STATS 6685 HWK4

Cody Grogan
A02313514

November 4, 2023

Problem 1

1)

Using standard gradient descent we see that there would be a large incentive to decrease λ as it is added to the cost of the network. Thus we would minimize the cost by setting λ to 0 which is not the point of the regularization coefficient. For the learning rate, we see that the cost function has no dependence on eta thus we cannot use gradient descent to optimize it.

2)

If we use an activation function with a derivative much higher we will experience the exploding gradient problem. This is because the gradient is a product of activation functions. Thus if the weights consistently grow then the gradients will also explode.

3)

a)

Starting with a graph of the derivative of the sigmoid function we see the maximum value it can take on is 0.25 at $wa + b = 0$. Thus if we substitute in this maximum we see,

$$|\frac{w}{4}| \geq 1 \tag{1}$$

$$|w| \geq 4 \tag{2}$$

Meaning that as long as we choose a b that results in the derivative being at its max value then to satisfy the inequality w must at least be 4.

b)

Starting with the equation we see,

$$|w\sigma'(wa+b)| \geq 1 \tag{3}$$
$$|w\sigma(wa+b)(1-\sigma(wa+b))| \geq 1 \tag{4}$$
$$|w\frac{1}{1+e^{-(wa+b)}}(1-\frac{1}{1+e^{-(wa+b)}})| \geq 1 \tag{5}$$
$$|w\left(\frac{1}{1+e^{-(wa+b)}}-\frac{1}{(1+e^{-(wa+b)})^2}\right)| \geq 1 \tag{6}$$
$$|w\frac{e^{-(wa+b)}}{(1+e^{-(wa+b)})^2}| \geq 1 \tag{7}$$

Now we know that the derivative of the sigmoid will always be greater than 0 so we can bring it out of the absolute value,

$$|w|\frac{e^{-(wa+b)}}{(1+e^{-(wa+b)})^2} \geq 1 \tag{8}$$
$$|w|e^{-(wa+b)} \geq (1+e^{-(wa+b)})^2 \tag{9}$$
$$|w|e^{-(wa+b)} \geq 1+2e^{-(wa+b)}+(e^{-(wa+b)})^2 \tag{10}$$
$$0 \geq 1+(2-|w|)e^{-(wa+b)}+(e^{-(wa+b)})^2 \tag{11}$$
$$\tag{12}$$

We know see we have a quadratic equation where $x = e^{-wa+b}$ so finding the roots we see,

$$r_1, r_2 = \frac{-(2-|w|) \pm \sqrt{(2-|w|)^2-4}}{2} \tag{13}$$
$$r_1, r_2 = \frac{|w|-2 \pm |w|\sqrt{1-\frac{4}{|w|}}}{2} \tag{14}$$
$$r_1, r_2 = \frac{|w|(1 \pm \sqrt{1-\frac{4}{|w|}})}{2} - 1 \tag{15}$$

Now because $|w| \geq 4$ we know that the polynomial will always have real roots and the function will be less than zero between the interval $r_1 \geq x \geq r_2$ where $r_1 < r_2$. Now selecting b = 0 because σ' is greatest around 0 we see,

$$r_1 \leq e^{-wa} \leq r_2 \tag{16}$$
$$\ln(r_1) \leq -wa \leq \ln(r_2) \tag{17}$$
$$-\ln(r_1) \geq wa \geq -\ln(r_2) \tag{18}$$

Now if we maintain the absolute value of w we know we can divide by w without changing the direction of the inequality,

$$-\frac{\ln(r_1)}{|w|} \geq a \geq -\frac{\ln(r_2)}{|w|} \tag{19}$$

Now we know that the interval in which a can lie is the left minus the right,

$$\frac{1}{|w|}(-\ln(r_1) + ln(r_2)) \tag{20}$$

$$\frac{1}{|w|}\ln(\frac{r_2}{r_1}) \tag{21}$$

Now using Vietta's formulas we know that $r_1 r_2 = \frac{c}{a}$ which in out case is $r_1 r_2 = 1$. Thus we know that $r_1 = \frac{1}{r_2}$. This leads the the simplification of the interval to,

