CS599 Lab 1 Report

Nicole Sylvester

February 2025

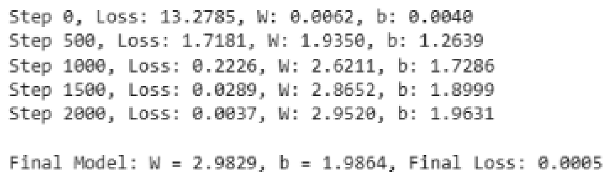
GitHub: https://github.com/nicolesylvester/CS599Lab1

1 Problem 1: Linear Regression (1 + 1 + 1 = 3 points)

- **Fork repo `https://github.com/AnkurMali/IST597_Fall2019_TF2.0` and modify the file `lin_reg.py`.**
- **Change the loss function. Which loss function works better and why? Write the mathematical formulation for each loss function.**

  MSE: Run with no noise

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2$$



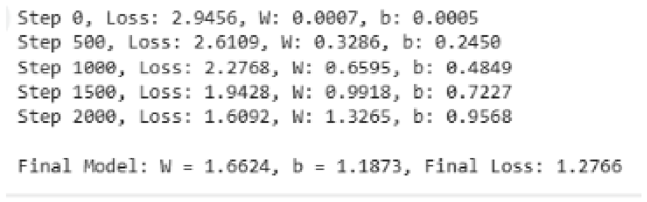
```
Step 0, Loss: 13.2785, W: 0.0062, b: 0.0040
Step 500, Loss: 1.7181, W: 1.9350, b: 1.2639
Step 1000, Loss: 0.2226, W: 2.6211, b: 1.7286
Step 1500, Loss: 0.0289, W: 2.8652, b: 1.8999
Step 2000, Loss: 0.0037, W: 2.9520, b: 1.9631

Final Model: W = 2.9829, b = 1.9864, Final Loss: 0.0005
```

Figure 1: MSE

MAE: Run with no noise

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y_i}|$$

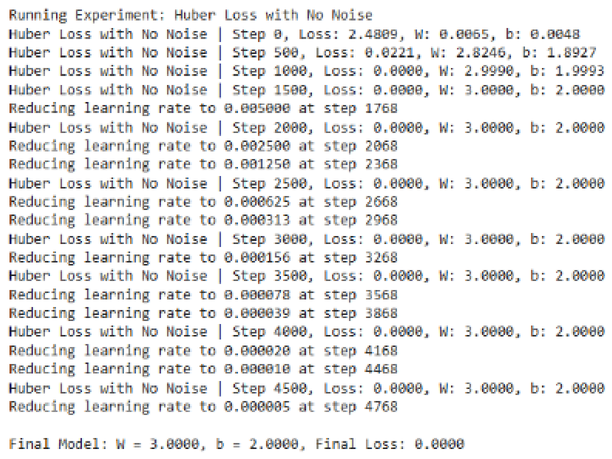


```
Step 0, Loss: 2.9456, W: 0.0007, b: 0.0005
Step 500, Loss: 2.6109, W: 0.3286, b: 0.2450
Step 1000, Loss: 2.2768, W: 0.6595, b: 0.4849
Step 1500, Loss: 1.9428, W: 0.9918, b: 0.7227
Step 2000, Loss: 1.6092, W: 1.3265, b: 0.9568

Final Model: W = 1.6624, b = 1.1873, Final Loss: 1.2766
```

Figure 2: MAE

Huber: Run with no noise



```
Running Experiment: Huber Loss with No Noise
Huber Loss with No Noise | Step 0, Loss: 2.4809, W: 0.0065, b: 0.0048
Huber Loss with No Noise | Step 500, Loss: 0.0221, W: 2.8246, b: 1.8927
Huber Loss with No Noise | Step 1000, Loss: 0.0000, W: 2.9990, b: 1.9993
Huber Loss with No Noise | Step 1500, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.005000 at step 1768
Huber Loss with No Noise | Step 2000, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.002500 at step 2068
Reducing learning rate to 0.001250 at step 2368
Huber Loss with No Noise | Step 2500, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.000625 at step 2668
Reducing learning rate to 0.000313 at step 2968
Huber Loss with No Noise | Step 3000, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.000156 at step 3268
Huber Loss with No Noise | Step 3500, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.000078 at step 3568
Reducing learning rate to 0.000039 at step 3868
Huber Loss with No Noise | Step 4000, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.000020 at step 4168
Reducing learning rate to 0.000010 at step 4468
Huber Loss with No Noise | Step 4500, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.000005 at step 4768

Final Model: W = 3.0000, b = 2.0000, Final Loss: 0.0000
```

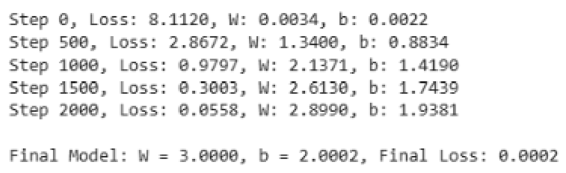
Figure 3: Huber

The MSE loss function works better The weights W and bias b values for MSE are closer to the expected values of 3 and 2, compared to the MAE results. Additionally, the final loss for MSE (0.0005) is lower than MAE (1.2766), indicating a better fit.

MSE penalizes larger errors more because errors are squared. If the data has few outliers or the errors are small, MSE can work better.

- **Create a hybrid loss function (For example, $L_1 + L_2$).**

Hybrid Loss function had smaller loss than L1 or L2.



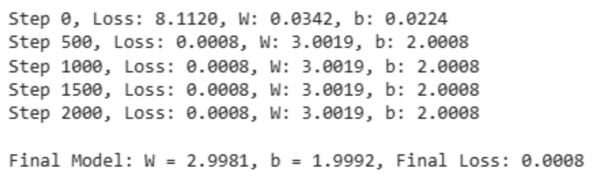
```
Step 0, Loss: 8.1120, W: 0.0034, b: 0.0022
Step 500, Loss: 2.8672, W: 1.3400, b: 0.8834
Step 1000, Loss: 0.9797, W: 2.1371, b: 1.4190
Step 1500, Loss: 0.3003, W: 2.6130, b: 1.7439
Step 2000, Loss: 0.0558, W: 2.8990, b: 1.9381

Final Model: W = 3.0000, b = 2.0002, Final Loss: 0.0002
```

Figure 4: Hybrid Loss

- **Change the learning rate.**

  Changed Learning Rate From 0.001 to 0.01

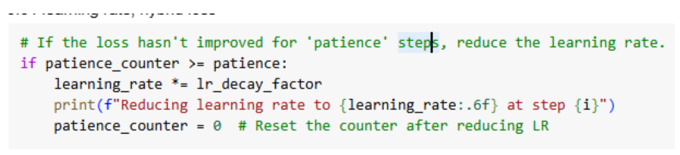


```
Step 0, Loss: 8.1120, W: 0.0342, b: 0.0224
Step 500, Loss: 0.0008, W: 3.0019, b: 2.0008
Step 1000, Loss: 0.0008, W: 3.0019, b: 2.0008
Step 1500, Loss: 0.0008, W: 3.0019, b: 2.0008
Step 2000, Loss: 0.0008, W: 3.0019, b: 2.0008

Final Model: W = 2.9981, b = 1.9992, Final Loss: 0.0008
```

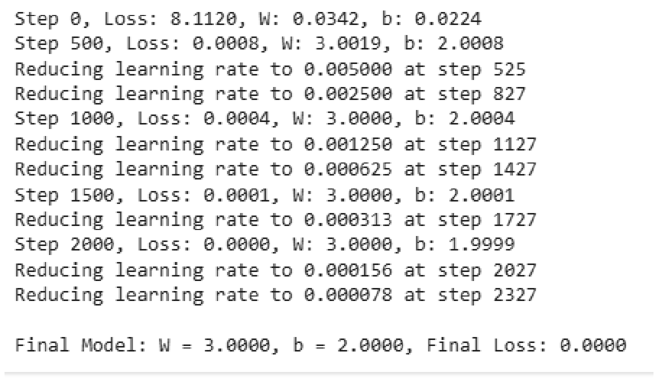
Figure 5: Learning Rate 0.01

- **Use patience scheduling (Whenever loss does not change, divide the learning rate by half).**



```
# If the loss hasn't improved for 'patience' steps, reduce the learning rate.
if patience_counter >= patience:
    learning_rate *= lr_decay_factor
    print(f"Reducing learning rate to {learning_rate:.6f} at step {i}")
    patience_counter = 0  # Reset the counter after reducing LR
```

Figure 6: Patience Scheduling



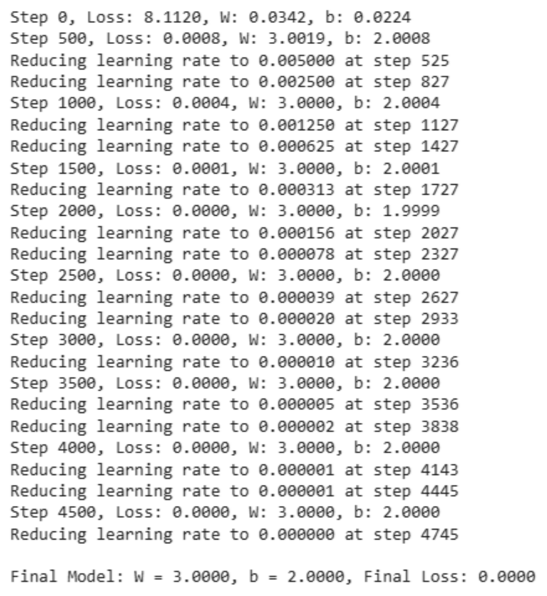
```
Step 0, Loss: 8.1120, W: 0.0342, b: 0.0224
Step 500, Loss: 0.0008, W: 3.0019, b: 2.0008
Reducing learning rate to 0.005000 at step 525
Reducing learning rate to 0.002500 at step 827
Step 1000, Loss: 0.0004, W: 3.0000, b: 2.0004
Reducing learning rate to 0.001250 at step 1127
Reducing learning rate to 0.000625 at step 1427
Step 1500, Loss: 0.0001, W: 3.0000, b: 2.0001
Reducing learning rate to 0.000313 at step 1727
Step 2000, Loss: 0.0000, W: 3.0000, b: 1.9999
Reducing learning rate to 0.000156 at step 2027
Reducing learning rate to 0.000078 at step 2327

Final Model: W = 3.0000, b = 2.0000, Final Loss: 0.0000
```

Figure 7: Patience Scheduling

Train for a longer duration. Change the initial values for W and B. What effect does this have on the end result?

Increased training steps from 2500 to 5000.

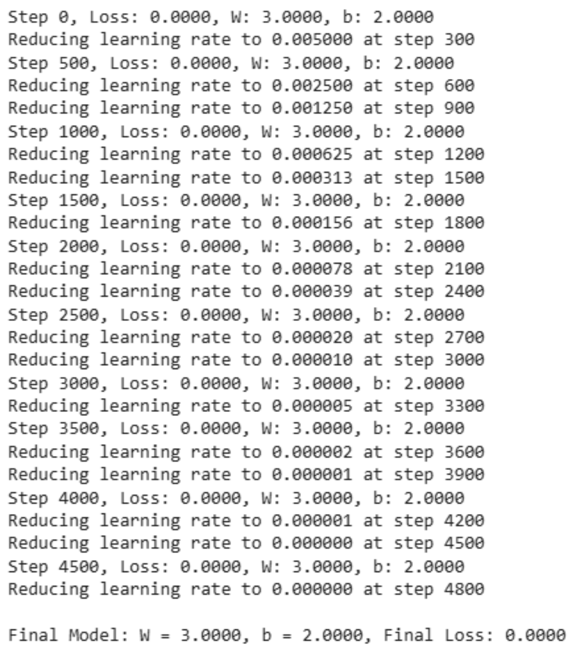


```
Step 0, Loss: 8.1120, W: 0.0342, b: 0.0224
Step 500, Loss: 0.0008, W: 3.0019, b: 2.0008
Reducing learning rate to 0.005000 at step 525
Reducing learning rate to 0.002500 at step 827
Step 1000, Loss: 0.0004, W: 3.0000, b: 2.0004
Reducing learning rate to 0.001250 at step 1127
Reducing learning rate to 0.000625 at step 1427
Step 1500, Loss: 0.0001, W: 3.0000, b: 2.0001
Reducing learning rate to 0.000313 at step 1727
Step 2000, Loss: 0.0000, W: 3.0000, b: 1.9999
Reducing learning rate to 0.000156 at step 2027
Reducing learning rate to 0.000078 at step 2327
Step 2500, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.000039 at step 2627
Reducing learning rate to 0.000020 at step 2933
Step 3000, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.000010 at step 3236
Step 3500, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.000005 at step 3536
Reducing learning rate to 0.000002 at step 3838
Step 4000, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.000001 at step 4143
Reducing learning rate to 0.000001 at step 4445
Step 4500, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.000000 at step 4745

Final Model: W = 3.0000, b = 2.0000, Final Loss: 0.0000
```

