Forward Pass of RNN

```
import numpy as np


def softmax(x):
  e_x = np.exp(x - np.max(x))
  return e_x / e_x.sum(axis=0)


def sigmoid(x):
  return 1 / (1 + np.exp(-x))


def CellForwardRNN(InputData, PreviousState, Parameters):
  Wax = Parameters["Wax"]
  Waa = Parameters["Waa"]
  Wya = Parameters["Wya"]
  ba = Parameters["ba"]
  by = Parameters["by"]

  NextState = np.tanh(np.dot(Wax, InputData) + np.dot(Waa, PreviousState) + ba)
  Output = softmax(np.dot(Wya, NextState) + by)
  Cache = (NextState, PreviousState, InputData, Parameters)

  return NextState, Output, Cache


np.random.seed(1)
InputData = np.random.randn(3,10)
PreviousState = np.random.randn(5,10)
Waa = np.random.randn(5,5)
Wax = np.random.randn(5,3)
Wya = np.random.randn(2,5)
ba = np.random.randn(5,1)
by = np.random.randn(2,1)
Parameters = {"Waa": Waa, "Wax": Wax, "Wya": Wya, "ba": ba, "by": by}

NextState, Output, cache = CellForwardRNN(InputData, PreviousState, Parameters)
print("NextStatet[4] = ", NextState[4])
print("NextState.shape = ", NextState.shape)
print("Output[1] =", Output[1])
print("Output.shape = ", Output.shape)
```

```
NextStatet[4] =  [ 0.59584544  0.18141802  0.61311866  0.99808218  0.85016201  0.999809
 -0.18887155  0.99815551  0.6531151   0.82872037]
NextState.shape =  (5, 10)
Output[1] = [0.9888161  0.01682021 0.21140899 0.36817467 0.98988387 0.88945212
 0.36920224 0.9966312  0.9982559  0.17746526]
Output.shape =  (2, 10)
```

```
def ForwardPassRNN(InputData, InitialHiddenState, Parameters):
  Caches = []

  n_x, m, T_x = InputData.shape
  n_y, n_a = Parameters["Wya"].shape
  a = np.zeros((n_a, m, T_x))
  y_pred = np.zeros((n_y, m, T_x))
  NextState = InitialHiddenState
  for t in range(T_x):
      NextState, Output, Cache = CellForwardRNN(InputData[:,:,t], NextState, Parameters)
      a[:,:,t] = NextState
      y_pred[:,:,t] = Output
      Caches.append(Cache)

  Caches = (Caches, InputData)

  return a, y_pred, Caches


np.random.seed(1)
InputData = np.random.randn(3,10,4)
InitialHiddenState = np.random.randn(5,10)
Waa = np.random.randn(5,5)
Wax = np.random.randn(5,3)
Wya = np.random.randn(2,5)
ba = np.random.randn(5,1)
by = np.random.randn(2,1)
Parameters = {"Waa": Waa, "Wax": Wax, "Wya": Wya, "ba": ba, "by": by}

a, Output, Caches = ForwardPassRNN(InputData, InitialHiddenState, Parameters)
print("a[4][1] = ", a[4][1])
print("a.shape = ", a.shape)
print("Output[1][3] =", Output[1][3])
print("Output.shape = ", Output.shape)
print("Caches[1][1][3] =", Caches[1][1][3])
print("len(Caches) = ", len(Caches))
```

```
a[4][1] =  [-0.99999375  0.77911235 -0.99861469 -0.99833267]
a.shape =  (5, 10, 4)
Output[1][3] = [0.79560373 0.86224861 0.11118257 0.81515947]
Output.shape =  (2, 10, 4)
Caches[1][1][3] = [-1.1425182  -0.34934272 -0.20889423  0.58662319]
len(Caches) =  2
```

Error Backpropogation through time

```
def CellBackwardRNN(GradLoss, Cache):
```

```
    (NextState, PreviousState, InputData, Parameters) = Cache
    Wax = Parameters["Wax"]
    Waa = Parameters["Waa"]
    Wya = Parameters["Wya"]
    ba = Parameters["ba"]
    by = Parameters["by"]
    dtanh = (1 - NextState ** 2) * GradLoss

    dxt = np.dot(Wax.T, dtanh)
    dWax = np.dot(dtanh, InputData.T)

    da_prev = np.dot(Waa.T, dtanh)
    dWaa = np.dot(dtanh, PreviousState.T)

    dba = np.sum(dtanh, axis = 1,keepdims=1)

    Gradients = {"dxt": dxt, "da_prev": da_prev, "dWax": dWax, "dWaa": dWaa, "dba": dba}

    return Gradients


np.random.seed(1)
InputData = np.random.randn(3,10)
PreviousState = np.random.randn(5,10)
Wax = np.random.randn(5,3)
Waa = np.random.randn(5,5)
Wya = np.random.randn(2,5)
b = np.random.randn(5,1)
by = np.random.randn(2,1)
