16-385 Computer Vision, Fall 2019

Homework 3
Neural Networks for Recognition

Due Date: Mon Oct 21, 2019 23:59

1 Theory

1.1 Theory

$$
\begin{aligned}
softmax(x+c) &= \frac{exp(x_i+c)}{\sum_{k=1}^{K} e^{x_k+c}}, \qquad for \quad i=1,...,K \\
&= \frac{e_i^x e^c}{\sum_{k=1}^{K} e^{x_k+c}} \\
&= \frac{e_i^x e^c}{e^c \sum_{k=1}^{K} e^{x_k}} \\
&= \frac{e_i^x}{\sum_{k=1}^{K} e^{x_k}}, \qquad for \quad i=1,...,K \\
&= softmax(x)
\end{aligned}
$$

When we use $c = -maxx_i$, we can compute softmax probabilites in numerical stability. So it can subtract its minimum element from all elements of x. Therefore, it is a good idea.

1.2 Theory

- The range of each element in softmax is 0 to 1. Each element means the probability of each class, then its range is between 0 and 1. Therefore the sum over all elements means the sum of probability. It will be 1.
- The probability of each class for vector x.
- Step 1 means score of each class for x. Step 2 means the sum of scores for all classes. So we divides each class score by sum of scores at step 3. So step 3 means the probability for each class.

1.3 Theory

It we suppose case of two hidden layers. The first layer's parameter is w_1 and b_1 and the second layer's parameter is w_2 and b_2. The first layer's output is $y_1 = w_1 x_{input} + b_1$. And our layer is without non-linear activation, so this result goes directly to second layer. Then the second layer's output is similar.

$$
\begin{aligned}
y_2 &= w_2 y_1 + b_1 \\
&= w_2(w_1 x_{input} + b_1) + b_2 \\
&= w_1 w_2 x_{input} + w_2 b_1 + b_2 \\
&= w_1 w_2 x_{input} + (w_2 b_1 + b_2)
\end{aligned}
$$

At the result of second layer, we can say it is also linear function. So we can compress first hidden layer and second hidden layer to just one hidden layer. Therefore, even if we stack more hidden layer, we can compress all hidden layers to just one hidden layer when there are no activation function. So we can say it is same as linear regression.

1.4 Theory

Given the sigmoid activation functino $\sigma(x) = \frac{1}{1+e^{-x}}$, derivative of the sigmoid is like below.

$$\begin{aligned}\frac{d\sigma(x)}{dx} &= \frac{e^{-x}}{(1+e^{-x})^2} \\ &= \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} \\ &= \frac{1}{1+e^{-x}} \cdot (1 - \frac{1}{1+e^{-x}}) \\ &= \sigma(x)(1-\sigma(x))\end{aligned}$$

So derivative of the simgoid can be written as a function of $\sigma(x)$.

1.5 Theory

Given $z = x^T W + b$, and the result of activation is $y = activation(z)$. Then we pass this value to loss J. In this case, we define J as below. Each input data size $x = d \times 1$, y size is $y = k \times 1$. There are n samples for training data. So input data $X = n \times d$ and $Y = n \times k$.

$$X \in R^{n \times d} \quad W \in R^{d \times k} \quad b \in R^{k \times 1} \quad Y \in R^{n \times k}$$

$$\begin{aligned}\frac{\partial J}{\partial y} &= \delta \in R^{n \times k} \\ \frac{\partial J}{\partial W} &= \frac{\partial J}{\partial y} \cdot \frac{y}{\partial z} \cdot \frac{\partial z}{\partial W} \\ &= \delta \cdot activation'(z) \cdot x \in R^{d \times k} \\ \frac{\partial J}{\partial b} &= \frac{\partial J}{y} \cdot \frac{y}{\partial z} \cdot \frac{\partial z}{\partial b} \\ &= \delta activation'(z) \cdot 1 \in R^{k \times 1} \\ \frac{\partial J}{\partial x} &= \frac{\partial J}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial x} \\ &= \delta \cdot activation'(z) \cdot W \in R^{n \times d}\end{aligned}$$

The derivatives are like above.