Figure 8: 5000 steps

Changed initial values from 0 to 3 and 2



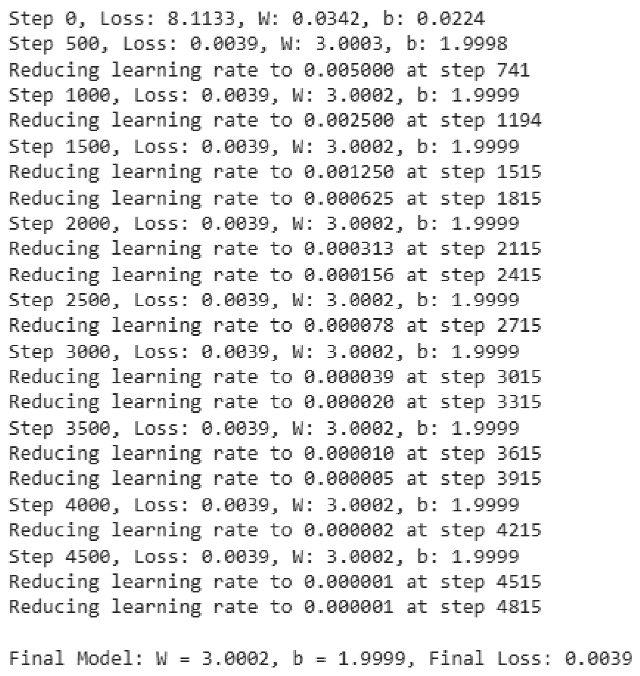
```
Step 0, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.005000 at step 300
Step 500, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.002500 at step 600
Reducing learning rate to 0.001250 at step 900
Step 1000, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.000625 at step 1200
Reducing learning rate to 0.000313 at step 1500
Step 1500, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.000156 at step 1800
Step 2000, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.000078 at step 2100
Reducing learning rate to 0.000039 at step 2400
Step 2500, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.000020 at step 2700
Reducing learning rate to 0.000010 at step 3000
Step 3000, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.000005 at step 3300
Step 3500, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.000002 at step 3600
Reducing learning rate to 0.000001 at step 3900
Step 4000, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.000001 at step 4200
Reducing learning rate to 0.000000 at step 4500
Step 4500, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.000000 at step 4800

Final Model: W = 3.0000, b = 2.0000, Final Loss: 0.0000
```

Figure 9: Initial W and b values

- **Change the level of noise.**

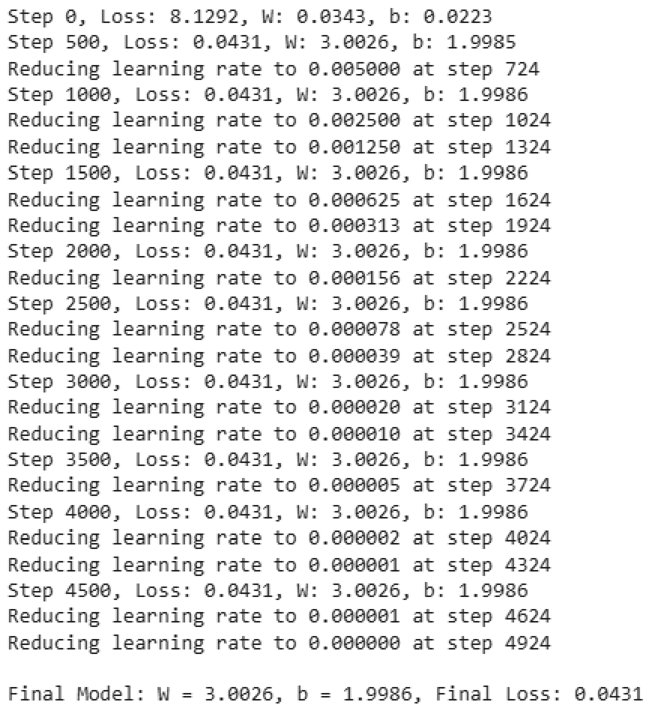
All noise has been added to input, W and b are initialized to 0



```
Step 0, Loss: 8.1133, W: 0.0342, b: 0.0224
Step 500, Loss: 0.0039, W: 3.0003, b: 1.9998
Reducing learning rate to 0.005000 at step 741
Step 1000, Loss: 0.0039, W: 3.0002, b: 1.9999
Reducing learning rate to 0.002500 at step 1194
Step 1500, Loss: 0.0039, W: 3.0002, b: 1.9999
Reducing learning rate to 0.001250 at step 1515
Reducing learning rate to 0.000625 at step 1815
Step 2000, Loss: 0.0039, W: 3.0002, b: 1.9999
Reducing learning rate to 0.000313 at step 2115
Reducing learning rate to 0.000156 at step 2415
Step 2500, Loss: 0.0039, W: 3.0002, b: 1.9999
Reducing learning rate to 0.000078 at step 2715
Step 3000, Loss: 0.0039, W: 3.0002, b: 1.9999
Reducing learning rate to 0.000039 at step 3015
Reducing learning rate to 0.000020 at step 3315
Step 3500, Loss: 0.0039, W: 3.0002, b: 1.9999
Reducing learning rate to 0.000010 at step 3615
Reducing learning rate to 0.000005 at step 3915
Step 4000, Loss: 0.0039, W: 3.0002, b: 1.9999
Reducing learning rate to 0.000002 at step 4215
Step 4500, Loss: 0.0039, W: 3.0002, b: 1.9999
Reducing learning rate to 0.000001 at step 4515
Reducing learning rate to 0.000001 at step 4815

Final Model: W = 3.0002, b = 1.9999, Final Loss: 0.0039
```

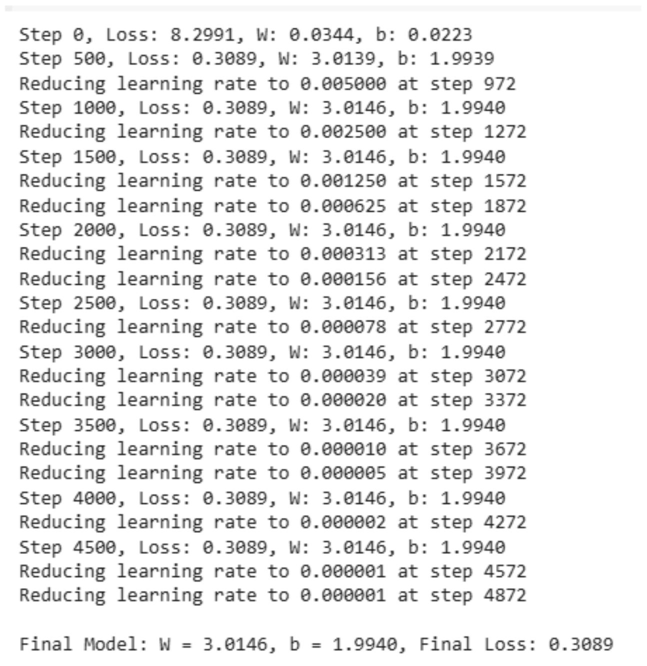
Figure 10: Gaussian 0.01



```
Step 0, Loss: 8.1292, W: 0.0343, b: 0.0223
Step 500, Loss: 0.0431, W: 3.0026, b: 1.9985
Reducing learning rate to 0.005000 at step 724
Step 1000, Loss: 0.0431, W: 3.0026, b: 1.9986
Reducing learning rate to 0.002500 at step 1024
Reducing learning rate to 0.001250 at step 1324
Step 1500, Loss: 0.0431, W: 3.0026, b: 1.9986
Reducing learning rate to 0.000625 at step 1624
Reducing learning rate to 0.000313 at step 1924
Step 2000, Loss: 0.0431, W: 3.0026, b: 1.9986
Reducing learning rate to 0.000156 at step 2224
Step 2500, Loss: 0.0431, W: 3.0026, b: 1.9986
Reducing learning rate to 0.000078 at step 2524
Reducing learning rate to 0.000039 at step 2824
Step 3000, Loss: 0.0431, W: 3.0026, b: 1.9986
Reducing learning rate to 0.000020 at step 3124
Reducing learning rate to 0.000010 at step 3424
Step 3500, Loss: 0.0431, W: 3.0026, b: 1.9986
Reducing learning rate to 0.000005 at step 3724
Step 4000, Loss: 0.0431, W: 3.0026, b: 1.9986
Reducing learning rate to 0.000002 at step 4024
Reducing learning rate to 0.000001 at step 4324
Step 4500, Loss: 0.0431, W: 3.0026, b: 1.9986
Reducing learning rate to 0.000001 at step 4624
Reducing learning rate to 0.000000 at step 4924

Final Model: W = 3.0026, b = 1.9986, Final Loss: 0.0431
```

Figure 11: Gaussian 0.1



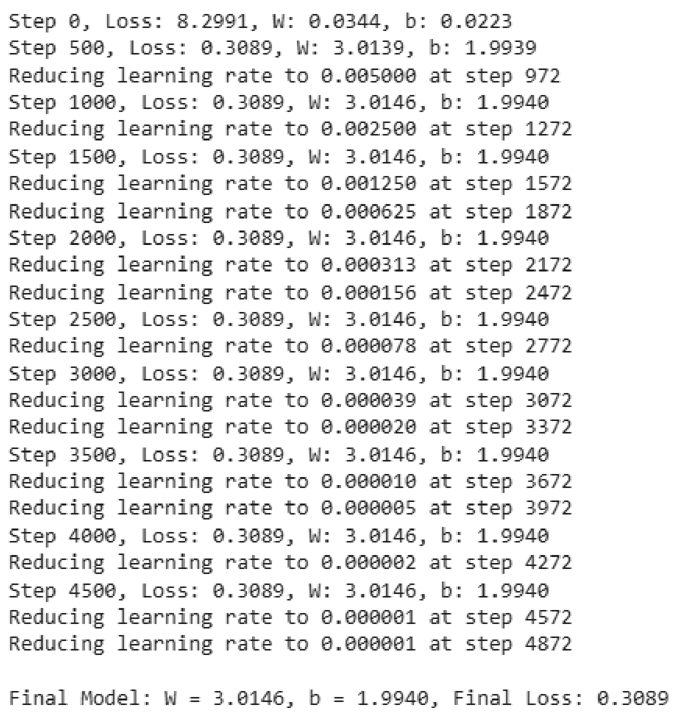
```
Step 0, Loss: 8.2991, W: 0.0344, b: 0.0223
Step 500, Loss: 0.3089, W: 3.0139, b: 1.9939
Reducing learning rate to 0.005000 at step 972
Step 1000, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.002500 at step 1272
Step 1500, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.001250 at step 1572
Reducing learning rate to 0.000625 at step 1872
Step 2000, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.000313 at step 2172
Reducing learning rate to 0.000156 at step 2472
Step 2500, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.000078 at step 2772
Step 3000, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.000039 at step 3072
Reducing learning rate to 0.000020 at step 3372
Step 3500, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.000010 at step 3672
Reducing learning rate to 0.000005 at step 3972
Step 4000, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.000002 at step 4272
Step 4500, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.000001 at step 4572
Reducing learning rate to 0.000001 at step 4872

Final Model: W = 3.0146, b = 1.9940, Final Loss: 0.3089
```

Figure 12: Gaussian 0.5

- **Use various types of noise.**

All at level 0.5 Gaussian noise

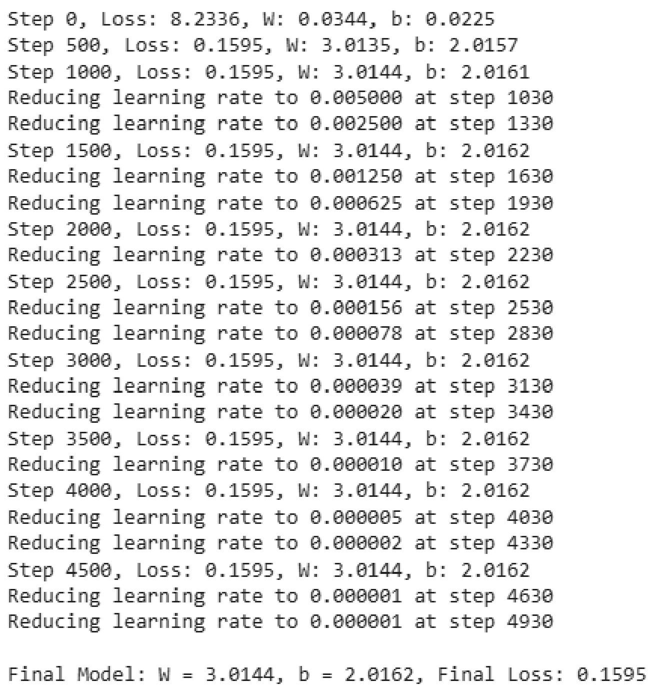


```
Step 0, Loss: 8.2991, W: 0.0344, b: 0.0223
Step 500, Loss: 0.3089, W: 3.0139, b: 1.9939
Reducing learning rate to 0.005000 at step 972
Step 1000, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.002500 at step 1272
Step 1500, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.001250 at step 1572
Reducing learning rate to 0.000625 at step 1872
Step 2000, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.000313 at step 2172
Reducing learning rate to 0.000156 at step 2472
Step 2500, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.000078 at step 2772
Step 3000, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.000039 at step 3072
Reducing learning rate to 0.000020 at step 3372
Step 3500, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.000010 at step 3672
Reducing learning rate to 0.000005 at step 3972
Step 4000, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.000002 at step 4272
Step 4500, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.000001 at step 4572
Reducing learning rate to 0.000001 at step 4872

Final Model: W = 3.0146, b = 1.9940, Final Loss: 0.3089
```

