Neural Networks and Deep Learning 2021-22

Menglu Tao

Feb 2022

1 Introduction

1.1 Homework Goals

The home work is divided into two parts, regression and classification task.The goal of regression task is to build a regression model,and try to map the relation between inputs and outputs also to predict the output when given certain inputs.

The goal of classification task is to train a neural network that maps an input image (from fashionMNIST) to one of ten classes.

1.2 Main implementation strategies

In regression task, a fully feed-forward network model is built with 2 hidden layers and using a sigmoid activation function.

In classification task, I used a linear model to do the multi class classfication task also I built a convolutional neural network to perform the task.

2 Method

2.1 Regression task methods

2.1.1 Model Architecture

```python
class Net(nn.Module):
    def __init__(self, Ni, Nh1, Nh2, No):

        super().__init__()
        print('Network initialized')
        self.fc1 = nn.Linear(in_features=Ni, out_features=Nh1)
        self.fc2 = nn.Linear(in_features=Nh1, out_features=Nh2)
        self.out = nn.Linear(in_features=Nh2, out_features=No)
        self.act = nn.Sigmoid()

    def forward(self, x, additional_out=False):
        x = self.act(self.fc1(x))
        x = self.act(self.fc2(x))
        x = self.out(x)
        return x
```

Figure 1: Network Definition

A fully connected feed-forward network with 2 hidden layers is defined. And use a sigmoid activation function.

2.1.2 Hyperparameters

```
torch.manual_seed(0)
# initialize network
Ni = 1
Nh1 = 128
Nh2 = 256
No = 1
net_reg = Net(Ni, Nh1, Nh2, No)

# define the loss function
loss_fn = nn.MSELoss()4
# define the optimizer
optimizer = optim.Adam(net_reg.parameters(), lr=0.002,weight_decay=1e-5)

# Check the divice
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
net_reg.to(device)

# Use Kfold to cross validate
k=5
cv=KFold(n_splits=k,shuffle=True,random_state=42)


#define number of epochs
num_epochs = 600
#define batch size
batch_size=16
```
✓ 0.2s

Figure 2: Hyperparameters Definition

The initialization of neural network is below: Ni = 1 Nh1 = 128 Nh2 = 256 No = 1

Loss function: nn.MSELoss()

Optimatizer: Adam

Learning rate:0.02

Regularization method: L2 Regularization Weight decay equals to 1e-5

Batch size: 256

Number of epochs:100

2.2 Classification task methods

2.2.1 Model Architecture

```python
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(784, 256)
        self.dropout1 = nn.Dropout(p=0.2)
        self.fc2 = nn.Linear(256, 128)
        self.dropout1 = nn.Dropout(p=0.1)
        self.fc3 = nn.Linear(128, 64)
        self.dropout1 = nn.Dropout(p=0.1)
        self.fc4 = nn.Linear(64, 10)
        self.dropout2 = nn.Dropout(p=0.1)

    def forward(self, x):
        # make sure input tensor is flattened
        x = x.view(x.shape[0], -1)

        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.log_softmax(self.fc4(x), dim=1)

        return x
```

Figure 3: Network Definition

This model used 2 convolutional layers to extract features from the images. And also used a fully connected dense layer to classify those features into their respective categories.

2.2.2 Hyperparameters

```python
# Initialize the network
torch.manual_seed(0)

net = Net()
net.to(device)

# Define the loss function
loss_fn = torch.nn.CrossEntropyLoss()

# Define the optimizer
optimizer = optim.Adam(net.parameters(), lr=0.001)

num_epochs = 50
```

Figure 4: Hyperparameters Definition

The initialization of neural is below:
Ni = 1 Nh1 = 128 Nh2 = 256 No = 1

Loss function: CrossEntropyLoss()
Optimatizer: Adam
Learning rate:0.02
Regularization method: L2 Regularization Weight decay equals to 1e-5
Batch size: 256
Number of epochs:100

2.2.3 CNN Implementation

```
class CNN(NN.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.layer1 = NN.Sequential(
            NN.Conv2d(1, 16, kernel_size=5,
            NN.BatchNorm2d(16),
            NN.ReLU()) #16, 28, 28
        self.pool1=NN.MaxPool2d(2) #16, 14,
        self.layer2 = NN.Sequential(
            NN.Conv2d(16, 32, kernel_size=3)
            NN.BatchNorm2d(32),
            NN.ReLU())#32, 12, 12
        self.layer3 = NN.Sequential(
            NN.Conv2d(32, 64, kernel_size=3)
            NN.BatchNorm2d(64),
            NN.ReLU()) #64, 10, 10
        self.pool2=NN.MaxPool2d(2)  #64, 5,
        self.fc = NN.Linear(5*5*64, 10)
    def forward(self, x):
        out = self.layer1(x)
        #print(out.shape)
        out=self.pool1(out)
        #print(out.shape)
        out = self.layer2(out)
        #print(out.shape)
        out=self.layer3(out)
        #print(out.shape)
        out=self.pool2(out)
        #print(out.shape)
        out = out.view(out.size(0), -1)
        #print(out.shape)
        out = self.fc(out)
        return out
```