1.6 Theory

1. The derivative of sigmoid is $\sigma(1-\sigma)$. It is between 0 and 1/4. When we stack multilayers, the derivative of first layer weight becomes small very fast. For example, we have 2 hidden layers, the derivative is $\frac{\partial error}{\partial w_1} = \frac{\partial error}{\partial output} \cdot \frac{\partial output}{\partial hidden2} \cdot \frac{\partial hidden2}{\partial hidden1} \cdot \frac{\partial hidden1}{\partial w_1}$. In this case, $\frac{\partial output}{\partial hidden2} \cdot \frac{\partial hidden2}{\partial hidden1}$, we are multiplying values between 0 and 1. So it will become small very fast. Even if weight initialization technique is not employed, the vanishing gradient problem will most likely still occur. If neural network is more deeper, we backpropagate further back. Therefore, we have many more small numbers partaking in a product, creating an even tinier gradient. Thus, with deep neural nets, the vanishing gradient problem becomes a major concern.

2. The range of sigmoid is between 0 and 1. For tanh, we can revise this function as $tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}} = \frac{2}{1+e^{-2x}} - 1$. We can explain this function using sigmoid. It is $tanh(x) = 2sigmoid(2x) - 1$. Therefore, the range of tanh is between -1 and 1. Because of range for tanh, there is no worries of activations blowing up. So the gradient is stronger for tanh than sigmoid. For this reason, we prefer tanh to sigmoid.

3. 3. As you see above, we can revise thanh function as $tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}} = \frac{2}{1+e^{-2x}} - 1 = 2\sigma(2x) - 1$.

1.7 Theory

1. We consider a 3×3 convolutional kernel W, $\begin{matrix} w_0 & w_1 & w_2 \\ w_3 & w_4 & w_5 \\ w_6 & w_7 & w_8 \end{matrix}$, and 3×3 single-channel image I, $\begin{matrix} x_0 & x_1 & x_2 \\ x_3 & x_4 & x_5 \\ x_6 & x_7 & x_8 \end{matrix}$.

If we apply 1 dimensional zero padding to image, than the image will be 5×5 matrix, $\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & x_0 & x_1 & x_2 & 0 \\ 0 & x_3 & x_4 & x_5 & 0 \\ 0 & x_6 & x_7 & x_8 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$.

Unlike no padding, there are 9 valid location in the image to apply the filter W. The calculation is like below.

i. Calculate multiply matrix and image patch starting at left top.

$$W \cdot \begin{matrix} 0 & 0 & 0 \\ 0 & x_0 & x_1 \\ 0 & x_3 & x_4 \end{matrix} = \begin{bmatrix} w_0 & w_1 & w_2 & w_3 & w_4 & w_5 & w_6 & w_7 & w_8 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ x_0 \\ x_1 \\ 0 \\ x_3 \\ x_4 \end{bmatrix}$$

ii. Reply above calculation by sliding filter left to right.

iii. Finallly, calculate multiply matrix and image patch at right below.

$$W \cdot \begin{matrix} x_4 & x_5 & 0 \\ x_7 & x_8 & 0 \\ 0 & 0 & 0 \end{matrix} = \begin{bmatrix} w_0 & w_1 & w_2 & w_3 & w_4 & w_5 & w_6 & w_7 & w_8 \end{bmatrix} \cdot \begin{bmatrix} x_4 \\ x_5 \\ x \\ 0 \\ x_7 \\ x_8 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Therefore, the overall equation is like below.

$$\begin{bmatrix} w_0 & w_1 & w_2 & w_3 & w_4 & w_5 & w_6 & w_7 & w_8 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 & 0 & x_0 & x_1 & 0 & x_3 & x_4 \\ 0 & 0 & 0 & x_0 & x_1 & x_2 & x_3 & x_4 & x_5 \\ 0 & 0 & 0 & x_1 & x_2 & 0 & x_4 & x_5 & 0 \\ 0 & x_0 & x_1 & 0 & x_3 & x_4 & 0 & x_6 & x_7 \\ x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \\ x_1 & x_2 & 0 & x_4 & x_5 & 0 & x_7 & x_8 & 0 \\ 0 & x_3 & x_4 & 0 & x_6 & x_7 & 0 & 0 & 0 \\ x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & 0 & 0 & 0 \\ x_4 & x_5 & 0 & x_7 & x_8 & 0 & 0 & 0 & 0 \end{bmatrix}$$

