# torch

October 15, 2024

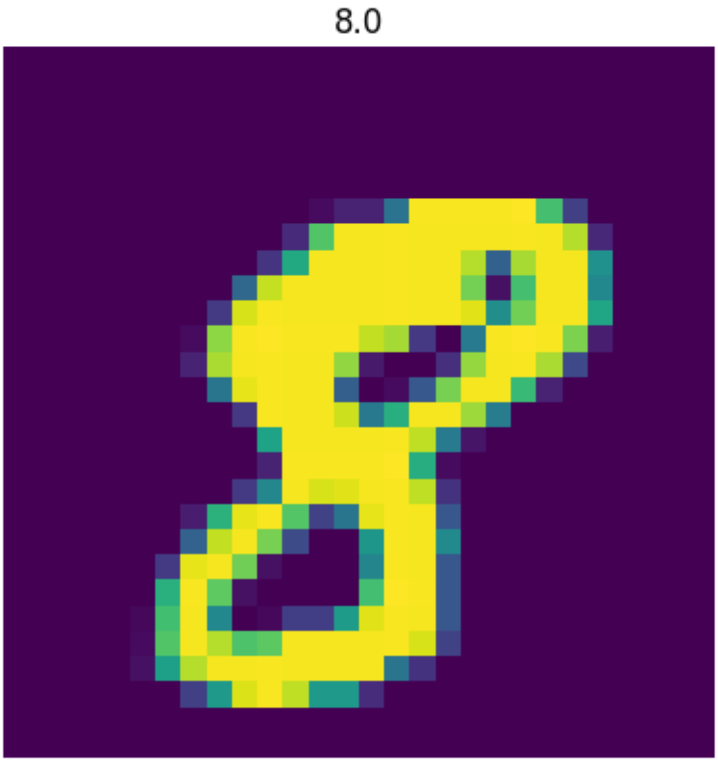
[12]:
```
#
import numpy as np  #
import pandas as pd  #       CSV    /
import matplotlib.pyplot as plt  #
import os  #
print(os.listdir("./input"))  #    ./input
```

```
['test.csv', 'train.csv', 'sample_submission.csv']
```

[13]:
```
#   PyTorch
import torch  # PyTorch
import torch.nn as nn  #
from torch.autograd import Variable  #
from sklearn.model_selection import train_test_split  #
from torch.utils.data import DataLoader, TensorDataset  #
```

[14]:
```
#
#   CSV          float32
train = pd.read_csv(r"./input/train.csv", dtype=np.float32)

#          0-9
targets_numpy = train.label.values  #    0-9
features_numpy = train.loc[:, train.columns != "label"].values / 255  #     ␣
 ↪0-1

#        80%    20%
features_train, features_test, targets_train, targets_test = train_test_split(
    features_numpy, targets_numpy, test_size=0.2, random_state=42)

#         PyTorch   Tensor
featuresTrain = torch.from_numpy(features_train)  #
targetsTrain = torch.from_numpy(targets_train).type(torch.LongTensor)  #      ␣
 ↪long

#         PyTorch   Tensor
featuresTest = torch.from_numpy(features_test)  #
```

```
targetsTest = torch.from_numpy(targets_test).type(torch.LongTensor)  #
 ↪long

#   batch_size epoch
batch_size = 100  #
n_iters = 10000  #
num_epochs = n_iters / (len(features_train) / batch_size)  #   epoch
num_epochs = int(num_epochs)

#   PyTorch
train = TensorDataset(featuresTrain, targetsTrain)  #
test = TensorDataset(featuresTest, targetsTest)  #

#   DataLoader
train_loader = DataLoader(train, batch_size=batch_size, shuffle=False)  #
test_loader = DataLoader(test, batch_size=batch_size, shuffle=False)  #

#
plt.imshow(features_numpy[10].reshape(28, 28))  #   10     28x28
plt.axis("off")  #
plt.title(str(targets_numpy[10]))  #
plt.savefig('graph.png')  #
plt.show()  #
```



