Introduction to Neural Networks

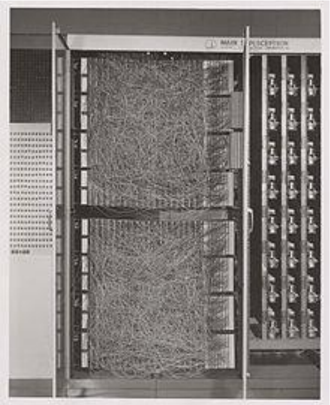
Perceptron, XOR Problem, Multi-layer Perceptron (MLP), Cost Functions, Activation functions, Output units

EE 5179

Instructor: Kaushik Mitra

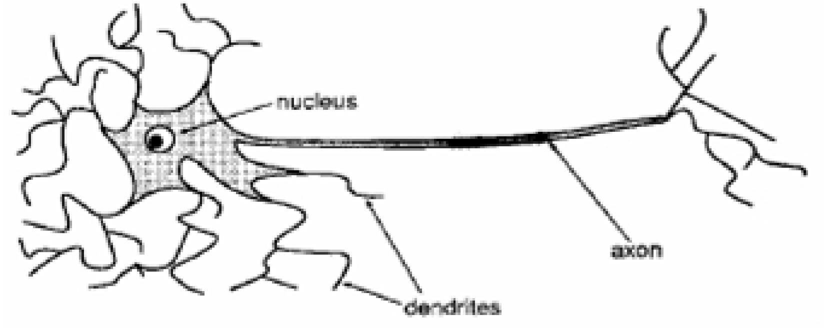
History

- 1932 - *The Integrative Action of the Nervous System*, **Sir Charles Scott Sherrington**
  - Nervous system - interconnection of individual entities (neuron)
- 1943 - *A Logical Calculus of Ideas Immanent in Nervous Activity*, **Warren McCulloch** and **Walter Pitts**
  - McCulloch and Pitt’s model
- 1949 - *The Organization of Behavior,* **Donald Hebb**
  - Hebbian learning
- 1953 - *The Perceptron*, **Rosenblatt**
- 1969 - *Limitation of Perceptrons,* **Minsky** and **Papert**
- 1980’s - *Connectionism*
  - *Error Back Propagation,* **Proposed simultaneously by Many**
- 20th century Deep learning
  - CNNs - LeNet, AlexNet, VGGNet, ResNet
  - Deep LSTMs,
- The **deep saga………** what followed is discussed in the last class

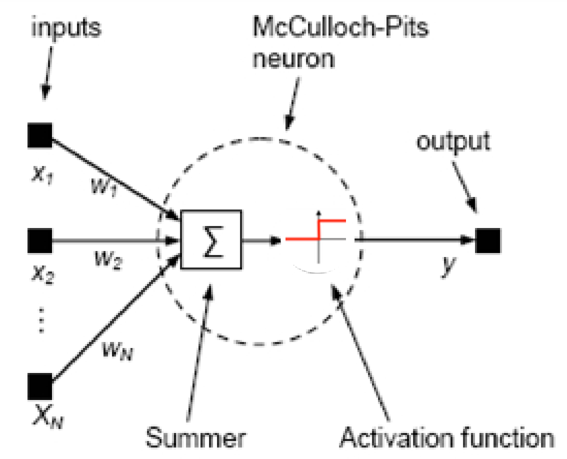


The Mark 1 Perceptron
By Rosenblat

McCulloch - Pits model



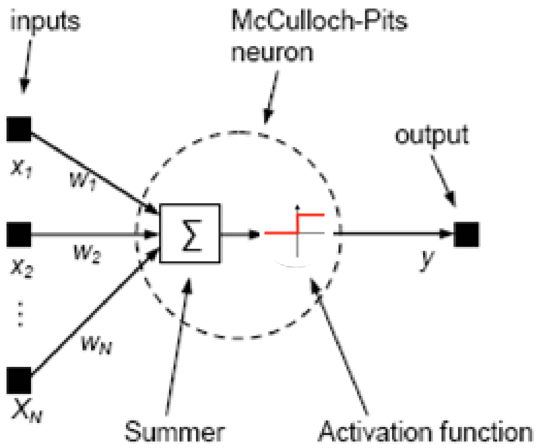
Biological neuron



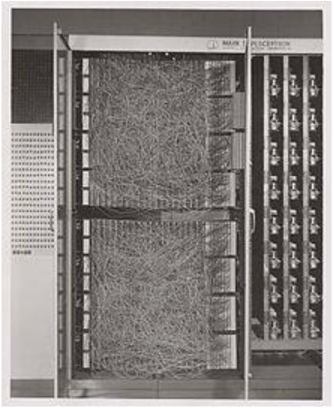
$$\sum_{i=1}^{n} w_i x_i > \mu$$

The input and output are binary and the weights are either excitatory (+1) or inhibitory (-1).

The Perceptron - Rosenblatt (1953)



$$\sum_{i=1}^{n} w_i x_i > \mu$$

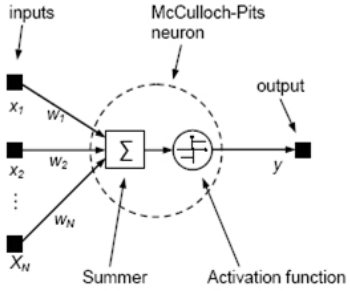


*Pic courtesy, wikipedia

The Mark 1 Perceptron
By Rosenblat
for digit recognition

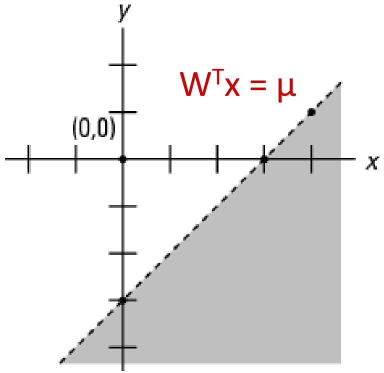
Weights can be any real number. He also proposed a learning algorithm for learning the weights from training data.

Perceptron - geometrical interpretation



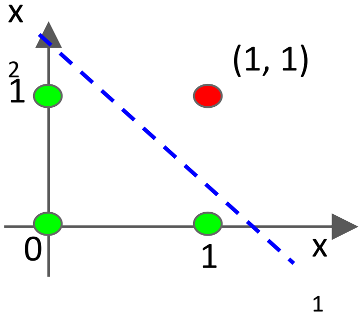
$$\sum_{i=1}^{n} w_i x_i > \mu$$

, What does this inequality imply in 2D case? Half plane



*Pic courtesy, cliffsnotes

$\mathbb{X}$	*AND*
(0, 0)	0
(0, 1)	0
(1, 0)	0
(1, 1)	1



Solve for W, μ:

$$x_1 + x_2 > 1.5$$

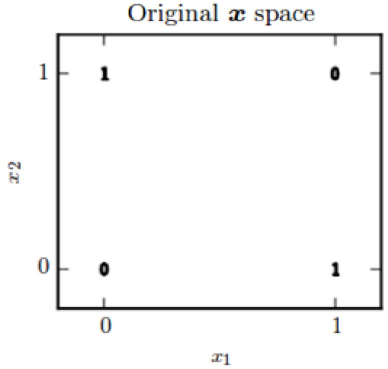
$$w_1 = 1,\ w_2 = 1 \text{ and } \mu = 1.5$$

Any function that is linearly separable can be computed by a perceptron

Perceptron - Limitations

Goal: learn the XoR function (*f**)

$\mathbb{X}$	*f**
(0, 0)	0
(0, 1)	1
(1, 0)	1
(1, 1)	0



Task is adjust parameters ϑ, such that f is as close as to f^*

$$y = f(x,\vartheta) \qquad L = \sum_{\{x\in\mathbb{X}\}} \left(f^*(x) - f(x,\vartheta) \right)^2$$

Lets use our perceptron for f, $\vartheta = \{w,b\}$

$$f(x; w,b) = w^T x + b$$

Solve for {w,b}

W = 0, b = 0.5; output is 0.5 everywhere

Why this linear function can't model XoR**?**

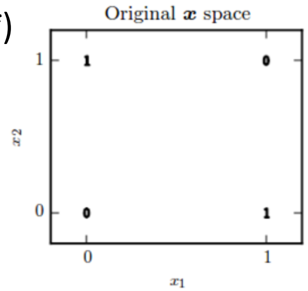
How to tackle this problem?

