Define model and import dependencies

In [ ]:
```
import torch
from torch import optim, nn, utils, Tensor
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torch.utils.data import Dataset
import pandas as pd
import numpy as np
```

In [ ]:
```
# Defining our model
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, num_classes):
        super(LSTMModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device=x.dev
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device=x.dev

        out, _ = self.lstm(x, (h0, c0))

        out = self.fc(out[:, -1, :])

        return out
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

In [ ]:
```
# Check if GPU is available. But should work fine on CPU too.
if torch.cuda.is_available():
    device = torch.device('cuda')
else:
    device = torch.device('cpu')

print('device:', device)
```

device: cuda

If you're training the model:

In [ ]:
```
# train and validation data
!unzip -q /content/drive/MyDrive/ybigta/2023-2/win_prediction_model/initial.zip -d /cc

# Open data_label, contains keypoints numpy array names and labels
data_label = pd.read_csv("/content/initial/data_label.csv", index_col = 0)
```

In [ ]:
```
# custom defined dataset for our data format
class KeypointsDataset(Dataset):
```

```python
    def __init__(self, array_paths, labels, num_input):
        """
        Generates a custom numpy dataset

        Args:
            array_paths (list): list of numpy array inputs
            labels (list): list of labels
            num_input (int) : number of inputs to enter
        """
        self.array_paths = array_paths
        self.labels = labels
        self.num_input = num_input

    def __len__(self):
        """
        Returns length of entire dataset
        """
        return len(self.array_paths)

    def __getitem__(self, idx, ):
        """
        Gets the sample that corresponds to the sample id (idx)

        Args:
            idx (int): sample index

        Returns:
            keypoints (torch.Tensor): keypoints input tensor
            label (torch.Tensor): winning label tensor
        """
        keypoints_path = "/content/initial/data/"
        # load and turn numpy arrays into torch tensors
        keypoints = np.load(f"{keypoints_path}{self.array_paths[idx]}.npy")
        keypoints = keypoints[:,-self.num_input:]
        keypoints = torch.tensor(keypoints, dtype = torch.float32)
        label = torch.tensor(self.labels[idx])

        return keypoints, label
```

In [ ]:

```python
from sklearn.model_selection import train_test_split
train_array, val_array = train_test_split(data_label, test_size = 0.1, random_state = 1
```

In [ ]:

```python
# Initialise the dataset and dataloader
num_in = 100
train_data = KeypointsDataset(
    array_paths = train_array['modified_json_filename'].reset_index(drop = True),
    labels = train_array['label'].reset_index(drop=True),
    num_input = num_in
    )
val_data = KeypointsDataset(
    array_paths = val_array['modified_json_filename'].reset_index(drop= True),
    labels = val_array['label'].reset_index(drop= True),
    num_input = num_in
    )
train_loader = DataLoader(train_data)
val_loader = DataLoader(val_data)
```

In [ ]:

```python
def train_loop(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
```

```
    # Set the model to training mode - important for batch normalization and dropout lay
    # Unnecessary in this situation but added for best practices
    model.train()
    for i, (inputs, labels) in enumerate(dataloader):
        inputs = inputs.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()

        outputs = model(inputs)

        loss = loss_fn(outputs, labels)
        loss.backward()

        optimizer.step()

        if i %  51 == 0:
            loss, current = loss.item(), (i + 1) * len(inputs)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")


def test_loop(dataloader, model, loss_fn):
    # Set the model to evaluation mode - important for batch normalization and dropout
    # Unnecessary in this situation but added for best practices
    model.eval()
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    test_loss, correct = 0, 0

    # Evaluating the model with torch.no_grad() ensures that no gradients are computed d
    # also serves to reduce unnecessary gradient computations and memory usage for tenso
    with torch.no_grad():
        for i, (inputs, labels) in enumerate(dataloader):
          inputs = inputs.to(device)
          labels = labels.to(device)

          outputs = model(inputs)