[15]:
```
#   RNN
class RNNModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, layer_dim, output_dim):
        super(RNNModel, self).__init__()

        #
        self.hidden_dim = hidden_dim

        #
        self.layer_dim = layer_dim

        #   RNN
        self.rnn = nn.RNN(input_dim, hidden_dim, layer_dim, batch_first=True,␣
 ↪nonlinearity='relu')

        #
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):

        #
        h0 = Variable(torch.zeros(self.layer_dim, x.size(0), self.hidden_dim))

        # RNN
        out, hn = self.rnn(x, h0)

        #
        out = self.fc(out[:, -1, :])
        return out

#
input_dim = 28  #         28
hidden_dim = 100  #
layer_dim = 1  #
output_dim = 10  #    0-9

#   RNN
model = RNNModel(input_dim, hidden_dim, layer_dim, output_dim)

#
error = nn.CrossEntropyLoss()

#   SGD
learning_rate = 0.05
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

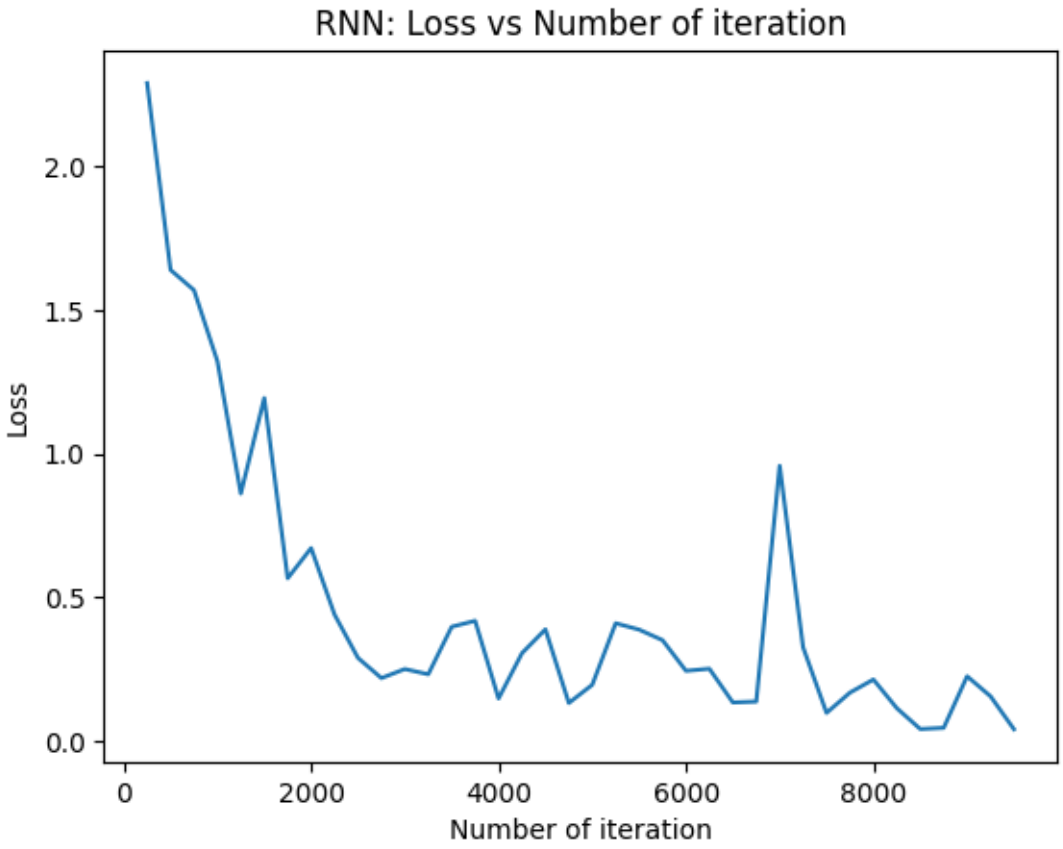
[16]:
```
seq_dim = 28  # RNN    28

#
loss_list = []  #
iteration_list = []  #
accuracy_list = []  #
count = 0  #

#
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):

        #      RNN    (batch_size, seq_dim, input_dim)
        train = Variable(images.view(-1, seq_dim, input_dim))
        labels = Variable(labels)

