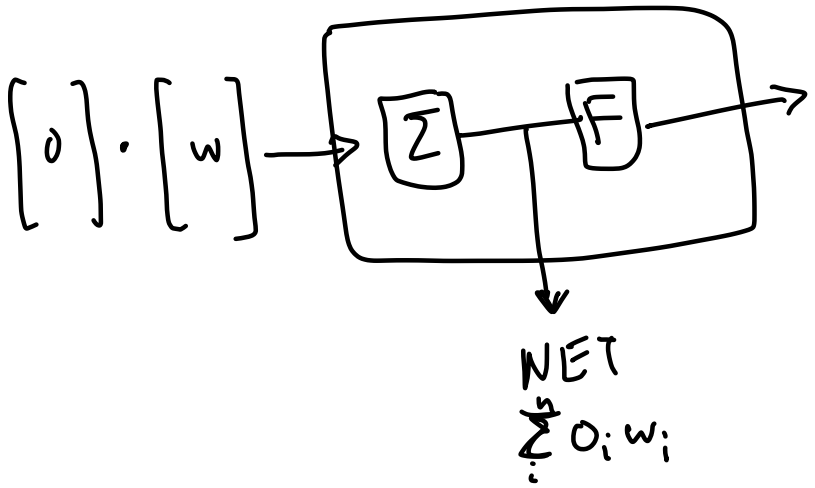
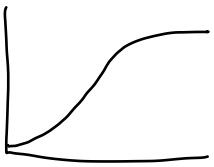


$$\frac{1}{1+e^{-NET}}$$

$out = 1/(1+e^{-NET})$



out = f(net)
(activationfunction)

f(INPUT) = OUTPUT

$$T = \begin{bmatrix} & & 784 \\ & & \\ & & \end{bmatrix} 10$$

$$y = wx$$

$$Net = xw$$

$$w = x^{-1} NET$$

$$\frac{1}{1+e^{-net}}$$

$$\left(\sum_{i=1}^{n} o_i w_i\right)$$

- **vi.- Training the network** A network will learn by iteratively adapting the values of $w_{i,j}$. Each input is associated to an output. These are called *training pairs*. Follow these steps (i.e. general structure of your algorithm)
  - Select the next training pair (INPUT, OUTPUT) and apply the INPUT to the network
  - Calculate the output using the network
  - Calculate the error between the network's output and the desired output
  - Adjust weights in a way that minimizes the error
  - Repeat steps above for each training pair

Calculations are performed by layers, that is all calculations are performed in the hidden layer before any calculation is performed in the output layer. The same applies when several hidden layers are present. The first two steps above are called *forward pass* and the second two the *reverse pass*.

Forward pass: Notice that the weights in between layers of neurons can be represented by the matrix W and that if X is the input vector, then $NET = XW$ and the output vector is $O = F(NET)$. The output vector will be the input vector for the next layer.

Reverse pass. Adjusting the weights of the output layer: The ERROR signal is produced by comparing the OUTPUT with the TARGET value. We will consider the training process for a single weight from neuron p in the hidden layer j to neuron q in the output layer k (See Fig. 5). First we calculate the ERROR ($= TARGET - OUT$) for that output neuron k. This is multiplied by the derivative of the activation function for neuron k .

$$\delta_{q,k} = OUT_{q,k} * (1 - OUT_{q,k})(ERROR)$$

This value is further multiplied by the $OUT_{p,j}$ of the neuron p in hidden layer j and a training rate coefficient $\eta \in [0.01, 0.1]$. The result is added to the weight connecting the two neurons.

"η" "α"

$$\Delta w_{pq,k} = \eta * \delta_{q,k} * OUT_{p,j};$$

Therefore the configuration of the output layer will change for the next training set as follows

$$w_{pq,k}(n+1) = w_{pq,k}(n) + \Delta w_{pq,k}$$

Reverse pass. Adjusting the weights of the hidden layers: Backpropagation will train the hidden layers by "propagating" the error back and adjusting the weights on its way. The algorithm uses the same two equations as above for $w_{pq,k}$ and $w_{pq,k}(n+1)$ however the value of δ needs to be computed differently.

The process is shown in Figure 6. Once δ has been calculated for the output layer it is used to compute the value of δ for each neuron.

$$\delta_{p,j} = OUT_{p,j}(1 - OUT_{p,j})(\sum_k \delta_{q,k} w_{pq,k})$$ ← delta for adjusting weights of hidden layers

It is important to realize that all the weights associated with each layer must be adjusted moving back from the output to the first layer.

