Programming Assignment

Q1

Model Structure

My model has 4 convolution layers decreasing in size gradually from the input number of features to 1. I have a combination of relu and gelu.

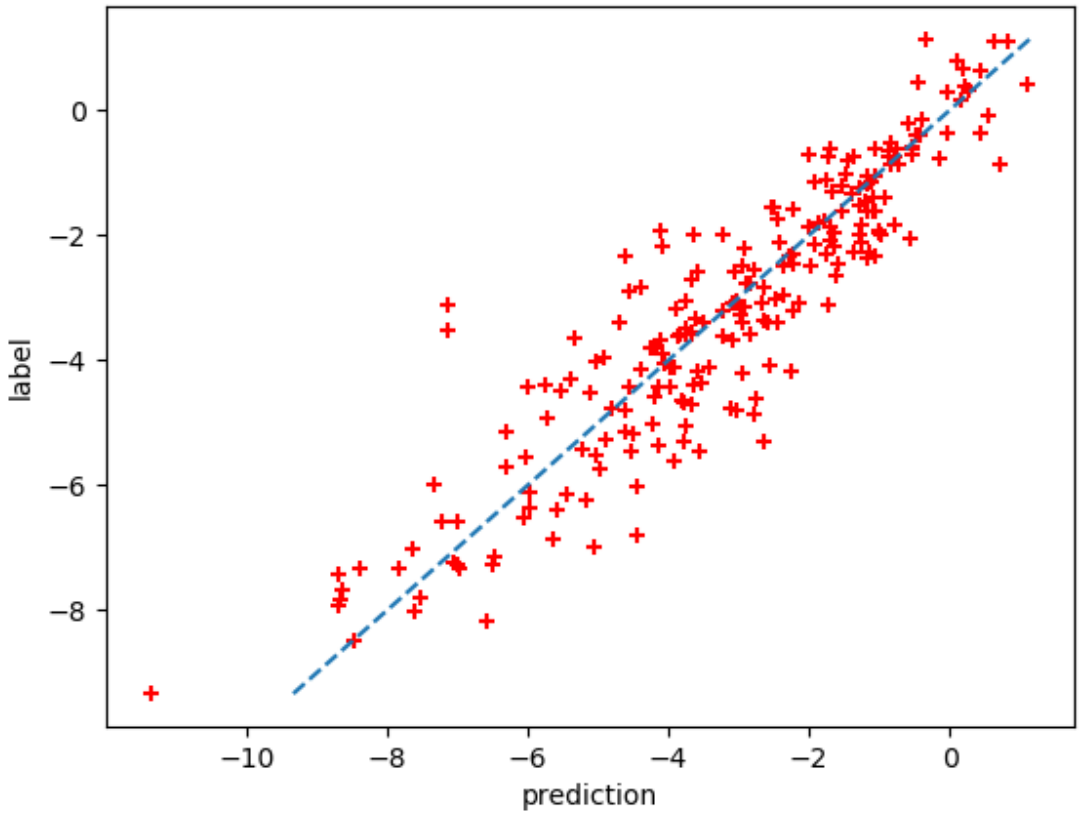
```
def __init__(self, n_features):
    '''
    n_features: number of features from dataset, should be 37
    '''
    super(GraphNet, self).__init__()
    # define your GNN model here
    self.conv1 = GCNConv(n_features,25)
    self.conv2 = GCNConv(25,18)
    self.conv3 = GCNConv(18,12)
    self.conv4 = GCNConv(12,1)
    #self.conv5 = GCNConv(5,1)
    #raise NotImplementedError

def forward(self, data):
    # define the forward pass here
    x, edge_index, edge_weight = data.x, data.edge_index, data.edge_attr
    x = F.gelu(self.conv1(x,edge_index))
    #x = F.dropout(x,training=self.training,p=0.1)
    x = F.relu(self.conv2(x,edge_index))
    #x = F.dropout(x,training=self.training,p=0.2)
    x = F.gelu(self.conv3(x,edge_index))
    #x = F.dropout(x,training=self.training,p=0.2)
    #x = F.relu(self.conv4(x,edge_index))
    x = self.conv4(x,edge_index)
    #x = F.log_softmax(x,dim=0)
    return scatter_mean(torch.squeeze(x),data.batch)
    #raise NotImplementedError
```