Figure 5: CNN Model

3 Result

3.1 Regression Result

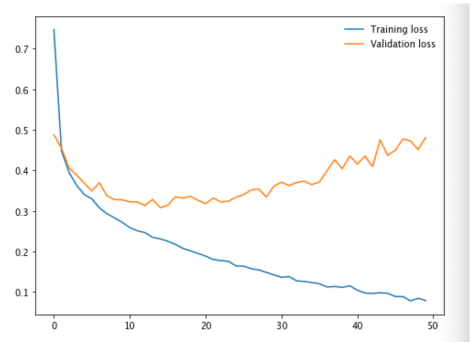

Figure 6: loss for train and validation

The average train loss, validation loss and test loss for regression task is:0.23,0.43 and 0.24.

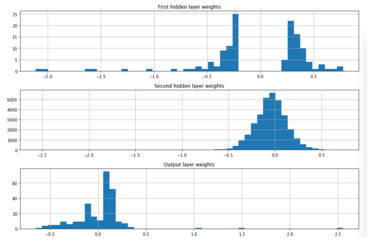

Figure 7: layer Weights

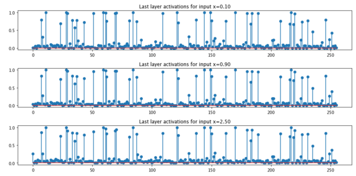

Figure 8: Activation Profile

3.2 Classification Result

```
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = images.reshape(-1, 28*28).to(device)
        labels = labels.to(device)
        outputs = net(images)
        # 统计预测概率最大的下标
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    print('Accuracy of the network on the 10000 test images: {} %'.
✓ 1.5s
Accuracy of the network on the 10000 test images: 89.12 %
```

Figure 9: accuracy for model

The result of classification task is : validation accuracy is 90.42% and test accuracy is 89.88 %.

```
  epoch    train_loss    valid_acc    valid_loss     dur
-------  ------------  -----------  ------------  ------
      1           nan       0.0470           nan  2.0227
      2           nan       0.0021           nan  2.0164
      3           nan       0.0368           nan  2.0862
      4           nan       0.0998           nan  1.9652
      5           nan       0.0998           nan  1.9853
      6           nan       0.0998           nan  1.7520
      7           nan       0.0998           nan  1.7926
      8           nan       0.0998           nan  1.8468
      9           nan       0.0998           nan  1.7975
     10           nan       0.0998           nan  1.7540
Fitting 3 folds for each of 9 candidates, totalling 27 fits
[CV] END ...........................lr=0.001, max_epochs=10; total time=  13.8s
[CV] END ...........................lr=0.001, max_epochs=10; total time=  16.8s
[CV] END ...........................lr=0.001, max_epochs=10; total time=  15.0s
[CV] END ...........................lr=0.001, max_epochs=20; total time=  25.7s
[CV] END ...........................lr=0.001, max_epochs=20; total time=  26.8s
[CV] END ...........................lr=0.001, max_epochs=20; total time=  27.8s
[CV] END ...........................lr=0.001, max_epochs=30; total time=  46.4s
[CV] END ...........................lr=0.001, max_epochs=30; total time=  48.5s
[CV] END ...........................lr=0.001, max_epochs=30; total time=  43.3s
[CV] END ............................lr=0.01, max_epochs=10; total time=  15.7s
[CV] END ............................lr=0.01, max_epochs=10; total time=  14.4s
[CV] END ............................lr=0.01, max_epochs=10; total time=  14.1s
...

[CV] END ............................lr=0.02, max_epochs=20; total time=  25.6s
[CV] END ............................lr=0.02, max_epochs=30; total time=  36.1s
[CV] END ............................lr=0.02, max_epochs=30; total time=  35.4s
[CV] END ............................lr=0.02, max_epochs=30; total time=  35.2s
best score: 0.100, best params: {'lr': 0.001, 'max_epochs': 10}
```

Figure 10: grid search

Through grid search, the best score of the model parameters is 0.1 and best parameters combination is lr: 0.001, maxepochs: 10.