          _, predicted = torch.max(outputs.data, 1)
          test_loss += loss_fn(outputs, labels).item()
          correct += (outputs.argmax(1) == labels).type(torch.float).sum().item()
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: ₩n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} ₩
```

In [ ]:

```
# Setting up the model and initiating the train & validation loops
model = LSTMModel(input_size=num_in, hidden_size=10, num_layers=1, num_classes=2).to(de

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001, weight_decay = 0.1)

num_epochs = 60
for t in range(num_epochs):
    print(f"Epoch {t+1}₩n-------------------------------")
    train_loop(train_loader, model, criterion, optimizer)
    test_loop(val_loader, model, criterion)
print("Done!")
```

```
Epoch 1
-------------------------------
loss: 0.753100  [    1/   54]
loss: 0.844226  [   52/   54]
Test Error: 
 Accuracy: 33.3%, Avg loss: 0.737354 

Epoch 2
-------------------------------
loss: 0.754023  [    1/   54]
loss: 0.845346  [   52/   54]
Test Error: 
 Accuracy: 33.3%, Avg loss: 0.735843 

Epoch 3
-------------------------------
loss: 0.755222  [    1/   54]
loss: 0.846515  [   52/   54]
Test Error: 
 Accuracy: 33.3%, Avg loss: 0.734260 

Epoch 4
-------------------------------
loss: 0.756355  [    1/   54]
loss: 0.847704  [   52/   54]
Test Error: 
 Accuracy: 33.3%, Avg loss: 0.732601 

Epoch 5
-------------------------------
loss: 0.757482  [    1/   54]
loss: 0.848895  [   52/   54]
Test Error: 
 Accuracy: 33.3%, Avg loss: 0.730864 

Epoch 6
-------------------------------
loss: 0.758621  [    1/   54]
loss: 0.850077  [   52/   54]
Test Error: 
 Accuracy: 33.3%, Avg loss: 0.729048 

Epoch 7
-------------------------------
loss: 0.759782  [    1/   54]
loss: 0.851233  [   52/   54]
Test Error: 
 Accuracy: 33.3%, Avg loss: 0.727152 

Epoch 8
-------------------------------
loss: 0.760967  [    1/   54]
loss: 0.852350  [   52/   54]
Test Error: 
 Accuracy: 33.3%, Avg loss: 0.725174 

Epoch 9
-------------------------------
loss: 0.762177  [    1/   54]
loss: 0.853411  [   52/   54]
```

```
Test Error: 
 Accuracy: 33.3%, Avg loss: 0.723113 

Epoch 10
-------------------------------
loss: 0.763412  [    1/   54]
loss: 0.854398  [   52/   54]
Test Error: 
 Accuracy: 33.3%, Avg loss: 0.720966 

Epoch 11
-------------------------------
loss: 0.764672  [    1/   54]
loss: 0.855291  [   52/   54]
Test Error: 
 Accuracy: 33.3%, Avg loss: 0.718730 

Epoch 12
-------------------------------
loss: 0.765954  [    1/   54]
loss: 0.856072  [   52/   54]
Test Error: 
 Accuracy: 33.3%, Avg loss: 0.716403 

Epoch 13
-------------------------------
loss: 0.767257  [    1/   54]
loss: 0.856721  [   52/   54]
Test Error: 
 Accuracy: 33.3%, Avg loss: 0.713981 

Epoch 14
-------------------------------
loss: 0.768579  [    1/   54]
loss: 0.857220  [   52/   54]
Test Error: 
 Accuracy: 33.3%, Avg loss: 0.711463 

Epoch 15
-------------------------------
loss: 0.769916  [    1/   54]
loss: 0.857551  [   52/   54]
Test Error: 
 Accuracy: 33.3%, Avg loss: 0.708845 

Epoch 16
-------------------------------
loss: 0.771267  [    1/   54]
loss: 0.857699  [   52/   54]
Test Error: 
 Accuracy: 33.3%, Avg loss: 0.706126 

Epoch 17
-------------------------------
loss: 0.772627  [    1/   54]
loss: 0.857655  [   52/   54]
Test Error: 
 Accuracy: 33.3%, Avg loss: 0.703305 

Epoch 18
```

```
-------------------------------
loss: 0.773994  [    1/   54]
loss: 0.857410  [   52/   54]
Test Error:
 Accuracy: 33.3%, Avg loss: 0.700380

Epoch 19
-------------------------------
loss: 0.775363  [    1/   54]
loss: 0.856965  [   52/   54]
Test Error:
 Accuracy: 33.3%, Avg loss: 0.697349

Epoch 20
-------------------------------
loss: 0.776730  [    1/   54]
loss: 0.856322  [   52/   54]
Test Error:
 Accuracy: 33.3%, Avg loss: 0.694207

Epoch 21
-------------------------------
loss: 0.778091  [    1/   54]
loss: 0.855491  [   52/   54]
Test Error:
 Accuracy: 33.3%, Avg loss: 0.690947

Epoch 22
-------------------------------
loss: 0.779440  [    1/   54]
loss: 0.854485  [   52/   54]
Test Error:
 Accuracy: 33.3%, Avg loss: 0.687556

Epoch 23
-------------------------------
loss: 0.780770  [    1/   54]
loss: 0.853324  [   52/   54]
Test Error:
 Accuracy: 33.3%, Avg loss: 0.684013

Epoch 24
-------------------------------
loss: 0.782076  [    1/   54]
loss: 0.852027  [   52/   54]
Test Error:
 Accuracy: 33.3%, Avg loss: 0.680290

Epoch 25
-------------------------------
loss: 0.783349  [    1/   54]
loss: 0.850616  [   52/   54]
Test Error:
 Accuracy: 33.3%, Avg loss: 0.676348

Epoch 26
-------------------------------
loss: 0.784582  [    1/   54]
loss: 0.849112  [   52/   54]
Test Error:
```

```
 Accuracy: 33.3%, Avg loss: 0.672157

Epoch 27
-------------------------------
loss: 0.785767  [    1/   54]
loss: 0.847522  [   52/   54]
Test Error:
 Accuracy: 33.3%, Avg loss: 0.667716

Epoch 28
-------------------------------
loss: 0.786895  [    1/   54]
loss: 0.845837  [   52/   54]
Test Error:
 Accuracy: 33.3%, Avg loss: 0.663079

Epoch 29
-------------------------------
loss: 0.787960  [    1/   54]
loss: 0.844027  [   52/   54]
Test Error:
 Accuracy: 33.3%, Avg loss: 0.658367

Epoch 30
-------------------------------
loss: 0.788960  [    1/   54]
loss: 0.842068  [   52/   54]
Test Error:
 Accuracy: 33.3%, Avg loss: 0.653736

Epoch 31
-------------------------------
loss: 0.789901  [    1/   54]
loss: 0.839984  [   52/   54]
Test Error:
 Accuracy: 33.3%, Avg loss: 0.649314

Epoch 32
-------------------------------
loss: 0.790797  [    1/   54]
loss: 0.837869  [   52/   54]
Test Error:
 Accuracy: 33.3%, Avg loss: 0.645132

Epoch 33
-------------------------------
loss: 0.791672  [    1/   54]
loss: 0.835827  [   52/   54]
Test Error:
 Accuracy: 50.0%, Avg loss: 0.641122

Epoch 34
-------------------------------
loss: 0.792546  [    1/   54]
loss: 0.833888  [   52/   54]
Test Error:
 Accuracy: 50.0%, Avg loss: 0.637193

Epoch 35
-------------------------------
```

```
loss: 0.793424  [    1/   54]
loss: 0.832006  [   52/   54]
Test Error: 
 Accuracy: 50.0%, Avg loss: 0.633305 

Epoch 36
-------------------------------
loss: 0.794302  [    1/   54]
loss: 0.830107  [   52/   54]
Test Error: 
 Accuracy: 50.0%, Avg loss: 0.629461 

Epoch 37
-------------------------------
loss: 0.795168  [    1/   54]
loss: 0.828129  [   52/   54]
Test Error: 
 Accuracy: 50.0%, Avg loss: 0.625678 

Epoch 38
-------------------------------
loss: 0.796018  [    1/   54]
loss: 0.826027  [   52/   54]
Test Error: 
 Accuracy: 50.0%, Avg loss: 0.621969 

Epoch 39
-------------------------------
loss: 0.796846  [    1/   54]
loss: 0.823774  [   52/   54]
Test Error: 
 Accuracy: 50.0%, Avg loss: 0.618336 

Epoch 40
-------------------------------
loss: 0.797650  [    1/   54]
loss: 0.821353  [   52/   54]
Test Error: 
 Accuracy: 50.0%, Avg loss: 0.614779 

Epoch 41
-------------------------------
loss: 0.798430  [    1/   54]
loss: 0.818757  [   52/   54]
Test Error: 
 Accuracy: 50.0%, Avg loss: 0.611292 

Epoch 42
-------------------------------
loss: 0.799186  [    1/   54]
loss: 0.815979  [   52/   54]
Test Error: 
 Accuracy: 50.0%, Avg loss: 0.607869 

Epoch 43
-------------------------------
loss: 0.799919  [    1/   54]
loss: 0.813020  [   52/   54]
Test Error: 
 Accuracy: 66.7%, Avg loss: 0.604503 
```

```
Epoch 44
-------------------------------
loss: 0.800629  [    1/   54]
loss: 0.809878  [   52/   54]
Test Error: 
 Accuracy: 66.7%, Avg loss: 0.601188 

Epoch 45
-------------------------------
loss: 0.801317  [    1/   54]
loss: 0.806558  [   52/   54]
Test Error: 
 Accuracy: 66.7%, Avg loss: 0.597916 

Epoch 46
-------------------------------
loss: 0.801984  [    1/   54]
loss: 0.803063  [   52/   54]
Test Error: 
 Accuracy: 66.7%, Avg loss: 0.594682 

Epoch 47
-------------------------------
loss: 0.802630  [    1/   54]
loss: 0.799401  [   52/   54]
Test Error: 
 Accuracy: 66.7%, Avg loss: 0.591482 

Epoch 48
-------------------------------
loss: 0.803257  [    1/   54]
loss: 0.795585  [   52/   54]
Test Error: 
 Accuracy: 66.7%, Avg loss: 0.588311 

Epoch 49
-------------------------------
loss: 0.803864  [    1/   54]
loss: 0.791629  [   52/   54]
Test Error: 
 Accuracy: 66.7%, Avg loss: 0.585167 

Epoch 50
-------------------------------
loss: 0.804453  [    1/   54]
loss: 0.787553  [   52/   54]
Test Error: 
 Accuracy: 66.7%, Avg loss: 0.582049 

Epoch 51
-------------------------------
loss: 0.805022  [    1/   54]
loss: 0.783378  [   52/   54]
Test Error: 
 Accuracy: 83.3%, Avg loss: 0.578955 

Epoch 52
-------------------------------
loss: 0.805574  [    1/   54]
```

```
loss: 0.779128  [   52/   54]
Test Error:
 Accuracy: 83.3%, Avg loss: 0.575886

Epoch 53
-------------------------------
loss: 0.806108  [    1/   54]
loss: 0.774824  [   52/   54]
Test Error:
 Accuracy: 83.3%, Avg loss: 0.572842

Epoch 54
-------------------------------
loss: 0.806623  [    1/   54]
loss: 0.770486  [   52/   54]
Test Error:
 Accuracy: 83.3%, Avg loss: 0.569824

Epoch 55
-------------------------------
loss: 0.807120  [    1/   54]
loss: 0.766131  [   52/   54]
Test Error:
 Accuracy: 83.3%, Avg loss: 0.566831

Epoch 56
-------------------------------
loss: 0.807598  [    1/   54]
loss: 0.761770  [   52/   54]
Test Error:
 Accuracy: 83.3%, Avg loss: 0.563865

Epoch 57
-------------------------------
loss: 0.808056  [    1/   54]
loss: 0.757412  [   52/   54]
Test Error:
 Accuracy: 83.3%, Avg loss: 0.560926

Epoch 58
-------------------------------
loss: 0.808496  [    1/   54]
loss: 0.753059  [   52/   54]
Test Error:
 Accuracy: 83.3%, Avg loss: 0.558014

Epoch 59
-------------------------------
loss: 0.808915  [    1/   54]
loss: 0.748713  [   52/   54]
Test Error:
 Accuracy: 83.3%, Avg loss: 0.555128

Epoch 60
-------------------------------
loss: 0.809313  [    1/   54]
loss: 0.744372  [   52/   54]
Test Error:
 Accuracy: 83.3%, Avg loss: 0.552268
```

```
Done!
```

In [ ]:
```
# If you're continuing to train from a pre-trained version:
pretrained_path = "/content/drive/MyDrive/ybigta/2023-2/win_prediction_model/winpred_mo
model = LSTMModel(input_size=num_in, hidden_size=10, num_layers=1, num_classes=2).to(de
model.load_state_dict(torch.load(pretrained_path, map_location= torch.device("cpu")), s

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001, weight_decay = 0.1)

num_epochs =
for t in range(num_epochs):
    print(f"Epoch {t+1}₩n-------------------------------")
    train_loop(train_loader, model, criterion, optimizer)
    test_loop(val_loader, model, criterion)
print("Done!")
```

```
Epoch 1
-------------------------------
loss: 0.783462  [    1/   54]
loss: 0.599811  [   52/   54]
Test Error:
 Accuracy: 83.3%, Avg loss: 0.532518

Epoch 2
-------------------------------
loss: 0.779629  [    1/   54]
loss: 0.564055  [   52/   54]
Test Error:
 Accuracy: 83.3%, Avg loss: 0.524813

Epoch 3
-------------------------------
loss: 0.777854  [    1/   54]
loss: 0.543219  [   52/   54]
Test Error:
 Accuracy: 83.3%, Avg loss: 0.520422

Epoch 4
-------------------------------
loss: 0.776490  [    1/   54]
loss: 0.530013  [   52/   54]
Test Error:
 Accuracy: 83.3%, Avg loss: 0.517872

Epoch 5
-------------------------------
loss: 0.775345  [    1/   54]
loss: 0.520610  [   52/   54]
Test Error:
 Accuracy: 83.3%, Avg loss: 0.516346

Done!
```

If you're doing a test:

Test Dataset Version

In [ ]:

```
!unzip -q /content/drive/MyDrive/ybigta/2023-2/win_prediction_model/test.zip -d /conte
```

In [ ]:

```
PATH_TO_NUMPY_ARRAY_FOLDER = "/content/test/data/"
PATH_TO_NUMPY_ARRAY_FILENAMES_AND_LABELS = "/content/test/test_label.csv"
PATH_TO_PRETRAINED_MODEL_STATE_DICT = "/content/drive/MyDrive/ybigta/2023-2/win_predict
```

In [ ]:

```
class KeypointsDataset(Dataset):
    def __init__(self, array_paths, labels, num_input):
        """
        Generates a custom numpy dataset

        Args:
            array_paths (list): list of numpy array inputs
            labels (list): list of labels
            num_input (int) : number of inputs to enter
        """
        self.array_paths = array_paths
        self.labels = labels
        self.num_input = num_input

    def __len__(self):
        """
        Returns length of entire dataset
        """
        return len(self.array_paths)

    def __getitem__(self, idx, ):
        """
        Gets the sample that corresponds to the sample id (idx)

        Args:
            idx (int): sample index

        Returns:
            keypoints (torch.Tensor): keypoints input tensor
            label (torch.Tensor): winning label tensor
        """
        keypoints_path = PATH_TO_NUMPY_ARRAY_FOLDER
        # Turn numpy arrays into torch tensors
        keypoints = np.load(f"{keypoints_path}{self.array_paths[idx]}.npy")
        keypoints = keypoints[:,-self.num_input:]
        keypoints = torch.tensor(keypoints, dtype = torch.float32)
        label = torch.tensor(self.labels[idx])

        return keypoints, label
def test_loop(dataloader, model, loss_fn):
    # Set the model to evaluation mode - important for batch normalization and dropout
    # Unnecessary in this situation but added for best practices
    model.eval()
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    test_loss, correct = 0, 0

    # Evaluating the model with torch.no_grad() ensures that no gradients are computed d
    # also serves to reduce unnecessary gradient computations and memory usage for tens
    with torch.no_grad():
        for i, (inputs, labels) in enumerate(dataloader):
          inputs = inputs.to(device)
```

```
            labels = labels.to(device)

            outputs = model(inputs)

            _, predicted = torch.max(outputs.data, 1)
            test_loss += loss_fn(outputs, labels).item()
            correct += (outputs.argmax(1) == labels).type(torch.float).sum().item()
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} \
test_array = pd.read_csv(PATH_TO_NUMPY_ARRAY_FILENAMES_AND_LABELS)
num_in = 100 # DO NOT ALTER
test_data = KeypointsDataset(
    array_paths = test_array['modified_json_filename'].reset_index(drop= True),
    labels = test_array['label'].reset_index(drop= True),
    num_input = num_in
    )
test_loader = DataLoader(test_data)
pretrained_path = PATH_TO_PRETRAINED_MODEL_STATE_DICT
model = LSTMModel(input_size=num_in, hidden_size=10, num_layers=1, num_classes=2).to(de
model.load_state_dict(torch.load(pretrained_path, map_location= torch.device("cuda")),
criterion = nn.CrossEntropyLoss()
```

In [ ]:

```
# Run this after setting all the paths, and the above cell (loading the model)
test_loop(test_loader, model, criterion)
```

```
Test Error:
 Accuracy: 50.0%, Avg loss: 0.715470
```

In [ ]:

```
print(model)
```

```
LSTMModel(
  (lstm): LSTM(100, 10, batch_first=True)
  (fc): Linear(in_features=10, out_features=2, bias=True)
)
```

Individual File Version

In [ ]:

```
!unzip -q /content/drive/MyDrive/ybigta/2023-2/win_prediction_model/test_keypoints.zip
```

```
replace /content/test_keypoints/31_person.npy? [y]es, [n]o, [A]ll, [N]one, [r]ename: A
```

In [ ]:

```
PATH_TO_NUMPY_ARRAY = "/content/test/data/lee_kim_1_person.npy"
PATH_TO_PRETRAINED_MODEL_STATE_DICT = "/content/drive/MyDrive/ybigta/2023-2/win_predict
```

In [ ]:

```
numpy_data = np.load(PATH_TO_NUMPY_ARRAY)

pretrained_path = PATH_TO_PRETRAINED_MODEL_STATE_DICT
model = LSTMModel(input_size=num_in, hidden_size=10, num_layers=1, num_classes=2).to(de
# If there's no GPU, change from device("cuda") to device("cpu")
model.load_state_dict(torch.load(pretrained_path, map_location= torch.device("cuda")),

def test(numpy_data_raw, model):
    data = torch.tensor(numpy_data_raw[:, -100:], dtype = torch.float32)
    data = data[None, :]
    data = data.to(device)
    model.eval()
    outputs = model(data)
```

```python
    _, predicted = torch.max(outputs.data, 1)
    predicted_int = predicted.type(torch.int)[0]
    print(f"The result is:{predicted_int:d}")
    return predicted_int
test(numpy_data, model)
```

```
The result is:0
```

Out[ ]:

```
tensor(0, device='cuda:0', dtype=torch.int32)
```