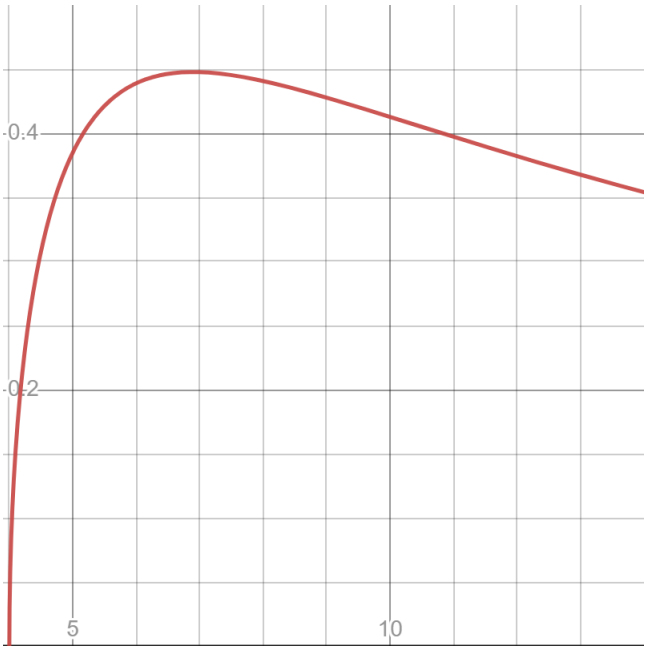
$$\frac{1}{|w|}\ln(r_2^2) \tag{22}$$

$$\frac{2}{|w|}\ln(r_2) \tag{23}$$

$$\frac{2}{|w|}\ln(\frac{|w|(1+\sqrt{1-\frac{4}{|w|}})}{2} - 1) \tag{24}$$

c)

Plotting the interval function over all w's we see,



Thus we see there is a peak at roughly w = 7 with a value of roughly .45.

4)

Using a μ of 1 we would see that the network would overshoot minima. Moreover, thinking about a ball rolling down a hill, if the ball has negative friction the ball will continuously accelerate and never settle. This then translates to the network because it will constantly move and overshoot minima.

Using a $\mu < 0$ we see that the velocity would appose the direction of the previous step. This would then have the effect of reducing our gradient steps and for low enough μ cause steps in the wrong direction.

Problem 2

1)

Grid search is the simplest to implement but doesn't exploit knowledge from searching the hyper-parameters space. It is also extremely inefficient form continuous hyper parameters.
Random search picks random combinations of hyper parameters and is thus slightly more efficient because it's search isn't exhaustive however it still does not exploit knowledge gained.

Successive halving is better than both Grid and Random search because it uses

the knowledge it gains to fully evaluate promising hyper parameter configurations. This is better than the other search method because it doesn't waste time evaluating obviously bad configurations. However at the start it isn't obvious whether to evaluate a few networks with less halving or a lot of networks with frequent halving

HyperBand is an extension of the successive halving approach but instead of having a set budget for each iteration the budget varies allowing for good but slow converging configurations to not be thrown out.

2)

Bayesian Optimization can be used for hyperparameter tuning by using a surrogate model for the validation accuracy and an acquisition function to that helps obtain the next guess for the best parameter value. This HPO method is very efficient in tuning a few hyper parameters but fails to scale because the maximum of the acquisition function must be found and it is sequential.
Genetic Algorithms essentially use evolution to select the best hyper parameters by finding the best few and combining their values. However the downfalls of this are that it introduces more hyper parameters and it still takes a significant amount of time to converge.
Particle swarm optimization uses a semi random search and information sharing to narrow in on good hyper parameter regions. The benefits of this is that it simpler and parallelizable. However it is sensitive to initialization because the space for the random search must be in a good region of the hyper parameter space.

3)

Trainables

Something that takes in hyper-parameters and then reports the score of the hyper-parameters after a predefined amount of iterations.

Tune Search Space

This is essentially a dictionary that defines valid hyper-parameter ranges and how they will be sampled.

Tune Trials

This is used to execute the tuning process and evaluate the best parameter configurations based on the trainable function.

