lab8q3

March 11, 2025

[63]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F

from torch.utils.data import Dataset, DataLoader

from tqdm.notebook import tqdm

device = "cuda" if torch.cuda.is_available() else "cpu"
```

[64]:
```
with open('./input.txt', 'r', encoding='utf-8') as f:
    text = f.read()

unique_chars = sorted(list(set(text)))
vocab_size = len(unique_chars)

stoi = { ch:i for i,ch in enumerate(unique_chars) }
itos = { i:ch for i,ch in enumerate(unique_chars) }

tokens = torch.Tensor([stoi[x] for x in list(text)][:5000])

train_split = 0.9
n = int(train_split * len(tokens))
train, test = tokens[:n], tokens[n:]
```

[65]:
```
class TokensDataset(Dataset):

    def __init__(self, series, seq_length=10, device="cpu"):
        self.series = series
        self.seq_length = seq_length

    def __len__(self):
        return len(self.series) - self.seq_length - 1

    def __getitem__(self, idx):
        X = self.series[idx:idx+self.seq_length].reshape(self.seq_length, -1).
 ↪to(device)
        y = self.series[idx+self.seq_length].to(device)
```

```
        return X, y
```

[66]:
```
train_dataset = TokensDataset(train, seq_length=10, device=device)
test_dataset = TokensDataset(test, seq_length=10, device=device)
train_data = DataLoader(train_dataset, batch_size=1024, shuffle=True)
test_data = DataLoader(test_dataset, batch_size=1, shuffle=False)
```

[67]:
```
class RNN(nn.Module):

    def __init__(self, input_size, hidden_size):

        super(RNN, self).__init__()
        self.rnn = nn.RNN(input_size, hidden_size, batch_first=True) # (batch, time, input_size) -> (batch, time, hidden_size)
        self.fc = nn.Sequential(
            nn.ReLU(),
            nn.Linear(in_features=hidden_size, out_features=1), # out_features = 1 to predict the next value
        )

    def forward(self, x):
        output, status = self.rnn(x)
        output = output[:, -1, :] # (batch, time, hidden_size) -> (batch, 1, hidden_size). We take only the final output of the RNN
        output = self.fc(output)

        return output
```

[73]:
```
input_size = 1
hidden_size = 5
learning_rate = 1e-4
num_epochs = 100
# ----

model = RNN(input_size, hidden_size).to(device)

