Computer Vision HW5

Maxim Gluhovskoi

April 2024

Problem 2: Theory Questions

Q2.1

First we will prove that softmax is invariant to translation. We want to show that:
$softmax(x) = softmax(x + c)$ for every c within the reals. We know that for each index i in a vertex x softmax is defined as follows: $softamx(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$.
Let x be some arbitrarly large vector then if we add an arbitrary c to each x_i within x we get $softmax(x_i + c) = \frac{e^{x_i+c}}{\sum_j e^{x_j+c}} = \frac{e^{x_i}e^c}{\sum_j e^{x_j}e^c} = \frac{e^{x_i}e^c}{e^c\sum_j e^{x_j}} = \frac{e^{x_i}}{\sum_j e^{x_j}}$ as desired. Therefore we have proven that softmax is invariant to translation.
Now lets show why using $c = -max(x_i)$ is a good idea. If our $c = 0$ then our max x_i value could be quite large resulting in stability, overflow, and more difficult computation. On the other hand if we somewhat normalize our x_i values by setting the max to 0 we will limit our exponentials to be less than 1 fixing our instability issues and potential overflow. This also keeps the results the same since we previously proved that softmax is invariant to translation.

Q2.2

a) Softmax has a range of 0 to 1 and the sum over all elements is 1 since we are doing the following $\sum_i(\frac{e^{x_i}}{\sum_j(e^{x_j})}) = \frac{\sum_i e^{x_i}}{\sum_j e^{x_j}} = 1$.
b) Probability distribution.
c) The first step: $s_i = e^{x_i}$ takes the individual value/probability. The exponent eleminates any negative values and results in larger differences in resulting values resulting in better differentiation between values and better training. The second step: $S = \sum(s_i)$ takes the total probability value and will act as a normalizer for each s_i to make the output values range from 0 to 1. The third steps: $\frac{s_i}{S}$ applies the normalizer resulting in a clean looking probability value of s_i.

Q2.3

First I will show the individual scalar derivatives of W, x, and b. We are given the formula:
$y_j = \sum_{i=1}^{d} x_i W_{ij} + b_j$ so the derivative: $\frac{dj}{dw_{ij}} = \frac{dj}{dy}\frac{dy}{dw_{ij}} = \frac{dj}{dy} * x_i$ and $\frac{dj}{dx_i} = \frac{dj}{dy}\frac{dy}{dx_i} = \frac{dj}{dy} * \sum_{i=1}^{k} w_{ij}$ and $\frac{dj}{b_j} = \frac{dj}{dy}\frac{dy}{dw_j} = \frac{dj}{dy} * 1$. Re-forming these into matrix form we get $\frac{\partial J}{\partial W} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial W} = x(\frac{\partial J}{\partial y})^T$ and $\frac{\partial J}{\partial x} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial x} = W\frac{\partial J}{\partial y}$ and $\frac{\partial J}{\partial b} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial b} = \frac{\partial J}{\partial y} * 1$.

Q2.4

part 1) $[w_0w_1w_2w_3w_4w_5w_6w_7w_8]$
$$\begin{bmatrix} 0 & 0 & 0 & 0 & x_0 & x_1 & 0 & x_3 & x_4 \\ 0 & 0 & 0 & x_0 & x_1 & x_2 & x_3 & x_4 & x_5 \\ 0 & 0 & 0 & x_1 & x_2 & 0 & x_4 & x_5 & 0 \\ 0 & x_0 & x_1 & 0 & x_3 & x_4 & 0 & x_6 & x_7 \\ x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \\ x_1 & x_2 & 0 & x_4 & x_5 & 0 & x_7 & x_8 & 0 \\ 0 & x_3 & x_4 & 0 & x_6 & x_7 & 0 & 0 & 0 \\ x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & 0 & 0 & 0 \\ x_4 & x_5 & 0 & x_7 & x_8 & 0 & 0 & 0 & 0 \end{bmatrix}$$
part 2) The size of the first matrix representing the kernels is 64by25 and the second is 25by3136 so our resulting matrix is 64by3136 which is then reshaped to 64by56by56.

Q3.1.1

Initializing a network to all zeros would result in an output of all zeros making it extremely hard to impossible to do further training if most layers are relu or other activation functions where f(0)=0. This may also result in symmetry between neurons since all of the neurons will have the same gradient resulting in lots of neurons doing the same thing making the neural network inefficient.

