ECE655 Project 02

Author: Stewart Schuler

Due: 09/28/2025

Contents

1 Part A

The dataset was generated using a simulated circuit in LTSpice. The diode using test is a model of the *1N4148* and the current limiting resistor has a value of 15 Ω.

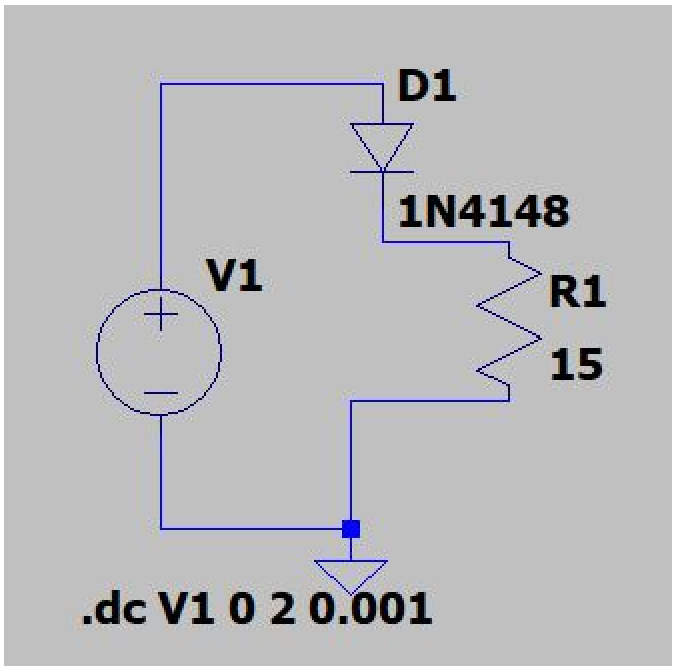

Figure 1: LTSpice Schematic

The voltage source *V1* was simulated to sweep from 0V to 2V DC. At each step measurements were taken for V_d and I_d, the voltage across the diode and current respectively. Figure 2 plots the diode current as a function of voltage. As can be seen there is a clear exponential relationship between the two values.

Because the dataset is being generated synthetically errors representing those of a typical multimeter are introduced. The synthetic error follows the error metrics for the *Agilent U1272A* presented in the project guide lines. Using the synthetically noisy voltage measurement I_d was computed as $I_d = V_d/15$, which produces a noisy current measurement as opposed to the pristine simulated value.

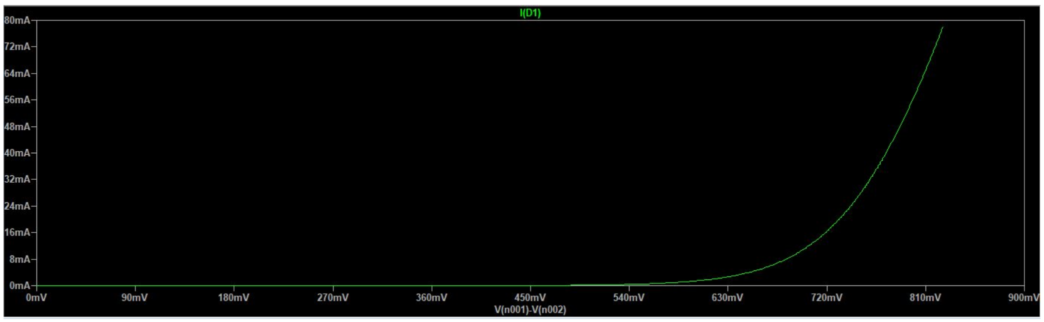

Figure 2: I_d vs V_d

```python
# Introduce measurement error - Using project example
v_r = dataset[:,2]; # Voltage at current limiting resitor
v_s = dataset[:,1]; # DC Source Voltage

# Clip to 4 decimal palces
v_r = (np.round(v_r*100000))/100000;

# Create noiseM - Normal distribtuion clipped between +- 0.05% = 0.0005
noiseM_scale = np.random.normal( loc=0, scale=0.0005, size=v_r.shape);
# Create noiseQ - unitform distribution -0.0002 -> + 0.0002
noiseQ = np.random.randint(-2, 2, size=v_r.shape)/10000;

# Add noise to voltage measurement
v_r = (v_r*(1+noiseM_scale)) + (noiseQ);

# Compute Diode current
R = 15; # Resistor Value
i_d = v_r/R;

# Compute diode voltage
v_d = v_s - v_r;
```

Listing 1: part_a.py

2 Part B

For training and validation 50 datapoints were chosen at random from the dataset during the period when the diode is active. Figure 3 shows the noisy exponential measurements that make up the training set.

