

```
In [1]: import torch
        from torch import nn, optim
        import numpy as np
        from torchvision import datasets, transforms
        import torch.nn.functional as F
        import matplotlib.pyplot as plt
```

```
In [2]: with open("RNN_data/anna.txt", 'r') as f:
        text = f.read()
```

```
In [3]: chars = tuple(set(text))
        chars[:10]
```

```
Out[3]: ('c', 'r', 'u', '&', ',', 'A', 'Y', 't', '3', 'X')
```

```
In [4]: int2char = dict(enumerate(chars))
        char2int = {ch: i for i, ch in int2char.items()}

        encode = np.array([char2int[i] for i in text])
```

```
In [5]: def one_hot_encode(arr, n_labels):

        # Initialize the encoded array
        one_hot = np.zeros((arr.size, n_labels), dtype=np.float32)

        # Fill the appropriate elements with ones
        one_hot[np.arange(one_hot.shape[0]), arr.flatten()] = 1.

        # Finally reshape it to get back to the original array
        one_hot = one_hot.reshape((*arr.shape, n_labels))

        return one_hot
```

```
In [22]: test_seq = np.array([[3, 5, 1],
                             [6, 7, 2]])
        print(test_seq.flatten(), test_seq.size)
        one_hot = one_hot_encode(test_seq, 8)

        print(one_hot)
```

```
[3 5 1 6 7 2] 6
[[[0. 0. 0. 1. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 1. 0. 0.]
  [0. 1. 0. 0. 0. 0. 0. 0.]]

 [[0. 0. 0. 0. 0. 0. 1. 0.]
  [0. 0. 0. 0. 0. 0. 0. 1.]
  [0. 0. 1. 0. 0. 0. 0. 0.]]
```

```
In [7]: def get_batches(arr, batch_size, seq_length):
```

```

n_batches = len(arr)//(batch_size*seq_length)

arr = arr[:n_batches * (batch_size*seq_length)]

arr = arr.reshape((batch_size,-1))

for i in range(0,arr.shape[1],seq_length):
    x = arr[:,i:i+seq_length]
    y = np.zeros_like(x)
    try:
        y[:, :-1], y[:, -1] = x[:, 1:], arr[:, i+seq_length]
    except IndexError:
        y[:, :-1], y[:, -1] = x[:, 1:], arr[:, 0]

    yield x, y

```

```

In [8]: batches = get_batches(encode, 8, 50)
        x, y = next(batches)

```

```

In [9]: print('x\n', x[:10, :10])
        print('\ny\n', y[:10, :10])

```

```

x
[[55 78 65 48  7 72  1 50 35 73]
 [54 22 82 50  7 78 65  7 50 65]
 [72 82 16 50 22  1 50 65 50 44]
 [54 50  7 78 72 50  0 78 40 72]
 [50 54 65 10 50 78 72  1 50  7]
 [ 0 57 54 54 40 22 82 50 65 82]
 [50  5 82 82 65 50 78 65 16 50]
 [56 43 21 22 82 54 76 61 29 50]]

```

```

y
[[78 65 48  7 72  1 50 35 73 73]
 [22 82 50  7 78 65  7 50 65  7]
 [82 16 50 22  1 50 65 50 44 22]
 [50  7 78 72 50  0 78 40 72 44]
 [54 65 10 50 78 72  1 50  7 72]
 [57 54 54 40 22 82 50 65 82 16]
 [ 5 82 82 65 50 78 65 16 50 54]
 [43 21 22 82 54 76 61 29 50 33]]

```

```

In [10]: train_on_gpu = torch.cuda.is_available()

        if train_on_gpu:
            print("CUDA is available. Training on GPU...")
        else:
            print("CUDA is unavailable. Training on CPU...")

