**Deep Learning EX1**

1. Take the FizzBuzz example and make it work.

(solution code in code file “Fizzbuzz.ipynb”)

1. Add to the code a function that analyzes the results.
   1. Print the accuracy of the classifier. (attach in report)
   2. Generate a confusion matrix.

Chart

Description automatically generated

1. Run it once and put the results in a word file. (attach in report)
2. Rerun the algorithm 10 times and see if there are differences in the accuracy.

Generate a graph showing the differences in the results.

Chart, line chart

Description automatically generated

1. Change the representation from binary to prime based. That means each number is coded by how many time each prime appears in the product. For large primes you can put them all in one bucket.

Example: 24 is coded as 2^3\* 3^1 -🡪 [3,1,0,0,0,0…]

Which of the algorithms (first and second) will work better? Run them and write the results and compare them.

(solution code in code file “Fizzbuzz\_primary.ipynb”)

We can see from the graph that primary representation is better and more accurate.

Chart, line chart

Description automatically generated

1. Find the smallest network for the prime representation that will yield perfect results.

The smallest network is 2 neurons in hidden layer, one responsible of numbers that divides by 3 and the second is responsible of numbers divides by 5.

1. Write a function that is given as input the prime representation and it classifies the number. It does not have to be a neural net classifier but works on the representation without any additional knowledge or data. (attached also in the code)

def primary\_classifier\_without\_neural\_net(primry\_encoded\_numer):

if primry\_encoded\_numer[1] > 0 and primry\_encoded\_numer[2] > 0 : return "fizzbuzz"

elif primry\_encoded\_numer[2] > 0 : return "buzz"

elif primry\_encoded\_numer[1] > 0 : return "fizz"

else: return primry\_encoded\_numer

1. Take the smallest classifier you built and instead of training it, hardcode the weights into it and show that it yields perfect results like in 7 & 8.

(solution code in code file “Fizzbuzz\_primary\_hardcoded\_weights.ipynb”)

1. Implement in Python a Neuron class. It should not be based on a well-known system like TF. Just pure code. Here is the main that uses it:

n = Neuron (10, 0.1) # init assumption is that the activation function is Relu

x = np.random.normal(0,0.1,10)

res = n.forward(x) #given x compute the result and maintain results in the object

db,dw,dx = n.backward(0.3) # Given the error compute the derivatives

print(res,db,dw,dx)

x = np.ones(10)

w = np.array([1,2,3,4,5,6,7,8,9,10])

n.SetWb(w,3) # change w and b

res = n.forward(x)

db,dw,dx = n.backward(0.3)

print(res,db,dw,dx)

(solution code in code file “Neuron.ipynb”)