Tune Search Algorithms

This is the search algorithm that will be used to find hyperparameters. There are various ones that are ready to use out of the box.

Tune Schedulers

This is how one would decide to perform early stopping on a hyper parameter set or slightly tweak the parameters.

Tune Result Grid

This is the object that contains all of the results from the parameter sweep. It has various metrics for finding the best hyper parameters.

Problem 3

Since the last layer is a fully connected layer we know it remains the same,

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \tag{25}$$

For the second equation we know that the max pooling layer does not have any weights for itself so we only have to compute $\delta_{j,k}^1$. Now doing the second equation where m is the number of output neurons.

$$\delta_{j,k}^1 = \frac{\partial C}{\partial z_{j,k}^1} \tag{26}$$

$$\delta_{j,k}^1 = \sum_m \frac{\partial C}{\partial z_l^3}\frac{\partial z_l^3}{\partial z_{jk}^1} \tag{27}$$

$$\delta_{j,k}^1 = \sum_m \frac{\partial C}{\partial z_l^3}\frac{\partial z_l^3}{\partial a_{pq}^2}\frac{a_{pq}^2}{\partial z_{jk}^1} \tag{28}$$

$$\tag{29}$$

Where we see that the only max pooling layer activation dependent on z_{jk}^1 is the one where $p = \frac{j}{2}$ and $q = \frac{k}{2}$. Also simplifying as one of the terms is defined,

$$\delta^1_{j,k} = \sum_m \delta^L_l \frac{\partial z^3_l}{\partial a^2_{pq}} \frac{\partial a^2_{pq}}{\partial z^1_{jk}} \tag{30}$$

$$\delta^1_{j,k} = \sum_m \delta^L_l \frac{\partial z^3_l}{\partial a^2_{pq}} \frac{\partial a^2_{pq}}{\partial a^1_{jk}} \frac{a^1_{jk}}{z^1_{jk}} \tag{31}$$

$$\delta^1_{j,k} = \sum_m \delta^L_l w^3_{l;p,q} \frac{\partial a^2_{pq}}{\partial a^1_{jk}} \sigma'(z^1_{jk}) \tag{32}$$

$$\tag{33}$$

Now looking at the max pooling layer we know the it will be 1 when our the activation is chose was a^1_{jk} and 0 otherwise. Thus we get,

$$\delta^1_{jk} = \begin{cases} 0 & \text{if } a^1_{jk} \neq max\{\ldots\} \\ \sum_m \delta^L_l w^3_{l;pq} \sigma'(z^1_{jk}) & \text{if } a^1_{jk} = max\{\ldots\} \end{cases} \tag{34}$$

Now for the third equation we find the partial of the cost with respect to the two weights b^1 and b^3_l. Using our previous derivations we see,

$$\frac{\partial C}{\partial b^3_l} = \frac{\partial C}{\partial z^L_l} \frac{\partial z^L_l}{\partial b^3_l} \tag{35}$$

$$\frac{\partial C}{\partial b^3_l} = \delta^L_l \tag{36}$$

Now for the bias of the first layer,

$$\frac{\partial C}{\partial b^1} = \sum_{jk} \frac{\partial C}{\partial z^1_{jk}} \frac{\partial z^1_{jk}}{\partial b^1} \tag{37}$$

$$\frac{\partial C}{\partial b^1} = \sum_{jk} \delta^1_{jk} \tag{38}$$

Now for the final equation we take into account the weights in the last layer and the first layer. Starting with the last layer,

$$\frac{\partial C}{\partial w^L_{l;jk}} = \frac{\partial C}{\partial z^L_l} \frac{\partial z^L_l}{\partial w^L_{l;jk}} \tag{39}$$

$$\frac{\partial C}{\partial w^L_{l;jk}} = \delta^L_l \frac{\partial}{\partial w^L_{l;jk}} (b_l + \sum_{jk} w_{l;jk} a^2_{jk}) \tag{40}$$