Q3.1.3

We initialize with random numbers so that our gradients will be random and result in different patterns being learned (breaking symmetry and reducing redundant neurons). We scale the initialization depending on layer size since we want the variance between layers to be the same in regards to the layer size this will help prevent exploding and vanishing gradients through the multiplicative effect that gradients undergo in deep networks.

Q4.2

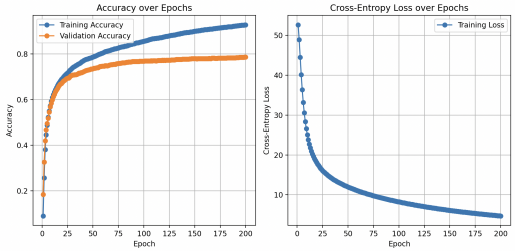


Figure 1: best learning rate

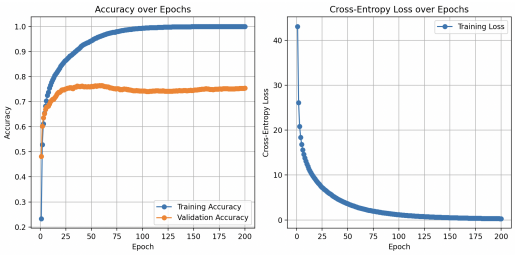


Figure 2: 10x learning rate

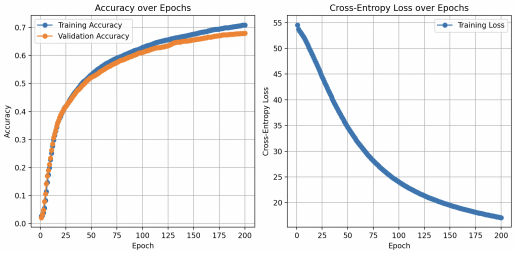


Figure 3: 1/10th learning rate

Higher learning rates result in faster optimization of training accuracy, but validation accuracy suffers greatly since we are over training. Meanwhile, lower learning rates result in slower optimization of training accuracy, but we are not overtraining as much so our validation accuracy stays much closer to our training accuracy. My best learning rate was 1e-3 with 200 epochs and batch size 15. My best validation accuracy was 78.67%.

Q4.3

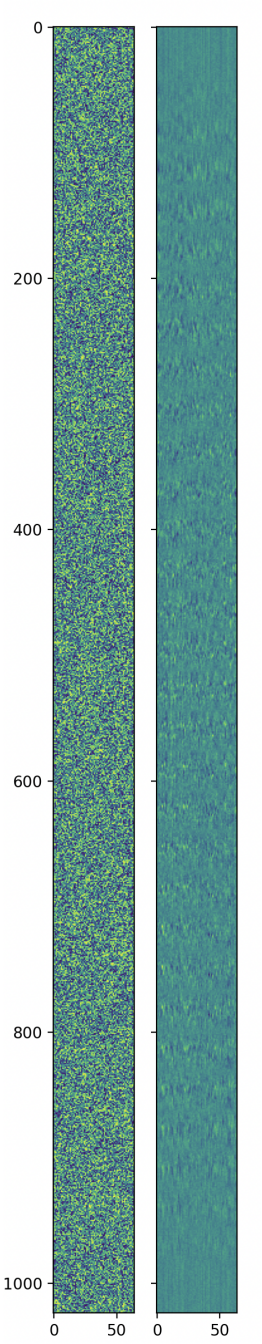


Figure 4: 1st layer weights init (left) trained (right)

The learned weights develop streaking lines downwards. These patches repeat every 36 or so rows corresponding to the rows of the input images. It basically changes from streaks to just blur and repeats.

