```
import torch
import torch.nn as nn
import numpy as np
from torch.autograd import Variable
import pandas as pd
import matplotlib.pyplot as plt
```

```
is_cuda = torch.cuda.is_available()
if is_cuda:
    device = torch.device("cuda")
    print("GPU is available")
else:
    device = torch.device("cpu")
    print("GPU not available, CPU used")
```

```
GPU is available
```

Data (Input and Ouput)

```
from google.colab import files
uploaded = files.upload()
```

Choose files No file chosen

- **valid.csv**(text/csv) - 70080 bytes, last modified: 26/02/2022 - 100% done
- **test.csv**(text/csv) - 49689 bytes, last modified: 26/02/2022 - 100% done
- **train.csv**(text/csv) - 671057 bytes, last modified: 26/02/2022 - 100% done

```
Saving valid.csv to valid.csv
Saving test.csv to test.csv
Saving train.csv to train.csv
```

```
import io
df_train = pd.read_csv("train.csv", index_col = "Date", parse_dates=True)
df_valid = pd.read_csv("valid.csv", index_col = "Date", parse_dates=True)
df_test = pd.read_csv("test.csv", index_col = "Date", parse_dates=True)
```

```
y_train = df_train.iloc[:,0]
X_train = df_train.iloc[:,1:]

y_test = df_test.iloc[:,0]
X_test = df_test.iloc[:,1:]

y_valid = df_valid.iloc[:,0]
X_valid = df_valid.iloc[:,1:]

print("Training Shape", X_train.shape, y_train.shape)
print("Testing Shape", X_test.shape, y_test.shape)
print("Validate Shape", X_valid.shape, y_valid.shape)
```

```
Training Shape (3239, 28) (3239,)
Testing Shape (225, 28) (225,)
Validate Shape (336, 28) (336,)
```

```
#Remove strings in data
# H: win   0
# NHL lose 1
# remove HM1 to AM5

X_train = X_train.drop(['HM1', 'HM2', 'HM3', 'HM4', 'HM5', 'AM1', 'AM2', 'AM3', 'AM4', 'AM5'], axis=1)
X_test = X_test.drop(['HM1', 'HM2', 'HM3', 'HM4', 'HM5', 'AM1', 'AM2', 'AM3', 'AM4', 'AM5'], axis=1)
X_valid = X_valid.drop(['HM1', 'HM2', 'HM3', 'HM4', 'HM5', 'AM1', 'AM2', 'AM3', 'AM4', 'AM5'], axis=1)

y_train = y_train.replace('H', 1)
y_train = y_train.replace('NH', 0)

y_test = y_test.replace('H', 1)
y_test = y_test.replace('NH', 0)

y_valid = y_valid.replace('H', 1)
y_valid = y_valid.replace('NH', 0)
```

```
X_train_tensors = Variable(torch.Tensor(X_train.values))
X_test_tensors = Variable(torch.Tensor(X_test.values))
X_valid_tensors = Variable(torch.Tensor(X_valid.values))
```

```
4
5  y_train_tensors = Variable(torch.Tensor(y_train.values))
6  y_test_tensors = Variable(torch.Tensor(y_test.values))
7  y_valid_tensors = Variable(torch.Tensor(y_valid.values))
```

```
1 X_train_tensors_final = torch.reshape(X_train_tensors,   (X_train_tensors.shape[0], 1, X_train_tensors.shape[1]))
2 X_test_tensors_final = torch.reshape(X_test_tensors,  (X_test_tensors.shape[0], 1, X_test_tensors.shape[1]))
3
4 y_train_tensors_final = torch.reshape(y_train_tensors, (y_train_tensors.shape[0], 1))
5 y_test_tensors_final = torch.reshape(y_test_tensors, (y_test_tensors.shape[0], 1))
6
7 print(f"X_train_tensors_final:\n{X_train_tensors_final[:3,:]}\n")
```

```
X_train_tensors_final:
tensor([[[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]],

        [[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]],

        [[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]]])
```

```
1 print("Training Shape", X_train_tensors_final.shape, y_train_tensors_final.shape)
2 print("Testing Shape", X_test_tensors_final.shape, y_test_tensors_final.shape)
3
4 print(X_train_tensors_final)
```

```
Training Shape torch.Size([3239, 1, 18]) torch.Size([3239, 1])
Testing Shape torch.Size([225, 1, 18]) torch.Size([225, 1])
tensor([[[ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]],

        [[ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]],

        [[ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]],

        ...,

        [[ 0.2451,  0.3143,  0.3373,  ...,  0.3000, -0.4000, -0.0500]],

        [[ 0.4608,  0.2762,  0.1687,  ...,  0.2500,  1.2500,  0.2500]],

        [[ 0.5098,  0.2190,  0.2771,  ...,  0.3000,  0.4500, -0.0500]]])
```

