model

October 8, 2024

[1]:
```python
import pandas as pd
import ast
import torch.nn as nn
import torch
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from torch.autograd import Variable
```

0.0.1 Load Data

[2]:
```python
class RedDataset(Dataset):
    def __init__(self, datafile, train):
        self.data = pd.read_csv(datafile)
        self.data = self.data[self.data['train'] == train]
        self.data = self.data.reset_index(drop=True)

    def __len__(self):
        return self.data.shape[0]

    def __getitem__(self, idx):
        item = self.data.loc[idx]
        return torch.tensor(ast.literal_eval(item['data'])), torch.
 ↪tensor(item['label'])

    def shape(self):
        item = self.data.loc[0]
        lst = ast.literal_eval(item['data'])
        return len(lst), len(lst[0])
```

[3]:
```python
filename = 'RedData/RedData_12i.csv'
train_data = RedDataset(filename, train=True)
print(len(train_data))
test_data = RedDataset(filename, train=False)
print(len(test_data))

loaders = {
    'train': DataLoader(train_data,
                        batch_size=100,
```

```
                         shuffle=True,
                         num_workers=1),

    'test': DataLoader(test_data,
                        batch_size=100,
                        shuffle=True,
                        num_workers=1),
}
```

```
60000
10000
```

0.0.2 Model architecture

[4]:
```
class RedModel(nn.Module):
    def __init__(self, input_size):
        super(RedModel, self).__init__()
        layer_size = input_size[0] * input_size[1]
        self.linear1 = nn.Linear(layer_size, layer_size)
        self.relu1 = nn.ReLU()
        self.out = nn.Linear(layer_size, 10)

    def forward(self, x):
        x1 = x.view(x.size(0), -1)
        x2 = self.linear1(x1)
        x3 = self.relu1(x2)
        output = self.out(x3)
        return {
            'in': x,
            'out': output,
            'trans': x1,
            'linear1': x2,
            'relu1': x3,
        }
```

0.0.3 Train model

[5]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f'Device: {device}')

model = RedModel(train_data.shape())
loss_func = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
num_epochs = 10


def train(num_epochs, model, loaders):
```

```python
    model.train()

    # Train the model
    total_step = len(loaders['train'])

    for epoch in range(num_epochs):
        for i, (images, labels) in enumerate(loaders['train']):

            # gives batch data, normalize x when iterate train_loader
            b_x = Variable(images)   # batch x
            b_y = Variable(labels)   # batch output = model(b_x)[0]

            results = model(b_x)['out']
            loss = loss_func(results, b_y)

            # clear gradients for this training step
            optimizer.zero_grad()

            # backpropagation, compute gradients
            loss.backward()                # apply gradients
            optimizer.step()

            if (i + 1) % 100 == 0:
                print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                       .format(epoch + 1, num_epochs, i + 1, total_step, loss.
 ↪item()))
                pass

        pass

    pass


train(num_epochs, model, loaders)
```

```
Device: cpu
Epoch [1/10], Step [100/600], Loss: 0.5688
Epoch [1/10], Step [200/600], Loss: 0.4407
Epoch [1/10], Step [300/600], Loss: 0.5738
Epoch [1/10], Step [400/600], Loss: 0.3984
Epoch [1/10], Step [500/600], Loss: 0.3463
Epoch [1/10], Step [600/600], Loss: 0.1510
Epoch [2/10], Step [100/600], Loss: 0.2273
Epoch [2/10], Step [200/600], Loss: 0.2678
Epoch [2/10], Step [300/600], Loss: 0.1827
Epoch [2/10], Step [400/600], Loss: 0.3214
Epoch [2/10], Step [500/600], Loss: 0.1390
Epoch [2/10], Step [600/600], Loss: 0.2126
```

```
Epoch [3/10], Step [100/600], Loss: 0.2820
Epoch [3/10], Step [200/600], Loss: 0.1484
Epoch [3/10], Step [300/600], Loss: 0.3872
Epoch [3/10], Step [400/600], Loss: 0.1065
Epoch [3/10], Step [500/600], Loss: 0.2004
Epoch [3/10], Step [600/600], Loss: 0.2744
Epoch [4/10], Step [100/600], Loss: 0.1232
Epoch [4/10], Step [200/600], Loss: 0.0957
Epoch [4/10], Step [300/600], Loss: 0.2391
Epoch [4/10], Step [400/600], Loss: 0.2852
Epoch [4/10], Step [500/600], Loss: 0.1151
Epoch [4/10], Step [600/600], Loss: 0.1220
Epoch [5/10], Step [100/600], Loss: 0.2656
Epoch [5/10], Step [200/600], Loss: 0.3062
Epoch [5/10], Step [300/600], Loss: 0.0930
Epoch [5/10], Step [400/600], Loss: 0.1934
Epoch [5/10], Step [500/600], Loss: 0.2202
Epoch [5/10], Step [600/600], Loss: 0.2858
Epoch [6/10], Step [100/600], Loss: 0.2146
Epoch [6/10], Step [200/600], Loss: 0.1377
Epoch [6/10], Step [300/600], Loss: 0.1678
Epoch [6/10], Step [400/600], Loss: 0.0852
Epoch [6/10], Step [500/600], Loss: 0.1638
Epoch [6/10], Step [600/600], Loss: 0.1106
Epoch [7/10], Step [100/600], Loss: 0.1890
Epoch [7/10], Step [200/600], Loss: 0.1168
Epoch [7/10], Step [300/600], Loss: 0.1328
Epoch [7/10], Step [400/600], Loss: 0.2817
Epoch [7/10], Step [500/600], Loss: 0.2194
Epoch [7/10], Step [600/600], Loss: 0.0426
Epoch [8/10], Step [100/600], Loss: 0.1782
Epoch [8/10], Step [200/600], Loss: 0.1196
Epoch [8/10], Step [300/600], Loss: 0.0540
Epoch [8/10], Step [400/600], Loss: 0.0545
Epoch [8/10], Step [500/600], Loss: 0.2172
Epoch [8/10], Step [600/600], Loss: 0.1303
Epoch [9/10], Step [100/600], Loss: 0.1131
Epoch [9/10], Step [200/600], Loss: 0.0067
Epoch [9/10], Step [300/600], Loss: 0.0746
Epoch [9/10], Step [400/600], Loss: 0.0485
Epoch [9/10], Step [500/600], Loss: 0.1696
Epoch [9/10], Step [600/600], Loss: 0.1318
Epoch [10/10], Step [100/600], Loss: 0.0761
Epoch [10/10], Step [200/600], Loss: 0.1825
Epoch [10/10], Step [300/600], Loss: 0.1863
Epoch [10/10], Step [400/600], Loss: 0.1351
Epoch [10/10], Step [500/600], Loss: 0.1005
Epoch [10/10], Step [600/600], Loss: 0.0497
```

[6]:
```
def test():
    model.eval()
    with torch.no_grad():
        for images, labels in loaders['test']:
            results = model(images)
            pred_y = torch.max(results['out'], 1)[1].data.squeeze()
            accuracy = (pred_y == labels).sum().item() / float(labels.size(0))
            pass

    print('Test Accuracy of the model on the 10000 test images: %.2f' %␣
 ↪accuracy)
    pass


test()

torch.save(model.state_dict(), 'model.pt')
```

```
Test Accuracy of the model on the 10000 test images: 0.96
```