- Can we use more than one line?
- Yes, but how?

*Slide courtesy, Ian Goodfellow et al., deep learning book

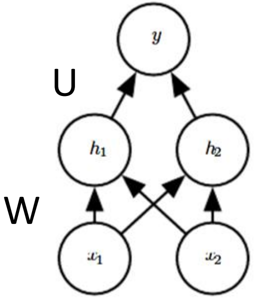
Perceptron - Limitations

Goal: learn the XoR function (f^*)



How to tackle this problem?

– Add a hidden layer with two units



$y = f^{(2)}(h; U, c)$

$y = f^{(2)}(\, f^{(1)}(x)\,)$

$h = f^{(1)}(x; W, b)$

What should $f^{(1)}$ compute?

– If its linear again the composition still remains linear

$f^{(2)}(h) = U^T h$; since $h = Wx$

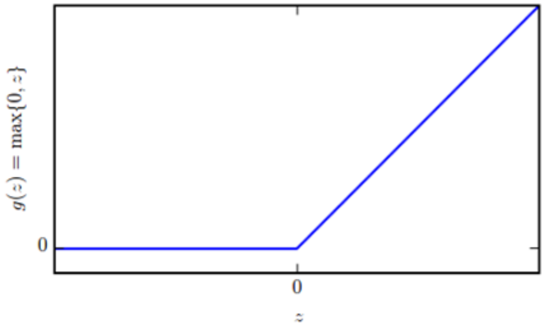
$y = U^T W x = W'x$

– $f^{(1)}$ should be nonlinear to extract useful features

$h = f^{(1)}(x; W, b) = g(Wx+b)$

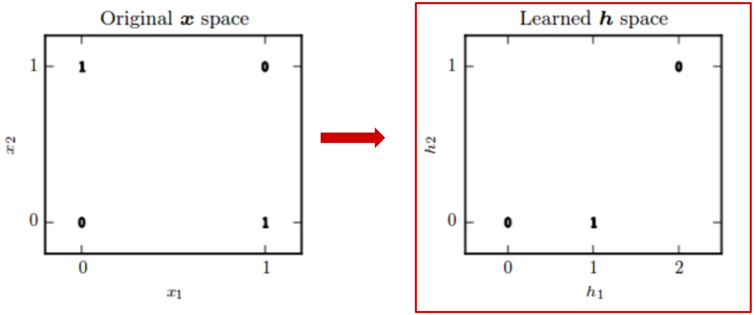
– g is referred as activation function commonly

– We will use ReLU here
  - ❑ Rectified Linear Unit (widely used)
  - ❑ $g(z) = \max\{0,z\}$



*Slide courtesy, Ian Goodfellow et al., deep learning book

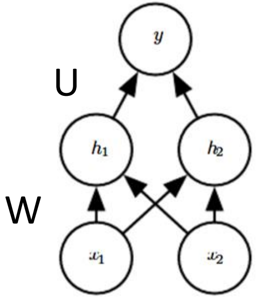
Goal: learn the XoR function (f^*)



Perceptron - Limitations

How to tackle this problem?

- Add a hidden layer with two units
- Use ReLU activation in 1st layer



$$y = U^T h + c \;; \quad y = U^T \underbrace{\max\{0, Wx+b\}}_{\text{ReLU}} + c$$

$$h = g(Wx+b)$$

Let,

$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 \\ -2 \end{bmatrix},$$

$$c = 0$$

$$X = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \qquad WX = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 1 & 1 & 2 \end{bmatrix}$$

$$WX + b = \begin{bmatrix} 0 & 1 & 1 & 2 \\ -1 & 0 & 0 & 1 \end{bmatrix} \qquad \boxed{h = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{Upon ReLU}}$$

