Assignment 4 - Exploring Hyperparameters
Winter 2024

Atieh Armin
Student ID : 14658308

1 Theory

Whenever possible, please leave your answers as fractions so the question of rounding and loss of precision therein does not come up.

1. What would the *one–hot encoding* be for the following set of multi–class labels (5pts)?

$$Y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 2 \\ 3 \\ 0 \end{bmatrix}$$

Solution:

$$Y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

2. Given inputs $X = \begin{bmatrix} 1 & 2 & -1 \\ 0 & -4 & -4 \end{bmatrix}$ and the fully connected layer having weights $W = \begin{bmatrix} -1 & 1 & 1 \\ 2 & 2 & 0 \\ 4 & -1 & -3 \end{bmatrix}$, $b = \begin{bmatrix} 1 & 0 & 2 \end{bmatrix}$, what is the output of the following architecture? Show intermediate computations. For simplicity *do not* z-score your inputs. (5pts)

$$Input \rightarrow \text{Fully Connected} \rightarrow \text{Softmax}$$

Solutuin:

Output of Fully Connected layer :

$$z = xW + b = \begin{bmatrix} 1 & 2 & -1 \\ 0 & -4 & -4 \end{bmatrix} \begin{bmatrix} -1 & 1 & 1 \\ 2 & 2 & 0 \\ 4 & -1 & -3 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 2 \end{bmatrix} = \begin{bmatrix} -1 & 6 & 4 \\ -24 & -4 & 12 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 6 & 6 \\ -23 & -4 & 14 \end{bmatrix}$$

Output of Softmax layer :

$$\hat{Y} = g(z) = \frac{e^z}{\sum_i e^{z_i}} = \begin{bmatrix} \frac{1}{1+e^6+e^6} & \frac{e^6}{1+e^6+e^6} & \frac{e^6}{1+e^6+e^6} \\ \frac{e^{-23}}{e^{-23}+e^{-4}+e^{14}} & \frac{e^{-4}}{e^{-23}+e^{-4}+e^{14}} & \frac{e^{14}}{e^{-23}+e^{-4}+e^{14}} \end{bmatrix} \approx \begin{bmatrix} 0.001238 & 0.499381 & 0.499381 \\ 0 & 0 & 1 \end{bmatrix}$$

3. Using the same setup as the previous question, what are the gradients to update the fully connected layer's weights (both W and b) if we're using a cross-entropy objective function if we have three (3) total classes as the observations' targets are $Y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$? Make sure to show the intermediate gradients being passed backwards to make these computations. (5pts)

Solution:

(a) The $\hat{Y}$ is equal to $\begin{bmatrix} 0.001238 & 0.499381 & 0.499381 \\ 8.53304750 * 10^{-17} & 1.52299795 * 10^{-8} & 1 \end{bmatrix}$

(b) Since we have three classes, we have to do the one-hot encoding for Y. So we'll have $Y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$

(c) We use chain rules to calculate the gradients: $\frac{\partial J}{\partial z} = \frac{\partial J}{\partial \hat{Y}} * \frac{\partial \hat{Y}}{\partial z}$

(d) So we have to first calculate $\frac{\partial J}{\partial \hat{Y}}$ based on cross entropy gradient:

$\frac{\partial J}{\partial \hat{Y}} = -\frac{Y}{\hat{Y}+\epsilon} = \begin{bmatrix} \frac{-1}{0.001238+\epsilon} & 0 & 0 \\ 0 & \frac{-1}{1.52299795*10^{-8}+\epsilon} & 0 \end{bmatrix}$

So if epsilon is 10^{-6} we will have $\frac{\partial J}{\partial \hat{Y}} = \begin{bmatrix} \frac{-1}{0.001239} & 0 & 0 \\ 0 & \frac{-1}{1.0152299795*10^{-6}} & 0 \end{bmatrix} = \begin{bmatrix} -807.1025 & 0 & 0 \\ 0 & -65659970 & 0 \end{bmatrix}$

(e) We can calculate $\frac{\partial \hat{Y}}{\partial z}$ based on softmax gradient. $\frac{\partial \hat{Y}}{\partial z} = diag(\hat{Y}) - \hat{Y}^T \hat{Y}$

=> We will have a tensor with two matrices. The first matrix regarding the first input is:

$$\frac{\partial \hat{Y}}{\partial z} = \begin{bmatrix} 0.001238 & 0 & 0 \\ 0 & 0.499381 & 0 \\ 0 & 0 & 0.499381 \end{bmatrix} - \begin{bmatrix} 0.001238 \\ 0.499381 \\ 0.499381 \end{bmatrix} \begin{bmatrix} 0.001238 & 0.499381 & 0.499381 \end{bmatrix}$$

$$= \begin{bmatrix} 0.001238 & 0 & 0 \\ 0 & 0.499381 & 0 \\ 0 & 0 & 0.499381 \end{bmatrix} - \begin{bmatrix} 0.000001 & 0.000618 & 0.000618 \\ 0.000618 & 0.249381 & 0.249381 \\ 0.000618 & 0.249381 & 0.249381 \end{bmatrix}$$

$$= \begin{bmatrix} 0.001237 & -0.000618 & -0.000618 \\ -0.000618 & 0.25 & -0.249381 \\ -0.000618 & -0.249381 & 0.25 \end{bmatrix}$$

The second matrix regarding the second input is:

$$\frac{\partial \hat{Y}}{\partial z} = \begin{bmatrix} 8.53304750 * 10^{-17} & 0 & 0 \\ 0 & 1.52299795 * 10^{-8} & 0 \\ 0 & 0 & 1 \end{bmatrix} -$$

$$\begin{bmatrix} 8.53304750 * 10^{-17} \\ 1.52299795 * 10^{-8} \\ 1 \end{bmatrix} \begin{bmatrix} 8.53304750 * 10^{-17} & 1.52299795 * 10^{-8} & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 8.53304750 * 10^{-17} & 0 & 0 \\ 0 & 1.52299795 * 10^{-8} & 0 \\ 0 & 0 & 1 \end{bmatrix} -$$

$$\begin{bmatrix} 7.28128 * 10^{-33} & 1.29958 * 10^{-24} & 8.53304750 * 10^{-17} \\ 1.29958 * 10^{-24} & 2.31952 * 10^{-16} & 1.52299795 * 10^{-8} \\ 8.53304750 * 10^{-17} & 1.52299795 * 10^{-8} & 1 \end{bmatrix}$$
$$= \begin{bmatrix} 8.5330475 * 10^{-17} & -1.29958 * 10^{-24} & -8.53304750 * 10^{-17} \\ -1.29958 * 10^{-24} & 1.52299793 * 10^{-8} & -1.52299795 * 10^{-8} \\ -8.53304750 * 10^{-17} & -1.52299795 * 10^{-8} & 0 \end{bmatrix}$$

(f) So now we have to do tensor multiplication between $\frac{\partial J}{\partial \hat{Y}}$ and $\frac{\partial \hat{Y}}{\partial \hat{z}}$ and as result we will have:
For the first observation we will have:
$$\begin{bmatrix} -807.1025 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.001237 & -0.000618 & -0.000618 \\ -0.000618 & 0.25 & -0.249381 \\ -0.000618 & -0.249381 & 0.25 \end{bmatrix}$$
$$= \begin{bmatrix} -0.998385 & 0.498789 & 0.498789 \end{bmatrix}$$
And for the second observation we will have:
$$\begin{bmatrix} 0 & -65659970 & 0 \end{bmatrix} \begin{bmatrix} 8.5330475 * 10^{-17} & -1.29958 * 10^{-24} & -8.53304750 * 10^{-17} \\ -1.29958 * 10^{-24} & 1.52299793 * 10^{-8} & -1.52299795 * 10^{-8} \\ -8.53304750 * 10^{-17} & -1.52299795 * 10^{-8} & 0 \end{bmatrix}$$
$$= \begin{bmatrix} 8.5330384 * 10^{-17} & -0.999999 & 0.999999 \end{bmatrix} \approx \begin{bmatrix} 0 & -1 & 1 \end{bmatrix}$$
So overall we will have:
$$\frac{\partial J}{\partial z} = \frac{\partial J}{\partial \hat{Y}} * \frac{\partial \hat{Y}}{\partial \hat{z}} = \begin{bmatrix} -0.998385 & 0.498789 & 0.498789 \\ 0 & -1 & 1 \end{bmatrix}$$
This process is done with einsum function in python!

