**Problem 1.**

**Q(1)**

n: 4

m: 4

k: 16

**Q(2)**

$W_{conv}$ = (3,3,1,1)

$W_{fc} = (10 \times 16)$

$b = (10 \times 1)$

**Problem 2**

**Algorithm:**

**ReLU activation function:**

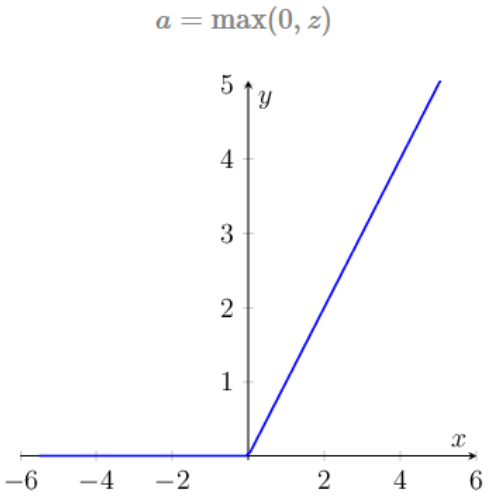


Figure 1: The Graph of ReLU Function

For ReLU activation function, we would set the negative input as 0 and positive input as itself.

**Softmax activation function:**

$$h_\theta(x) = \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \vdots \\ P(y=K|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \\ \vdots \\ \exp(\theta^{(K)\top} x) \end{bmatrix} \tag{1}$$

where

$\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)} \in \Re^n$ are the parameters of the model.

**Cross Entropy Loss Function:**

For the sample point (x, y), y is the true label. We assume that there are K label values, and the probability that the i-th sample is predicted to be the k-th label value is $p_{i,k}$. For N samples, the cross entropy loss function is:

$L_{log}(Y,P) = -logPr(Y|P) = -\frac{1}{n}\sum_{i=0}^{N-1}\sum_{k=0}^{K-1} y_{i,k} log p_{i,k}$

**SGD:**

If we assume the objective function is:

$J(x) = \frac{1}{n}\sum_{i=1}^{n} J(x_i)$

$J(x_i)$ is the objective function of i objects in x, then the gradient for x would be:

$\nabla J(x) = \frac{1}{i}\nabla\sum_{j=1}^{i} J(x_j)$

If we use the gradient descent method (batch gradient descent method), then we need to iterate N samples every time. If we use the SGD(mini -batch gradient descent method), we only need to iterate small amount of samples every time. Thus, then time complexity of the programming would decrease.

**Backpropagation:**

The squared error on a single example is $E = \frac{1}{2}\sum_k (y_k - \hat{y_k}^2)$

Gradient to propagate errors back through second layer:

$\frac{\partial E}{\partial w_{k,j}} = -(y_k - \hat{y_k})\frac{\partial \hat{y_k}}{\partial w_{k,j}}$

$= -(y_k - \hat{y_k})\frac{\partial h a_k}{\partial w_{k,j}}$

$= -(y_k - \hat{y_k}) h'(a_k)\frac{\partial}{\partial w_{k,j}}(\sum_j w_{k,j} z_j)$

$= -(y_k - \hat{y_k}) h'(a_k) z_j$

$= -z_j \times \triangle_k$

So, backpropagate error to putput layer:

$w_{k,j} < -w_{k,j} + \alpha \times z_j \times \triangle_k$, where $\triangle_k = (y_k - \hat{y_k}) h'(a_k)$ and $\alpha$ is learning rate.

For hidden layer:

$\frac{\partial E}{\partial w_{j,i}} = -(y_k - \hat{y_k})\frac{\partial \hat{y_k}}{\partial w_{j,i}}$

$= -\sum_k \triangle_k \frac{\partial}{\partial w_{j,i}} (\sum_j w_{k,j} z_j)$

$= -\sum_k \triangle_k w_{k,j} \frac{\partial h(a_j)}{\partial w_{j,i}}$

$= -\sum_k \triangle_k w_{k,j} h'(a_j) \frac{\partial}{\partial w_{j,i}} (\sum_i w_{j,i} x_i)$

$= -\sum_k \triangle_k w_{k,j} h'(a_j) x_i$

$= -x_i \times \triangle_j$

So, backpropagate error from the output layer to hidden layer:

$w_{j,i} < -w_{j,i} + \alpha \times x_i \times \triangle_j$, where $\triangle_j = h'(a_j) \sum_k w_{k,j} \triangle k$ and $\alpha$ is learning rate.

**Result:**

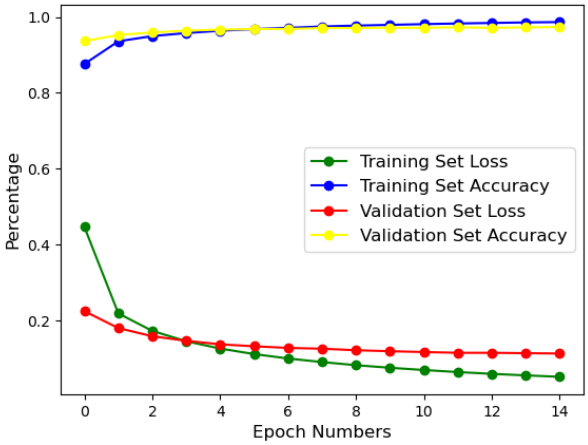


Figure 2: SGD optimizer, multi-layer fully connected neural network, batch size = 32

**Testing accuracy:** 0.9688

**Problem 3**

**Algorithm:**

Besides the algorithms we used in problem 2, some new algorithms are used in convolutional neural network.