        #
        optimizer.zero_grad()

        #
        outputs = model(train)

        #
        loss = error(outputs, labels)

        #
        loss.backward()

        #
        optimizer.step()

        count += 1

        #  250
        if count % 250 == 0:
            correct = 0
            total = 0

            #
            for images, labels in test_loader:
                images = Variable(images.view(-1, seq_dim, input_dim))

                #
                outputs = model(images)

                #
                predicted = torch.max(outputs.data, 1)[1]
```

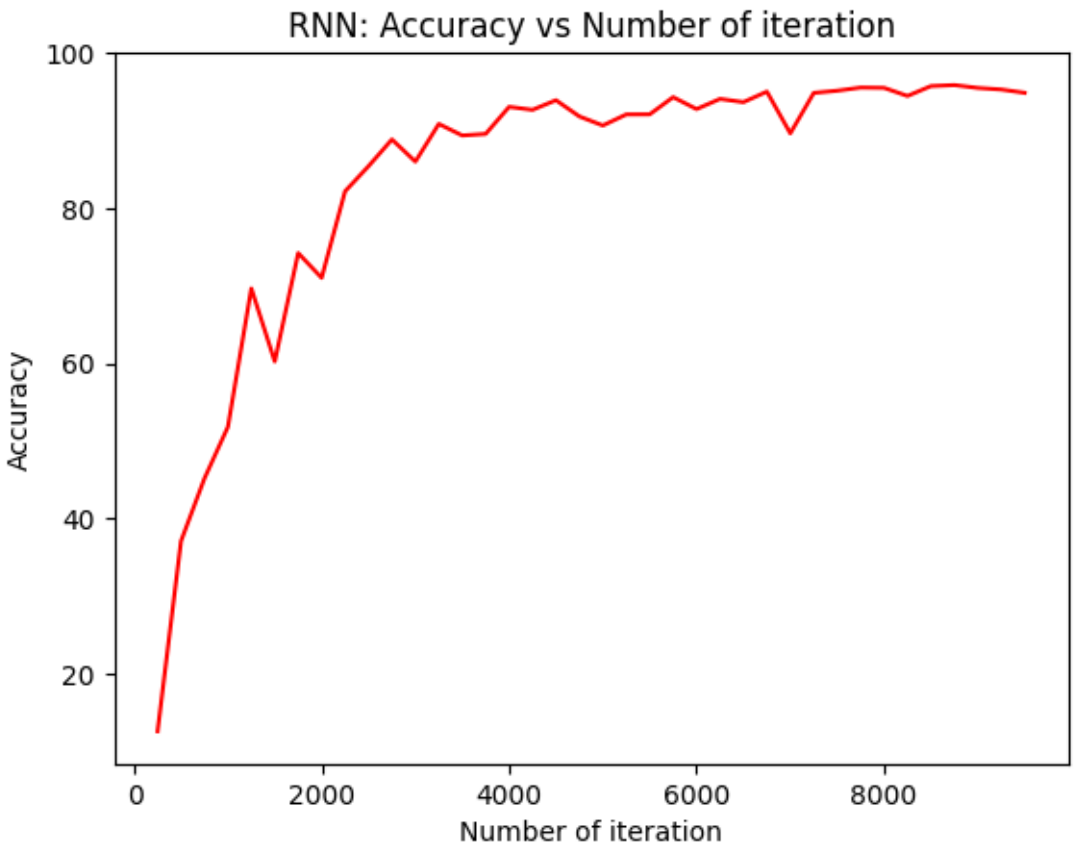
```
                #
                total += labels.size(0)

                #
                correct += (predicted == labels).sum()

            #
            accuracy = 100 * correct / float(total)

            #
            loss_list.append(loss.item())
            iteration_list.append(count)
            accuracy_list.append(accuracy)

            #  500
            if count % 500 == 0:
                print('Iteration: {}  Loss: {}  Accuracy: {} %'.format(count,␣
↪loss.item(), accuracy))
```

```
Iteration: 500  Loss: 1.6385140419006348  Accuracy: 37.10714340209961 %
Iteration: 1000  Loss: 1.3212376832962036  Accuracy: 51.83333206176758 %
Iteration: 1500  Loss: 1.1935008764266968  Accuracy: 60.27381134033203 %
Iteration: 2000  Loss: 0.6718716621398926  Accuracy: 71.05952453613281 %
Iteration: 2500  Loss: 0.29014164209365845  Accuracy: 85.45237731933594 %
Iteration: 3000  Loss: 0.25143879652023315  Accuracy: 86.04762268066406 %
Iteration: 3500  Loss: 0.3983972668647766  Accuracy: 89.41666412353516 %
Iteration: 4000  Loss: 0.14903174340724945  Accuracy: 93.10713958740234 %
Iteration: 4500  Loss: 0.38932570815086365  Accuracy: 93.96428680419922 %
Iteration: 5000  Loss: 0.19674894213676453  Accuracy: 90.70237731933594 %
Iteration: 5500  Loss: 0.3884493112564087  Accuracy: 92.14286041259766 %
Iteration: 6000  Loss: 0.2452794462442398  Accuracy: 92.82142639160156 %
Iteration: 6500  Loss: 0.13537877798080444  Accuracy: 93.71428680419922 %
Iteration: 7000  Loss: 0.9577561020851135  Accuracy: 89.69047546386719 %
Iteration: 7500  Loss: 0.09915570169687271  Accuracy: 95.19047546386719 %
Iteration: 8000  Loss: 0.21516083180904388  Accuracy: 95.55952453613281 %
Iteration: 8500  Loss: 0.042971521615982056  Accuracy: 95.78571319580078 %
Iteration: 9000  Loss: 0.22638843953609467  Accuracy: 95.54762268066406 %
Iteration: 9500  Loss: 0.04231536015868187  Accuracy: 94.91666412353516 %
```

[17]:
```
#
plt.plot(iteration_list, loss_list)
plt.xlabel("Number of iteration")
plt.ylabel("Loss")
plt.title("RNN: Loss vs Number of iteration")
plt.show()
```

```
# %%
#
plt.plot(iteration_list, accuracy_list, color="red")
plt.xlabel("Number of iteration")
plt.ylabel("Accuracy")
plt.title("RNN: Accuracy vs Number of iteration")
plt.savefig('graph.png')
plt.show()
```




RNN: Accuracy vs Number of iteration
100
80
60
40
20
Accuracy
0
2000
4000
6000
8000
Number of iteration