Figure 13: Gaussian

Uniform Noise

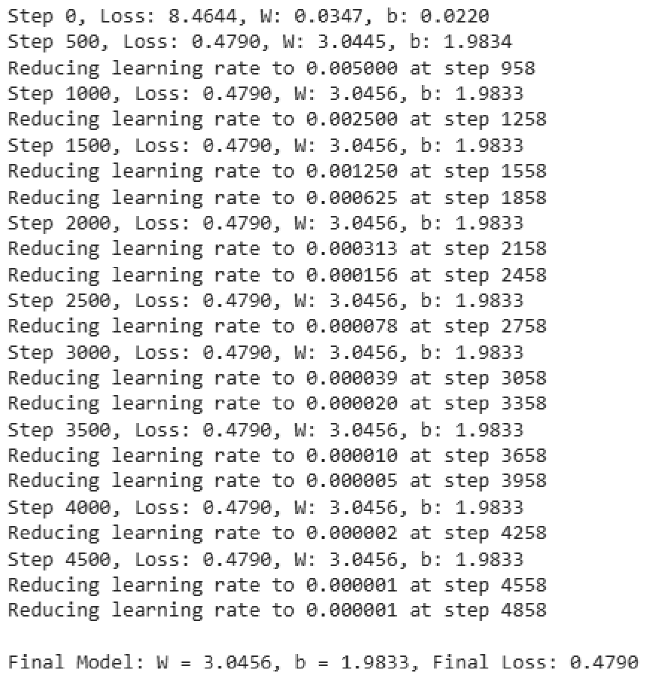


```
Step 0, Loss: 8.2336, W: 0.0344, b: 0.0225
Step 500, Loss: 0.1595, W: 3.0135, b: 2.0157
Step 1000, Loss: 0.1595, W: 3.0144, b: 2.0161
Reducing learning rate to 0.005000 at step 1030
Reducing learning rate to 0.002500 at step 1330
Step 1500, Loss: 0.1595, W: 3.0144, b: 2.0162
Reducing learning rate to 0.001250 at step 1630
Reducing learning rate to 0.000625 at step 1930
Step 2000, Loss: 0.1595, W: 3.0144, b: 2.0162
Reducing learning rate to 0.000313 at step 2230
Step 2500, Loss: 0.1595, W: 3.0144, b: 2.0162
Reducing learning rate to 0.000156 at step 2530
Reducing learning rate to 0.000078 at step 2830
Step 3000, Loss: 0.1595, W: 3.0144, b: 2.0162
Reducing learning rate to 0.000039 at step 3130
Reducing learning rate to 0.000020 at step 3430
Step 3500, Loss: 0.1595, W: 3.0144, b: 2.0162
Reducing learning rate to 0.000010 at step 3730
Step 4000, Loss: 0.1595, W: 3.0144, b: 2.0162
Reducing learning rate to 0.000005 at step 4030
Reducing learning rate to 0.000002 at step 4330
Step 4500, Loss: 0.1595, W: 3.0144, b: 2.0162
Reducing learning rate to 0.000001 at step 4630
Reducing learning rate to 0.000001 at step 4930

Final Model: W = 3.0144, b = 2.0162, Final Loss: 0.1595
```

Figure 14: Uniform

Laplacian Noise



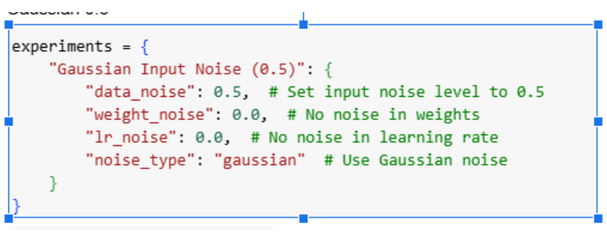
```
Step 0, Loss: 8.4644, W: 0.0347, b: 0.0220
Step 500, Loss: 0.4790, W: 3.0445, b: 1.9834
Reducing learning rate to 0.005000 at step 958
Step 1000, Loss: 0.4790, W: 3.0456, b: 1.9833
Reducing learning rate to 0.002500 at step 1258
Step 1500, Loss: 0.4790, W: 3.0456, b: 1.9833
Reducing learning rate to 0.001250 at step 1558
Reducing learning rate to 0.000625 at step 1858
Step 2000, Loss: 0.4790, W: 3.0456, b: 1.9833
Reducing learning rate to 0.000313 at step 2158
Reducing learning rate to 0.000156 at step 2458
Step 2500, Loss: 0.4790, W: 3.0456, b: 1.9833
Reducing learning rate to 0.000078 at step 2758
Step 3000, Loss: 0.4790, W: 3.0456, b: 1.9833
Reducing learning rate to 0.000039 at step 3058
Reducing learning rate to 0.000020 at step 3358
Step 3500, Loss: 0.4790, W: 3.0456, b: 1.9833
Reducing learning rate to 0.000010 at step 3658
Reducing learning rate to 0.000005 at step 3958
Step 4000, Loss: 0.4790, W: 3.0456, b: 1.9833
Reducing learning rate to 0.000002 at step 4258
Step 4500, Loss: 0.4790, W: 3.0456, b: 1.9833
Reducing learning rate to 0.000001 at step 4558
Reducing learning rate to 0.000001 at step 4858

Final Model: W = 3.0456, b = 1.9833, Final Loss: 0.4790
```

Figure 15: Laplacian

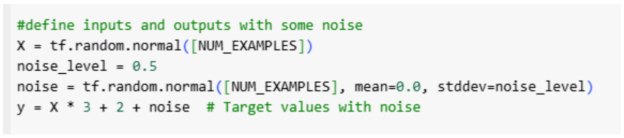
- **Add noise in data.**

The previous examples have all added noise to data. Gaussian 0.5



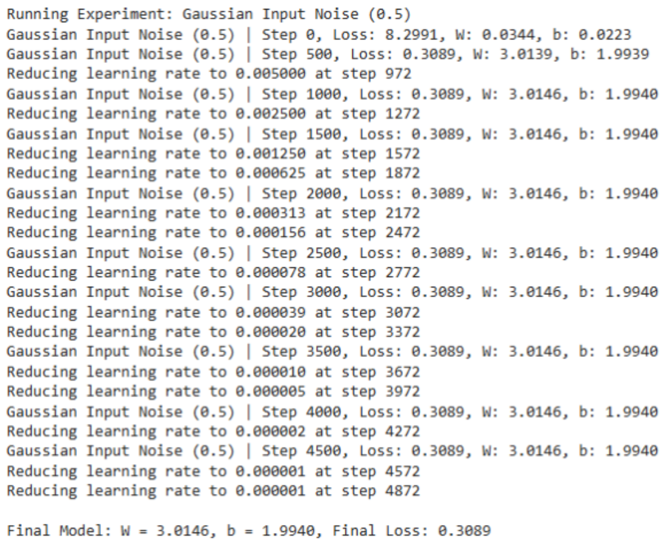
```
experiments = {
    "Gaussian Input Noise (0.5)": {
        "data_noise": 0.5,  # Set input noise level to 0.5
        "weight_noise": 0.0,  # No noise in weights
        "lr_noise": 0.0,  # No noise in learning rate
        "noise_type": "gaussian"  # Use Gaussian noise
    }
}
```

Figure 16: Input noise code



```
#define inputs and outputs with some noise
X = tf.random.normal([NUM_EXAMPLES])
noise_level = 0.5
noise = tf.random.normal([NUM_EXAMPLES], mean=0.0, stddev=noise_level)
y = X * 3 + 2 + noise  # Target values with noise
```

Figure 17: Input noise code

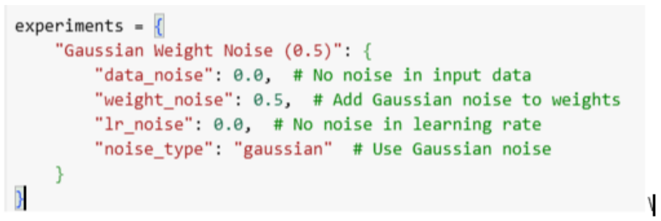


```
Running Experiment: Gaussian Input Noise (0.5)
Gaussian Input Noise (0.5) | Step 0, Loss: 8.2991, W: 0.0344, b: 0.0223
Gaussian Input Noise (0.5) | Step 500, Loss: 0.3089, W: 3.0139, b: 1.9939
Reducing learning rate to 0.005000 at step 972
Gaussian Input Noise (0.5) | Step 1000, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.002500 at step 1272
Gaussian Input Noise (0.5) | Step 1500, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.001250 at step 1572
Reducing learning rate to 0.000625 at step 1872
Gaussian Input Noise (0.5) | Step 2000, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.000313 at step 2172
Reducing learning rate to 0.000156 at step 2472
Gaussian Input Noise (0.5) | Step 2500, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.000078 at step 2772
Gaussian Input Noise (0.5) | Step 3000, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.000039 at step 3072
Reducing learning rate to 0.000020 at step 3372
Gaussian Input Noise (0.5) | Step 3500, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.000010 at step 3672
Reducing learning rate to 0.000005 at step 3972
Gaussian Input Noise (0.5) | Step 4000, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.000002 at step 4272
Gaussian Input Noise (0.5) | Step 4500, Loss: 0.3089, W: 3.0146, b: 1.9940
Reducing learning rate to 0.000001 at step 4572
Reducing learning rate to 0.000001 at step 4872

Final Model: W = 3.0146, b = 1.9940, Final Loss: 0.3089
```

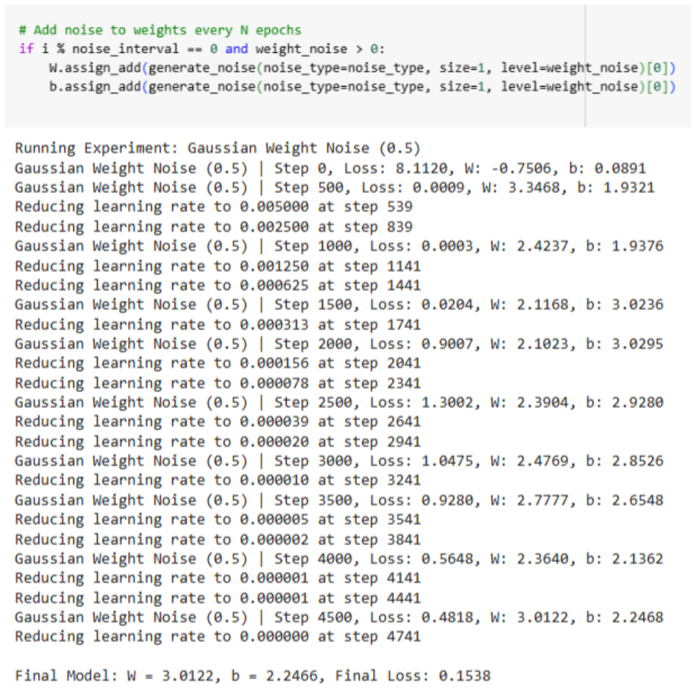
Figure 18: Run with Input Noise

- **Add noise in your weights.**



```
experiments = {
    "Gaussian Weight Noise (0.5)": {
        "data_noise": 0.0,  # No noise in input data
        "weight_noise": 0.5,  # Add Gaussian noise to weights
        "lr_noise": 0.0,  # No noise in learning rate
        "noise_type": "gaussian"  # Use Gaussian noise
    }
}
```

Figure 19: Weight Noise



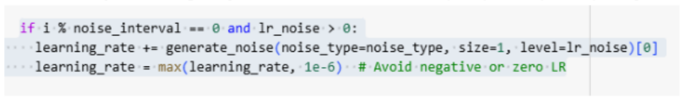
```
# Add noise to weights every N epochs
if i % noise_interval == 0 and weight_noise > 0:
    W.assign_add(generate_noise(noise_type=noise_type, size=1, level=weight_noise)[0])
    b.assign_add(generate_noise(noise_type=noise_type, size=1, level=weight_noise)[0])
```

```
Running Experiment: Gaussian Weight Noise (0.5)
Gaussian Weight Noise (0.5) | Step 0, Loss: 8.1120, W: -0.7506, b: 0.0891
Gaussian Weight Noise (0.5) | Step 500, Loss: 0.0009, W: 3.3468, b: 1.9321
Reducing learning rate to 0.005000 at step 539
Reducing learning rate to 0.002500 at step 839
Gaussian Weight Noise (0.5) | Step 1000, Loss: 0.0003, W: 2.4237, b: 1.9376
Reducing learning rate to 0.001250 at step 1141
Reducing learning rate to 0.000625 at step 1441
Gaussian Weight Noise (0.5) | Step 1500, Loss: 0.0204, W: 2.1168, b: 3.0236
Reducing learning rate to 0.000313 at step 1741
Gaussian Weight Noise (0.5) | Step 2000, Loss: 0.9007, W: 2.1023, b: 3.0295
Reducing learning rate to 0.000156 at step 2041
Reducing learning rate to 0.000078 at step 2341
Gaussian Weight Noise (0.5) | Step 2500, Loss: 1.3002, W: 2.3904, b: 2.9280
Reducing learning rate to 0.000039 at step 2641
Reducing learning rate to 0.000020 at step 2941
Gaussian Weight Noise (0.5) | Step 3000, Loss: 1.0475, W: 2.4769, b: 2.8526
Reducing learning rate to 0.000010 at step 3241
Gaussian Weight Noise (0.5) | Step 3500, Loss: 0.9280, W: 2.7777, b: 2.6548
Reducing learning rate to 0.000005 at step 3541
Reducing learning rate to 0.000002 at step 3841
Gaussian Weight Noise (0.5) | Step 4000, Loss: 0.5648, W: 2.3640, b: 2.1362
Reducing learning rate to 0.000001 at step 4141
Reducing learning rate to 0.000001 at step 4441
Gaussian Weight Noise (0.5) | Step 4500, Loss: 0.4818, W: 3.0122, b: 2.2468
Reducing learning rate to 0.000000 at step 4741

Final Model: W = 3.0122, b = 2.2466, Final Loss: 0.1538
```

