Part 2 : Remebering Number

In [ ]:

```
import torch, os, sys
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
import random
from matplotlib import pyplot as plt
DEVICE_DEFAULT = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

Utility Functions

In [ ]:

```
def pbar(p=0, msg="", bar_len=20):
    sys.stdout.write("\033[K")
    sys.stdout.write("\x1b[2K" + "\r")
    block = int(round(bar_len * p))
    text = "Progress: [{}] {}% {}".format(
        "\x1b[32m" + "=" * (block - 1) + ">" + "\033[0m" + "-" * (bar_len - block),
        round(p * 100, 2),
        msg,
    )
    print(text, end="\r")
    if p == 1:
        print()

class AvgMeter:
    def __init__(self):
        self.reset()

    def reset(self):
        self.metrics = {}

    def add(self, batch_metrics):
        for key, value in batch_metrics.items():
            if key in self.metrics.items():
                self.metrics[key].append(value)
            else:
                self.metrics[key] = [value]

    def get(self):
        return {key: np.mean(value) for key, value in self.metrics.items()}

    def msg(self):
        avg_metrics = {key: np.mean(value) for key, value in self.metrics.items()}
        return "".join(["[{}] {:.5f} ".format(key, value) for key, value in avg_metrics.items()])

def train(model, optim, lr_sched=None, epochs=200, device=DEVICE_DEFAULT, criterion=None, metric_meter=None, out_path=
    model.to(device)
    best_acc = 0
    for epoch in range(epochs):
        model.train()
        metric_meter.reset()
        for indx, (seq, target) in enumerate(train_data):
            seq = seq.to(device)
            target = target.to(device)

            optim.zero_grad()
            out = model.forward(seq)
            loss = criterion(out, target)
            loss.backward()
            optim.step()

            metric_meter.add({"train loss": loss.item()})
#             pbar(indx / len(train_data), msg=metric_meter.msg())
#         pbar(1, msg=metric_meter.msg())
        train_loss_for_plot.append(metric_meter.get()["train loss"])
        print(train_loss_for_plot[-1], epoch)


        model.eval()
        metric_meter.reset()
        for indx, (seq, target) in enumerate(train_data):
            seq = seq.to(device)
            target = target.to(device)

            out = model.forward(seq)
            loss = criterion(out, target)
            acc = (out.argmax(1) == target).sum().item() * (100 / seq.shape[0])
            metric_meter.add({"train acc": acc})
#              pbar(indx / len(train_data), msg=metric_meter.msg())
#          pbar(1, msg=metric_meter.msg())

        train_metrics = metric_meter.get()
        train_acc_for_plot.append(max(train_metrics["train acc"], best_acc))
        if train_metrics["train acc"] > best_acc:
            print(
              "\x1b[33m"
              + f"train acc improved from {round(best_acc, 5)} to {round(train_metrics['train acc'], 5)}"
              + "\033[0m"
            )
            best_acc = train_metrics['train acc']
#              torch.save(model.state_dict(), out_path)
    lr_sched.step()
```

Generate Train and Test data

In [ ]:

```
TRAIN_SIZE, TEST_SIZE = 1000, 500

# Generate a data point of given L
# Output - (Seq of length L, One of Hot Rep of number at position pos)
def gen_example(L, pos=2):
    # (Batch S, seq S, feature S) = (1, L, 10)
    assert L > pos
    inp_before = torch.randint(0, 9 + 1, (3, L,))
    inp = F.one_hot(inp_before, num_classes=10).type(torch.float)
    out = inp_before[:,pos]
    return (inp, out)

data = []
for i in range(TRAIN_SIZE + TEST_SIZE):
    L = random.randint(3, 10+1)
    data.append(gen_example(L))

train_data, test_data = data[:TRAIN_SIZE], data[-TEST_SIZE:]
```

Training

In [ ]:

```
class LSTM(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_layers, bidirectional, output_dim):

        super().__init__()

        self.lstm = nn.LSTM(input_size = input_dim,
                          hidden_size = hidden_dim,
                          num_layers = num_layers,
                          batch_first = True,
                          bidirectional = bidirectional
                         )