```

CUDA is available. Training on GPU...

```

In [39]: class LSTM_Model(nn.Module):

        def __init__(self, token, n_hidden = 256, n_layers = 2, drop_prob = 0.2, lr = 0.001)
            super().__init__()

```

```

self.chars = token
self.int2char = dict(enumerate(self.chars))
self.char2int = {ch: i for i, ch in self.int2char.items()}

self.drop_prob = drop_prob
self.n_hidden = n_hidden
self.n_layers = n_layer
self.lr = lr

self.lstm = nn.LSTM(len(self.chars),n_hidden,n_layers,dropout = drop_prob,batch

self.dropout = nn.Dropout(drop_prob)

self.fc = nn.Linear(n_hidden,len(self.chars))

def forward(self,x,hidden):

    r_output, hidden = self.lstm(x,hidden)

    out = self.dropout(r_output)

    out = out.contiguous().view(-1,self.n_hidden)

    out = self.fc(out)

    return out, hidden

def init_hidden(self, batch_size):
    weight = next(self.parameters()).data

    if (train_on_gpu):
        hidden = (weight.new(self.n_layers, batch_size, self.n_hidden).zero_().cuda
                    weight.new(self.n_layers, batch_size, self.n_hidden).zero_().cuda())
    else:
        hidden = (weight.new(self.n_layers, batch_size, self.n_hidden).zero_(),
                    weight.new(self.n_layers, batch_size, self.n_hidden).zero_())

    return hidden

```

In [40]:

```

n_epochs = 10
batch_size = 128
n_layer = 2
n_hidden = 200
seq_length = 100

net = LSTM_Model(chars,n_hidden,n_layer)
print(net)

```

```

LSTM_Model(
  (lstm): LSTM(83, 200, num_layers=2, batch_first=True, dropout=0.2)
  (dropout): Dropout(p=0.2, inplace=False)
  (fc): Linear(in_features=200, out_features=83, bias=True)
)

```

In [45]:

```

def train(net,data,seq_length = 50,batch_size =100,epochs = 10, lr = 0.001,clip = 5, va
net.train()

```

```

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(),lr = lr)

val_idx = int(len(data)*(1-val_frac))
data, val_data = data[:val_idx], data[val_idx:]

train_losses, valid_losses = [], []

if train_on_gpu:
    net.cuda()

counter = 0
n_chars = len(net.chars)

for i in range(epochs):
    h = net.init_hidden(batch_size)
    rn_loss = 0
    for x,y in get_batches(data,batch_size,seq_length):
        counter +=1

        x = one_hot_encode(x,n_chars)
        inputs, targets = torch.from_numpy(x), torch.from_numpy(y)

        if train_on_gpu:
            inputs,targets = inputs.cuda(),targets.cuda()

        h = tuple([each.data for each in h])

        optimizer.zero_grad()

        output,h = net(inputs,h)

        loss = criterion(output,targets.view(batch_size*seq_length).long())
        loss.backward()
        nn.utils.clip_grad_norm_(net.parameters(),clip)
        optimizer.step()
        rn_loss+= loss.item()

    if counter % prt_stp == 0:
        val_h = net.init_hidden(batch_size)
        valid_loss = 0
        net.eval()
        for val_x,val_y in get_batches(val_data,batch_size,seq_length):
            val_x = one_hot_encode(val_x,n_chars)
            val_x,val_y = torch.from_numpy(val_x),torch.from_numpy(val_y)

            if train_on_gpu:
                val_inputs,val_targets = val_x.cuda(),val_y.cuda()

            val_h = tuple([each.data for each in val_h])

            val_out,val_h = net(val_inputs,val_h)
            val_loss = criterion(val_out,val_targets.view(batch_size*seq_length

            valid_loss +=val_loss.item()

        valid_losses.append(val_loss.item())

```

```

net.train()

print("Epoch: {}/{}.format(i+1,epochs),
      "Step: {}.format(counter),
      "Training loss: {:.3f}.format(rn_loss/len(data)),
      "Validation loss: {:.3f}.format(np.mean(valid_losses)))

