Foundations of NN

- Perceptrons (single neuron)
  - primarily for object recognition (visual contex)
- Gradient Learning
- non-threshold activation functions
  - sigmoid
  - ReLu
  - tah
  - Softmax
  - SBAF

ANN mimics human brains (only partially) for pattern recognition jobs – basic functions for living animals. – ANN's don't have the ability to recall an answer to a search problem at a later point.

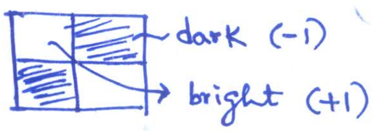
Building blocks:

(A) Perceptron → single neuron model

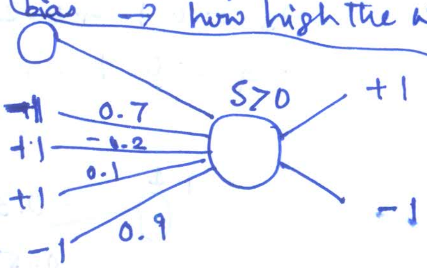
bias ~ sort of a barrier to overcome if the threshold value has to reach 0 for activation.

threshold activation unit (neurons fired at/beyond threshold point)

Ex: Image Recognition Problem



target label: bright image (+1)



bias → how high the weighted sum needs to be for the neuron to be active

Goal: to classify image as bright or dark.

Steps:

1. $S = \sum w x = w^T x$; weighted combination of inputs
2. provide S to the activation function: $O = f(S)$
3. Compute Error: $E = t - O$
4. Calculate $\Delta w_i = \alpha E_i x_i$ ($\alpha \to$ learning rate)
5. update weight, $w_i = w_i + \Delta w_i$

Iterate till Error is minimized

$\underbrace{\hspace{6cm}}_{\text{Perceptron Learning Rule}}$

Iteration 1

$$\begin{bmatrix} 1 & -1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} -0.5 \\ 0.7 \\ -0.2 \\ 0.1 \\ 0.9 \end{bmatrix} \Longrightarrow W^T X \quad ; \quad \begin{array}{l} w \rightarrow \text{weight vector} \\ x \rightarrow \text{input vector} \end{array}$$

1. $S = -0.5 - 0.7 - 0.2 + 0.1 - 0.9 = -2.2 < 0$

2. $O = -1, \quad t = +1,$

3. $E = t - 0 = 2$

4. $\Delta w_i = \alpha E x_i$; set $\alpha = 0.1$

$\Delta w_0 = 0.1 \times 2 \times 1 = 0.2$ → for bias element

$\Delta w_1 = 0.1 \times 2 \times -1 = -0.2$

$\Delta w_2 = 0.1 \times 2 \times 1 = 0.2$

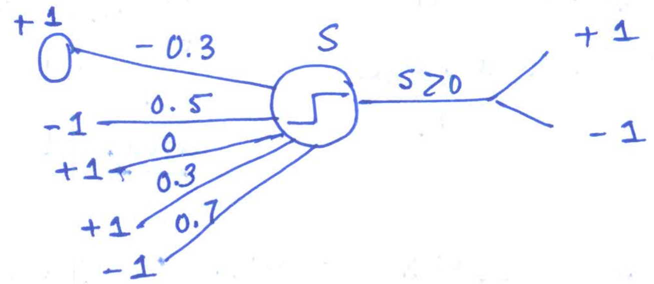
$\Delta w_3 = 0.1 \times 2 \times 1 = 0.2$

$\Delta w_4 = 0.1 \times 2 \times -1 = -0.2$

5. $w_i = w_i + \Delta w_i$

$$\left.\begin{array}{l} w_0 = -0.5 + 0.2 = -0.3 \\ w_1 = 0.7 + 0.2 = 0.5 \\ w_2 = -0.2 + 0.2 = 0 \\ w_3 = 0.1 + 0.2 = 0.3 \\ w_4 = 0.9 - 0.2 = 0.7 \end{array}\right\} \text{new weight vector}$$

Iteration 2



1. $S = 1 \times (-0.3) - 1 * (0.5) + 1\,(0) + 1 * (0.3) - 1 * (0.7)$
$= -1.2 < 0$

2. $O = -1, \quad t = +1$

3. $E = t - 0 = 2$

4. $\Delta w_i = \alpha E x_i$

$\Delta w_0 = 0.1 \times 2 \times 1 = 0.2$

$\Delta w_1 \rightarrow \Delta w_4$ (remain the same)

updated weight vector

$$\begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} -0.3 + 0.2 \\ 0.5 + 0.2 \\ 0 + 0.2 \\ 0.3 + 0.2 \\ 0.7 + 0.2 \end{bmatrix} = \begin{bmatrix} -0.1 \\ 0.7 \\ 0.2 \\ 0.5 \\ 0.9 \end{bmatrix}$$

Continue iterating....

