

The background is a complex digital landscape. It features a dense network of glowing orange and blue lines that resemble circuit traces or data paths. In the center, there's a prominent structure of glowing yellow and orange arcs, possibly representing a stylized 'AI' or a data flow. The overall color palette is dominated by warm oranges and reds, contrasted with cool blues and greens. The lighting is soft and ethereal, creating a sense of depth and movement.

Security in AI

general formatting for slides

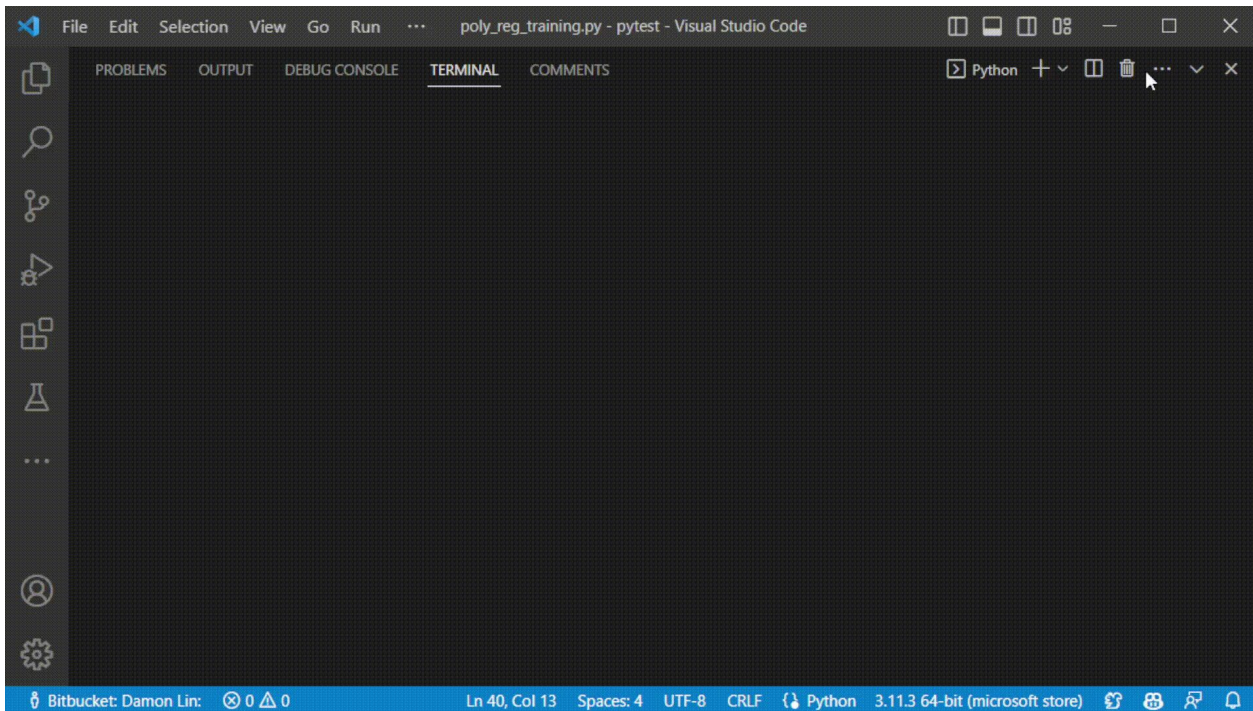
- Direction of the project/research changed slightly
 - instead of targeting voice assistance systems, it will be WiFi-enabled IoT devices
- What we worked on this week:
 - Familiarizing ourselves with PyTorch
 - Reading paper (check Slack for this if haven't seen already)
 - (Maybe?) exploration/research into attack mitigation

