figure_lstm

October 15, 2024

[26]:

```
#
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']
```

[27]:

```
#   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  #
```

[28]:

```
#
#   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
```

[29]:

```
#   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)   #
print(len(features_train))
#
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()   #
```

```
33600
```

[30]:

```
class LSTMModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, layer_dim, output_dim):
        super(LSTMModel, self).__init__()

        #
        self.hidden_dim = hidden_dim

        #   LSTM
        self.layer_dim = layer_dim

        #   LSTM
        self.lstm = nn.LSTM(input_dim, hidden_dim, layer_dim, batch_first=True)

        #
        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)) ␣
 ↪#
        c0 = Variable(torch.zeros(self.layer_dim, x.size(0), self.hidden_dim)) ␣
 ↪#

        # LSTM
        out, (hn, cn) = self.lstm(x, (h0, c0))

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


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

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

#
error = nn.CrossEntropyLoss()
```

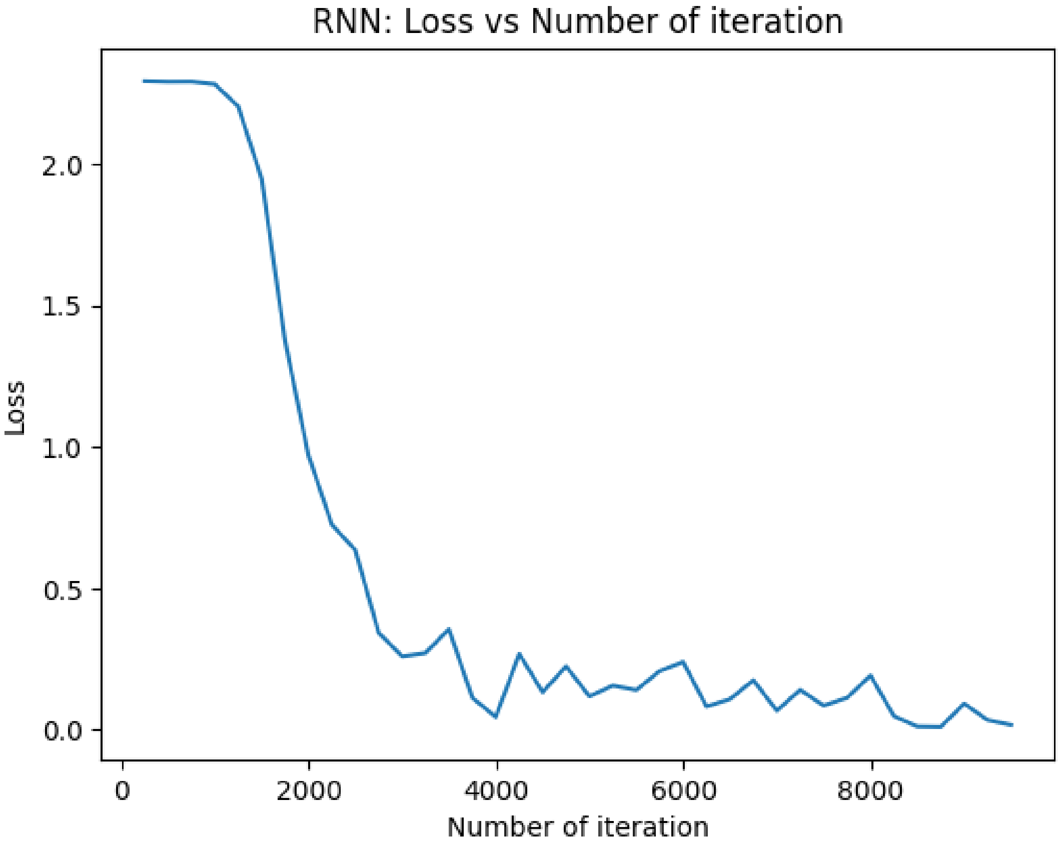
```
#   SGD
learning_rate = 0.05
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

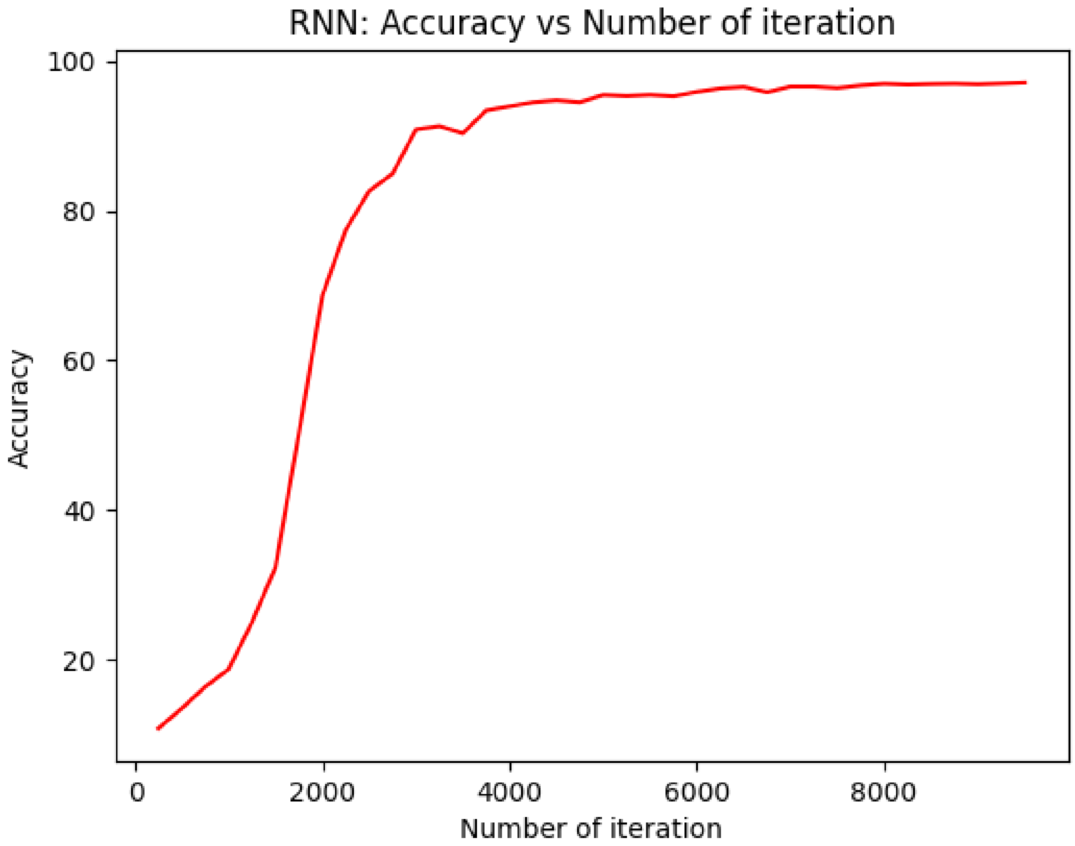
[31]:

```
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: 2.291813850402832  Accuracy: 13.476190567016602 %
Iteration: 1000  Loss: 2.2829360961914062  Accuracy: 18.714284896850586 %
Iteration: 1500  Loss: 1.9491546154022217  Accuracy: 32.345237731933594 %
Iteration: 2000  Loss: 0.9706793427467346  Accuracy: 68.60713958740234 %
Iteration: 2500  Loss: 0.635261595249176  Accuracy: 82.61904907226562 %
Iteration: 3000  Loss: 0.2595059275627136  Accuracy: 90.83333587646484 %
Iteration: 3500  Loss: 0.356059730052948  Accuracy: 90.33333587646484 %
Iteration: 4000  Loss: 0.04518391564488411  Accuracy: 93.92857360839844 %
Iteration: 4500  Loss: 0.13296236097812653  Accuracy: 94.75 %
Iteration: 5000  Loss: 0.11834557354450226  Accuracy: 95.46428680419922 %
Iteration: 5500  Loss: 0.14057557284832  Accuracy: 95.47618865966797 %
Iteration: 6000  Loss: 0.2393888682126999  Accuracy: 95.86904907226562 %
Iteration: 6500  Loss: 0.10809016972780228  Accuracy: 96.53571319580078 %
Iteration: 7000  Loss: 0.0670095682144165  Accuracy: 96.58333587646484 %
Iteration: 7500  Loss: 0.08491396903991699  Accuracy: 96.38095092773438 %
Iteration: 8000  Loss: 0.19161094725131989  Accuracy: 96.96428680419922 %
Iteration: 8500  Loss: 0.011680859141051770  Accuracy: 96.92857360839844 %
Iteration: 9000  Loss: 0.09191018342971802  Accuracy: 96.89286041259766 %
Iteration: 9500  Loss: 0.01728264056146145  Accuracy: 97.11904907226562 %
```

[32]:
```
#
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()
```





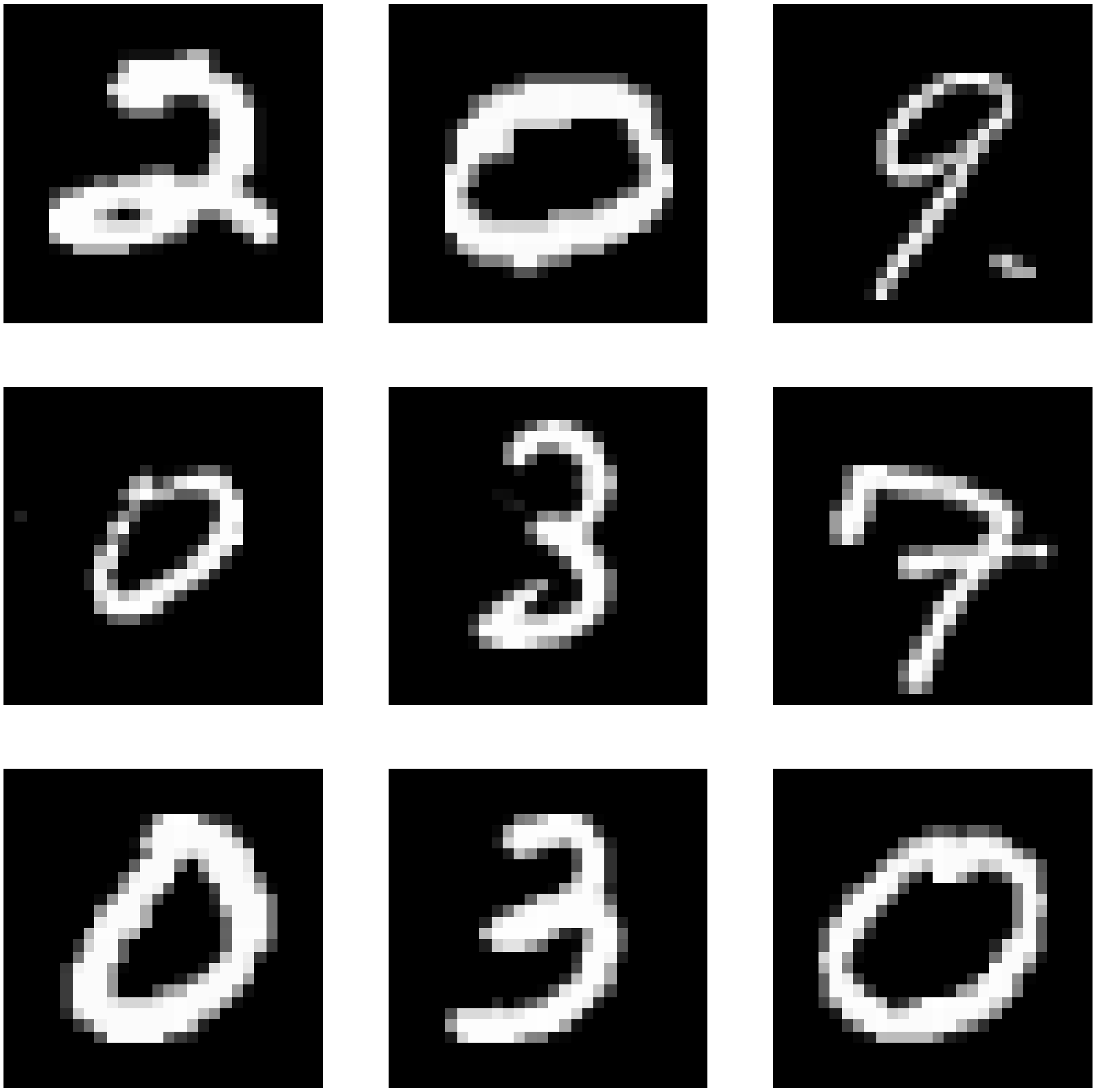
[33]:

```
#
model.eval()

#
with torch.no_grad():
    correct = 0
    total = 0

    #   test_loader
    for images, labels in test_loader:
        images = images.view(-1, 28, 28)  #      (batch_size, seq_dim,␣
 ↪input_dim)

        #
        outputs = model(images)

        #
        _, predicted = torch.max(outputs.data, 1)

        #
        total += labels.size(0)
        correct += (predicted == labels).sum().item()  #  Tensor

    #
    accuracy = 100 * correct / total
    print(f'Test Accuracy: {accuracy} %')
```

```
Test Accuracy: 97.01190476190476 %
```

[36]:

```
#  test.csv
test_data = pd.read_csv(r"./input/test.csv", dtype=np.float32)

test_features_numpy = test_data.values / 255  #     0-255 -> 0-1

#  test     PyTorch   Tensor
test_features = torch.from_numpy(test_features_numpy)

#  DataLoader
test_loader = DataLoader(test_features, batch_size=9, shuffle=False)

#  2x3   6
fig, axes = plt.subplots(3, 3, figsize=(9, 9))

#   6
with torch.no_grad():
    images = next(iter(test_loader))  #  DataLoader
    images = images.view(-1, 28, 28)  #     (batch_size, 28, 28)

    for i, ax in enumerate(axes.flat):
        ax.imshow(images[i], cmap='gray')  #   i
        ax.axis('off')  #

plt.show()  #   6

#
#
model.eval()

#
with torch.no_grad():
    images = Variable(images.view(-1, 28, 28))  #   Variable      (batch_size,␣
 ↪seq_dim, input_dim)

    #
    outputs = model(images)

    #
    _, predicted = torch.max(outputs.data, 1)

# 4.
print("Predicted Labels:", predicted.numpy())  #
```



```
Predicted Labels: [2 0 9 9 3 9 0 3 0]
```