model

October 8, 2024

[7]:
```
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

[8]:
```
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])
```

[9]:
```
filename = 'RedData/RedData_16i.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

[10]:
```
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

[11]:
```
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):
```

```
    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: 0.9141
Epoch [1/10], Step [200/600], Loss: 0.3665
Epoch [1/10], Step [300/600], Loss: 0.4117
Epoch [1/10], Step [400/600], Loss: 0.3728
Epoch [1/10], Step [500/600], Loss: 0.2599
Epoch [1/10], Step [600/600], Loss: 0.3261
Epoch [2/10], Step [100/600], Loss: 0.3587
Epoch [2/10], Step [200/600], Loss: 0.4109
Epoch [2/10], Step [300/600], Loss: 0.3684
Epoch [2/10], Step [400/600], Loss: 0.3174
Epoch [2/10], Step [500/600], Loss: 0.1987
Epoch [2/10], Step [600/600], Loss: 0.3453
```

```
Epoch [3/10], Step [100/600], Loss: 0.3464
Epoch [3/10], Step [200/600], Loss: 0.1998
Epoch [3/10], Step [300/600], Loss: 0.5029
Epoch [3/10], Step [400/600], Loss: 0.1824
Epoch [3/10], Step [500/600], Loss: 0.2846
Epoch [3/10], Step [600/600], Loss: 0.2351
Epoch [4/10], Step [100/600], Loss: 0.3485
Epoch [4/10], Step [200/600], Loss: 0.2893
Epoch [4/10], Step [300/600], Loss: 0.3206
Epoch [4/10], Step [400/600], Loss: 0.3246
Epoch [4/10], Step [500/600], Loss: 0.3731
Epoch [4/10], Step [600/600], Loss: 0.5104
Epoch [5/10], Step [100/600], Loss: 0.2172
Epoch [5/10], Step [200/600], Loss: 0.2801
Epoch [5/10], Step [300/600], Loss: 0.1689
Epoch [5/10], Step [400/600], Loss: 0.3751
Epoch [5/10], Step [500/600], Loss: 0.1929
Epoch [5/10], Step [600/600], Loss: 0.2960
Epoch [6/10], Step [100/600], Loss: 0.3066
Epoch [6/10], Step [200/600], Loss: 0.2350
Epoch [6/10], Step [300/600], Loss: 0.2053
Epoch [6/10], Step [400/600], Loss: 0.5057
Epoch [6/10], Step [500/600], Loss: 0.2284
Epoch [6/10], Step [600/600], Loss: 0.2839
Epoch [7/10], Step [100/600], Loss: 0.2488
Epoch [7/10], Step [200/600], Loss: 0.1577
Epoch [7/10], Step [300/600], Loss: 0.4135
Epoch [7/10], Step [400/600], Loss: 0.2964
Epoch [7/10], Step [500/600], Loss: 0.2015
Epoch [7/10], Step [600/600], Loss: 0.2392
Epoch [8/10], Step [100/600], Loss: 0.1867
Epoch [8/10], Step [200/600], Loss: 0.2191
Epoch [8/10], Step [300/600], Loss: 0.4202
Epoch [8/10], Step [400/600], Loss: 0.1202
Epoch [8/10], Step [500/600], Loss: 0.2621
Epoch [8/10], Step [600/600], Loss: 0.3663
Epoch [9/10], Step [100/600], Loss: 0.2231
Epoch [9/10], Step [200/600], Loss: 0.2704
Epoch [9/10], Step [300/600], Loss: 0.2082
Epoch [9/10], Step [400/600], Loss: 0.2182
Epoch [9/10], Step [500/600], Loss: 0.4105
Epoch [9/10], Step [600/600], Loss: 0.2552
Epoch [10/10], Step [100/600], Loss: 0.1242
Epoch [10/10], Step [200/600], Loss: 0.1428
Epoch [10/10], Step [300/600], Loss: 0.2218
Epoch [10/10], Step [400/600], Loss: 0.2103
Epoch [10/10], Step [500/600], Loss: 0.2951
Epoch [10/10], Step [600/600], Loss: 0.2812
```

[12]:
```
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: 0.92
```