(g) Now we can calculate $\frac{\partial J}{\partial W} = \frac{1}{N} X^T \frac{\partial J}{\partial z}$.
$$\frac{\partial J}{\partial W} = \frac{1}{N} X^T \frac{\partial J}{\partial z} = \frac{1}{2} \begin{bmatrix} 1 & 0 \\ 2 & -4 \\ -1 & -4 \end{bmatrix} \begin{bmatrix} -0.998385 & 0.498789 & 0.498789 \\ 0 & -1 & 1 \end{bmatrix}$$
$$= \frac{1}{2} \begin{bmatrix} -0.998385 & 0.498789 & 0.498789 \\ -1.99677 & 4.997578 & -3.002422 \\ 0.998385 & 3.501211 & -4.498789 \end{bmatrix}$$
$$=> \frac{\partial J}{\partial W} = \begin{bmatrix} -0.4991925 & 0.2493945 & 0.2493945 \\ -0.998385 & 2.498789 & -1.501211 \\ 0.4991925 & 1.7506055 & -2.2493945 \end{bmatrix}$$

(h) And the updating rule for W will be : $W = W + \eta(-\frac{\partial J}{\partial W})$

(i) Now we can move on to $\frac{\partial J}{\partial b}$ which is the average of each column in $\frac{\partial J}{\partial z}$
$\frac{\partial J}{\partial b} = \begin{bmatrix} -0.4991925 & -0.2506055 & 0.7493945 \end{bmatrix}$

(j) And the updating rule for b will be : $b = b + \eta(-\frac{\partial J}{\partial b})$

4. Given the objective function $J = \frac{1}{4}(x_1 w_1)^4 - \frac{4}{3}(x_1 w_1)^3 + \frac{3}{2}(x_1 w_1)^2$ (*I know you already did this in HW3, but it will be relevant for HW4 as well*):

(a) What is the gradient $\frac{\partial J}{\partial w_1}$ (1pt)?
We can use chain rule to calculate the gradients.
If we consider $(x_1 w_1)$ as h, we will have $J = \frac{1}{4}(h)^4 - \frac{4}{3}(h)^3 + \frac{3}{2}(h)^2$.
$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial h} * \frac{\partial h}{\partial w_1}$
$\frac{\partial J}{\partial h} = \frac{\partial(\frac{1}{4}(h)^4 - \frac{4}{3}(h)^3 + \frac{3}{2}(h)^2)}{\partial h} = (h)^3 - 4(h)^2 + 3(h) = (x_1 w_1)^3 - 4(x_1 w_1)^2 + 3(x_1 w_1)$

$\frac{\partial h}{\partial w_1} = \frac{\partial (x_1 w_1)}{\partial w_1} = x_1$

$=> \frac{\partial J}{\partial w_1} = ((x_1 w_1)^3 - 4(x_1 w_1)^2 + 3(x_1 w_1)) * x_1$

(b) What are the locations of the extrema points for your objective function if $x_1 = 1$? Recall that to find these you set the derivative to zero and solve for, in this case, w_1. (3pts)

If $x_1 = 1$, $\frac{\partial J}{\partial w_1} = ((w_1)^3 - 4(w_1)^2 + 3(w_1))$.

We can find the extrema points if we put $\frac{\partial J}{\partial w_1} = 0$.

$=> \frac{\partial J}{\partial w_1} = ((w_1)^3 - 4(w_1)^2 + 3(w_1)) = 0$

$=> (w_1)(w_1 - 3)(w_1 - 1) = 0$

=> The exterma points are $w_1 = 0, w_1 = 1$, and $w_1 = 3$

(c) What does J evaluate to at each of your extrema points, again when $x_1 = 1$ (1pts)?

If $x_1 = 1$, $J = \frac{1}{4}(w_1)^4 - \frac{4}{3}(w_1)^3 + \frac{3}{2}(w_1)^2$.

$w_1 = 0 => J = 0$

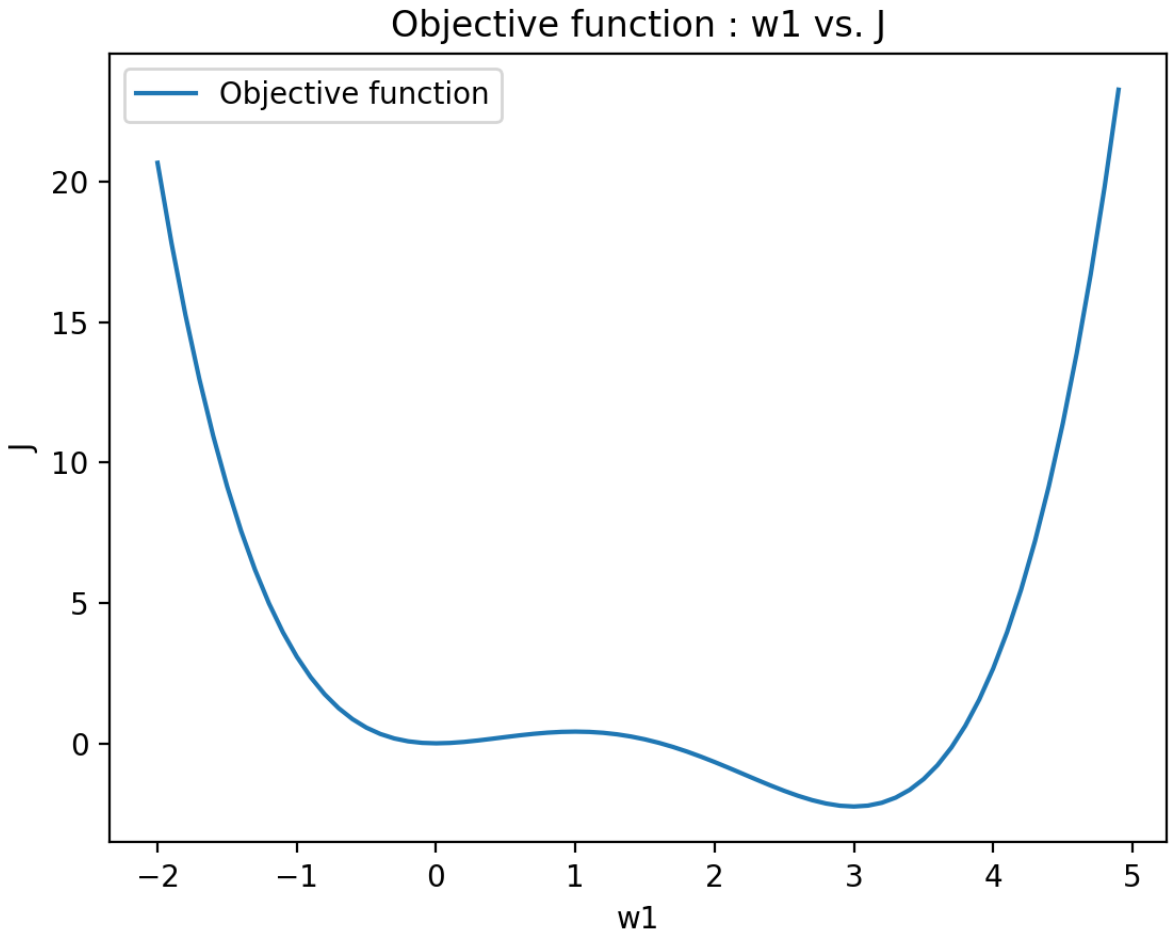
$w_1 = 1 => J = \frac{1}{4} - \frac{4}{3} + \frac{3}{2} = \frac{3}{12} - \frac{16}{12} + \frac{18}{12} = \frac{3-16+18}{12} = \frac{5}{12} \approx 0.417$

$w_1 = 3 => J = \frac{1}{4}(3)^4 - \frac{4}{3}(3)^3 + \frac{3}{2}(3)^2 = \frac{1}{4}(81) - \frac{4}{3}(27) + \frac{3}{2}(9) = \frac{81}{4} - (4 * 9) + \frac{27}{2} = \frac{81-36*4+27*2}{4} = \frac{81-144+54}{4} = \frac{135-144}{4} = \frac{-9}{4} = -2.25$

2 Visualizing an Objection Function

For the next few parts we'll use the objective function $J = \frac{1}{4}(x_1 w_1)^4 - \frac{4}{3}(x_1 w_1)^3 + \frac{3}{2}(x_1 w_1)^2$ from the theory section. First let's get a look at this objective function. Using $x_1 = 1$, plot w_1 vs J, varying w_1 from -2 to +5 in increments of 0.1. You will put this figure in your report.

Here is the plot:



3 Exploring Model Initialization Effects

Let's explore the effects of choosing different initializations for our parameter(s). In the theory part you derived the partial of $J = \frac{1}{4}(x_1 w_1)^4 - \frac{4}{3}(x_1 w_1)^3 + \frac{3}{2}(x_1 w_1)^2$ with respect to the parameter w_1. Now you will run gradient descent on this for four different initial values of w_1 to see the effect of weight initialization and local solutions.