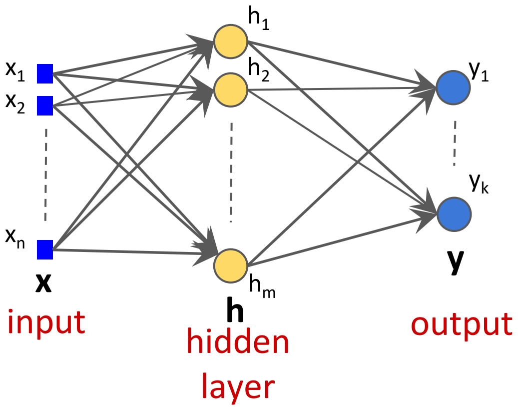
After layer2

$$[0 \; 1 \; 1 \; 0]$$

*Slide courtesy, Ian Goodfellow et al., deep learning book

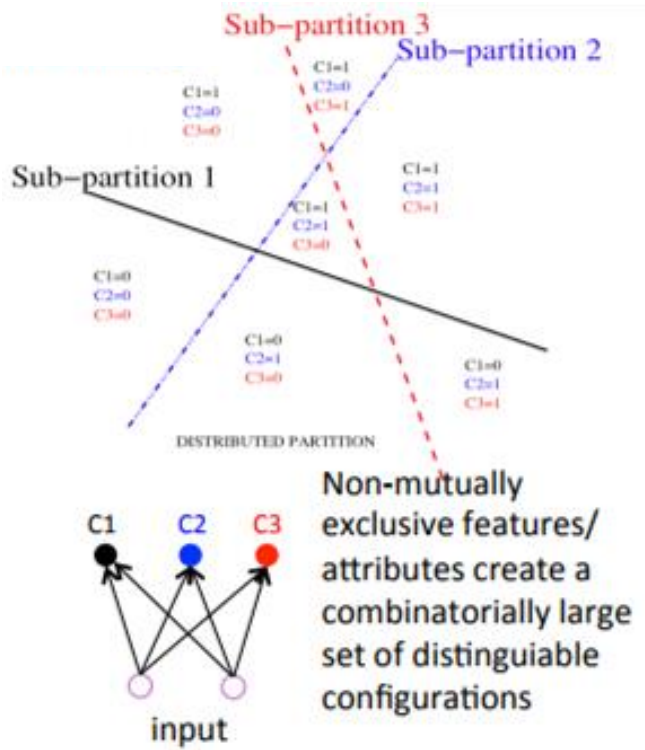
Multi-layer Perceptrons (MLP)

A typical feed forward neural network



$$\mathbf{h} = f(W\mathbf{x} + \mathbf{b}_1); \quad \mathbf{y} = g(U\mathbf{h} + \mathbf{b}_2)$$

With more hidden units network is more expressible



*Pic courtesy, Yoshua Bengio

Feedforward Neural Networks - Cost functions

For regression,

$$J(\theta) = \frac{1}{2}\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}} ||\boldsymbol{y} - f(\boldsymbol{x};\boldsymbol{\theta})||^2$$

$$\frac{1}{2}\sum_{\{x_i, y_i\}} ||y_i - f(x_i, \theta)||^2$$

For classification,

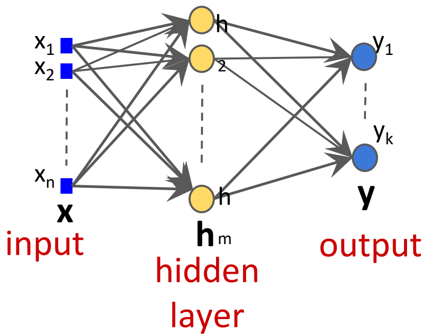
- Typically outputs a probability vector $q(c = k|\mathrm{x})\ \forall\ k$
- How do you compare two distributions?
  - ❑ KL divergence, KL($p \| q$)

$$D_{KL}(p(x)||q(x)) = \sum_{x \in X} p(x) \ln \frac{p(x)}{q(x)}$$

$$= \sum p(\mathrm{x}) \ln p(\mathrm{x}) - p(\mathrm{x}) \ln q(\mathrm{x})$$

$$= \underbrace{-H(p)}_{\text{entropy}} + \underbrace{H(p,q)}_{\text{cross-entropy}}$$

