In [1]:

```
import torch
# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device
```

Out[1]:

```
device(type='cpu')
```

In [2]:

```
from torchvision import datasets, transforms
from torchvision.transforms import ToTensor
from torch import nn
import torch.nn.functional as F
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

In [21]:

```
learning_rate = 0.001
batch_size    = 100
N_epochs      = 20
N_iter_train  = 250
N_iter_test   = 40
N_iter_check  = 5
input_size    = 10

hidden_size   = 10
```

In [30]:

```
def sequence_generator(L,batch_size,K = 1): #K=1, the model has to remember value at 2n
d position

    random_seq = np.random.randint(0, 9,(batch_size, L)) #generated random number seque
nce

    x = np.zeros((batch_size,L,10)) #second dimension is 10 as we're looking at one-hot
vectors
    y = np.zeros((batch_size,10)) #output

    for i in range(batch_size):
        x[i,np.arange(L), random_seq[i]] = 1
        y[i,random_seq[i,K]] = 1

    #converting to torch
    random_seq = torch.tensor(random_seq, dtype=torch.float)
    x = torch.tensor(x, dtype=torch.int)
    y = torch.tensor(y, dtype=torch.float)


    return random_seq,x.float(),y #as input x is reqd to be float
```

In [23]:

```
class vanilla_RNN(nn.Module):
    def __init__(self,hidden_size):

        super().__init__()

        self.rnn = nn.RNN(input_size = 10,
                          hidden_size = hidden_size,
                          num_layers = 1,
                          bidirectional = False,
                          batch_first = True)
        self.fc = nn.Linear(hidden_size, 10)

    def forward(self, x):

        output, hidden = self.rnn(x)
        out = self.fc(output[:,-1,:])

        return out
```

In [24]:

```
from torch import optim
loss_func = nn.CrossEntropyLoss()
model = vanilla_RNN(hidden_size).to(device)
optimizer = optim.Adam(model.parameters(), lr = 0.001)
```

In [25]:

```
def Train_sequence(model,optimizer,loss_func,N_iter_train = 250):

    model.train() #setting the model in training mode
    #initializing the total training loss and total correct training predictions to 0
    train_loss    = 0
    train_correct = 0 #correct predictions made

    train_length = batch_size*N_iter_train

    for i in range(N_iter_train):

        L = np.random.randint(3,10) #randomizing L

        random_seq,x,y = sequence_generator(L)
        x = x.to(device)
        y = y.to(device)

        pred = model(x) #prediction using the input data

        loss = loss_func(pred,y)

        optimizer.zero_grad() #zeroing out the gradients before backprop
        loss.backward()       #backprop from the loss
        optimizer.step()      #updating the weights

        pred = pred.cpu()
        loss = loss.cpu()

        #Adding this loss to  training loss and computing correct predictions
        train_loss    += loss
        train_correct += (np.asarray(pred.argmax(axis = 1)-y.cpu().argmax(axis = 1))==0
).sum() #as subtraction will result in 0 for correct pred

    #Computing training accuracy
    train_loss = train_loss/(i+1)
    train_correct /= train_length #training accuracy

    return train_loss.detach().cpu().numpy(), train_correct #returning loss and accurac
y
```

In [26]:

```
def Test_sequence(model,loss_func,N_iter_test = N_iter_test):

    model.eval()  #setting the model in eval/test mode

    #initializing the total test loss and total correct test predictions to 0
    test_loss    = 0
    test_correct = 0 #correct predictions made
    test_length = batch_size*N_iter_test

    #switching off the gradient for eval
    with torch.no_grad():

        for i in range(N_iter_test):

            L = np.random.randint(3,10) #randomizing L

            random_seq,x,y = sequence_generator(L)
            x = x.to(device)
            y = y.to(device)

            pred = model(x) #prediction using the input data

            loss = loss_func(pred,y.argmax(axis = 1))

            pred = pred.cpu()
            loss = loss.cpu()

            #Adding this loss to  testing loss and computing correct predictions
            test_loss    += loss
            test_correct += (np.asarray(pred.argmax(axis = 1)-y.cpu().argmax(axis = 1))
==0).sum() #as subtraction will result in 0 for correct pred

    #Computing prediction accuracy
    test_loss = test_loss/(i+1)
    test_correct /= test_length #prediction accuracy

    return test_loss.detach().cpu().numpy(), test_correct #returning loss and accuracy
```

In [27]:

```
#initialising the lists
train_losses   = []
test_losses    = []
train_accuracy = []
test_accuracy  = []

#for epoch in range(1, N_epochs+1):
for epoch in range(1, N_epochs+1):
    print(epoch,"/", N_epochs)

    #train the model
    loss,accuracy = Train_sequence(model,optimizer,loss_func,N_iter_train = N_iter_trai
n)
    train_losses.append(loss)
    train_accuracy.append(accuracy)
    print('Train loss for Epoch ',epoch,': ',loss,' | ', 'Train accuracy for Epoch ',ep
och, ': ',accuracy)

    #test the model
    loss,accuracy = Test_sequence(model,loss_func,N_iter_test = N_iter_test)
    test_losses.append(loss)
    test_accuracy.append(accuracy)
    print('Test loss for Epoch ',epoch,': ',loss, ' | ', 'Test accuracy for Epoch ',epo
ch, ': ',accuracy)

#Plotting the Loss and the accuracy curves
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.plot(np.asfarray(train_losses),label = 'Train Loss') #converting to float array
ax1.plot(np.asfarray(test_losses),label = 'Validation Loss')
ax1.set(xlabel='Epochs', ylabel='Normalised Loss')
ax1.set_title('Training and Validation error')

ax2.plot(np.asfarray(train_accuracy),label = 'Training Accuracy')
ax2.plot(np.asfarray(test_accuracy),label = 'Testing Accuracy')
ax2.set(xlabel='Epochs', ylabel='Accuracy')
ax2.set_title('Training and Validation accuracy')
ax2.legend()
fig.tight_layout()
```

```
1 / 20
Train loss for Epoch  1 :  2.2575984  |  Train accuracy for Epoch  1 :  0.
12164
Test loss for Epoch  1 :  2.2061765  |  Test accuracy for Epoch  1 :  0.13
1
2 / 20
Train loss for Epoch  2 :  2.1297002  |  Train accuracy for Epoch  2 :  0.
17872
Test loss for Epoch  2 :  2.0904086  |  Test accuracy for Epoch  2 :  0.20
575
3 / 20
Train loss for Epoch  3 :  2.0817266  |  Train accuracy for Epoch  3 :  0.
20388
Test loss for Epoch  3 :  2.0877497  |  Test accuracy for Epoch  3 :  0.19
9
4 / 20
Train loss for Epoch  4 :  2.0503883  |  Train accuracy for Epoch  4 :  0.
2298
Test loss for Epoch  4 :  2.0093217  |  Test accuracy for Epoch  4 :  0.24
225
5 / 20
Train loss for Epoch  5 :  1.9422538  |  Train accuracy for Epoch  5 :  0.
272
Test loss for Epoch  5 :  1.847606  |  Test accuracy for Epoch  5 :  0.307
25
6 / 20
Train loss for Epoch  6 :  1.6689517  |  Train accuracy for Epoch  6 :  0.
34336
Test loss for Epoch  6 :  1.5011898  |  Test accuracy for Epoch  6 :  0.38
525
7 / 20
Train loss for Epoch  7 :  1.3966227  |  Train accuracy for Epoch  7 :  0.
424
Test loss for Epoch  7 :  1.2967812  |  Test accuracy for Epoch  7 :  0.45
8 / 20
Train loss for Epoch  8 :  1.213383  |  Train accuracy for Epoch  8 :  0.4
9148
Test loss for Epoch  8 :  1.1543994  |  Test accuracy for Epoch  8 :  0.52
625
9 / 20
Train loss for Epoch  9 :  1.0568869  |  Train accuracy for Epoch  9 :  0.
5812
Test loss for Epoch  9 :  0.99806774  |  Test accuracy for Epoch  9 :  0.6
425
10 / 20
Train loss for Epoch  10 :  0.9453788  |  Train accuracy for Epoch  10 :
0.6408
Test loss for Epoch  10 :  0.9335408  |  Test accuracy for Epoch  10 :  0.
6575
11 / 20
Train loss for Epoch  11 :  0.8235563  |  Train accuracy for Epoch  11 :
0.68824
Test loss for Epoch  11 :  0.82683456  |  Test accuracy for Epoch  11 :
0.69225
12 / 20
Train loss for Epoch  12 :  0.7469852  |  Train accuracy for Epoch  12 :
0.725
Test loss for Epoch  12 :  0.63433844  |  Test accuracy for Epoch  12 :
0.76125
13 / 20
Train loss for Epoch  13 :  0.6672661  |  Train accuracy for Epoch  13 :
```

```
0.7534
Test loss for Epoch  13 :  0.5742372  |  Test accuracy for Epoch  13 :  0.
773
14 / 20
Train loss for Epoch  14 :  0.6066345  |  Train accuracy for Epoch  14 :
0.77472
Test loss for Epoch  14 :  0.5709286  |  Test accuracy for Epoch  14 :  0.
78625
15 / 20
Train loss for Epoch  15 :  0.56055045  |  Train accuracy for Epoch  15 :
0.8018
Test loss for Epoch  15 :  0.53982866  |  Test accuracy for Epoch  15 :
0.808
16 / 20
Train loss for Epoch  16 :  0.48291773  |  Train accuracy for Epoch  16 :
0.8354
Test loss for Epoch  16 :  0.47338468  |  Test accuracy for Epoch  16 :
0.8715
17 / 20
Train loss for Epoch  17 :  0.39103648  |  Train accuracy for Epoch  17 :
0.92376
Test loss for Epoch  17 :  0.37058526  |  Test accuracy for Epoch  17 :
0.94375
18 / 20
Train loss for Epoch  18 :  0.30331954  |  Train accuracy for Epoch  18 :
0.95852
Test loss for Epoch  18 :  0.2590236  |  Test accuracy for Epoch  18 :  0.
979
19 / 20
Train loss for Epoch  19 :  0.24369499  |  Train accuracy for Epoch  19 :
0.97636
Test loss for Epoch  19 :  0.24138863  |  Test accuracy for Epoch  19 :
0.98075
20 / 20
Train loss for Epoch  20 :  0.21773176  |  Train accuracy for Epoch  20 :
0.98348
Test loss for Epoch  20 :  0.22008765  |  Test accuracy for Epoch  20 :
0.983
```

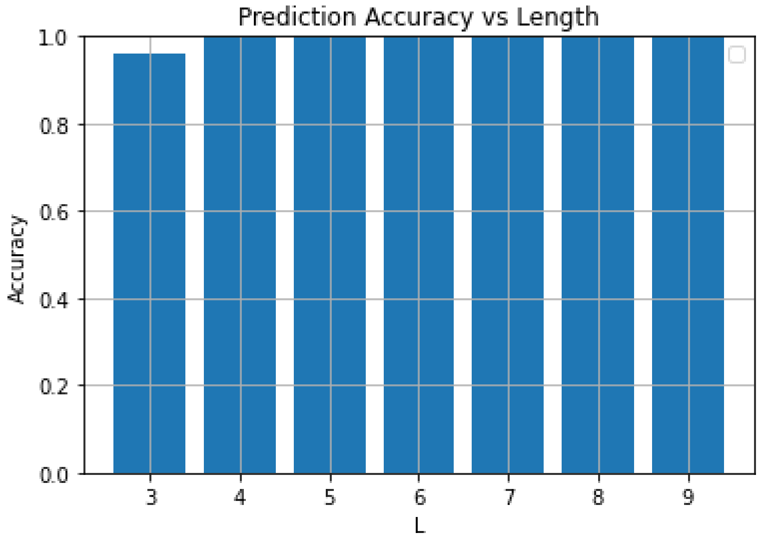
In [33]:

```
def Check_sequence(model,lossfn,N_iter_check):

    model.eval()

    test_accuracies = []

    for L in range(3,10): #iterating through L in the required range

        #initializing the total test loss and total correct test predictions to 0
        test_loss    = 0
        test_correct = 0 #correct predictions made
        test_length = N_iter_check

        #switching off the gradient for eval
        with torch.no_grad():


            random_seq,x,y = sequence_generator(L,N_iter_check)
            x = x.to(device)
            y = y.to(device)

            pred = model(x)

            loss = loss_func(pred,y.argmax(axis = 1))

            pred = pred.cpu()
            loss = loss.cpu()

            #Adding this loss to  testing loss and computing correct predictions
            test_loss    += loss
            test_correct += (np.asarray(pred.argmax(axis = 1)-y.cpu().argmax(axis = 1))
==0).sum() #as subtraction will result in 0 for correct pred

        #Computing prediction accuracy

        test_correct /= test_length #prediction accuracy

        test_accuracies.append(test_correct)


    #plotting test accuracies vs L

    plt.bar(np.arange(3,10),test_accuracies)
    plt.xlabel('L')
    plt.ylabel('Accuracy')
    plt.ylim(0,1) #as accuracy is between 0 and 1
    plt.grid()
    plt.legend()
    plt.title('Prediction Accuracy vs Length')
    plt.show()

    print(f'Prediction Accuracies : {test_accuracies}')
```

In [35]:

```
Check_sequence(model,loss_func,100)
```

```
WARNING:matplotlib.legend:No handles with labels found to put in legend.
```




```
Prediction Accuracies : [0.96, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```