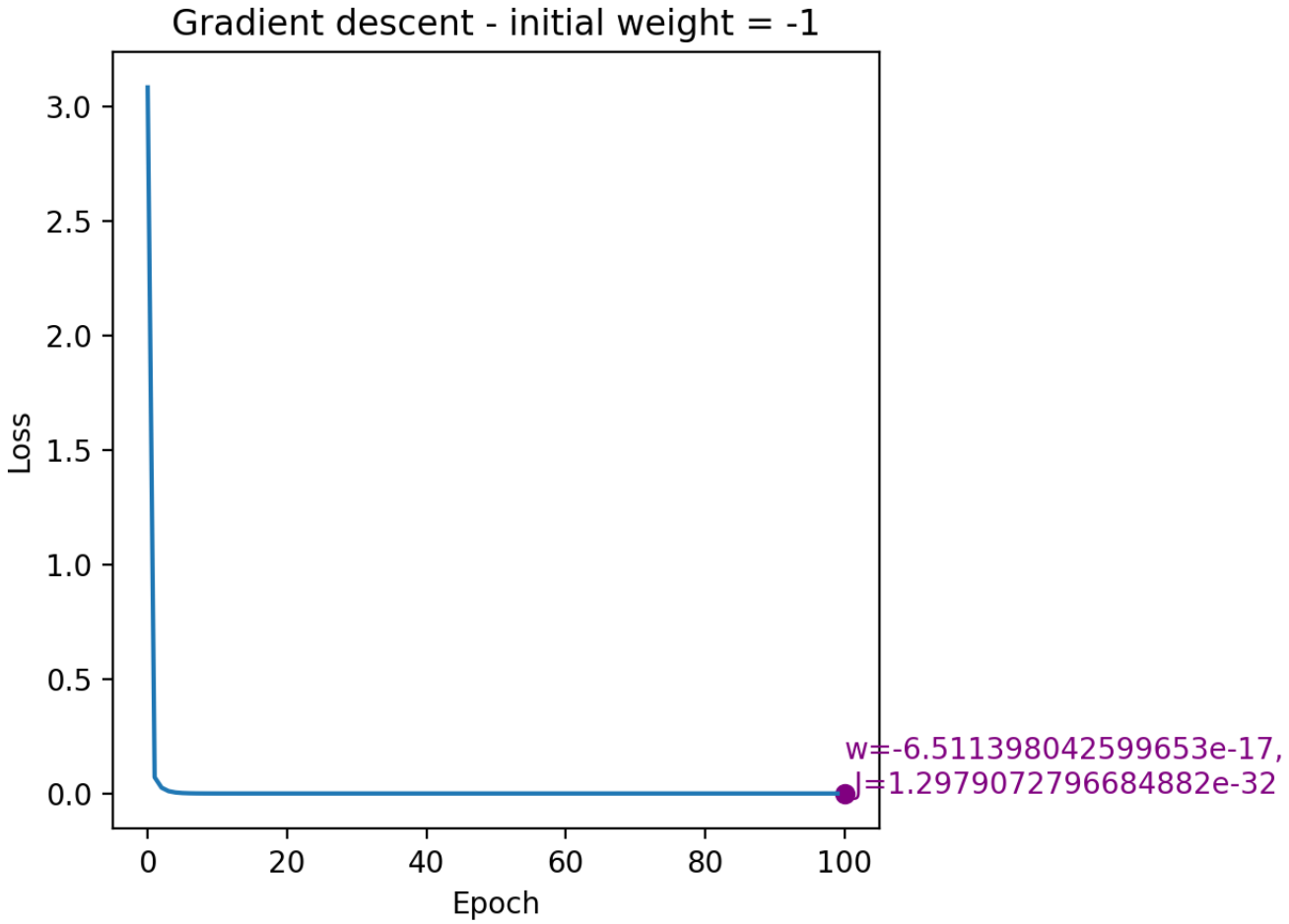
Perform gradient descent as follows:

- Run through 100 epochs.
- Use a learning rate of $\eta = 0.1$.
- Evaluate J at each epoch so we can see how/if it converges.
- Assume our only data point is $x = 1$

In your report provide the four plots of epoch vs. J, superimposing on your plots the final value of w_1 and J once 100 epochs has been reached. In addition, based on your visualization of the objective function in Section 2, describe why you think w_1 converged to its final place in each case.

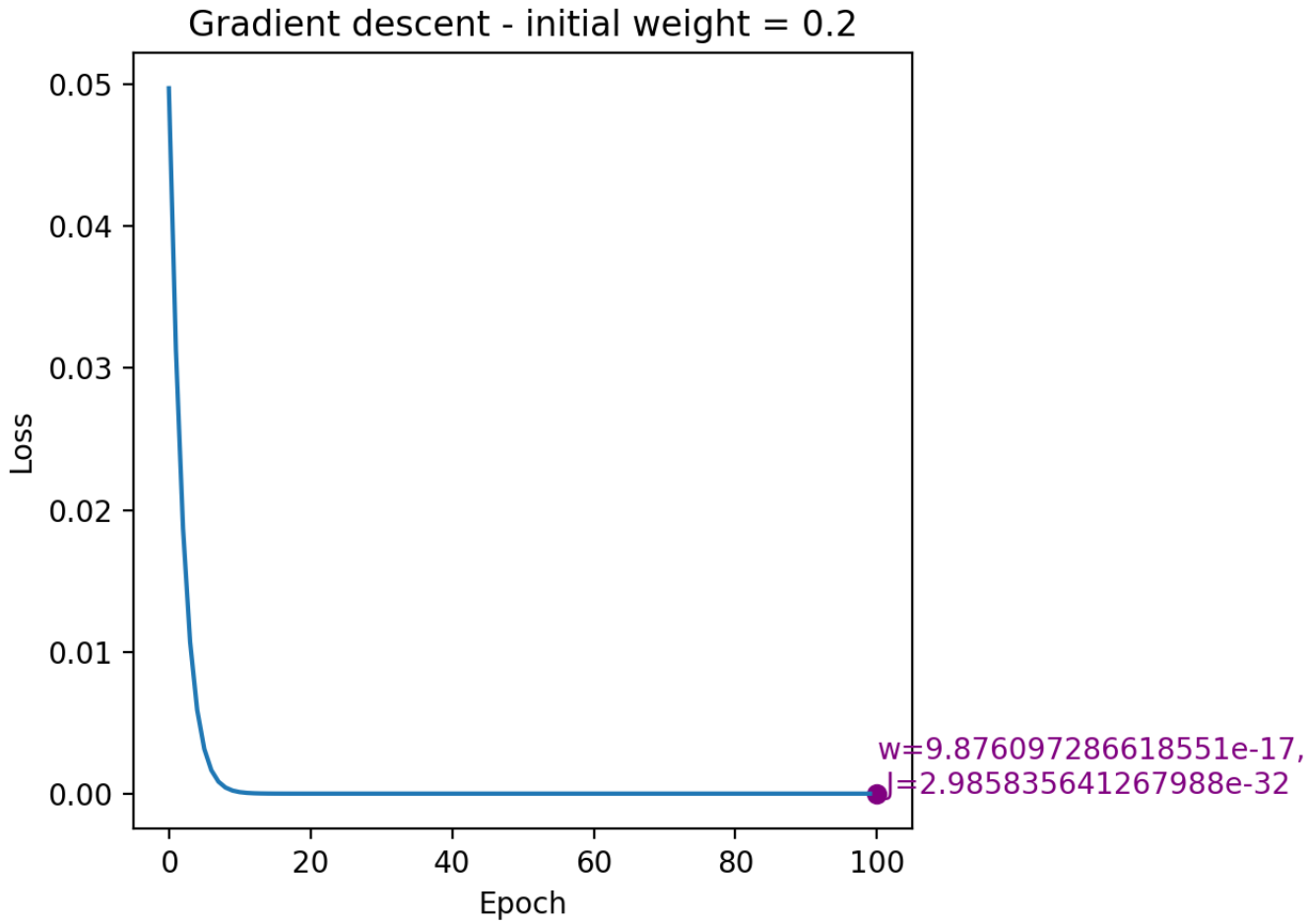
Answer:

- $w_1 = -1$.



