Part 3 : Addition

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
from torch.utils.data import TensorDataset, DataLoader
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, criterion=None, metric_meter=None, out_path="best.ckpt", device=DEV
    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"])

        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 >= 0.5).type(torch.float32) == target).sum().item() * (100 / (seq.shape[0] * seq.shape[1]))
```

```
            metric_meter.add({"test loss": loss.item(), "test acc": acc})
            pbar(indx / len(test_data), msg=metric_meter.msg())
        pbar(1, msg=metric_meter.msg())

        test_metrics = metric_meter.get()
        test_acc_for_plot.append(test_metrics["test acc"])
        test_loss_for_plot.append(test_metrics["test loss"])
        if test_metrics["test acc"] > best_acc:
            print(
              "\x1b[33m"
              + f"test acc improved from {round(best_acc, 5)} to {round(test_metrics['test acc'], 5)} in epoch {epoch}
              + "\033[0m"
            )
            best_acc = test_metrics['test acc']
            torch.save(model.state_dict(), out_path)
    lr_sched.step()
```

Generate Train and Test data

In [ ]:

```
def pair2tensor(a, b, bits):
    t = torch.zeros(1, bits, 2)
    for i in range(bits):
        t[0, i, 0] = (1.0 if i < len(a) and a[-i-1] == '1' else 0.0)
        t[0, i, 1] = (1.0 if i < len(b) and b[-i-1] == '1' else 0.0)
    return t

def num2tensor(c, bits):
    t = torch.zeros(1, bits)
    for i in range(bits):
        t[0, i] = (1.0 if i < len(c) and c[-i-1] == '1' else 0.0)
    return t

def Bits(b):
    return 1 + len(bin(b)[2:])

def generate(data_size, L):
    # (Batch S, seq S, feature S) = (1, L, 2)
    data = []
    for i in range(data_size):
        a, b = random.randint(0, 2**(L-1)), random.randint(0, 2**(L-1))
```

```
        c = a+b
        data.append((pair2tensor(bin(a)[2:], bin(b)[2:], L), num2tensor(bin(c)[2:], L)))
    return data
```

In [ ]:
```
TRAIN_SIZE, TEST_SIZE_PER_L = 10000, 100
TRAIN_L = 5

train_data = []
for _ in range(TRAIN_SIZE):
    L = random.randint(1, 20 + 1)
    train_data += generate(1, L)

test_data = []
for L in range(1, 20 + 1):
    test_data += generate(TEST_SIZE_PER_L, L)
```

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 * 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]

        seq_len = output.shape[1]

        out = torch.cat([torch.sigmoid(self.fc(output[:,i,:])) for i in range(seq_len)], dim=1)

        return out
```

In [ ]:
```
INPUT_DIM = 2
HIDDEN_DIM = 5
OUTPUT_DIM = 1
NUM_LAYERS = 1
BIDIRECTIONAL = False
EPOCHS = 5

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

out_dir = "Part3/"
out_path = out_dir + "config3_state5.ckpt"
os.makedirs(out_dir, exist_ok=True)

# UNCOMMENT FROM HERE FOR TRAINING

# optim = torch.optim.SGD(model.parameters(), lr=10**-3, momentum=0.9, weight_decay=5e-4)
optim = torch.optim.Adam(model.parameters(), lr=10**-3, weight_decay=5e-4)
lr_sched = torch.optim.lr_scheduler.CosineAnnealingLR(optim, T_max=EPOCHS)
criterion = nn.MSELoss()
# criterion = nn.BCELoss()
metric_meter = AvgMeter()

train_loss_for_plot = []
test_loss_for_plot = []
test_acc_for_plot = []

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

In [ ]:
```
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(test_loss_for_plot)
plt.xlabel("Epoch #")
plt.ylabel("Test Loss")
plt.title("Test Loss vs. Epochs")

plt.subplot(1, 3, 3)
plt.plot(test_acc_for_plot)
plt.xlabel("Epoch #")
plt.ylabel("Test Accuracy")
plt.ylim([0,105])
plt.title("Test Accuracy vs. Epochs")
plt.show()
```

In [ ]:

```
model.load_state_dict(torch.load(out_path))
print(out_path)
model.to(DEVICE_DEFAULT)
model.eval()
metric_meter = AvgMeter()
for indx, (seq, target) in enumerate(test_data):
    seq = seq.to(DEVICE_DEFAULT)
    target = target.to(DEVICE_DEFAULT)
    out = model.forward(seq)
    acc = ((out >= 0.5).type(torch.float32) == target).sum().item() * (100 / (seq.shape[0] * seq.shape[1]))
    metric_meter.add({"L" + str(seq.shape[1]) : acc})
test_metrics = metric_meter.get()
plt.plot([i+1 for i in range(20)], [test_metrics["L"+str(i+1)] for i in range(20)])
plt.xlabel('Length of Seq (L)')
plt.ylabel("Accuracy")
plt.title("L vs. Accuracy for MSE")
plt.show()
```