Chunyuan Shen

ID:801322013

Homework 5

https://github.com/cryyin/ECGR5105/tree/main/homework5

Problem 1 (20 pts):

In our temperature prediction example, let's change our model to a non-linear system. Consider the following description for our model:

w2 * t_u ** 2 + w1 * t_u + b.

1.a Modify the training loop properly to accommodate this redefinition.

```
[3]: def model(t_u, w1, w2, b):
         return w2*t_u**2 + w1*t_u + b
```

```
def training_loop(n_epochs, optimizer, params, train_t_u, train_t_c, val_t_u, val_t_c):
    for epoch in range(1, n_epochs + 1):
        train_t_p = model(train_t_u, *params)
        train_loss = loss_fn(train_t_p, train_t_c)

        val_t_p = model(val_t_u, *params)
        val_loss = loss_fn(val_t_p, val_t_c)
        val_loss_list.append(val_loss.item())

        optimizer.zero_grad()
        train_loss.backward()
        optimizer.step()
        train_loss_list.append(train_loss.item())

        if epoch <= 1 or epoch % 500 ==0:
            #epoch_list.append(epoch)
            print(f"Epoch {epoch}, Training loss {train_loss.item():.4f}, "f"Validation loss {val_loss.item():.4f}")
```

Modify the model and training.

1.b Use 5000 epochs for your training. Explore different learning rates from 0.1 to 0.0001 (you need four separate trainings). Report your loss for every 500 epochs per training.

Lr=0.1:

```
params = torch.tensor([1.0,1.0,0.0], requires_grad=True)
learning_rate = 0.1
optimizer = optim.Adam([params], lr = learning_rate)

training_loop(
    n_epochs=5000,
    optimizer = optimizer,
    params = params,
    train_t_u = train_t_un,
    train_t_c = train_t_c,
    val_t_u = val_t_un,
    val_t_c = val_t_c)
```

```
Epoch 1, Training loss 715.8550, Validation loss 495.5213
Epoch 500, Training loss 2.6680, Validation loss 5.1354
Epoch 1000, Training loss 2.5798, Validation loss 4.3309
Epoch 1500, Training loss 2.4783, Validation loss 3.4408
Epoch 2000, Training loss 2.3819, Validation loss 2.6476
Epoch 2500, Training loss 2.3044, Validation loss 2.0809
Epoch 3000, Training loss 2.2523, Validation loss 1.7851
Epoch 3500, Training loss 2.2241, Validation loss 1.7127
Epoch 4000, Training loss 2.2123, Validation loss 1.7610
Epoch 4500, Training loss 2.2087, Validation loss 1.8332
Epoch 5000, Training loss 2.2080, Validation loss 1.8812
tensor([  2.9666,   0.2432, -11.8646], requires_grad=True)
```

Lr=0.01:

```
params = torch.tensor([1.0,1.0,0.0], requires_grad=True)
learning_rate = 0.01
optimizer = optim.Adam([params], lr = learning_rate)

training_loop(
    n_epochs=5000,
    optimizer = optimizer,
    params = params,
    train_t_u = train_t_un,
    train_t_c = train_t_c,
    val_t_u = val_t_un,
    val_t_c = val_t_c)
```

```
Epoch 1, Training loss 519.7859, Validation loss 1377.8323
Epoch 500, Training loss 6.4291, Validation loss 4.5225
Epoch 1000, Training loss 3.5718, Validation loss 3.1108
Epoch 1500, Training loss 2.2457, Validation loss 6.7859
Epoch 2000, Training loss 1.8943, Validation loss 10.9993
Epoch 2500, Training loss 1.8424, Validation loss 13.2007
Epoch 3000, Training loss 1.8371, Validation loss 13.7966
Epoch 3500, Training loss 1.8348, Validation loss 13.7756
Epoch 4000, Training loss 1.8320, Validation loss 13.5993
Epoch 4500, Training loss 1.8289, Validation loss 13.3682
Epoch 5000, Training loss 1.8252, Validation loss 13.0897
tensor([-0.6664,  0.6324, -4.4482], requires_grad=True)
```

