Saurabh Mylavaram
ID# 5593072
mylav008@umn.edu

Homework 3
CSCI 5525-Machine Learning

2020-11-09

1 Problem 1

1.1 Part a

$$\mathbf{n = 4, m = 4}$$

With a padding of 1, the image dimensions become 12×12. A filter of size 3×3 can be calculated 4 times in each direction on this image with a stride of 3.

$$\mathbf{k = 16}$$

Since the number of elements has to be preserved in the flatten operation, k must be $= m * n$

1.2 Part b

$$\mathbf{W_{conv} : 3 \times 3}, \quad \mathbf{W_{fc} : 10 \times 16}, \quad \mathbf{b : 10 \times 1}$$

2 Problem 2: Fully Connected Neural Network

The architecture of neural network we implement in this problem is shown in the below image:

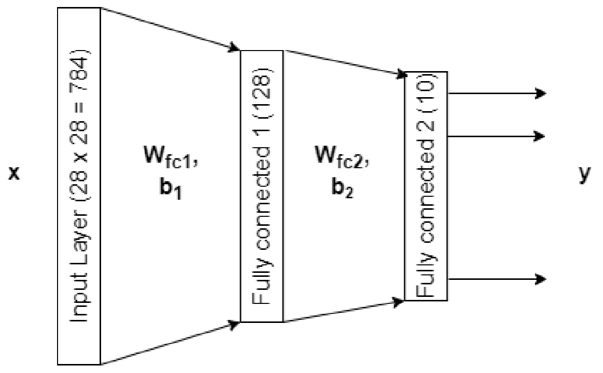


Figure 1: Network Architecture for Problem 2

Theory and equations

We can see that the final output vector is got from the following equation:

$$\mathbf{y} = \text{Softmax}\left(\mathbf{W_{fc2}}\ \text{ReLU}\left(\mathbf{W_{fc1}}\ \mathbf{x} + \mathbf{b_1}\right) + \mathbf{b_2}\right)$$

Where the activation functions are given by Softmax$(a) = \left[\frac{e^{a_i}}{\sum_j e^{a_j}}\right]$ and ReLU$(a) = \max(0, a)$.
This network is trained with Stochastic Gradient Descent (SGD) optimizer with the gradients of each layer calculated using the back-propagation algorithm.

$$\mathbf{W_j} \leftarrow \mathbf{W_j} - \eta \frac{\mathbf{dE}}{\mathbf{dW_j}}$$

$$\mathbf{w_{k,j}} \leftarrow \mathbf{w_{k,j}} - \eta \times \mathbf{z_j} \times \mathbf{\Delta_k}$$

Where $\mathbf{z_j}$ is the output on j^{th} layer and $\mathbf{\Delta_k} = (\mathbf{y_k} - \mathbf{\hat{y}_k}) \times \mathbf{h}'(\mathbf{a_k})$. Here $\mathbf{E}$ is the cross-entropy loss. For a single input data-point, it is given by $\mathbf{E} = -\sum_{\mathbf{i}}^{\mathbf{C}} \mathbf{t_i} \log(\mathbf{y_i})$, where $\mathbf{t_i}$ is the binary target variable indicating if the point belongs to class i.

Implementation details

Out input data consists of 28×28 matrices which represent the true pixel values of images ranging between 0 and 255. For each image like this, we have the target value indicating what the number drawn in image is. We have 60,000 data-points in training set and 10,000 for testing.

Flattening: Since the input dimensions to the first fully connected layer is a 784 length vector, we flatten each input data image by joining together the rows of the matrix end-to-end to form a 784 length vector.

Scaling: We also scale out data from the range $[0, 255]$ to the range $[0, 1]$. This can be achieved by dividing all the vector entries by 255.

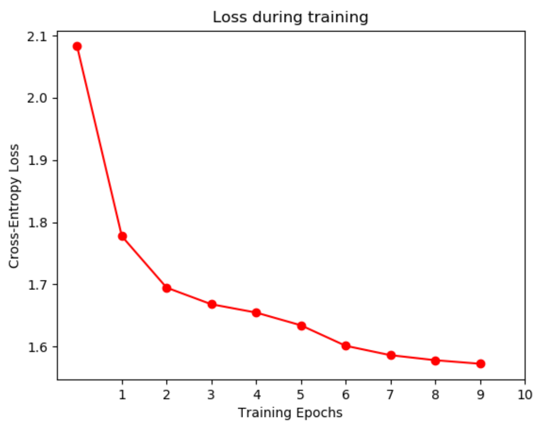
Training: Next, we build the neural network using functional style layers and compile the model with *SparseCategoricalCrossentropy* loss function. This loss function is used instead of *CategoricalCrossentropy* when the target values in our input data are given as integers instead of one-hot vectors.