LSTM Model Architecture

```
1 class rnnLSTM(nn.Module):
2
3   def __init__(self, num_classes, input_size, hidden_size, num_layers, seq_length):
4
5     super(rnnLSTM, self).__init__()
6     self.num_classes = num_classes
7     self.num_layers = num_layers
8     self.input_size = input_size
9     self.hidden_size = hidden_size
10    self.seq_length = seq_length
11
12    self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size, num_layers=num_layers, batch_first=True)
13    self.fc_1 = nn.Linear(hidden_size, 128)
14    self.fc_2 = nn.Linear(128, num_classes)
15    self.relu = nn.ReLU()
16
17  def forward(self, x):
18
19    h_0 = Variable(torch.zeros(self.num_layers, x.size(0), self.hidden_size))
20    c_0 = Variable(torch.zeros(self.num_layers, x.size(0), self.hidden_size))
21
22    output, (hn, cn) = self.lstm(x, (h_0, c_0))
23    hn = hn.view(-1, self.hidden_size) #reshaping data for Dense layer
24    out = self.relu(hn)
25    out = self.fc_1(out)
26    out = self.relu(out)
27    out = self.fc_2(out)


28
```

Initializing LSTM Model

```
num_epochs = 100
learning_rate = 0.001

input_size = 18 #Number of features
hidden_size = 5 #Number of features in Short Term memory (hidden state)
num_layers = 1 #Number of stacked LSTM layers
num_classes = 2 #number of output classes
lstm1 = rnnLSTM(num_classes, input_size, hidden_size, num_layers, X_train_tensors_final.shape[1])
```

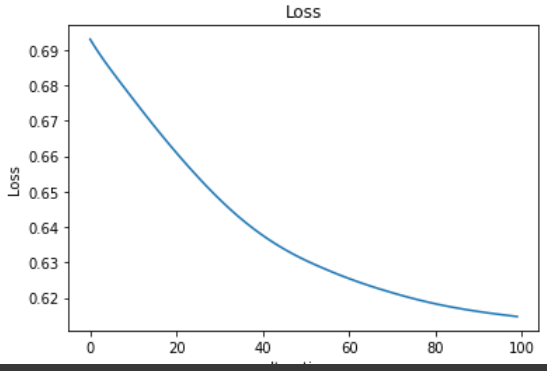
Training (Implement Batches)

```
def get_accuracy(X, y):
  output = lstm1(X)
  y = y.squeeze()
  pred = output.max(1)[1]
  correct = np.count_nonzero(y.eq(output.max(1)[1]))
  total = X.shape[0]
  res = correct/total
  return res

criterion = torch.nn.CrossEntropyLoss() #Mean squared error for regression
optimizer = torch.optim.Adam(lstm1.parameters(), lr=learning_rate)
iters, losses, train_acc, = [], [], []

for epoch in range(num_epochs):
  ouputs = lstm1(X_train_tensors_final) #forward pass
  optimizer.zero_grad() #calculate the gradient, mnually set to 0y_test_tensors_final
  #print(f"Ouput: {ouputs} and Y:{y_train_tensors_final.squeeze().long()}")
  #break
  loss = criterion(ouputs, y_train_tensors_final.squeeze().long()) #obtain loss function
  loss.backward() #compute loss
  optimizer.step() #imporve from loss #backprop
  # print(f"Epoch: {epoch}, loss:{loss.item()}")

  iters.append(epoch)
  losses.append(float(loss))             # compute *average* loss
  train_acc.append(get_accuracy(X_train_tensors_final, y_train_tensors_final)) # compute training accuracy
  print(f"Epoch {epoch+1}: Train acc: {train_acc[epoch]} Losses: {losses[epoch]}")

# plotting
plt.title("Loss")
plt.plot(iters, losses, label="Train")
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.show()

plt.title("Accuracy")
plt.plot(iters, train_acc, label="Train")
plt.xlabel("Iterations")
plt.ylabel("Training Accuracy")
plt.legend(loc='best')
plt.show()

print("Final Training Accuracy: {}".format(train_acc[-1]))
```