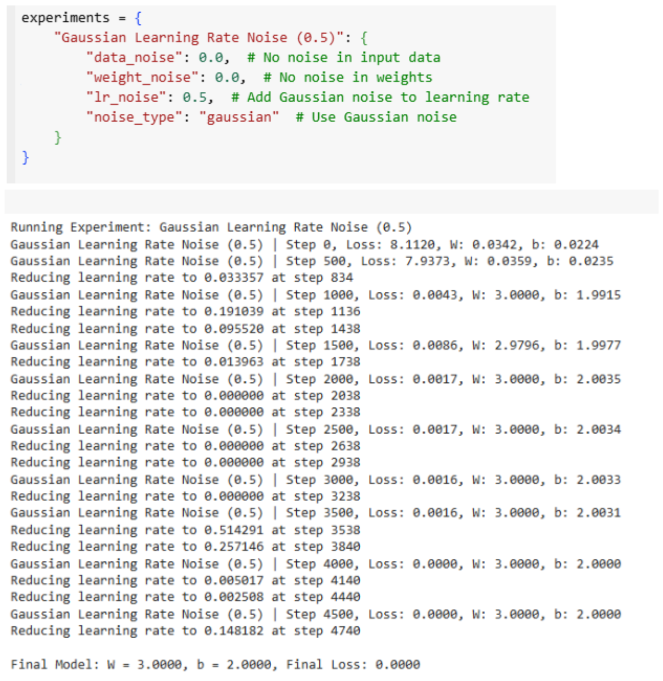
Figure 20: Weight Noise

- **Add noise in your learning rate. For all of the above, the scheme can be per epoch or per N epochs.**



```
if i % noise_interval == 0 and lr_noise > 0:
    learning_rate += generate_noise(noise_type=noise_type, size=1, level=lr_noise)[0]
    learning_rate = max(learning_rate, 1e-6)  # Avoid negative or zero LR
```

Figure 21: Learning Rate Noise



```
experiments = {
    "Gaussian Learning Rate Noise (0.5)": {
        "data_noise": 0.0,  # No noise in input data
        "weight_noise": 0.0,  # No noise in weights
        "lr_noise": 0.5,  # Add Gaussian noise to learning rate
        "noise_type": "gaussian"  # Use Gaussian noise
    }
}
```

```
Running Experiment: Gaussian Learning Rate Noise (0.5)
Gaussian Learning Rate Noise (0.5) | Step 0, Loss: 8.1120, W: 0.0342, b: 0.0224
Gaussian Learning Rate Noise (0.5) | Step 500, Loss: 7.9373, W: 0.0359, b: 0.0235
Reducing learning rate to 0.033357 at step 834
Gaussian Learning Rate Noise (0.5) | Step 1000, Loss: 0.0043, W: 3.0000, b: 1.9915
Reducing learning rate to 0.191039 at step 1136
Reducing learning rate to 0.095520 at step 1438
Gaussian Learning Rate Noise (0.5) | Step 1500, Loss: 0.0086, W: 2.9796, b: 1.9977
Reducing learning rate to 0.013963 at step 1738
Gaussian Learning Rate Noise (0.5) | Step 2000, Loss: 0.0017, W: 3.0000, b: 2.0035
Reducing learning rate to 0.000000 at step 2038
Reducing learning rate to 0.000000 at step 2338
Gaussian Learning Rate Noise (0.5) | Step 2500, Loss: 0.0017, W: 3.0000, b: 2.0034
Reducing learning rate to 0.000000 at step 2638
Reducing learning rate to 0.000000 at step 2938
Gaussian Learning Rate Noise (0.5) | Step 3000, Loss: 0.0016, W: 3.0000, b: 2.0033
Reducing learning rate to 0.000000 at step 3238
Gaussian Learning Rate Noise (0.5) | Step 3500, Loss: 0.0016, W: 3.0000, b: 2.0031
Reducing learning rate to 0.514291 at step 3538
Reducing learning rate to 0.257146 at step 3840
Gaussian Learning Rate Noise (0.5) | Step 4000, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.005017 at step 4140
Reducing learning rate to 0.002508 at step 4440
Gaussian Learning Rate Noise (0.5) | Step 4500, Loss: 0.0000, W: 3.0000, b: 2.0000
Reducing learning rate to 0.148182 at step 4740

Final Model: W = 3.0000, b = 2.0000, Final Loss: 0.0000
```

Figure 22: Learning Rate Noise

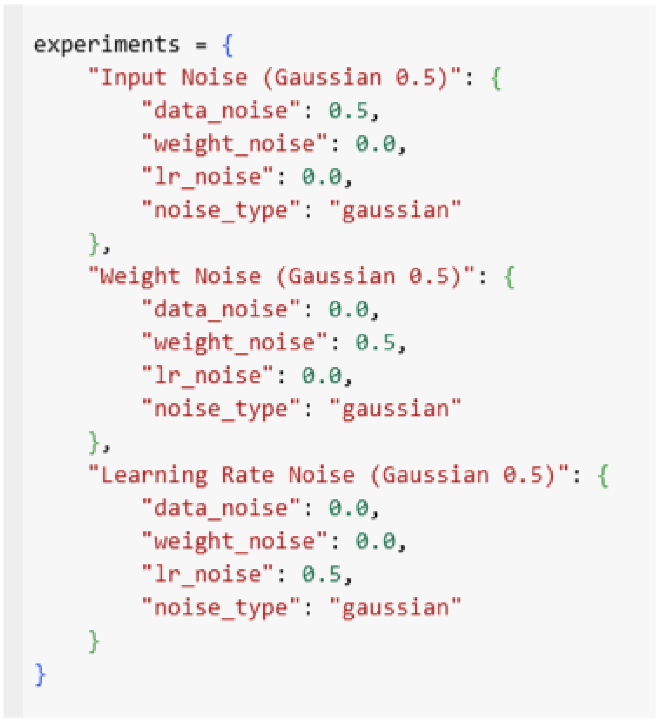
- **How do these changes affect the performance?**

Adding noise to inputs, weights, and the learning rate can impact a model's performance, influencing convergence, stability, and generalization. Input noise introduces variability in the data, which can enhance robustness and generalization but may lead to underfitting. Weight noise adds randomness to model parameters, helping the model escape local minima but can cause instability. Learning rate noise modifies the step size during gradient descent, allowing learning speeds that accelerate convergence or introduce instability. While low levels of noise can improve generalization and preventing overfitting, high levels of noise might destabilize training and affect performance.

- **Do you think these changes will have the same effect (if any) on other classification problems and mathematical models?**

The effects of adding noise to inputs, weights, and the learning rate vary across different classification problems and mathematical models. In simpler models like linear regression, noise can introduce variability but still allow for effective learning if the noise level is moderate. However, in more complex models like deep neural networks, input noise can enhance generalization, while weight and learning rate noise help avoid overfitting. For classification tasks, noise can help models generalize better to unseen data.

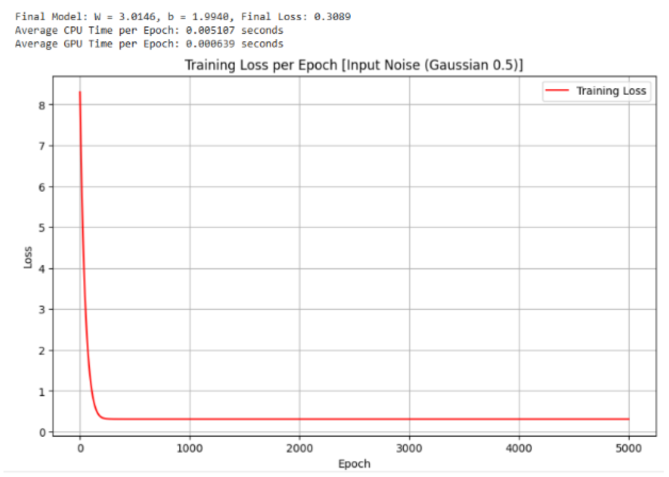
- **Plot the different results.**



```python
experiments = {
    "Input Noise (Gaussian 0.5)": {
        "data_noise": 0.5,
        "weight_noise": 0.0,
        "lr_noise": 0.0,
        "noise_type": "gaussian"
    },
    "Weight Noise (Gaussian 0.5)": {
        "data_noise": 0.0,
        "weight_noise": 0.5,
        "lr_noise": 0.0,
        "noise_type": "gaussian"
    },
    "Learning Rate Noise (Gaussian 0.5)": {
        "data_noise": 0.0,
        "weight_noise": 0.0,
        "lr_noise": 0.5,
        "noise_type": "gaussian"
    }
}
```

Figure 23: Types of Noise added

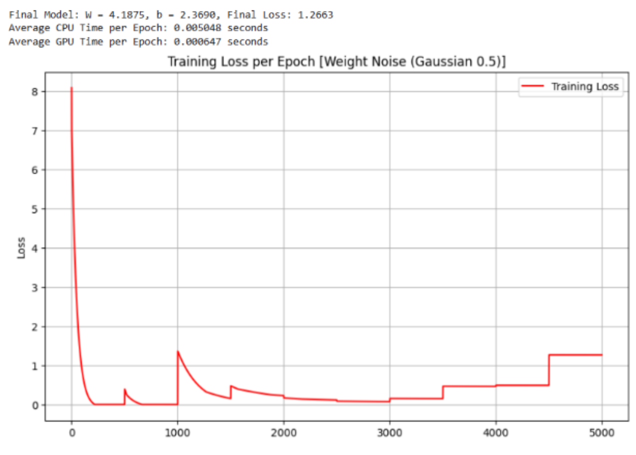
Input Noise: The model reduces the loss and maintains a low loss throughout the training process.



```
Final Model: W = 3.0146, b = 1.9940, Final Loss: 0.3089
Average CPU Time per Epoch: 0.005107 seconds
Average GPU Time per Epoch: 0.000639 seconds
```

Figure 24: Input Noise

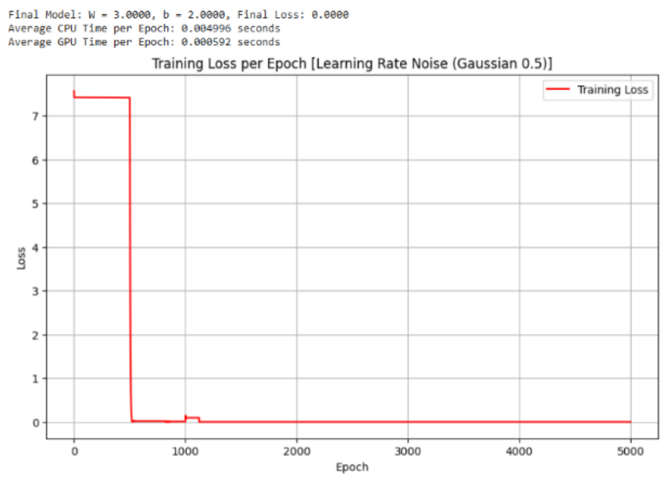
Weight Noise: The loss initially decreases, but then shows spikes and increases in loss after 2000 epochs. The weight noise led to instability in the training process.



```
Final Model: W = 4.1875, b = 2.3690, Final Loss: 1.2663
Average CPU Time per Epoch: 0.005048 seconds
Average GPU Time per Epoch: 0.000647 seconds
```

Figure 25: Enter Caption

Learning Rate Noise: The model quickly converges to a very low loss within the first 1000 epochs, after which the loss remains zero.



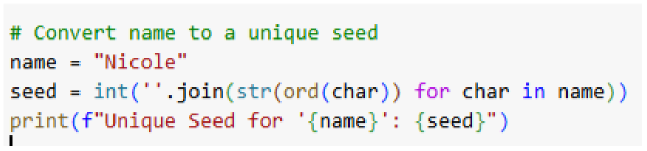
```
Final Model: W = 3.0000, b = 2.0000, Final Loss: 0.0000
Average CPU Time per Epoch: 0.004996 seconds
Average GPU Time per Epoch: 0.000592 seconds
```

Figure 26: Enter Caption

- **Do you get the exact same results if you run the notebook multiple times without changing any parameters? Why or why not? [Explain significance of the seed].**

When running a notebook multiple times without changing any parameters, you do not get exact same results unless a random seed is set. This is because many processes, such as weight initialization involve random operations. Setting a seed ensures that random operations produce the same outputs every time, leading to reproducible results.

