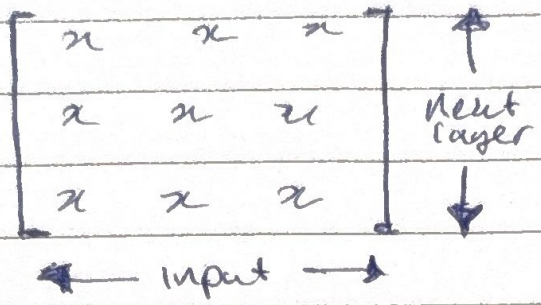
Bot Academy - NN from Stratch

Weights Structure



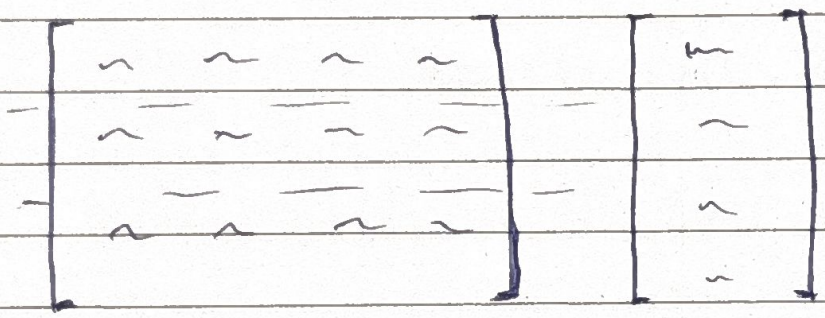
rows = next layer
cols = current layer

np.random.uniform(x, y, (x, y))

For bias = A column. rows = next layer
col = current

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

logic = the input layer is store as a column



recall that MatMu
Multi goes
LHS rows by
RHS cols

therefore the weights are initalized as the input or current neurons accross the cols

Another trick to remember is that when reading Matrix Mutiplication you read from right to left

Start with image & convert to colum

the next matrix thing needs to match this
→ input weigts accoss the top

init weights layers as random & small
init Bias values as zeros
(for whole network layers)

Mnist data: from data import get_mnist

images, labels = get_mnist()
60k, 10

↳ (60,000 by 784) Pixels/features as cols

labels are not encoded: (60k, 10)

training params: learn rate, epochs ← loops

- for epoch in range(epochs):

  then loop through each sample/image

  - For img, L in zip(images, labes):

    recall the get-MNIST has sample as row/vector form but we are going to do matrix vector multiplication. Hence we need the input img to be col wise (and the labels)

    - img.shape += (1,)
    - L.shape += (1,)

    } reshape
    img.shape = 784
    img.shape = 784, 1
    Adds dimension

    now do ff using inputs & weights using wx + b

    - bias + weight @ input
      not right to left
    - n = 1/(1 + np.exp(-h_Pre))
      normalize the values between 0-1 to avoid huge values (Sigmoid activation)

After feedforward comes the cost functn

∂ Check if output is correct or not (counter)
(not important for training but good code practise for Transparency / Information

BackProp

output ↓ label ↓ very important to use easy/correct cost function

1 {
delta_o = O - L
w_h_o = -LR * delta_o @ np.trans(h)
b_h_o = -LR * delta_o
}

repeat {
1. Read right to left
2. Update the weights
4. Start w/ hidden layer neurons as vector
5. Negation learn rate @ Minimise cost
6. += to alter init original weights
}

Notes: h = hidden layer neurons : b + w @ ims
the Dimension output = (20, 1) , as column

delta_o is also a column (error)

the transpose turns h for col to row
why?
(10, 1) @ (1, 20) = (10, 20) = same shape as weight
Errors neurons

the RHS is manipulate to get the correct output
= Gradient Matrix

MM retains shape 1 LHS 2 RHS

Second Backprop layer:

delta_h is the only thing different intem of the process

(1) need the deriv'd the sigmoid FIRST
(h * (1-h)
Also vary per activation used

(2) get weights & transpose

(3) Mm between weights & delta_0

(4) (3) * (1)
previous SigDeriv

repeat the same steps as before

w_1_ht= -LR * delta_h @ np.transpose(img)

Code for running code