1)write a python code to implement a single neuron.

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

class my\_network():

def \_\_init\_\_(self):

random.seed(1)

# 3x1 Weight matrix

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

defmy\_tanh(self, x):

return tanh(x)

defmy\_tanh\_derivative(self, x):

return 1.0 - tanh(x) \*\* 2

# forward propagation

defmy\_forward\_propagation(self, inputs):

return self.my\_tanh(dot(inputs, self.weight\_matrix))

# training the neural network.

deftrain(self, train\_inputs, train\_outputs,

num\_train\_iterations):

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

output = self.my\_forward\_propagation(train\_inputs)

# Calculate the error in the output.

error = train\_outputs - output

adjustment = dot(train\_inputs.T, error \*self.my\_tanh\_derivative(output))

# Adjust the weight matrix

self.weight\_matrix += adjustment

# Driver Code

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

my\_neural = my\_network()

print ('Random weights when training has started')

print (my\_neural.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

my\_neural.train(train\_inputs, train\_outputs, 10000)

print ('Displaying new weights after training')

print (my\_neural.weight\_matrix)

# Test the neural network with a new situation.

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

print (my\_neural.my\_forward\_propagation(array([1, 0, 0])))

Output : Random weights when training has started

[[-0.16595599]

[ 0.44064899]

[-0.99977125]]

Displaying new weights after training

[[5.39428067]

[0.19482422]

[0.34317086]]

Testing network on new examples

[0.99995873]

2)write python code to implement ReLU.

Ans :def relu(x):

return max(0.0, x)

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)))

Output : Applying Relu on (1.0) gives 1.0

Applying Relu on (-10.0) gives 0.0

Applying Relu on (0.0) gives 0.0

Applying Relu on (15.0) gives 15.0

Applying Relu on (-20.0) gives 0.0

3)write a python code for a dense layer in terms of matrix multiplication.

ans : 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 arguments: number of inputs and numbers of neurons

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

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

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

def forward(self, inputs):

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

4)write a python code for a dense layer in plain python .

Ans :tf.keras.layers.Dense(

units,

activation=None,

use\_bias=True,

kernel\_initializer="glorot\_uniform",

bias\_initializer="zeros",

kernel\_regularizer=None,

bias\_regularizer=None,

activity\_regularizer=None,

kernel\_constraint=None,

bias\_constraint=None,

\*\*kwargs

)

>> # Create a `Sequential` model and add a Dense layer as the first layer.

>>> model = tf.keras.models.Sequential()

>>> model.add(tf.keras.Input(shape=(16,)))

>>> model.add(tf.keras.layers.Dense(32, activation='relu'))

>>> # Now the model will take as input arrays of shape (None, 16)

>>> # and output arrays of shape (None, 32).

>>> # Note that after the first layer, you don't need to specify

>>> # the size of the input anymore:

>>> model.add(tf.keras.layers.Dense(32))

>>> model.output\_shape

(None, 32)

5)what is hidden size of layer ?

Ans : The number of hidden neurons should be between the size of the input layer and the size of the output layer. The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer. The number of hidden neurons should be less than twice the size of the input layer.

6)what does the t method do in pytorch.?

Ans : Returns a view of this tensor with its dimensions reversed.

If n is the number of dimensions in x, x.T is equivalent to x.permute(n-1, n-2, ..., 0).

The use of Tensor.T() on tensors of dimension other than 2 to reverse their shape is deprecated and it will throw an error in a future release. Consider mT to transpose batches of matrices or x.permute(\*torch.arange(x.ndim - 1, -1, -1)) to reverse the dimensions of a tensor

7)why is matrix multiplication written in plain python very slow.

Ans : Python is the most popular programming language for data-science-related tasks. And many of those tasks are computation-heavy ones. One of them is matrix multiplication – a basic linear algebra operation, used by almost all ML algorithms, for example, to compute activations of layers of a neural network based on activations.Python is a simple language with great tooling, enormous community, and a lot of packages available. But it is too slow for running computational-heavy algorithms, crucial for any data science task. However, there is a simple bypass for this limit. We can use libraries like NumPy, which provides a nice Python API for running code written in faster languages.

We can have the performance of those languages, without their complexity. And we can still use an ecosystem of Python packages, tooling, and community. In the case of using Python for data science, we not only shouldn’t reinvent the wheel – we mustn’t do it.

8)In matmul why is ac=br ?

Ans : excel we multiplied rows and columns element wise and then added the intermediate products to get the final result. If there is mismatch in dimensions, and one vector is longer than the other, we can no longer carry out element wise multiplication.The dimensions of the resulting matrix will always be ar,bc. That is the number of rows comes from A and the number of columns comes from B.

