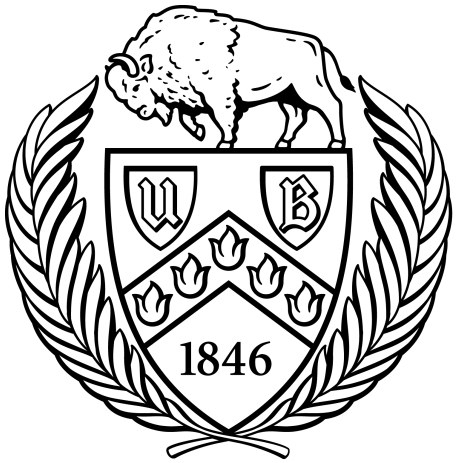
University at Buffalo,
The State University of New York

CSE 676

Deep Learning

Assignment - 1

Yash Narendra Saraf (50290453)



CSE 676: Deep Learning

Assignment - 1

Yash Narendra Saraf
UB ID: 50290453
ysaraf@buffalo.edu

March 15, 2019

Q1 Softmax [2 points]

(1) [1 point] Prove that softmax is invariant to constant sifts in the input, i.e., for any input vector x and a constant scalar c, the following holds:

$$softmax(x) = softmax(x + c)$$

where $softmax(x)_i \triangleq \frac{e^{x_i}}{\sum_{i\prime} e^{x_i^{'}}}$, **and** $x + c$ **means adding c to every dimension of x.**

The definition of Softmax is -

$$softmax(x)_i \triangleq \frac{e^{x_i}}{\sum_{i\prime} e^{x_i^{'}}},$$

Now, we have to prove that even after constant shifts the output of the softmax won't change i.e.

$$softmax(x) = softmax(x + c)$$

So, replacing x by $x + c$ in the original softmax equation, we have

$$\begin{aligned} softmax(x)_i &\triangleq \frac{e^{x_i}}{\sum_{i\prime} e^{x_i^{'}}} \\ &= \frac{e^{x_i+c}}{\sum_{i\prime} e^{x_i^{'}+c}} \\ &= \frac{e^c * e^{x_i}}{e^c * \sum_{i\prime} e^{x_i^{'}}} \\ &= \frac{e^{x_i}}{\sum_{i\prime} e^{x_i^{'}}} \end{aligned}$$

Hence Proved, softmax function is invariant to constant shifts.

(2) [1 point] Let $z = Wx + c$**, where W and c are some matrix and vector, respectively. Let**

$$J = \sum_i log\, softmax(z_i)$$

Calculate the derivatives of J w.r.t. W and c, respectively, i.e., calculate $\frac{\partial J}{\partial W}$ **and** $\frac{\partial J}{\partial c}$**.**

For the calculation of $\frac{\partial J}{\partial z}$,

$$\frac{\partial J}{\partial z} = -\frac{\partial}{\partial z} \sum_i log\, softmax(z_i)$$

Calculating the derivative by considering the derivative while doing calculations for Class i. This is necessary as when we use chain rule to calculate derivatives we will have different derivative for the class under considerations and all other classes. This is because of the derivative of the softmax function.

$$\frac{\partial J}{\partial z} = -\frac{\partial}{\partial z}(log(softmax(z_i))) - \frac{\partial}{\partial z} \sum_{k \neq i} log(softmax(z_k))$$

$$\frac{\partial J}{\partial z} = -\Big(\frac{1}{(softmax(z_i))} \frac{\partial(softmax(z_i))}{\partial z}\Big) - \sum_{k \neq i} y_k \frac{1}{(softmax(z_k))} \frac{\partial(softmax(z_k))}{\partial z}$$

$$\frac{\partial J}{\partial z} = -(\ 1 - (softmax(z_i))) + \sum_{k \neq i} (softmax(z_i))$$

$$\frac{\partial J}{\partial z} = -1 + (softmax(z_i)) + \sum_{k \neq i} (softmax(z_i))$$

$$\frac{\partial J(z)}{\partial z} = -1 + \Big(\sum_{k=1}^{N} 1\Big)\ (softmax(z_i))$$

$$\frac{\partial J(z)}{\partial z} = -1 + \ N(softmax(z_i))$$

Now, we need to calculate the $\frac{\partial J}{\partial W}$ and $\frac{\partial J}{\partial c}$. This can be simply done by applying the chain rule as we know the values for $\frac{\partial J}{\partial z}$ and z.