- **Use a unique seed for each experiment. (Note: Convert your first name into decimal).**



```python
# Convert name to a unique seed
name = "Nicole"
seed = int(''.join(str(ord(char)) for char in name))
print(f"Unique Seed for '{name}': {seed}")
```

Figure 27: My seed

I used this seed for all runs.

- **Later report per epoch GPU vs CPU time.**

Input Noise:

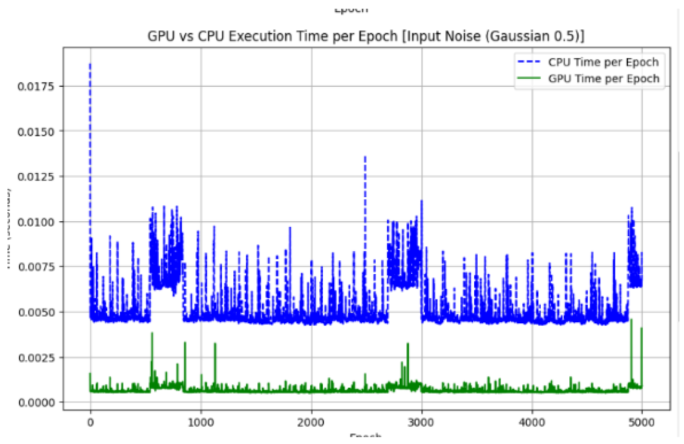


Figure 28: Input Noise

Weight Noise:

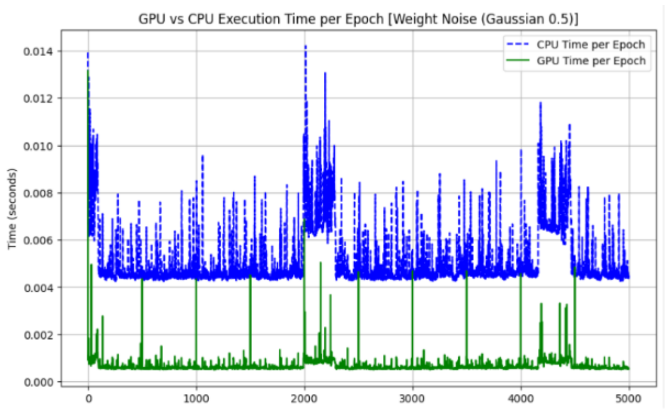


Figure 29: Weight Noise

Learning Rate Noise

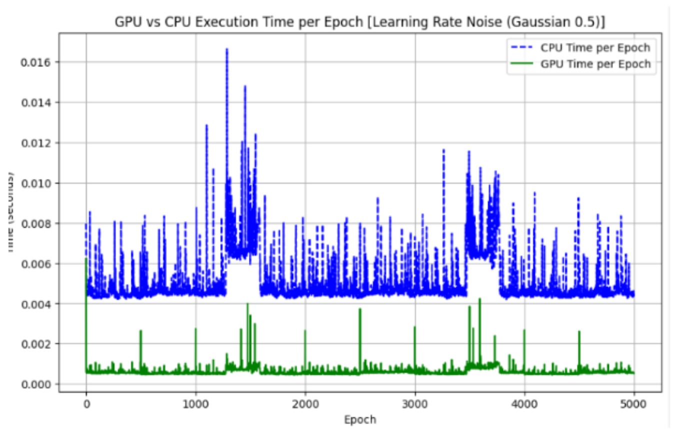


Figure 30: Learning Rate Noise

- **Can you get a model that is robust to noise? Does the model lead to faster convergence? Do you get better local minima? Is noise beneficial?**

It is possible to develop a model that is robust to noise by incorporating noise handling techniques and regularization strategies. From the plots above, the model with learning rate noise showed high robustness, achieving a final loss of 0.0000 despite noisy learning rates. This suggests that the model adapted to variable learning speeds. Additionally, the input noise experiment showed that the model could generalize well even with noisy data.

The learning rate noise experiment demonstrated fast convergence, reaching zero loss within the first 1000 epochs. The input noise model also showed relatively fast convergence, with the loss dropping quickly and stabilizing. Weight noise did not lead to faster convergence.

In the learning rate noise, the model found a global minimum. With input noise, the model achieved a local minimum with a final loss of 0.3089. The weight noise model ended with a higher final loss (1.2663), suggesting that perturbations might have caused local minimum.

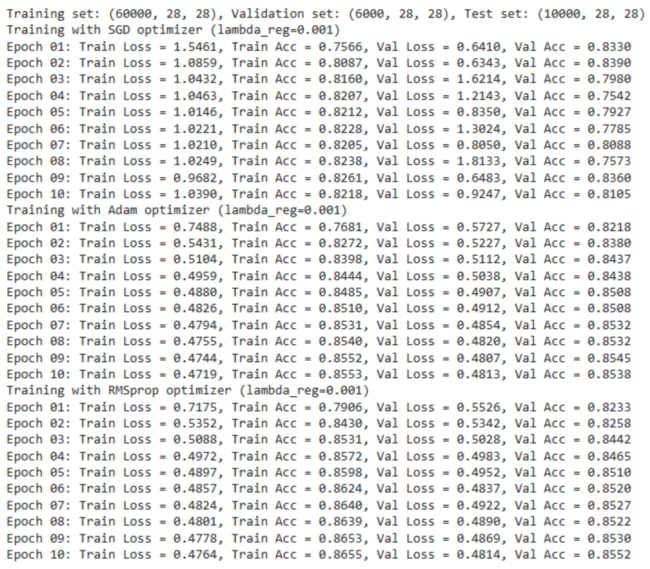
Noise can be beneficial. Input noise acts as a regularizer, improving generalization by making the model more resilient to noise. Learning rate noise can help achieve faster convergence and avoid local minima. However, weight noise lead to instability and poor convergence.

- **Collect everything and report your findings. See above**

2 Problem 2: Logistic Regression (2+2+1=5 points)

- **Fork the repo** `https://github.com/AnkurMali/IST597_Fall2019_TF2.0` **and modify the file** `log_reg.py`**.**

- **TODO: Implement the required section/function/formula as instructed in the repository.**
- **The report should include `matplotlib` plots from the function `plot_images` and `plot_weights`.**
- **Change the optimizer and report which one converges faster and which one reaches better local minima/generalizes better. (Note: You can use TensorFlow optimizers, but not Keras).**



```
Training set: (60000, 28, 28), Validation set: (6000, 28, 28), Test set: (10000, 28, 28)
Training with SGD optimizer (lambda_reg=0.001)
Epoch 01: Train Loss = 1.5461, Train Acc = 0.7566, Val Loss = 0.6410, Val Acc = 0.8330
Epoch 02: Train Loss = 1.0859, Train Acc = 0.8087, Val Loss = 0.6343, Val Acc = 0.8390
Epoch 03: Train Loss = 1.0432, Train Acc = 0.8160, Val Loss = 1.6214, Val Acc = 0.7980
Epoch 04: Train Loss = 1.0463, Train Acc = 0.8207, Val Loss = 1.2143, Val Acc = 0.7542
Epoch 05: Train Loss = 1.0146, Train Acc = 0.8212, Val Loss = 0.8350, Val Acc = 0.7927
Epoch 06: Train Loss = 1.0221, Train Acc = 0.8228, Val Loss = 1.3024, Val Acc = 0.7785
Epoch 07: Train Loss = 1.0210, Train Acc = 0.8205, Val Loss = 0.8050, Val Acc = 0.8088
Epoch 08: Train Loss = 1.0249, Train Acc = 0.8238, Val Loss = 1.8133, Val Acc = 0.7573
Epoch 09: Train Loss = 0.9682, Train Acc = 0.8261, Val Loss = 0.6483, Val Acc = 0.8360
Epoch 10: Train Loss = 1.0390, Train Acc = 0.8218, Val Loss = 0.9247, Val Acc = 0.8105
Training with Adam optimizer (lambda_reg=0.001)
Epoch 01: Train Loss = 0.7488, Train Acc = 0.7681, Val Loss = 0.5727, Val Acc = 0.8218
Epoch 02: Train Loss = 0.5431, Train Acc = 0.8272, Val Loss = 0.5227, Val Acc = 0.8380
Epoch 03: Train Loss = 0.5104, Train Acc = 0.8398, Val Loss = 0.5112, Val Acc = 0.8437
Epoch 04: Train Loss = 0.4959, Train Acc = 0.8444, Val Loss = 0.5038, Val Acc = 0.8438
Epoch 05: Train Loss = 0.4880, Train Acc = 0.8485, Val Loss = 0.4907, Val Acc = 0.8508
Epoch 06: Train Loss = 0.4826, Train Acc = 0.8510, Val Loss = 0.4912, Val Acc = 0.8508
Epoch 07: Train Loss = 0.4794, Train Acc = 0.8531, Val Loss = 0.4854, Val Acc = 0.8532
Epoch 08: Train Loss = 0.4755, Train Acc = 0.8540, Val Loss = 0.4820, Val Acc = 0.8532
Epoch 09: Train Loss = 0.4744, Train Acc = 0.8552, Val Loss = 0.4807, Val Acc = 0.8545
Epoch 10: Train Loss = 0.4719, Train Acc = 0.8553, Val Loss = 0.4813, Val Acc = 0.8538
Training with RMSprop optimizer (lambda_reg=0.001)
Epoch 01: Train Loss = 0.7175, Train Acc = 0.7906, Val Loss = 0.5526, Val Acc = 0.8233
Epoch 02: Train Loss = 0.5352, Train Acc = 0.8430, Val Loss = 0.5342, Val Acc = 0.8258
Epoch 03: Train Loss = 0.5088, Train Acc = 0.8531, Val Loss = 0.5028, Val Acc = 0.8442
Epoch 04: Train Loss = 0.4972, Train Acc = 0.8572, Val Loss = 0.4983, Val Acc = 0.8465
Epoch 05: Train Loss = 0.4897, Train Acc = 0.8598, Val Loss = 0.4952, Val Acc = 0.8510
Epoch 06: Train Loss = 0.4857, Train Acc = 0.8624, Val Loss = 0.4837, Val Acc = 0.8520
Epoch 07: Train Loss = 0.4824, Train Acc = 0.8640, Val Loss = 0.4922, Val Acc = 0.8527
Epoch 08: Train Loss = 0.4801, Train Acc = 0.8639, Val Loss = 0.4890, Val Acc = 0.8522
Epoch 09: Train Loss = 0.4778, Train Acc = 0.8653, Val Loss = 0.4869, Val Acc = 0.8530
Epoch 10: Train Loss = 0.4764, Train Acc = 0.8655, Val Loss = 0.4814, Val Acc = 0.8552
```

Figure 31: 3 Optimizers

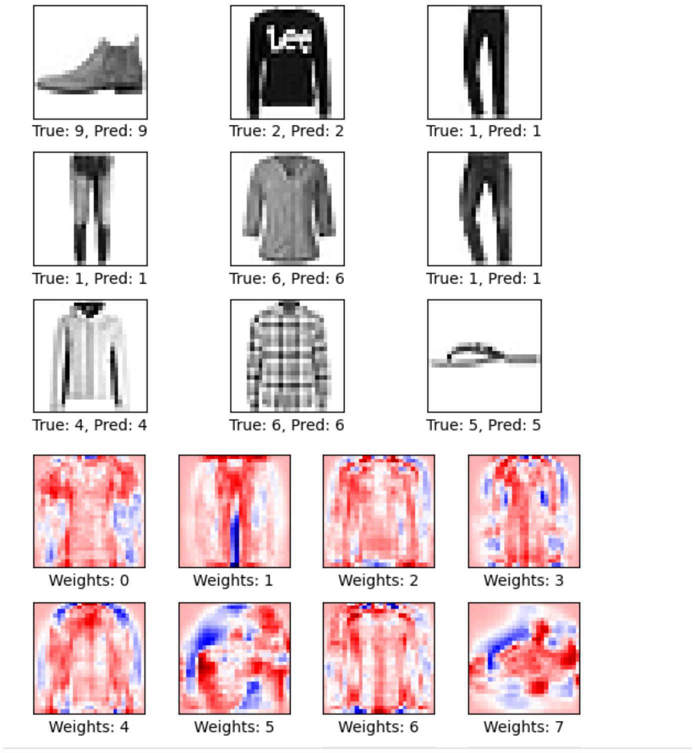
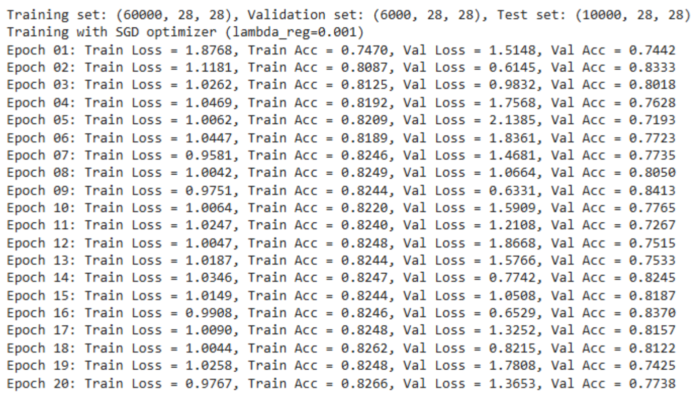


Figure 32: Plot output

The Adam optimizer converges the fastest, achieving stable accuracy and loss reduction by Epoch 3, while RMSprop also converges quickly but slower than Adam. SGD, however, converges slower with more variance in performance across epochs.