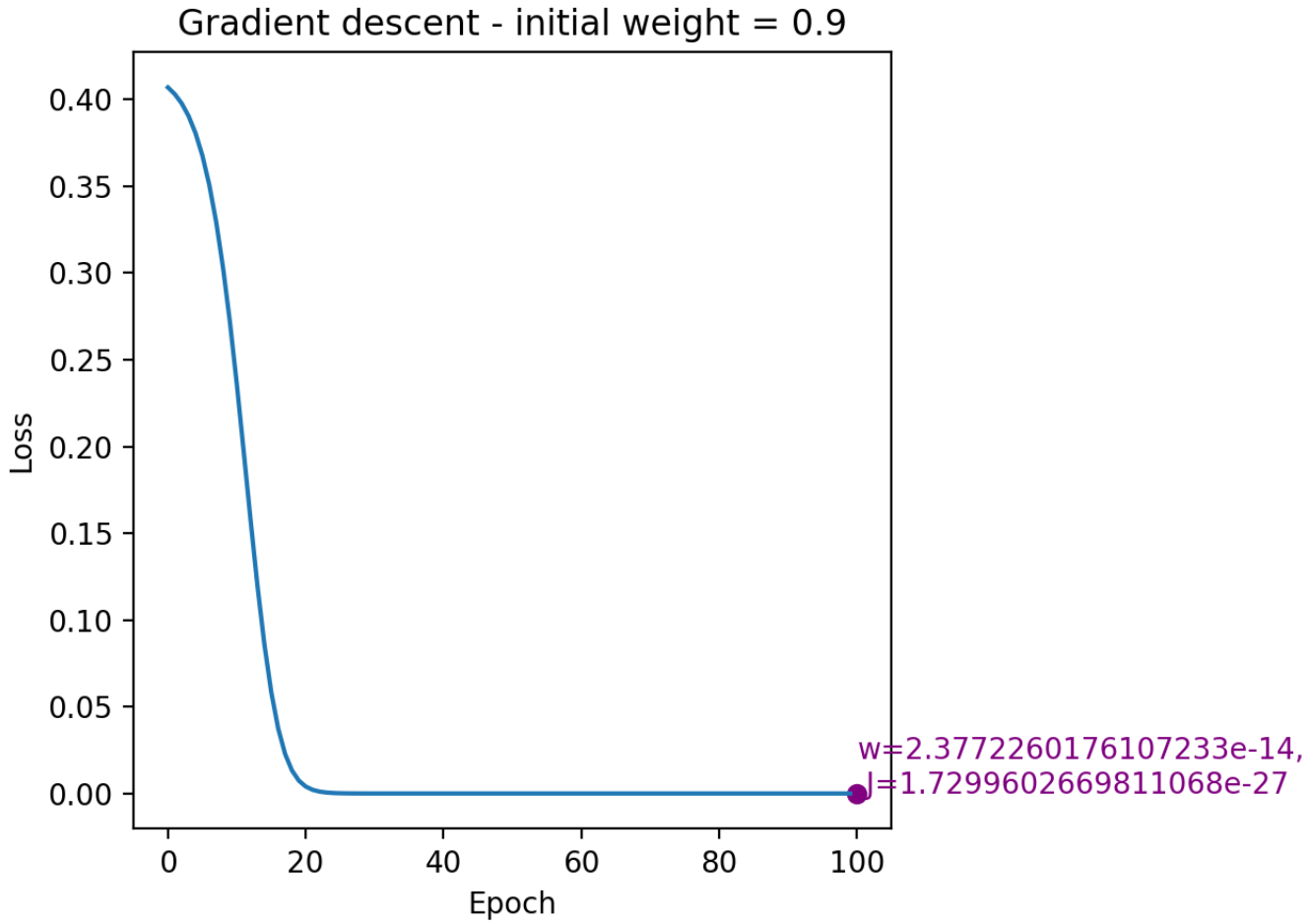
Here, the weight starts on the left side of the parabola, where the slope is negative. Gradient descent increases the value of w iteratively to find the minimum. Given that the slope changes more gradually on this side of the parabola (it's less steep), w updates slowly towards the minimum. The final weight converges to a point near the bottom of the parabola, as indicated by the final loss approaching zero. However, w updates would stuck in this local minimum point either because the learning rate is low or because of the low number of epochs. Nonetheless, this local minimum performs nearly as well as the global one and it is acceptable halting point.

- $w_1 = 0.2$.



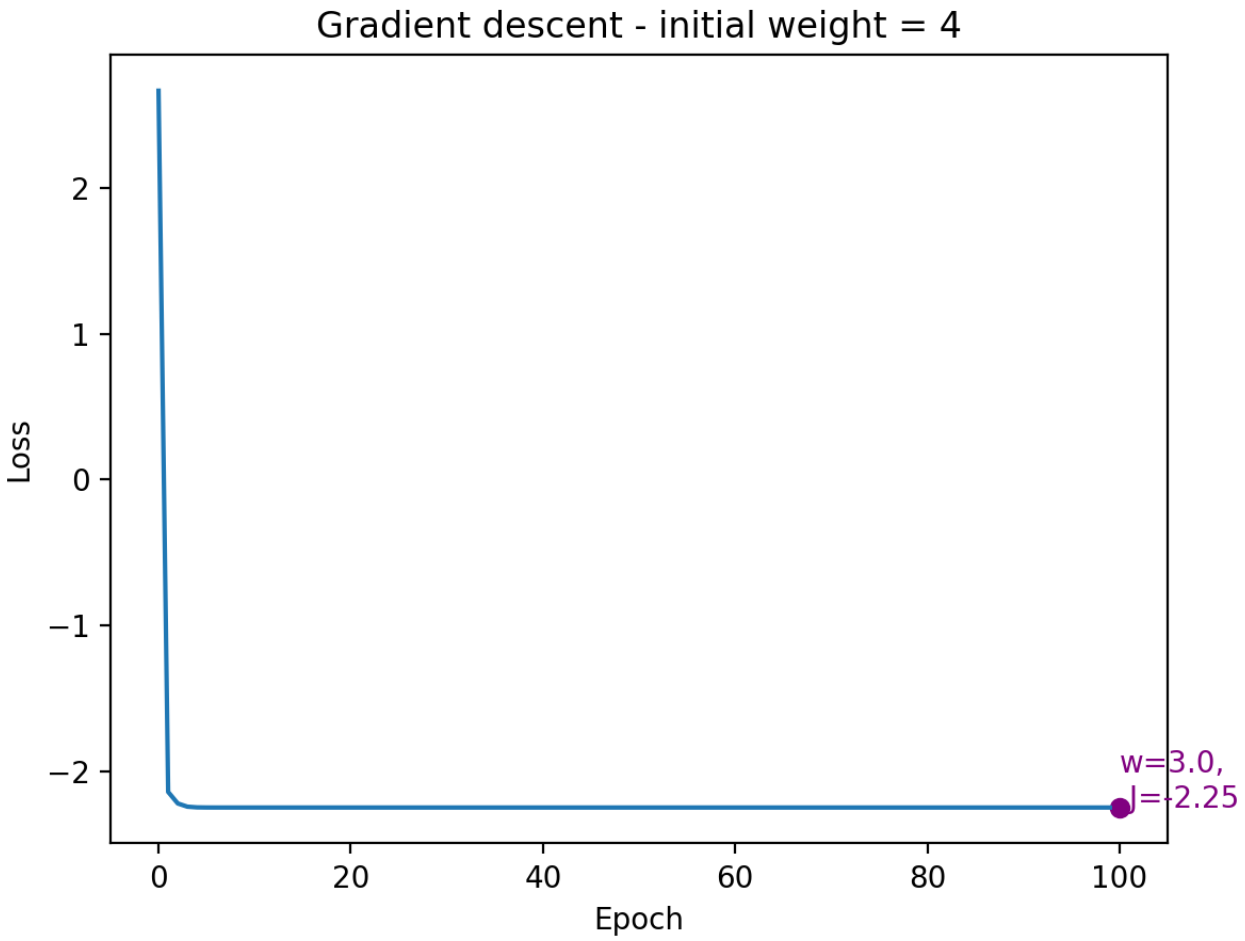
With an initial weight of 0.2, the weight is already close to the bottom of the parabola. Since the initial position is near the minimum of the objective function, w converges quickly to a value very close to the global minimum, indicated by the small final loss. The algorithm makes only a few updates before the slope becomes too small to make significant changes to w, hence converging rapidly to the minimum. However, w updates would stuck in the local minimum point either because the learning rate is low or because of the low number of epochs.

- $w_1 = 0.9$.



With an initial weight of 0.9, the trajectory is similar to that with an initial weight of 0.2. However, it is near to the saddle point and really close to the minimum. So the starting loss is very low, but it gets lower, because the algorithm would get out of the saddle point and the new weights start on the left side of the parabola, where the slope is negative. Gradient descent increases the value of w iteratively to find the minimum. Given that the slope changes more gradually on this side of the parabola (it's less steep), w updates slowly towards the minimum. The final weight converges to a point near the bottom of the parabola, as indicated by the final loss is very small. However, the final loss is not the global minimum of the objective function, and that is because the number of epochs were low and the algorithm stopped when the weights were almost reaching the minimum.

- $w_1 = 4$.



The gradient descent started with a high initial weight of 4, and the objective function's plot suggests that this starting point is on the right side of the parabola, where the slope is positive. As gradient descent moves in the direction of the steepest descent (i.e., the negative gradient), the weight updated towards the left, seeking the minimum. The final value of w converged to 3 because the gradient becomes very small near the minimum, causing the updates to w to become tiny, eventually stopping at the global minimum.

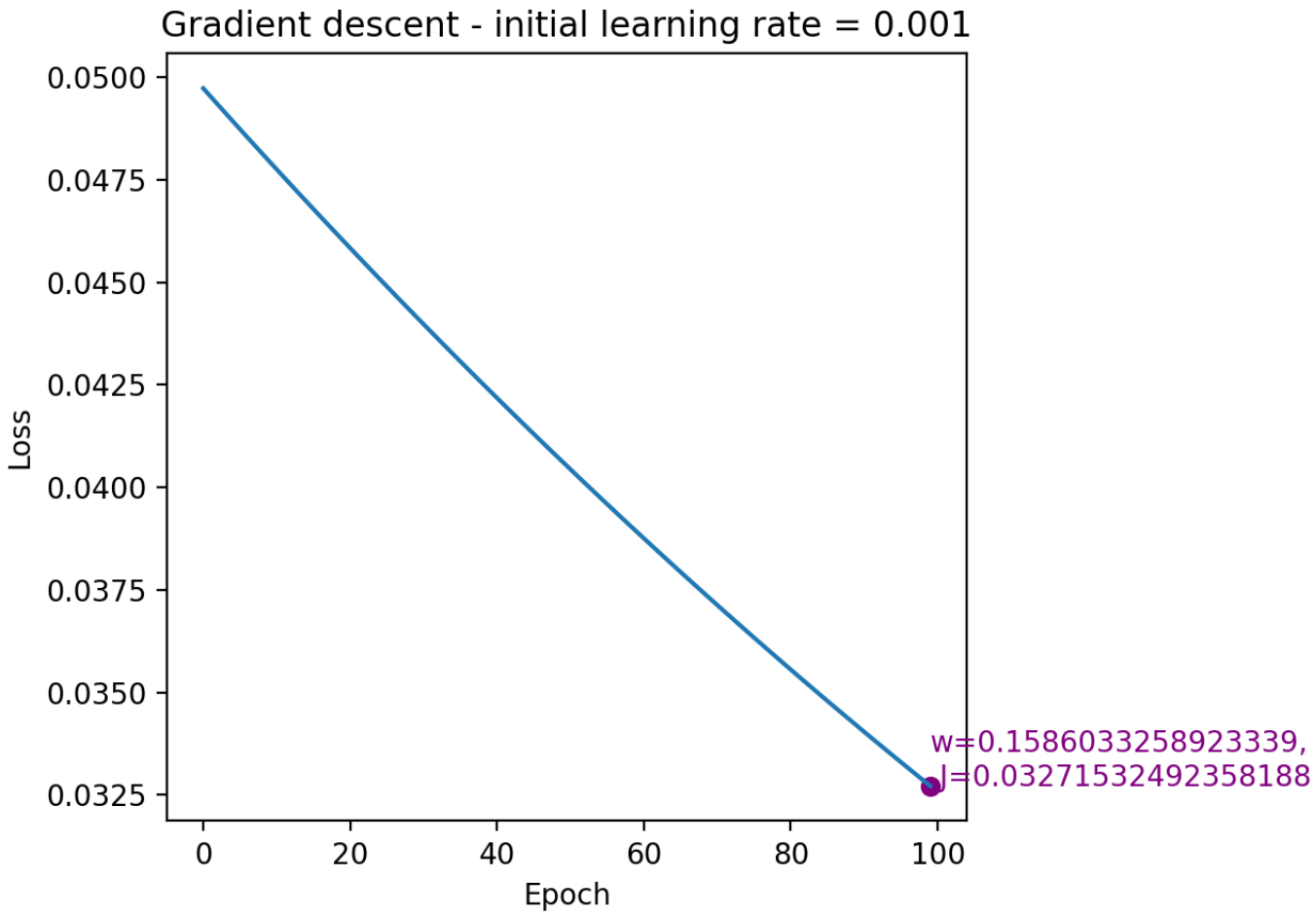
4 Explore Learning Rate Effects

Next we're going to look at how your choice of learning rate can affect things. We'll use the same objective function as the previous sections, namely $J = \frac{1}{4}(x_1 w_1)^4 - \frac{4}{3}(x_1 w_1)^3 + \frac{3}{2}(x_1 w_1)^2$.

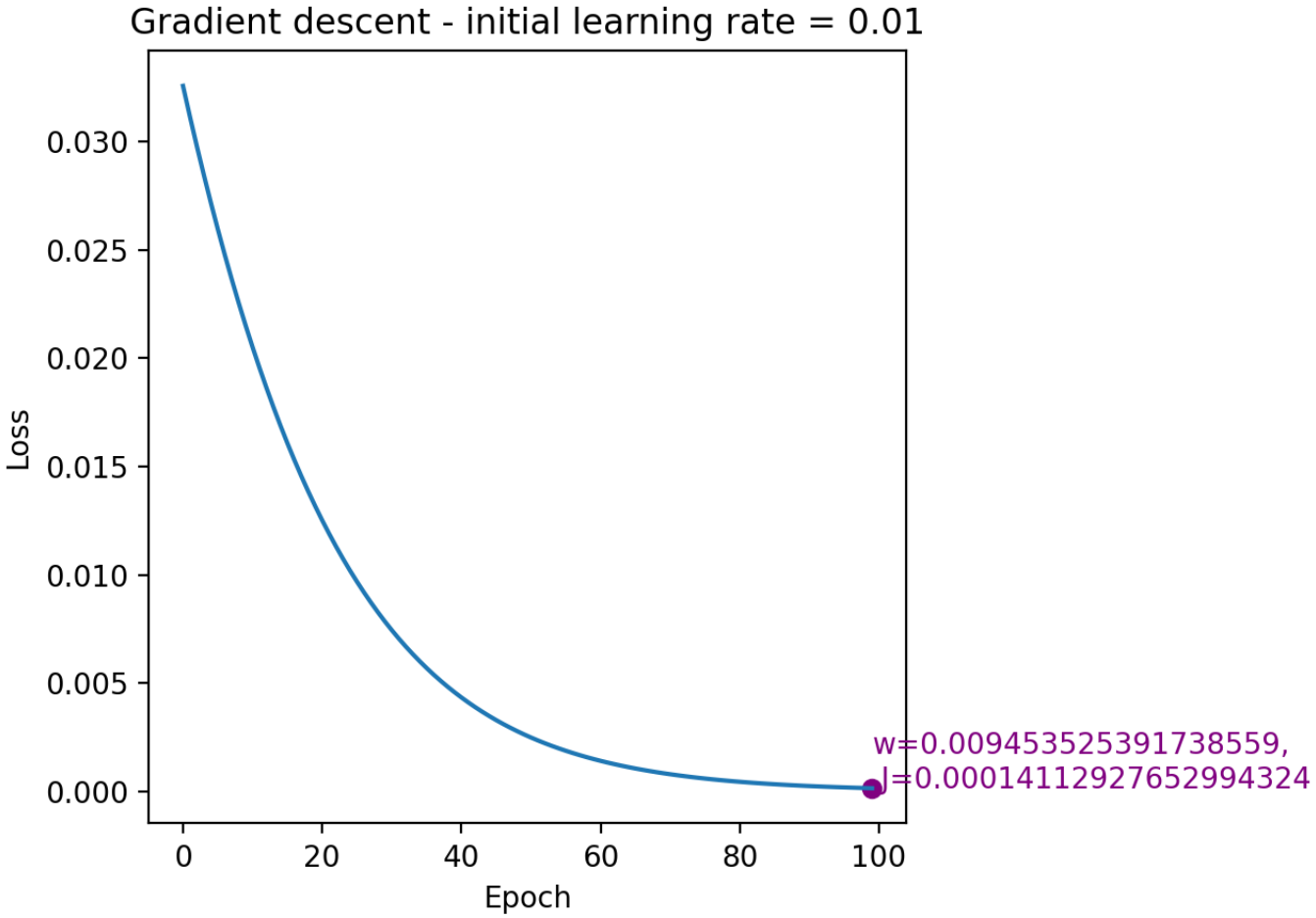
For each experiment initialize $w_1 = 0.2$ and use $x = 1$ as your only data point and once again run each experiment for 100 epochs.
And once again, create plots of *epoch* vs J for each experiment and superimpose the final values of w_1 and J.

NOTE: Due to the potential of overflow, you likely will want to have the evaluation of your J function in a try/except block where you break out of the gradient decent loop if an exception happens. The learning rates for the experiments are:

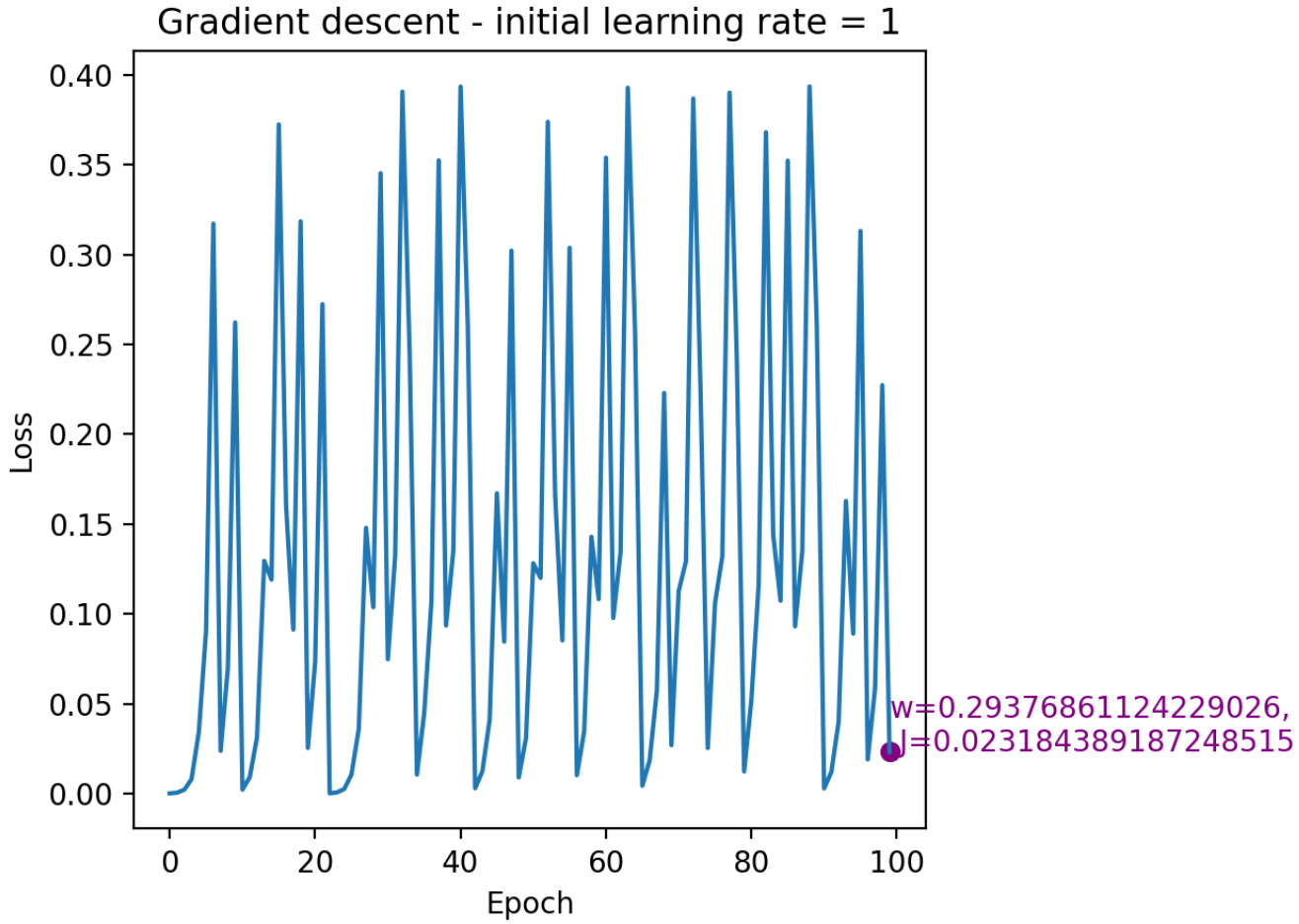
- $\eta = 0.001$



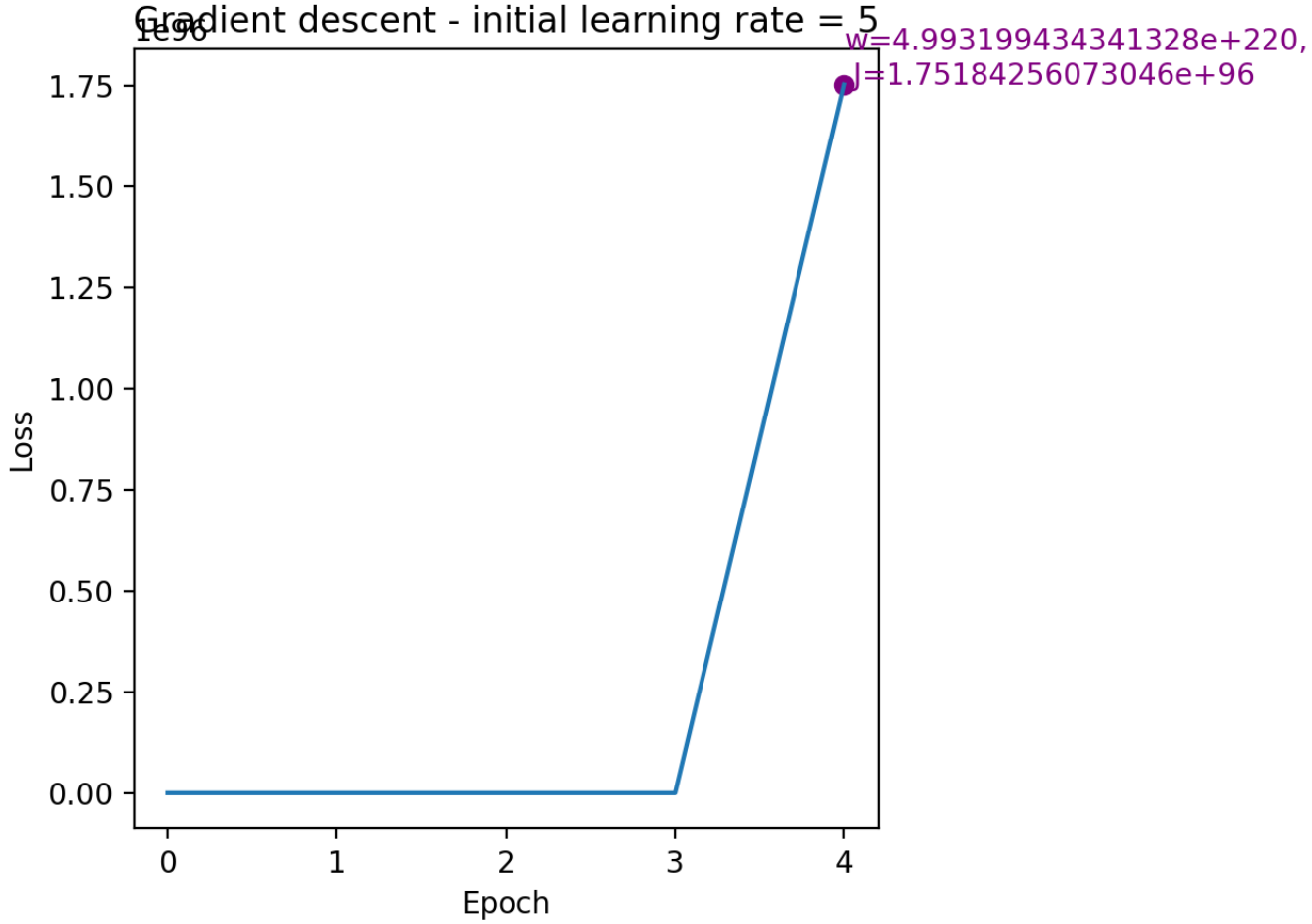
- $\eta = 0.01$


Gradient descent - initial learning rate = 0.01
Loss
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
Epoch
w=0.009453525391738559,
J=0.00014112927652994324

- $\eta = 1.0$


Gradient descent - initial learning rate = 1
Loss
Epoch
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100
w=0.29376861124229026,
J=0.023184389187248515

- $\eta = 5.0$


Gradient descent - initial learning rate = 5
1e96
w=4.993199434341328e+220,
J=1.75184256073046e+96
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Loss
0
1
2
3
4
Epoch

5 Adaptive Learning Rate

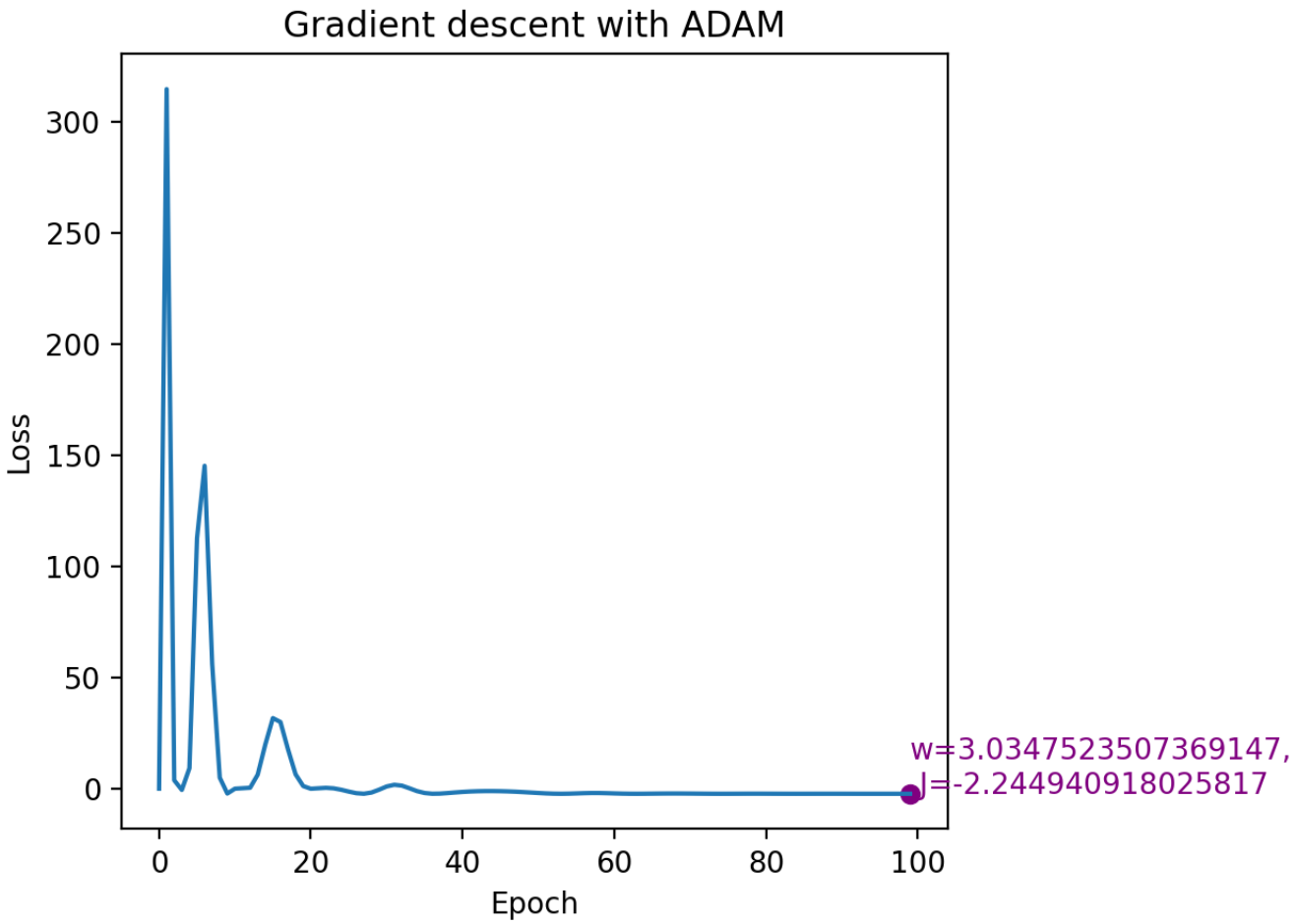
Finally let's look at using an adaptive learning rate, á la the Adam algorithm.