So, as $z = Wx + c$. We can write,

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial z} \frac{\partial z}{\partial W}$$

$$\frac{\partial J}{\partial W} = (-1 + \ N(softmax(z_i))) \frac{\partial z}{\partial W}$$

$$\frac{\partial J}{\partial W} = (-1 + \ N(softmax(z_i)))\ x$$

and similarly for $\frac{\partial J}{\partial c}$,

$$\frac{\partial J}{\partial c} = \frac{\partial J}{\partial z} \frac{\partial z}{\partial c}$$

$$\frac{\partial J}{\partial c} = (-1 + \ N(softmax(z_i))) \frac{\partial z}{\partial c}$$

$$\frac{\partial J}{\partial c} = (-1 + \ N(softmax(z_i)))$$

Q2 Logistic Regression with Regularization [2 points]

(1) [1 point] Let the data be $(x_i, y_i)_{i=1}^N$ where $x_i \in R^d$ and $y_i \in \{0, 1\}$. Logistic regression is a binary classification model, with the probability of y_i being 1 as:

$$p(y_i; x_i, \theta) = \sigma(\theta^T x_i) \triangleq \frac{1}{1 + e^{-\theta^T x_i}}$$

where θ is the model parameter. Assume we impose an L2 regularization term on the parameter, defined as:

$$R(\theta) = \frac{\lambda}{2}\theta^T\theta$$

with a positive constant λ. Write out the final objective function for this logistic regression with regularization model.

Let the given Hypothesis function be represented as -

$$h_\theta(x_i) = p(y_i = 1|x_i, \theta) = \sigma(\theta^T x_i) \triangleq \frac{1}{1 + e^{-\theta^T x_i}}$$

and the Regularization term given is -

$$R(\theta) = \frac{\lambda}{2}\theta^T\theta$$

The above Hypothesis function gives the probability of predicting 1 for the given input.

So, we can't use the Mean Square Error as the Loss function as it would not give convex function. So to solve this problem the loss function can be designed as following -

$$Cost(h_\theta(x_i), y_i) = -y_i\, log(h_\theta(x_i)) - (1 - y_i)\, log(1 - h_\theta(x_i))$$

Using the above cost function the Objective function can be designed as -

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} Cost(h_\theta(x_i), y_i)$$

The regularization term can be added to the equation, as the function of the regularizer is to make sure that the model does not overfit. It also makes sure that the value of weights remain as small as possible. There is a scaling factor involved with the regularization term to make sure that the regularizer does not hinder the model from learning just to minimize the weights. Let the parameter be λ.

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} Cost(h_\theta(x_i), y_i) \;+\; R(\theta)$$

The above function represents the Objective function for Logistic Regression Model.

(2) [1 point] If we use gradient descent to solve the model parameter. Derive the updating rule for θ . Your answer should contain the derivation, not just the final answer

Now, since we have obtained the Objective function, our goal is to use this function to find optimum weights for the model using Gradient Descent. The gradient descent algorithm tries to find the minima of the Objective function for optimization. Here, the minima can be easily found by derivating the cost function with respect to Weights i.e. θ and updating the weights. Derivating with respect to theta gives us the slope of the graph and we use the slope to travel to the minima of Objective function.

So, we now have following equations -

$$h_\theta(x_i) = p(y_i = 1|x_i, \theta) = \sigma(\theta^T x_i) \triangleq \frac{1}{1 + e^{-\theta^T x_i}}$$

$$R(\theta) = \frac{\lambda}{2}\theta^T\theta$$

$$Cost(h_\theta(x_i), y_i) = -y_i\, log(h_\theta(x_i)) - (1 - y_i)\, log(1 - h_\theta(x_i))$$

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} Cost(h_\theta(x_i), y_i) \;+\; R(\theta)$$

To calculate gradient equation first let's just find derivative of Cost function. We can later substitute the same in the main equation for calculation of finding final gradient.

$$\frac{\partial Cost(h_\theta(x_i), y_i)}{\partial \theta_j} = -\Big(\frac{y_i}{h_\theta(x_i)} - \frac{1 - y_i}{1 - h_\theta(x_i)}\Big)\, \frac{\partial\, h_\theta(x_i)}{\partial\, \theta_j}$$

Now, $h_\theta(x_i)$ is a sigmoid function and the derivative of sigmoid function is given by -

$$\frac{\partial Cost(h_\theta(x_i), y_i)}{\partial x} = h_\theta(x)(1 - h_\theta(x))$$