Hyperparameters

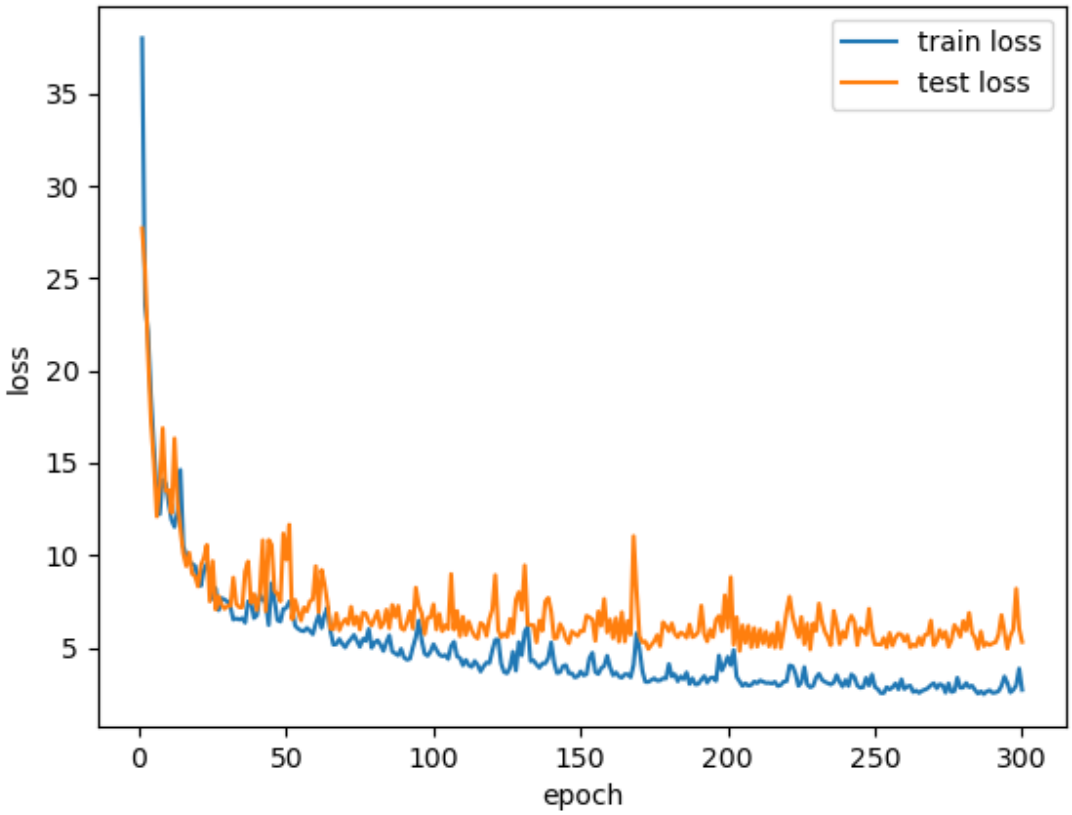
After multiple iterations with the hyperparameters by changing the optimizer, scheduler, lr and the number of epochs I found the following combination of parameters that worked well for my case
Optimizer: RMSprop
Loss: MSE Loss

Lr: 3e-3
Number of Epochs: 300
Scheduler: CosineAnnealingLR with eta_min =1e-3
I tried Adam, and SGD with momentum and weight decay and also varying number of epochs and learning rates but they did not work well.

