Selected Topics From CS: Assignment 2

Anirudh Srinivasan (2015A7PS0382H)
Bharat Raghunathan (2015AAPS0263H)
Nitish Ravishankar (2015A7PS0152H)

April 6, 2018

Contents

1 Task 1: Architecture of the Neural Network

The main aim of the task was to come up with an implementation or representation of the neural network using $C{+}{+}STL$ containers and some variables of our user defined **Matrix** type

1.1 Main subtasks presented by the problem

1. Implementing a generic interface: It was decided to implement a generic template to take the number of layers, the number of nodes in each layer as input via the n_layers variable and $layers\_desc$ array which is an array of size n_layers and contains the description of each layer, in this case, the number of nodes in each layer

2. Random initialization of weights and biases: The weights were initialized with the help of a uniform distribution in the range using an $std :: random\_device$ and the default $std :: default\_random\_engine$ in $C{+}{+}$.

The biases are initialized separately and are handled separately throughout the code

2 Task 2: Feedforward and Backpropagation

The main aim of this task is to implement a feedforward operation through each layer of the neural network, make a prediction of the possible target digit, calculate the **derivatives/gradient** of the **cross-entropy** error function with respect to each weight using the method of **backpropagation**.

2.1 Main subtasks of the problem

1. **Feedforward**: The value stored at each node in the neural network is a linear combination of the weights of the previous layer augmented with the bias term, and the value at the node is activated by using a **sigmoid** function (except the inputs themselves), and **sigmoid** of the final values of the last layer (a 10 x 1 matrix for each example) is interpreted as the probability of the given training instance being equal to the i^{th} digit $(i = 0 - 9)$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$

$$Z^{[i]} = W^{[i]}A^{[i-1]} + b^{[i]}$$
$$A^{[i]} = \sigma(Z^{[i]})$$

Time complexity is $O(|w| + |b|)$, where $|w|$ is the total number of weights in the network and $|b|$ is the total number of biases in the network.

2. **Backpropagation**: Once an iteration of feedforward is over, the error for the output layer is simply the difference between the actual output matrix and predicted output matrix, which is then used to compute the gradient/derivative for the output layer, then the error for the ith layer is derived based on the error in the $(i + 1)^{th}$ layer, the weights in the ith layer, as well as the derivative of the sigmoid function, which follows the property

$$\frac{d\sigma(x)}{dx} = \sigma(x) \cdot (1 - \sigma(x))$$

and hence, the error gets back propagated from the last layer to the first

layer and the derivative of each layer is then defined by

$$
\begin{aligned}
dZ^{[n]} &= A^{[n]} - Y \\
dW^{[n]} &= \frac{1}{m} dZ^{[n]} A^{[n-1]^T} \\
db^{[n]} &= \frac{1}{m} row\_sum(dZ^{[n]})
\end{aligned}
$$

$$
\begin{aligned}
dZ^{[i]} &= W^{[i+1]^T} dZ^{[i+1]} * \sigma^{'}(Z^{[i]}) \\
dW^{[i]} &= \frac{1}{m} dZ^{[i]} A^{[i-1]^T} \\
db^{[i]} &= \frac{1}{m} row\_sum(dZ^{[i]})
\end{aligned}
$$

Time complexity is similarly $O(|w| + |b|)$, where $|w|$ is the total number of weights in the network and $|b|$ is the total number of biases in the network.

3 Task 3: Adaptive Gradient and Momentum

The goal of this task was to implement an **Adaptive Learning Rate** aided with **Momentum** to perform an update of the weights, having obtained the gradient from backpropagation.

This technique is an iterative approach which tries to reduce the **misclassification error** in each iteration using the gradient information

Momentum:

$$
\begin{aligned}
v_t &= \gamma v_{t-1} + \eta \nabla J(\theta) \\
\theta &= \theta - v_t
\end{aligned}
$$

Adagrad:

$$
\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t
$$

where η is the learning rate
Combining the two, we gets

$$
\begin{aligned}
v_t &= \gamma v_{t-1} + \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t \\
\theta &= \theta - v_t
\end{aligned}
$$