Q4.4

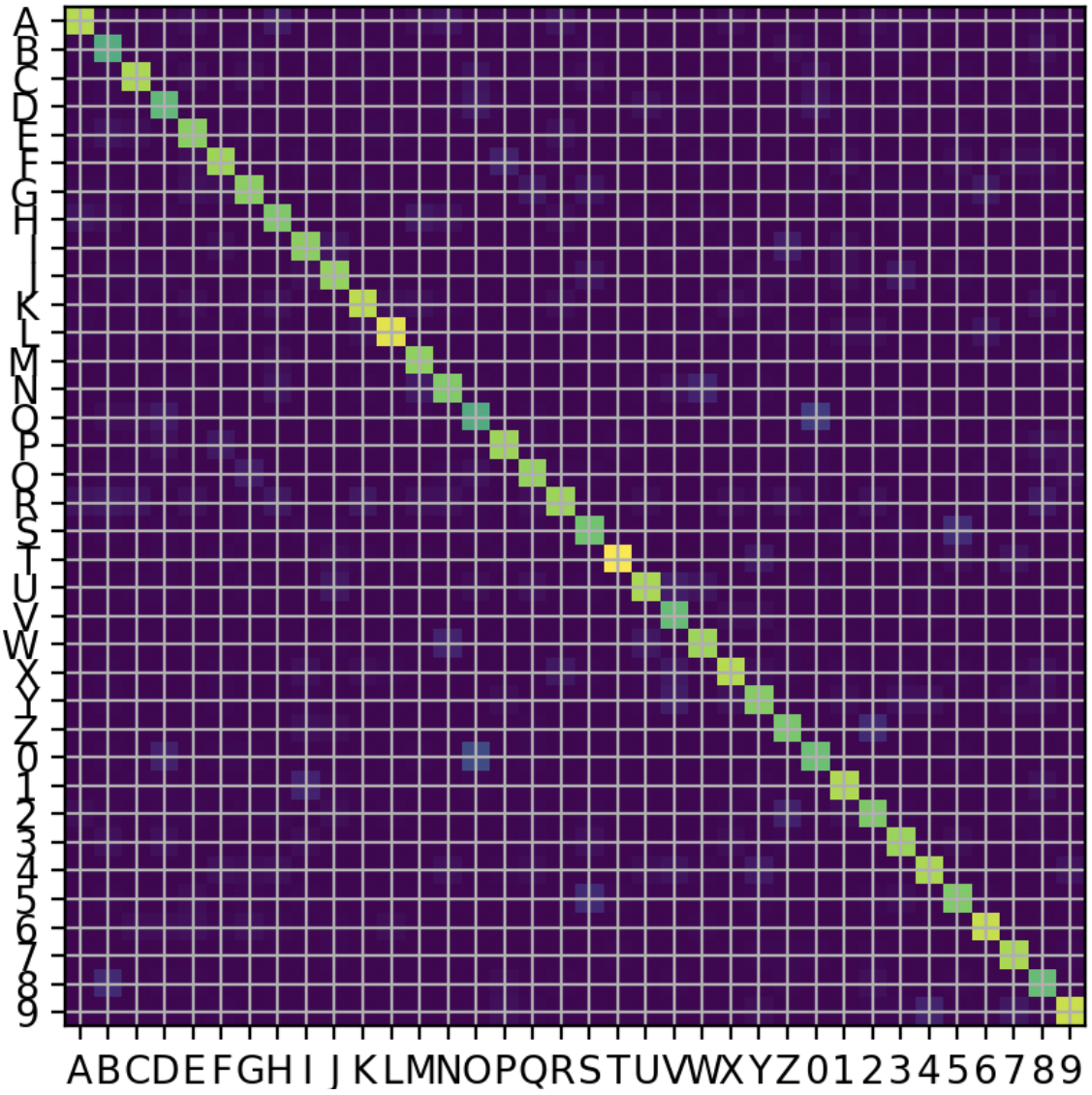


Figure 5: confusion matrix

Some of the pairs that the algoirthm is getting cofused on for example include: 0 and O, 5 and S, and B and 8. These all look pretty similiar to each other so it makes sense that the network occasionally gets these wrong/mismatched.

Q5.1

The method outlined in the writeup assumes that each letter/number is 32 by 32 pixels in size. If it is not in this size then it might provide wrong letters or skip over letters. It is also assuming that we are not using anything except for letters and numbers so any punctuation will result in miss labeled data. Here are some example images that may fail:

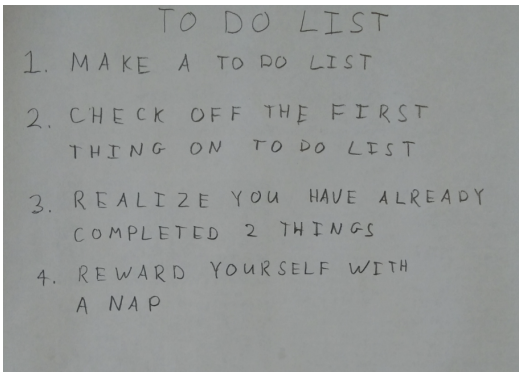


Figure 6: punctuation results in non-letter characters being labeled

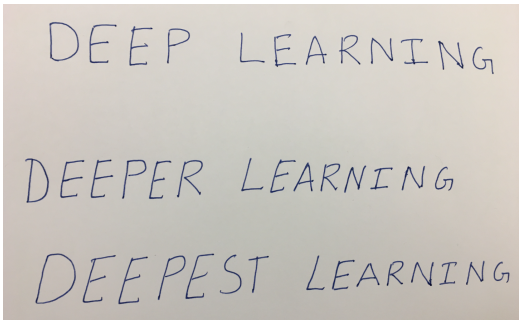
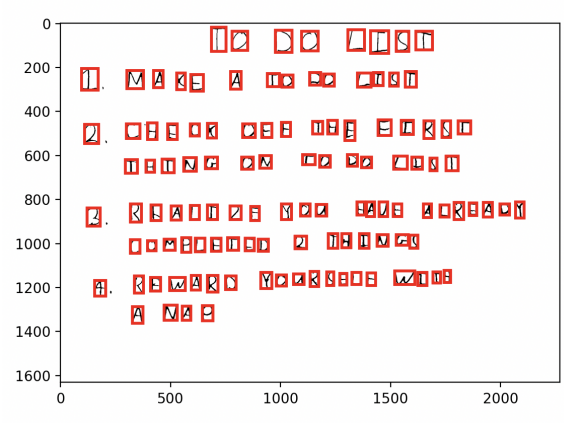
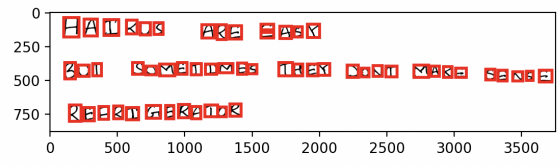
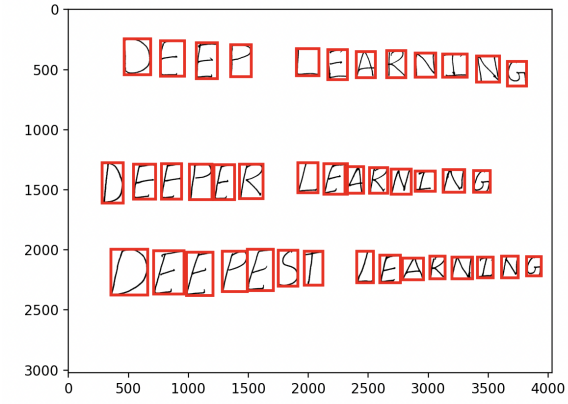


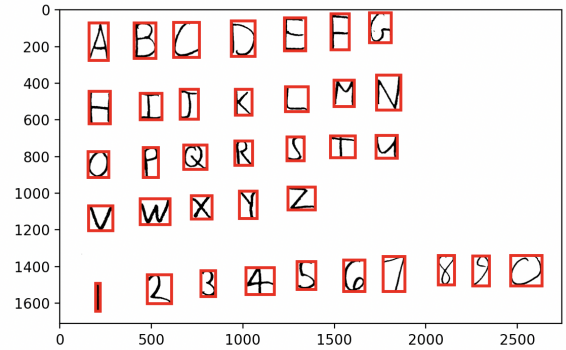
Figure 7: Letters beign too small or too big can result in skipped or added letters

Q5.3









Q5.4

Results: first: DEEPLEARMINGDEEPFKLEARNINGD1EPF5ILEARNING
second: TQDQLISTIN3IEATDDQLZST2CHGCRQFFTH5F8RSP8HZNG
QNPQDQL8ST3RSAL8ZBTQHBWEBQLRGQDTCQMPLBTID2PR8NG5QRBWQRDTQWRSBLBW8TQRN3P
third: ABLDEF6HIIKLMNQPQRSTWVWX9ZIZ3GS6789D
fourth: HAIKUSBYR8B4AGBUTSQMBTZMBBTRGXDDNTMAKGBBNGERBGR8GBRRTQR