$$\frac{\partial C}{\partial w^L_{l;jk}} = \delta^L_l a^2_{jk} \tag{41}$$

Now for the convolutional layers,

$$\frac{\partial C}{\partial w^1_{lm}} = \sum_{jk} \frac{\partial C}{\partial z^1_{jk}} \frac{\partial z^1_{jk}}{\partial w^1_{lm}} \tag{42}$$

$$\frac{\partial C}{\partial w^1_{lm}} = \sum_{jk} \delta^1_{jk} \frac{\partial z^1_{jk}}{\partial w^1_{lm}} \tag{43}$$

$$\frac{\partial C}{\partial w^1_{lm}} = \sum_{jk} \delta^1_{jk} a^0_{j+l,k+m} \tag{44}$$

$$\tag{45}$$

Problem 4

1)

My final test accuracy for this section was 99.22%. I designed my network to have 2 convolutional layers expanding from $1- > 16- > 32$ channels and 2 hidden layers. One from the convolutional layer and one to the fully connected layers. The input to the first fully connected layer having dropout. I used standard SGD with a learning rate: 0.022, dropout: 0.4, and weight decay: 0.0005, and hidden nodes at 256.

2)

The get loss function is effectively summing together the absolute value of all of the weights in the entire network and multiplying by the regularization coefficient. It seemed like both approaches were fairly comparable after 100 epochs but l1 regularization performed slightly better. However the reinitialized weights reached an accuracy peak slightly before the network without initialization.

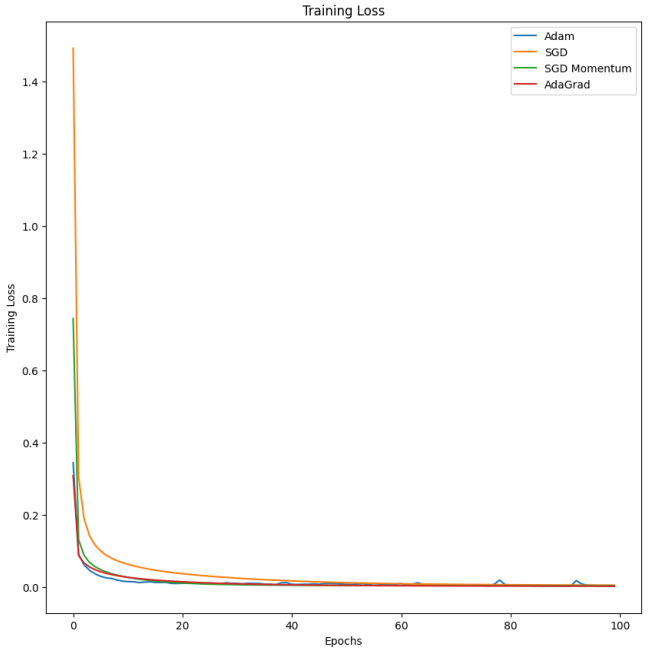
3)

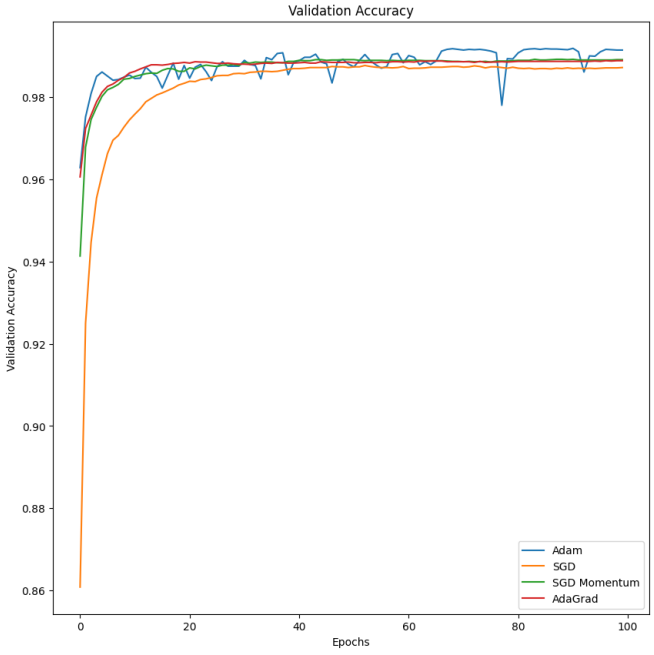
a)