In matrix notation this is written as follows: If D_k is the set of the deltas at the output layer, W_k the set of weights at the output layer and D_j the vector of deltas for the hidden layer in the previous Fig. 3

$$D_j = D_k W_k^t \otimes [O_j \otimes (I - O_j)]$$

where $\otimes$ is defined to indicate the component-by-component multiplication of the two vectors. O_j is the output vector of layer j and I is the vector with all components equal to 1. Show that this last formula is correct.

$$\delta_2 = \delta_3 \cdot W_3^T \cdot (O_2 \cdot (I - O_2))$$

Calculating Error between final output & target & adjusting weight of last layer

Error = Target − Output (last layer)

δ = Output (1 − Output) (Error)

ΔW = δ × η × Output (last layer − 1)

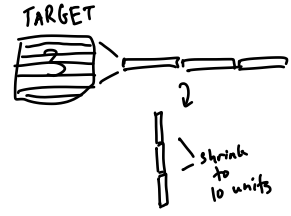
new W_n = $W_n + \Delta W$

$W = [\ 784\times100 \quad 100\times100 \quad 100\times10\ ]$

train0 matrix

5923 × 784 → 300 × 784

final out put



TARGET = mean (test #)

6. Training the network

- Select the next training pair (INPUT, OUTPUT) and apply the INPUT to the network
- Calculate the output using the network

} forward pass

- Calculate the error between the network's output and the desired output
- Adjust weights in a way that minimizes the error
- Repeat steps above for each training pair

} back pass

Required

Input: All train # matrices. *Dump training images & have it learn

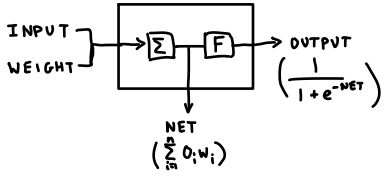
One test matrix.

Output: give recognized number,

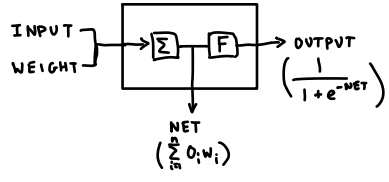
given  x100 to train. given  . Output # = 5.

What's a neuron?

1 neuron



2 neuron

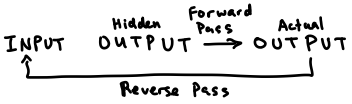


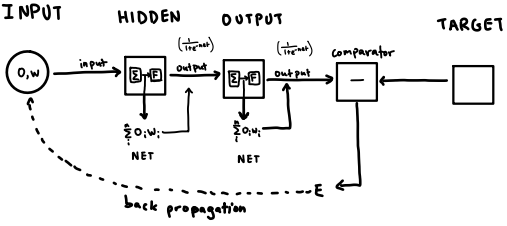
784 columns because a normalized 28x28 pixel image has 784 total pixels

The values in each matrix range from 0-256 since the image has been converted to an 8-bit greyscale image. 8-bit data has 256 values max.
(Ex: 0 = white ; 256 = black)

Backpropagation

Back Propagation increases or decreases weight to decrease error





$$E_n = \frac{1}{2}(target_n - output)^2$$

$$E_{total} = \sum E = E_1 + \cdots E_n$$

Rate of change of error

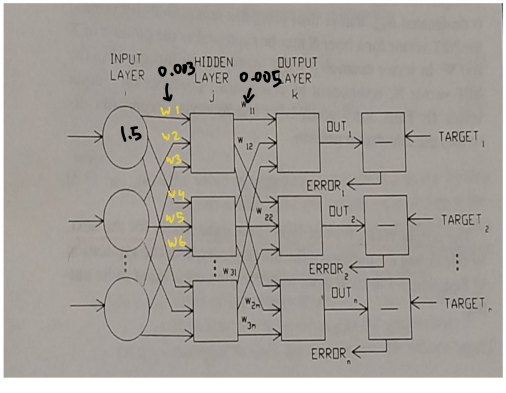
$$\frac{\delta E_{total}}{\delta W_n} = \frac{\delta E_{total}}{\delta output\,1} \cdot \frac{\delta output\,1}{\delta net\,1} \cdot \frac{\delta net\,1}{\delta W_n}$$

$$\frac{\delta E_{total}}{\delta output\,1} = -(target\,1 - Output\,1)$$

$$\frac{\delta output\,1}{\delta net\,1} = Output\,1 \cdot (1 - Output\,1)$$

$$\downarrow \quad \frac{1}{1+e^{net\,1}}$$

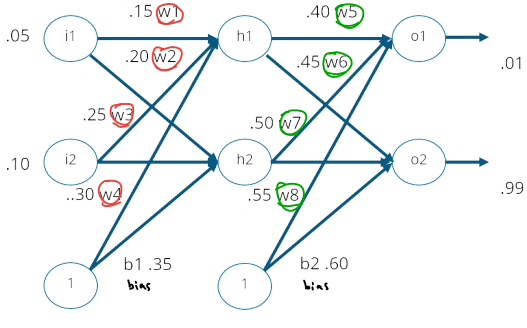
$$\frac{\delta net\,1}{\delta W_n} = W_n \cdot (Output\,1 + W_{net\,1})_{Hidden}$$