```

In [46]: `train(net,encode,seq_length=seq_length,batch_size=batch_size,epochs=n_epochs,lr=0.001)`

```

Epoch: 1/10.... Step: 5.... Training loss: 0.000 Validation loss: 2.456
Epoch: 1/10.... Step: 10.... Training loss: 0.000 Validation loss: 2.434
Epoch: 1/10.... Step: 15.... Training loss: 0.000 Validation loss: 2.414
Epoch: 1/10.... Step: 20.... Training loss: 0.000 Validation loss: 2.405
Epoch: 1/10.... Step: 25.... Training loss: 0.000 Validation loss: 2.397
Epoch: 1/10.... Step: 30.... Training loss: 0.000 Validation loss: 2.391
Epoch: 1/10.... Step: 35.... Training loss: 0.000 Validation loss: 2.385
Epoch: 1/10.... Step: 40.... Training loss: 0.000 Validation loss: 2.381
Epoch: 1/10.... Step: 45.... Training loss: 0.000 Validation loss: 2.377
Epoch: 1/10.... Step: 50.... Training loss: 0.000 Validation loss: 2.374
Epoch: 1/10.... Step: 55.... Training loss: 0.000 Validation loss: 2.371
Epoch: 1/10.... Step: 60.... Training loss: 0.000 Validation loss: 2.368
Epoch: 1/10.... Step: 65.... Training loss: 0.000 Validation loss: 2.366
Epoch: 1/10.... Step: 70.... Training loss: 0.000 Validation loss: 2.364
Epoch: 1/10.... Step: 75.... Training loss: 0.000 Validation loss: 2.362
Epoch: 1/10.... Step: 80.... Training loss: 0.000 Validation loss: 2.359
Epoch: 1/10.... Step: 85.... Training loss: 0.000 Validation loss: 2.357
Epoch: 1/10.... Step: 90.... Training loss: 0.000 Validation loss: 2.355
Epoch: 1/10.... Step: 95.... Training loss: 0.000 Validation loss: 2.353
Epoch: 1/10.... Step: 100.... Training loss: 0.000 Validation loss: 2.351
Epoch: 1/10.... Step: 105.... Training loss: 0.000 Validation loss: 2.350
Epoch: 1/10.... Step: 110.... Training loss: 0.000 Validation loss: 2.348
Epoch: 1/10.... Step: 115.... Training loss: 0.000 Validation loss: 2.346
Epoch: 1/10.... Step: 120.... Training loss: 0.000 Validation loss: 2.344
Epoch: 1/10.... Step: 125.... Training loss: 0.000 Validation loss: 2.342
Epoch: 1/10.... Step: 130.... Training loss: 0.000 Validation loss: 2.340
Epoch: 1/10.... Step: 135.... Training loss: 0.000 Validation loss: 2.339
Epoch: 2/10.... Step: 140.... Training loss: 0.000 Validation loss: 2.337
Epoch: 2/10.... Step: 145.... Training loss: 0.000 Validation loss: 2.335
Epoch: 2/10.... Step: 150.... Training loss: 0.000 Validation loss: 2.333
Epoch: 2/10.... Step: 155.... Training loss: 0.000 Validation loss: 2.332
Epoch: 2/10.... Step: 160.... Training loss: 0.000 Validation loss: 2.330
Epoch: 2/10.... Step: 165.... Training loss: 0.000 Validation loss: 2.328
Epoch: 2/10.... Step: 170.... Training loss: 0.000 Validation loss: 2.327
Epoch: 2/10.... Step: 175.... Training loss: 0.000 Validation loss: 2.325
Epoch: 2/10.... Step: 180.... Training loss: 0.000 Validation loss: 2.323
Epoch: 2/10.... Step: 185.... Training loss: 0.000 Validation loss: 2.321
Epoch: 2/10.... Step: 190.... Training loss: 0.000 Validation loss: 2.320
Epoch: 2/10.... Step: 195.... Training loss: 0.000 Validation loss: 2.318
Epoch: 2/10.... Step: 200.... Training loss: 0.000 Validation loss: 2.316
Epoch: 2/10.... Step: 205.... Training loss: 0.000 Validation loss: 2.314
Epoch: 2/10.... Step: 210.... Training loss: 0.000 Validation loss: 2.312
Epoch: 2/10.... Step: 215.... Training loss: 0.000 Validation loss: 2.311
Epoch: 2/10.... Step: 220.... Training loss: 0.000 Validation loss: 2.309
Epoch: 2/10.... Step: 225.... Training loss: 0.000 Validation loss: 2.307

```

[illegible]

[illegible]

[illegible]

```

Epoch: 9/10.... Step: 1130.... Training loss: 0.000 Validation loss: 2.040
Epoch: 9/10.... Step: 1135.... Training loss: 0.000 Validation loss: 2.038
Epoch: 9/10.... Step: 1140.... Training loss: 0.000 Validation loss: 2.037
Epoch: 9/10.... Step: 1145.... Training loss: 0.000 Validation loss: 2.036
Epoch: 9/10.... Step: 1150.... Training loss: 0.000 Validation loss: 2.035
Epoch: 9/10.... Step: 1155.... Training loss: 0.000 Validation loss: 2.034
Epoch: 9/10.... Step: 1160.... Training loss: 0.000 Validation loss: 2.033
Epoch: 9/10.... Step: 1165.... Training loss: 0.000 Validation loss: 2.032
Epoch: 9/10.... Step: 1170.... Training loss: 0.000 Validation loss: 2.031
Epoch: 9/10.... Step: 1175.... Training loss: 0.000 Validation loss: 2.030
Epoch: 9/10.... Step: 1180.... Training loss: 0.000 Validation loss: 2.028
Epoch: 9/10.... Step: 1185.... Training loss: 0.000 Validation loss: 2.027
Epoch: 9/10.... Step: 1190.... Training loss: 0.000 Validation loss: 2.026
Epoch: 9/10.... Step: 1195.... Training loss: 0.000 Validation loss: 2.025
Epoch: 9/10.... Step: 1200.... Training loss: 0.000 Validation loss: 2.024
Epoch: 9/10.... Step: 1205.... Training loss: 0.000 Validation loss: 2.023
Epoch: 9/10.... Step: 1210.... Training loss: 0.000 Validation loss: 2.022
Epoch: 9/10.... Step: 1215.... Training loss: 0.000 Validation loss: 2.021
Epoch: 9/10.... Step: 1220.... Training loss: 0.000 Validation loss: 2.020
Epoch: 9/10.... Step: 1225.... Training loss: 0.000 Validation loss: 2.019
Epoch: 9/10.... Step: 1230.... Training loss: 0.000 Validation loss: 2.018
Epoch: 9/10.... Step: 1235.... Training loss: 0.000 Validation loss: 2.017
Epoch: 9/10.... Step: 1240.... Training loss: 0.000 Validation loss: 2.016
Epoch: 9/10.... Step: 1245.... Training loss: 0.000 Validation loss: 2.015
Epoch: 9/10.... Step: 1250.... Training loss: 0.000 Validation loss: 2.014
Epoch: 10/10.... Step: 1255.... Training loss: 0.000 Validation loss: 2.013
Epoch: 10/10.... Step: 1260.... Training loss: 0.000 Validation loss: 2.012
Epoch: 10/10.... Step: 1265.... Training loss: 0.000 Validation loss: 2.011
Epoch: 10/10.... Step: 1270.... Training loss: 0.000 Validation loss: 2.009
Epoch: 10/10.... Step: 1275.... Training loss: 0.000 Validation loss: 2.008
Epoch: 10/10.... Step: 1280.... Training loss: 0.000 Validation loss: 2.007
Epoch: 10/10.... Step: 1285.... Training loss: 0.000 Validation loss: 2.006
Epoch: 10/10.... Step: 1290.... Training loss: 0.000 Validation loss: 2.005
Epoch: 10/10.... Step: 1295.... Training loss: 0.000 Validation loss: 2.004
Epoch: 10/10.... Step: 1300.... Training loss: 0.000 Validation loss: 2.003
Epoch: 10/10.... Step: 1305.... Training loss: 0.000 Validation loss: 2.002
Epoch: 10/10.... Step: 1310.... Training loss: 0.000 Validation loss: 2.001
Epoch: 10/10.... Step: 1315.... Training loss: 0.000 Validation loss: 2.000
Epoch: 10/10.... Step: 1320.... Training loss: 0.000 Validation loss: 1.999
Epoch: 10/10.... Step: 1325.... Training loss: 0.000 Validation loss: 1.998
Epoch: 10/10.... Step: 1330.... Training loss: 0.000 Validation loss: 1.997
Epoch: 10/10.... Step: 1335.... Training loss: 0.000 Validation loss: 1.996
Epoch: 10/10.... Step: 1340.... Training loss: 0.000 Validation loss: 1.995
Epoch: 10/10.... Step: 1345.... Training loss: 0.000 Validation loss: 1.994
Epoch: 10/10.... Step: 1350.... Training loss: 0.000 Validation loss: 1.993
Epoch: 10/10.... Step: 1355.... Training loss: 0.000 Validation loss: 1.992
Epoch: 10/10.... Step: 1360.... Training loss: 0.000 Validation loss: 1.991
Epoch: 10/10.... Step: 1365.... Training loss: 0.000 Validation loss: 1.991
Epoch: 10/10.... Step: 1370.... Training loss: 0.000 Validation loss: 1.990
Epoch: 10/10.... Step: 1375.... Training loss: 0.000 Validation loss: 1.989
Epoch: 10/10.... Step: 1380.... Training loss: 0.000 Validation loss: 1.988
Epoch: 10/10.... Step: 1385.... Training loss: 0.000 Validation loss: 1.987
Epoch: 10/10.... Step: 1390.... Training loss: 0.000 Validation loss: 1.986

