Homework 2

1 i) No learning is occuring in the neural network with weights initialized to zero because, relu will turn outputs to zero, the gradients of the weights will 0. In terms of back propagation, the gradients will all be 0, thus no lerning/ will waynt updates will occur.

1 B) The large negative bias will close Relu gates, when many neurons return 0, gradients can't flow during backpropagation, portions of the network can go dormant as closed relu's can't chage input parameters.

1 ii) With Sigmoid the learning is a bit diffrent compared to relu, instead of 0 learning occurs In terms of back propgation eventually the weight's were learned the gradients could update Similarly, it took almost the entire weights but the 4000 iterations and could only learn initialization at 0 a simple linear boundry leving a will limit the full higher error compared to random power of gradients, weights which were able to the weights will learn a complex boundary with close be learned to be identical to 0 test error. A boundry could still be compared to random weights. learned because Sigmoid garuntees outputs between 0 and 1.

Homework 2

2 i) $(8 \cdot 5 \cdot 5 \cdot 3) + 8$

- 8: # of filters
- $5 \cdot 5$: square filter size
- 3: # of channels
- $+8$: add bias terms per filter

608 parameters

2 ii) NO zero padding

$$W_{out} = \frac{W - F + 2P}{S} + 1$$

($P = 0$)

image is square

$$\frac{W - F}{S} + 1 = 27$$

$$\frac{32 - 5}{1} + 1 = 28$$

28 × 28 × 3

3.)

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0.5 \\ 0.5 & 0.25 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0.5 & 1 \\ 0.25 & 0.5 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0.25 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0.5 & 0.25 \\ 1 & 0.5 \end{bmatrix}$$

3 ii)

→ $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

→ $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

→ $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

→ $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

~~They all have at~~ atleast a single

each image has all 1 in every quadrant if the 4x4 matrix is split up, every quadrant will find a max value of 1 resulting in identical pooling matrices.

problem c

Also in ~~pxt~~ Jupyter notebook

3c i)

1: 85.1233 %
2: 95.6817 %
3: 97.0417 %
4: 97.710 %
5: 98.0483 %
6: 98.3317 %
7: 98.5200 %
8: 98.6950 %
9: 98.8483 %
10: 98.9033 %

3c ii) After A/B testing different layers and activation functions, I was able to get to a final test accuracy of 98.9% by using conv2d, linear and dropout layers and relu and max pooling activation functions, the model was fairly simple.

3c iii) Yes, this is very computationally expensive, it can take a minute or two just to get a single epoch of training and 15-30 minutes to get the fully trained model, it's not an efficient way to hyper parameterize layers and activation functions.