Stochastic Gradient Algorithm

$$W_{n+1} = W_n - \eta \frac{\delta E_{total}}{\delta W_n}$$

η: training rate coeff $\in [0.01, 0.1]$



W = Each column vector of weighted matrix represents connection between 2 layers

input W output

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \quad \begin{bmatrix} w_1 & w_5 \\ w_2 & w_6 \\ w_3 & w_7 \\ w_4 & w_8 \end{bmatrix} \quad \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

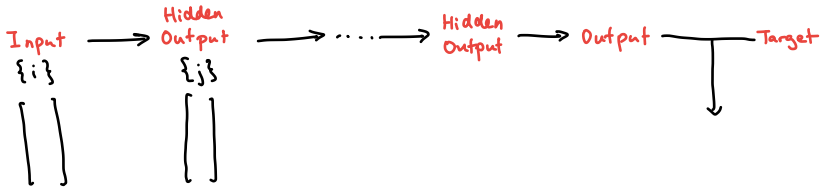
$a = 1w_1 + 2w_2 + 3w_3 + 4w_4$

$b = 2w_2 \cdots$

W = i x j values for 1 column

Output = $\sum$ input $\cdot$ W(:,1)

Code



Output: An minimum error showing the weights are trained to the data
[MIN_ERROR]

A percentage of how confident the answer is
[HOWSURE]

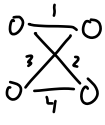
Ex: 86.7% sure this matrix of pixels is a "3"

Input: WEIGHT matrix
TARGET matrix
train # matrix

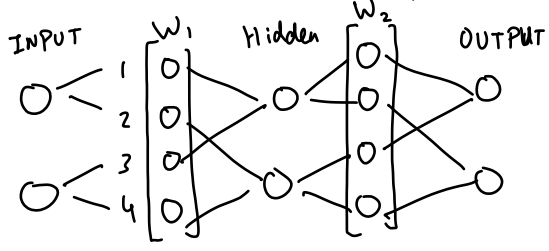
Appendix A: Precondition for faster convergence
Weight values rand $(-2.4/\text{inputs}, +2.4/\text{inputs})$
# inputs = 784 pixels
WEIGHT = rand (- 0.003, +0.003)

2 columns
784 rows

WEIGHT
$((\text{inputrow})^2, \#\text{layers})$

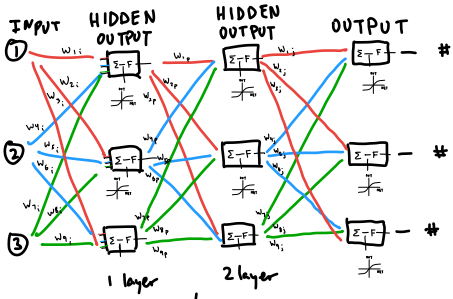


The issue is getting the input value to use the specific weight values saved for it



Is the output 1 value or a vector?

Hidden Output = vector
OUTPUT = 1 value



Hidden Output = vector

OUTPUT = 1 value

train 0
train 1
train 2
⋮
train 9

First step: normalize all input training matrice

6131 x 784

↓

[ ] normalized size

300 x 784 ←

you want to train

Let = 300 images so

Each train# set cut to 300 x 784 matrix

3-Layer NN (2 Hid. Layers)

Input = cell (1,3) For : train3 test3 1010×784

Weight = cell (1,3)

Weight{i} = rand([1010, 784]) between (-0.5, 0.5)

Output = cell (1,3)

Visualization : cell (1,3) = [1010×784 ; 1010×784 ; 1010×784]

Code

```
for 1: 1010
   O = Input{i}
   Output = cell(1,3)
   for k = 1:3
      Net = O * Weight{j}
      O = 1 / (1 + e^-Net)
      Out{m} = O{i}
   end
```

Forward Pass

input = [ ] 784

Weight : [784×784 ; ;]

