Write the Python code to implement a single neuron.

# Python program to implement a

# single neuron neural network

# import all necessary libraries

from numpy import exp, array, random, dot, tanh

# Class to create a neural

# network with single neuron

class NeuralNetwork():

    def \_\_init\_\_(self):

        # Using seed to make sure it'll

        # generate same weights in every run

        random.seed(1)

        # 3x1 Weight matrix

        self.weight\_matrix = 2 \* random.random((3, 1)) - 1

    # tanh as activation function

    def tanh(self, x):

        return tanh(x)

    # derivative of tanh function.

    # Needed to calculate the gradients.

    def tanh\_derivative(self, x):

        return 1.0 - tanh(x) \*\* 2

    # forward propagation

    def forward\_propagation(self, inputs):

        return self.tanh(dot(inputs, self.weight\_matrix))

    # training the neural network.

    def train(self, train\_inputs, train\_outputs,

                            num\_train\_iterations):

        # Number of iterations we want to

        # perform for this set of input.

        for iteration in range(num\_train\_iterations):

            output = self.forward\_propagation(train\_inputs)

            # Calculate the error in the output.

            error = train\_outputs - output

            # multiply the error by input and then

            # by gradient of tanh function to calculate

            # the adjustment needs to be made in weights

            adjustment = dot(train\_inputs.T, error \*

                             self.tanh\_derivative(output))

            # Adjust the weight matrix

            self.weight\_matrix += adjustment

# Driver Code

if \_\_name\_\_ == "\_\_main\_\_":

    neural\_network = NeuralNetwork()

    print ('Random weights at the start of training')

    print (neural\_network.weight\_matrix)

    train\_inputs = array([[0, 0, 1], [1, 1, 1], [1, 0, 1], [0, 1, 1]])

    train\_outputs = array([[0, 1, 1, 0]]).T

    neural\_network.train(train\_inputs, train\_outputs, 10000)

    print ('New weights after training')

    print (neural\_network.weight\_matrix)

    # Test the neural network with a new situation.

    print ("Testing network on new examples ->")

    print (neural\_network.forward\_propagation(array([1, 0, 0])))

Write the Python code to implement ReLU.

The code for ReLu is as follows :

def relu(x):

return max(0.0, x)

To test the function, let’s run it on a few inputs.

x = 1.0

print('Applying Relu on (%.1f) gives %.1f' % (x, relu(x)))

x = -10.0

print('Applying Relu on (%.1f) gives %.1f' % (x, relu(x)))

x = 0.0

print('Applying Relu on (%.1f) gives %.1f' % (x, relu(x)))

x = 15.0

print('Applying Relu on (%.1f) gives %.1f' % (x, relu(x)))

x = -20.0

print('Applying Relu on (%.1f) gives %.1f' % (x, relu(x)))

Write the Python code for a dense layer in terms of matrix multiplication.

import numpy as np

#using random numbers generator

np.random.seed(0)

# define our dataset

X = [[1, 2, 3, 2.5],

[2.0, 5.0, -1.0, 2.0],

[-1.5, 2.7, 3.3, -0.8]]

#define dense layer class

class Dense\_layer:

def \_\_init\_\_(self, n\_inputs, n\_neurons): # 2 argments: number of inputs and numbers of neurons

self.weight = 0.10 \* np.random.randn(n\_inputs, n\_neurons) #generate weight randomly and multply with 0.1 to make the numbers smaller (between 0, 1)

self.bias = np.zeros((1, n\_neurons)) # generate bias

# define the forword function, it takes only 1 arrg : input (the dataset)

def forward(self, inputs):

self.output = np.dot(inputs, self.weight) + self.bias

Write the Python code for a dense layer in plain Python (that is, with list comprehensions and functionality built into Python).

import numpy as np

#using random numbers generator

np.random.seed(0)

# define our dataset

X = [[1, 2, 3, 2.5],

[2.0, 5.0, -1.0, 2.0],

[-1.5, 2.7, 3.3, -0.8]]