Parameters = {"Wax": Wax, "Waa": Waa, "Wya": Wya, "ba": ba, "by": by}

NextState, Output, Cache = CellForwardRNN(InputData, PreviousState, Parameters)

da_next = np.random.randn(5,10)
Gradients = CellBackwardRNN(da_next, cache)
print("Gradients[\"dxt\"][1][2] =", Gradients["dxt"][1][2])
print("Gradients[\"dxt\"].shape =", Gradients["dxt"].shape)
print("Gradients[\"da_prev\"][2][3] =", Gradients["da_prev"][2][3])
print("Gradients[\"da_prev\"].shape =", Gradients["da_prev"].shape)
print("Gradients[\"dWax\"][3][1] =", Gradients["dWax"][3][1])
print("Gradients[\"dWax\"].shape =", Gradients["dWax"].shape)
print("Gradients[\"dWaa\"][1][2] =", Gradients["dWaa"][1][2])
print("Gradients[\"dWaa\"].shape =", Gradients["dWaa"].shape)
print("Gradients[\"dba\"][4] =", Gradients["dba"][4])
print("Gradients[\"dba\"].shape =", Gradients["dba"].shape)

     Gradients["dxt"][1][2] = 1.3653821219712916
     Gradients["dxt"].shape = (3, 10)
     Gradients["da_prev"][2][3] = -0.04357779106461625
     Gradients["da_prev"].shape = (5, 10)
     Gradients["dWax"][3][1] = -1.5012584841864745
     Gradients["dWax"].shape = (5, 3)
```

```
        Gradients["dWaa"][1][2] = 1.1441951795389382
        Gradients["dWaa"].shape = (5, 5)
        Gradients["dba"][4] = [1.42397243]
        Gradients["dba"].shape = (5, 1)


    def BackwardPassRNN(da, Caches):
        (Caches, x) = Caches
        (a1, InitialState, x1, parameters) = Caches[0]

        n_a, m, T_x = da.shape
        n_x, m = x1.shape

        dx = np.zeros((n_x, m, T_x))
        dWax = np.zeros((n_a, n_x))
        dWaa = np.zeros((n_a, n_a))
        dba = np.zeros((n_a, 1))
        da0 = np.zeros((n_a, m))
        GradLossPrev = np.zeros((n_a, m))
        for t in reversed(range(T_x)):
            Gradients = CellBackwardRNN(da[:,:,t] + GradLossPrev, Caches[t])
            dxt, GradLossPrev, dWaxt, dWaat, dbat = Gradients["dxt"], Gradients["da_prev"], Gradi
            dx[:, :, t] = dxt
            dWax += dWaxt
            dWaa += dWaat
            dba += dbat

        da0 = GradLossPrev
        Gradients = {"dx": dx, "da0": da0, "dWax": dWax, "dWaa": dWaa,"dba": dba}

        return Gradients


    np.random.seed(1)
    InputData = np.random.randn(3,10,4)
    InitialHiddenState = np.random.randn(5,10)
    Wax = np.random.randn(5,3)
    Waa = np.random.randn(5,5)
    Wya = np.random.randn(2,5)
    ba = np.random.randn(5,1)
    by = np.random.randn(2,1)
    Parameters = {"Wax": Wax, "Waa": Waa, "Wya": Wya, "ba": ba, "by": by}
    a, y, Caches = ForwardPassRNN(InputData, InitialHiddenState, Parameters)
    da = np.random.randn(5, 10, 4)
    Gradients = BackwardPassRNN(da, Caches)

    print("Gradients[\"dx\"][1][2] =", Gradients["dx"][1][2])
    print("Gradients[\"dx\"].shape =", Gradients["dx"].shape)
    print("Gradients[\"da0\"][2][3] =", Gradients["da0"][2][3])
    print("Gradients[\"da0\"].shape =", Gradients["da0"].shape)
    print("Gradients[\"dWax\"][3][1] =", Gradients["dWax"][3][1])
    print("Gradients[\"dWax\"].shape =", Gradients["dWax"].shape)
```

```
print("Gradients[\"dWaa\"][1][2] =", Gradients["dWaa"][1][2])
print("Gradients[\"dWaa\"].shape =", Gradients["dWaa"].shape)
print("Gradients[\"dba\"][4] =", Gradients["dba"][4])
print("Gradients[\"dba\"].shape =", Gradients["dba"].shape)
```

```
Gradients["dx"][1][2] = [-2.07101689 -0.59255627  0.02466855  0.01483317]
Gradients["dx"].shape = (3, 10, 4)
Gradients["da0"][2][3] = -0.31494237512664996
Gradients["da0"].shape = (5, 10)
Gradients["dWax"][3][1] = 11.264104496527777
Gradients["dWax"].shape = (5, 3)
Gradients["dWaa"][1][2] = 2.303333126579893
Gradients["dWaa"].shape = (5, 5)
Gradients["dba"][4] = [-0.74747722]
Gradients["dba"].shape = (5, 1)
```