So, using that updating our Gradient calculation,

$$\frac{\partial Cost(h_\theta(x_i), y_i)}{\partial \theta_j} = -\Big(\frac{y_i}{h_\theta(x_i)} - \frac{1 - y_i}{1 - h_\theta(x_i)}\Big)\; h_\theta(x_i)\; (1 - h_\theta(x_i))\; \frac{\partial\, \theta^T x_i}{\partial\, \theta_j}$$

$$\frac{\partial Cost(h_\theta(x_i), y_i))}{\partial \theta_j} = -(y_i(1 - h_\theta(x_i)) - (1 - y_i)h_\theta(x_i))\ x_{ij}$$

Here, x_{ij} represents the j^{th} weight when working with input x_i.

$$\frac{\partial Cost(h_\theta(x_i), y_i)}{\partial \theta_j} = -(y_i - h_\theta(x_i))\ x_{ij}$$

Now, calculating the complete gradient equation using the above partial derivative.
So,

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{\partial}{\partial \theta_j}\Big(\frac{1}{N}\sum_{i=1}^{N} Cost(h_\theta(x_i), y_i)\ +\ R(\theta)\Big)$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial}{\partial \theta_j}\ Cost(h_\theta(x_i), y_i)\ +\ \frac{\partial}{\partial \theta_j} R(\theta)$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{N}\sum_{i=1}^{N}(y_i - h_\theta(x_i))\ x_{ij}\ +\ \frac{\partial}{\partial \theta_j} R(\theta)$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{N}\sum_{i=1}^{N}(y_i - h_\theta(x_i))\ x_{ij}\ +\ \frac{\partial}{\partial \theta_j} R(\theta)$$

Now, calculating the derivative of Regularization term -

$$\frac{\partial}{\partial \theta_j} R(\theta) = \frac{\partial}{\partial \theta_j}\ \frac{\lambda}{2}\theta^T\theta$$

$$\frac{\partial}{\partial \theta_j} R(\theta) = \frac{\lambda}{2}\ \frac{\partial}{\partial \theta_j}(\theta^T\theta)$$

$$\frac{\partial}{\partial \theta_j} R(\theta) = \frac{\lambda}{2}\ (2\ \theta_j)$$

$$\frac{\partial}{\partial \theta_j} R(\theta) = \lambda\ \theta_j$$

Substituting, the same in the gradient equation.

$$\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{N}\sum_{i=1}^{N}(y_i - h_\theta(x_i))\ x_{ij}\ +\ \lambda\ \theta_j$$

So, weight updation equation will look like,

$$\theta_j = \theta_j - \alpha\ \frac{\partial J(\theta)}{\partial \theta_j}$$

$$\theta_j = \theta_j - \alpha\ (-\frac{1}{N}\sum_{i=1}^{N}(y_i - h_\theta(x_i))\ x_{ij}\ +\ \lambda\ \theta_j)$$

$$\theta_j = \theta_j - \alpha\ (\frac{1}{N}\sum_{i=1}^{N}(h_\theta(x_i) - y_i)\ x_{ij}\ +\ \lambda\ \theta_j)$$

The constants can be combined and the final equation looks like -

$$\theta_j = \theta_j\Big(1 - \frac{\lambda}{N}\Big) - \alpha\ (\sum_{i=1}^{N}(h_\theta(x_i) - y_i)\ x_{ij})$$

In, the above equation the λ represents the regularization constant and α represents the learning rate.

Q3 Derivative of the Softmax Function [3 points]

(1) [1 point] Define the loss function as

$$J(z) = -\sum_{k=1}^{K} y_k \, log\widetilde{y}_k$$

where $\widetilde{y}_k = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}}$, and $(y_1, .., y_K)$ is a known probability vector. Derive the $\frac{\partial J(z)}{\partial z}$. Note $z = (z_1, .., z_K)$ is a vector so $\frac{\partial J(z)}{\partial z}$ is in the form of a vector. Your answer should contain the derivation, not just the final answer.