2. Input dimension is 225×225 and apply filter size of 5×5 with stride 4 and no paddings, the result will be 56×56. Because result size follows this function, $\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$. We apply this filter number of 64, it also means we apply weights dimension of $5 \times 5 \times 64$, so the final result size is $56 \times 56 \times 64$

2 Implement a Fully Connected Network

2.1 Network Initialization

2.1.1 Theory

If all the weights are initialized with 0, the derivative with respect to loss function is same for every W. Thus all weights have same value in subsequent iterations. This makes hidden units symmetric and continues for all the n iterations. However, the biases have no effect what so ever when initialized with 0.

2.1.2 Code

2.1.3 Theory

The basic intuition is just the bias of the system is broken and weight values can move along and away and apart to different values. If we initialize weights randomly into small value, usually between 0 and 1, there are a small gap between weights. This gap expands out as we want to go along and forces the weights to be a bit larger at every iteration, and this helps the network to converge faster. So the learning process speeds up.
And when we initialize weight depending on layer size, it can satisfy our objectives of maintaining activation variances and backpropagated gradient variance as one moves up or down the network.

2.2 Forward Propagation

2.2.1 Code

2.2.2 Code

2.2.3 Code

2.3 Backwards Propagation

2.3.1 Code

2.4 Training Loop

2.4.1 Code

2.5 Numerical Gradient Checker

2.5.1 Code

3 Training Models

3.1

3.1.1 Code

3.1.2 Writeup

At first, when we using default learning rate as $2.5e^{-3}$, then loss finds local optimal smoothly. That is about 30 in average. However, the 10 times one can not find local optima. It is too big for finding local optima. So it is bounded around local optima but not find it directly. Also, the one tenth case also does not find local optima too. Because it moves too slow to find local optima. So it finishes its training before they arrive at local optima. We can check this truth at figure 6, it has final loss value as around 100. It is higher than default learning rate case, about 30. Therefore, we confirm that learning rate affects to whether loss will be reach local optima.
In this case, the test accuracy is 78.17.