3rd iteration

$$\begin{bmatrix} 0.1 \\ 0.1 \\ 0.4 \\ 0.7 \\ 0.3 \end{bmatrix} \longrightarrow$$

4th iteration

$$[1 \; -1 \; +1 \; +1 \; -1] \begin{bmatrix} 0.1 \\ 0.1 \\ 0.4 \\ 0.7 \\ 0.3 \end{bmatrix}$$

1. $S = 0.8 > 0$
2. $O = +1$
3. $t = +1$
4. $E = t - O = 0$

STOP

Converged weights

Ⓑ Building Block 2 → Gradient Learning

How machines learn? (ANN) → Calculus based tools → comes down to finding minimum of some function

Initially, ANN ~~perform~~ performs horribly → random initialization of bias & weight vectors ⇒ combination of neurons to another neuron — bad computing! ⇒ there's a cost ⇒ define a loss function!!

Cost of a single training example = difference b/w what we want (target) & output produced by activation (trash)

What's the cost of this difference? Squared difference.

Consider the average cost → determines how good/bad the network is!

Cost function: I/P : thousands of weights/biases
O/P : 1 number (cost)
Parameters: Many training examples.

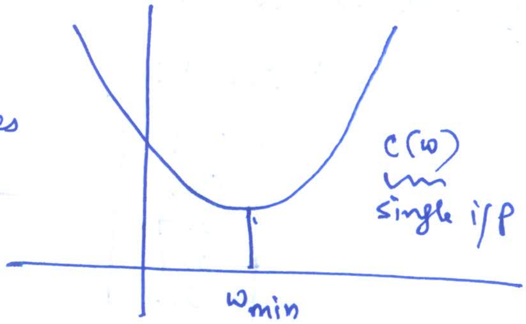
change those weights/biases so that the performance improved!

Define the Cost function: $C(w_1, \ldots w_n)$

Imagine: Single input single o/p function

— how do we find a number that minimizes the o/p?

$\frac{dc}{dw} = 0 \Rightarrow$ easy for a simple function



but very complicated for a function w/ thousands of i/p's

Ball downhill problem $\Rightarrow \nabla C(x,y)$

weights & biases

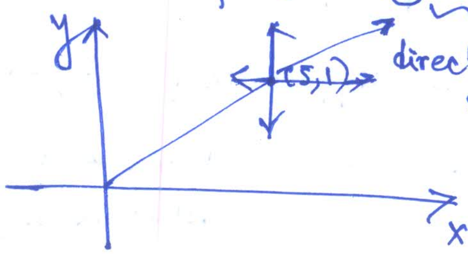
$$\vec{w} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix} \qquad \underset{\substack{\downarrow \\ \text{direction of steepest decent}}}{-\nabla C(\vec{w})} = \underbrace{\begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix}}_{\text{Column vector}} \longrightarrow \text{Minimize Cost}$$

Local (valley) minima

What does this tell us?

relative magnitude of the components of $\nabla C(\vec{w})$ $\Longrightarrow$ how to make changes in the i/p matrix (since a huge difference b/w i/p & o/p matrix will make the cost look bad & the task of classification won't benefit.

Imagine a function: $f(x,y) = \frac{5}{2}x^2 + \frac{1}{2}y^2$; $\nabla f = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$



direction of steepest ascent

Note the coefficients of x & y.

if the i/p x is changed by 1 unit, overall change in o/p (f) is more as compared to change in y. This tells us, gradients can help us decide which components in i/p should be changed & by how much so that the difference b/w i/p & o/p is squashed.

More on Gradient descent:

Gradient, $\nabla E(\vec{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots \frac{\partial E}{\partial w_n}\right]$

Training Rule: $\Delta\vec{w} = -\eta \nabla E(\vec{w})$

i.e. $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$; for each weight component

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2 = \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

$$= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d)$$

$$= \frac{1}{\cancel{2}} \sum_d \cancel{2} (t_d - o_d)(-\vec{x}_d)$$

: GD (training examples, η):

Each training example is a pair $\langle \vec{x}, t \rangle$ where $\vec{x}$ is the vector of i/p values & t is the target o/p value. η is the learning rate.