```

```

In [47]: model_name = 'LSTM_text_generation_10_epoch.net'

checkpoint = {'n_hidden': net.n_hidden,
             'n_layers': net.n_layers,
             'state_dict': net.state_dict(),

```

```

        'tokens': net.chars}

with open(model_name, 'wb') as f:
    torch.save(checkpoint, f)

```

```

In [48]: def predict(net, char, h=None, top_k=None):
        ''' Given a character, predict the next character.
            Returns the predicted character and the hidden state.
            ...

        # tensor inputs
        x = np.array([[net.char2int[char]]])
        x = one_hot_encode(x, len(net.chars))
        inputs = torch.from_numpy(x)

        if(train_on_gpu):
            inputs = inputs.cuda()

        # detach hidden state from history
        h = tuple([each.data for each in h])
        # get the output of the model
        out, h = net(inputs, h)

        # get the character probabilities
        p = F.softmax(out, dim=1).data
        if(train_on_gpu):
            p = p.cpu() # move to cpu

        # get top characters
        if top_k is None:
            top_ch = np.arange(len(net.chars))
        else:
            p, top_ch = p.topk(top_k)
            top_ch = top_ch.numpy().squeeze()

        # select the likely next character with some element of randomness
        p = p.numpy().squeeze()
        char = np.random.choice(top_ch, p=p/p.sum())

        # return the encoded value of the predicted char and the hidden state
        return net.int2char[char], h

```

```

In [49]: def sample(net, size, prime='The', top_k=None):

        if(train_on_gpu):
            net.cuda()
        else:
            net.cpu()

        net.eval() # eval mode

        # First off, run through the prime characters
        chars = [ch for ch in prime]
        h = net.init_hidden(1)
        for ch in prime:
            char, h = predict(net, ch, h, top_k=top_k)

```

```

chars.append(char)

# Now pass in the previous character and get a new one
for ii in range(size):
    char, h = predict(net, chars[-1], h, top_k=top_k)
    chars.append(char)

return ''.join(chars)

```

```

In [50]: print(sample(net, 1000, prime='Anna', top_k=5))

```

Anna happined to say wish over his somettion with the hess and
 hor and that the mare hand of too
 heart, and all that he was not to
 the counting an the postion. "Talk of the singer, and a could but the cersion time
 thing. I'm neters to sis of the pail, what his constom tore to but the room? I say. I ma
 nd the sind is to aster in so murt," all the poust it to be to the mind a from simper, a
 nd had that and shorted of all wonder thein that so see so there with the cropse one, at
 to the hand, and hord thring had
 and with and her. "What he sound telrone a for and and as im alone of
 mere and at the reart, strece the
 seather with only tall, she sount themeren at her and there had to the soon, and the pei
 nce. She had but the said to bur timp his
 cantess that the sout and her say to be a mound the prisian or the prestring him. "You wa
 n ant to sint. There she same then't to be a prince of the minds, and that he was to and
 him the pair, she all, when't at she
 seet to
 have on the prentaring that he was no

In []: