Parisha Joshi
parishamaheshj18@vt.edu

Assignment 2

ECE 6604
November 4, 2019

Problem 1

In order to implement the backpropagation algorithm, you need to obtain the derivatives of the loss function with respect to all the weight parameters for this problem setup. This is the objective of this part. Using the notations in the handout, please show that the derivatives of the loss function (corresponding to the ith training example) with respect to the weight parameters are given by

$$\frac{\partial L^{(i)}}{\partial w^{(0)}} = (g(a_0) - y_i) \begin{bmatrix} \beta_0 1 * g'(a_1^{(1)}) * x \\ \beta_0 2 * g'(a_2^{(1)}) * x \\ ... \\ \beta_0 C * g'(a_C^{(1)}) * x \end{bmatrix} \tag{1}$$

$$\frac{\partial L^{(i)}}{\partial w^{(1)}} = (g(a_0) - y_i) * \bar{h}^{(1)^T} \tag{2}$$

Solution:

For solving the deep neural network architecture, Following are the basic equations that we have.

$$\textit{Prediction for Output neutron,} \tag{3}$$
$$\hat{y} = g(a_0) \tag{4}$$
$$= \frac{1}{1 + e^{-a_0}}; \tag{5}$$
$$\textit{Activation for output neutron,} \tag{6}$$
$$a_0 = \beta_{01}h_1 + \beta_{02}h_2 + \beta_{03}h_3 + ... + \beta_{0C}h_C + \beta_{outbias}; \tag{7}$$
$$\textit{Hypothesis for hidden layer neutron,} \tag{8}$$
$$h_1 = g(a^{(0)}) \tag{9}$$
$$= \frac{1}{1 + e^{-a_0^{(0)}}}; \tag{10}$$
$$\textit{Activation for hidden layer neutron,} \tag{11}$$
$$a_0^{(0)} = \beta_1 * 1 + \beta_2 * x_2 + \beta_3 * x_3; \tag{12}$$
$$\textit{Cost function used for classification problem,} \tag{13}$$
$$L = -(y_i * log(\hat{y}_i)) - (1 - y_i)log(1 - \hat{y}_i); \tag{14}$$

Now, We want to find the partial derivative of cost function with respect to output weights.

Total out put weights are C + 1. Here an additional weight is for the output bias term. We want to find out only the derivative terms w.r.t non-bias weights.

Applying the chain rule,

$$\frac{\partial L}{\partial \beta^{out}} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial \beta^{out}}; \tag{15}$$

$$\frac{\partial L}{\partial \beta^{out}} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial a_0} * \frac{\partial a_0}{\partial \beta^{out}}; \tag{16}$$

Taking the derivatives of the equations : (4),(7) and (14), We get,

$$\frac{\partial L}{\partial \hat{y}} = \frac{-y + \hat{y}}{\hat{y}(1-\hat{y})}; \tag{17}$$

$$\frac{\partial \hat{y}}{\partial a_0} = \hat{y} * (1 - \hat{y}) \tag{18}$$

$$\textit{Because Derivative of } \sigma = \sigma(1-\sigma); \tag{19}$$

$$\frac{\partial a_0}{\partial \beta_1^{out}} = h_1, \frac{\partial a_0}{\partial \beta_1^{out}} = h_2, ..., \frac{\partial a_0}{\partial \beta_C^{out}} = h_C \tag{20}$$

$$\tag{21}$$

Because Derivative cost function w.r.t $\beta_1 = h_1$*, Accordingly change the equation for all the Beta till C.Replacing results of (17),(18) and (20) in equation (16), we get..* ;

$$\frac{\partial L}{\partial \beta^{out}} = \frac{-y + \hat{y}}{\hat{y}(1-\hat{y})} * \hat{y} * (1 - \hat{y}) * \begin{bmatrix} 1 \\ h_1 \\ h_2 \\ . \\ . \\ . \\ h_C \end{bmatrix}; \tag{22}$$

$$\frac{\partial L}{\partial \beta^{out}} = (\hat{y} - y) * \begin{bmatrix} 1 \\ h_1 \\ h_2 \\ . \\ . \\ . \\ h_C \end{bmatrix}; \tag{23}$$

$$\tag{24}$$

Hence,taking $\beta^{out} = w^{(1)}$ and $\hat{y} = g(a_0)$,Equation (2) is proved

$$\boldsymbol{\frac{\partial L^{(i)}}{\partial w^{(1)}} = (g(a_0) - y_i) * \bar{h}^{(1)^T}} \tag{25}$$

Now lets prove equation 1 Before deriving Equation 1, I have updated the output weights with following equation.

$$\beta_{out} = \beta_{out} - \alpha * \frac{\partial L}{\partial w^{(1)}} \tag{26}$$

Now,Derivative of cost function w.r.t input β values:

$$\frac{\partial L}{\partial \beta^{(0)}} = \frac{\partial L}{\partial a_0^{(0)}} * \frac{\partial a_0^{(0)}}{\partial \beta^{(0)}}; \tag{27}$$

$$= \frac{\partial L}{\partial h_1} * \frac{\partial h_1}{\partial a_0^{(0)}} * \frac{\partial a_0^{(0)}}{\partial \beta^{(0)}}; \tag{28}$$

$$= \frac{\partial L}{\partial a_0} * \frac{\partial a_0}{\partial h_1} * \frac{\partial h_1}{\partial a_0^{(0)}} * \frac{\partial a_0^{(0)}}{\partial \beta^{(0)}}; \tag{29}$$

$$= \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial a_0} * \frac{\partial a_0}{\partial h_1} * \frac{\partial h_1}{\partial a_0^{(0)}} * \frac{\partial a_0^{(0)}}{\partial \beta^{(0)}}; \tag{30}$$

$$= \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} * \hat{y}(1-\hat{y}) * \beta^{(C*1matrix)} * h^{(C*1matrix)}(1 - h^{(C*1matrix)}) * \begin{bmatrix} 1 & x_1 & x_2 \\ 1 & x_1 & x_2 \\ ..... & & \\ 1 & x_1 & x_2 \end{bmatrix} \tag{31}$$

$$= \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} * \hat{y}(1-\hat{y}) * \begin{bmatrix} \beta_{01} \\ \beta_{02} \\ . \\ . \\ . \\ \beta_{0C} \end{bmatrix} * \begin{bmatrix} h_1 \\ h_2 \\ . \\ . \\ . \\ h_C \end{bmatrix} (1 - \begin{bmatrix} h_1 \\ h_2 \\ . \\ . \\ . \\ h_C \end{bmatrix}) \begin{bmatrix} 1 & x_1 & x_2 \\ 1 & x_1 & x_2 \\ ..... & & \\ 1 & x_1 & x_2 \end{bmatrix} \tag{32}$$

$$\frac{\partial L^{(i)}}{\partial \beta^{(0)}} = (\hat{y} - y_i) \begin{bmatrix} \beta_0 1 * h' * x \\ \beta_0 2 * h' * x \\ ... \\ \beta_0 C * h' * x \end{bmatrix} \tag{33}$$

Hence, The equation 1 is proved.

Problem 2

Write a function which performs the feedforward operation. In other words, the inputs to this function should be: i) the weight parameters of the neural network, and ii) a training example, while the outputs of this function are: i) the response vector at the hidden layer, and ii) the response variable at the output layer.

Solution:
Matlab file Q1.m and Q2.m both contains the function feedfwd(), Which takes inputs as the weight parameters of the neural network, and training example one by one. And gives an output containing the hidden layer hypothesis vector and the final neuron output. Which is the response of out model for one example.

The operations happening under this function is pretty straight forward.

$$activation = \beta^T * X; \tag{34}$$

$$Hypothesisvector = \sigma(activation); \tag{35}$$

$$\bar{h} = \begin{bmatrix} 1 \\ Hypothesisvector \end{bmatrix} \tag{36}$$

$$finalactivation = \beta_{out} * \bar{h}; \tag{37}$$

$$\hat{y} = \sigma(finalactivation); \tag{38}$$

$$hidden_vector = \bar{h}; \tag{39}$$

return hidden vector and prediction

Problem 3

Write a function which evaluates the loss function value corresponding to a single training example and its derivatives with respect to the weight parameters. The inputs to this function are: i) the weight parameters of the neural network, and ii) a training example, while the outputs of this function are: i) the value of the loss function, and ii) the derivatives of the loss function with respect to the weight parameters.

Solution:

Both the submitted matlab files have a function loss-func(). Which takes Beta and alpha parameters and input example as input and returns loss value and derivatives of the cost function with respect to the beta parameters.

Pseudocode is given below.

1) Find the predicted output with feed forward function wrote in Question 2.
2) Loss value is found from the function $-(y * log(\hat{y}) - (1-y) * log(1-\hat{y}))$
3) Find the derivative of loss function wrt output weights and input weights using equations from question 1.
4) Return Loss value and derivative values.

Problem 4

Now you are going to implement the back propagation algorithm using the functions you created in parts 2 and 3. In this part, consider that 1 $\leq$ C $\leq$ 20. For each network architecture, i.e., for each value of C, state the training accuracy. It is important to randomly initialize the weight parameters to small values for breaking symmetry. One way of doing so is to randomly select each weight parameter in the l^{th} layer uniformly in the range $\left[-\psi^l, \psi^l\right]$, where $\psi^l = \sqrt{\frac{6}{n(l)+n(l+1)}}$ Notice that the value of $\psi(1)$ depends on which layer the weight parameter belongs to. Clearly state your choices for the learning rate and the number of iterations in the gradient descent algorithm.

