Exercise Sheet 1: Recurrent Models

Compare Vanilla Recurrent Neural Networks (RNN) with Long-Short Term Networks (LSTM). Implement a vanilla RNN and LSTM from scratch.

```python
import json
import os
import time
import math
import sys
import numpy as np
from numpy import sqrt
import random

import torch
import torch.nn as nn
import torch.optim as optim
import torch.utils.data as data
import torch.functional as F

# import RNN from torch
from torch.nn import RNN

import matplotlib.pyplot as plt
%matplotlib inline

plt.rcParams['figure.figsize'] = [6, 4]

# set seed for reproducibility
seed = 42
np.random.seed(seed)
torch.manual_seed(seed)

# set device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# set paths
data_path = './data/'
model_path = './model/'
results_path = './results/'

# make directories if they don't exist
if not os.path.exists(data_path):
    os.makedirs(data_path)
if not os.path.exists(model_path):
    os.makedirs(model_path)
if not os.path.exists(results_path):
    os.makedirs(results_path)
```

```python
# Data
p1 = [5, 10, 15, 20]
p2_acc_rnn = []
p3_acc_lstm = []

# Hyperparameters
config = {
    "input_length": 12,
    "input_dim": 1,
    "num_classes": 10,
    "num_hidden": 128,
    "batch_size": 128,
    "rnn_learning_rate": 0.001,
    "lstm_learning_rate": 0.001,
    "train_steps": 10000,
    "test_steps": 100,
    "max_norm": 1.0
}

# save config
with open(model_path + 'config.json', 'w') as file:
    json.dump(config, file)
```

Task 1: Toy Problem: Palindrome Numbers

Use a a recurrent neural network to predict the next digit of the palindrome at every timestep. This can become difficult for very long sequences since the network has to memorise information from very far away earlier timesteps. Goal is to study the memoization capability of recurrent networks.

```python
class PalindromeDataset(data.Dataset):
    """ Randomly generates palindromes of a given length.
        The input is the first N-1 digits of the palindrome, the
target is the last digit.
        For short palindromes, the number of possible palindromes is
limited.
    """
    def __init__(self, seq_length):
        self.seq_length = seq_length

    def __len__(self):
        # Number of possible palindroms can be very big:
        # (10**(seq_length/2) or (10**((seq_length+1)/2)
        # Therefore we return the maximum integer value
        return sys.maxsize

    def __getitem__(self, idx):
        # Keep last digit as target label. Note: one-hot encoding for
```

```
inputs is
        # more suitable for training, but this also works.
        full_palindrome = self.generate_palindrome()
        # Split palindrome into inputs (N-1 digits) and target (1
digit)
        return full_palindrome[0:-1], int(full_palindrome[-1])

    def generate_palindrome(self):
        # Generates a single, random palindrome number of 'length'
digits.
        left = [np.random.randint(0, 10) for _ in
range(math.ceil(self.seq_length / 2))]
        left = np.asarray(left, dtype=np.float32)
        right = np.flip(left, 0) if self.seq_length % 2 == 0 else
np.flip(left[:-1], 0)
        return np.concatenate((left, right))
```

Question 1.1: Backpropagation through Time

Backpropagation by computing derivatives from L_t with respect to W_{ph}

\begin{aligned} \frac{\partial L_t}{\partial W_{ph}} = \frac{\partial L_t}{\partial h_t} \frac{\partial h_t}{\partial W_{ph}} \end{aligned}

Backpropagation by computing derivatives from L_t with respect to W_{hh} \begin{aligned} \frac{\partial L_{t}}{\partial W_{hh}} = \frac{\partial L_{t}}{\partial h_{t}} \frac{\partial h_{t}}{\partial h_{t-1} } \dots \frac{\partial h_{1}}{\partial W_{hh}} \ = \frac{\partial L_{t}}{\partial h_{t}}(\prod_{t=2}^{T} \frac{\partial h_{t}}{\partial h_{t-1}})\frac{\partial h_{1}}{\partial W_{hh}} \ = \frac{\partial L_{t}}{\partial h_{t}}(\prod_{t=2}^{T} softmax^{'}(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}^{T-1})\frac{\partial h_{1}}{\partial W_{hh}} \ \end{aligned}

Difference in temporal dependence of both gradients in the first equation is noly on the current hidden state h_t, and not on the previous hidden states. The $softmax$ activated value will almost always be less than 1 because the activation is always between zero and one with a long tail on both ends. Thus, as the timestep t gets larger (i.e. longer timesteps), the gradient will descrease in value due to the repeated multiplication and get close to zero. On the other side it can equally lead to exploding gradients if the values are very large.

Question 1.2: Implement a Vanilla Recurrent Network

```
class VanillaRNN(nn.Module):

    def __init__(
        self, seq_length, input_dim, num_hidden, num_classes,
batch_size, device=None):
        super(VanillaRNN, self).__init__()
        self.seq_length = seq_length
        self.input_dim = input_dim
        self.num_hidden = num_hidden
        self.num_classes = num_classes
```

```
        self.batch_size = batch_size

        if device is None:
            self.device = torch.device('cuda' if
torch.cuda.is_available() else 'cpu')

        self.device = device

        # Define the RNN layer
        self.hidden_state = torch.zeros(self.batch_size,
self.num_hidden)
        self.W_hx = nn.Parameter(torch.Tensor(self.input_dim,
self.num_hidden))      # input to hidden
        self.W_hh = nn.Parameter(torch.Tensor(self.num_hidden,
self.num_hidden))     # hidden to hidden
        self.B_h = nn.Parameter(torch.Tensor(self.num_hidden))
# hidden bias
        # Define the output layer
        self.W_ph = nn.Parameter(torch.Tensor(self.num_hidden,
self.num_classes))    # hidden to output
        self.B_y = nn.Parameter(torch.Tensor(self.num_classes))
# output bias

        # Initialize weights
        self.init_weights()

    def forward(self, x):
        # Initialize hidden state
        h_t = torch.zeros(self.num_hidden)

        for t in range(self.seq_length): # iterate over the time steps
            x_t = x[:, t].view(128,-1)
            h_t = torch.tanh(x_t @ self.W_hx + h_t @ self.W_hh +
self.B_h)

        output = h_t @ self.W_ph + self.B_y
        y = torch.softmax(output, dim=1)
        return y

    def init_weights(self):
        """ Initialize weights to avoid gradients vanishing or
exploding.
            Source: https://dennybritz.com/posts/wildml/recurrent-
neural-networks-tutorial-part-2/

        """
        # Initialize weights with uniform distribution
        n_hx = self.W_hx.size(0)  # number of incoming connections for
W_hx
        nn.init.uniform_(self.W_hx, -1 / sqrt(n_hx), 1 / sqrt(n_hx))
```

```python
        n_hh = self.W_hh.size(0)  # number of incoming connections for
W_hh
        nn.init.uniform_(self.W_hh, -1 / sqrt(n_hh), 1 / sqrt(n_hh))

        n_ph = self.W_ph.size(0)  # number of incoming connections for
W_ph
        nn.init.uniform_(self.W_ph, -1 / sqrt(n_ph), 1 / sqrt(n_ph))

        # Initialize biases to zeros
        nn.init.zeros_(self.B_h)
        nn.init.zeros_(self.B_y)


    def set_grad(self, requires_grad):
        # Set requires_grad for all parameters
        for param in self.parameters():
            param.requires_grad = requires_grad

def compute_accuracy(outputs, targets):
    """ Compute the accuracy of the model's predictions."""
    # Compute accuracy of outputs compared to targets
    _, predicted = torch.max(outputs, 1)
    correct = predicted.eq(targets)
    return 100 * correct.sum().item() / targets.size(0)


def train(config:json, input_length=5, lr=0.001, step_size=1000,
gamma=0.1, type='RNN', opt='Adam', device=None):
    """ Train the model on the training set.
        Returns the trained model, losses and accuracies.
    """
    if device is None:
        device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')

    if input_length == 0:
        input_length = config['input_length']

    # Initialize the model that we are going to use
    if type == 'RNN':
        model = VanillaRNN(input_length, config['input_dim'],
config['num_hidden'], config['num_classes'], config['batch_size'])
    elif type == 'LSTM':
        model = LSTM(input_length, config['input_dim'],
config['num_hidden'], config['num_classes'], config['batch_size'])
    else:
        raise ValueError('Model type not supported')

    model.to(device)
    model.train()
```

```
    # Initialize the dataset and data loader (note the +1)
    dataset = PalindromeDataset(input_length + 1)
    data_loader = data.DataLoader(dataset, config['batch_size'],
num_workers=0)

    # Define the loss function and optimizer
    criterion = nn.CrossEntropyLoss()

    # Define optimizer
    if opt == 'Adam':
        optimizer = optim.Adam(model.parameters(), lr=lr)
    elif opt == 'RMSprop':
        optimizer = optim.RMSprop(model.parameters(), lr=lr)
        scheduler = optim.lr_scheduler.StepLR(optimizer,
step_size=step_size, gamma=gamma)
    else:
        raise ValueError('Optimizer not supported')

    # Train the model
    losses = []
    accuracies = []
    loss = 0.0

    for i, (inputs, targets) in enumerate(data_loader, 0):

        # Only for time measurement of step through network
        t1 = time.time()
        inputs = inputs.to(device)
        targets = targets.to(device)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Update learning rate
        if opt == 'RMSprop':
            scheduler.step()

        # Forward pass, backward pass, and optimize
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()

        # Clip gradients to prevent exploding gradients
        nn.utils.clip_grad_norm(model.parameters(),
max_norm=config['max_norm'])

        loss += loss.item()
        accuracy = 0.0
```

```
        # Print statistics
        if i % 100 == 0:
            # Just for time measurement
            t2 = time.time()
            # print accuracy/loss here
            accuracy = compute_accuracy(outputs, targets)
            accuracies.append(accuracy)
            print('[step: %5d] loss: %.4f acc: %.4f time: %5d' %
                          (i, loss / 100, accuracy, t2-t1 / 100))
            losses.append(loss.detach().numpy() / 100)
            loss = 0.0

        if i == config['train_steps']:
            # If you receive a PyTorch data-loader error, check this
bug report:
            # https://github.com/pytorch/pytorch/pull/9655
            break

    print('Finished Training')
    return model, losses, accuracies


def test(model, config:json, input_length=5, device=None):
    """ Test the model on the test set.
        Returns the accuracies.
    """
    if device is None:
        device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')

    if input_length == 0:
        input_length = config['input_length']

    # Initialize the dataset and data loader (leave the +1)
    dataset = PalindromeDataset(input_length+1)
    data_loader = data.DataLoader(dataset, config['batch_size'],
num_workers=0)

    model.to(device)
    model.eval()

    # Test the model
    accuracies = []

    with torch.no_grad():
        for i, (inputs, targets) in enumerate(data_loader, 0):
            inputs = inputs.to(device)
            targets = targets.to(device)
```

```
            outputs = model(inputs)
            accuracy = 0.0
            if i % 10 == 0:
                accuracy = compute_accuracy(outputs, targets)
                accuracies.append(accuracy)
                print('Accuracy: %.4f' % accuracy)

            if i == config['test_steps']:
              # If you receive a PyTorch data-loader error, check this
bug report:
              # https://github.com/pytorch/pytorch/pull/9655
              break

    print('Finished Testing')
    return accuracies

# Load the configuration
with open(model_path + 'config.json', 'r') as file:
    config = json.load(file)

def plot_loss(losses, title='Training Loss', path=None):
    """ Plot the losses of the model."""
    if path is None:
        path = results_path + 'training_loss.png'
    plt.figure(figsize=(6,4))
    plt.plot(losses)
    plt.xlabel('Steps')
    plt.ylabel('Loss')
    plt.title(title)
    plt.savefig(path)
    plt.show()

def plot_accuracy(accuracies, title='Training Accuracy', path=None):
    """ Plot the accuracies of the model."""
    if path is None:
        path = results_path + 'training_accuracy.png'
    plt.figure(figsize=(6,4))
    plt.plot(accuracies)
    plt.xlabel('Steps')
    plt.ylabel('Accuracy')
    plt.title(title)
    plt.savefig(path)
    plt.show()
```

Question 1.3: Train RNN with varying T for sequence lengths

- Train and evaluate model on Palindromes with length N = 5

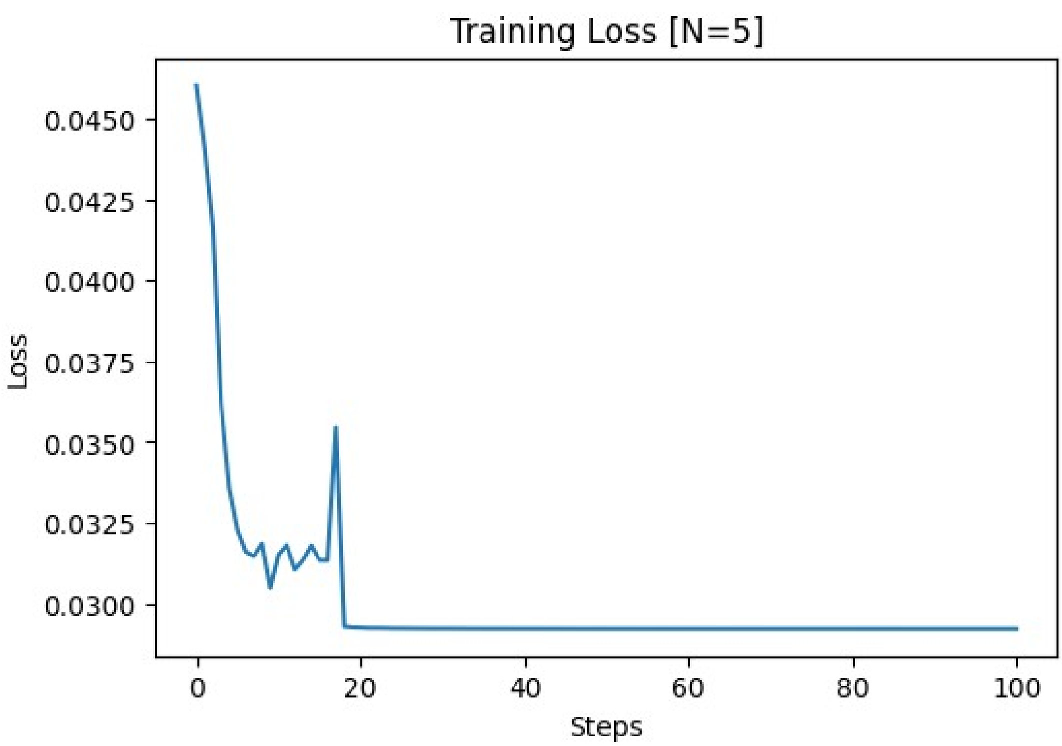
```
# Train the model on T=5
model, losses, accuracies = train(config, input_length=p1[0],
lr=config['rnn_learning_rate'], type='RNN', device=device)
```

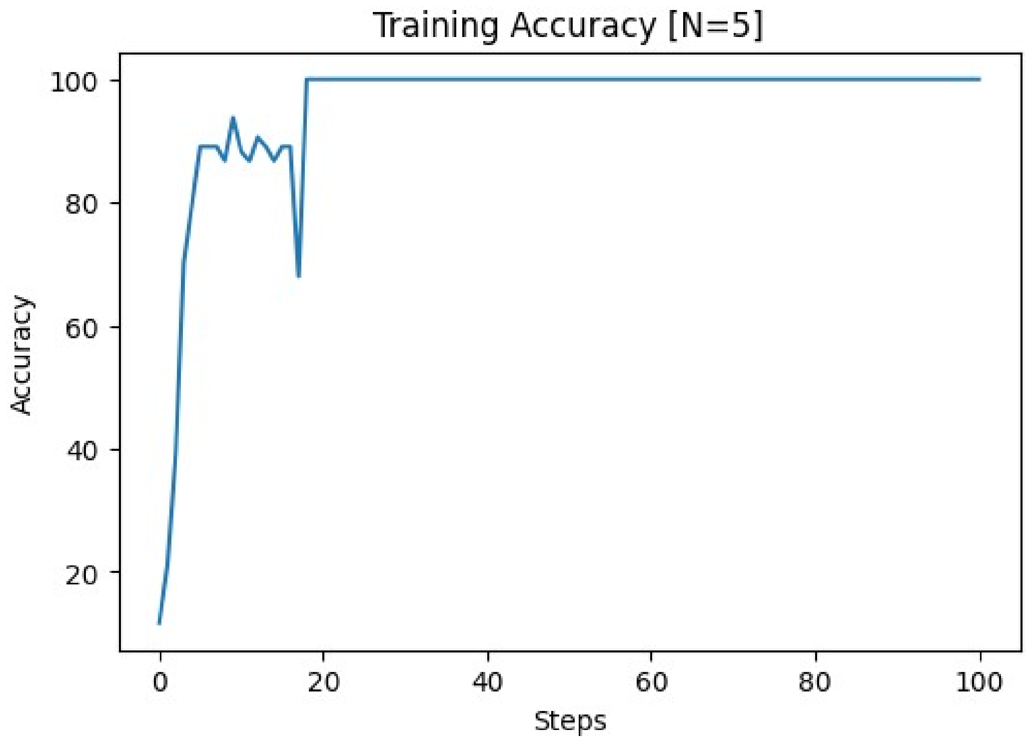
```
<ipython-input-6-042aa7a8086d>:64: UserWarning:
torch.nn.utils.clip_grad_norm is now deprecated in favor of
torch.nn.utils.clip_grad_norm_.
  nn.utils.clip_grad_norm(model.parameters(),
max_norm=config['max_norm'])
```

```
[step:     0] loss: 0.0460 acc: 11.7188 time: 1702117320
[step:   100] loss: 0.0441 acc: 21.0938 time: 1702117320
[step:   200] loss: 0.0416 acc: 39.0625 time: 1702117321
[step:   300] loss: 0.0362 acc: 70.3125 time: 1702117322
[step:   400] loss: 0.0336 acc: 79.6875 time: 1702117323
[step:   500] loss: 0.0323 acc: 89.0625 time: 1702117324
[step:   600] loss: 0.0316 acc: 89.0625 time: 1702117325
[step:   700] loss: 0.0315 acc: 89.0625 time: 1702117325
[step:   800] loss: 0.0319 acc: 86.7188 time: 1702117326
[step:   900] loss: 0.0305 acc: 93.7500 time: 1702117328
[step:  1000] loss: 0.0315 acc: 88.2812 time: 1702117329
[step:  1100] loss: 0.0318 acc: 86.7188 time: 1702117330
[step:  1200] loss: 0.0311 acc: 90.6250 time: 1702117331
[step:  1300] loss: 0.0314 acc: 89.0625 time: 1702117331
[step:  1400] loss: 0.0318 acc: 86.7188 time: 1702117332
[step:  1500] loss: 0.0313 acc: 89.0625 time: 1702117333
[step:  1600] loss: 0.0313 acc: 89.0625 time: 1702117334
[step:  1700] loss: 0.0355 acc: 67.9688 time: 1702117335
[step:  1800] loss: 0.0293 acc: 100.0000 time: 1702117335
[step:  1900] loss: 0.0293 acc: 100.0000 time: 1702117336
[step:  2000] loss: 0.0293 acc: 100.0000 time: 1702117337
[step:  2100] loss: 0.0292 acc: 100.0000 time: 1702117338
[step:  2200] loss: 0.0292 acc: 100.0000 time: 1702117339
[step:  2300] loss: 0.0292 acc: 100.0000 time: 1702117340
[step:  2400] loss: 0.0292 acc: 100.0000 time: 1702117341
[step:  2500] loss: 0.0292 acc: 100.0000 time: 1702117342
[step:  2600] loss: 0.0292 acc: 100.0000 time: 1702117343
[step:  2700] loss: 0.0292 acc: 100.0000 time: 1702117344
[step:  2800] loss: 0.0292 acc: 100.0000 time: 1702117344
[step:  2900] loss: 0.0292 acc: 100.0000 time: 1702117345
[step:  3000] loss: 0.0292 acc: 100.0000 time: 1702117346
[step:  3100] loss: 0.0292 acc: 100.0000 time: 1702117347
[step:  3200] loss: 0.0292 acc: 100.0000 time: 1702117347
[step:  3300] loss: 0.0292 acc: 100.0000 time: 1702117349
[step:  3400] loss: 0.0292 acc: 100.0000 time: 1702117350
[step:  3500] loss: 0.0292 acc: 100.0000 time: 1702117351
[step:  3600] loss: 0.0292 acc: 100.0000 time: 1702117352
[step:  3700] loss: 0.0292 acc: 100.0000 time: 1702117353
[step:  3800] loss: 0.0292 acc: 100.0000 time: 1702117354
[step:  3900] loss: 0.0292 acc: 100.0000 time: 1702117355
[step:  4000] loss: 0.0292 acc: 100.0000 time: 1702117356
[step:  4100] loss: 0.0292 acc: 100.0000 time: 1702117356
[step:  4200] loss: 0.0292 acc: 100.0000 time: 1702117357
[step:  4300] loss: 0.0292 acc: 100.0000 time: 1702117358
```

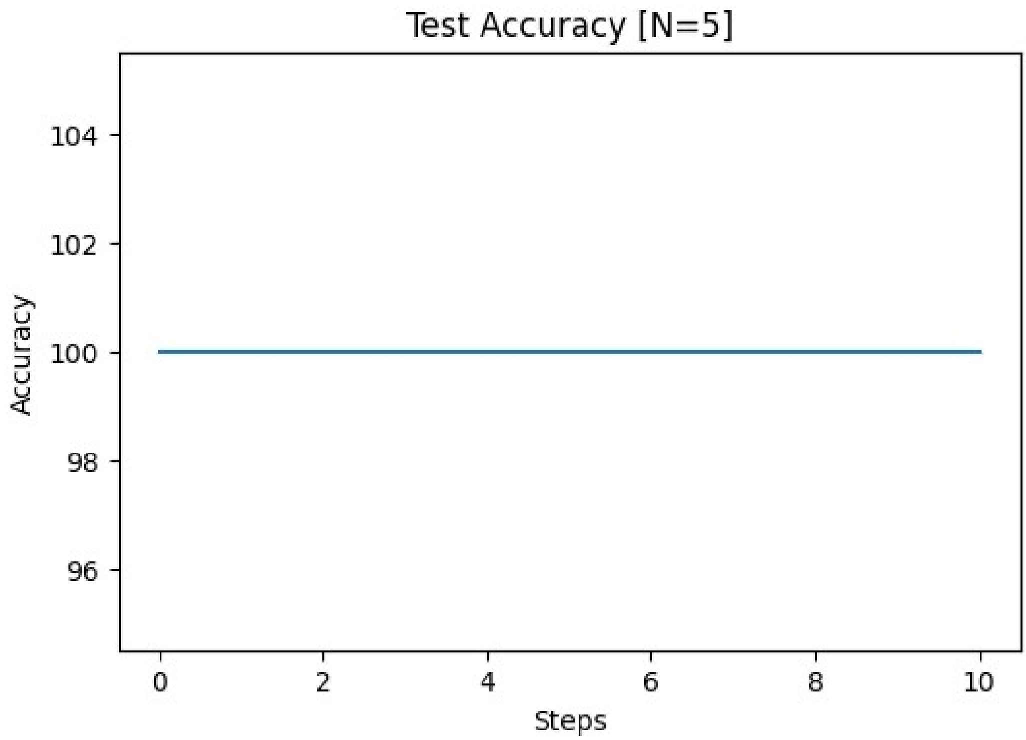
```
[step:  4400] loss: 0.0292 acc: 100.0000 time: 1702117359
[step:  4500] loss: 0.0292 acc: 100.0000 time: 1702117359
[step:  4600] loss: 0.0292 acc: 100.0000 time: 1702117360
[step:  4700] loss: 0.0292 acc: 100.0000 time: 1702117361
[step:  4800] loss: 0.0292 acc: 100.0000 time: 1702117362
[step:  4900] loss: 0.0292 acc: 100.0000 time: 1702117363
[step:  5000] loss: 0.0292 acc: 100.0000 time: 1702117363
[step:  5100] loss: 0.0292 acc: 100.0000 time: 1702117364
[step:  5200] loss: 0.0292 acc: 100.0000 time: 1702117365
[step:  5300] loss: 0.0292 acc: 100.0000 time: 1702117367
[step:  5400] loss: 0.0292 acc: 100.0000 time: 1702117367
[step:  5500] loss: 0.0292 acc: 100.0000 time: 1702117368
[step:  5600] loss: 0.0292 acc: 100.0000 time: 1702117369
[step:  5700] loss: 0.0292 acc: 100.0000 time: 1702117370
[step:  5800] loss: 0.0292 acc: 100.0000 time: 1702117371
[step:  5900] loss: 0.0292 acc: 100.0000 time: 1702117371
[step:  6000] loss: 0.0292 acc: 100.0000 time: 1702117372
[step:  6100] loss: 0.0292 acc: 100.0000 time: 1702117373
[step:  6200] loss: 0.0292 acc: 100.0000 time: 1702117374
[step:  6300] loss: 0.0292 acc: 100.0000 time: 1702117375
[step:  6400] loss: 0.0292 acc: 100.0000 time: 1702117375
[step:  6500] loss: 0.0292 acc: 100.0000 time: 1702117376
[step:  6600] loss: 0.0292 acc: 100.0000 time: 1702117378
[step:  6700] loss: 0.0292 acc: 100.0000 time: 1702117379
[step:  6800] loss: 0.0292 acc: 100.0000 time: 1702117380
[step:  6900] loss: 0.0292 acc: 100.0000 time: 1702117380
[step:  7000] loss: 0.0292 acc: 100.0000 time: 1702117381
[step:  7100] loss: 0.0292 acc: 100.0000 time: 1702117382
[step:  7200] loss: 0.0292 acc: 100.0000 time: 1702117383
[step:  7300] loss: 0.0292 acc: 100.0000 time: 1702117384
[step:  7400] loss: 0.0292 acc: 100.0000 time: 1702117384
[step:  7500] loss: 0.0292 acc: 100.0000 time: 1702117385
[step:  7600] loss: 0.0292 acc: 100.0000 time: 1702117386
[step:  7700] loss: 0.0292 acc: 100.0000 time: 1702117387
[step:  7800] loss: 0.0292 acc: 100.0000 time: 1702117387
[step:  7900] loss: 0.0292 acc: 100.0000 time: 1702117388
[step:  8000] loss: 0.0292 acc: 100.0000 time: 1702117389
[step:  8100] loss: 0.0292 acc: 100.0000 time: 1702117390
[step:  8200] loss: 0.0292 acc: 100.0000 time: 1702117392
[step:  8300] loss: 0.0292 acc: 100.0000 time: 1702117392
[step:  8400] loss: 0.0292 acc: 100.0000 time: 1702117393
[step:  8500] loss: 0.0292 acc: 100.0000 time: 1702117394
[step:  8600] loss: 0.0292 acc: 100.0000 time: 1702117395
[step:  8700] loss: 0.0292 acc: 100.0000 time: 1702117395
[step:  8800] loss: 0.0292 acc: 100.0000 time: 1702117396
[step:  8900] loss: 0.0292 acc: 100.0000 time: 1702117397
[step:  9000] loss: 0.0292 acc: 100.0000 time: 1702117398
[step:  9100] loss: 0.0292 acc: 100.0000 time: 1702117399
[step:  9200] loss: 0.0292 acc: 100.0000 time: 1702117400
```

```
[step:  9300] loss: 0.0292 acc: 100.0000 time: 1702117400
[step:  9400] loss: 0.0292 acc: 100.0000 time: 1702117401
[step:  9500] loss: 0.0292 acc: 100.0000 time: 1702117403
[step:  9600] loss: 0.0292 acc: 100.0000 time: 1702117404
[step:  9700] loss: 0.0292 acc: 100.0000 time: 1702117405
[step:  9800] loss: 0.0292 acc: 100.0000 time: 1702117405
[step:  9900] loss: 0.0292 acc: 100.0000 time: 1702117406
[step: 10000] loss: 0.0292 acc: 100.0000 time: 1702117407
Finished Training

# Plot the losses
plot_loss(losses, title='Training Loss [N=5]', path=results_path +
'training_loss_5_rnn.png')

# Plot accuracies
plot_accuracy(accuracies, title='Training Accuracy [N=5]',
path=results_path + 'training_accuracy_5_rnn.png')
```






```
# Test the model
test_accuracies = test(model, input_length=p1[0], config=config,
device=device)

# Add accuracies
p2_acc_rnn.append(np.mean(test_accuracies))

Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Finished Testing

# plot the test accuracies
plot_accuracy(test_accuracies, title='Test Accuracy [N=5]',
path=results_path + 'test_accuracy_5_rnn.png')

# Average accuracy over all Steps
print(f"Average test accuracy: {np.mean(test_accuracies):.2f}%")
```



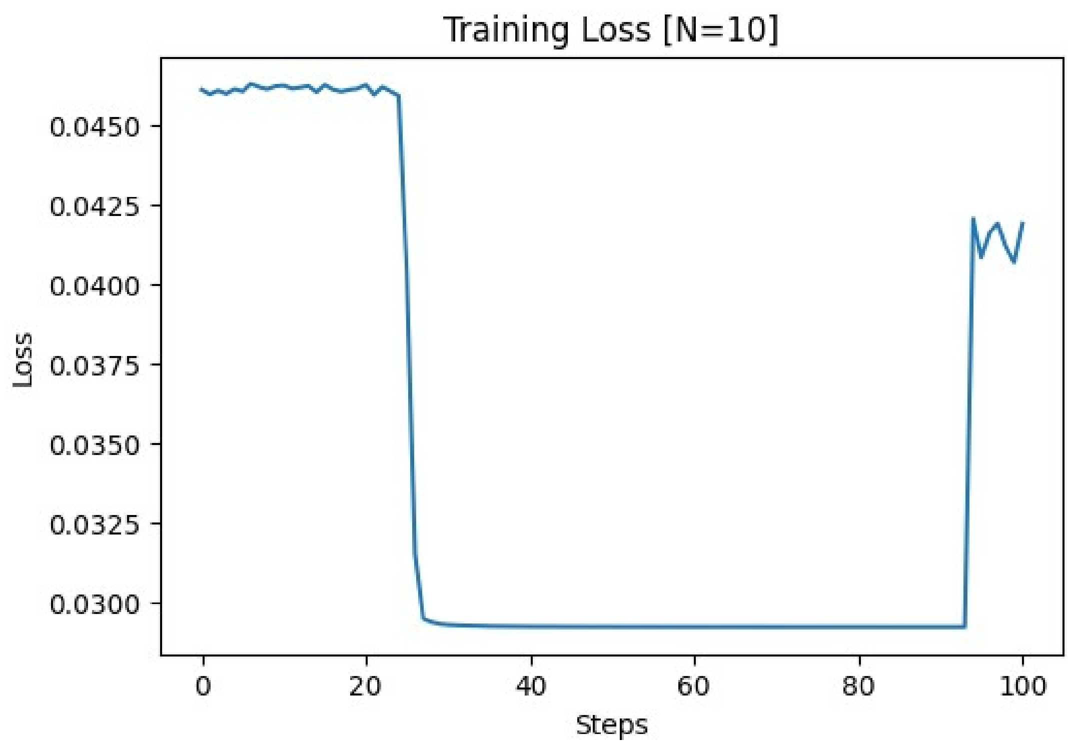

```
Average test accuracy: 100.00%
```

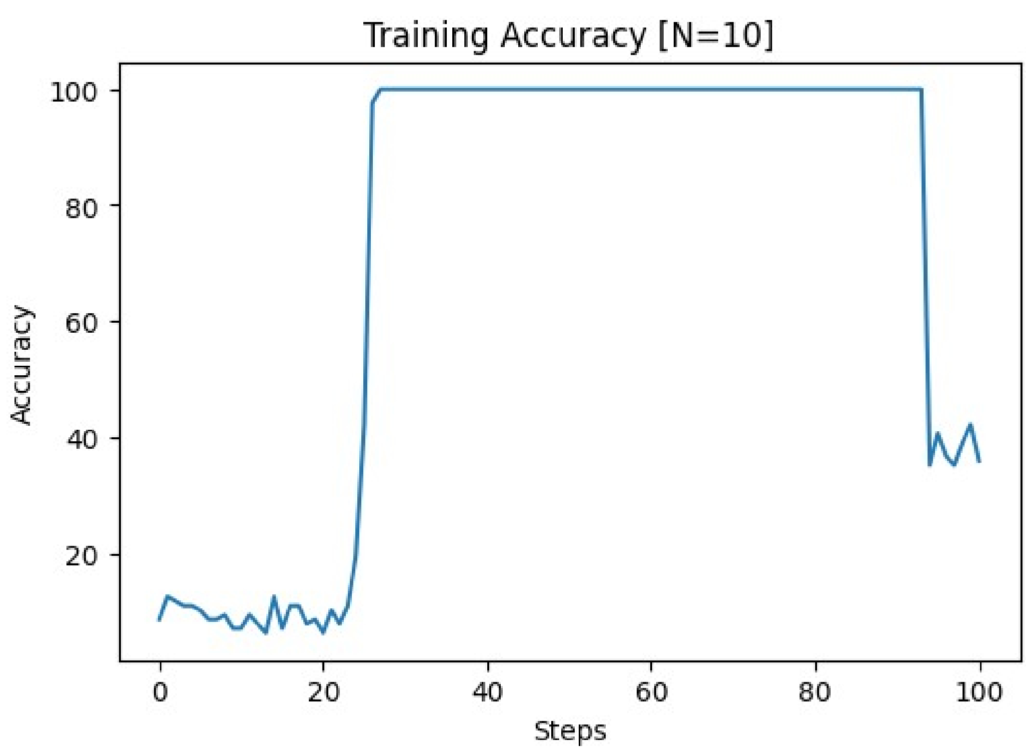
Task 2: Vanilla RNN in PyTorch

```
# Train the model on T=10
model, losses, accuracies = train(config, input_length=p1[1],
lr=config['rnn_learning_rate'], type='RNN', device=device)

[step:     0] loss: 0.0461 acc: 8.5938 time: 1702117408

<ipython-input-6-042aa7a8086d>:64: UserWarning:
torch.nn.utils.clip_grad_norm is now deprecated in favor of
torch.nn.utils.clip_grad_norm_.
  nn.utils.clip_grad_norm(model.parameters(),
max_norm=config['max_norm'])

[step:   100] loss: 0.0460 acc: 12.5000 time: 1702117410
[step:   200] loss: 0.0461 acc: 11.7188 time: 1702117411
[step:   300] loss: 0.0460 acc: 10.9375 time: 1702117413
[step:   400] loss: 0.0461 acc: 10.9375 time: 1702117414
[step:   500] loss: 0.0460 acc: 10.1562 time: 1702117416
[step:   600] loss: 0.0463 acc: 8.5938 time: 1702117417
[step:   700] loss: 0.0462 acc: 8.5938 time: 1702117419
[step:   800] loss: 0.0461 acc: 9.3750 time: 1702117420
[step:   900] loss: 0.0462 acc: 7.0312 time: 1702117421
[step:  1000] loss: 0.0462 acc: 7.0312 time: 1702117422
[step:  1100] loss: 0.0461 acc: 9.3750 time: 1702117424
```

```
[step:  1200] loss: 0.0462 acc: 7.8125 time: 1702117425
[step:  1300] loss: 0.0462 acc: 6.2500 time: 1702117426
[step:  1400] loss: 0.0460 acc: 12.5000 time: 1702117428
[step:  1500] loss: 0.0463 acc: 7.0312 time: 1702117429
[step:  1600] loss: 0.0461 acc: 10.9375 time: 1702117430
[step:  1700] loss: 0.0460 acc: 10.9375 time: 1702117432
[step:  1800] loss: 0.0461 acc: 7.8125 time: 1702117433
[step:  1900] loss: 0.0461 acc: 8.5938 time: 1702117434
[step:  2000] loss: 0.0463 acc: 6.2500 time: 1702117435
[step:  2100] loss: 0.0459 acc: 10.1562 time: 1702117436
[step:  2200] loss: 0.0462 acc: 7.8125 time: 1702117438
[step:  2300] loss: 0.0461 acc: 10.9375 time: 1702117440
[step:  2400] loss: 0.0459 acc: 19.5312 time: 1702117442
[step:  2500] loss: 0.0404 acc: 42.1875 time: 1702117443
[step:  2600] loss: 0.0315 acc: 97.6562 time: 1702117444
[step:  2700] loss: 0.0295 acc: 100.0000 time: 1702117445
[step:  2800] loss: 0.0294 acc: 100.0000 time: 1702117446
[step:  2900] loss: 0.0293 acc: 100.0000 time: 1702117448
[step:  3000] loss: 0.0293 acc: 100.0000 time: 1702117449
[step:  3100] loss: 0.0293 acc: 100.0000 time: 1702117450
[step:  3200] loss: 0.0293 acc: 100.0000 time: 1702117451
[step:  3300] loss: 0.0293 acc: 100.0000 time: 1702117453
[step:  3400] loss: 0.0293 acc: 100.0000 time: 1702117454
[step:  3500] loss: 0.0292 acc: 100.0000 time: 1702117456
[step:  3600] loss: 0.0292 acc: 100.0000 time: 1702117457
[step:  3700] loss: 0.0292 acc: 100.0000 time: 1702117458
[step:  3800] loss: 0.0292 acc: 100.0000 time: 1702117459
[step:  3900] loss: 0.0292 acc: 100.0000 time: 1702117460
[step:  4000] loss: 0.0292 acc: 100.0000 time: 1702117462
[step:  4100] loss: 0.0292 acc: 100.0000 time: 1702117463
[step:  4200] loss: 0.0292 acc: 100.0000 time: 1702117465
[step:  4300] loss: 0.0292 acc: 100.0000 time: 1702117466
[step:  4400] loss: 0.0292 acc: 100.0000 time: 1702117467
[step:  4500] loss: 0.0292 acc: 100.0000 time: 1702117469
[step:  4600] loss: 0.0292 acc: 100.0000 time: 1702117470
[step:  4700] loss: 0.0292 acc: 100.0000 time: 1702117471
[step:  4800] loss: 0.0292 acc: 100.0000 time: 1702117472
[step:  4900] loss: 0.0292 acc: 100.0000 time: 1702117473
[step:  5000] loss: 0.0292 acc: 100.0000 time: 1702117475
[step:  5100] loss: 0.0292 acc: 100.0000 time: 1702117476
[step:  5200] loss: 0.0292 acc: 100.0000 time: 1702117478
[step:  5300] loss: 0.0292 acc: 100.0000 time: 1702117479
[step:  5400] loss: 0.0292 acc: 100.0000 time: 1702117480
[step:  5500] loss: 0.0292 acc: 100.0000 time: 1702117481
[step:  5600] loss: 0.0292 acc: 100.0000 time: 1702117483
[step:  5700] loss: 0.0292 acc: 100.0000 time: 1702117484
[step:  5800] loss: 0.0292 acc: 100.0000 time: 1702117485
[step:  5900] loss: 0.0292 acc: 100.0000 time: 1702117486
[step:  6000] loss: 0.0292 acc: 100.0000 time: 1702117488
```

```
[step:  6100] loss: 0.0292 acc: 100.0000 time: 1702117490
[step:  6200] loss: 0.0292 acc: 100.0000 time: 1702117491
[step:  6300] loss: 0.0292 acc: 100.0000 time: 1702117492
[step:  6400] loss: 0.0292 acc: 100.0000 time: 1702117493
[step:  6500] loss: 0.0292 acc: 100.0000 time: 1702117494
[step:  6600] loss: 0.0292 acc: 100.0000 time: 1702117496
[step:  6700] loss: 0.0292 acc: 100.0000 time: 1702117497
[step:  6800] loss: 0.0292 acc: 100.0000 time: 1702117498
[step:  6900] loss: 0.0292 acc: 100.0000 time: 1702117499
[step:  7000] loss: 0.0292 acc: 100.0000 time: 1702117501
[step:  7100] loss: 0.0292 acc: 100.0000 time: 1702117503
[step:  7200] loss: 0.0292 acc: 100.0000 time: 1702117504
[step:  7300] loss: 0.0292 acc: 100.0000 time: 1702117505
[step:  7400] loss: 0.0292 acc: 100.0000 time: 1702117506
[step:  7500] loss: 0.0292 acc: 100.0000 time: 1702117508
[step:  7600] loss: 0.0292 acc: 100.0000 time: 1702117509
[step:  7700] loss: 0.0292 acc: 100.0000 time: 1702117510
[step:  7800] loss: 0.0292 acc: 100.0000 time: 1702117511
[step:  7900] loss: 0.0292 acc: 100.0000 time: 1702117513
[step:  8000] loss: 0.0292 acc: 100.0000 time: 1702117514
[step:  8100] loss: 0.0292 acc: 100.0000 time: 1702117516
[step:  8200] loss: 0.0292 acc: 100.0000 time: 1702117517
[step:  8300] loss: 0.0292 acc: 100.0000 time: 1702117518
[step:  8400] loss: 0.0292 acc: 100.0000 time: 1702117519
[step:  8500] loss: 0.0292 acc: 100.0000 time: 1702117520
[step:  8600] loss: 0.0292 acc: 100.0000 time: 1702117522
[step:  8700] loss: 0.0292 acc: 100.0000 time: 1702117523
[step:  8800] loss: 0.0292 acc: 100.0000 time: 1702117524
[step:  8900] loss: 0.0292 acc: 100.0000 time: 1702117526
[step:  9000] loss: 0.0292 acc: 100.0000 time: 1702117527
[step:  9100] loss: 0.0292 acc: 100.0000 time: 1702117528
[step:  9200] loss: 0.0292 acc: 100.0000 time: 1702117530
[step:  9300] loss: 0.0292 acc: 100.0000 time: 1702117531
[step:  9400] loss: 0.0420 acc: 35.1562 time: 1702117532
[step:  9500] loss: 0.0408 acc: 40.6250 time: 1702117533
[step:  9600] loss: 0.0416 acc: 36.7188 time: 1702117534
[step:  9700] loss: 0.0419 acc: 35.1562 time: 1702117535
[step:  9800] loss: 0.0412 acc: 39.0625 time: 1702117537
[step:  9900] loss: 0.0407 acc: 42.1875 time: 1702117539
[step: 10000] loss: 0.0419 acc: 35.9375 time: 1702117540
Finished Training

# Plot the losses
plot_loss(losses, title='Training Loss [N=10]', path=results_path +
'training_loss_10_rnn.png')

# Plot the accuracies
plot_accuracy(accuracies, title='Training Accuracy [N=10]',
path=results_path + 'training_accuracy_10_rnn.png')
```

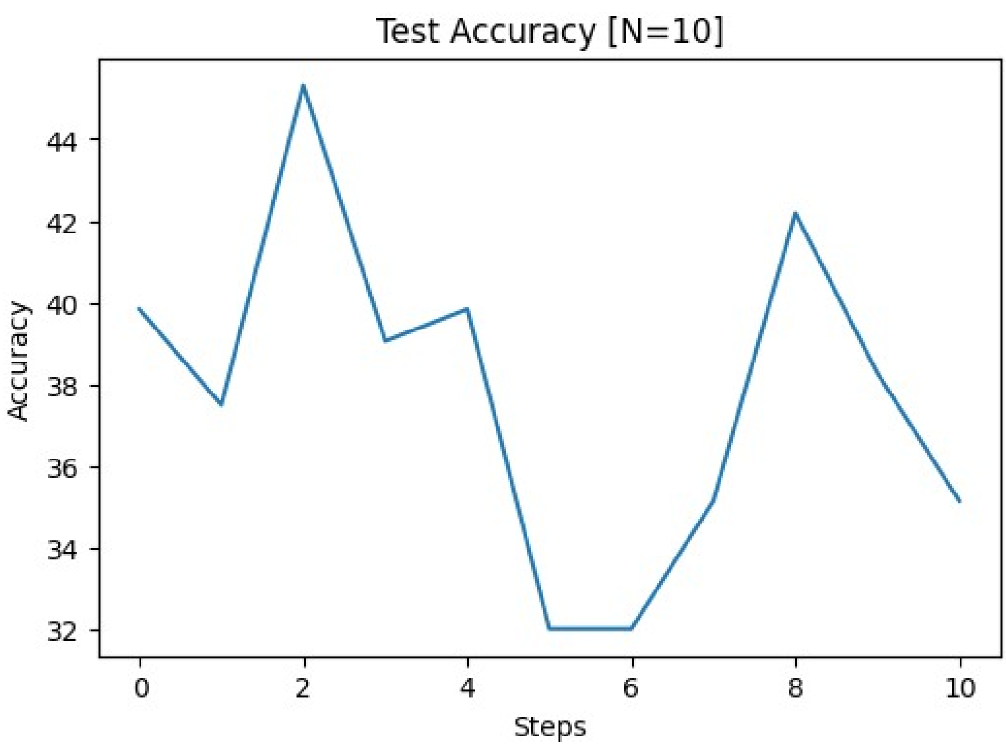





```
# Test the model
test_accuracies = test(model, input_length=p1[1], config=config,
device=device)
```

```
Accuracy: 39.8438
Accuracy: 37.5000
Accuracy: 45.3125
Accuracy: 39.0625
Accuracy: 39.8438
Accuracy: 32.0312
Accuracy: 32.0312
Accuracy: 35.1562
Accuracy: 42.1875
Accuracy: 38.2812
Accuracy: 35.1562
Finished Testing
```

```python
# Add accuracies
p2_acc_rnn.append(np.mean(test_accuracies))

# plot the test accuracies
plot_accuracy(test_accuracies, title='Test Accuracy [N=10]',
path=results_path + 'test_accuracy_10_rnn.png')

# Average accuracy over all Steps
print(f"Average test accuracy: {np.mean(test_accuracies):.2f}%")
```




```
Average test accuracy: 37.86%
```

```python
# Train the model on T=15
model, losses, accuracies = train(config, input_length=p1[2],
lr=config['rnn_learning_rate'], type='RNN', device=device)

# Plot the losses
plot_loss(losses, title='Training Loss [N=15]', path=results_path +
'training_loss_15_rnn.png')

# Plot the accuracies
plot_accuracy(accuracies, title='Training Accuracy [N=15]',
path=results_path + 'training_accuracy_15_rnn.png')
```

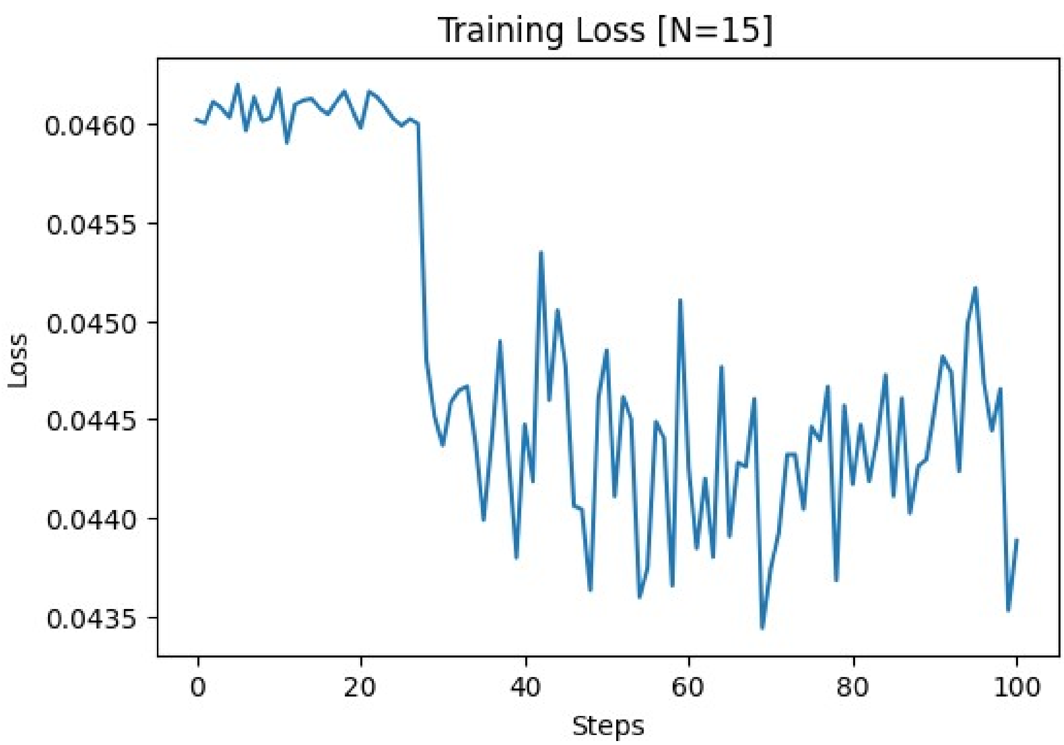
```
[step:     0] loss: 0.0460 acc: 12.5000 time: 1702117542
```

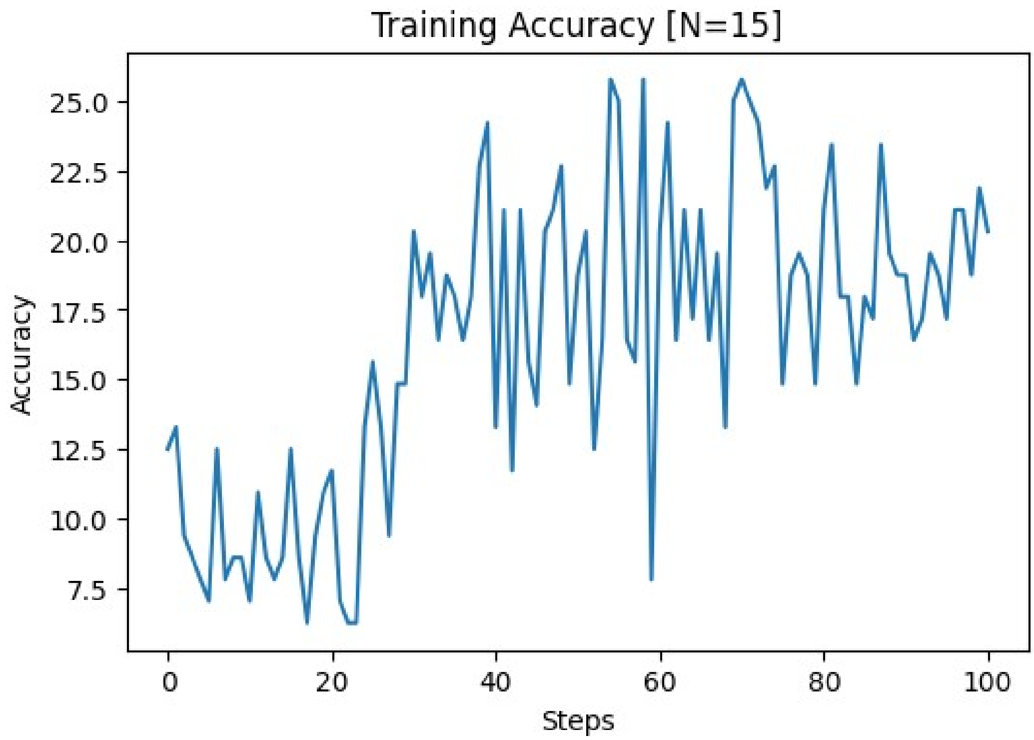
```
<ipython-input-6-042aa7a8086d>:64: UserWarning:
torch.nn.utils.clip_grad_norm is now deprecated in favor of
torch.nn.utils.clip_grad_norm_.
  nn.utils.clip_grad_norm(model.parameters(),
max_norm=config['max_norm'])
```

```
[step:   100] loss: 0.0460 acc: 13.2812 time: 1702117543
[step:   200] loss: 0.0461 acc: 9.3750 time: 1702117545
[step:   300] loss: 0.0461 acc: 8.5938 time: 1702117546
[step:   400] loss: 0.0460 acc: 7.8125 time: 1702117547
[step:   500] loss: 0.0462 acc: 7.0312 time: 1702117549
[step:   600] loss: 0.0460 acc: 12.5000 time: 1702117551
[step:   700] loss: 0.0461 acc: 7.8125 time: 1702117553
[step:   800] loss: 0.0460 acc: 8.5938 time: 1702117554
[step:   900] loss: 0.0460 acc: 8.5938 time: 1702117556
[step:  1000] loss: 0.0462 acc: 7.0312 time: 1702117557
[step:  1100] loss: 0.0459 acc: 10.9375 time: 1702117559
[step:  1200] loss: 0.0461 acc: 8.5938 time: 1702117560
[step:  1300] loss: 0.0461 acc: 7.8125 time: 1702117562
[step:  1400] loss: 0.0461 acc: 8.5938 time: 1702117564
[step:  1500] loss: 0.0461 acc: 12.5000 time: 1702117566
[step:  1600] loss: 0.0460 acc: 8.5938 time: 1702117567
[step:  1700] loss: 0.0461 acc: 6.2500 time: 1702117569
[step:  1800] loss: 0.0462 acc: 9.3750 time: 1702117571
[step:  1900] loss: 0.0461 acc: 10.9375 time: 1702117572
[step:  2000] loss: 0.0460 acc: 11.7188 time: 1702117574
[step:  2100] loss: 0.0462 acc: 7.0312 time: 1702117576
[step:  2200] loss: 0.0461 acc: 6.2500 time: 1702117578
[step:  2300] loss: 0.0461 acc: 6.2500 time: 1702117579
[step:  2400] loss: 0.0460 acc: 13.2812 time: 1702117581
[step:  2500] loss: 0.0460 acc: 15.6250 time: 1702117582
[step:  2600] loss: 0.0460 acc: 13.2812 time: 1702117584
[step:  2700] loss: 0.0460 acc: 9.3750 time: 1702117587
[step:  2800] loss: 0.0448 acc: 14.8438 time: 1702117589
[step:  2900] loss: 0.0445 acc: 14.8438 time: 1702117590
[step:  3000] loss: 0.0444 acc: 20.3125 time: 1702117592
```

```
[step:  3100] loss: 0.0446 acc: 17.9688 time: 1702117593
[step:  3200] loss: 0.0446 acc: 19.5312 time: 1702117595
[step:  3300] loss: 0.0447 acc: 16.4062 time: 1702117596
[step:  3400] loss: 0.0444 acc: 18.7500 time: 1702117598
[step:  3500] loss: 0.0440 acc: 17.9688 time: 1702117600
[step:  3600] loss: 0.0444 acc: 16.4062 time: 1702117602
[step:  3700] loss: 0.0449 acc: 17.9688 time: 1702117603
[step:  3800] loss: 0.0443 acc: 22.6562 time: 1702117605
[step:  3900] loss: 0.0438 acc: 24.2188 time: 1702117606
[step:  4000] loss: 0.0445 acc: 13.2812 time: 1702117608
[step:  4100] loss: 0.0442 acc: 21.0938 time: 1702117609
[step:  4200] loss: 0.0453 acc: 11.7188 time: 1702117612
[step:  4300] loss: 0.0446 acc: 21.0938 time: 1702117613
[step:  4400] loss: 0.0451 acc: 15.6250 time: 1702117615
[step:  4500] loss: 0.0448 acc: 14.0625 time: 1702117616
[step:  4600] loss: 0.0441 acc: 20.3125 time: 1702117618
[step:  4700] loss: 0.0440 acc: 21.0938 time: 1702117619
[step:  4800] loss: 0.0436 acc: 22.6562 time: 1702117621
[step:  4900] loss: 0.0446 acc: 14.8438 time: 1702117623
[step:  5000] loss: 0.0449 acc: 18.7500 time: 1702117625
[step:  5100] loss: 0.0441 acc: 20.3125 time: 1702117626
[step:  5200] loss: 0.0446 acc: 12.5000 time: 1702117628
[step:  5300] loss: 0.0445 acc: 16.4062 time: 1702117629
[step:  5400] loss: 0.0436 acc: 25.7812 time: 1702117631
[step:  5500] loss: 0.0437 acc: 25.0000 time: 1702117632
[step:  5600] loss: 0.0445 acc: 16.4062 time: 1702117633
[step:  5700] loss: 0.0444 acc: 15.6250 time: 1702117636
[step:  5800] loss: 0.0437 acc: 25.7812 time: 1702117637
[step:  5900] loss: 0.0451 acc: 7.8125 time: 1702117639
[step:  6000] loss: 0.0442 acc: 20.3125 time: 1702117640
[step:  6100] loss: 0.0438 acc: 24.2188 time: 1702117642
[step:  6200] loss: 0.0442 acc: 16.4062 time: 1702117643
[step:  6300] loss: 0.0438 acc: 21.0938 time: 1702117645
[step:  6400] loss: 0.0448 acc: 17.1875 time: 1702117647
[step:  6500] loss: 0.0439 acc: 21.0938 time: 1702117649
[step:  6600] loss: 0.0443 acc: 16.4062 time: 1702117651
[step:  6700] loss: 0.0443 acc: 19.5312 time: 1702117652
[step:  6800] loss: 0.0446 acc: 13.2812 time: 1702117654
[step:  6900] loss: 0.0434 acc: 25.0000 time: 1702117655
[step:  7000] loss: 0.0437 acc: 25.7812 time: 1702117657
[step:  7100] loss: 0.0439 acc: 25.0000 time: 1702117658
[step:  7200] loss: 0.0443 acc: 24.2188 time: 1702117660
[step:  7300] loss: 0.0443 acc: 21.8750 time: 1702117662
[step:  7400] loss: 0.0440 acc: 22.6562 time: 1702117664
[step:  7500] loss: 0.0445 acc: 14.8438 time: 1702117665
[step:  7600] loss: 0.0444 acc: 18.7500 time: 1702117667
[step:  7700] loss: 0.0447 acc: 19.5312 time: 1702117668
[step:  7800] loss: 0.0437 acc: 18.7500 time: 1702117670
[step:  7900] loss: 0.0446 acc: 14.8438 time: 1702117672
```

```
[step:  8000] loss: 0.0442 acc: 21.0938 time: 1702117674
[step:  8100] loss: 0.0445 acc: 23.4375 time: 1702117675
[step:  8200] loss: 0.0442 acc: 17.9688 time: 1702117677
[step:  8300] loss: 0.0444 acc: 17.9688 time: 1702117678
[step:  8400] loss: 0.0447 acc: 14.8438 time: 1702117680
[step:  8500] loss: 0.0441 acc: 17.9688 time: 1702117682
[step:  8600] loss: 0.0446 acc: 17.1875 time: 1702117683
[step:  8700] loss: 0.0440 acc: 23.4375 time: 1702117686
[step:  8800] loss: 0.0443 acc: 19.5312 time: 1702117687
[step:  8900] loss: 0.0443 acc: 18.7500 time: 1702117689
[step:  9000] loss: 0.0446 acc: 18.7500 time: 1702117690
[step:  9100] loss: 0.0448 acc: 16.4062 time: 1702117692
[step:  9200] loss: 0.0447 acc: 17.1875 time: 1702117693
[step:  9300] loss: 0.0442 acc: 19.5312 time: 1702117695
[step:  9400] loss: 0.0450 acc: 18.7500 time: 1702117697
[step:  9500] loss: 0.0452 acc: 17.1875 time: 1702117699
[step:  9600] loss: 0.0447 acc: 21.0938 time: 1702117700
[step:  9700] loss: 0.0444 acc: 21.0938 time: 1702117702
[step:  9800] loss: 0.0447 acc: 18.7500 time: 1702117703
[step:  9900] loss: 0.0435 acc: 21.8750 time: 1702117705
[step: 10000] loss: 0.0439 acc: 20.3125 time: 1702117707
Finished Training
```

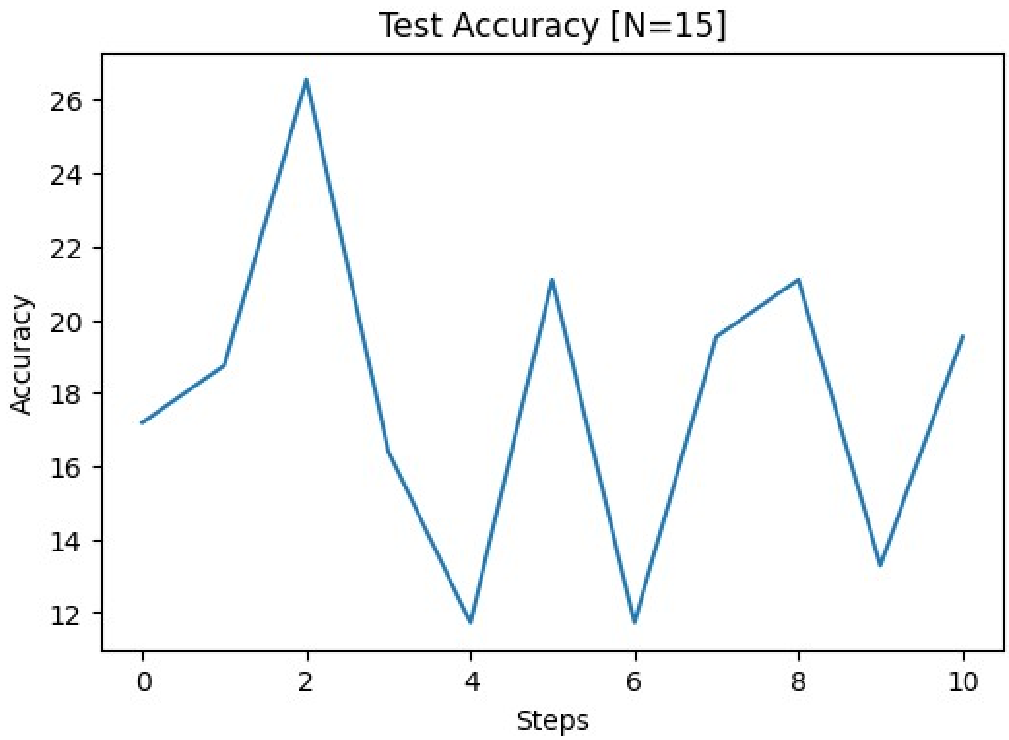





```
# Test the model
test_accuracies = test(model, input_length=p1[2], config=config,
device=device)

# Add accuracies
p2_acc_rnn.append(np.mean(test_accuracies))
```

```
Accuracy: 17.1875
Accuracy: 18.7500
Accuracy: 26.5625
Accuracy: 16.4062
Accuracy: 11.7188
Accuracy: 21.0938
Accuracy: 11.7188
Accuracy: 19.5312
Accuracy: 21.0938
Accuracy: 13.2812
Accuracy: 19.5312
Finished Testing
```

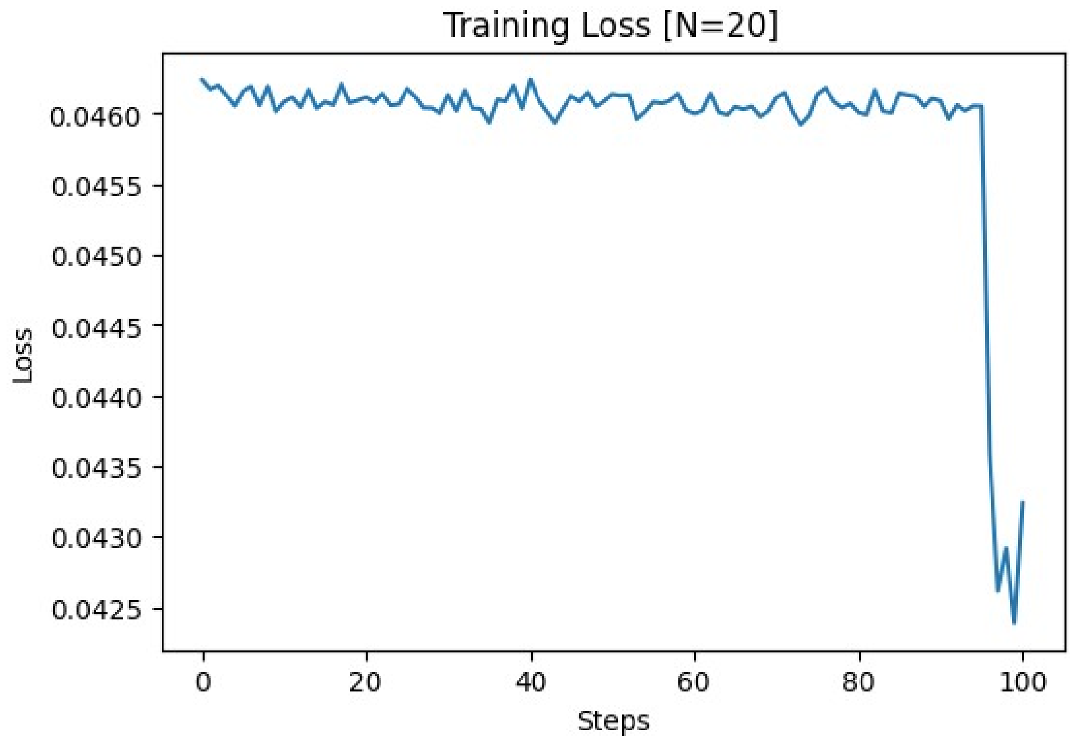
```
# plot the test accuracies
plot_accuracy(test_accuracies, title='Test Accuracy [N=15]',
path=results_path + 'test_accuracy_15_rnn.png')

# Average accuracy over all Steps
print(f"Average test accuracy: {np.mean(test_accuracies):.2f}%")
```

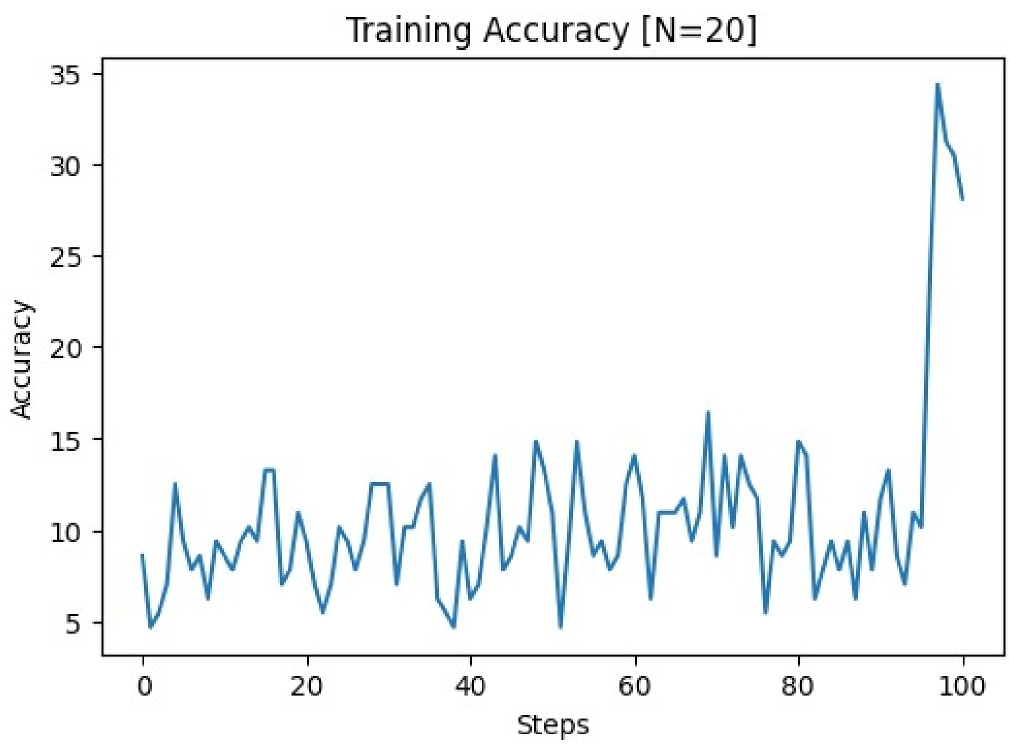



```
Average test accuracy: 17.90%

# Train the model on T=20
model, losses, accuracies = train(config, input_length=p1[3],
lr=config['rnn_learning_rate'], type='RNN', device=device)

[step:     0] loss: 0.0462 acc: 8.5938 time: 1702117710

<ipython-input-6-042aa7a8086d>:64: UserWarning:
torch.nn.utils.clip_grad_norm is now deprecated in favor of
torch.nn.utils.clip_grad_norm_.
  nn.utils.clip_grad_norm(model.parameters(),
max_norm=config['max_norm'])

[step:   100] loss: 0.0462 acc: 4.6875 time: 1702117711
[step:   200] loss: 0.0462 acc: 5.4688 time: 1702117713
[step:   300] loss: 0.0461 acc: 7.0312 time: 1702117715
[step:   400] loss: 0.0461 acc: 12.5000 time: 1702117717
[step:   500] loss: 0.0462 acc: 9.3750 time: 1702117719
[step:   600] loss: 0.0462 acc: 7.8125 time: 1702117722
[step:   700] loss: 0.0461 acc: 8.5938 time: 1702117724
[step:   800] loss: 0.0462 acc: 6.2500 time: 1702117726
[step:   900] loss: 0.0460 acc: 9.3750 time: 1702117728
[step:  1000] loss: 0.0461 acc: 8.5938 time: 1702117730
[step:  1100] loss: 0.0461 acc: 7.8125 time: 1702117732
[step:  1200] loss: 0.0460 acc: 9.3750 time: 1702117735
[step:  1300] loss: 0.0462 acc: 10.1562 time: 1702117736
```

```
[step:  1400] loss: 0.0460 acc: 9.3750 time: 1702117738
[step:  1500] loss: 0.0461 acc: 13.2812 time: 1702117740
[step:  1600] loss: 0.0461 acc: 13.2812 time: 1702117742
[step:  1700] loss: 0.0462 acc: 7.0312 time: 1702117745
[step:  1800] loss: 0.0461 acc: 7.8125 time: 1702117747
[step:  1900] loss: 0.0461 acc: 10.9375 time: 1702117749
[step:  2000] loss: 0.0461 acc: 9.3750 time: 1702117751
[step:  2100] loss: 0.0461 acc: 7.0312 time: 1702117753
[step:  2200] loss: 0.0461 acc: 5.4688 time: 1702117755
[step:  2300] loss: 0.0461 acc: 7.0312 time: 1702117758
[step:  2400] loss: 0.0461 acc: 10.1562 time: 1702117760
[step:  2500] loss: 0.0462 acc: 9.3750 time: 1702117762
[step:  2600] loss: 0.0461 acc: 7.8125 time: 1702117764
[step:  2700] loss: 0.0460 acc: 9.3750 time: 1702117766
[step:  2800] loss: 0.0460 acc: 12.5000 time: 1702117768
[step:  2900] loss: 0.0460 acc: 12.5000 time: 1702117771
[step:  3000] loss: 0.0461 acc: 12.5000 time: 1702117773
[step:  3100] loss: 0.0460 acc: 7.0312 time: 1702117775
[step:  3200] loss: 0.0462 acc: 10.1562 time: 1702117777
[step:  3300] loss: 0.0460 acc: 10.1562 time: 1702117779
[step:  3400] loss: 0.0460 acc: 11.7188 time: 1702117781
[step:  3500] loss: 0.0459 acc: 12.5000 time: 1702117783
[step:  3600] loss: 0.0461 acc: 6.2500 time: 1702117785
[step:  3700] loss: 0.0461 acc: 5.4688 time: 1702117787
[step:  3800] loss: 0.0462 acc: 4.6875 time: 1702117789
[step:  3900] loss: 0.0460 acc: 9.3750 time: 1702117791
[step:  4000] loss: 0.0462 acc: 6.2500 time: 1702117794
[step:  4100] loss: 0.0461 acc: 7.0312 time: 1702117796
[step:  4200] loss: 0.0460 acc: 10.1562 time: 1702117798
[step:  4300] loss: 0.0459 acc: 14.0625 time: 1702117800
[step:  4400] loss: 0.0460 acc: 7.8125 time: 1702117802
[step:  4500] loss: 0.0461 acc: 8.5938 time: 1702117804
[step:  4600] loss: 0.0461 acc: 10.1562 time: 1702117807
[step:  4700] loss: 0.0462 acc: 9.3750 time: 1702117809
[step:  4800] loss: 0.0461 acc: 14.8438 time: 1702117811
[step:  4900] loss: 0.0461 acc: 13.2812 time: 1702117813
[step:  5000] loss: 0.0461 acc: 10.9375 time: 1702117814
[step:  5100] loss: 0.0461 acc: 4.6875 time: 1702117816
[step:  5200] loss: 0.0461 acc: 9.3750 time: 1702117820
[step:  5300] loss: 0.0460 acc: 14.8438 time: 1702117822
[step:  5400] loss: 0.0460 acc: 10.9375 time: 1702117824
[step:  5500] loss: 0.0461 acc: 8.5938 time: 1702117826
[step:  5600] loss: 0.0461 acc: 9.3750 time: 1702117828
[step:  5700] loss: 0.0461 acc: 7.8125 time: 1702117830
[step:  5800] loss: 0.0461 acc: 8.5938 time: 1702117833
[step:  5900] loss: 0.0460 acc: 12.5000 time: 1702117835
[step:  6000] loss: 0.0460 acc: 14.0625 time: 1702117837
[step:  6100] loss: 0.0460 acc: 11.7188 time: 1702117839
[step:  6200] loss: 0.0461 acc: 6.2500 time: 1702117841
```

```
[step:  6300] loss: 0.0460 acc: 10.9375 time: 1702117844
[step:  6400] loss: 0.0460 acc: 10.9375 time: 1702117846
[step:  6500] loss: 0.0460 acc: 10.9375 time: 1702117848
[step:  6600] loss: 0.0460 acc: 11.7188 time: 1702117850
[step:  6700] loss: 0.0461 acc: 9.3750 time: 1702117852
[step:  6800] loss: 0.0460 acc: 10.9375 time: 1702117854
[step:  6900] loss: 0.0460 acc: 16.4062 time: 1702117857
[step:  7000] loss: 0.0461 acc: 8.5938 time: 1702117858
[step:  7100] loss: 0.0461 acc: 14.0625 time: 1702117860
[step:  7200] loss: 0.0460 acc: 10.1562 time: 1702117862
[step:  7300] loss: 0.0459 acc: 14.0625 time: 1702117864
[step:  7400] loss: 0.0460 acc: 12.5000 time: 1702117866
[step:  7500] loss: 0.0461 acc: 11.7188 time: 1702117869
[step:  7600] loss: 0.0462 acc: 5.4688 time: 1702117871
[step:  7700] loss: 0.0461 acc: 9.3750 time: 1702117873
[step:  7800] loss: 0.0460 acc: 8.5938 time: 1702117875
[step:  7900] loss: 0.0461 acc: 9.3750 time: 1702117877
[step:  8000] loss: 0.0460 acc: 14.8438 time: 1702117879
[step:  8100] loss: 0.0460 acc: 14.0625 time: 1702117881
[step:  8200] loss: 0.0462 acc: 6.2500 time: 1702117883
[step:  8300] loss: 0.0460 acc: 7.8125 time: 1702117885
[step:  8400] loss: 0.0460 acc: 9.3750 time: 1702117887
[step:  8500] loss: 0.0461 acc: 7.8125 time: 1702117889
[step:  8600] loss: 0.0461 acc: 9.3750 time: 1702117891
[step:  8700] loss: 0.0461 acc: 6.2500 time: 1702117894
[step:  8800] loss: 0.0461 acc: 10.9375 time: 1702117896
[step:  8900] loss: 0.0461 acc: 7.8125 time: 1702117898
[step:  9000] loss: 0.0461 acc: 11.7188 time: 1702117900
[step:  9100] loss: 0.0460 acc: 13.2812 time: 1702117902
[step:  9200] loss: 0.0461 acc: 8.5938 time: 1702117904
[step:  9300] loss: 0.0460 acc: 7.0312 time: 1702117906
[step:  9400] loss: 0.0461 acc: 10.9375 time: 1702117908
[step:  9500] loss: 0.0461 acc: 10.1562 time: 1702117910
[step:  9600] loss: 0.0436 acc: 23.4375 time: 1702117912
[step:  9700] loss: 0.0426 acc: 34.3750 time: 1702117914
[step:  9800] loss: 0.0429 acc: 31.2500 time: 1702117917
[step:  9900] loss: 0.0424 acc: 30.4688 time: 1702117919
[step: 10000] loss: 0.0432 acc: 28.1250 time: 1702117921
Finished Training

# Plot the losses
plot_loss(losses, title='Training Loss [N=20]', path=results_path +
'training_loss_20_rnn.png')

# Plot the accuracies
plot_accuracy(accuracies, title='Training Accuracy [N=20]',
path=results_path + 'training_accuracy_20_rnn.png')
```



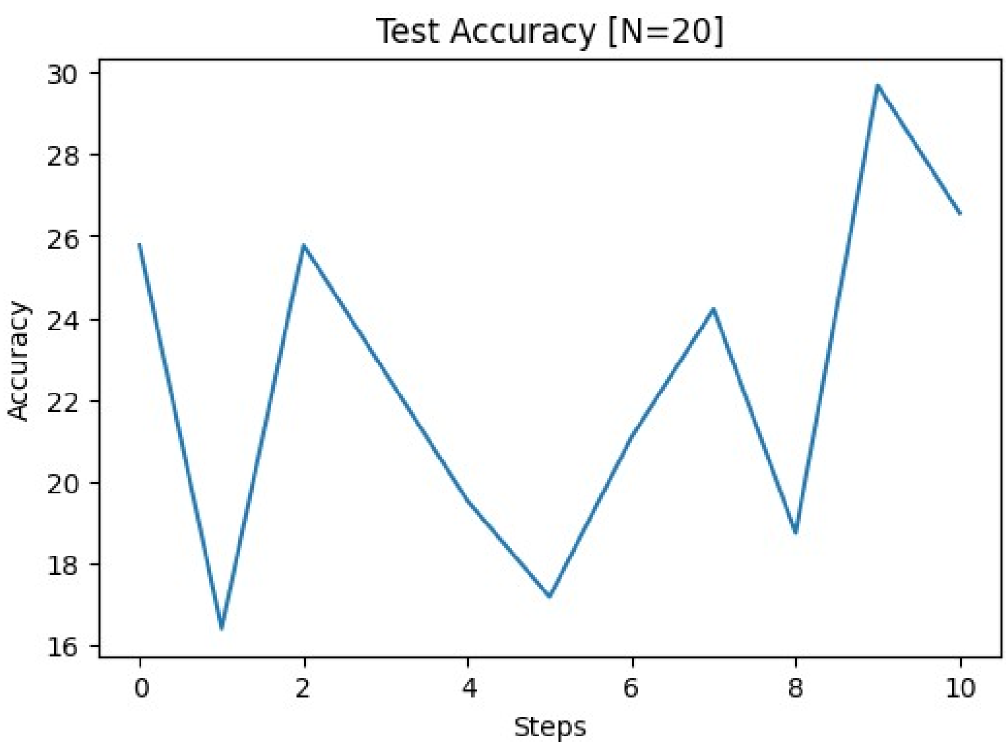



```
# Test the model
test_accuracies = test(model, input_length=p1[3], config=config,
device=device)
```

```
# Add accuracies
p2_acc_rnn.append(np.mean(test_accuracies))
```

```
Accuracy: 25.7812
Accuracy: 16.4062
Accuracy: 25.7812
Accuracy: 22.6562
Accuracy: 19.5312
Accuracy: 17.1875
Accuracy: 21.0938
Accuracy: 24.2188
Accuracy: 18.7500
Accuracy: 29.6875
Accuracy: 26.5625
Finished Testing
```

```
# plot the test accuracies
plot_accuracy(test_accuracies, title='Test Accuracy [N=20]',
path=results_path + 'test_accuracy_20_rnn.png')

# Average accuracy over all Steps
print(f"Average test accuracy: {np.mean(test_accuracies):.2f}%")
```




```
Average test accuracy: 22.51%
```

Question 1.4: Optimization Methods (momentum, adaptive learning rate)

Vanilla Gradient Descent computes gradients of loss function with respect to the model weights and updates them which can make the loss function slowly oscillate towards vertical axes. This happens since no history about the previously computed gradients are kept track off, making the gradient steps less deterministic at each step, which can lead to a slower convergence. Due to this osicllation a large learning rate could lead to disconvergence.

Hence, to achieve faster and more stable convergence, it would be desirable for the loss function to take larger steps in the horizontal direction while taking smaller steps in the vertical direction. Momentum is a technique used to accomplish this. It incorporates an exponentially weighted moving average of the gradient values, which helps the optimizer maintain a consistent direction in the parameter space. By doing so, the momentum term increases for dimensions whose gradients point in the same directions, while reducing updates for dimensions whose gradients frequently change directions. Consequently, this results in faster convergence and reduced oscillation.

Adam Optimizer

Adam is an *adaptive learningrate* optimization algorithm that combines the ideas from RMSprop and Stochastic Gradient Descent with momentum. It utilizes the squared gradients to scale the learning rate, similar to RMSprop, and employs a moving average of the gradients instead of the gradients themselves, like momentum. This combination of techniques allows Adam to adapt the learning rate for each parameter individually, leading to more efficient optimization and faster convergence in many cases.

\begin{aligned} m_t & = \beta_1 m_{t-1} + (1 - \beta_1) g_t \ v_t & = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \ \hat{m}_t & = \frac{m_t}{1 - \beta_1^t} \ \hat{v}*t & = |frac{v_t}{1 - |beta_2^t} | |theta_t & = |theta*{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \end{aligned}

- m_t and v_t represent the first moment (mean) and second moment (uncentered variance) estimates, respectively.
- ß$_1$ and ß$_2$ are the exponential decay rates for the moment estimates.
- g_t denotes the gradient at time step t.
- $\widehat{m}_t$ and $\widehat{v}_t$ are the bias-corrected first moment and second moment estimates, respectively. The bias correction is applied to counteract the effect of initialization on the moment estimates.
- The term $1-\beta_1^t$ and $1-\beta_2^t$ in the denominators of $\widehat{m}_t$ and $\widehat{v}_t$ are used to adjust for the decay of the past gradients as t increases.
- θ_t is the parameter value at time step t. It is updated using the previous parameter value $\theta_t - 1$, the learning rate α, and the ratio of the bias-corrected first moment to the square root of the bias-corrected second moment.
- The term $\sqrt{\widehat{v}_t}+\epsilon$ in the denominator of the update rule for θ_t is used to prevent division by zero and to improve numerical stability. The small positive constant ϵ is typically set to a value such as 10^{-8}.

Task 3: Long-Short Term Network (LSTM) in PyTorch

```
class LSTM(nn.Module):

    def __init__(self, seq_length, input_dim, num_hidden, num_classes,
batch_size=128, device=None):
        super(LSTM, self).__init__()

        self.seq_length = seq_length
        self.input_dim = input_dim
        self.num_hidden = num_hidden
        self.num_classes = num_classes
        self.batch_size = batch_size

        if device is None:
          device = torch.device('cuda' if torch.cuda.is_available()
else 'cpu')

        self.device = device

        # Hidden Layer
        self.W_gx = nn.Parameter(torch.Tensor(self.input_dim,
self.num_hidden))
        self.W_gh = nn.Parameter(torch.Tensor(self.num_hidden,
self.num_hidden))
        self.B_g = nn.Parameter(torch.Tensor(self.num_hidden))

        # Cell State
        # (1) Input gate
        self.W_ix = nn.Parameter(torch.Tensor(self.input_dim,
self.num_hidden))
        self.W_ih = nn.Parameter(torch.Tensor(self.num_hidden,
self.num_hidden))
        self.B_i = nn.Parameter(torch.Tensor(self.num_hidden))

        # (2) Forget gate
        self.W_fx = nn.Parameter(torch.Tensor(self.input_dim,
self.num_hidden))
        self.W_fh = nn.Parameter(torch.Tensor(self.num_hidden,
self.num_hidden))
        self.B_f = nn.Parameter(torch.Tensor(self.num_hidden))

        # (3) Output gate
        self.W_ox = nn.Parameter(torch.Tensor(self.input_dim,
self.num_hidden))
        self.W_oh = nn.Parameter(torch.Tensor(self.num_hidden,
self.num_hidden))
        self.B_o = nn.Parameter(torch.Tensor(self.num_hidden))

        # Output Layer
        self.W_ph = nn.Parameter(torch.Tensor(self.num_hidden,
```

```
self.num_classes))
        self.B_y = nn.Parameter(torch.Tensor(self.num_classes))

        # Initialize weights
        self.init_weights()


    def forward(self, x):
        # Initialize hidden state and cell state
        h_t = torch.zeros(self.batch_size, self.num_hidden,
device=self.device)
        c_t = torch.zeros(self.batch_size, self.num_hidden,
device=self.device)

        for t in range(self.seq_length):
            x_t = x[:, t].view(self.batch_size, -1)

            # Compute the hidden state
            i_t = torch.sigmoid(x_t @ self.W_ix + h_t @ self.W_ih +
self.B_i)
            f_t = torch.sigmoid(x_t @ self.W_fx + h_t @ self.W_fh +
self.B_f)
            o_t = torch.sigmoid(x_t @ self.W_ox + h_t @ self.W_oh +
self.B_o)
            g_t = torch.tanh(x_t @ self.W_gx + h_t @ self.W_gh +
self.B_g)

            c_t = f_t * c_t + i_t * g_t
            h_t = o_t * torch.tanh(c_t)

        # Compute the output
        output = h_t @ self.W_ph + self.B_y
        y = torch.softmax(output, dim=1)
        return y

    def init_weights(self):
        """ Initialize weights to avoid gradients vanishing or
exploding.
            Source: https://dennybritz.com/posts/wildml/recurrent-
neural-networks-tutorial-part-2/
        """
        n_gx = self.W_gx.size(0)
        nn.init.uniform_(self.W_gx, -1 / sqrt(n_gx), 1 / sqrt(n_gx))
        n_gh = self.W_gh.size(0)

        nn.init.uniform_(self.W_gh, -1 / sqrt(n_gh), 1 / sqrt(n_gh))
        n_ix = self.W_ix.size(0)
        nn.init.uniform_(self.W_ix, -1 / sqrt(n_ix), 1 / sqrt(n_ix))

        n_ih = self.W_ih.size(0)
```

```
        nn.init.uniform_(self.W_ih, -1 / sqrt(n_ih), 1 / sqrt(n_ih))
        n_fx = self.W_fx.size(0)
        nn.init.uniform_(self.W_fx, -1 / sqrt(n_fx), 1 / sqrt(n_fx))

        n_fh = self.W_fh.size(0)
        nn.init.uniform_(self.W_fh, -1 / sqrt(n_fh), 1 / sqrt(n_fh))
        n_ox = self.W_ox.size(0)
        nn.init.uniform_(self.W_ox, -1 / sqrt(n_ox), 1 / sqrt(n_ox))

        n_oh = self.W_oh.size(0)
        nn.init.uniform_(self.W_oh, -1 / sqrt(n_oh), 1 / sqrt(n_oh))
        n_ph = self.W_ph.size(0)
        nn.init.uniform_(self.W_ph, -1 / sqrt(n_ph), 1 / sqrt(n_ph))

        nn.init.zeros_(self.B_g)
        nn.init.zeros_(self.B_i)
        nn.init.zeros_(self.B_f)
        nn.init.zeros_(self.B_o)
        nn.init.zeros_(self.B_y)


    def init_hidden(self):
        # Initialize hidden state
        self.hidden_state = torch.zeros(self.batch_size,
self.self.num_hidden, device=self.device)

    def set_grad(self, requires_grad):
        # Set requires_grad for all parameters
        for param in self.parameters():
            param.requires_grad = requires_grad
```

Question 1.5(a): LSTM Gates

Idea is that humans don't understand a sequence of information from scratch every time, but that there is a contextual understanding based on previous context (time-sequence datat). Hence Recurrent networks have a *recursive step* or recurrence in them which persists prior information along the way. In theory RNNs should be able to learn very long-term dependcies, but in practice as describe above (Vanishing Gradient Problem) they are not.

Long Short Term Memory Networks and Long-Term Dependencies

However, such long-term dependencies might on one hand connect previous infromation to the present taks, but equally for certain tasks only the persent information might be needed. LSTM networks avoid the Vanishing Gradient problem by keeping a cell state c_t which allows the gradient to flow backwards through the network without the need to go through any learned NN weight layers. Information can flow along like on a conveyabelt. Thereby information can be added, updated or removed.

LSTM Gates - The gated memory cell

(1) Input Gate: Decides which values will be updated in the cell state.

(2) Forget Gate: Decides which information can be discarded from the cell state.

(3) Output Gate: Decides which information will be added to the cell state.

Question 1.5(b): LSTM Trainable Parameters

(1) For each gate, there are d x n parameters in the input weight matrix; n x n parameters in the recurrent weight matrix and n parameters in the bias term:

\begin{aligned} d \times n + n \times n + n \end{aligned}

(2) Since there are 3 gates (input, forget, and output) and one cell state, the total number of trainable parameters is:

\begin{aligned} 3 \times (d \times n + n \times n + n) + (d \times n + n \times n + n) = 4 \times (d \times n + n \times n + n). \end{aligned}

The formula for the total number of trainable parameters in the LSTM cell is:

\begin{aligned} 4 \times (d \times n + n \times n + n) \end{aligned}

Question 1.6: Implement a LSTM network

```
# Train the model T=5
model, losses, accuracies = train(config, input_length=p1[0],
lr=config['lstm_learning_rate'], type='LSTM', device=device)
```

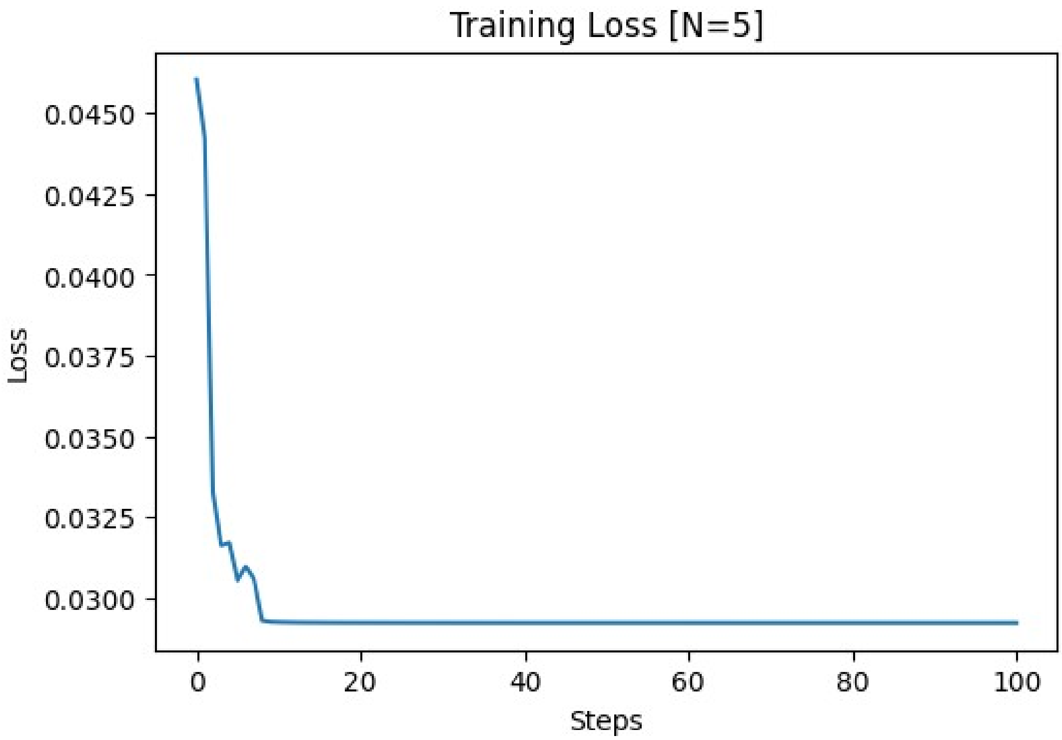
```
[step:     0] loss: 0.0461 acc: 8.5938 time: 1702117923
```

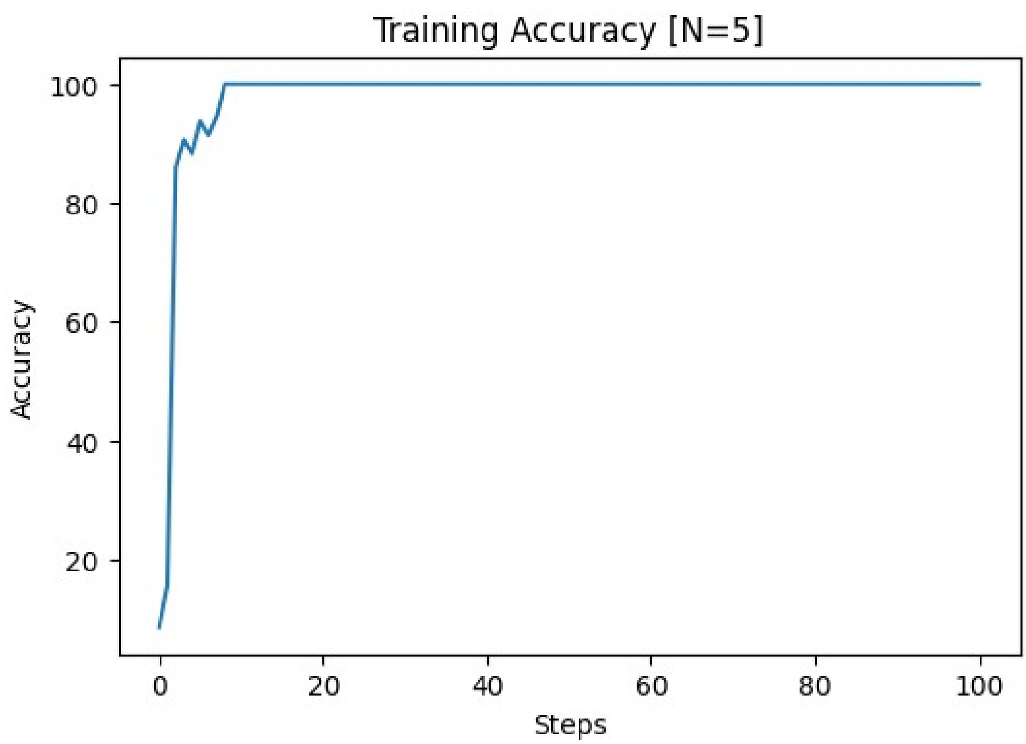
```
<ipython-input-6-042aa7a8086d>:64: UserWarning:
torch.nn.utils.clip_grad_norm is now deprecated in favor of
torch.nn.utils.clip_grad_norm_.
  nn.utils.clip_grad_norm(model.parameters(),
max_norm=config['max_norm'])
```

```
[step:   100] loss: 0.0442 acc: 15.6250 time: 1702117925
[step:   200] loss: 0.0333 acc: 85.9375 time: 1702117927
[step:   300] loss: 0.0316 acc: 90.6250 time: 1702117930
[step:   400] loss: 0.0317 acc: 88.2812 time: 1702117932
[step:   500] loss: 0.0305 acc: 93.7500 time: 1702117933
[step:   600] loss: 0.0310 acc: 91.4062 time: 1702117935
[step:   700] loss: 0.0306 acc: 94.5312 time: 1702117937
[step:   800] loss: 0.0293 acc: 100.0000 time: 1702117939
[step:   900] loss: 0.0293 acc: 100.0000 time: 1702117941
[step:  1000] loss: 0.0292 acc: 100.0000 time: 1702117943
[step:  1100] loss: 0.0292 acc: 100.0000 time: 1702117945
[step:  1200] loss: 0.0292 acc: 100.0000 time: 1702117947
[step:  1300] loss: 0.0292 acc: 100.0000 time: 1702117949
[step:  1400] loss: 0.0292 acc: 100.0000 time: 1702117950
[step:  1500] loss: 0.0292 acc: 100.0000 time: 1702117952
[step:  1600] loss: 0.0292 acc: 100.0000 time: 1702117955
[step:  1700] loss: 0.0292 acc: 100.0000 time: 1702117957
[step:  1800] loss: 0.0292 acc: 100.0000 time: 1702117959
```

```
[step:  1900] loss: 0.0292 acc: 100.0000 time: 1702117960
[step:  2000] loss: 0.0292 acc: 100.0000 time: 1702117962
[step:  2100] loss: 0.0292 acc: 100.0000 time: 1702117964
[step:  2200] loss: 0.0292 acc: 100.0000 time: 1702117967
[step:  2300] loss: 0.0292 acc: 100.0000 time: 1702117969
[step:  2400] loss: 0.0292 acc: 100.0000 time: 1702117970
[step:  2500] loss: 0.0292 acc: 100.0000 time: 1702117972
[step:  2600] loss: 0.0292 acc: 100.0000 time: 1702117974
[step:  2700] loss: 0.0292 acc: 100.0000 time: 1702117976
[step:  2800] loss: 0.0292 acc: 100.0000 time: 1702117978
[step:  2900] loss: 0.0292 acc: 100.0000 time: 1702117980
[step:  3000] loss: 0.0292 acc: 100.0000 time: 1702117982
[step:  3100] loss: 0.0292 acc: 100.0000 time: 1702117984
[step:  3200] loss: 0.0292 acc: 100.0000 time: 1702117985
[step:  3300] loss: 0.0292 acc: 100.0000 time: 1702117987
[step:  3400] loss: 0.0292 acc: 100.0000 time: 1702117989
[step:  3500] loss: 0.0292 acc: 100.0000 time: 1702117992
[step:  3600] loss: 0.0292 acc: 100.0000 time: 1702117994
[step:  3700] loss: 0.0292 acc: 100.0000 time: 1702117996
[step:  3800] loss: 0.0292 acc: 100.0000 time: 1702117997
[step:  3900] loss: 0.0292 acc: 100.0000 time: 1702117999
[step:  4000] loss: 0.0292 acc: 100.0000 time: 1702118001
[step:  4100] loss: 0.0292 acc: 100.0000 time: 1702118004
[step:  4200] loss: 0.0292 acc: 100.0000 time: 1702118006
[step:  4300] loss: 0.0292 acc: 100.0000 time: 1702118007
[step:  4400] loss: 0.0292 acc: 100.0000 time: 1702118009
[step:  4500] loss: 0.0292 acc: 100.0000 time: 1702118011
[step:  4600] loss: 0.0292 acc: 100.0000 time: 1702118013
[step:  4700] loss: 0.0292 acc: 100.0000 time: 1702118015
[step:  4800] loss: 0.0292 acc: 100.0000 time: 1702118017
[step:  4900] loss: 0.0292 acc: 100.0000 time: 1702118019
[step:  5000] loss: 0.0292 acc: 100.0000 time: 1702118021
[step:  5100] loss: 0.0292 acc: 100.0000 time: 1702118022
[step:  5200] loss: 0.0292 acc: 100.0000 time: 1702118024
[step:  5300] loss: 0.0292 acc: 100.0000 time: 1702118026
[step:  5400] loss: 0.0292 acc: 100.0000 time: 1702118029
[step:  5500] loss: 0.0292 acc: 100.0000 time: 1702118030
[step:  5600] loss: 0.0292 acc: 100.0000 time: 1702118032
[step:  5700] loss: 0.0292 acc: 100.0000 time: 1702118034
[step:  5800] loss: 0.0292 acc: 100.0000 time: 1702118036
[step:  5900] loss: 0.0292 acc: 100.0000 time: 1702118038
[step:  6000] loss: 0.0292 acc: 100.0000 time: 1702118041
[step:  6100] loss: 0.0292 acc: 100.0000 time: 1702118042
[step:  6200] loss: 0.0292 acc: 100.0000 time: 1702118044
[step:  6300] loss: 0.0292 acc: 100.0000 time: 1702118046
[step:  6400] loss: 0.0292 acc: 100.0000 time: 1702118049
[step:  6500] loss: 0.0292 acc: 100.0000 time: 1702118051
[step:  6600] loss: 0.0292 acc: 100.0000 time: 1702118055
[step:  6700] loss: 0.0292 acc: 100.0000 time: 1702118059
```

```
[step:  6800] loss: 0.0292 acc: 100.0000 time: 1702118061
[step:  6900] loss: 0.0292 acc: 100.0000 time: 1702118063
[step:  7000] loss: 0.0292 acc: 100.0000 time: 1702118064
[step:  7100] loss: 0.0292 acc: 100.0000 time: 1702118067
[step:  7200] loss: 0.0292 acc: 100.0000 time: 1702118069
[step:  7300] loss: 0.0292 acc: 100.0000 time: 1702118071
[step:  7400] loss: 0.0292 acc: 100.0000 time: 1702118072
[step:  7500] loss: 0.0292 acc: 100.0000 time: 1702118074
[step:  7600] loss: 0.0292 acc: 100.0000 time: 1702118076
[step:  7700] loss: 0.0292 acc: 100.0000 time: 1702118078
[step:  7800] loss: 0.0292 acc: 100.0000 time: 1702118080
[step:  7900] loss: 0.0292 acc: 100.0000 time: 1702118082
[step:  8000] loss: 0.0292 acc: 100.0000 time: 1702118084
[step:  8100] loss: 0.0292 acc: 100.0000 time: 1702118086
[step:  8200] loss: 0.0292 acc: 100.0000 time: 1702118087
[step:  8300] loss: 0.0292 acc: 100.0000 time: 1702118089
[step:  8400] loss: 0.0292 acc: 100.0000 time: 1702118092
[step:  8500] loss: 0.0292 acc: 100.0000 time: 1702118094
[step:  8600] loss: 0.0292 acc: 100.0000 time: 1702118095
[step:  8700] loss: 0.0292 acc: 100.0000 time: 1702118097
[step:  8800] loss: 0.0292 acc: 100.0000 time: 1702118099
[step:  8900] loss: 0.0292 acc: 100.0000 time: 1702118101
[step:  9000] loss: 0.0292 acc: 100.0000 time: 1702118103
[step:  9100] loss: 0.0292 acc: 100.0000 time: 1702118105
[step:  9200] loss: 0.0292 acc: 100.0000 time: 1702118107
[step:  9300] loss: 0.0292 acc: 100.0000 time: 1702118109
[step:  9400] loss: 0.0292 acc: 100.0000 time: 1702118111
[step:  9500] loss: 0.0292 acc: 100.0000 time: 1702118113
[step:  9600] loss: 0.0292 acc: 100.0000 time: 1702118115
[step:  9700] loss: 0.0292 acc: 100.0000 time: 1702118118
[step:  9800] loss: 0.0292 acc: 100.0000 time: 1702118120
[step:  9900] loss: 0.0292 acc: 100.0000 time: 1702118121
[step: 10000] loss: 0.0292 acc: 100.0000 time: 1702118123
Finished Training

# Plot the losses
plot_loss(losses, title='Training Loss [N=5]', path=results_path +
'training_loss_5_lstm.png')

# Plot the accuracies
plot_accuracy(accuracies, title='Training Accuracy [N=5]',
path=results_path + 'training_accuracy_5_lstm.png')
```



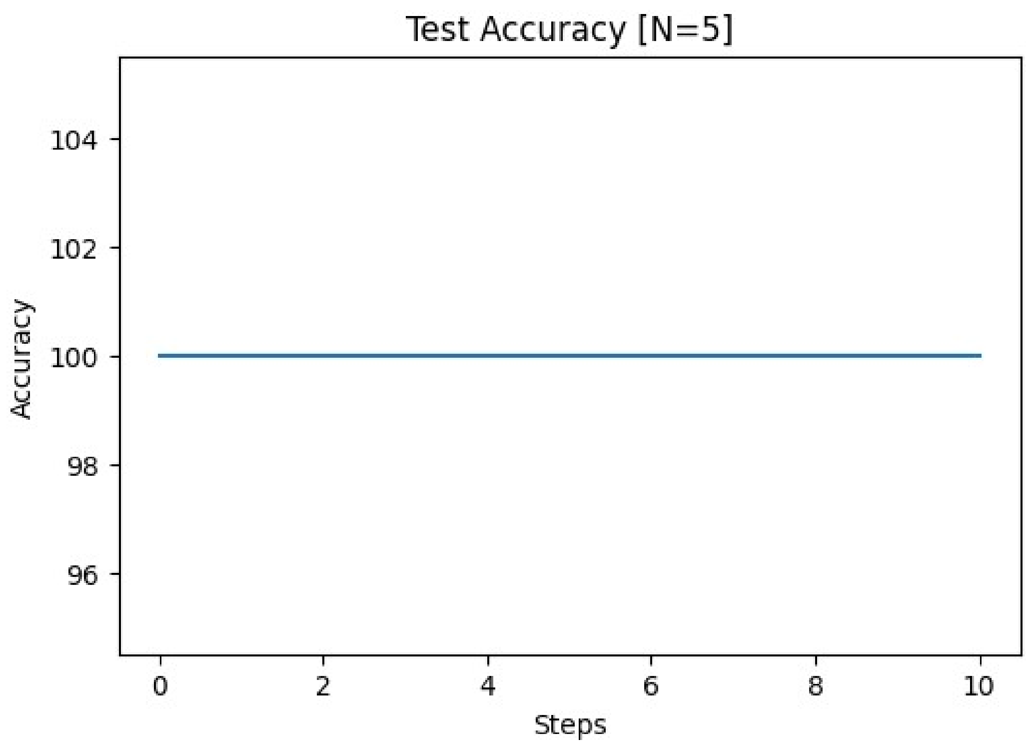



```
# Test the model
test_accuracies = test(model, input_length=p1[0], config=config,
device=device)
```

```
# Add accuracies
p3_acc_lstm.append(np.mean(test_accuracies))
```

```
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Finished Testing
```

```
# plot the test accuracies
plot_accuracy(test_accuracies, title='Test Accuracy [N=5]',
path=results_path + 'test_accuracy_5_lstm.png')

# Average accuracy over all Steps
print(f"Average test accuracy: {np.mean(test_accuracies):.2f}%")
```




```
Average test accuracy: 100.00%
```

```
# Train the model T=10
model, losses, accuracies = train(config, input_length=p1[1],
lr=config['lstm_learning_rate'], type='LSTM', device=device)
```

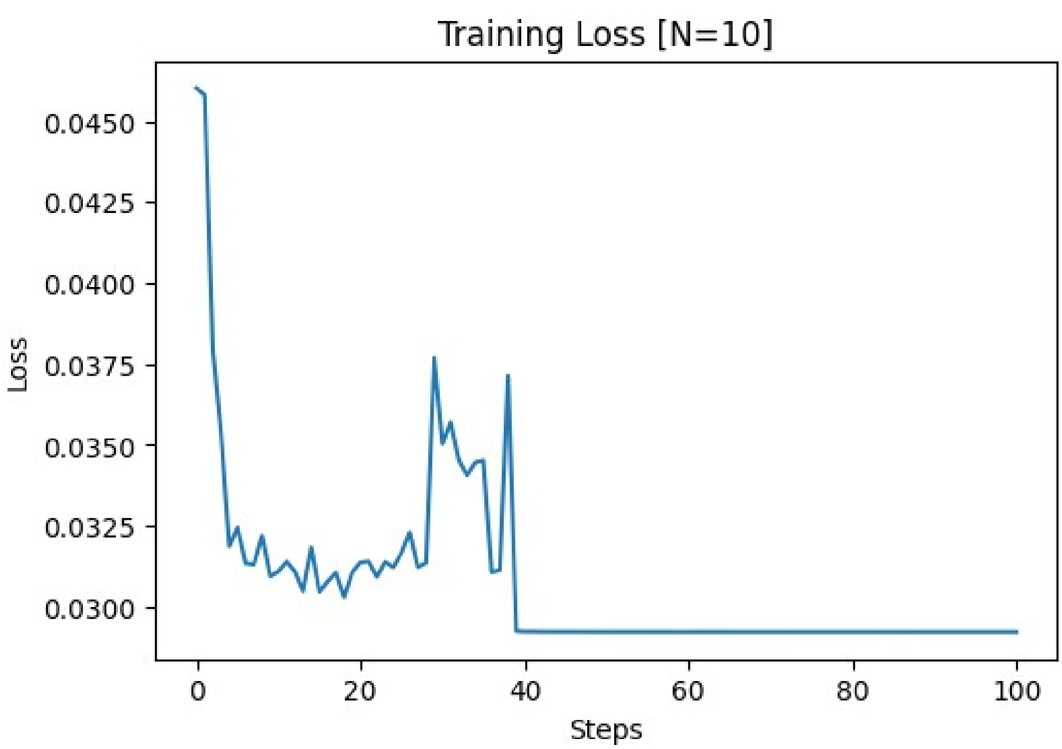
```
[step:     0] loss: 0.0460 acc: 15.6250 time: 1702118125
```

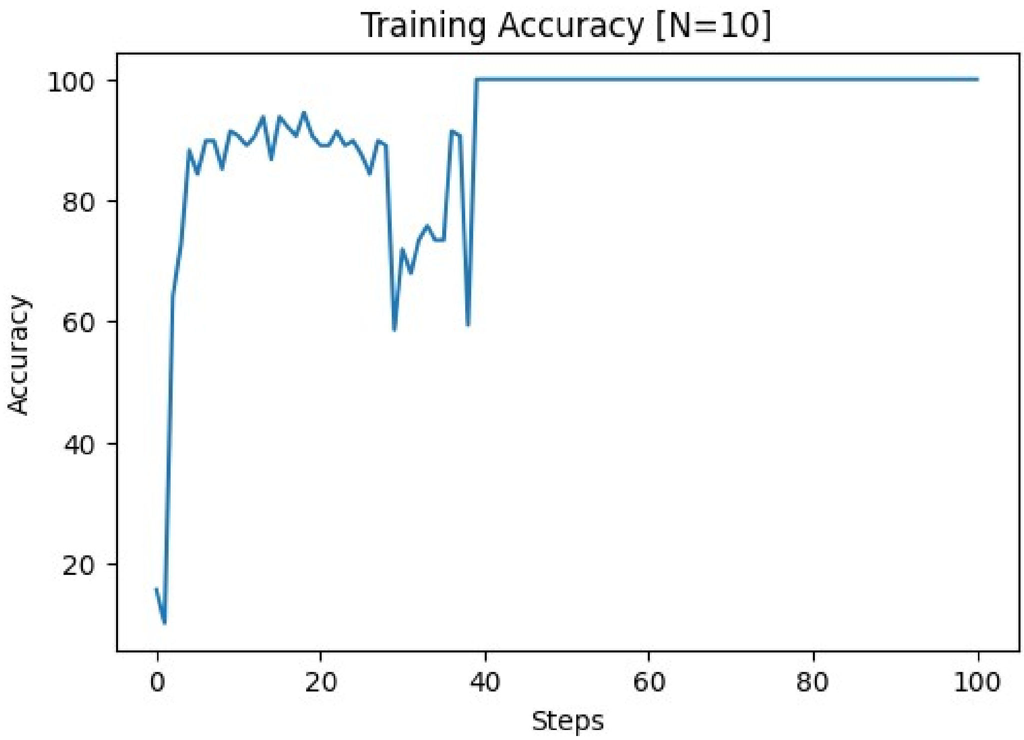
```
<ipython-input-6-042aa7a8086d>:64: UserWarning:
torch.nn.utils.clip_grad_norm is now deprecated in favor of
torch.nn.utils.clip_grad_norm_.
  nn.utils.clip_grad_norm(model.parameters(),
max_norm=config['max_norm'])
```

```
[step:   100] loss: 0.0458 acc: 10.1562 time: 1702118129
[step:   200] loss: 0.0379 acc: 64.0625 time: 1702118133
[step:   300] loss: 0.0354 acc: 72.6562 time: 1702118136
[step:   400] loss: 0.0319 acc: 88.2812 time: 1702118139
[step:   500] loss: 0.0324 acc: 84.3750 time: 1702118143
[step:   600] loss: 0.0313 acc: 89.8438 time: 1702118146
[step:   700] loss: 0.0313 acc: 89.8438 time: 1702118149
[step:   800] loss: 0.0322 acc: 85.1562 time: 1702118153
[step:   900] loss: 0.0309 acc: 91.4062 time: 1702118156
[step:  1000] loss: 0.0311 acc: 90.6250 time: 1702118159
[step:  1100] loss: 0.0314 acc: 89.0625 time: 1702118163
[step:  1200] loss: 0.0311 acc: 90.6250 time: 1702118167
[step:  1300] loss: 0.0305 acc: 93.7500 time: 1702118170
[step:  1400] loss: 0.0318 acc: 86.7188 time: 1702118173
[step:  1500] loss: 0.0305 acc: 93.7500 time: 1702118176
[step:  1600] loss: 0.0308 acc: 92.1875 time: 1702118180
[step:  1700] loss: 0.0311 acc: 90.6250 time: 1702118183
[step:  1800] loss: 0.0303 acc: 94.5312 time: 1702118186
[step:  1900] loss: 0.0311 acc: 90.6250 time: 1702118190
[step:  2000] loss: 0.0314 acc: 89.0625 time: 1702118194
[step:  2100] loss: 0.0314 acc: 89.0625 time: 1702118197
[step:  2200] loss: 0.0309 acc: 91.4062 time: 1702118200
[step:  2300] loss: 0.0314 acc: 89.0625 time: 1702118204
[step:  2400] loss: 0.0312 acc: 89.8438 time: 1702118207
[step:  2500] loss: 0.0317 acc: 87.5000 time: 1702118210
[step:  2600] loss: 0.0323 acc: 84.3750 time: 1702118214
[step:  2700] loss: 0.0312 acc: 89.8438 time: 1702118217
[step:  2800] loss: 0.0314 acc: 89.0625 time: 1702118220
[step:  2900] loss: 0.0377 acc: 58.5938 time: 1702118223
[step:  3000] loss: 0.0350 acc: 71.8750 time: 1702118227
[step:  3100] loss: 0.0357 acc: 67.9688 time: 1702118231
[step:  3200] loss: 0.0345 acc: 73.4375 time: 1702118234
[step:  3300] loss: 0.0341 acc: 75.7812 time: 1702118237
[step:  3400] loss: 0.0345 acc: 73.4375 time: 1702118241
[step:  3500] loss: 0.0345 acc: 73.4375 time: 1702118244
[step:  3600] loss: 0.0311 acc: 91.4062 time: 1702118247
[step:  3700] loss: 0.0311 acc: 90.6250 time: 1702118251
[step:  3800] loss: 0.0371 acc: 59.3750 time: 1702118254
```

```
[step:  3900] loss: 0.0292 acc: 100.0000 time: 1702118257
[step:  4000] loss: 0.0292 acc: 100.0000 time: 1702118260
[step:  4100] loss: 0.0292 acc: 100.0000 time: 1702118264
[step:  4200] loss: 0.0292 acc: 100.0000 time: 1702118268
[step:  4300] loss: 0.0292 acc: 100.0000 time: 1702118271
[step:  4400] loss: 0.0292 acc: 100.0000 time: 1702118274
[step:  4500] loss: 0.0292 acc: 100.0000 time: 1702118278
[step:  4600] loss: 0.0292 acc: 100.0000 time: 1702118281
[step:  4700] loss: 0.0292 acc: 100.0000 time: 1702118284
[step:  4800] loss: 0.0292 acc: 100.0000 time: 1702118288
[step:  4900] loss: 0.0292 acc: 100.0000 time: 1702118292
[step:  5000] loss: 0.0292 acc: 100.0000 time: 1702118296
[step:  5100] loss: 0.0292 acc: 100.0000 time: 1702118299
[step:  5200] loss: 0.0292 acc: 100.0000 time: 1702118303
[step:  5300] loss: 0.0292 acc: 100.0000 time: 1702118306
[step:  5400] loss: 0.0292 acc: 100.0000 time: 1702118309
[step:  5500] loss: 0.0292 acc: 100.0000 time: 1702118313
[step:  5600] loss: 0.0292 acc: 100.0000 time: 1702118316
[step:  5700] loss: 0.0292 acc: 100.0000 time: 1702118320
[step:  5800] loss: 0.0292 acc: 100.0000 time: 1702118323
[step:  5900] loss: 0.0292 acc: 100.0000 time: 1702118327
[step:  6000] loss: 0.0292 acc: 100.0000 time: 1702118330
[step:  6100] loss: 0.0292 acc: 100.0000 time: 1702118333
[step:  6200] loss: 0.0292 acc: 100.0000 time: 1702118337
[step:  6300] loss: 0.0292 acc: 100.0000 time: 1702118340
[step:  6400] loss: 0.0292 acc: 100.0000 time: 1702118344
[step:  6500] loss: 0.0292 acc: 100.0000 time: 1702118347
[step:  6600] loss: 0.0292 acc: 100.0000 time: 1702118351
[step:  6700] loss: 0.0292 acc: 100.0000 time: 1702118354
[step:  6800] loss: 0.0292 acc: 100.0000 time: 1702118357
[step:  6900] loss: 0.0292 acc: 100.0000 time: 1702118361
[step:  7000] loss: 0.0292 acc: 100.0000 time: 1702118365
[step:  7100] loss: 0.0292 acc: 100.0000 time: 1702118368
[step:  7200] loss: 0.0292 acc: 100.0000 time: 1702118371
[step:  7300] loss: 0.0292 acc: 100.0000 time: 1702118375
[step:  7400] loss: 0.0292 acc: 100.0000 time: 1702118378
[step:  7500] loss: 0.0292 acc: 100.0000 time: 1702118382
[step:  7600] loss: 0.0292 acc: 100.0000 time: 1702118385
[step:  7700] loss: 0.0292 acc: 100.0000 time: 1702118389
[step:  7800] loss: 0.0292 acc: 100.0000 time: 1702118392
[step:  7900] loss: 0.0292 acc: 100.0000 time: 1702118395
[step:  8000] loss: 0.0292 acc: 100.0000 time: 1702118399
[step:  8100] loss: 0.0292 acc: 100.0000 time: 1702118402
[step:  8200] loss: 0.0292 acc: 100.0000 time: 1702118405
[step:  8300] loss: 0.0292 acc: 100.0000 time: 1702118408
[step:  8400] loss: 0.0292 acc: 100.0000 time: 1702118412
[step:  8500] loss: 0.0292 acc: 100.0000 time: 1702118416
[step:  8600] loss: 0.0292 acc: 100.0000 time: 1702118419
[step:  8700] loss: 0.0292 acc: 100.0000 time: 1702118423
```

```
[step:  8800] loss: 0.0292 acc: 100.0000 time: 1702118427
[step:  8900] loss: 0.0292 acc: 100.0000 time: 1702118430
[step:  9000] loss: 0.0292 acc: 100.0000 time: 1702118433
[step:  9100] loss: 0.0292 acc: 100.0000 time: 1702118437
[step:  9200] loss: 0.0292 acc: 100.0000 time: 1702118440
[step:  9300] loss: 0.0292 acc: 100.0000 time: 1702118444
[step:  9400] loss: 0.0292 acc: 100.0000 time: 1702118448
[step:  9500] loss: 0.0292 acc: 100.0000 time: 1702118451
[step:  9600] loss: 0.0292 acc: 100.0000 time: 1702118454
[step:  9700] loss: 0.0292 acc: 100.0000 time: 1702118457
[step:  9800] loss: 0.0292 acc: 100.0000 time: 1702118461
[step:  9900] loss: 0.0292 acc: 100.0000 time: 1702118464
[step: 10000] loss: 0.0292 acc: 100.0000 time: 1702118467
Finished Training

# Plot the losses
plot_loss(losses, title='Training Loss [N=10]', path=results_path +
'training_loss_10_lstm.png')

# Plot the accuracies
plot_accuracy(accuracies, title='Training Accuracy [N=10]',
path=results_path + 'training_accuracy_10_lstm.png')
```

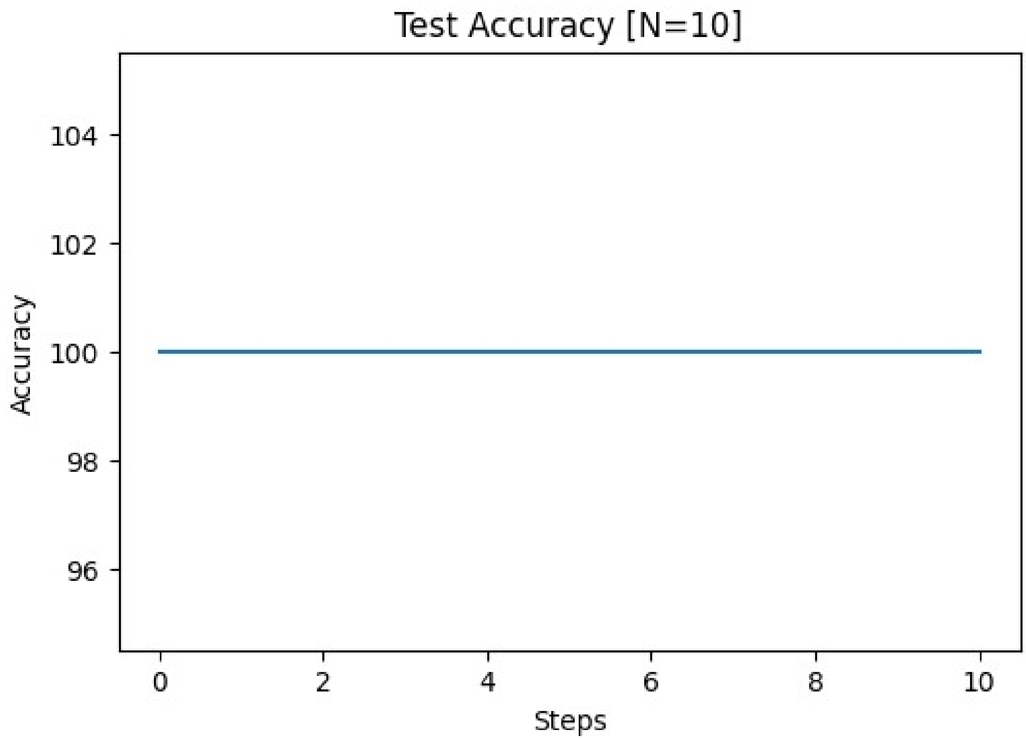





```python
# Test the model
test_accuracies = test(model, input_length=p1[1], config=config,
device=device)

# Add accuracies
p3_acc_lstm.append(np.mean(test_accuracies))
```

```
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Accuracy: 100.0000
Finished Testing
```

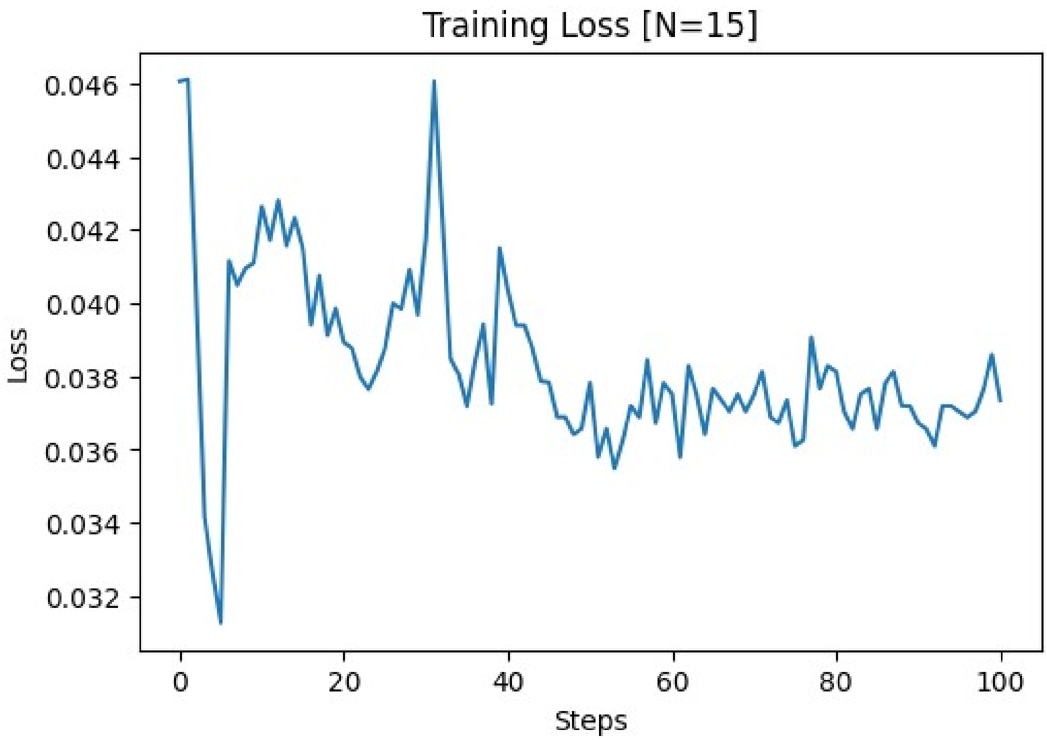
```python
# plot the test accuracies
plot_accuracy(test_accuracies, title='Test Accuracy [N=10]',
path=results_path + 'test_accuracy_10_lstm.png')

# Average accuracy over all Steps
print(f"Average test accuracy: {np.mean(test_accuracies):.2f}%")
```

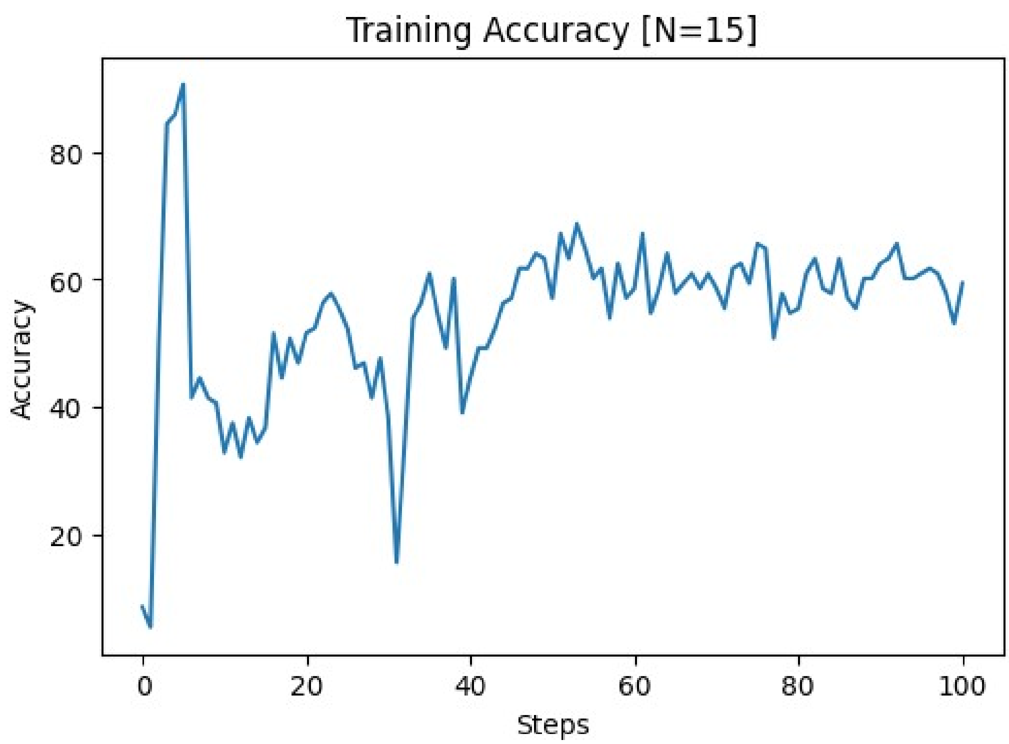



```
Average test accuracy: 100.00%

# Train the model T=15
model, losses, accuracies = train(config, input_length=p1[2],
lr=config['lstm_learning_rate'], type='LSTM', device=device)

[step:     0] loss: 0.0461 acc: 8.5938 time: 1702118470

<ipython-input-6-042aa7a8086d>:64: UserWarning:
torch.nn.utils.clip_grad_norm is now deprecated in favor of
torch.nn.utils.clip_grad_norm_.
  nn.utils.clip_grad_norm(model.parameters(),
max_norm=config['max_norm'])

[step:   100] loss: 0.0461 acc: 5.4688 time: 1702118475
[step:   200] loss: 0.0402 acc: 50.0000 time: 1702118480
[step:   300] loss: 0.0342 acc: 84.3750 time: 1702118485
[step:   400] loss: 0.0326 acc: 85.9375 time: 1702118490
[step:   500] loss: 0.0313 acc: 90.6250 time: 1702118494
[step:   600] loss: 0.0412 acc: 41.4062 time: 1702118500
[step:   700] loss: 0.0405 acc: 44.5312 time: 1702118504
[step:   800] loss: 0.0410 acc: 41.4062 time: 1702118509
[step:   900] loss: 0.0411 acc: 40.6250 time: 1702118514
[step:  1000] loss: 0.0426 acc: 32.8125 time: 1702118518
[step:  1100] loss: 0.0417 acc: 37.5000 time: 1702118523
[step:  1200] loss: 0.0428 acc: 32.0312 time: 1702118529
[step:  1300] loss: 0.0416 acc: 38.2812 time: 1702118534
```

```
[step:  1400] loss: 0.0423 acc: 34.3750 time: 1702118538
[step:  1500] loss: 0.0415 acc: 36.7188 time: 1702118543
[step:  1600] loss: 0.0394 acc: 51.5625 time: 1702118548
[step:  1700] loss: 0.0408 acc: 44.5312 time: 1702118553
[step:  1800] loss: 0.0391 acc: 50.7812 time: 1702118557
[step:  1900] loss: 0.0399 acc: 46.8750 time: 1702118562
[step:  2000] loss: 0.0389 acc: 51.5625 time: 1702118567
[step:  2100] loss: 0.0388 acc: 52.3438 time: 1702118572
[step:  2200] loss: 0.0380 acc: 56.2500 time: 1702118577
[step:  2300] loss: 0.0376 acc: 57.8125 time: 1702118582
[step:  2400] loss: 0.0381 acc: 55.4688 time: 1702118587
[step:  2500] loss: 0.0388 acc: 52.3438 time: 1702118592
[step:  2600] loss: 0.0400 acc: 46.0938 time: 1702118597
[step:  2700] loss: 0.0398 acc: 46.8750 time: 1702118602
[step:  2800] loss: 0.0409 acc: 41.4062 time: 1702118606
[step:  2900] loss: 0.0397 acc: 47.6562 time: 1702118611
[step:  3000] loss: 0.0418 acc: 38.2812 time: 1702118616
[step:  3100] loss: 0.0461 acc: 15.6250 time: 1702118621
[step:  3200] loss: 0.0423 acc: 34.3750 time: 1702118625
[step:  3300] loss: 0.0385 acc: 53.9062 time: 1702118630
[step:  3400] loss: 0.0381 acc: 56.2500 time: 1702118635
[step:  3500] loss: 0.0372 acc: 60.9375 time: 1702118639
[step:  3600] loss: 0.0384 acc: 54.6875 time: 1702118644
[step:  3700] loss: 0.0394 acc: 49.2188 time: 1702118649
[step:  3800] loss: 0.0372 acc: 60.1562 time: 1702118654
[step:  3900] loss: 0.0415 acc: 39.0625 time: 1702118660
[step:  4000] loss: 0.0403 acc: 44.5312 time: 1702118666
[step:  4100] loss: 0.0394 acc: 49.2188 time: 1702118670
[step:  4200] loss: 0.0394 acc: 49.2188 time: 1702118675
[step:  4300] loss: 0.0388 acc: 52.3438 time: 1702118680
[step:  4400] loss: 0.0379 acc: 56.2500 time: 1702118685
[step:  4500] loss: 0.0378 acc: 57.0312 time: 1702118690
[step:  4600] loss: 0.0369 acc: 61.7188 time: 1702118694
[step:  4700] loss: 0.0369 acc: 61.7188 time: 1702118699
[step:  4800] loss: 0.0364 acc: 64.0625 time: 1702118704
[step:  4900] loss: 0.0366 acc: 63.2812 time: 1702118709
[step:  5000] loss: 0.0378 acc: 57.0312 time: 1702118714
[step:  5100] loss: 0.0358 acc: 67.1875 time: 1702118719
[step:  5200] loss: 0.0366 acc: 63.2812 time: 1702118724
[step:  5300] loss: 0.0355 acc: 68.7500 time: 1702118728
[step:  5400] loss: 0.0363 acc: 64.8438 time: 1702118734
[step:  5500] loss: 0.0372 acc: 60.1562 time: 1702118739
[step:  5600] loss: 0.0369 acc: 61.7188 time: 1702118744
[step:  5700] loss: 0.0384 acc: 53.9062 time: 1702118749
[step:  5800] loss: 0.0367 acc: 62.5000 time: 1702118754
[step:  5900] loss: 0.0378 acc: 57.0312 time: 1702118759
[step:  6000] loss: 0.0375 acc: 58.5938 time: 1702118764
[step:  6100] loss: 0.0358 acc: 67.1875 time: 1702118769
[step:  6200] loss: 0.0383 acc: 54.6875 time: 1702118774
```

```
[step:  6300] loss: 0.0375 acc: 58.5938 time: 1702118779
[step:  6400] loss: 0.0364 acc: 64.0625 time: 1702118784
[step:  6500] loss: 0.0377 acc: 57.8125 time: 1702118788
[step:  6600] loss: 0.0373 acc: 59.3750 time: 1702118793
[step:  6700] loss: 0.0370 acc: 60.9375 time: 1702118798
[step:  6800] loss: 0.0375 acc: 58.5938 time: 1702118802
[step:  6900] loss: 0.0370 acc: 60.9375 time: 1702118808
[step:  7000] loss: 0.0375 acc: 58.5938 time: 1702118812
[step:  7100] loss: 0.0381 acc: 55.4688 time: 1702118817
[step:  7200] loss: 0.0369 acc: 61.7188 time: 1702118822
[step:  7300] loss: 0.0367 acc: 62.5000 time: 1702118826
[step:  7400] loss: 0.0373 acc: 59.3750 time: 1702118832
[step:  7500] loss: 0.0361 acc: 65.6250 time: 1702118836
[step:  7600] loss: 0.0363 acc: 64.8438 time: 1702118841
[step:  7700] loss: 0.0391 acc: 50.7812 time: 1702118846
[step:  7800] loss: 0.0377 acc: 57.8125 time: 1702118850
[step:  7900] loss: 0.0383 acc: 54.6875 time: 1702118855
[step:  8000] loss: 0.0381 acc: 55.4688 time: 1702118860
[step:  8100] loss: 0.0370 acc: 60.9375 time: 1702118864
[step:  8200] loss: 0.0366 acc: 63.2812 time: 1702118869
[step:  8300] loss: 0.0375 acc: 58.5938 time: 1702118874
[step:  8400] loss: 0.0377 acc: 57.8125 time: 1702118878
[step:  8500] loss: 0.0366 acc: 63.2812 time: 1702118883
[step:  8600] loss: 0.0378 acc: 57.0312 time: 1702118888
[step:  8700] loss: 0.0381 acc: 55.4688 time: 1702118893
[step:  8800] loss: 0.0372 acc: 60.1562 time: 1702118898
[step:  8900] loss: 0.0372 acc: 60.1562 time: 1702118902
[step:  9000] loss: 0.0367 acc: 62.5000 time: 1702118908
[step:  9100] loss: 0.0366 acc: 63.2812 time: 1702118912
[step:  9200] loss: 0.0361 acc: 65.6250 time: 1702118917
[step:  9300] loss: 0.0372 acc: 60.1562 time: 1702118922
[step:  9400] loss: 0.0372 acc: 60.1562 time: 1702118926
[step:  9500] loss: 0.0370 acc: 60.9375 time: 1702118931
[step:  9600] loss: 0.0369 acc: 61.7188 time: 1702118936
[step:  9700] loss: 0.0370 acc: 60.9375 time: 1702118940
[step:  9800] loss: 0.0377 acc: 57.8125 time: 1702118945
[step:  9900] loss: 0.0386 acc: 53.1250 time: 1702118950
[step: 10000] loss: 0.0373 acc: 59.3750 time: 1702118955
Finished Training

# Plot the losses
plot_loss(losses, title='Training Loss [N=15]', path=results_path +
'training_loss_15_lstm.png')

# Plot the accuracies
plot_accuracy(accuracies, title='Training Accuracy [N=15]',
path=results_path + 'training_accuracy_15_lstm.png')
```



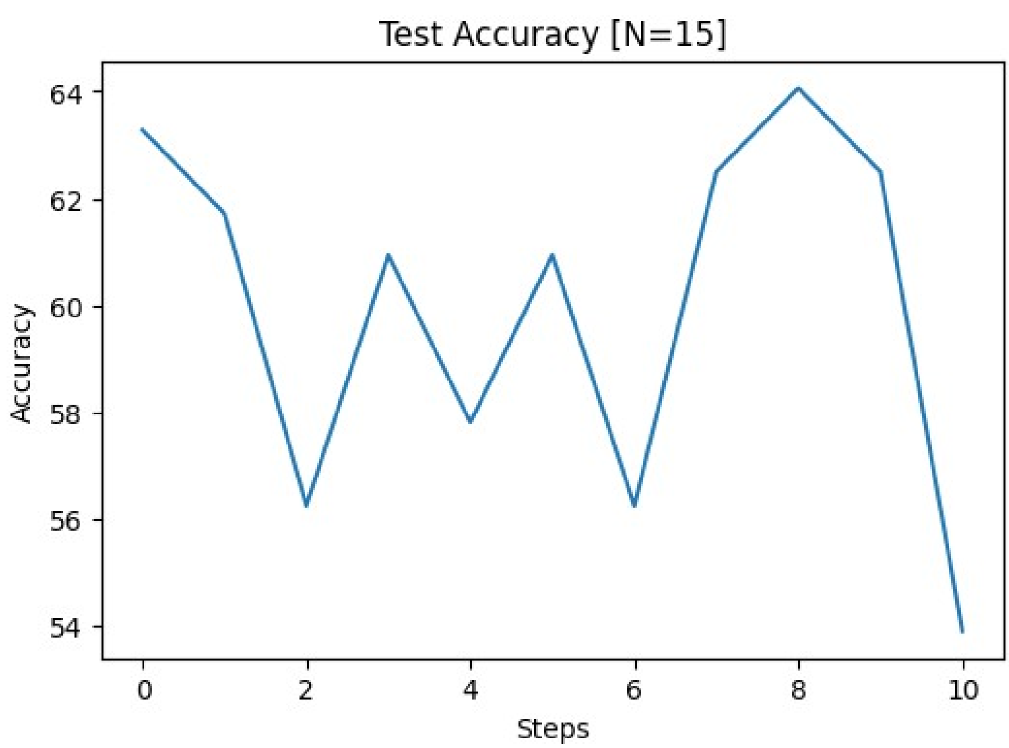



```
# Test the model
test_accuracies = test(model, input_length=p1[2], config=config,
device=device)
```

```python
# Add accuracies
p3_acc_lstm.append(np.mean(test_accuracies))
```

```
Accuracy: 63.2812
Accuracy: 61.7188
Accuracy: 56.2500
Accuracy: 60.9375
Accuracy: 57.8125
Accuracy: 60.9375
Accuracy: 56.2500
Accuracy: 62.5000
Accuracy: 64.0625
Accuracy: 62.5000
Accuracy: 53.9062
Finished Testing
```

```python
# plot the test accuracies
plot_accuracy(test_accuracies, title='Test Accuracy [N=15]',
path=results_path + 'test_accuracy_15_lstm.png')

# Average accuracy over all Steps
print(f"Average test accuracy: {np.mean(test_accuracies):.2f}%")
```




```
Average test accuracy: 60.01%
```

```python
# Train the model T=20
model, losses, accuracies = train(config, input_length=p1[3],
lr=config['lstm_learning_rate'], type='LSTM', device=device)
```

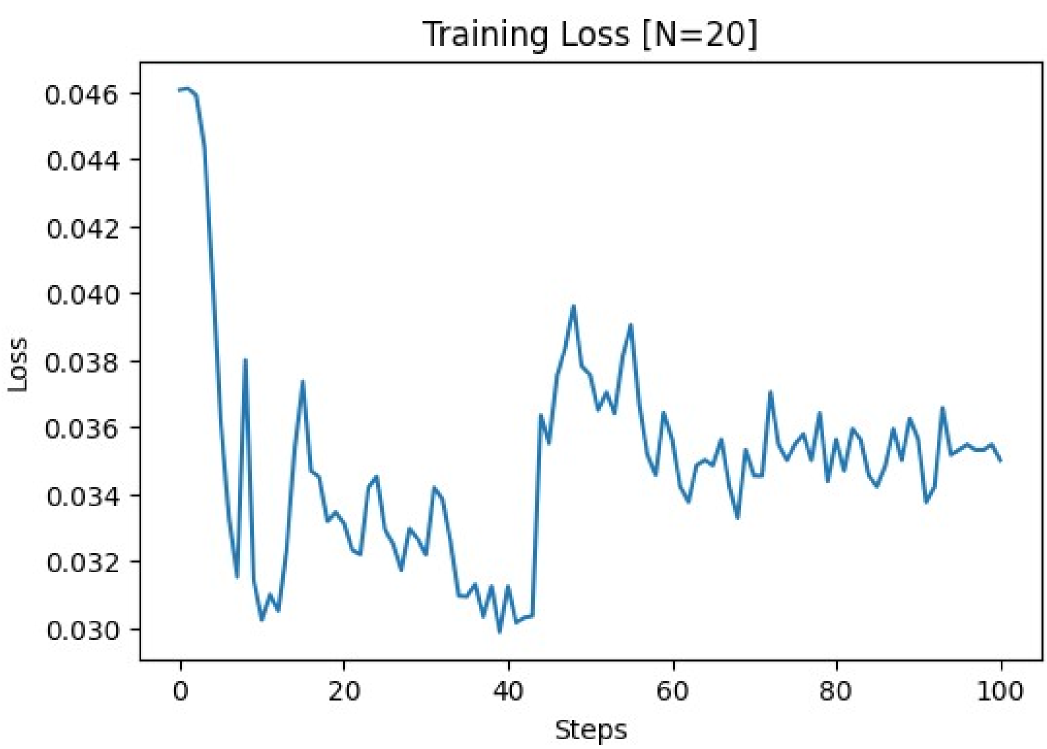
```
[step:     0] loss: 0.0461 acc: 12.5000 time: 1702118958
```

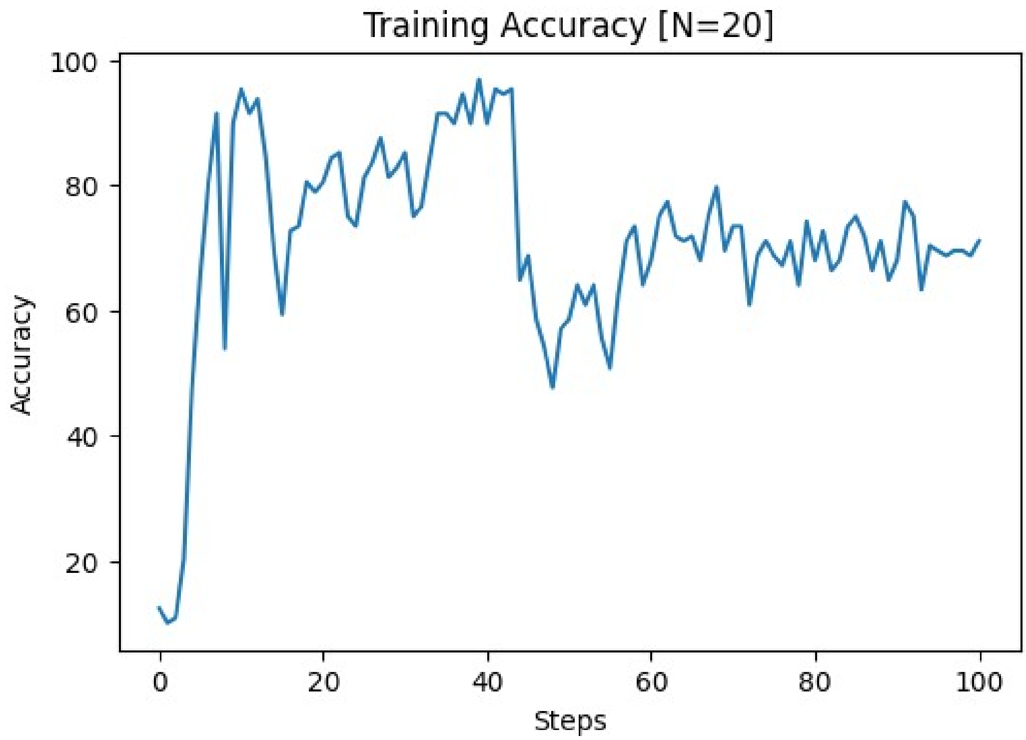
```
<ipython-input-6-042aa7a8086d>:64: UserWarning:
torch.nn.utils.clip_grad_norm is now deprecated in favor of
torch.nn.utils.clip_grad_norm_.
  nn.utils.clip_grad_norm(model.parameters(),
max_norm=config['max_norm'])
```

```
[step:   100] loss: 0.0461 acc: 10.1562 time: 1702118964
[step:   200] loss: 0.0459 acc: 10.9375 time: 1702118971
[step:   300] loss: 0.0443 acc: 20.3125 time: 1702118976
[step:   400] loss: 0.0404 acc: 47.6562 time: 1702118983
[step:   500] loss: 0.0361 acc: 65.6250 time: 1702118989
[step:   600] loss: 0.0333 acc: 80.4688 time: 1702118995
[step:   700] loss: 0.0315 acc: 91.4062 time: 1702119002
[step:   800] loss: 0.0380 acc: 53.9062 time: 1702119008
[step:   900] loss: 0.0314 acc: 89.8438 time: 1702119014
[step:  1000] loss: 0.0302 acc: 95.3125 time: 1702119020
[step:  1100] loss: 0.0310 acc: 91.4062 time: 1702119026
[step:  1200] loss: 0.0305 acc: 93.7500 time: 1702119032
[step:  1300] loss: 0.0323 acc: 84.3750 time: 1702119038
[step:  1400] loss: 0.0353 acc: 69.5312 time: 1702119045
[step:  1500] loss: 0.0374 acc: 59.3750 time: 1702119050
[step:  1600] loss: 0.0347 acc: 72.6562 time: 1702119057
[step:  1700] loss: 0.0345 acc: 73.4375 time: 1702119062
[step:  1800] loss: 0.0332 acc: 80.4688 time: 1702119069
[step:  1900] loss: 0.0335 acc: 78.9062 time: 1702119074
[step:  2000] loss: 0.0331 acc: 80.4688 time: 1702119081
[step:  2100] loss: 0.0323 acc: 84.3750 time: 1702119087
[step:  2200] loss: 0.0322 acc: 85.1562 time: 1702119093
[step:  2300] loss: 0.0342 acc: 75.0000 time: 1702119099
[step:  2400] loss: 0.0345 acc: 73.4375 time: 1702119105
[step:  2500] loss: 0.0329 acc: 81.2500 time: 1702119110
[step:  2600] loss: 0.0325 acc: 83.5938 time: 1702119117
[step:  2700] loss: 0.0317 acc: 87.5000 time: 1702119123
[step:  2800] loss: 0.0330 acc: 81.2500 time: 1702119129
[step:  2900] loss: 0.0327 acc: 82.8125 time: 1702119135
[step:  3000] loss: 0.0322 acc: 85.1562 time: 1702119141
[step:  3100] loss: 0.0342 acc: 75.0000 time: 1702119146
[step:  3200] loss: 0.0338 acc: 76.5625 time: 1702119153
[step:  3300] loss: 0.0326 acc: 84.3750 time: 1702119158
[step:  3400] loss: 0.0310 acc: 91.4062 time: 1702119165
[step:  3500] loss: 0.0309 acc: 91.4062 time: 1702119171
[step:  3600] loss: 0.0313 acc: 89.8438 time: 1702119177
[step:  3700] loss: 0.0303 acc: 94.5312 time: 1702119183
[step:  3800] loss: 0.0312 acc: 89.8438 time: 1702119190
```

```
[step:  3900] loss: 0.0299 acc: 96.8750 time: 1702119196
[step:  4000] loss: 0.0312 acc: 89.8438 time: 1702119202
[step:  4100] loss: 0.0302 acc: 95.3125 time: 1702119208
[step:  4200] loss: 0.0303 acc: 94.5312 time: 1702119214
[step:  4300] loss: 0.0304 acc: 95.3125 time: 1702119220
[step:  4400] loss: 0.0363 acc: 64.8438 time: 1702119226
[step:  4500] loss: 0.0355 acc: 68.7500 time: 1702119232
[step:  4600] loss: 0.0375 acc: 58.5938 time: 1702119239
[step:  4700] loss: 0.0384 acc: 53.9062 time: 1702119245
[step:  4800] loss: 0.0396 acc: 47.6562 time: 1702119251
[step:  4900] loss: 0.0378 acc: 57.0312 time: 1702119257
[step:  5000] loss: 0.0376 acc: 58.5938 time: 1702119263
[step:  5100] loss: 0.0365 acc: 64.0625 time: 1702119269
[step:  5200] loss: 0.0370 acc: 60.9375 time: 1702119275
[step:  5300] loss: 0.0364 acc: 64.0625 time: 1702119281
[step:  5400] loss: 0.0381 acc: 55.4688 time: 1702119288
[step:  5500] loss: 0.0390 acc: 50.7812 time: 1702119293
[step:  5600] loss: 0.0367 acc: 62.5000 time: 1702119300
[step:  5700] loss: 0.0352 acc: 71.0938 time: 1702119306
[step:  5800] loss: 0.0346 acc: 73.4375 time: 1702119312
[step:  5900] loss: 0.0364 acc: 64.0625 time: 1702119318
[step:  6000] loss: 0.0356 acc: 67.9688 time: 1702119324
[step:  6100] loss: 0.0342 acc: 75.0000 time: 1702119330
[step:  6200] loss: 0.0338 acc: 77.3438 time: 1702119337
[step:  6300] loss: 0.0349 acc: 71.8750 time: 1702119343
[step:  6400] loss: 0.0350 acc: 71.0938 time: 1702119349
[step:  6500] loss: 0.0349 acc: 71.8750 time: 1702119355
[step:  6600] loss: 0.0356 acc: 67.9688 time: 1702119361
[step:  6700] loss: 0.0342 acc: 75.0000 time: 1702119367
[step:  6800] loss: 0.0333 acc: 79.6875 time: 1702119373
[step:  6900] loss: 0.0353 acc: 69.5312 time: 1702119379
[step:  7000] loss: 0.0345 acc: 73.4375 time: 1702119386
[step:  7100] loss: 0.0345 acc: 73.4375 time: 1702119391
[step:  7200] loss: 0.0370 acc: 60.9375 time: 1702119398
[step:  7300] loss: 0.0355 acc: 68.7500 time: 1702119404
[step:  7400] loss: 0.0350 acc: 71.0938 time: 1702119410
[step:  7500] loss: 0.0355 acc: 68.7500 time: 1702119416
[step:  7600] loss: 0.0358 acc: 67.1875 time: 1702119422
[step:  7700] loss: 0.0350 acc: 71.0938 time: 1702119428
[step:  7800] loss: 0.0364 acc: 64.0625 time: 1702119435
[step:  7900] loss: 0.0344 acc: 74.2188 time: 1702119440
[step:  8000] loss: 0.0356 acc: 67.9688 time: 1702119447
[step:  8100] loss: 0.0347 acc: 72.6562 time: 1702119452
[step:  8200] loss: 0.0359 acc: 66.4062 time: 1702119459
[step:  8300] loss: 0.0356 acc: 67.9688 time: 1702119465
[step:  8400] loss: 0.0345 acc: 73.4375 time: 1702119472
[step:  8500] loss: 0.0342 acc: 75.0000 time: 1702119478
[step:  8600] loss: 0.0348 acc: 71.8750 time: 1702119485
[step:  8700] loss: 0.0359 acc: 66.4062 time: 1702119490
```

```
[step:  8800] loss: 0.0350 acc: 71.0938 time: 1702119497
[step:  8900] loss: 0.0363 acc: 64.8438 time: 1702119503
[step:  9000] loss: 0.0356 acc: 67.9688 time: 1702119510
[step:  9100] loss: 0.0338 acc: 77.3438 time: 1702119515
[step:  9200] loss: 0.0342 acc: 75.0000 time: 1702119522
[step:  9300] loss: 0.0366 acc: 63.2812 time: 1702119527
[step:  9400] loss: 0.0352 acc: 70.3125 time: 1702119534
[step:  9500] loss: 0.0353 acc: 69.5312 time: 1702119539
[step:  9600] loss: 0.0355 acc: 68.7500 time: 1702119546
[step:  9700] loss: 0.0353 acc: 69.5312 time: 1702119552
[step:  9800] loss: 0.0353 acc: 69.5312 time: 1702119559
[step:  9900] loss: 0.0355 acc: 68.7500 time: 1702119564
[step: 10000] loss: 0.0350 acc: 71.0938 time: 1702119571
Finished Training

# Plot the losses
plot_loss(losses, title='Training Loss [N=20]', path=results_path +
'training_loss_20_lstm.png')

# Plot the accuracies
plot_accuracy(accuracies, title='Training Accuracy [N=20]',
path=results_path + 'training_accuracy_20_lstm.png')
```

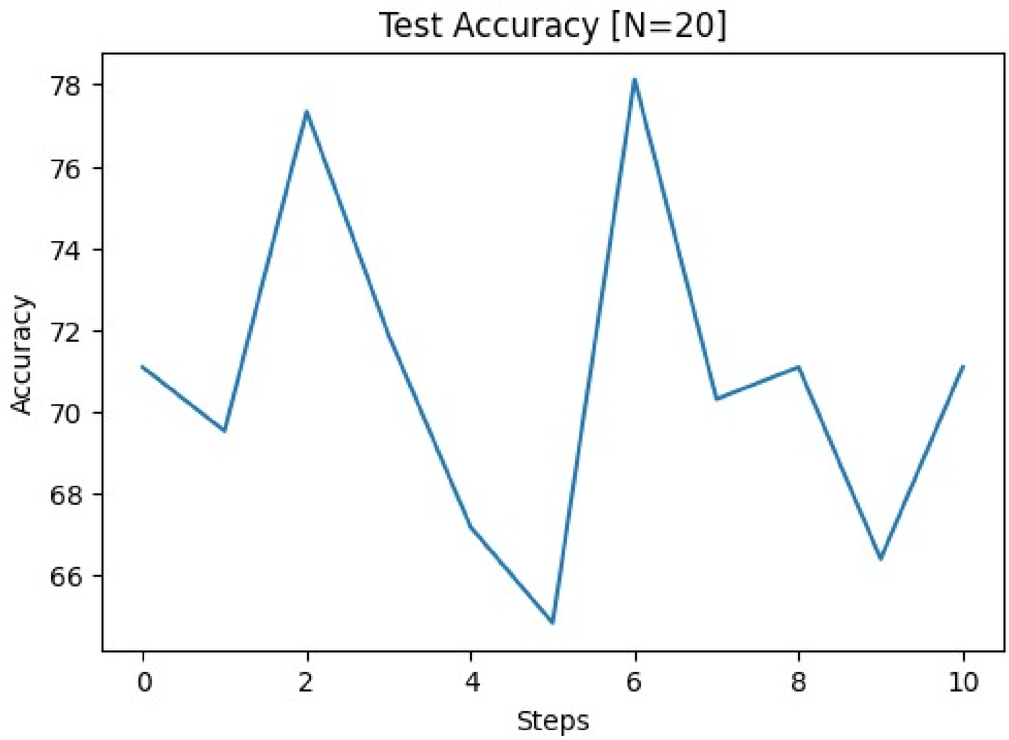





```
# Test the model
test_accuracies = test(model, input_length=p1[3], config=config,
device=device)

# Add accuracies
p3_acc_lstm.append(np.mean(test_accuracies))
```

```
Accuracy: 71.0938
Accuracy: 69.5312
Accuracy: 77.3438
Accuracy: 71.8750
Accuracy: 67.1875
Accuracy: 64.8438
Accuracy: 78.1250
Accuracy: 70.3125
Accuracy: 71.0938
Accuracy: 66.4062
Accuracy: 71.0938
Finished Testing
```

```
# plot the test accuracies
plot_accuracy(test_accuracies, title='Test Accuracy [N=20]',
path=results_path + 'test_accuracy_20_lstm.png')

# Average accuracy over all Steps
print(f"Average test accuracy: {np.mean(test_accuracies):.2f}%")
```




```
Average test accuracy: 70.81%
```

Question 1.6: Comparison LSTM & Vanilla RNN

Write down a comparison with the vanilla RNN and think of reasons for the different behavior on the Palindrome prediction task.

Convergence speed:

- LSTM network generally converges faster than the Vanilla RNN during training due to its gating mechanism, which helps mitigate the vanishing gradient problem. The Vanilla RNN is more susceptible to the Vanishing Gradient problem since it's gradients have to be propagated through sevaral learnable layers who's values might become arbitrarily small or large. For both training loops a gradient clipping is used to prevent the models from exploding gradients.

Performance with longer sequences:

- Vanilla RNNs: The accuracy tends to decrease, and the loss increases during training with longer sequences due to the vanishing gradient problem. Gradient clipping is used to prevent gradients from exploding, but it does not address the vanishing gradient issue noticable in Vanilla RNNs which prevents the Vanilla RNN to *memorise* the necessary information from an earlier timestep in order to solve the palindrome task.
- LSTM network: The LSTM also exhibits reduced accuracy with increasing lengths of palindrome sequences. However, in comparison to the Vanilla RNN this is less pronounced. The network needs to learn parameters to preserve the right amount of information from a very early stage onwards, which can be challenging.

LSTM's struggle with updating information:

- It is possible that the LSTM network struggles to learn to update the right information and not remove relevant signals, especially when dealing with longer sequences. This might be due to the difficulty in optimizing the complex interactions between the input, forget, and output gates.

```
plt.figure(figsize=(5,3))

plt.title('Test accuracy w.r.t. palindromes length.')
# Plot RNN avg. test acc
plt.plot(p1, p2_acc_rnn, color='red', label='RNN')
# Plot LSTM avg. test acc
plt.plot(p1, p3_acc_lstm, color='green', label='LSTM')

plt.xlabel('Palindromes Length')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```