#define dense layer class

class Dense\_layer:

def \_\_init\_\_(self, n\_inputs, n\_neurons): # 2 argments: number of inputs and numbers of neurons

self.weight = 0.10 \* np.random.randn(n\_inputs, n\_neurons) #generate weight randomly and multply with 0.1 to make the numbers smaller (between 0, 1)

self.bias = np.zeros((1, n\_neurons)) # generate bias

# define the forword function, it takes only 1 arrg : input (the dataset)

def forward(self, inputs):

self.output = np.dot(inputs, self.weight) + self.bias

What is the “hidden size” of a layer?

The size of the hidden layer is normally between the size of the input and output-. It should be should be 2/3 the size of the input layerplus the size of the o/p layer The number of hidden neurons should be less than twice the size of the input layer.

What does the t method do in PyTorch?

Returns a tensor that is a transposed version of input . The given dimensions dim0 and dim1 are swapped. If input is a strided tensor then the resulting out tensor shares its underlying storage with the input tensor, so changing the content of one would change the content of the other.

Why is matrix multiplication written in plain Python very slow?

Our plain Python solution takes 11.77 seconds to run, while using Numpy to perform the multiplications and generate the matrices takes 0.0097 seconds to run. Additionally, if we use the Numpy function power instead, we cut the runtime to 0.00065 seconds. As you can see to calculate 50 of these using python for loops took us 5.66 seconds. Remember that was 1/1000 of the dataset. If we multiply 6 seconds by 1000 we get 6,000 seconds to complete the matrix multiplication in python, which is a little over 4 days. A pretty long time to wait. So, let's see if we can speed that up.

In matmul, why is ac==br?

To make 3,200x as fast as pure python

In Jupyter Notebook, how do you measure the time taken for a single cell to execute?

Measure execution time with Jupyter Notebook: %timeit , %%timeit. In Jupyter Notebook (IPython), you can use the magic commands %timeit and %%timeit to measure the execution time of your code. No need to import timeit module.

What is elementwise arithmetic?

An element-wise operation is an operation between two tensors that operates on corresponding elements within the respective tensors. An element-wise operation operates on corresponding elements between tensors. Two elements are said to be corresponding if the two elements occupy the same position within the tensor.

Write the PyTorch code to test whether every element of a is greater than the corresponding element of b.

This below solution worked for me:

torch.equal(tensorA, tensorB)

From the documentation:

True if two tensors have the same size and elements, False otherwise.

What is a rank-0 tensor? How do you convert it to a plain Python data type?

A tensor with rank 0 is a zero-dimensional array. The element of a zero-dimensional array is a point. This is represented as a Scalar in Math and has magnitude. Eg: s = 48.3. Shape - [].How do you convert a TensorFlow tensor to a list?

“tensor to list” Code Answer's

import tensorflow as tf.

​

a = tf. constant([[1, 2], [3, 4]])

b = tf. add(a, 1)

​

a. numpy()

# array([[1, 2],

# [3, 4]], dtype=int32)

How does elementwise arithmetic help us speed up matmul?

Faster Matrix Multiplications in Numpy

Measure First. The first step is to measure everything. ...

Reduce precision. Ensure your arrays have a dtype of numpy. ...

Use BLAS directly. BLAS is a high-performance matrix library. ...

Use a faster BLAS. ...

Check data order. ...

Factor out common subexpressions. ...

Sparse vectors. ...

SVD compression.

What are the broadcasting rules?

Broadcasting Rules:

If the arrays don't have the same rank then prepend the shape of the lower rank array with 1s until both shapes have the same length.

The two arrays are compatible in a dimension if they have the same size in the dimension or if one of the arrays has size 1 in that dimension.

What is expand\_as? Show an example of how it can be used to match the results of broadcasting.

Tensor. expand\_as. Expand this tensor to the same size as other .