model

October 8, 2024

[19]:
```
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

[20]:
```
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])
```

[21]:
```
filename = 'RedData/RedData_24i.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

[22]:
```
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

[23]:
```
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: 1.6823
Epoch [1/10], Step [200/600], Loss: 1.4081
Epoch [1/10], Step [300/600], Loss: 1.4602
Epoch [1/10], Step [400/600], Loss: 1.4940
Epoch [1/10], Step [500/600], Loss: 1.5102
Epoch [1/10], Step [600/600], Loss: 1.4425
Epoch [2/10], Step [100/600], Loss: 1.4979
Epoch [2/10], Step [200/600], Loss: 1.4122
Epoch [2/10], Step [300/600], Loss: 1.3252
Epoch [2/10], Step [400/600], Loss: 1.8645
Epoch [2/10], Step [500/600], Loss: 1.4200
Epoch [2/10], Step [600/600], Loss: 1.2900
```

```
Epoch [3/10], Step [100/600], Loss: 1.3773
Epoch [3/10], Step [200/600], Loss: 1.5749
Epoch [3/10], Step [300/600], Loss: 1.2898
Epoch [3/10], Step [400/600], Loss: 1.3249
Epoch [3/10], Step [500/600], Loss: 1.5266
Epoch [3/10], Step [600/600], Loss: 1.3592
Epoch [4/10], Step [100/600], Loss: 1.1953
Epoch [4/10], Step [200/600], Loss: 1.2373
Epoch [4/10], Step [300/600], Loss: 1.4674
Epoch [4/10], Step [400/600], Loss: 1.3409
Epoch [4/10], Step [500/600], Loss: 1.2824
Epoch [4/10], Step [600/600], Loss: 1.2748
Epoch [5/10], Step [100/600], Loss: 1.4389
Epoch [5/10], Step [200/600], Loss: 1.5253
Epoch [5/10], Step [300/600], Loss: 1.5801
Epoch [5/10], Step [400/600], Loss: 1.3443
Epoch [5/10], Step [500/600], Loss: 1.4288
Epoch [5/10], Step [600/600], Loss: 1.4178
Epoch [6/10], Step [100/600], Loss: 1.2304
Epoch [6/10], Step [200/600], Loss: 1.3950
Epoch [6/10], Step [300/600], Loss: 1.2856
Epoch [6/10], Step [400/600], Loss: 1.3623
Epoch [6/10], Step [500/600], Loss: 1.2211
Epoch [6/10], Step [600/600], Loss: 1.3658
Epoch [7/10], Step [100/600], Loss: 1.2175
Epoch [7/10], Step [200/600], Loss: 1.1163
Epoch [7/10], Step [300/600], Loss: 1.5198
Epoch [7/10], Step [400/600], Loss: 1.2682
Epoch [7/10], Step [500/600], Loss: 1.3537
Epoch [7/10], Step [600/600], Loss: 1.2805
Epoch [8/10], Step [100/600], Loss: 1.3451
Epoch [8/10], Step [200/600], Loss: 1.3495
Epoch [8/10], Step [300/600], Loss: 1.2726
Epoch [8/10], Step [400/600], Loss: 1.1753
Epoch [8/10], Step [500/600], Loss: 1.2056
Epoch [8/10], Step [600/600], Loss: 1.4644
Epoch [9/10], Step [100/600], Loss: 1.2436
Epoch [9/10], Step [200/600], Loss: 1.2861
Epoch [9/10], Step [300/600], Loss: 1.3147
Epoch [9/10], Step [400/600], Loss: 1.3217
Epoch [9/10], Step [500/600], Loss: 1.2135
Epoch [9/10], Step [600/600], Loss: 1.3119
Epoch [10/10], Step [100/600], Loss: 1.2996
Epoch [10/10], Step [200/600], Loss: 1.4282
Epoch [10/10], Step [300/600], Loss: 1.3881
Epoch [10/10], Step [400/600], Loss: 1.2824
Epoch [10/10], Step [500/600], Loss: 1.3232
Epoch [10/10], Step [600/600], Loss: 1.4004
```

[24]:
```
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: 0.55
```