We choose *accuracy* as the metric to report and fit the model on training set for *10 epochs* and *batch size of 32*.

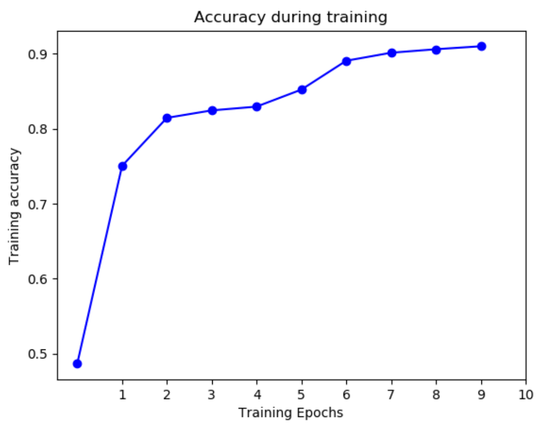
Results

We can get the training results from the *History* object returned by the *fit* function.
Training loss for 10 epochs: $[\mathbf{2.083}, \mathbf{1.777}, \mathbf{1.695}, \mathbf{1.668}, \mathbf{1.655}, \mathbf{1.634}, \mathbf{1.601}, \mathbf{1.586}, \mathbf{1.578}, \mathbf{1.572}]$
Training accuracy for 10 epochs: $[\mathbf{0.487}, \mathbf{0.751}, \mathbf{0.815}, \mathbf{0.825}, \mathbf{0.83}, \mathbf{0.852}, \mathbf{0.891}, \mathbf{0.901}, \mathbf{0.906}, \mathbf{0.91}]$

The graphs below show the change in loss and accuracy as training progresses:



(a) Training Loss vs Epochs



(b) Training Accuracy vs Epochs

This trained model is then evaluated on the testing data and the testing accuracy is calculated to be **91.56%**.

3 Problem 3: Convolutional Neural Network

The architecture of the CNN used in this problem is shown in the below image:

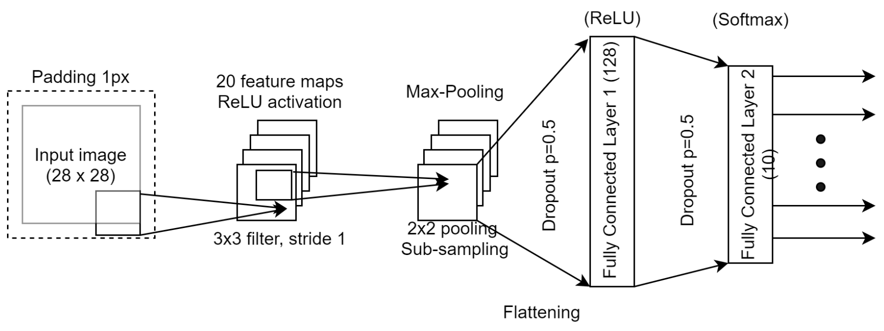


Figure 3: CNN Architecture for Problem 3

The theory and equations for training the neural network using back-propagation is same as that shown in the previous problem.

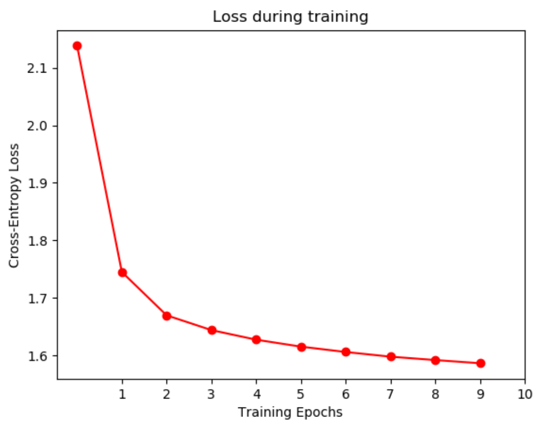
Implementation details

For the convolutional layer we choose the number of filters as 20, filter size as 3 and the stride as 1 in both directions. We set the padding parameter to *'same'* which preserves the image dimensions after convolution. The output of convolutional layer and fully connect layer are connected through a dropout mechanism to their respective next layers. This is implemented as a separate layer in sklearn, to which we pass a random seed of 42 to maintain reproducible results.