        D = (2 if bidirectional else 1)

        self.fc = nn.Linear(D * num_layers * hidden_dim, output_dim)

    def forward(self, batch):

        assert batch.dim() == 3

        output, (hidden, cell) = self.lstm(batch)

        # D = 2 if bidirectional, else D = 1
        # output = [batch size, seq length, D * hidden_dim]
        # hidden = [D * num_layers, batch size, hidden_dim]

        flat_hidden = torch.cat([hidden[i,:,:] for i in range(hidden.shape[0])], dim = 1)

        output = self.fc(flat_hidden)


        return output

# class RNN(nn.Module):
#     def __init__(self, input_dim, hidden_dim, num_layers, bidirectional, output_dim):

#         super().__init__()

#         self.rnn = nn.RNN(input_size = input_dim,
#                           hidden_size = hidden_dim,
#                           num_layers = num_layers,
#                            batch_first = True,
#                            bidirectional = bidirectional,
#                            nonlinearity = 'tanh'
#                           )

#         D = (2 if bidirectional else 1)

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

#     def forward(self, batch):

#         assert batch.dim() == 3

#         output, hidden = self.rnn(batch)

#         # D = 2 if bidirectional, else D = 1
#         # output = [batch size, seq length, D * hidden_dim]
#         # hidden = [D * num_layers, batch size, hidden_dim]

#         flat_hidden = torch.cat([hidden[i,:,:] for i in range(hidden.shape[0])], dim = 1)

#         output = self.fc(flat_hidden)

#         return output
```

In [ ]:

```
INPUT_DIM = 10
HIDDEN_DIM = 2
OUTPUT_DIM = 10
NUM_LAYERS = 1
BIDIRECTIONAL = False
EPOCHS = 200

model = LSTM(INPUT_DIM, HIDDEN_DIM, NUM_LAYERS, BIDIRECTIONAL, OUTPUT_DIM)

out_dir = "Part2/"
out_path = out_dir + "model2.ckpt"
os.makedirs(out_dir, exist_ok=True)

# UNCOMMENT FROM HERE FOR TRAINING

optim = torch.optim.Adam(model.parameters(), lr=10**-4, weight_decay=5e-4)
# optim = torch.optim.SGD(model.parameters(), lr=10**-4, momentum=0.9)
```

```python
lr_sched = torch.optim.lr_scheduler.CosineAnnealingLR(optim, T_max=EPOCHS)
criterion = nn.CrossEntropyLoss()
metric_meter = AvgMeter()

train_loss_for_plot = []
train_acc_for_plot = []

train(model, optim, lr_sched, device=DEVICE_DEFAULT, epochs=EPOCHS, criterion=criterion, metric_meter=metric_meter, ou
# After this the model will be saved in out_dir
```

In [ ]:

```python
plt.figure(figsize=(15, 3))
plt.subplot(1, 3, 1)
plt.plot(train_loss_for_plot)
plt.xlabel("Epoch #")
plt.ylabel("Train Loss")
plt.title("Train Loss vs. Epochs")

plt.subplot(1, 3, 2)
plt.plot(train_acc_for_plot)
plt.xlabel("Epoch #")
plt.ylabel("Train accuracy")
plt.title("Train Accuracy vs. Epochs")

plt.show()
```

In [ ]:

```python
device = DEVICE_DEFAULT
model.eval()
metric_meter.reset()
for indx, (seq, target) in enumerate(test_data):
    seq = seq.to(device)
    target = target.to(device)
    out = model.forward(seq)
#     loss = criterion(out, target)
    acc = (out.argmax(1) == target).sum().item() * (100 / seq.shape[0])
    metric_meter.add({"test acc": acc})

test_metrics = metric_meter.get()
print("Test Accuracy", test_metrics["test acc"], "%")
```

In [ ]:

```python
for L in range(3, 10 + 1):
    for _ in range(2):
        example_input = gen_example(L)
        print("Input", [example_input[0][0][i].argmax().item() for i in range(L)])
        print("Received", model.forward(example_input[0].to(device)).argmax(1).item())
```