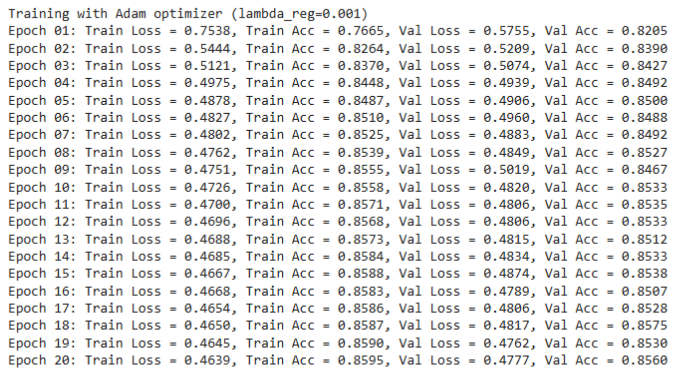
In terms of generalization and reaching a good local minima, Adam and RMSprop both perform well, achieving validation accuracy around 85 percent

- **Train for a longer number of epochs. Changed number of epochs from 10 to 20:**



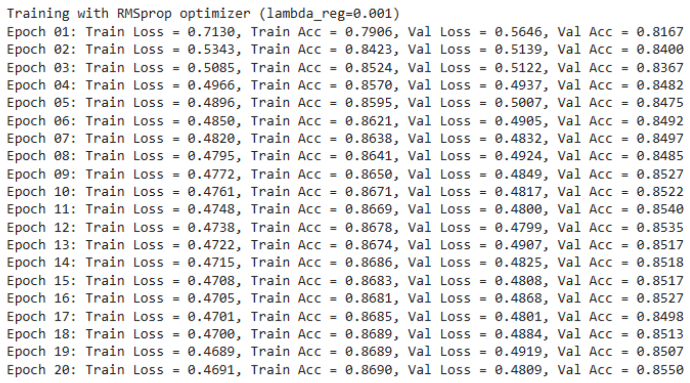
```
Training set: (60000, 28, 28), Validation set: (6000, 28, 28), Test set: (10000, 28, 28)
Training with SGD optimizer (lambda_reg=0.001)
Epoch 01: Train Loss = 1.8768, Train Acc = 0.7470, Val Loss = 1.5148, Val Acc = 0.7442
Epoch 02: Train Loss = 1.1181, Train Acc = 0.8087, Val Loss = 0.6145, Val Acc = 0.8333
Epoch 03: Train Loss = 1.0262, Train Acc = 0.8125, Val Loss = 0.9832, Val Acc = 0.8018
Epoch 04: Train Loss = 1.0469, Train Acc = 0.8192, Val Loss = 1.7568, Val Acc = 0.7628
Epoch 05: Train Loss = 1.0062, Train Acc = 0.8209, Val Loss = 2.1385, Val Acc = 0.7193
Epoch 06: Train Loss = 1.0447, Train Acc = 0.8189, Val Loss = 1.8361, Val Acc = 0.7723
Epoch 07: Train Loss = 0.9581, Train Acc = 0.8246, Val Loss = 1.4681, Val Acc = 0.7735
Epoch 08: Train Loss = 1.0042, Train Acc = 0.8249, Val Loss = 1.0664, Val Acc = 0.8050
Epoch 09: Train Loss = 0.9751, Train Acc = 0.8244, Val Loss = 0.6331, Val Acc = 0.8413
Epoch 10: Train Loss = 1.0064, Train Acc = 0.8220, Val Loss = 1.5909, Val Acc = 0.7765
Epoch 11: Train Loss = 1.0247, Train Acc = 0.8240, Val Loss = 1.2108, Val Acc = 0.7267
Epoch 12: Train Loss = 1.0047, Train Acc = 0.8248, Val Loss = 1.8668, Val Acc = 0.7515
Epoch 13: Train Loss = 1.0187, Train Acc = 0.8244, Val Loss = 1.5766, Val Acc = 0.7533
Epoch 14: Train Loss = 1.0346, Train Acc = 0.8247, Val Loss = 0.7742, Val Acc = 0.8245
Epoch 15: Train Loss = 1.0149, Train Acc = 0.8244, Val Loss = 1.0508, Val Acc = 0.8187
Epoch 16: Train Loss = 0.9908, Train Acc = 0.8246, Val Loss = 0.6529, Val Acc = 0.8370
Epoch 17: Train Loss = 1.0090, Train Acc = 0.8248, Val Loss = 1.3252, Val Acc = 0.8157
Epoch 18: Train Loss = 1.0044, Train Acc = 0.8262, Val Loss = 0.8215, Val Acc = 0.8122
Epoch 19: Train Loss = 1.0258, Train Acc = 0.8248, Val Loss = 1.7808, Val Acc = 0.7425
Epoch 20: Train Loss = 0.9767, Train Acc = 0.8266, Val Loss = 1.3653, Val Acc = 0.7738
```

Figure 33: 20 epochs



```
Training with Adam optimizer (lambda_reg=0.001)
Epoch 01: Train Loss = 0.7538, Train Acc = 0.7665, Val Loss = 0.5755, Val Acc = 0.8205
Epoch 02: Train Loss = 0.5444, Train Acc = 0.8264, Val Loss = 0.5209, Val Acc = 0.8390
Epoch 03: Train Loss = 0.5121, Train Acc = 0.8370, Val Loss = 0.5074, Val Acc = 0.8427
Epoch 04: Train Loss = 0.4975, Train Acc = 0.8448, Val Loss = 0.4939, Val Acc = 0.8492
Epoch 05: Train Loss = 0.4878, Train Acc = 0.8487, Val Loss = 0.4906, Val Acc = 0.8500
Epoch 06: Train Loss = 0.4827, Train Acc = 0.8510, Val Loss = 0.4960, Val Acc = 0.8488
Epoch 07: Train Loss = 0.4802, Train Acc = 0.8525, Val Loss = 0.4883, Val Acc = 0.8492
Epoch 08: Train Loss = 0.4762, Train Acc = 0.8539, Val Loss = 0.4849, Val Acc = 0.8527
Epoch 09: Train Loss = 0.4751, Train Acc = 0.8555, Val Loss = 0.5019, Val Acc = 0.8467
Epoch 10: Train Loss = 0.4726, Train Acc = 0.8558, Val Loss = 0.4820, Val Acc = 0.8533
Epoch 11: Train Loss = 0.4700, Train Acc = 0.8571, Val Loss = 0.4806, Val Acc = 0.8535
Epoch 12: Train Loss = 0.4696, Train Acc = 0.8568, Val Loss = 0.4806, Val Acc = 0.8533
Epoch 13: Train Loss = 0.4688, Train Acc = 0.8573, Val Loss = 0.4815, Val Acc = 0.8512
Epoch 14: Train Loss = 0.4685, Train Acc = 0.8584, Val Loss = 0.4834, Val Acc = 0.8533
Epoch 15: Train Loss = 0.4667, Train Acc = 0.8588, Val Loss = 0.4874, Val Acc = 0.8538
Epoch 16: Train Loss = 0.4668, Train Acc = 0.8583, Val Loss = 0.4789, Val Acc = 0.8507
Epoch 17: Train Loss = 0.4654, Train Acc = 0.8586, Val Loss = 0.4806, Val Acc = 0.8528
Epoch 18: Train Loss = 0.4650, Train Acc = 0.8587, Val Loss = 0.4817, Val Acc = 0.8575
Epoch 19: Train Loss = 0.4645, Train Acc = 0.8590, Val Loss = 0.4762, Val Acc = 0.8530
Epoch 20: Train Loss = 0.4639, Train Acc = 0.8595, Val Loss = 0.4777, Val Acc = 0.8560
```

Figure 34: 20 epochs



```
Training with RMSprop optimizer (lambda_reg=0.001)
Epoch 01: Train Loss = 0.7130, Train Acc = 0.7906, Val Loss = 0.5646, Val Acc = 0.8167
Epoch 02: Train Loss = 0.5343, Train Acc = 0.8423, Val Loss = 0.5139, Val Acc = 0.8400
Epoch 03: Train Loss = 0.5085, Train Acc = 0.8524, Val Loss = 0.5122, Val Acc = 0.8367
Epoch 04: Train Loss = 0.4966, Train Acc = 0.8570, Val Loss = 0.4937, Val Acc = 0.8482
Epoch 05: Train Loss = 0.4896, Train Acc = 0.8595, Val Loss = 0.5007, Val Acc = 0.8475
Epoch 06: Train Loss = 0.4850, Train Acc = 0.8621, Val Loss = 0.4905, Val Acc = 0.8492
Epoch 07: Train Loss = 0.4820, Train Acc = 0.8638, Val Loss = 0.4832, Val Acc = 0.8497
Epoch 08: Train Loss = 0.4795, Train Acc = 0.8641, Val Loss = 0.4924, Val Acc = 0.8485
Epoch 09: Train Loss = 0.4772, Train Acc = 0.8650, Val Loss = 0.4849, Val Acc = 0.8527
Epoch 10: Train Loss = 0.4761, Train Acc = 0.8671, Val Loss = 0.4817, Val Acc = 0.8522
Epoch 11: Train Loss = 0.4748, Train Acc = 0.8669, Val Loss = 0.4800, Val Acc = 0.8540
Epoch 12: Train Loss = 0.4738, Train Acc = 0.8678, Val Loss = 0.4799, Val Acc = 0.8535
Epoch 13: Train Loss = 0.4722, Train Acc = 0.8674, Val Loss = 0.4907, Val Acc = 0.8517
Epoch 14: Train Loss = 0.4715, Train Acc = 0.8686, Val Loss = 0.4825, Val Acc = 0.8518
Epoch 15: Train Loss = 0.4708, Train Acc = 0.8683, Val Loss = 0.4808, Val Acc = 0.8517
Epoch 16: Train Loss = 0.4705, Train Acc = 0.8681, Val Loss = 0.4868, Val Acc = 0.8527
Epoch 17: Train Loss = 0.4701, Train Acc = 0.8685, Val Loss = 0.4801, Val Acc = 0.8498
Epoch 18: Train Loss = 0.4700, Train Acc = 0.8689, Val Loss = 0.4884, Val Acc = 0.8513
Epoch 19: Train Loss = 0.4689, Train Acc = 0.8689, Val Loss = 0.4919, Val Acc = 0.8507
Epoch 20: Train Loss = 0.4691, Train Acc = 0.8690, Val Loss = 0.4809, Val Acc = 0.8550
```

Figure 35: 20 epochs

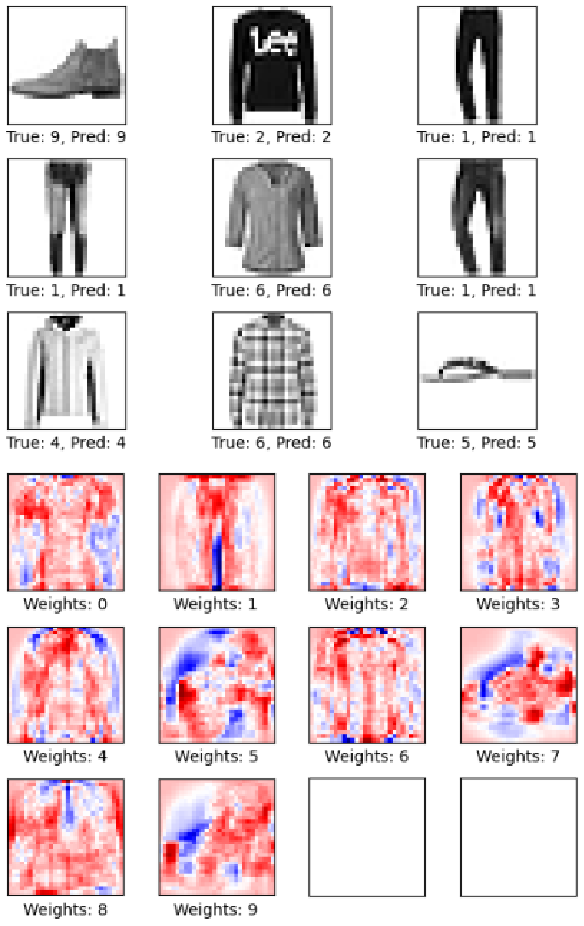
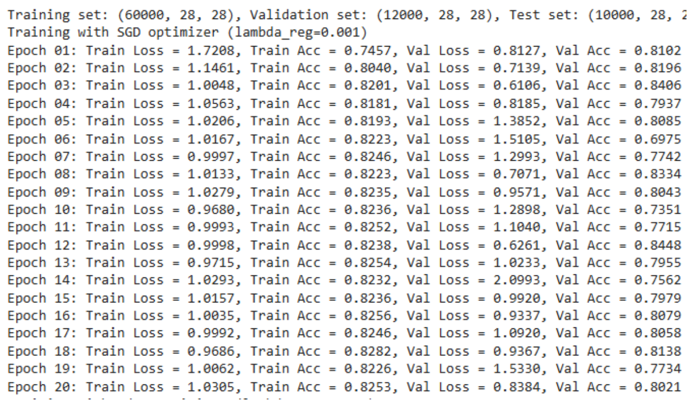


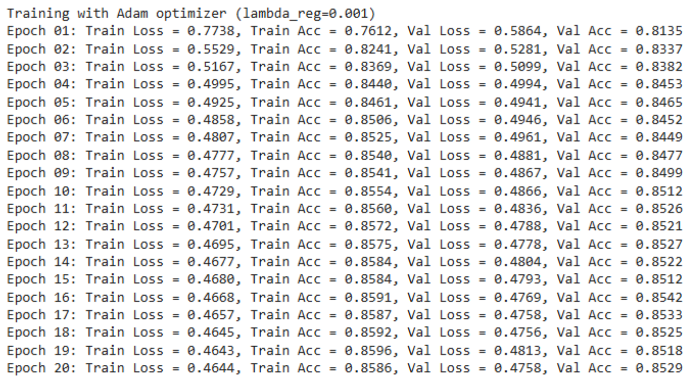
Figure 36: 20 epochs Plot output

- **Change the Train/Validation split. Report if you observe any performance drop or gain.** Changed the Train/Validation split from 0.1 to 0.2 There was a performance gain in SGD optimizer.