For the four approaches I used:

- Adam: lr: 0.001, lambda1: 0.000001, Accuracy: 99.41%
- SGD: lr: 0.025, lambda1: 0.000001, Accuracy: 98.92%
- SGD w/ momentum: lr: 0.025, mom: 0.7, lambda1: 0.000001 Accuracy: 99.05%
- AdaGrad: lr: 0.25, lambda1: 0.000001, Accuracy: 99.09%

Plots





From the graph we can see that all of the approaches do fairly well at picking learning rates. However the Adam optimizer tends to out perform the others

because it tends not to get stuck in a extremely low learning rate. As a result it had significantly higher validation accuracy at times. In addition the Adam optimizer requires almost no tuning.

b)

I chose to use the Adam optimizer and did not have to touch the global learning rate of 0.001 of as it performed the best with the previous rate. The final test accuracy was 99.21% which is a bit better than the training without normalization. As a result batch normalization most definitely does help in training.

Problem 6

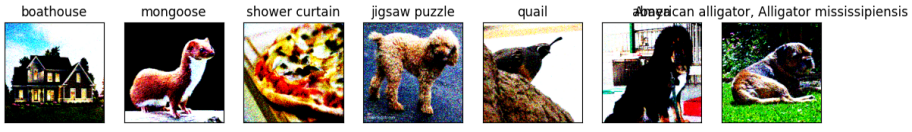
1)

For the data augmentation strategy I went with random crops and random horizontal flips to augment the data.

2)

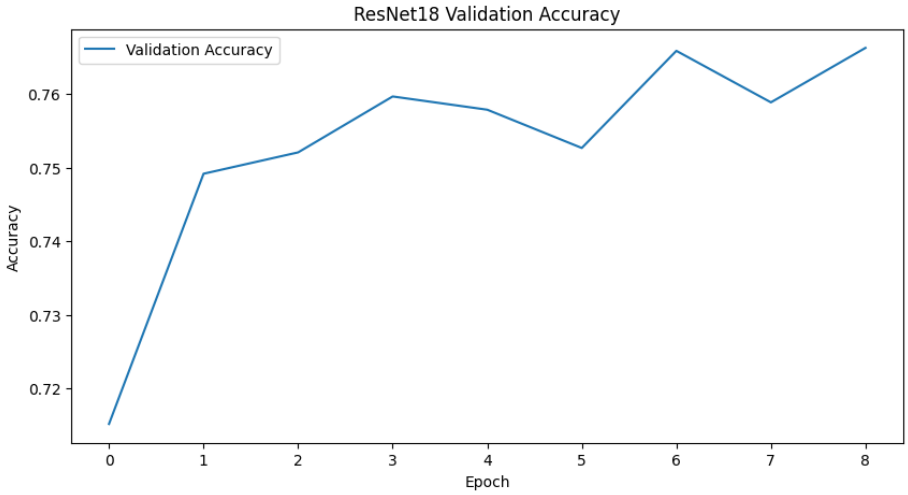
Using resnet, the unnoised test pictures give accurate results,

However after adding noise to the picture thhe network incorrectly classifies the images even though they look identical,



3)

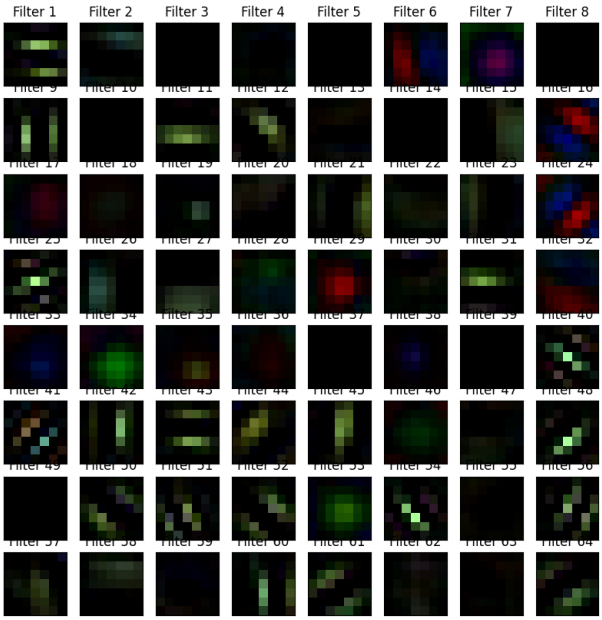
Modifying the network to output 10 classes and training only the final layer I get the following results. For my hyper-parameters I chose weight decay: 0.0001, Adam optimizer, lr: 0.001, early stopper with a improvement threshold of 0.01 patience of 5, and cross entropy loss.