Here’s another disclaimer, while implementing Matrix Multiplication in Excel, not only have we understood it, but we have also replicated Jeremy’s first method that used 3 FOR loops. Here’s how it looks like in code:

def matmul1(a,b):

ar,ac = a.shape

br,bc = b.shape

assert ac==br

c = torch.zeros(ar, bc)

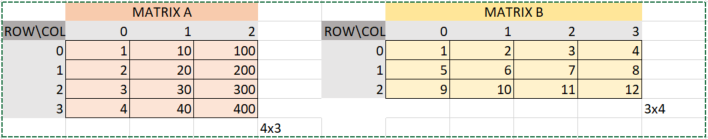
for i in range(ar):

for j in range(bc):

for k in range(ac): #or br

c[i,j] += a[i,k]\*b[k,j]

return c



9) In jupyter notebook how do you measure the time taken for a single cell to execute ?

Ans : 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 the timeit module.

%timeit

For %timeit, specify the target code after %timeit with a space.

By default, number and repeat in timeit.timeit() are set automatically. It can also be specified with -n and -r options.

The mean and standard deviation are calculated.

%%timeit

You can use the magic command %%timeit to measure the execution time of the cell.

As an example, try executing the same process using NumPy. As with %timeit, -n and -r are optional.

Note that %%timeit measures the execution time of the entire cell.

10)what is element wise arithmetic ?

Ans: 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.

11)write a pytorch code to test whether every element of a greater than the corresponding element of b.

12)what is a rank 0 tensor? How do you convert it to a plain python data type.

Ans :

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 - [].you can cast the datatype of a tensor to a new datatype by using the tf. cast function.

We can access the data type of a tensor using the ". dtype" attribute of the tensor. It returns the data type of the tensor.

13)how does element wise Arithmetic help us speed up matmul ?

Ans : elementwise (not comparable) (mathematics) Obtained by operating on one element (of a matrix etc) at a time.

The element wise multiplication operator (#) computes a new matrix with elements that are the products of the corresponding elements of matrix1 and matrix2. In addition to multiplying matrices that have the same dimensions, you can use the element wise multiplication operator to multiply a matrix and a scalar.

14)what are the broadcasting rules ?

Ans : When operating on two tensors, PyTorch compares their shapes elementwise. It starts with the trailing dimensions and works its way backward, adding 1 when it meets empty dimensions. Two dimensions are compatible when one of the following is true:

They are equal.

One of them is 1, in which case that dimension is broadcast to make it the same as the other.

Arrays do not need to have the same number of dimensions. For example, if you have a 256×256×3 array of RGB values, and you want to scale each color in the image by a different value, you can multiply the image by a one-dimensional array with three values. Lining up the sizes of the trailing axes of these arrays according to the broadcast rules, shows that they are compatible:

Image (3d tensor): 256 x 256 x 3

Scale (1d tensor): (1) (1) 3

Result (3d tensor): 256 x 256 x 3

However, a 2D tensor of size 256×256 isn't compatible with our image:

Image (3d tensor): 256 x 256 x 3

Scale (2d tensor): (1) 256 x 256

Error

In our earlier examples we had with a 3×3 matrix and a vector of size 3, broadcasting was done on the rows:

Matrix (2d tensor): 3 x 3

Vector (1d tensor): (1) 3

Result (2d tensor): 3 x 3

As an exercise, try to determine what dimensions to add (and where) when you need to normalize a batch of images of size 64 x 3 x 256 x 256 with vectors of three elements (one for the mean and one for the standard deviation).

Another useful way of simplifying tensor manipulations is the use of Einstein summations convention.

15)what is expand\_as ?show an example of how it can be used to match the result of broadcasting.

Ans : Here the elements of c are expanded to make three rows that match, making the operation possible. Again, PyTorch doesn't actually create three copies of c in memory. This is done by the expand\_as method behind the scenes:

c.expand\_as(m)

tensor([[10., 20., 30.],

[10., 20., 30.],

[10., 20., 30.]])

If we look at the corresponding tensor, we can ask for its storage property (which shows the actual contents of the memory used for the tensor) to check there is no useless data stored:

t = c.expand\_as(m)

t.storage()

10.0

20.0

30.0

[torch.FloatStorage of size 3]

Even though the tensor officially has nine elements, only three scalars are stored in memory. This is possible thanks to the clever trick of giving that dimension a stride of 0 (which means that when PyTorch looks for the next row by adding the stride, it doesn't move):

t.stride(), t.shape

((0, 1), torch.Size([3, 3]))