1. Initialize each w_i to some small random value
2. Until the termination condition is met, do
   - initialize each Δw_i to zero
   - for each $\langle \vec{x}, t \rangle$, do
     * input the instance $\vec{x}$ to the activation unit & compute the o/p, $o_d = f(\vec{x}_d)$
     * for each linear unit. w_i do
       $\Delta w_i \leftarrow \Delta w_i + \eta (t_d - o_d) x_d$
   - for each linear unit weight w_i, do
     $w_i \leftarrow w_i + \Delta w_i$

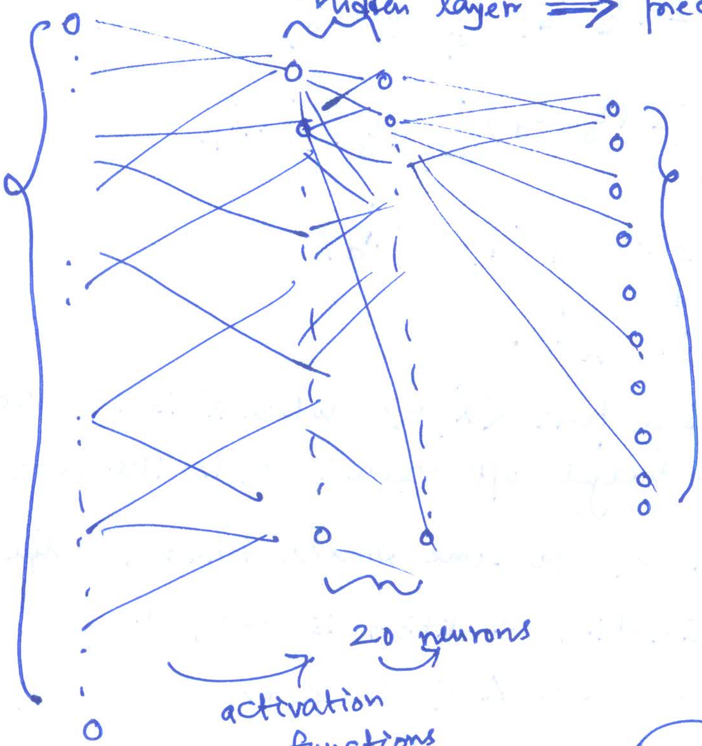
Additional Reads:

1. Hinton's slides
2. Goodfellow's text on Deep Learning.

: A high level abstraction of a Neural Network:

Hidden layer $\Longrightarrow$ pieces of digits (each of 0–9) via combination of weight & i/P's

28x28 pixel = 784



O/P layer to hold the digits 0–9 (MNIST data)

$\Updownarrow$

digit recognition

20 neurons

activation functions

1) Each neuron holds a # i.e $\overbrace{f(x_0, \dots x_{783})}^{\text{function}} = \begin{bmatrix} O_0 \\ \vdots \\ O_9 \end{bmatrix}$

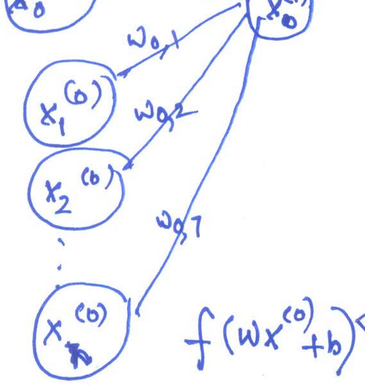
2) Entire network is just a function : takes in 784 units as i/P & produces 10 units as o/P.

$784 \times 20 + 20 \times 20 + 20 \times 10$ — weights

$20 + 20 + 10$ — biases

$\Big\}$ $15740 + 600 + 50 = 16390$ i/P's

neurons in i/P layer



$$x_0^{(1)} = f\left(w_{0,0}\, x_0^{(0)} + w_{0,1}\, x_1^{(0)} + \cdots w_{0,n}\, x_n^{(0)} + b_0\right)$$

$$f\left(\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & & & \\ w_{k,0} & w_{k,n} & \cdots & w_{k,n} \end{bmatrix} \begin{bmatrix} x_0^{(0)} \\ x_1^{(0)} \\ \vdots \\ x_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix} \right)$$

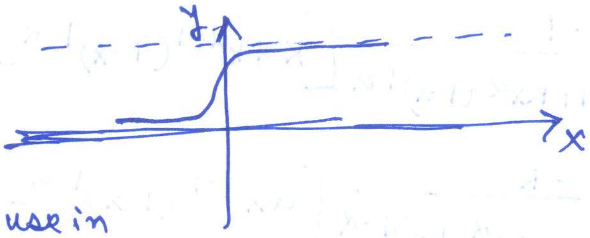
$f(Wx^{(0)} + b) \Longleftrightarrow$

Communicates b/w layers

Gradient descent regression; $\Rightarrow$ printed notes:

Building Block 3: (Activation function)

(i) Sigmoid:



$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Nice Properties: (to use in back propagation)

$$\frac{d\sigma}{dx} = \frac{1}{(1+e^{-x})^2}\left(+e^{-x}\right) = \frac{1+e^{-x}-1}{(1+e^{-x})^2}$$

$$= \frac{1}{1+e^{-x}} - \frac{1}{(1+e^{-x})^2}$$

$$= \sigma(x) - \sigma(x)^2 = \sigma(x)(1-\sigma(x))$$

O/P of the activation function $\Rightarrow$ how positive the relevant weighted sum is!

