1. Write the Python code to implement a single neuron.

ANS:

# 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])))

1. Write the Python code to implement ReLU.

ANS:

We can represent it mathematically as follows:

Relu FunctionRelu Function

The pseudo code for Relu is as follows:

If input > 0:

return input

else:

return 0

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

ANS:

Dense layer is the regular deeply connected neural network layer. It is most common and frequently used layer. Dense layer does the below operation on the input and return the output.

output = activation(dot(input, kernel) + bias)

where,

* input represent the input data
* kernel represent the weight data
* dot represent numpy dot product of all input and its corresponding weights
* bias represent a biased value used in machine learning to optimize the model
* activation represent the activation function.

Let us consider sample input and weights as below and try to find the result −

* input as 2 x 2 matrix [ [1, 2], [3, 4] ]
* kernel as 2 x 2 matrix [ [0.5, 0.75], [0.25, 0.5] ]
* bias value as 0
* activation as linear. As we learned earlier, linear activation does nothing.

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

ANS:

List comprehension in Python is an easy and compact syntax for creating a list from a string or another list. It is a very concise way to create a new list by performing an operation on each item in the existing list. List comprehension is considerably faster than processing a list using the for loop.

List comprehensions and dictionary comprehensions are a powerful substitute to for-loops and also lambda functions. Not only do list and dictionary comprehensions make code more concise and easier to read, they are also faster than traditional for-loops.

1. What is the “hidden size” of a layer?

ANS:

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.

Hidden layer(s) are the secret sauce of your network. They allow you to model complex data thanks to their nodes/neurons. They are “hidden” because the true values of their nodes are unknown in the training dataset. In fact, we only know the input and output.

1. What does the t method do in PyTorch?

ANS:

T : 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.

1. Why is matrix multiplication written in plain Python very slow?

ANS:

2

I try to find an explanation why my matrix multiplication with Numba is much slower than using NumPy's dot function. Although I am using the most basic code for writing a matrix multiplication function with Numba, I don't think that the significantly slower performance is due to the algorithm. For simplicity, I consider two k x k square matrices, A and B. My code reads

1. In matmul, why is ac==br?

ANS:

From Rule-1, we know that ac==br, so it really doesn’t matter whether we put range(ac)or range(br)in the third loop. Essentially, this is where the individual elements get multiplied together and added. From our previous Excel example, for C[0][0]this is where the step C[0][0] = 1\*1 + 10\*5 + 100\*9 = 1 + 50 + 900 = 951 would happen and finally we will move on to the next box. The number of items that get multiplied and added together is equal to ac or br.

That’s really it! That’s method-1 for you! Here’s a song to remember Matrix Multiplication. (I learned about this first from fast.ai, the original author is unknown)

1. 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 without needing to import the timeit module.

1. What is elementwise arithmetic?

ANS:

Each pair of elements in corresponding locations are added together to produce a new tensor of the same shape. So, addition is an element-wise operation, and in fact, all the arithmetic operations, add, subtract, multiply, and divide are element-wise operations.

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

ANS:

To compare two tensors element-wise in PyTorch, we use the torch. eq() method. It compares the corresponding elements and returns "True" if the two elements are same, else it returns "False"

The torch package contains data structures for multi-dimensional tensors and defines mathematical operations over these tensors. Additionally, it provides many utilities for efficient serialization of Tensors and arbitrary types, and other useful utilities.

1. 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 - []

Rank 0 is the highest ranking result on the Google Search Engine Results Pages (SERPs). It is found above the fold or above all organic search results but is just below any paid adverts for that query. It is an expanded text or video result that generally answers a query or question.

To convert between types, you simply use the type name as a function. There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value.

1. How does elementwise arithmetic help us speed up matmul?

ANS:

Numpy can multiply two 1024x1024 matrices on a 4-core Intel CPU in ~8ms. This is incredibly fast, considering this boils down to 18 FLOPs / core / cycle, with a cycle taking a third of a nanosecond. Numpy does this using a highly optimized BLAS implementation. BLAS is short for Basic Linear Algebra Subprograms

In linear algebra, the Strassen algorithm, named after Volker Strassen, is an algorithm for matrix multiplication. It is faster than the standard matrix multiplication algorithm for large matrices, with a better asymptotic complexity, although the naive algorithm is often better for smaller matrices.

1. What are the broadcasting rules?

ANS:

* All the input arrays have the same shape.
* Arrays have the same number of dimensions, and the length of each dimension is either a common length or 1.
* Array with the fewer dimension can be appended with '1' in its shape.

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

ANS:

These include radio and television broadcasting technology, satellite broadcasting technology, digital cable television and satellite television technology, digital terrestrial television, and satellite TV technologies.