Lr=0.001:

```
params = torch.tensor([1.0,1.0,0.0], requires_grad=True)
learning_rate = 0.001
optimizer = optim.Adam([params], lr = learning_rate)

training_loop(
    n_epochs=5000,
    optimizer = optimizer,
    params = params,
    train_t_u = train_t_un,
    train_t_c = train_t_c,
    val_t_u = val_t_un,
    val_t_c = val_t_c)
```

```
Epoch 1, Training loss 519.7859, Validation loss 1377.8323
Epoch 500, Training loss 87.5958, Validation loss 162.9495
Epoch 1000, Training loss 13.4346, Validation loss 0.9132
Epoch 1500, Training loss 8.7090, Validation loss 5.7738
Epoch 2000, Training loss 8.2820, Validation loss 7.3488
Epoch 2500, Training loss 7.8330, Validation loss 6.6663
Epoch 3000, Training loss 7.2956, Validation loss 5.7663
Epoch 3500, Training loss 6.6758, Validation loss 4.8303
Epoch 4000, Training loss 5.9894, Validation loss 3.9517
Epoch 4500, Training loss 5.2624, Validation loss 3.2530
Epoch 5000, Training loss 4.5309, Validation loss 2.8763

tensor([-0.1546,  0.4611, -1.9273], requires_grad=True)
```

Lr=0.0001:

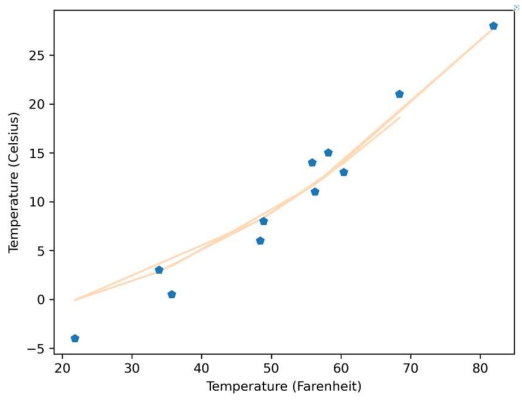
```
params = torch.tensor([1.0,1.0,0.0], requires_grad=True)
learning_rate = 0.0001
optimizer = optim.Adam([params], lr = learning_rate)

training_loop(
    n_epochs=5000,
    optimizer = optimizer,
    params = params,
    train_t_u = train_t_un,
    train_t_c = train_t_c,
    val_t_u = val_t_un,
    val_t_c = val_t_c)
```

```
Epoch 1, Training loss 519.7859, Validation loss 1377.8323
Epoch 500, Training loss 447.2988, Validation loss 1167.2705
Epoch 1000, Training loss 382.3910, Validation loss 980.0179
Epoch 1500, Training loss 324.4459, Validation loss 814.1627
Epoch 2000, Training loss 272.8312, Validation loss 667.7673
Epoch 2500, Training loss 227.0313, Validation loss 539.2454
Epoch 3000, Training loss 186.6240, Validation loss 427.2912
Epoch 3500, Training loss 151.2583, Validation loss 330.8061
Epoch 4000, Training loss 120.6321, Validation loss 248.8293
Epoch 4500, Training loss 94.4727, Validation loss 180.4769
Epoch 5000, Training loss 72.5197, Validation loss 124.8842
tensor([ 0.5668,  0.5693, -0.4369], requires_grad=True)
```

1.c Pick the best non-linear model and compare your final best loss against the linear model that we did during the lecture. For this, visualize the non-linear model against the linear model over the input dataset, as we did during the lecture. Is the actual result better or worse than our baseline linear model?

The best one is learning_rate = 0.001. The loss of it is only 2.8763



The non-linear model is better because it has a smaller loss.

Problem 2 (30 pts):

2.a. Develop preprocessing and a training loop to train a linear regression model that predicts housing price based on the following input variables:

area, bedrooms, bathrooms, stories, parking

For this, you need to use the housing dataset. For training and validation use 80% (training) and 20% (validation) split. Identify the best parameters for your linear regression model, based on the above input variables. In this case, you will have six parameters:

U=W5*X5 + W4*X4 + W3*X3 + W2*X2 + W1*X1 + B

input variables:

```
num_vars = ['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'parking']
df_new = housing[num_vars]
```

```
scaler = MinMaxScaler()
df_new[num_vars] = scaler.fit_transform(df_new[num_vars])
df_new.head()
```

Split the dataset:

```
n_samples = X.shape[0]
n_val = int(0.2 * n_samples)
```

New model:

```
def model(X, W1, W2, W3, W4, W5, B):
    return W5*X[:,4] + W4*X[:,3] + W3*X[:,2] + W2*X[:,1] + W1*X[:,0] + B
```

2.b Use 5000 epochs for your training. Explore different learning rates from 0.1 to 0.0001 (you need four separate trainings). Report your loss and validation accuracy for every 500 epochs per each training. Pick the best linear model.