$$J(\theta) = \sum_{x_i, y_i} H(p(x_i), q(x_i))$$

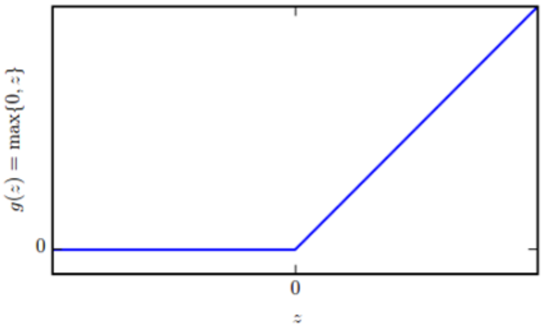


Activation functions

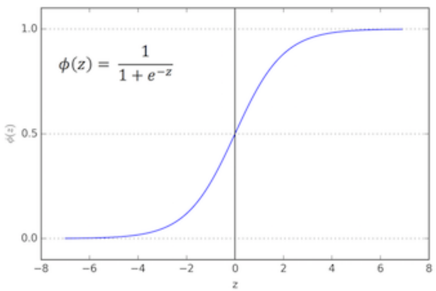
h = *g*(Wx+b); Affine transformation followed by activation function, *g*

– Very important factor in learning features

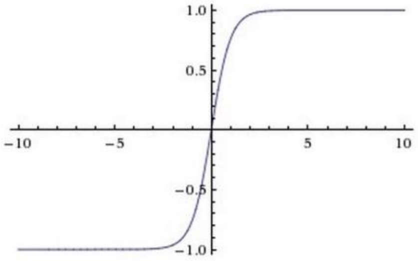
g(z) = max{0,z}, ReLU



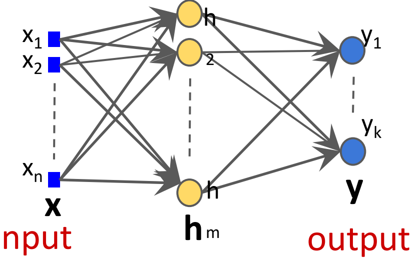
g(z) = σ(z), sigmoid



tanh(z) = 2σ(2z) - 1



Output units



- Linear units for real valued outputs
  - ❑ Activation function is left to be linear
  - ❑ Given features h,

    y’ = Wh+b

  - ❑ Most commonly used with regression tasks

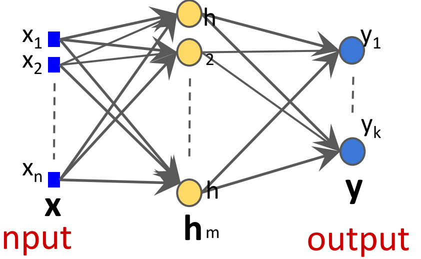
- Say you want to do binary classification
  - ❑ What kind of distribution describes output?
    Bernouli
  - ❑ How to constrain the output - valid probability?
    Can you use linear activation?

$$P(y = 1 \mid \boldsymbol{x}) = \max\left\{0, \min\left\{1, \boldsymbol{w}^\top \boldsymbol{h} + b\right\}\right\}.$$

  - ❑ What is the problem? *Not amenable for gradient based learning*
  - ❑ Instead, use sigmoid unit - output ∈ [0,1]

$$\hat{y} = \sigma\left(\boldsymbol{w}^\top \boldsymbol{h} + b\right)$$

Output units



– Now, say we want to do multi-class classification (K classes)
  - Output should be K probabilities,
    $p_k = p(class = k|x)\ \forall\ k = 1$ to K
  - Can we use K sigmoid units?
    Won't be sufficient, since probabilities are not constrained to sum to 1
    $\sum_k p_k = 1$
  - We will look at softmax unit for this
    Idea is to convert a vector of real values to valid probabilities,
    How?
    - Make all the elements positive
    - Normalize the values

– Let, $\mathbf{z} = [z_1, \ldots, z_K]^T$; $\mathbf{z} = W\mathbf{h} + \mathbf{b}$

$$\text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}.$$