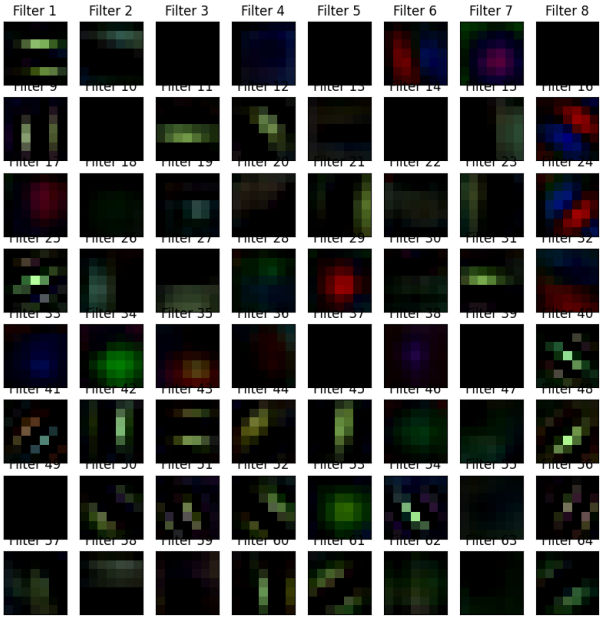
4)

Plotting the stock and tuned first convolutional filters.

Stock ResNet18 Conv1 Filters



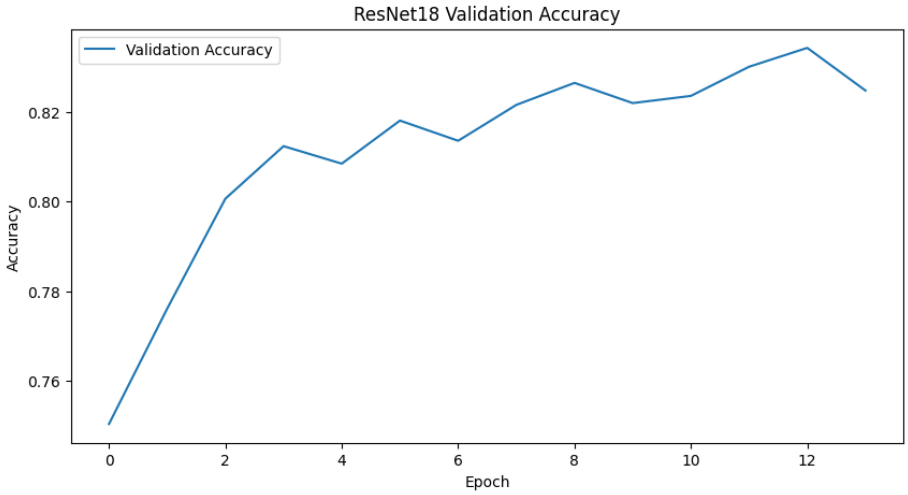
Fine Tuned ResNet18 Conv1 Filters



After visual inspection of the filters we see that relatively little has changed in the first convolutional layer after training on CIFAR10. The only noticeable

change is in filter 4 where there seems to a bit of blue recognition on the right side of the image.

Plotting the validation accuracy,



As can be seen, the model that had more fine tuning had substantially better accuracy than the first training. This is because the network is allowed to specialize more on the picture format of CIFAR which was substantially more blurry because of the resize. I chose the hyper-parameters for this section the same as the previous so a more fair comparison could be made between training the final layer and training multiple layers.