Lr=0.1:

```
params = torch.tensor([1.0, 1.0, 1.0, 1.0, 1.0, 0.0], requires_grad=True)
learning_rate = 0.1
optimizer = optim.Adam([params], lr=learning_rate)

training_loop(
    n_epochs = 5000,
    optimizer = optimizer,
    params = params,
    train_X = train_X,
    val_X = val_X,
    train_Y = train_Y,
    val_Y = val_Y)
```

```
Epoch 500, Training loss 0.0118, Validation loss 0.0104
Epoch 1000, Training loss 0.0118, Validation loss 0.0104
Epoch 1500, Training loss 0.0118, Validation loss 0.0104
Epoch 2000, Training loss 0.0118, Validation loss 0.0104
Epoch 2500, Training loss 0.0118, Validation loss 0.0104
Epoch 3000, Training loss 0.0118, Validation loss 0.0104
Epoch 3500, Training loss 0.0118, Validation loss 0.0104
Epoch 4000, Training loss 0.0118, Validation loss 0.0104
Epoch 4500, Training loss 0.0118, Validation loss 0.0104
Epoch 5000, Training loss 0.0118, Validation loss 0.0104
```

```
tensor([0.4426, 0.0927, 0.2979, 0.1320, 0.0960, 0.0339], requires_grad=True)
```

Lr=0.01:

```
params = torch.tensor([1.0, 1.0, 1.0, 1.0, 1.0, 0.0], requires_grad=True)
learning_rate = 0.01
optimizer = optim.Adam([params], lr=learning_rate)

training_loop(
    n_epochs = 5000,
    optimizer = optimizer,
    params = params,
    train_X = train_X,
    val_X = val_X,
    train_Y = train_Y,
    val_Y = val_Y)
```

```
Epoch 500, Training loss 0.0136, Validation loss 0.0144
Epoch 1000, Training loss 0.0120, Validation loss 0.0111
Epoch 1500, Training loss 0.0118, Validation loss 0.0105
Epoch 2000, Training loss 0.0118, Validation loss 0.0104
Epoch 2500, Training loss 0.0118, Validation loss 0.0104
Epoch 3000, Training loss 0.0118, Validation loss 0.0104
Epoch 3500, Training loss 0.0118, Validation loss 0.0104
Epoch 4000, Training loss 0.0118, Validation loss 0.0104
Epoch 4500, Training loss 0.0118, Validation loss 0.0104
Epoch 5000, Training loss 0.0118, Validation loss 0.0104
```

```
tensor([0.4426, 0.0927, 0.2979, 0.1320, 0.0960, 0.0339], requires_grad=True)
```

Lr=0.001:

```
params = torch.tensor([1.0, 1.0, 1.0, 1.0, 1.0, 0.0], requires_grad=True)
learning_rate = 0.001
optimizer = optim.Adam([params], lr=learning_rate)

training_loop(
    n_epochs = 5000,
    optimizer = optimizer,
    params = params,
    train_X = train_X,
    val_X = val_X,
    train_Y = train_Y,
    val_Y = val_Y)
```

```
Epoch 500, Training loss 0.1296, Validation loss 0.1134
Epoch 1000, Training loss 0.0519, Validation loss 0.0480
Epoch 1500, Training loss 0.0343, Validation loss 0.0332
Epoch 2000, Training loss 0.0230, Validation loss 0.0235
Epoch 2500, Training loss 0.0172, Validation loss 0.0184
Epoch 3000, Training loss 0.0148, Validation loss 0.0159
Epoch 3500, Training loss 0.0137, Validation loss 0.0146
Epoch 4000, Training loss 0.0130, Validation loss 0.0135
Epoch 4500, Training loss 0.0126, Validation loss 0.0125
Epoch 5000, Training loss 0.0122, Validation loss 0.0117
```

```
tensor([ 0.4557,  0.2596,  0.2528,  0.1079,  0.0900, -0.0241],
       requires_grad=True)
```