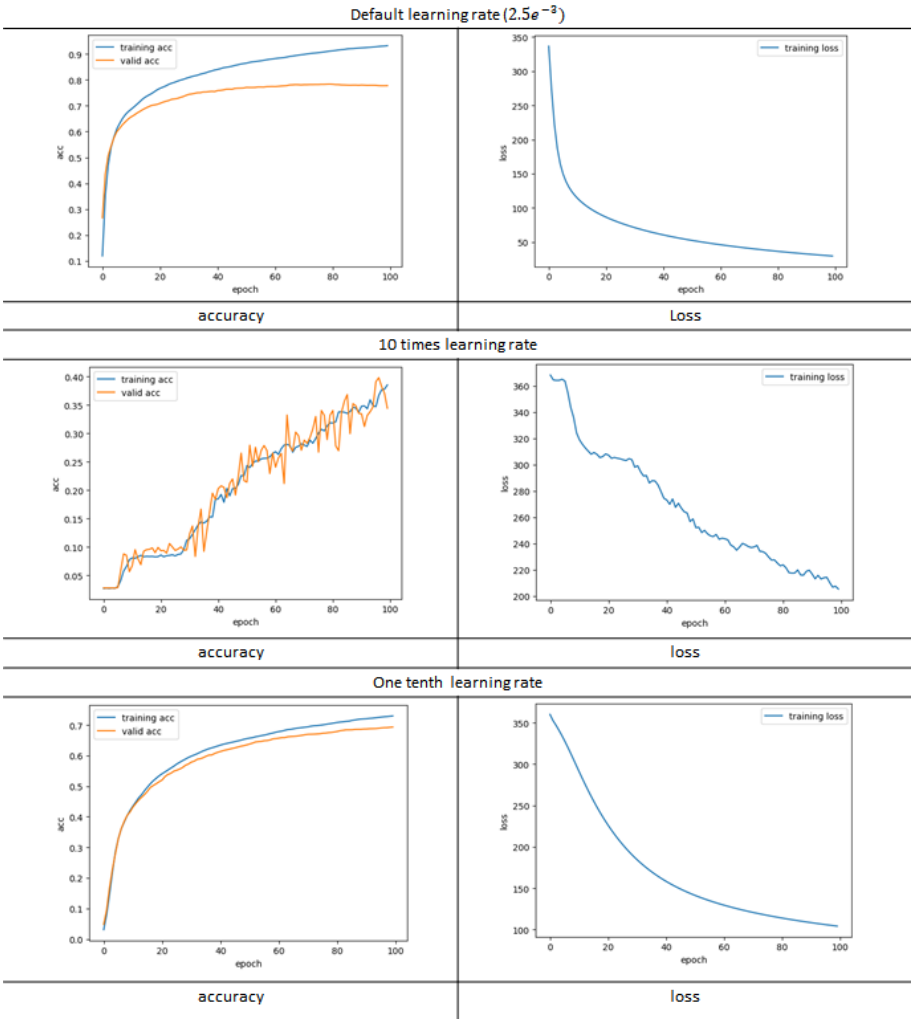

Figure 1: Accuracy and loss of each learning rate

3.1.3 Writeup

The network weights immediately after initialization is upper figure and the first layer weights taht network learned is below figure. The first one is randomly initialized. So the visualized value looks like white noise. The next one is learned weights. So it contains information of training data. Each weight extracts features of training data.

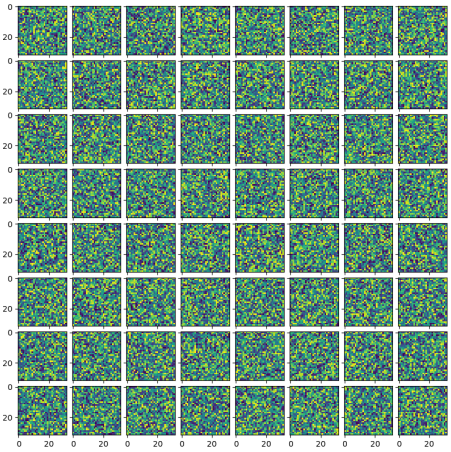

Figure 2: initialized weight

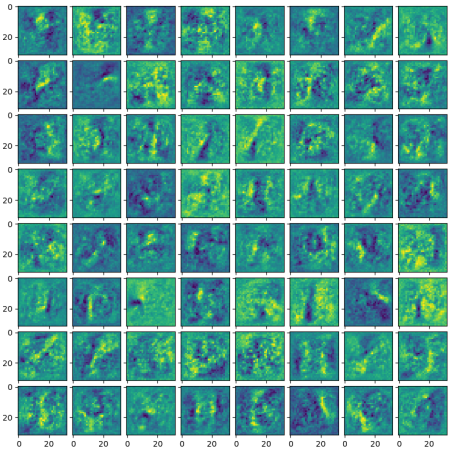

Figure 3: weight

3.1.4 Theory

The confusion matrix of network result is below. Most results are matched correctly. We can confirm it at diagonal values brightly. However, some of other values are little bit bright. For example, the case of '0' and

'O', the case of '2' and 'Z' or the case of '5' and 'S'. This is because they have similar shapes. So the network is confusing their difference.