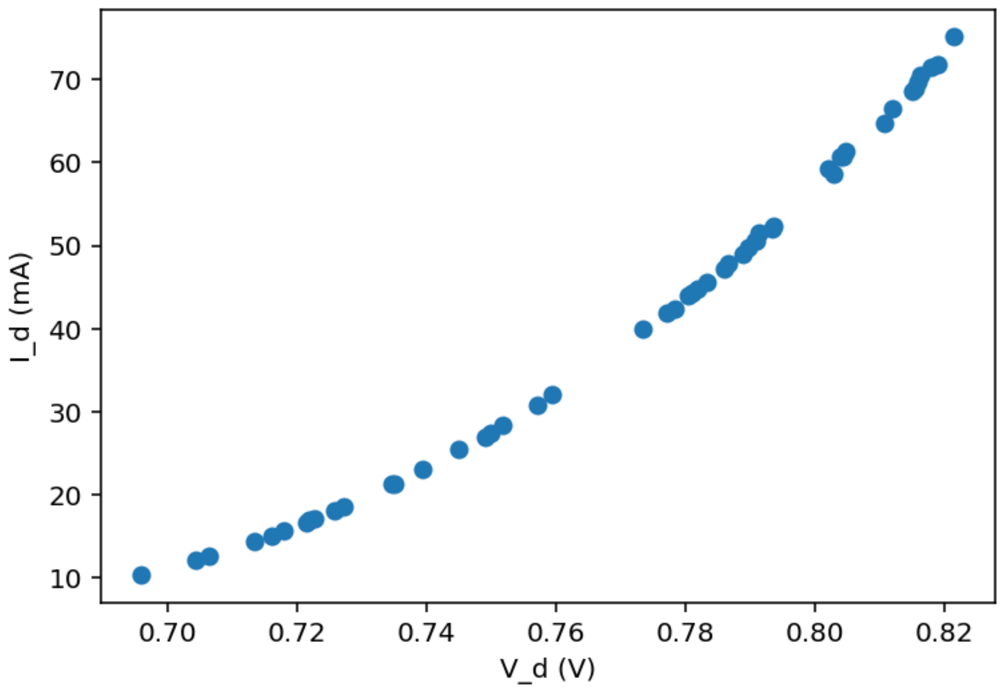

Figure 3: Training dataset

3 Part C

Next, because we are using linear regression the dataset must be made linear. In order to do so, the natural log was taken of I_d. Linear regression was than ran by manually computing the gradients as shown in the below code. Training lasted for 500 epochs and has a learning rate of 0.01. MSE was used to compute the loss and the loss curve is shown in Figure 4. After training the final value of b and w were found to be -15.2738 and 15.5091 respectively when unnormalized. The training loss for the final epoch was 0.001532

```python
lr = 0.01;
n_epochs= 500;

b = np.random.randn(1);
w = np.random.randn(1);
for epoch in range(n_epochs):
    y_hat = b + w * x_train;
    err = (y_hat - y_train);
    loss = (err**2).mean();

    b_grad = 2*err.mean();
    w_grad = 2*(x_train*err).mean();

    b = b - lr*b_grad;
    w = w - lr*w_grad;
```