For the calculation of $\frac{\partial J(z)}{\partial z}$,

$$\frac{\partial J(z)}{\partial z} = -\frac{\partial}{\partial z} \sum_{k=1}^{K} y_k \, log\widetilde{y}_k$$

Calculating the derivative by considering the derivative while doing calculations for Class i. This is necessary as when we use chain rule to calculate derivatives we will have different derivative for the class under considerations and all other classes. This is because of the derivative of the softmax function.

$$\frac{\partial J(z)}{\partial z} = -\frac{\partial}{\partial z}(y_i \; log(\widetilde{y}_i)) - \frac{\partial}{\partial z} \sum_{k \neq i} y_k \, log\widetilde{y}_k$$

$$\frac{\partial J(z)}{\partial z} = -\Big(y_i \; \frac{1}{\widetilde{y}_i} \; \frac{\partial y_i}{\partial z}\Big) - \sum_{k \neq i} y_k \; \frac{1}{\widetilde{y}_k} \frac{\partial \widetilde{y}_k}{\partial z}$$

$$\frac{\partial J(z)}{\partial z} = -(y_i \; (1 - \widetilde{y}_i)) + \sum_{k \neq i} y_k \; \widetilde{y}_i$$

$$\frac{\partial J(z)}{\partial z} = -y_i + y_i \; \widetilde{y}_i + \sum_{k \neq i} y_k \; \widetilde{y}_i$$

$$\frac{\partial J(z)}{\partial z} = -y_i + \; \Big(\sum_{k=1}^{N} y_k\Big) \; \widetilde{y}_i$$

$$\frac{\partial J(z)}{\partial z} = -y_i + \; \widetilde{y}_i$$

(2) [1 point] Assume the above softmax is the output layer of an FNN. Briefly explain how the derivative is used in the back propagation algorithm.

In the above answer, the J(z) function given is the Cross Entropy Loss function. So we found the derivative of the Cross Entropy with respect to the output of the final layer i.e. $\frac{\partial J(z)}{\partial z}$ as $z = W^T \; h + b$. So, now we know gradient for the last layer.

Now, to when we update the value of each weights i.e. W_{ij}, where W is the node from Node i to Node j, we need the value of $\frac{\partial J(z)}{\partial W_{ij}}$, so to calculate which there is a need to propagate the error calculated at the last layer, and use chain rule in Calculus to propagate the error backwards..

The $\frac{\partial J(z)}{\partial W_{ij}}$ is calculated for each weight and all weights are updated for that particular iteration.

The property of the loss function for softmax regression is that it only penalizes the output from the correct class. But when we calculate the gradient equation it can be seen that all the the incorrect classes also get affected based on how incorrect those classes are.

(3) [1 points] Let $z = W^T h + b$, where W is a matrix, b and h are vectors. Use the chain rule to calculate the gradient of W and b, i.e., $\frac{\partial J}{\partial W}$ and $\frac{\partial J}{\partial b}$, respectively

In the first part we calculated the following,

$$\frac{\partial J(z)}{\partial z} = -y_i + \widetilde{y}_i$$

.

Now, if we need to calculate $\frac{\partial J}{\partial W}$ and $\frac{\partial J}{\partial b}$, we can simply use $z = W^T h + b$ and update based on this.
So,

$$\frac{\partial J(z)}{\partial W} = \frac{\partial J(z)}{\partial z} \frac{\partial z}{\partial W}$$

$$\frac{\partial J(z)}{\partial W} = (-y_i + \widetilde{y}_i) \frac{\partial z}{\partial W}$$

$$\frac{\partial J(z)}{\partial W} = (-y_i + \widetilde{y}_i) \frac{\partial (W^T h + b)}{\partial W}$$

$$\frac{\partial J(z)}{\partial W} = (-y_i + \widetilde{y}_i) \frac{\partial (W^T h + b)}{\partial W}$$

$$\frac{\partial J(z)}{\partial W} = (-y_i + \widetilde{y}_i)\ h$$

and similarly for $\frac{\partial J}{\partial b}$,

$$\frac{\partial J(z)}{\partial b} = (-y_i + \widetilde{y}_i) \frac{\partial (W^T h + b)}{\partial b}$$

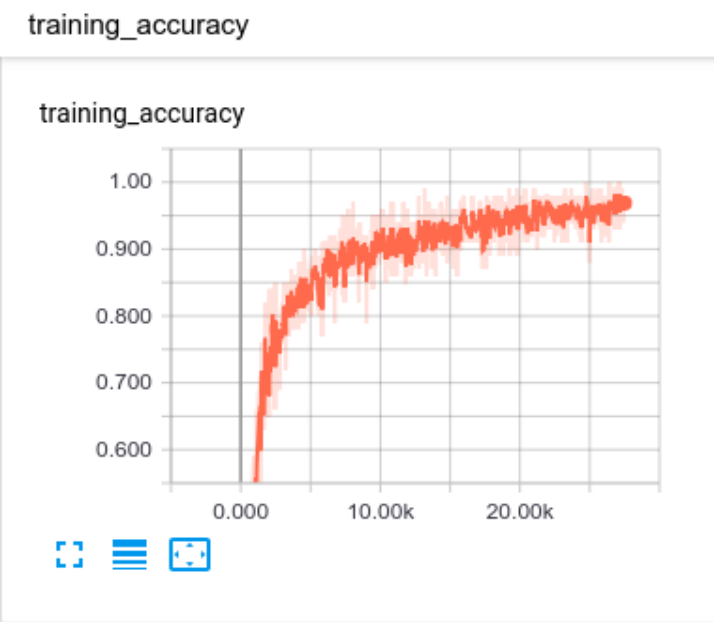
$$\frac{\partial J(z)}{\partial b} = (-y_i + \widetilde{y}_i)$$