For this part of your homework assignment we'll once again look to learn the w_1 that minimizes $J = \frac{1}{4}(x_1 w_1)^4 - \frac{4}{3}(x_1 w_1)^3 + \frac{3}{2}(x_1 w_1)^2$ given the data point $x = 1$. Run gradient descent *with ADAM* adaptive learning on this objective function for 100 epochs and produce a graph of epoch vs J. Ultimately, you are implementing ADAM from scratch here.

Your hyperparameter initializations are:

- $w_1 = 0.2$
- $\eta = 5$
- $\rho_1 = 0.9$
- $\rho_2 = 0.999$
- $\delta = 10^{-8}$

In your report provide a plot of epoch vs J.



6 Multi–Class Classification

Finally, in preparation for our next assignment, let's do multi–class classification. For this we'll use the architecture:

$$Input \rightarrow \text{Fully Connected} \rightarrow \text{Softmax} \rightarrow \text{Output w/ Cross–Entropy Objective Function}$$

Download the MNIST dataset from BBlearn and read in the training data. Train your system using the training data, keeping track of the value of your objective function with regards to the training set as you go. In addition, we'll compute the cross entropy loss for the validation as well, the watch for overfitting.
Here's some additional implementation details/specifications:

Implementation Details

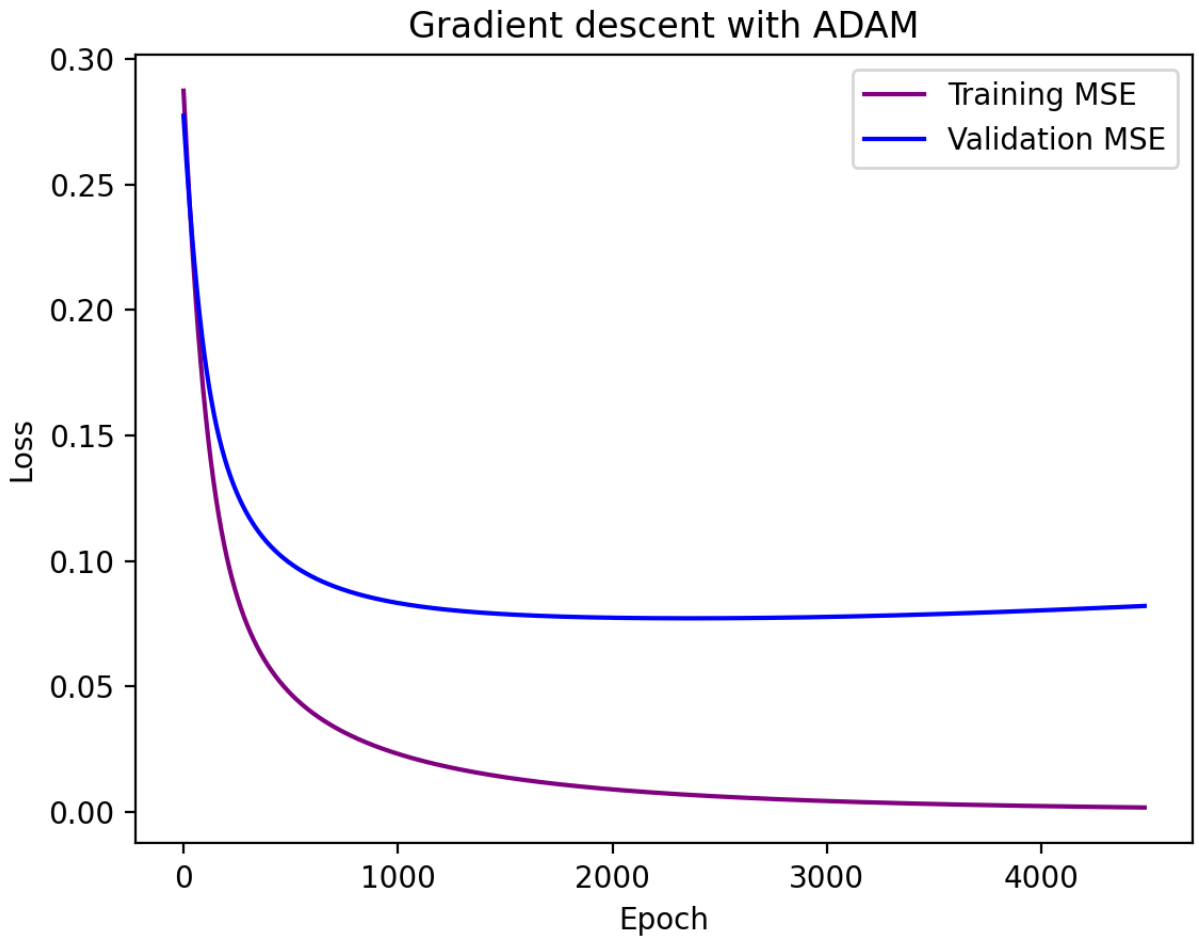
- Use *Xavier Initialzation* to initialize your weights and biases.
- Use *ADAM* learning.
- Run your iterations until near-convergence appears (things are mostly flattening out).
- You can decide on your own about things like hyperparameters, batch sizes, z-scoring, etc.. Just report those design decisions in your report and state *why* you made them.

In your final report provide:

- A graph of epoch vs. J for the training data and the validation data. Both plots should be on the same graph with legends indicating which is which.
- Your final training and validatino *accuracy*. Make sure predict the enumarated class using the *argmax* of the output as well as the original target enumerated classes.
- Your hyperparameter design decisions and why you made them.

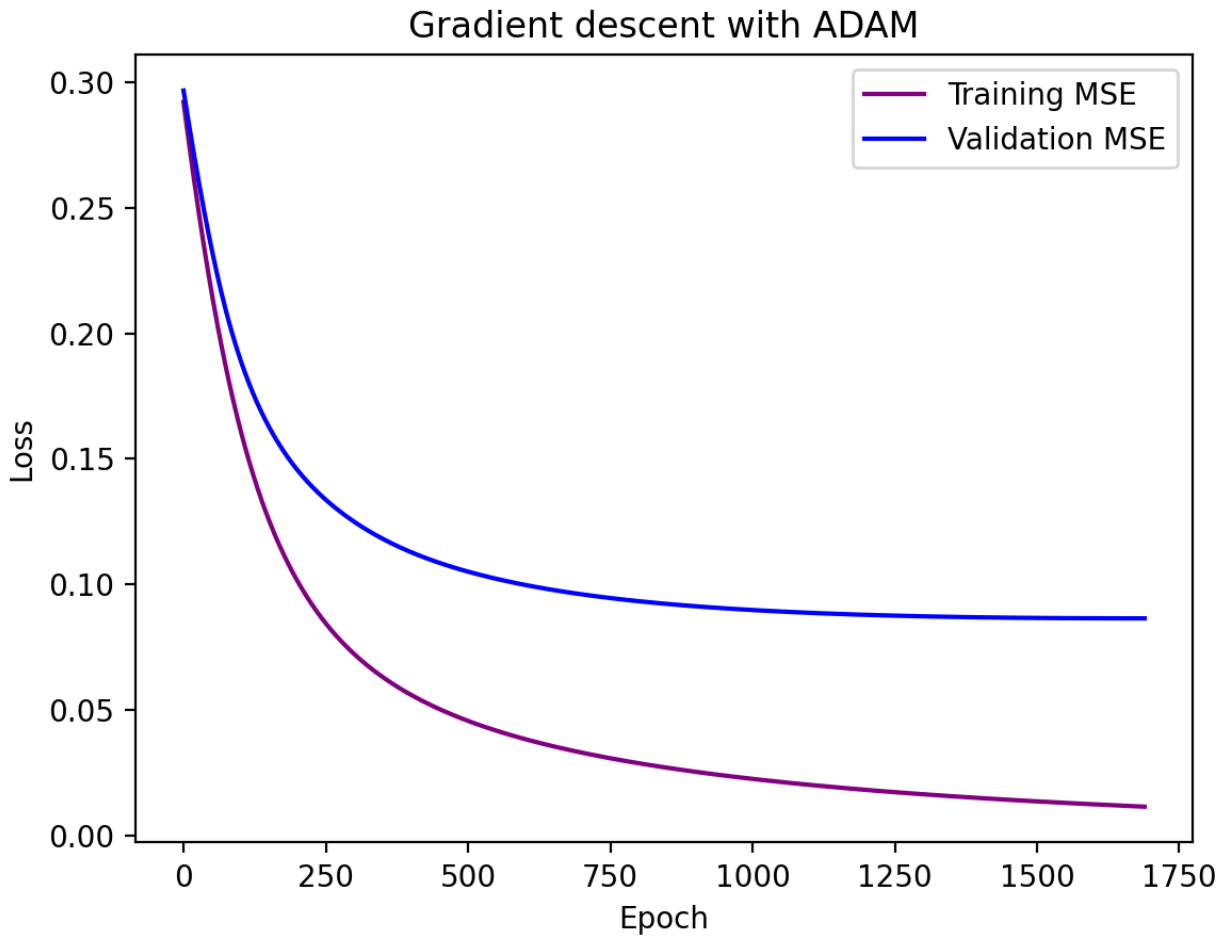
I used ADAM with $p1 = 0.9, p2 = 0.999$. To find the hyper-parameters, I used manual tuning. First, I tried with no Z-scoring, learning rate of $10^{-}4$, 5000 epochs, and added the termination of the learning process when the absolute change in the mean squared error on the training data is less than $10^{-}6$ or you pass $5,000$ epochs. I found out that not using the z-scoring will led to the same loss for each epoch and therefore out model will not be trained.