Error Gradient for a sigmoid unit:

$$w_i \rightarrow \text{net}_d \xrightarrow{\sigma} o_d$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2$$

$$= \frac{1}{\cancel{2}}\sum_{d\in D}\cancel{2}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$

$$= \sum_{d\in D}(t_d - o_d)\left(-\frac{\partial o_d}{\partial w_i}\right)$$

$$= -\sum_{d\in D}(t_d - o_d)\frac{\partial o_d}{\partial \text{net}_d}\cdot\frac{\partial \text{net}_d}{\partial w_i}$$

Note,

$$\frac{\partial o_d}{\partial \text{net}_d} = \frac{\partial \sigma(\text{net}_d)}{\partial \text{net}_d} \;;\; \text{since } o_d \rightarrow \sigma(\text{net}_d)$$

network data passed through the activation function σ

$$\cong \frac{\partial \sigma}{\partial x} = o_d(1-o_d)$$

$$\& \quad \frac{\partial \text{net}_d}{\partial w_i} = \frac{\partial(\vec{w}\cdot\vec{x}_d)}{\partial w_i} = x_{i,d}$$

$$\text{So,} \quad \frac{\partial E}{\partial w_i} = -\sum_{d\in D}\underbrace{(t_d - o_d)}_{\text{loss}}\;\underbrace{o_d(1-o_d)}_{\text{slope of loss function}}\;\underbrace{x_{i,d}}_{\text{i/p data}}$$

See, how helpful the property $\frac{d\sigma}{dx} = \sigma(1-\sigma)$

(ii) SBAF:
$$y = \frac{1}{1+Kx^{\alpha}(1-x)^{1-\alpha}}$$

$$\ln y = \ln 1 - \ln\left(1+Kx^{\alpha}(1-x)^{1-\alpha}\right) = -\ln\left(1+Kx^{\alpha}(1-x)^{1-\alpha}\right)$$

$$\Rightarrow \frac{1}{y}\frac{dy}{dx} = \frac{-1}{1+Kx^{\alpha}(1-x)^{1-\alpha}}\left[K\alpha x^{\alpha-1}(1-x)^{1-\alpha} - Kx^{\alpha}(1-\alpha)(1-x)^{1-\alpha-1}\right]$$

$$= \frac{-K}{1+Kx^{\alpha}(1-x)^{1-\alpha}}\left[\alpha x^{\alpha-1}(1-x)^{1-\alpha} - (1-\alpha)x^{\alpha}(1-x)^{1-\alpha-1}\right]$$

$$= -\frac{Kx^{\alpha}(1-x)^{1-\alpha}}{1+Kx^{\alpha}(1-x)^{1-\alpha}}\left[\frac{\alpha}{x} - \frac{(1-\alpha)}{1-x}\right]$$

$$= -\frac{Kx^{\alpha}(1-x)^{1-\alpha}}{1+Kx^{\alpha}(1-x)^{1-\alpha}}\left[\frac{\alpha(1-x)-(1-\alpha)x}{x(1-x)}\right]$$

$$= -\frac{Kx^{\alpha}(1-x)^{1-\alpha}}{1+Kx^{\alpha}(1-x)^{1-\alpha}}\left(\frac{\alpha-x}{x(1-x)}\right)$$

Note,

$$1+Kx^{\alpha}(1-x)^{1-\alpha} = \frac{1}{y}$$

$$\Rightarrow Kx^{\alpha}(1-x)^{1-\alpha} = \frac{1}{y} - 1 = \frac{1-y}{y}$$

$$\therefore \frac{1}{y}\frac{dy}{dx} = -\frac{1-y}{y}\cdot y\,\frac{\alpha-x}{x(1-x)}$$

$$\Rightarrow \frac{dy}{dx} = \frac{(y-1)y}{x(1-x)}(\alpha-x)$$

note

Exploding Gradient: $x = 0, 1$;
Vanishing Condition: $\alpha = x$ (critical point)
$\alpha > x$, $\frac{dy}{dx} < 0$; $\alpha < x$, $\frac{dy}{dx} > 0$

} input needs to be normalized