IE 534: Deep Learning, Fall 2018

Homework 1: Implementing a Neural Network on MNIST

Lecturer: Justin Sirignano *Submitted by: Rachneet Kaur, Sept. 7th, 2018*

Implementation

The objective is to **classify** the input 28*28 dimensional image of a digit into the corresponding one of the 10 distinct digit classes namely $\{0, 1, ..., 9\}$. The MNIST dataset has 60000 training images and 10000 images in the test set of dimensions 28*28.
Hence, we have:

- Dimensions of training set input X = (60000, 784) and labels Y = (60000, 1)
- In my code, I store training set in x_train and y_train.
- Dimensions of test set input X = (10000, 784) and labels Y = (10000, 1)
- In my code, I store training set in x_test and y_test.

The system in equation (1) represents a fully connected neural network with a single hidden layer.

$$\begin{aligned} Z &= Wx + b^1 \\ H_i &= \sigma(Z_i) \\ U &= CH + b^2 \\ f(x, \theta) &= F_{softmax}(U) \end{aligned} \tag{1}$$

Note that in order to perform 10 class classification, a softmax layer is added as the last layer of the neural network. Equation (1) is called as the **forward propagation or forward pass** in the training of a neural network.

Consider the steps implemented for the Stochastic Gradient Descent algorithm:

Step 1: Random Initialization of model parameters

The model parameters for a single layer neural network are $\{W, b^1, C, b^2\}$.

- For implementation, I used the $d_H = 140$ hidden units in the hidden layer H.
- Since this is a $k = 10$ class classification problem, I have 10 units in the output layer.

First, we randomly initialize the following:

- W = $d_H \times d$ matrix = 140×784 matrix
- $b^1 = d_H \times 1$ matrix = 140×1 matrix
- C = $k \times d_H$ matrix = 10×140 dimensional matrix
- $b^2 = k \times 1$ matrix = 10×1 matrix.

The forward, backward propagation and parameter update steps in SGD algorithm (Steps 1, 2 and 3 below) are repeated 60000 (number of rows in training set) times in each epoch. Using these 60000 iterations, the training accuracy is calculated in each epoch as mentioned in Step 2.

- For implementation, I trained the model for 15 epochs.

Step 2: Forward Step

Next, we randomly select a data sample (X, Y) from the training set and compute Z, H, U and $f(x, \theta)$ using the equations in 1. The dimensions of Z is $d\_H \times 1 = 140 \times 1$, H is $d\_H \times 1 = 140 \times 1$ and U is $k \times 1 = 10 \times 1$.

- For implementation in Python, I used the activation function as $\sigma(z) = tanh(z)$. The activation function is element wise applied to each element of the hidden layer.

Next, we calculate the predicted label as argmax($f(x, \theta)$) for the training sample which is compared with true label y to compute the accuracy of the training set.

- I wrote a function $forward(x, y, model)$ in my code to compute the forward step of the model. It returns the calculated Z, H and probability distribution of the 10 classes.
- $softmax\_function(z)$ function in my code computes the softmax function for the output layer.
- $activation(z, derivative = 0)$ function in my code computes the tanh(z) and it's derivative $1 - tanh^2(z)$ if derivative flag = 1) activation for the hidden layers.
- model = {} is the dictionary in my code containing the model parameters, namely, model['W'], model['b1'], model['C'] and model['b2'].

Step 3: Gradient Calculation

Equation (2) defines the negative log likelihood objective function to be minimized.

$$\begin{aligned} L(\theta) &= E_{(X,Y)}(\varrho(f(x,\theta),y)) \\ \varrho(v,y) &= -\sum_{k=0}^{K-1}(\mathbb{1}_{\{y-k\}}\ log\ v_k) \end{aligned} \tag{2}$$

We use Stochastic Gradient Descent to minimize the $\bigtriangledown_\theta \varrho$ = - $\bigtriangledown_\theta log(F_{softmax})$. Let us define $\bigtriangledown_\theta log(F_{softmax}) = \bigtriangledown_\theta \rho$.
To update the parameter values using SGD, we compute the derivatives of the parameters namely $\frac{\partial \rho}{\partial b^2}$, $\frac{\partial \rho}{\partial C}$, $\frac{\partial \rho}{\partial b^1}$ and $\frac{\partial \rho}{\partial W}$ using the equations in 3.