4 Results

```
Loss: 54.4008 Validation Loss: 206.842 for Epoch: 974
Loss: 54.3397 Validation Loss: 206.868 for Epoch: 975
Loss: 54.2787 Validation Loss: 206.894 for Epoch: 976
Loss: 54.2179 Validation Loss: 206.921 for Epoch: 977
Loss: 54.1572 Validation Loss: 206.947 for Epoch: 978
Loss: 54.0966 Validation Loss: 206.973 for Epoch: 979
Loss: 54.0361 Validation Loss: 207 for Epoch: 980
Loss: 53.9758 Validation Loss: 207.026 for Epoch: 981
Loss: 53.9156 Validation Loss: 207.053 for Epoch: 982
Loss: 53.8555 Validation Loss: 207.079 for Epoch: 983
Loss: 53.7956 Validation Loss: 207.105 for Epoch: 984
Loss: 53.7358 Validation Loss: 207.132 for Epoch: 985
Loss: 53.6761 Validation Loss: 207.158 for Epoch: 986
Loss: 53.6165 Validation Loss: 207.184 for Epoch: 987
Loss: 53.5571 Validation Loss: 207.211 for Epoch: 988
Loss: 53.4978 Validation Loss: 207.237 for Epoch: 989
Loss: 53.4386 Validation Loss: 207.263 for Epoch: 990
Loss: 53.3796 Validation Loss: 207.29 for Epoch: 991
Loss: 53.3206 Validation Loss: 207.316 for Epoch: 992
Loss: 53.2618 Validation Loss: 207.343 for Epoch: 993
Loss: 53.2031 Validation Loss: 207.369 for Epoch: 994
Loss: 53.1446 Validation Loss: 207.395 for Epoch: 995
Loss: 53.0861 Validation Loss: 207.422 for Epoch: 996
Loss: 53.0278 Validation Loss: 207.448 for Epoch: 997
Loss: 52.9696 Validation Loss: 207.474 for Epoch: 998
Loss: 52.9115 Validation Loss: 207.501 for Epoch: 999
Accuracy is 0.970659
```

Figure 1: Execution of Program and Display of Results

Metric used:

$$\text{Accuracy} = \frac{\text{Number of Correct Classifications}}{\text{Total number of Test Examples}}$$

4.1 Understanding of Results

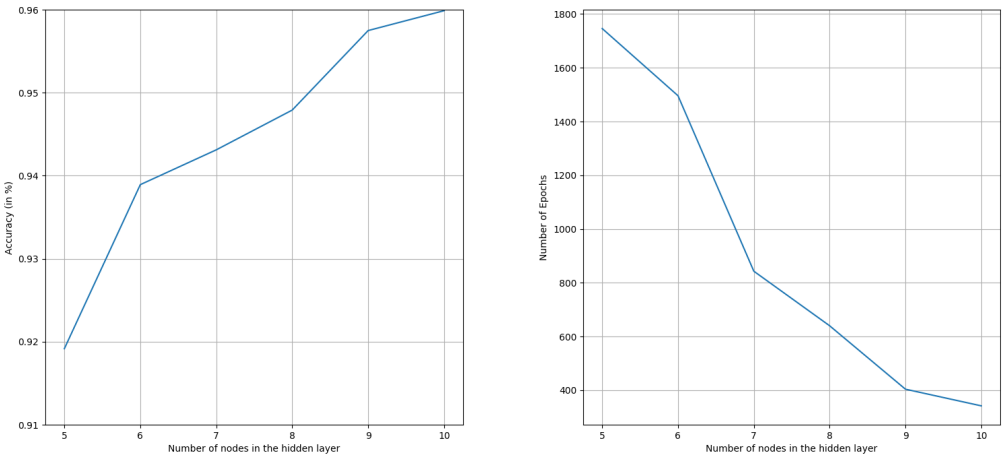
1. Accuracy of the neural network is over 90%, indicating that it has almost been able to capture most of the features necessary for predicting a handwritten digit

2. More number of hidden layers take a lesser no of epochs to converge and also gives higher Accuracy

3. Adagrad+Momentum seems to converge faster than plain SGD

Number of Nodes in Hidden Layer	Accuracy
5	91.91%
6	93.89%
7	94.31%
8	94.79%
9	95.74%
10	95.98%

Table 1: Accuracy for different number of hidden layer nodes



(a) Accuracy vs Hidden Layer Nodes

(b) Epochs vs Hidden Layer Nodes

Figure 2: Behaviour on variation of Hidden Layer Nodes

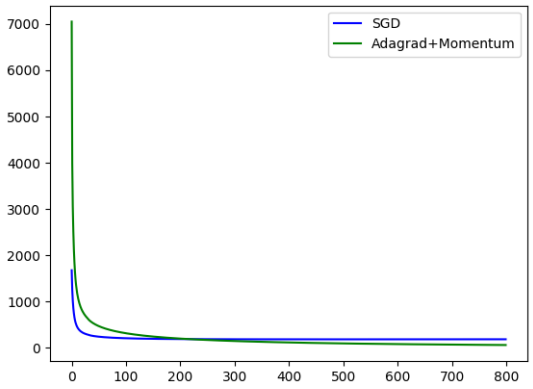


Figure 3: Performance of SGD vs Adagrad+Momentum

A Notation used for Equations

$$
\begin{aligned}
m &= \text{batch\_size} \\
Z^{[i]} &= \text{values at node without activation} \\
A^{[i]} &= \text{values at node with activation} \\
dZ^{[i]} &= \text{derivatives of } Z^{[i]} \\
dA^{[i]} &= \text{derivatives of } A^{[i]} \\
W^{[i]} &= \text{weights for transition from layer } i \text{ to layer } i+1 \\
dW^{[i]} &= \text{derivatives of } W^{[i]} \\
b^{[i]} &= \text{biases for transition from layer } i \text{ to layer } i+1 \\
db^{[i]} &= \text{derivatives of } b^{[i]} \\
Y &= \text{target outputs} \\
X &= \text{inputs}
\end{aligned}
$$