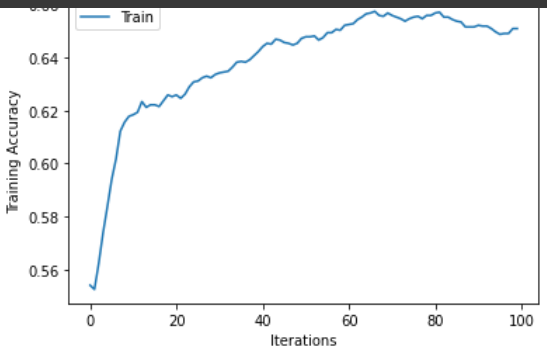
```
Epoch 1: Train acc: 0.5541833899351651 Losses: 0.6930657029151917
Epoch 2: Train acc: 0.552639703612226 Losses: 0.6910938024520874
Epoch 3: Train acc: 0.5625192960790367 Losses: 0.6892340183258057
Epoch 4: Train acc: 0.5739425748687866 Losses: 0.6874601244926453
Epoch 5: Train acc: 0.5838221673355974 Losses: 0.6857529282569885
```

```
Epoch 6: Train acc: 0.5940104970669959 Losses: 0.6840946078300476
Epoch 7: Train acc: 0.6017289286816919 Losses: 0.682469367980957
Epoch 8: Train acc: 0.6122259956776783 Losses: 0.6808638572692871
Epoch 9: Train acc: 0.6156221055881445 Losses: 0.6792696118354797
Epoch 10: Train acc: 0.6177832664402594 Losses: 0.6776821613311768
Epoch 11: Train acc: 0.618400740969435 Losses: 0.6760988831520081
Epoch 12: Train acc: 0.6193269527631985 Losses: 0.674521267414093
Epoch 13: Train acc: 0.6233405372028403 Losses: 0.6729521155357361
Epoch 14: Train acc: 0.6211793763507255 Losses: 0.6713956594467163
Epoch 15: Train acc: 0.622105588144489 Losses: 0.6698541045188904
Epoch 16: Train acc: 0.622105588144489 Losses: 0.668327808380127
Epoch 17: Train acc: 0.6214881136153134 Losses: 0.6668187379837036
Epoch 18: Train acc: 0.6236492744674282 Losses: 0.6653275489807129
Epoch 19: Train acc: 0.6258104353195431 Losses: 0.6638573408126831
Epoch 20: Train acc: 0.6251929607903673 Losses: 0.6624076962471008
Epoch 21: Train acc: 0.6258104353195431 Losses: 0.6609777808189392
Epoch 22: Train acc: 0.6245754862611917 Losses: 0.6595679521560669
Epoch 23: Train acc: 0.6261191725841309 Losses: 0.6581764221191406
Epoch 24: Train acc: 0.6288978079654214 Losses: 0.6568040251731873
Epoch 25: Train acc: 0.6307502315529484 Losses: 0.6554518342018127
Epoch 26: Train acc: 0.6310589688175363 Losses: 0.6541218757629395
Epoch 27: Train acc: 0.6322939178758876 Losses: 0.6528154015541077
Epoch 28: Train acc: 0.6329113924050633 Losses: 0.6515331864356995
Epoch 29: Train acc: 0.6322939178758876 Losses: 0.650277316570282
Epoch 30: Train acc: 0.6335288669342389 Losses: 0.6490477323532104
Epoch 31: Train acc: 0.6341463414634146 Losses: 0.6478464603424072
Epoch 32: Train acc: 0.6344550787280024 Losses: 0.6466755270957947
Epoch 33: Train acc: 0.6347638159925904 Losses: 0.6455363631248474
Epoch 34: Train acc: 0.6363075023155295 Losses: 0.6444306969642639
Epoch 35: Train acc: 0.6381599259030565 Losses: 0.6433570384979248
Epoch 36: Train acc: 0.6384686631676443 Losses: 0.6423163414001465
Epoch 37: Train acc: 0.6381599259030565 Losses: 0.6413128972053528
Epoch 38: Train acc: 0.63908613769682 Losses: 0.6403443813323975
Epoch 39: Train acc: 0.6406298240197592 Losses: 0.639409065246582
Epoch 40: Train acc: 0.6421735103426983 Losses: 0.6385066509246826
Epoch 41: Train acc: 0.6440259339302253 Losses: 0.6376357078552246
Epoch 42: Train acc: 0.6452608829885768 Losses: 0.6367978453636169
Epoch 43: Train acc: 0.6449521457239888 Losses: 0.6359930634498596
Epoch 44: Train acc: 0.6468045693115159 Losses: 0.6352167725563049
Epoch 45: Train acc: 0.6464958320469281 Losses: 0.6344698667526245
Epoch 46: Train acc: 0.6455696202531646 Losses: 0.633750319480896
Epoch 47: Train acc: 0.6452608829885768 Losses: 0.6330591440200806
Epoch 48: Train acc: 0.644643408459401 Losses: 0.6323939561843872
Epoch 49: Train acc: 0.6452608829885768 Losses: 0.6317561268806458
Epoch 50: Train acc: 0.6471133065761038 Losses: 0.6311439275741577
Epoch 51: Train acc: 0.6477307811052794 Losses: 0.6305539608001709