$$\begin{aligned} \frac{\partial \rho}{\partial U} &= e(y) - f(x,\theta) \\ \frac{\partial \rho}{\partial b^2} &= \frac{\partial \rho}{\partial U} \\ \frac{\partial \rho}{\partial C} &= \frac{\partial \rho}{\partial U}H^T \\ \delta &= C^T\frac{\partial \rho}{\partial U} \\ \frac{\partial \rho}{\partial b^1} &= \delta \cdot \sigma'(Z) \\ \frac{\partial \rho}{\partial W} &= (\delta \cdot \sigma'(Z))X^T \end{aligned} \tag{3}$$

where e(y) = $\{\mathbb{1}_{\{y=0\}}, ..., \mathbb{1}_{\{y=K=10\}}\}$
Equation (3) defines the gradients of the model parameters.
The dimensions of $\frac{\partial \rho}{\partial b^2}$, $\frac{\partial \rho}{\partial C}$, $\frac{\partial \rho}{\partial b^1}$ and $\frac{\partial \rho}{\partial W}$ are same as dimensions of b^2, C, b^1, W in Step 1.

- *model_grads* = {} is the dictionary in my code containing the gradients of the model parameters, namely, model_grads['b2'], model_grads['C'], model_grads['b1'] and model_grads['W'].

- I wrote a function $backward(x, y, Z, H, prob\_dist, model, model\_grads)$ in my code to compute the gradients of the model parameters. It returns the dictionary model_grads.

- For implementation, the derivative of activation function as $\sigma'(z) = 1 - tanh^2(z)$. The derivative of activation function is element wise.

Step 4: Parameter Update

- For implementation, the learning rate $\alpha^{(l)}$ is defined based on no. of epoch (l) we are training at: $\alpha^{(l)} = \begin{cases} 0.01, & \text{if epoch } \ l \leq 5 \\ 0.001, & \text{if epoch } l > 5 \\ 0.0001, & \text{if epoch } l > 10 \\ 0.00001, & \text{if epoch } l > 15 \end{cases}$

For each iteration t out of 60000 iterations, in each epoch l, we update the parameters of the neural network model $\{W, b^1, C, b^2\}$ using the stochastic gradient descent update as below:

$$\begin{aligned} W^{(t)} &= W^{(t)} + \alpha^{(l)} \frac{\partial \rho}{\partial W^{(t)}} \\ b^{1(t)} &= b^{1(t)} + \alpha^{(l)} \frac{\partial \rho}{\partial b^{1(t)}} \\ C^{(t)} &= C^{(t)} + \alpha^{(l)} \frac{\partial \rho}{\partial C^{(t)}} \\ b^{2} &= b^{2(t)} + \alpha^{(l)} \frac{\partial \rho}{\partial b^{2(t)}} \end{aligned} \tag{4}$$

where the parameter values are calculated in forward step in Step 1 and derivatives are calculates in backward step in Step 2.
A snapshot of the training and test accuracies is given in Figure 1.
Note that the accuracy on the training set after 7 epochs reaches approx. 0.994, but clearly since the test accuracy is 0.979, we can notice that the model starts to **overfit** the training set.

Accuracy

- The accuracy of the model on the test set = 97.9%
- The estimated time to train the model for 15 epochs = 903 seconds.

I wrote a function *compute_accuracy*() in my code to compute the accuracy of the test set. I predict the label based on each element in test set and check if the true label is same as the predicted label to calculate the accuracy.

```
time2 = time.time()
print('Estimated time to train the model = ', time2-time1, ' seconds')
```

```
In epoch  0 , accuracy in training set =  0.92375
In epoch  1 , accuracy in training set =  0.9660666666666666
In epoch  2 , accuracy in training set =  0.9765
In epoch  3 , accuracy in training set =  0.98265
In epoch  4 , accuracy in training set =  0.9856
In epoch  5 , accuracy in training set =  0.9890166666666667
In epoch  6 , accuracy in training set =  0.9937
In epoch  7 , accuracy in training set =  0.9954833333333334
In epoch  8 , accuracy in training set =  0.9956
In epoch  9 , accuracy in training set =  0.9963166666666666
In epoch  10 , accuracy in training set =  0.99685
In epoch  11 , accuracy in training set =  0.997
In epoch  12 , accuracy in training set =  0.99695
In epoch  13 , accuracy in training set =  0.9972833333333333
In epoch  14 , accuracy in training set =  0.9976666666666667
Estimated time to train the model =  903.1648941040039  seconds
```

Test accuracy of the model

```
test_accuracy = compute_accuracy(x_test, y_test, model)
print('Accuracy in testing set =', test_accuracy)
```

```
Accuracy in testing set = 0.979
```

Figure 1: Results