criterion = nn.MSELoss().to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

[74]:
```
for epoch in tqdm(range(num_epochs), desc="Epochs"):
    for batch, targets in tqdm(train_data, desc="Batches", leave=False):

        outs = model(batch).reshape(-1)

        optimizer.zero_grad()
        loss = criterion(outs, targets)
```

```
        loss.backward()
        optimizer.step()

    print(f"After epoch {epoch}, loss={loss.item()}")
```

```
Epochs:   0%|          | 0/100 [00:00<?, ?it/s]
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 0, loss=1992.2626953125
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 1, loss=1885.878173828125
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 2, loss=1948.0467529296875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 3, loss=2013.9732666015625
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 4, loss=1798.8543701171875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 5, loss=2023.5047607421875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 6, loss=1881.8582763671875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 7, loss=1877.371337890625
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 8, loss=1983.71337890625
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 9, loss=1877.7977294921875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 10, loss=2007.5245361328125
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 11, loss=1950.912109375
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 12, loss=1860.246826171875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 13, loss=1865.693603515625
```

```
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 14, loss=1904.16796875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 15, loss=1949.6217041015625
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 16, loss=1898.039794921875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 17, loss=1829.5120849609375
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 18, loss=1961.5357666015625
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 19, loss=1817.76171875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 20, loss=2005.6490478515625
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 21, loss=1909.7557373046875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 22, loss=1849.711181640625
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 23, loss=1753.5135498046875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 24, loss=1848.5902099609375
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 25, loss=1879.3363037109375
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 26, loss=1835.8065185546875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 27, loss=1948.9007568359375
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 28, loss=1961.7918701171875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 29, loss=1804.4312744140625
```

```
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 30, loss=1742.824951171875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 31, loss=1850.6612548828125
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 32, loss=1940.376220703125
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 33, loss=1900.0343017578125
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 34, loss=1870.9178466796875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 35, loss=1961.1295166015625
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 36, loss=1885.566650390625
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 37, loss=1838.7528076171875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 38, loss=2005.67333984375
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 39, loss=1853.469482421875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 40, loss=1857.2978515625
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 41, loss=1791.9185791015625
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 42, loss=1927.1419677734375
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 43, loss=1977.2379150390625
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 44, loss=1855.255126953125
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 45, loss=1859.33935546875
```

```
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 46, loss=1865.2451171875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 47, loss=1923.648681640625
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 48, loss=1773.3778076171875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 49, loss=1834.9500732421875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 50, loss=1821.5635986328125
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 51, loss=1826.02099609375
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 52, loss=1823.6697998046875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 53, loss=1877.154052734375
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 54, loss=1855.5758056640625
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 55, loss=1907.3231201171875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 56, loss=1817.8345947265625
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 57, loss=1982.0594482421875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 58, loss=1822.922607421875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 59, loss=1951.8250732421875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 60, loss=1856.9027099609375
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 61, loss=1878.0267333984375
```

```
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 62, loss=1848.790771484375
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 63, loss=1856.056640625
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 64, loss=1809.7763671875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 65, loss=1898.186767578125
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 66, loss=1887.761962890625
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 67, loss=1802.655517578125
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 68, loss=1886.7816162109375
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 69, loss=1891.119873046875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 70, loss=1844.844482421875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 71, loss=1873.3714599609375
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 72, loss=1857.0550537109375
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 73, loss=1771.0908203125
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 74, loss=1869.6109619140625
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 75, loss=1929.2374267578125
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 76, loss=1889.684814453125
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 77, loss=1803.9615478515625
```

```
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 78, loss=2007.9227294921875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 79, loss=1858.7099609375
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 80, loss=1841.296630859375
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 81, loss=1871.027099609375
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 82, loss=1926.5225830078125
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 83, loss=1909.91748046875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 84, loss=1905.2021484375
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 85, loss=1833.14013671875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 86, loss=1918.470703125
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 87, loss=1947.6895751953125
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 88, loss=1901.3333740234375
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 89, loss=1885.99462890625
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 90, loss=1909.6016845703125
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 91, loss=1726.591796875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 92, loss=1888.141845703125
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 93, loss=1855.7220458984375
```

```
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 94, loss=1894.3126220703125
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 95, loss=1868.59619140625
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 96, loss=1928.863525390625
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 97, loss=1796.647216796875
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 98, loss=1938.6146240234375
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
After epoch 99, loss=1990.7935791015625
```

[75]:
```python
import torch.nn.functional as F

def predict(model, start_text, length=100, temperature=1.0):
    model.eval()

    # Convert input text to tensor
    input_seq = torch.tensor([stoi[ch] for ch in start_text], dtype=torch.
 ↪float32).reshape(1, -1, 1).to(device)

    generated = start_text
    for _ in range(length):
        with torch.no_grad():
            output = model(input_seq)  # Get model output
            probs = F.softmax(output / temperature, dim=-1)  # Apply␣
 ↪temperature scaling

            # Sample from probability distribution
            next_char_idx = torch.multinomial(probs, num_samples=1).item()

        generated += itos[next_char_idx]

        # Update input sequence
        next_input = torch.tensor([[next_char_idx]], dtype=torch.float32,␣
 ↪device=device).reshape(1, 1, 1)
        input_seq = torch.cat((input_seq[:, 1:, :], next_input), dim=1)  # Keep␣
 ↪seq length fixed

    return generated
```

[76]:
```
start_text = "hello worl"
generated_text = predict(model, start_text, length=50, temperature=0.8)
print("Generated Text:", generated_text)
```

```
Generated Text: hello worl
```

[ ]: