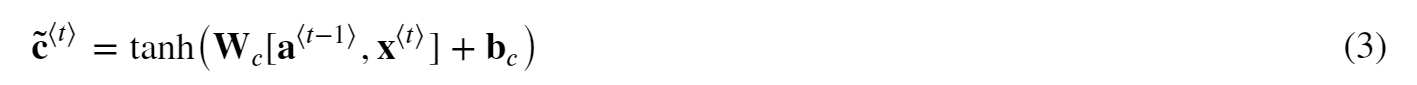
A math equations on a white background

Description automatically generated

A diagram of a circuit

Description automatically generated









A black text on a white background

Description automatically generated

A screenshot of a computer code

Description automatically generated

A math equations and formulas

Description automatically generated with medium confidence

A white sheet with black text

Description automatically generated

### START CODE HERE ###

# Retrieve values from the first cache (t=1) of caches (≈2 lines)

(caches, x) = caches

(a1, a0, x1, parameters) = caches[0]

# Retrieve dimensions from da's and x1's shapes (≈2 lines)

n\_a, m, T\_x = da.shape

n\_x, m = x1.shape

# initialize the gradients with the right sizes (≈6 lines)

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))

da\_prevt = np.zeros((n\_a, m))

# Loop through all the time steps

for t in reversed(range(T\_x)):

# Compute gradients at time step t. Choose wisely the "da\_next" and the "cache" to use in the backward propagation step. (≈1 line)

gradients = rnn\_cell\_backward(da[:,:,t], caches[t])

# Retrieve derivatives from gradients (≈ 1 line)

dxt, da\_prevt, dWaxt, dWaat, dbat = gradients["dxt"], gradients["da\_prev"], gradients["dWax"], gradients["dWaa"], gradients["dba"]

# Increment global derivatives w.r.t parameters by adding their derivative at time-step t (≈4 lines)

dx[:, :, t] = dxt

dWax += dWaxt

dWaa += dWaat

dba += dbat

print(f"Step {t}: dWax = {dWax}, dWaa = {dWaa}, dba = {dba}")

# Set da0 to the gradient of a which has been backpropagated through all time-steps (≈1 line)

da0 = da\_prevt

### END CODE HERE ###

# Store the gradients in a python dictionary

gradients = {"dx": dx, "da0": da0, "dWax": dWax, "dWaa": dWaa,"dba": dba}

return gradients