Solution:
In the matlab file Q1.m the Beta matrices i.e. Beta 1 which is 3*C and Beta 2 which is (C+1)*1, are initiated randomly with rand function.Moreover, The range is between $-\psi$ to $+\psi$ .As we only have one hidden layer the range would be : $-\sqrt{\frac{6}{C+1}}$ to $+\sqrt{\frac{6}{C+1}}$.

I have kept the learning rate to 10^-2 and number of iterations to 2500. According to different experiments with different combinations that I performed, this combination turned out to produce faster convergence with high accuracy.

Problem 5

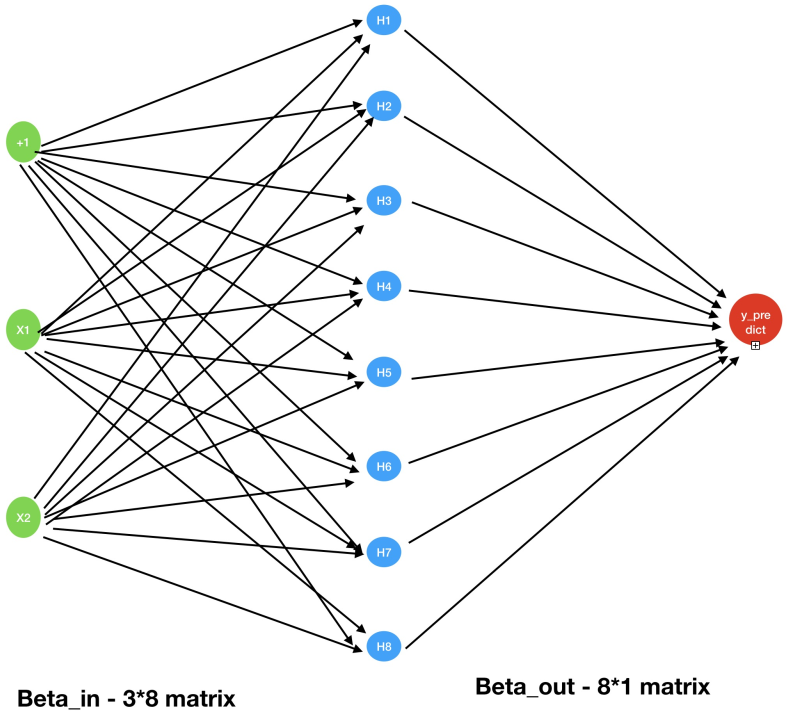
Use the validation dataset N-valdata.txt to select the most appropriate network architecture from amongst the ones that you trained in part 4. Justify your answer for full credit.

Solution:
After trying various combinations of the parameters and variables, The model given in Q2.m is the best model for our dataset.

I saved the Best Beta initialization matrix for the purpose of comparing different learning rate and iterations. For this purpose, The submitted folder includes Beta0.mat and Beta-out0.mat files. Which are essential to run the Q2.m script.

Best Parameters:
Number of hidden neurons = 8
Number of iterations = 2500
Best learning rate = 10^-2

Figure 1: Problem 1



Problem 6

Reading Assignment: In the last part of this project, you are going to read the following paper: https://arxiv.org/abs/1412.6980. This paper introduced a new algorithm for stochastic optimiza- tion called Adam. Based on your reading, please summarize the key differences between Adam and the stochastic gradient descent (SGD) algorithm.

Solution:

Stochastic Gradient Descent algorithm performs weight updation for every example in the model training. The process is faster and converges to local minima. But the method involves a hugh variance in each updation and the cost function varies frequently with huge changes. The method suggested in the give paper ”Adam: A method for stochastic optimization” tries to get rid of the huge changes that occurs in the weight parameters.As stated in the paper by Diederik P. Kingma and Jimmy Ba,In addition to storing an exponentially decaying average of past squared gradients like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients, similar to momentum. We compute the decaying averages of past and past squared gradients mt and vt respectively as follows:

$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$
$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$

Where mt and vt are first and second moment estimates.The initial values of mt and vt are vectors of 0's.The initial $\beta_1 and \beta_2$ are 1's. The corrected biases are calculated using this following equations:
$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$
$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$

The weight updates happen with,
$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}$
The major difference between these two algorithms are that in Adam makes use of both the AdaGrad and RMS prop algorithms, which are based on momentum parameters.

MATLAB files

1) Q1.m : Involves both functions for feed forward and loss functon. Calculates accuracy for 1 to 20 values of C, for both testing and training.Generates initial weights randomly every time. So results will be different in every run.

2) Q2.m : calculates testing and training accuracy for fixed values of initial beta parameters, $C = 8$ and iteration = 2500.Results will remain the same.

3)Beta0.mat and Beta-out0.mat files: Best initial weights saved for use of Q2.m file.