**Filter:**

Using filter to capture the similarity between the input image and the filter and form the convolution layer.

**Pooling:**

Pooling is to shrink the input image, reduce pixel information, and only retain important information of the input image.

If the pooling size is $n * n$, we take the block which has the maximum value in $n * n$ size of the pixels for max-pooling.

**Drop out:**

When we propagate forward, we could let the activation value of a certain neuron stop working with a certain probability $p$, which can make the model more generalized because it will not rely too much on some local features.

Feed-forward with dropout neural network for node i:

$r_j^{(l)} \sim Bernoulli(p)$

$\tilde{z}^{(l)} = r^{(l)} \odot z^{(l)}$

$a_i^{(l+1)} = w_i^{(l+1)} \tilde{z}^{(l)} + b_i^{(l+1)}$

$z_i^{(l+1)} = h(a_i^{(l+1)})$

**Adagrad:**

We would add an adaptive learning rate before the gradient:

$\eta_t^i = \frac{\eta_0}{\sqrt{\sum_{s=0}^{t}(g_s^i)^2}}$

**Adam:**

RMSprop + first and second order momentum + bias correction.

**Result:**

**Part 1:**

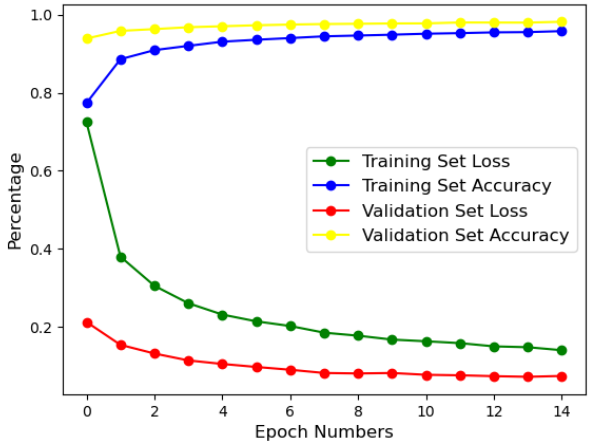


Figure 3: SGD optimizer, convolution neural network, batch size = 32

**Testing accuracy:** 0.9777

**Part 2:**

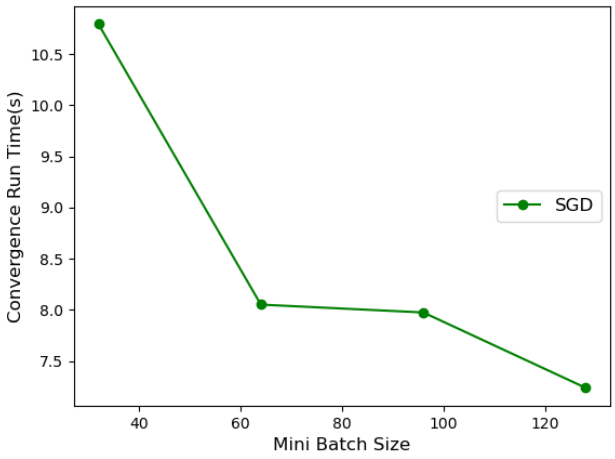


Figure 4: SGD optimizer, convolution neural network, convergence run time vs mini batch size

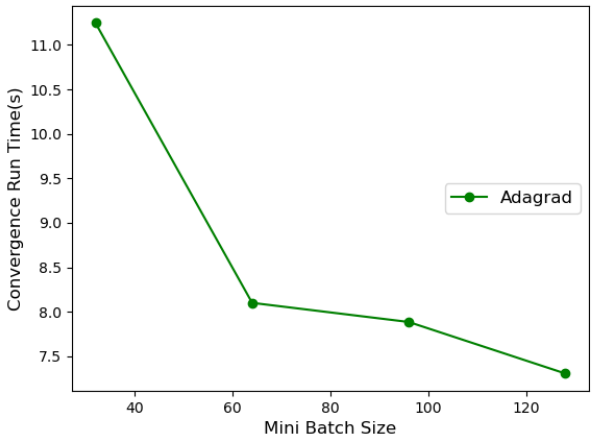


Figure 5: Adagrad optimizer, convolution neural network, convergence run time vs mini batch size

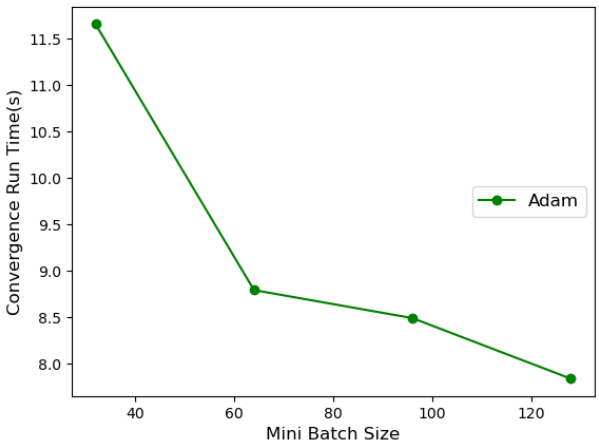


Figure 6: Adam optimizer, convolution neural network, convergence run time vs mini batch size

Along with the increasing mini batch size, the convergence run time is decreasing.

**Reference:**

1.Pattern Recognition and Machine Learning,CHRISTOPHER M.BISHOP

2.Lecture in Canvas,Nicholas Johnson