Q4 MNIST with FNN [3 points]

(1) [3 points] Design an FNN for MNIST classification. Implement the model and plot two curves in one figure: i) training loss vs. training iterations; ii) test loss vs. training iterations.
– You can use code from websites. However, you must reference (cite) the code in your answer.
– Submission includes the plot of the two curves and the runnable code (with a ReadMe file containing instructions on how to run the code).

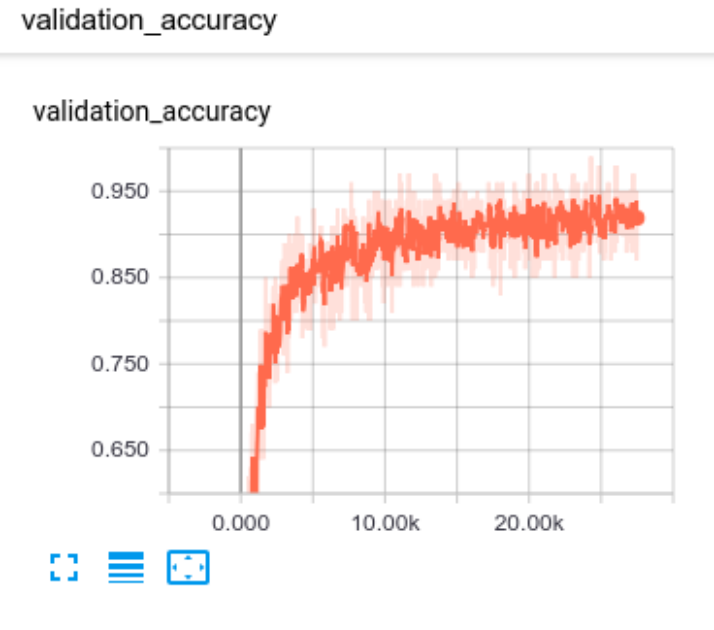
For this question code can be found in the submitted folder directory. The ReadMe has also been added for reference.
The name of the file is **Q4_MNIST_FNN.ipynb**
The code is written inside jupyter notebook. Also for reference the tensorflow tutorials slides were used along with this GitHub Repository.

The result graphs for the code can be found below -

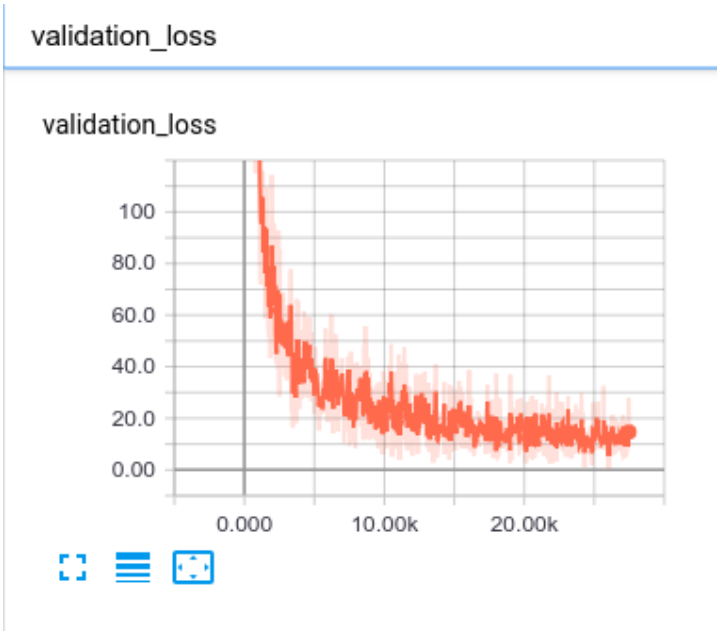


Training Accuracy Graph

Training Cost Graph



Validation Accuracy Graph



Validation Cost Graph