approximation to the error function, the error surface is specified by the quantities b and H, which contain a total of W(W + 3)/2 independent elements (because the matrix H is symmetric), where W is the dimensionality of w (i.e., the total number of adaptive parameters in the network).

---

8. [15] BONUS!! Consider a two-layer network (refer to Slide 11, Neural Network for the network definition)in which the hidden unit nonlinear activation functions g(·) are given by logistic sigmoid functions of the form $\sigma(a) = (1 + exp(-a))^{-1}$. Show that there exists an equivalent network, which computes exactly the same function, but with hidden unit activation functions given by *tanh*(*a*) (the definition of the function is defined in Slide 10, Neural Network). Hint: first find the relation between $\sigma(a)$ and *tanh*(*a*), and then show that the parameters of the two networks differ by linear transformations.

Ans.8.

We know that,

$$\sigma(a) = \frac{1}{1+e^{-a}}$$

Also,

$$\begin{aligned}
\tanh(a) &= \frac{e^a - e^{-a}}{e^a + e^{-a}} \\
&= -1 + \frac{2e^a}{e^a + e^{-a}} \\
&= -1 + 2\frac{1}{1+e^{-2a}} \\
&= 2\sigma(2a) - 1
\end{aligned}$$

Parameters of hidden units with logistic sigmoid activation function: $w_i^{(L1-sig)}, w_0^{(L1-sig)}$ and $w_i^{(L2-sig)}, w_0^{(L2-sig)}$

Parameters of hidden units with Tanh activation function: $w_i^{(L1-tan)}, w_0^{(L1-tan)}$ and $w_i^{(L2-tan)}, w_0^{(L2-tan)}$

Now calculating the standard output of a given hidden layer with tanh activation function will give us equivalent equation demanded in question:

For ‘tanh’ activation:

$$a_k^{(L2-tan)} = \sum_{i=1}^{N} w_{ki}^{(L2-tan)} \tanh\left(a_j^{(L1-tan)}\right) + w_{k0}^{(L2-tan)}$$

$$= \sum_{i=1}^{N} 2w_{ki}^{(L2-tan)} \sigma\left(2a_i^{(L1-tan)}\right) + \left[-\sum_{i=1}^{N} w_{ki}^{(L2-tan)} + w_{k0}^{(L2-tan)}\right]$$

Now to prove that parameters of the two networks differ by linear transformations, we’ll get the output for the sigmoid activation function:

For sigmoid activation:

$$a_k^{(L2-sig)} = \sum_{i=1}^{N} w_{ki}^{(L2-sig)} \sigma\left(a_i^{(L1-sig)}\right) + w_{k0}^{(L2-sig)}$$

Now equating both the equation, using the calculated equivalence relationship b/w sigmoid & tanh activation function in the beginning, we’ll get the following:

$$a_k^{(L2-sig)} = 2a_k^{(L2-tan)}$$
$$w_{ki}^{(L2-sig)} = 2w_{ki}^{(L2-tan)}$$
$$w_{k0}^{(L2-sig)} = -\sum_{i=1}^{N} w_{ki}^{(L2-tan)} + w_{k0}^{(L2-tan)}$$

Using these results, one can easily observe that the above the parameters of the two networks differ by linear transformations.