Thus, I added z-scoring and used the same numbers for the hyper-parameters. This is the graph of epoch vs. J :



And I got the Train accuracy of 1.0 and Validation accuracy of 0.83. As it is obvious our model is over-fitting.

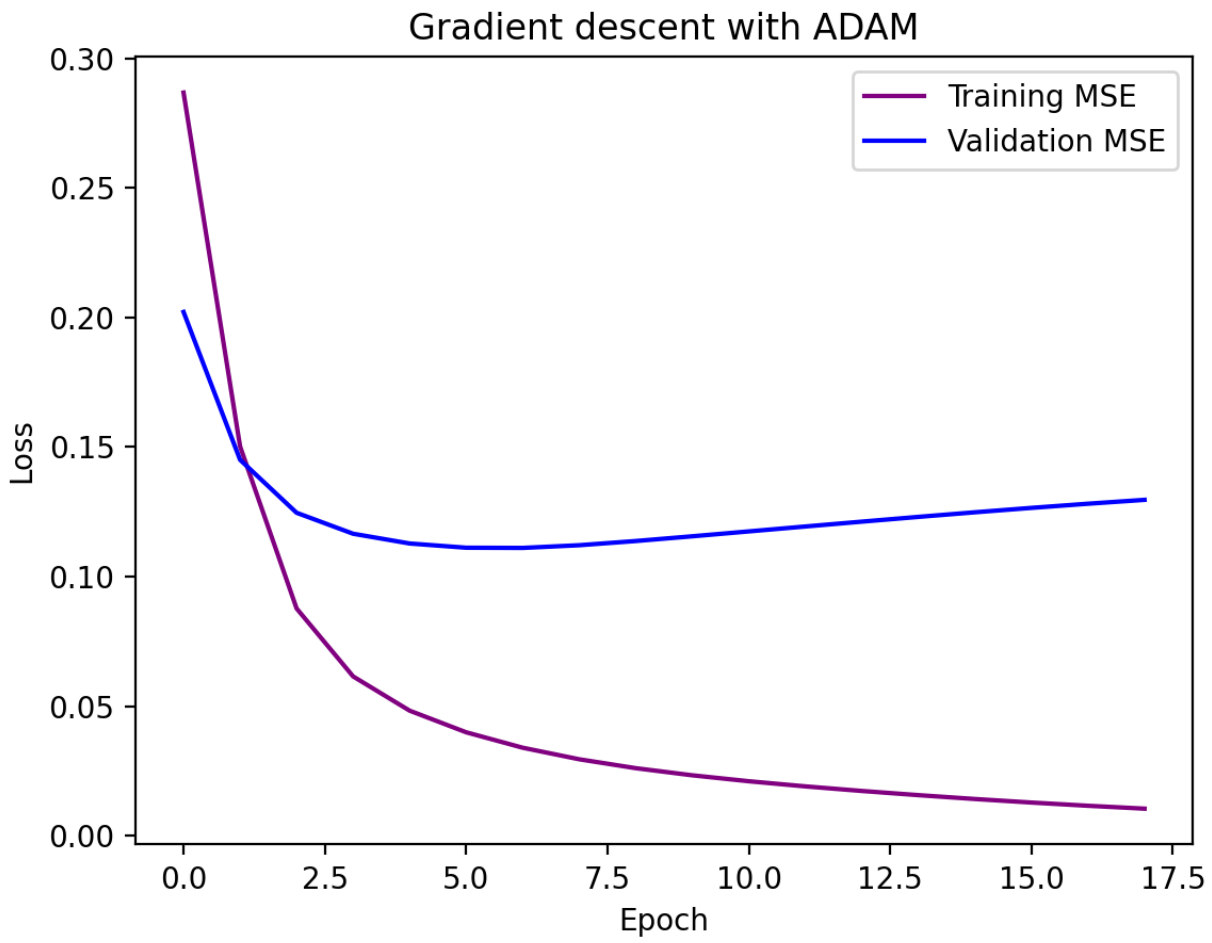
I added Stochastic Mini-Batch Gradient Descent to the implementation, and I got under-fitting for batches of 32, 64, and 128. So I decided to add early stopping instead of Stochastic Mini-Batch Gradient Descent. For early stopping we need to identify the number of epochs to wait for an improvement in validation loss before stopping the training. So, I tried 5 epochs and this is the result:



And I got the Train accuracy of 0.99 and Validation accuracy of 0.83. As it is obvious our model is still over-fitting.

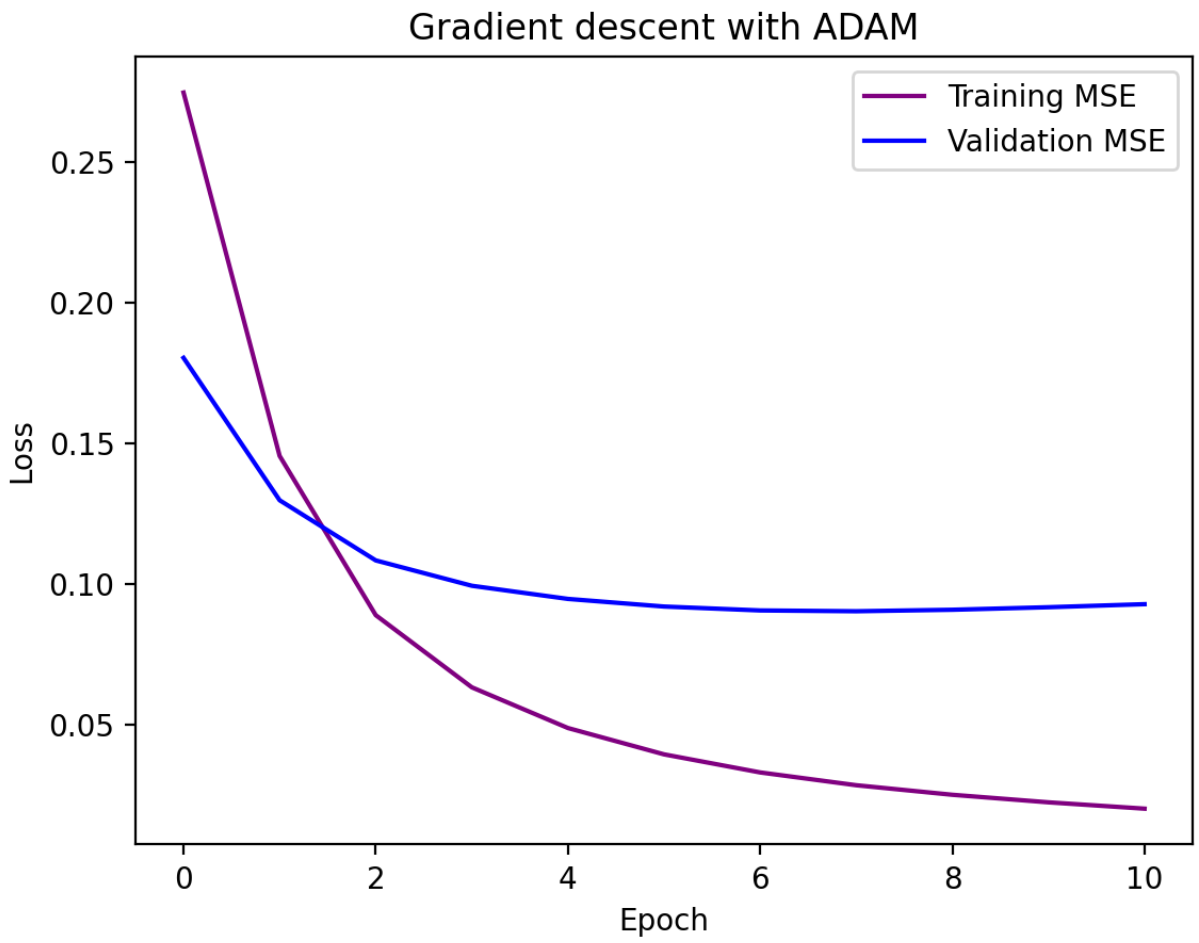
I repeated the same thing with the learning rate of 10^{-3} and got the same result.

At the end, I tried learning rate of 10^{-2} and got the following result:



And I got the Train accuracy of 0.96 and Validation accuracy of 0.76, which is good enough, but we can see the early stopping parameter is not good enough as we can see the validation loss is getting higher after epoch 10. So, I changed the number of epochs to wait for an improvement in validation loss before stopping the training to 2.

And you can see the final result:



And I got the Train accuracy of 0.94 and Validation accuracy of 0.79.