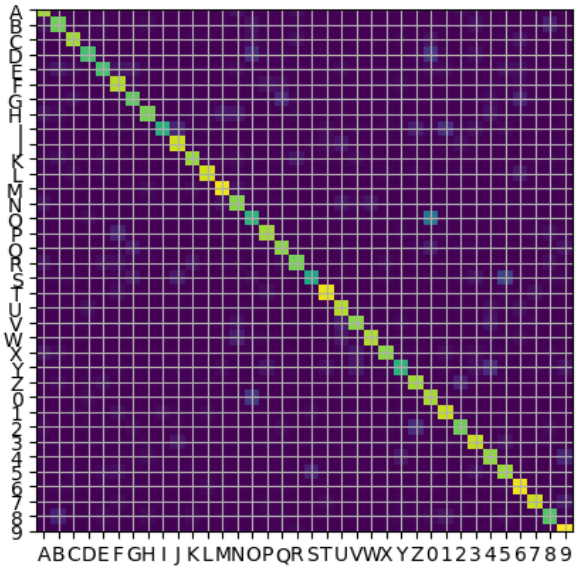

Figure 4: Confusion matrix of network result

4 Extract Text from Images

5 Image Compression with Autoencoders

5.1 Building the Autoencoder

5.1.1 Code

5.1.2 Code

5.2 Training the Autoencoder

5.2.1 Writeup/Code

The loss function figure is below. We can observe that the loss is decayed radically in the early stage. It is decayed rapidly. After this, the loss is decayed slower than early stage. During almost 90 epochs, it only reduces about 250 losses. It means in early stage, loss can be find around local optima location but can not find the exact local optima. It is because the learning rate is too big so the step is big and it is bounded around local optima.

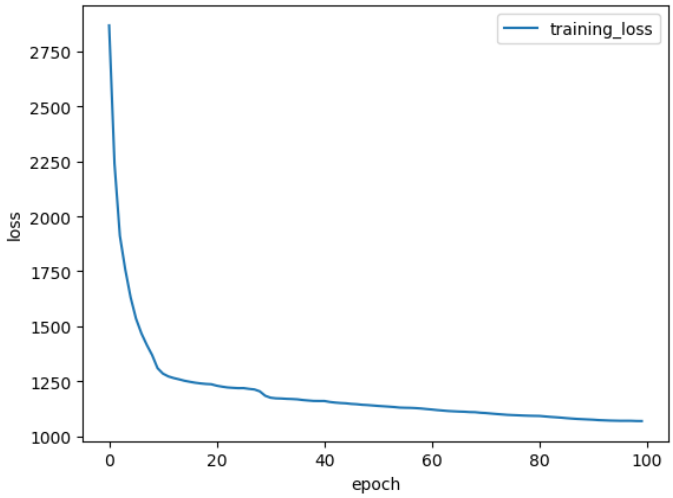

Figure 5: Training loss of default setting

5.3 Evaluating the Autoencoder

5.3.1 Writeup/Code

The validation results are below. The original ones is in first and third rows, and reconstructed ones is in second and fourth rows. The reconstructed results are too blurred and have unclear boundaries. And the background color is also stained. But the results usually follow the shape of original ones. Therefore, the result is unclear but shape is similar.

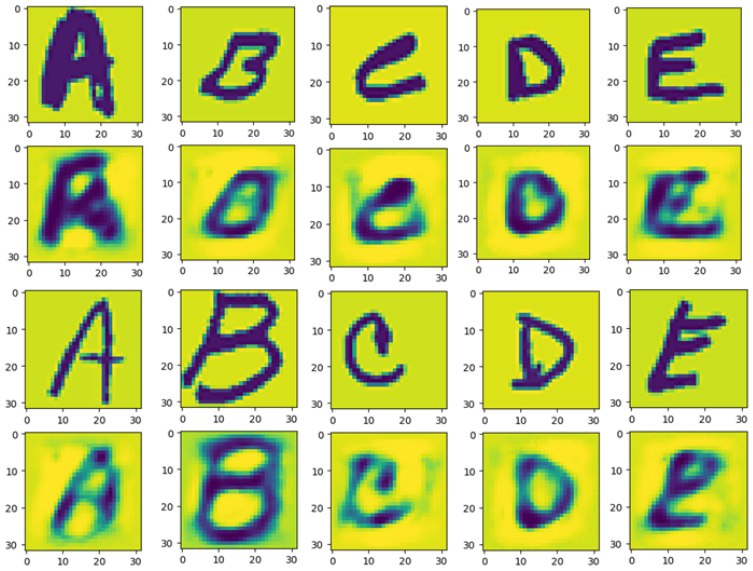

Figure 6: Validation results

5.3.2 Writeup/Code

The average of PSNR from the autoencoder across all validation images is 15.69.

```
psnr avg : 15.690676
```

Figure 7: Average PSNR result