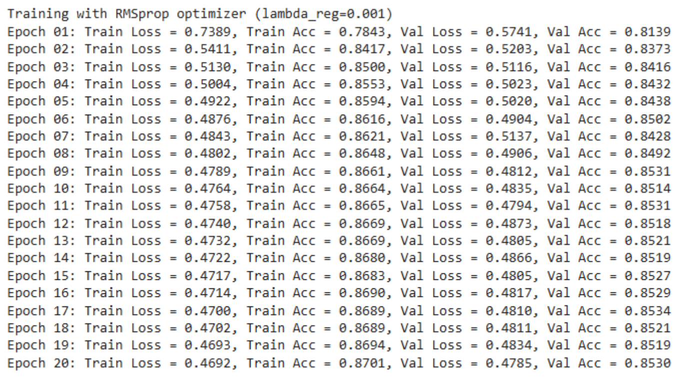
```
Training set: (60000, 28, 28), Validation set: (12000, 28, 28), Test set: (10000, 28, 2
Training with SGD optimizer (lambda_reg=0.001)
Epoch 01: Train Loss = 1.7208, Train Acc = 0.7457, Val Loss = 0.8127, Val Acc = 0.8102
Epoch 02: Train Loss = 1.1461, Train Acc = 0.8040, Val Loss = 0.7139, Val Acc = 0.8196
Epoch 03: Train Loss = 1.0048, Train Acc = 0.8201, Val Loss = 0.6106, Val Acc = 0.8406
Epoch 04: Train Loss = 1.0563, Train Acc = 0.8181, Val Loss = 0.8185, Val Acc = 0.7937
Epoch 05: Train Loss = 1.0206, Train Acc = 0.8193, Val Loss = 1.3852, Val Acc = 0.8085
Epoch 06: Train Loss = 1.0167, Train Acc = 0.8223, Val Loss = 1.5105, Val Acc = 0.6975
Epoch 07: Train Loss = 0.9997, Train Acc = 0.8246, Val Loss = 1.2993, Val Acc = 0.7742
Epoch 08: Train Loss = 1.0133, Train Acc = 0.8223, Val Loss = 0.7071, Val Acc = 0.8334
Epoch 09: Train Loss = 1.0279, Train Acc = 0.8235, Val Loss = 0.9571, Val Acc = 0.8043
Epoch 10: Train Loss = 0.9680, Train Acc = 0.8236, Val Loss = 1.2898, Val Acc = 0.7351
Epoch 11: Train Loss = 0.9993, Train Acc = 0.8252, Val Loss = 1.1040, Val Acc = 0.7715
Epoch 12: Train Loss = 0.9998, Train Acc = 0.8238, Val Loss = 0.6261, Val Acc = 0.8448
Epoch 13: Train Loss = 0.9715, Train Acc = 0.8254, Val Loss = 1.0233, Val Acc = 0.7955
Epoch 14: Train Loss = 1.0293, Train Acc = 0.8232, Val Loss = 2.0993, Val Acc = 0.7562
Epoch 15: Train Loss = 1.0157, Train Acc = 0.8236, Val Loss = 0.9920, Val Acc = 0.7979
Epoch 16: Train Loss = 1.0035, Train Acc = 0.8256, Val Loss = 0.9337, Val Acc = 0.8079
Epoch 17: Train Loss = 0.9992, Train Acc = 0.8246, Val Loss = 1.0920, Val Acc = 0.8058
Epoch 18: Train Loss = 0.9686, Train Acc = 0.8282, Val Loss = 0.9367, Val Acc = 0.8138
Epoch 19: Train Loss = 1.0062, Train Acc = 0.8226, Val Loss = 1.5330, Val Acc = 0.7734
Epoch 20: Train Loss = 1.0305, Train Acc = 0.8253, Val Loss = 0.8384, Val Acc = 0.8021
```

Figure 37: Validation split 0.2



```
Training with Adam optimizer (lambda_reg=0.001)
Epoch 01: Train Loss = 0.7738, Train Acc = 0.7612, Val Loss = 0.5864, Val Acc = 0.8135
Epoch 02: Train Loss = 0.5529, Train Acc = 0.8241, Val Loss = 0.5281, Val Acc = 0.8337
Epoch 03: Train Loss = 0.5167, Train Acc = 0.8369, Val Loss = 0.5099, Val Acc = 0.8382
Epoch 04: Train Loss = 0.4995, Train Acc = 0.8440, Val Loss = 0.4994, Val Acc = 0.8453
Epoch 05: Train Loss = 0.4925, Train Acc = 0.8461, Val Loss = 0.4941, Val Acc = 0.8465
Epoch 06: Train Loss = 0.4858, Train Acc = 0.8506, Val Loss = 0.4946, Val Acc = 0.8452
Epoch 07: Train Loss = 0.4807, Train Acc = 0.8525, Val Loss = 0.4961, Val Acc = 0.8449
Epoch 08: Train Loss = 0.4777, Train Acc = 0.8540, Val Loss = 0.4881, Val Acc = 0.8477
Epoch 09: Train Loss = 0.4757, Train Acc = 0.8541, Val Loss = 0.4867, Val Acc = 0.8499
Epoch 10: Train Loss = 0.4729, Train Acc = 0.8554, Val Loss = 0.4866, Val Acc = 0.8512
Epoch 11: Train Loss = 0.4731, Train Acc = 0.8560, Val Loss = 0.4836, Val Acc = 0.8526
Epoch 12: Train Loss = 0.4701, Train Acc = 0.8572, Val Loss = 0.4788, Val Acc = 0.8521
Epoch 13: Train Loss = 0.4695, Train Acc = 0.8575, Val Loss = 0.4778, Val Acc = 0.8527
Epoch 14: Train Loss = 0.4677, Train Acc = 0.8584, Val Loss = 0.4804, Val Acc = 0.8522
Epoch 15: Train Loss = 0.4680, Train Acc = 0.8584, Val Loss = 0.4793, Val Acc = 0.8512
Epoch 16: Train Loss = 0.4668, Train Acc = 0.8591, Val Loss = 0.4769, Val Acc = 0.8542
Epoch 17: Train Loss = 0.4657, Train Acc = 0.8587, Val Loss = 0.4758, Val Acc = 0.8533
Epoch 18: Train Loss = 0.4645, Train Acc = 0.8592, Val Loss = 0.4756, Val Acc = 0.8525
Epoch 19: Train Loss = 0.4643, Train Acc = 0.8596, Val Loss = 0.4813, Val Acc = 0.8518
Epoch 20: Train Loss = 0.4644, Train Acc = 0.8586, Val Loss = 0.4758, Val Acc = 0.8529
```

Figure 38: Validation split 0.2



```
Training with RMSprop optimizer (lambda_reg=0.001)
Epoch 01: Train Loss = 0.7389, Train Acc = 0.7843, Val Loss = 0.5741, Val Acc = 0.8139
Epoch 02: Train Loss = 0.5411, Train Acc = 0.8417, Val Loss = 0.5203, Val Acc = 0.8373
Epoch 03: Train Loss = 0.5130, Train Acc = 0.8500, Val Loss = 0.5116, Val Acc = 0.8416
Epoch 04: Train Loss = 0.5004, Train Acc = 0.8553, Val Loss = 0.5023, Val Acc = 0.8432
Epoch 05: Train Loss = 0.4922, Train Acc = 0.8594, Val Loss = 0.5020, Val Acc = 0.8438
Epoch 06: Train Loss = 0.4876, Train Acc = 0.8616, Val Loss = 0.4904, Val Acc = 0.8502
Epoch 07: Train Loss = 0.4843, Train Acc = 0.8621, Val Loss = 0.5137, Val Acc = 0.8428
Epoch 08: Train Loss = 0.4802, Train Acc = 0.8648, Val Loss = 0.4906, Val Acc = 0.8492
Epoch 09: Train Loss = 0.4789, Train Acc = 0.8661, Val Loss = 0.4812, Val Acc = 0.8531
Epoch 10: Train Loss = 0.4764, Train Acc = 0.8664, Val Loss = 0.4835, Val Acc = 0.8514
Epoch 11: Train Loss = 0.4758, Train Acc = 0.8665, Val Loss = 0.4794, Val Acc = 0.8531
Epoch 12: Train Loss = 0.4740, Train Acc = 0.8669, Val Loss = 0.4873, Val Acc = 0.8518
Epoch 13: Train Loss = 0.4732, Train Acc = 0.8669, Val Loss = 0.4805, Val Acc = 0.8521
Epoch 14: Train Loss = 0.4722, Train Acc = 0.8680, Val Loss = 0.4866, Val Acc = 0.8519
Epoch 15: Train Loss = 0.4717, Train Acc = 0.8683, Val Loss = 0.4805, Val Acc = 0.8527
Epoch 16: Train Loss = 0.4714, Train Acc = 0.8690, Val Loss = 0.4817, Val Acc = 0.8529
Epoch 17: Train Loss = 0.4700, Train Acc = 0.8689, Val Loss = 0.4810, Val Acc = 0.8534
Epoch 18: Train Loss = 0.4702, Train Acc = 0.8689, Val Loss = 0.4811, Val Acc = 0.8521
Epoch 19: Train Loss = 0.4693, Train Acc = 0.8694, Val Loss = 0.4834, Val Acc = 0.8519
Epoch 20: Train Loss = 0.4692, Train Acc = 0.8701, Val Loss = 0.4785, Val Acc = 0.8530
```

Figure 39: Validation split 0.2

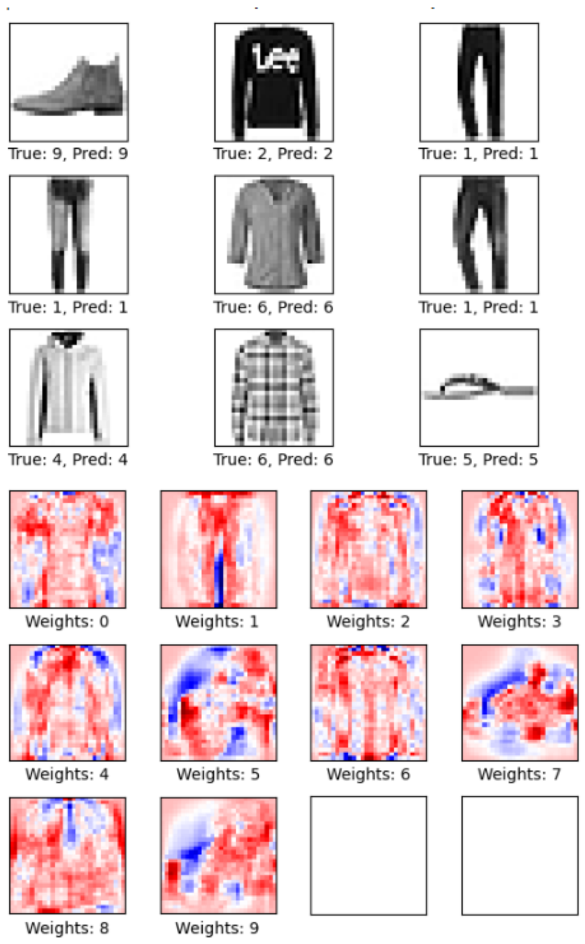
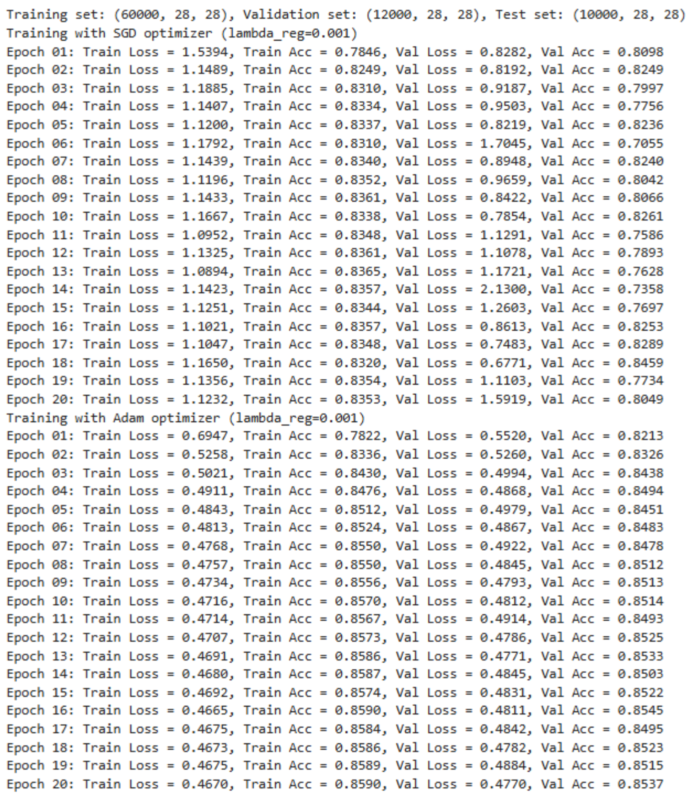