Exploring PyTorch

```
1 import torch
2 import torch.nn as nn
3 import numpy as np
4
5 def data_generator(data_size=1000):
6     inputs = []
7     labels = []
8
9     # loop data_size times to generate the data
10    for i in range(data_size):
11        # generate a random number between 0 and 1000
12        x = np.random.randint(2000) / 1000
13
14        # calculate the y value using the function
15        y = x**3 + 2*(x**2) - 2*x - 3 # change
16
17        # append the values to our input and labels lists
18        inputs.append([x**3, 2*x*x, -2*x]) # change
19        labels.append([y])
20
21    return inputs, labels
22
23 # Cubic regression model
24 class PolyRegression(nn.Module):
25
26     # init method
27     def __init__(self):
28         super(PolyRegression, self).__init__()
29         # define layers
30         self.fc1 = nn.Linear(3, 1) # change
31
32     # forward method
33     def forward(self, x):
34         return self.fc1(x)
35
36 model = PolyRegression()
```

```
38 # Training
39 learning_rate = 0.01
40 n_iters = 2000
41
42 loss = nn.MSELoss()
43 optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
44
45 for epoch in range(n_iters):
46     # generate data
47     X, y = data_generator(data_size=1000)
48     x = torch.tensor(X, dtype=torch.float32)
49     Y = torch.tensor(y, dtype=torch.float32)
50
51     # prediction = forward pass
52     y_pred = model(x)
53
54     # loss
55     l = loss(Y, y_pred)
56
57     # gradients
58     l.backward() # dl/dw
59
60     # update weights
61     optimizer.step()
62
63     # zero gradients
64     optimizer.zero_grad()
65
66     if epoch % 100 == 0:
67         [w, b] = model.parameters()
68         print(f'epoch {epoch + 1}: w = {w[0][0].item():.3f}, loss = {l:.8f}')
69
70 model.eval()
71 test_data = data_generator(1)
72 prediction = model(torch.autograd.Variable(torch.Tensor(test_data[0][0])))
73 print("Input: x = {}".format(test_data[0][0][2]/(-2))) # change based on polynomial
74 print("Prediction: {}".format(prediction.data[0]))
75 print("Expected: {}".format(test_data[1][0][0]))
```

Exploring PyTorch (Cont.)



The screenshot shows the Visual Studio Code interface with a terminal window open. The terminal title is "poly_reg_training.py - pytest - Visual Studio Code". The terminal content is as follows:

```
python poly_reg_training.py
1.329
0.199436902999987793
0.221816288999999994
```

The status bar at the bottom indicates the current file is "poly_reg_training.py" at line 40, column 13, with 4 spaces. The Python environment is "Python 3.11.3 64-bit (microsoft store)".

Results

```
Input: x = 1.329
Prediction: 0.199436902999987793
Expected: 0.221816288999999994
```


IoT Devices

Internet of Things (IoT) Devices

- Connected to internet
- Gather information
- Send and receive data through wireless channels
- **Channel State Information (CSI)** has the communications between devices

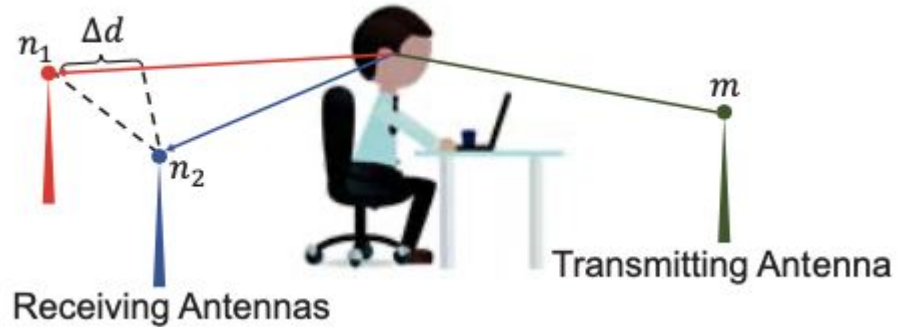


Using IoT Devices for Authentication

IoT Devices can pick up interference from:

- behavioral characteristics
 - body movements

Gives **unique biometric signature** of each user from **behavioral** and **physiological** characteristics



User-authentication can be deployed using deep neural networks on user activity information

- Without devices
- No password/fingerprint/facial recognition

Goals for Next Week

Continue learning PyTorch with more advanced mathematical functions to apply to IoT interference data

Set up experiments to collect interference data from mobile devices