Epoch 52: Train acc: 0.6477307811052794 Losses: 0.6299851536750793
Epoch 53: Train acc: 0.6480395183698673 Losses: 0.6294336318969727
Epoch 54: Train acc: 0.6464958320469281 Losses: 0.6288958191871643
Epoch 55: Train acc: 0.6474220438406916 Losses: 0.6283707618713379
Epoch 56: Train acc: 0.6492744674282186 Losses: 0.6278553009033203
Epoch 57: Train acc: 0.6492744674282186 Losses: 0.6273484826087952
Epoch 58: Train acc: 0.6505094164865699 Losses: 0.6268500685691833
Epoch 59: Train acc: 0.6502006792219821 Losses: 0.6263644695281982
Epoch 60: Train acc: 0.6520531028095091 Losses: 0.6258924007415771
Epoch 61: Train acc: 0.6523618400740969 Losses: 0.6254335641860962
Epoch 62: Train acc: 0.6526705773386848 Losses: 0.6249861121177673
Epoch 63: Train acc: 0.6542142636616239 Losses: 0.6245497465133667
Epoch 64: Train acc: 0.6551404754553874 Losses: 0.6241266131401062
Epoch 65: Train acc: 0.6563754245137388 Losses: 0.6237139701843262
Epoch 66: Train acc: 0.6566841617783267 Losses: 0.6233099102973938
Epoch 67: Train acc: 0.6573016363075023 Losses: 0.6229168772697449
Epoch 68: Train acc: 0.6557579499845632 Losses: 0.6225342750549316
Epoch 69: Train acc: 0.6554492127199752 Losses: 0.6221615076065063
Epoch 70: Train acc: 0.6566841617783267 Losses: 0.6217959523200989
Epoch 71: Train acc: 0.6557579499845632 Losses: 0.6214359998703003
Epoch 72: Train acc: 0.6551404754553874 Losses: 0.6210830211639404
Epoch 73: Train acc: 0.6545230009262117 Losses: 0.6207361221313477
Epoch 74: Train acc: 0.6535967891324483 Losses: 0.6203964948654175
Epoch 75: Train acc: 0.6545230009262117 Losses: 0.6200685501098633
Epoch 76: Train acc: 0.6551404754553874 Losses: 0.6197513341903687
Epoch 77: Train acc: 0.6554492127199752 Losses: 0.6194437742233276
Epoch 78: Train acc: 0.6545230009262117 Losses: 0.6191465854644775
Epoch 79: Train acc: 0.6557579499845632 Losses: 0.6188575625419617
Epoch 80: Train acc: 0.6557579499845632 Losses: 0.6185761094093323
Epoch 81: Train acc: 0.6566841617783267 Losses: 0.6183064579963684
Epoch 82: Train acc: 0.6569928990429145 Losses: 0.6180494427680969
Epoch 83: Train acc: 0.6551404754553874 Losses: 0.6178024411201477
Epoch 84: Train acc: 0.6551404754553874 Losses: 0.617563009262085
Epoch 85: Train acc: 0.6542142636616239 Losses: 0.6173350214958191
Epoch 86: Train acc: 0.6535967891324483 Losses: 0.617112934589386
Epoch 87: Train acc: 0.6532880518678604 Losses: 0.6168962717056274
Epoch 88: Train acc: 0.6514356282803334 Losses: 0.6166888475418091
Epoch 89: Train acc: 0.6514356282803334 Losses: 0.6164888739585876
Epoch 90: Train acc: 0.6514356282803334 Losses: 0.6162965893745422
Epoch 91: Train acc: 0.6520531028095091 Losses: 0.6161130666732788
```

```
Epoch 92: Train acc: 0.6517443655449213 Losses: 0.6159335970878601
Epoch 93: Train acc: 0.6517443655449213 Losses: 0.6157597303390503
Epoch 94: Train acc: 0.6508181537511578 Losses: 0.6155950427055359
Epoch 95: Train acc: 0.6495832046928064 Losses: 0.6154362559318542
Epoch 96: Train acc: 0.6486569928990429 Losses: 0.6152824759483337
Epoch 97: Train acc: 0.6489657301636308 Losses: 0.6151324510574341
Epoch 98: Train acc: 0.6489657301636308 Losses: 0.6149847507476807
Epoch 99: Train acc: 0.6508181537511578 Losses: 0.6148388981819153
Epoch 100: Train acc: 0.6508181537511578 Losses: 0.6146944165229797
```



Testing




```
Final Training Accuracy: 0.6508181537511578
```

✓ 1s completed at 4:06 PM