Listing 2: part_c.py

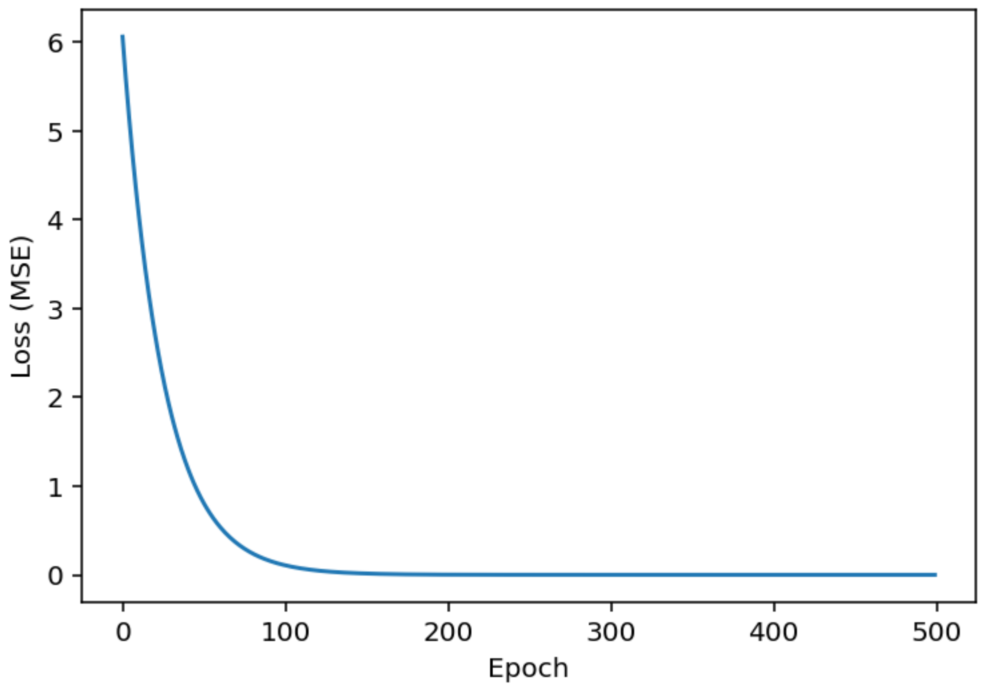

Figure 4: Training Loss Curve

4 Part D/E

Using the learned parameters from Section 3 the current values in the validation set. The learned regression line is shown in Figure **??**. And the validation loss was computed to be 0.001394 which is reasonably inline with the training loss. It being lower than the training loss can be attributed to the small dataset size and only have 10 validation points.

```
y_pred = b + w * x_val;
err = (y_pred - y_val);
loss = (err**2).mean();
```

Listing 3: part_d.py

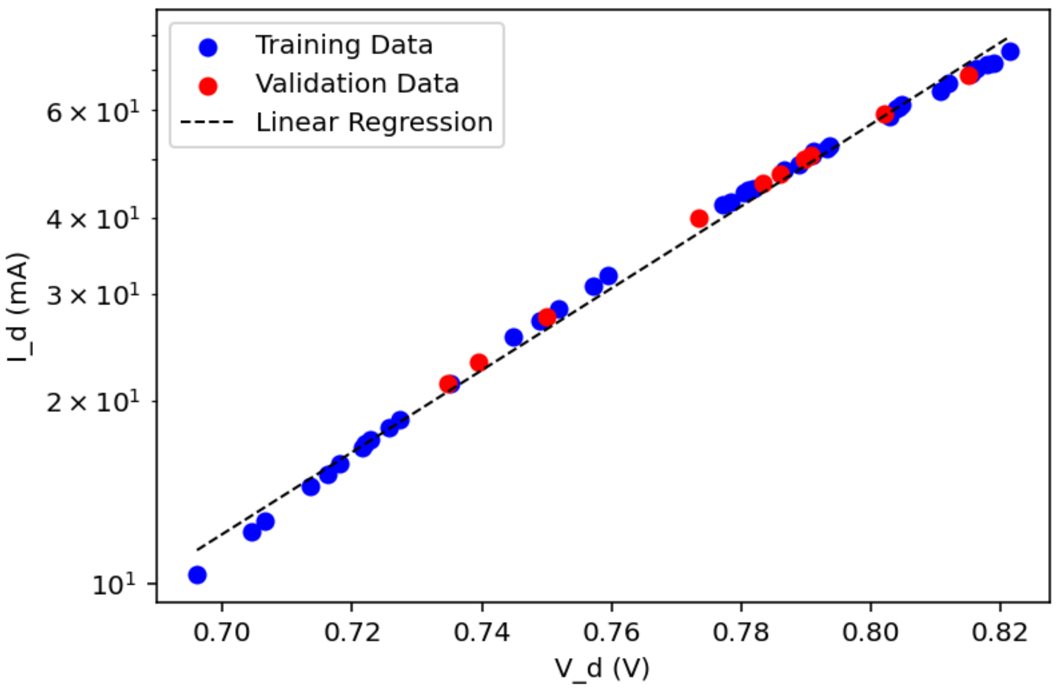

Figure 5: Regression Line

5 Part F

Lastly, the regression parameters were relearned using *PyTorch*. Again the training was computed for 500 epochs and a learning rate of 0.01 was used. Table 1 compares final learned results of the two approaches. Since the learned parameters b and w are nearly identical to the previous described *NumPy* method the resulting fit line would look nearly identical to that in Figure 5.

Method	Learned b	Learned w	Training Loss (MSE)	Validation Loss (MSE)
NumPy	-15.2738	15.50911	0.00153235	0.0013943
PyTorch	-15.2738	15.5109	0.00153235	0.0013934

Table 1: Numpy vs PyTorch Linear Regression Results

```python
class LR_Model(nn.Module):
    def __init__(self):
        super().__init__()
        torch.manual_seed(1140)
        self.b = nn.Parameter(torch.randn(1, requires_grad=True, dtype=torch.float))
        self.w = nn.Parameter(torch.randn(1, requires_grad=True, dtype=torch.float))
    def forward(self, x):
        return self.b + self.w * x


x_train_tensor = torch.as_tensor(x_train).float()
y_train_tensor = torch.as_tensor(y_train).float()

model = LR_Model()
optimizer=optim.SGD(model.parameters(), lr=lr)
loss_fn=nn.MSELoss(reduction='mean')

for epoch in range(n_epochs):
    model.train()
    yhat = model(x_train_tensor)
    loss = loss_fn(yhat, y_train_tensor)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
```

Listing 4: part_f.py