Lr=0.0001:

```
params = torch.tensor([1.0, 1.0, 1.0, 1.0, 1.0, 0.0], requires_grad=True)
learning_rate = 0.0001
optimizer = optim.Adam([params], lr=learning_rate)

training_loop(
    n_epochs = 5000,
    optimizer = optimizer,
    params = params,
    train_X = train_X,
    val_X = val_X,
    train_Y = train_Y,
    val_Y = val_Y)
```

```
Epoch 500, Training loss 1.0156, Validation loss 0.9287
Epoch 1000, Training loss 0.8172, Validation loss 0.7443
Epoch 1500, Training loss 0.6499, Validation loss 0.5892
Epoch 2000, Training loss 0.5103, Validation loss 0.4601
Epoch 2500, Training loss 0.3951, Validation loss 0.3541
Epoch 3000, Training loss 0.3019, Validation loss 0.2687
Epoch 3500, Training loss 0.2283, Validation loss 0.2017
Epoch 4000, Training loss 0.1719, Validation loss 0.1509
Epoch 4500, Training loss 0.1305, Validation loss 0.1141
Epoch 5000, Training loss 0.1016, Validation loss 0.0888
tensor([ 0.6147,  0.6150,  0.5916,  0.5953,  0.5937, -0.3704],
       requires_grad=True)
```

The best one is 0.1 and 0.01, they have the minimized loss.

Problem 3 (50 pts):

3.a Build a fully connected neural network for the housing dataset you did in previous problem. For training and validation use 80% (training) and 20% (validation) split. For this part, only use one hidden layer with 8 nodes. Train your network for 200 epochs. Report your training time, training loss, and evaluation accuracy after 200 epochs. Analyze your results in your report. Make sure to submit your code by providing the GitHub URL of your course repository for this course. (15pts)

8 nodes in 1 hidden layer:

```
model = nn.Sequential(
            nn.Linear(5,8),
            nn.ReLU(),
            nn.Linear(8,1),
            nn.LogSoftmax(dim=1))
model = model.float()
learning_rate = 0.1
```

Result:

```
print("Accuracy: %f" % (correct / total))
print('Time {:.2f}s'.format(time.time() - training_start_time))
```

```
Epoch: 0, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 20, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 40, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 60, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 80, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 100, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 120, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 140, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 160, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 180, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 200, Training Loss: 0.094215, Validation Loss: 0.095123
Accuracy: 8.000000
Time 0.00s
```

Because my learning rate is 0.1, it goes to the limit immediately. And because I used gpu to training it, time is 0.00s.

3.b Extend your network with two more additional hidden layers, like the example we did in lecture. Train your network for 200 epochs. Report your training time, training loss, and evaluation accuracy after 200 epochs. Analyze your results in your report. Make sure to submit your code by providing the GitHub URL of your course repository for this course. Analyze your results in your report and compare your model size and accuracy over the baseline implementation in Problem1. a. Do you see any over-fitting? Make sure to submit your code by providing the GitHub URL of your course repository for this course. (25pts)

2 more layers:

```
model2 = nn.Sequential(
            nn.Linear(5,16),
            nn.ReLU(),
            nn.Linear(16,8),
            nn.ReLU(),
            nn.Linear(8,1),
            nn.ReLU(),
            nn.LogSoftmax(dim=1))
model = model.float()
learning_rate = 0.1
```

Result:

```
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if epoch==0 or epoch % 20 ==0:
            print("Epoch: %d, Training Loss: %f, Validation Loss: %f" % (ep
        _, predicted = torch.max(out, dim=1)
        total += val_Y.shape[0]
        correct += int((predicted == train_Y).sum())

print("Accuracy: %f" % (correct / total))
print('Time {:.2f}s'.format(time.time() - training_start_time))
```

```
Epoch: 0, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 20, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 40, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 60, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 80, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 100, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 120, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 140, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 160, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 180, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 200, Training Loss: 0.094215, Validation Loss: 0.095123
Accuracy: 8.000000
Time 0.00s
```

Similar to the first one, is really quick for training. Compare with 1a the is a little over fitting. Valid loss is a little bit higher than training loss.