3 layers each 784×784

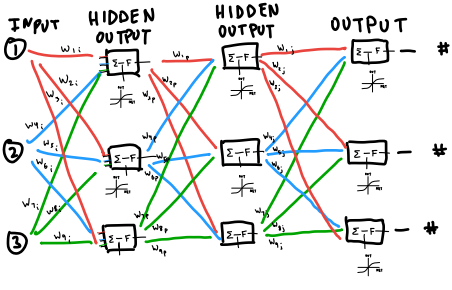
[weight * input] gives $\sum_i O_i w_i$ for O_i → net

[⋮] 784 ⇐ This is an array of net values for first layer

F(net) = Out for every value of first layer

Rinse & repeat for other layers

out = new input [⋮] 784



Answer Confidence

Check if output = target & how many are correct vs wrong. # correct

784 outputs

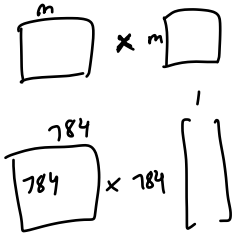
760 right
24 wrong

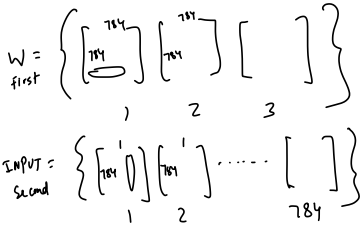
760 / 784

Create check if output is close enough to target & if it is correct, use correct / total to get answer confidence

Troubleshooting

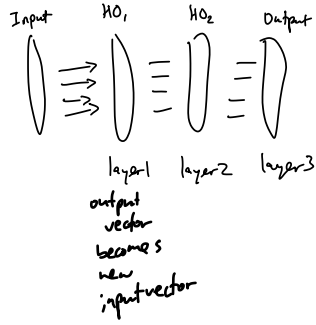
Make weight matrix same # columns as input matrix # rows



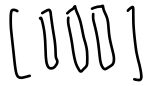


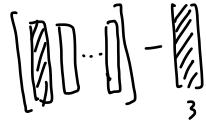

```
for 1:784
  0 = input first column
    for p=1:3
      for 1:784
        weight = Weight cell 1 first row
        Net = Wc1 1stRow * 0
        0 = F(Net)
end
```

```
hiddenoutput = cell(1,layers);
for L = 1 : layers
    weight = WEIGHT{L};
    for i = 1:784
        o = INPUT{i};
        for j = 1:784
            Net = weight(j,:) * o;
            o(i) = 1/(1+e^-NET)
        end
    end
    hiddenoutput{L} = o;
    OUTPUT = hiddenoutput{layers};
    for ii = 1:784
        ERROR = double(TARGET{ii}) - double(OUTPUT);
        δ = OUTPUT * (1 - OUTPUT) * ERROR;
        ΔW = η * δ * OUTPUT;
        WEIGHT{Layers} = WEIGHT{Layers} + ΔW;
        for w2 = (layers-1) : -1 : 1
            OUTPUT = hiddenoutput{w2}
            δ = OUTPUT * (1 - demoutput{w2+1}) * OUTPUT
```



Gives hiddenoutput cell





Weights { 784×100 ; 100×100 ; 100×10 }

o { 1×784 columns, 1×784, ... ; 1×784 }

First iteration

o = input{800} = 1×784

W = weights{1} = 784×100 100 10

NET = 1×100

o = F(NET) = 1×100

W = weight{2} = 100×100

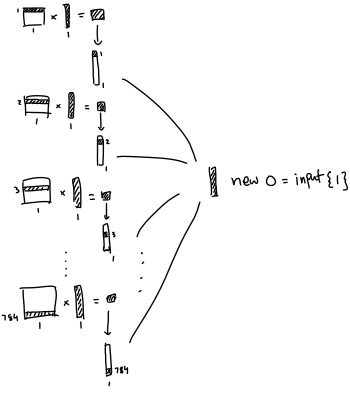
NET = 1×100

o = F(NET) = 1×100

W = weight{3} = 100×10

NET = 1×10

o = 1×10



TRAIN SET : [300x784, 300x784, ... , 300x784]

0 , 1 , - - - - 9


```
for i = 1: length(INPUT)_784
    O = INPUT(i)
  NET = O * W1
  output = 1./(1+exp(-NET)
    O = output
      for j = 1: length(W1)_100
```

```
function output = actfunc ( NET)
output = 1./(1+exp(NET))
end
```

For each train sample put in,
it gives output, compares to target
& back propagates & changes weights

Then it moves onto next sample image.

* What number doesn't matter.
As long as input & target are for same image, it works.

- We want the total cost of network not just for 1 #.
- Test vectors run through network & weights aren't changed.
Those vectors become our TARGET DATASET.