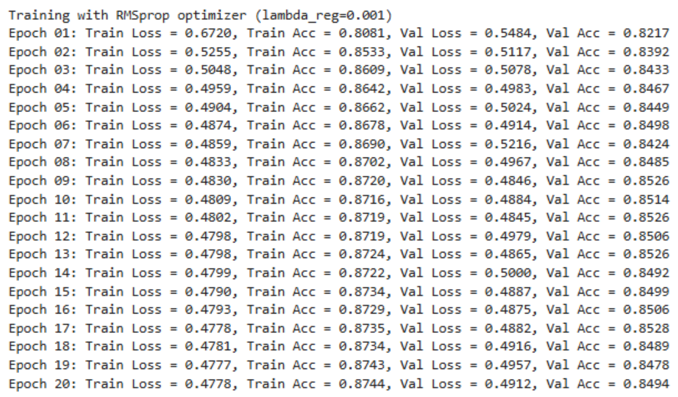
Figure 40: Validation split 0.2 Plot

- **Report Train/Validation accuracy over time. See above output.** Already reported for all 3 optimizations
- **Investigate whether batch size has any effect on performance.** I changed batch size from 128 to 64. Smaller batch size led to faster initial convergence but introduced more variability in accuracy and loss, while larger batch sizes provided more stable learning curves. The validation accuracy was similar across batch sizes, suggesting batch size did not significantly affect generalization.



```
Training set: (60000, 28, 28), Validation set: (12000, 28, 28), Test set: (10000, 28, 28)
Training with SGD optimizer (lambda_reg=0.001)
Epoch 01: Train Loss = 1.5394, Train Acc = 0.7846, Val Loss = 0.8282, Val Acc = 0.8098
Epoch 02: Train Loss = 1.1489, Train Acc = 0.8249, Val Loss = 0.8192, Val Acc = 0.8249
Epoch 03: Train Loss = 1.1885, Train Acc = 0.8310, Val Loss = 0.9187, Val Acc = 0.7997
Epoch 04: Train Loss = 1.1407, Train Acc = 0.8334, Val Loss = 0.9503, Val Acc = 0.7756
Epoch 05: Train Loss = 1.1200, Train Acc = 0.8337, Val Loss = 0.8219, Val Acc = 0.8236
Epoch 06: Train Loss = 1.1792, Train Acc = 0.8310, Val Loss = 1.7045, Val Acc = 0.7055
Epoch 07: Train Loss = 1.1439, Train Acc = 0.8340, Val Loss = 0.8948, Val Acc = 0.8240
Epoch 08: Train Loss = 1.1196, Train Acc = 0.8352, Val Loss = 0.9659, Val Acc = 0.8042
Epoch 09: Train Loss = 1.1433, Train Acc = 0.8361, Val Loss = 0.8422, Val Acc = 0.8066
Epoch 10: Train Loss = 1.1667, Train Acc = 0.8338, Val Loss = 0.7854, Val Acc = 0.8261
Epoch 11: Train Loss = 1.0952, Train Acc = 0.8348, Val Loss = 1.1291, Val Acc = 0.7586
Epoch 12: Train Loss = 1.1325, Train Acc = 0.8361, Val Loss = 1.1078, Val Acc = 0.7893
Epoch 13: Train Loss = 1.0894, Train Acc = 0.8365, Val Loss = 1.1721, Val Acc = 0.7628
Epoch 14: Train Loss = 1.1423, Train Acc = 0.8357, Val Loss = 2.1300, Val Acc = 0.7358
Epoch 15: Train Loss = 1.1251, Train Acc = 0.8344, Val Loss = 1.2603, Val Acc = 0.7697
Epoch 16: Train Loss = 1.1021, Train Acc = 0.8357, Val Loss = 0.8613, Val Acc = 0.8253
Epoch 17: Train Loss = 1.1047, Train Acc = 0.8348, Val Loss = 0.7483, Val Acc = 0.8289
Epoch 18: Train Loss = 1.1650, Train Acc = 0.8320, Val Loss = 0.6771, Val Acc = 0.8459
Epoch 19: Train Loss = 1.1356, Train Acc = 0.8354, Val Loss = 1.1103, Val Acc = 0.7734
Epoch 20: Train Loss = 1.1232, Train Acc = 0.8353, Val Loss = 1.5919, Val Acc = 0.8049
Training with Adam optimizer (lambda_reg=0.001)
Epoch 01: Train Loss = 0.6947, Train Acc = 0.7822, Val Loss = 0.5520, Val Acc = 0.8213
Epoch 02: Train Loss = 0.5258, Train Acc = 0.8336, Val Loss = 0.5260, Val Acc = 0.8326
Epoch 03: Train Loss = 0.5021, Train Acc = 0.8430, Val Loss = 0.4994, Val Acc = 0.8438
Epoch 04: Train Loss = 0.4911, Train Acc = 0.8476, Val Loss = 0.4868, Val Acc = 0.8494
Epoch 05: Train Loss = 0.4843, Train Acc = 0.8512, Val Loss = 0.4979, Val Acc = 0.8451
Epoch 06: Train Loss = 0.4813, Train Acc = 0.8524, Val Loss = 0.4867, Val Acc = 0.8483
Epoch 07: Train Loss = 0.4768, Train Acc = 0.8550, Val Loss = 0.4922, Val Acc = 0.8478
Epoch 08: Train Loss = 0.4757, Train Acc = 0.8550, Val Loss = 0.4845, Val Acc = 0.8512
Epoch 09: Train Loss = 0.4734, Train Acc = 0.8556, Val Loss = 0.4793, Val Acc = 0.8513
Epoch 10: Train Loss = 0.4716, Train Acc = 0.8570, Val Loss = 0.4812, Val Acc = 0.8514
Epoch 11: Train Loss = 0.4714, Train Acc = 0.8567, Val Loss = 0.4914, Val Acc = 0.8493
Epoch 12: Train Loss = 0.4707, Train Acc = 0.8573, Val Loss = 0.4786, Val Acc = 0.8525
Epoch 13: Train Loss = 0.4691, Train Acc = 0.8586, Val Loss = 0.4771, Val Acc = 0.8533
Epoch 14: Train Loss = 0.4680, Train Acc = 0.8587, Val Loss = 0.4845, Val Acc = 0.8503
Epoch 15: Train Loss = 0.4692, Train Acc = 0.8574, Val Loss = 0.4831, Val Acc = 0.8522
Epoch 16: Train Loss = 0.4665, Train Acc = 0.8590, Val Loss = 0.4811, Val Acc = 0.8545
Epoch 17: Train Loss = 0.4675, Train Acc = 0.8584, Val Loss = 0.4842, Val Acc = 0.8495
Epoch 18: Train Loss = 0.4673, Train Acc = 0.8586, Val Loss = 0.4782, Val Acc = 0.8523
Epoch 19: Train Loss = 0.4675, Train Acc = 0.8589, Val Loss = 0.4884, Val Acc = 0.8515
Epoch 20: Train Loss = 0.4670, Train Acc = 0.8590, Val Loss = 0.4770, Val Acc = 0.8537
```

Figure 41: Batch Size 64



```
Training with RMSprop optimizer (lambda_reg=0.001)
Epoch 01: Train Loss = 0.6720, Train Acc = 0.8081, Val Loss = 0.5484, Val Acc = 0.8217
Epoch 02: Train Loss = 0.5255, Train Acc = 0.8533, Val Loss = 0.5117, Val Acc = 0.8392
Epoch 03: Train Loss = 0.5048, Train Acc = 0.8609, Val Loss = 0.5078, Val Acc = 0.8433
Epoch 04: Train Loss = 0.4959, Train Acc = 0.8642, Val Loss = 0.4983, Val Acc = 0.8467
Epoch 05: Train Loss = 0.4904, Train Acc = 0.8662, Val Loss = 0.5024, Val Acc = 0.8449
Epoch 06: Train Loss = 0.4874, Train Acc = 0.8678, Val Loss = 0.4914, Val Acc = 0.8498
Epoch 07: Train Loss = 0.4859, Train Acc = 0.8690, Val Loss = 0.5216, Val Acc = 0.8424
Epoch 08: Train Loss = 0.4833, Train Acc = 0.8702, Val Loss = 0.4967, Val Acc = 0.8485
Epoch 09: Train Loss = 0.4830, Train Acc = 0.8720, Val Loss = 0.4846, Val Acc = 0.8526
Epoch 10: Train Loss = 0.4809, Train Acc = 0.8716, Val Loss = 0.4884, Val Acc = 0.8514
Epoch 11: Train Loss = 0.4802, Train Acc = 0.8719, Val Loss = 0.4845, Val Acc = 0.8526
Epoch 12: Train Loss = 0.4798, Train Acc = 0.8719, Val Loss = 0.4979, Val Acc = 0.8506
Epoch 13: Train Loss = 0.4798, Train Acc = 0.8724, Val Loss = 0.4865, Val Acc = 0.8526
Epoch 14: Train Loss = 0.4799, Train Acc = 0.8722, Val Loss = 0.5000, Val Acc = 0.8492
Epoch 15: Train Loss = 0.4790, Train Acc = 0.8734, Val Loss = 0.4887, Val Acc = 0.8499
Epoch 16: Train Loss = 0.4793, Train Acc = 0.8729, Val Loss = 0.4875, Val Acc = 0.8506
Epoch 17: Train Loss = 0.4778, Train Acc = 0.8735, Val Loss = 0.4882, Val Acc = 0.8528
Epoch 18: Train Loss = 0.4781, Train Acc = 0.8734, Val Loss = 0.4916, Val Acc = 0.8489
Epoch 19: Train Loss = 0.4777, Train Acc = 0.8743, Val Loss = 0.4957, Val Acc = 0.8478
Epoch 20: Train Loss = 0.4778, Train Acc = 0.8744, Val Loss = 0.4912, Val Acc = 0.8494
```

Figure 42: Batch Size 64

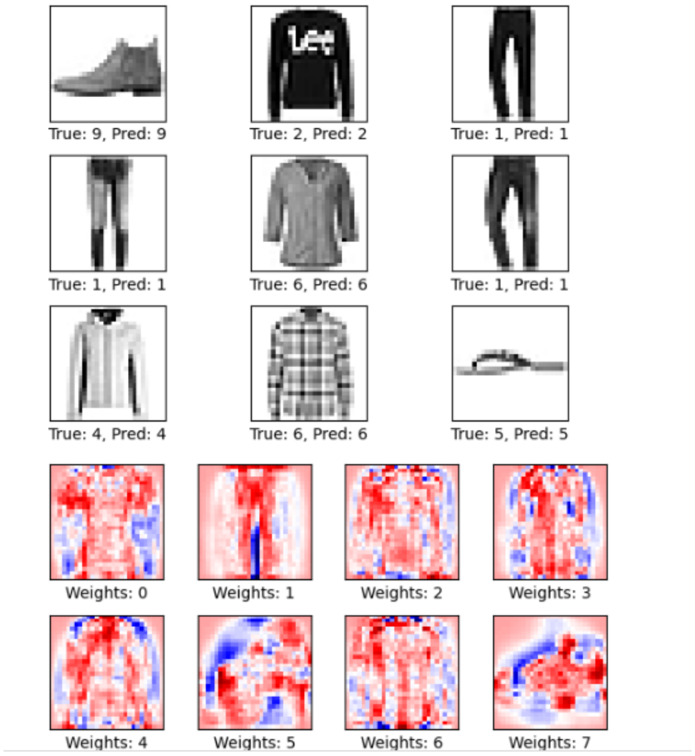


Figure 43: Batch Size 64

- **Report GPU vs CPU performance per epoch.**
- **Does the model overfit? If so, explain why and report the measures you took to avoid overfitting.**
- **Compare performance with Random Forest and Support Vector Machine (SVM). You can use any built-in library for these comparisons.**
- **Cluster the weights for each class using any clustering mechanism. Show t-SNE or k-means clusters.**