Performance on test set

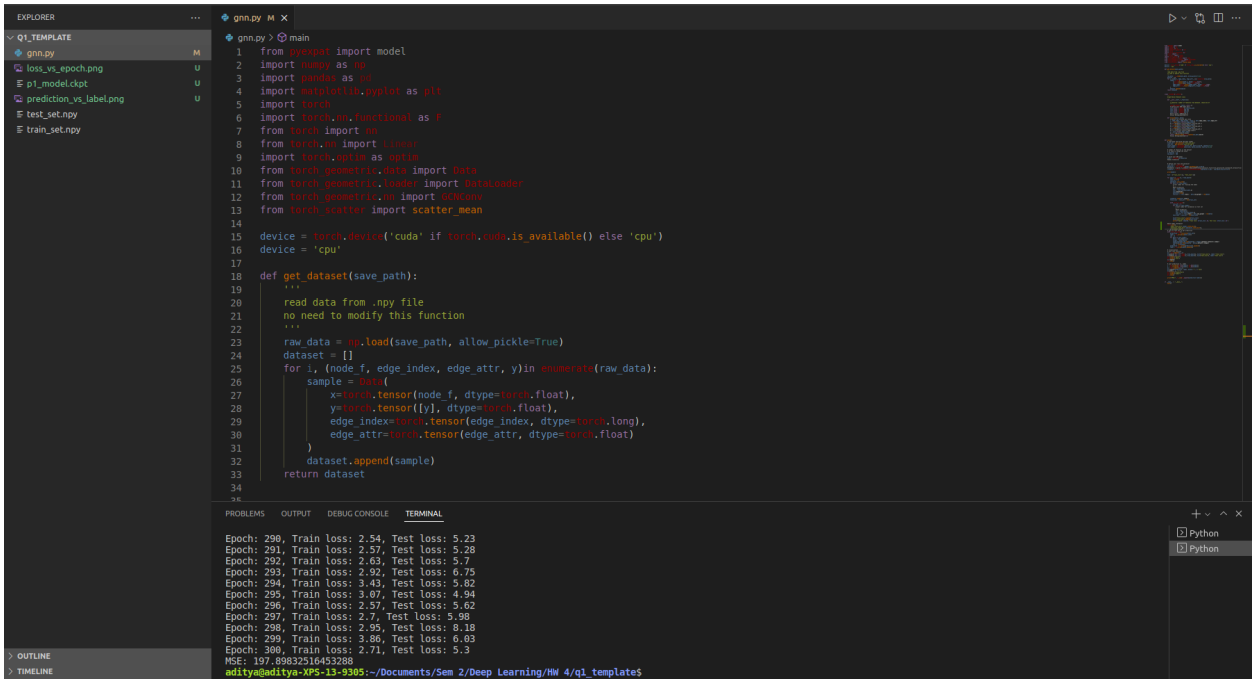


Training Process



MSE Loss

The final MSE loss that I got on the test set was 197.89



Q2

Model Structure

My model consists of 4 linear layers that first increase in size and then decrease in size to the output size. Relu activation functions were used in the intermediary layers.

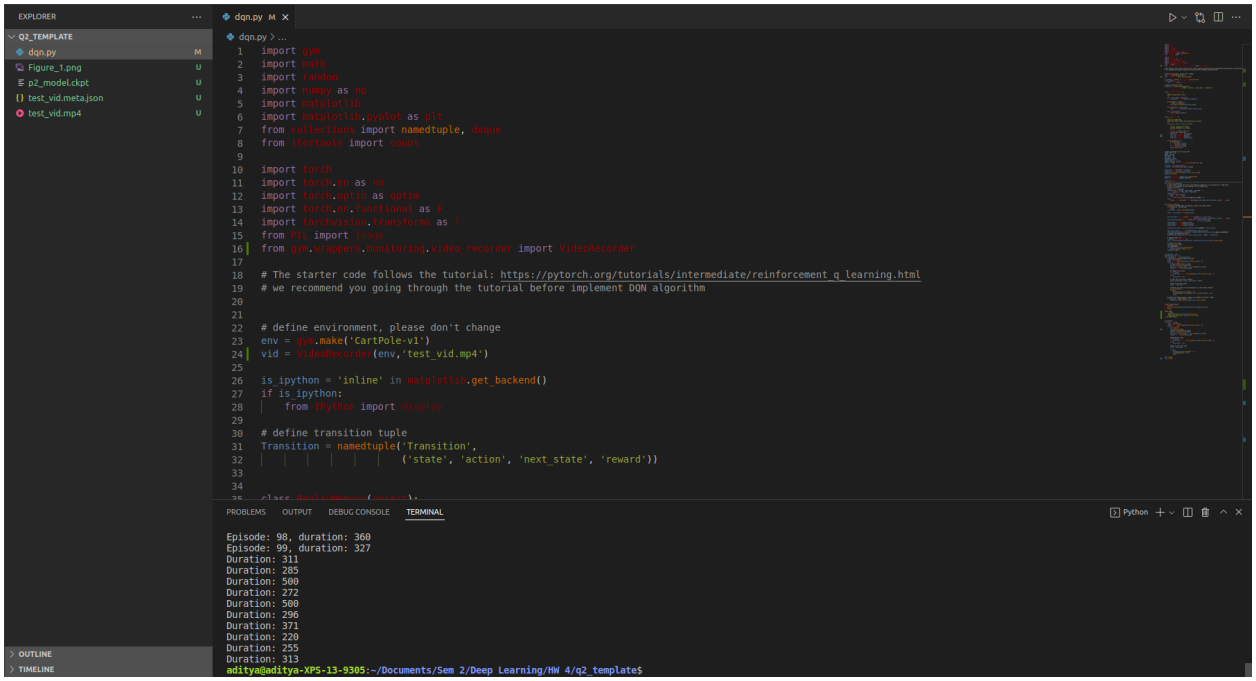
```python
class DQN(nn.Module):
    """
    build your DQN model:
    given the state, output the possiblity of actions
    """
    def __init__(self, in_dim, out_dim):
        """
        in_dim: dimension of states
        out_dim: dimension of actions
        """
        super(DQN, self).__init__()
        # build your model here
        self.fc1 = nn.Linear(in_dim,64)
        self.fc2 = nn.Linear(64,256)
        self.fc3 = nn.Linear(256,64)
        self.fc4 = nn.Linear(64,out_dim)

    def forward(self, x):
        # forward pass
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        return self.fc4(x)
```

Hyperparameters

The hyperparameters used for the DQN model were as follows
Batch size = 128
Gamma = 0.999
Eps start = 0.9
Eps end = 0.05
Eps decay = 2000
Target update = 10
Memory capacity = 10000
Number of train episodes = 100
Optimizer = Adam

Performance on test episodes

The model performed well on the test episodes and the mean duration of the 10 episodes was 332.3



Training Process