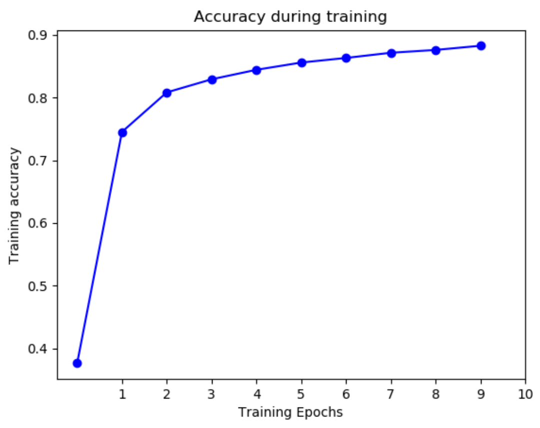
Results

First we run the CNN with SGD optimizer and report loss and accuracy as the training progresses over epochs:
Training loss for 10 epochs: $[\mathbf{2.098}, \mathbf{1.771}, \mathbf{1.687}, \mathbf{1.654}, \mathbf{1.636}, \mathbf{1.624}, \mathbf{1.613}, \mathbf{1.606}, \mathbf{1.597}, \mathbf{1.593}]$
Training accuracy for 10 epochs: $[\mathbf{0.414}, \mathbf{0.712}, \mathbf{0.791}, \mathbf{0.818}, \mathbf{0.836}, \mathbf{0.846}, \mathbf{0.856}, \mathbf{0.862}, \mathbf{0.872}, \mathbf{0.874}]$

Plots of training loss and accuracy vs training epochs are shown below:



(a) Training Loss vs Epochs



(b) Training Accuracy vs Epochs

The CNN model trained with SGD is evaluated on testing set and testing accuracy is **93.27%**

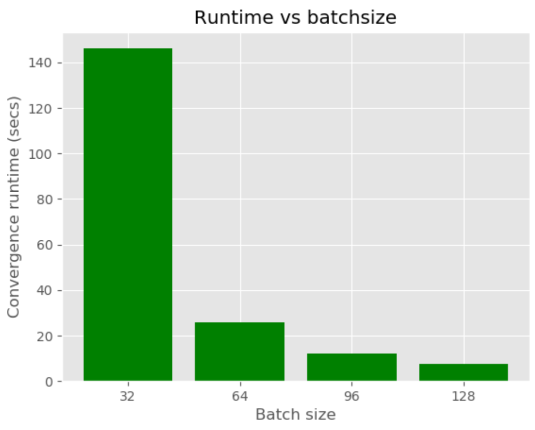
Next we modify the CNN to include an early stopping criteria which indicates convergence. We set the training to stop when the difference in training loss between epochs is less than $\epsilon = 10^{-4}$.

$$\text{Convergence criteria: } \Delta\text{Training loss} < \epsilon$$

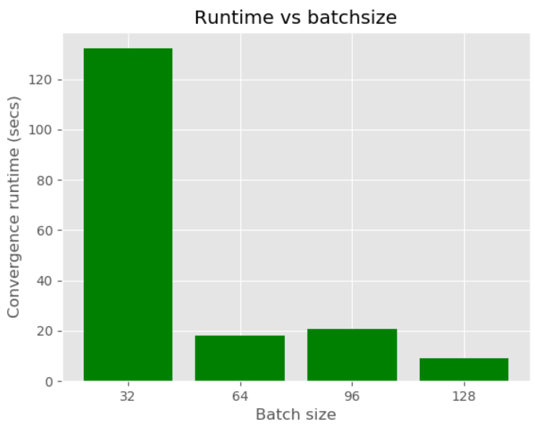
With this condition we examine the effect of batch size on time algorithm takes to converge. We do this for 3 different optimizers *SGD, Adagrad and Adam*. Results are shown in the table below:

	Batch size $\rightarrow$			
Optimizer $\downarrow$	**32**	**64**	**96**	**128**
SGD	146.09 secs	26.00 secs	12.29 secs	7.67 secs
Adagrad	132.21 secs	18.12 secs	20.92 secs	9.12 secs
Adam	19.81 secs	13.43 secs	11.22 secs	5.61 secs

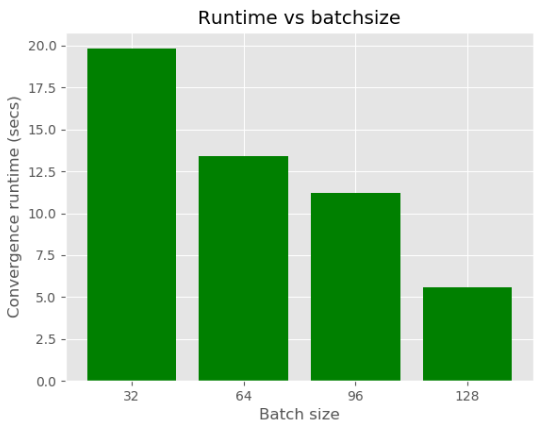
These results are shown as bar charts below:
(Please see next page.)



(a) SGD



(b) Adagrad



(c) Adam

Figure 5: Convergence time vs Batch-size for difference optimizers