Introduction to Neural Networks

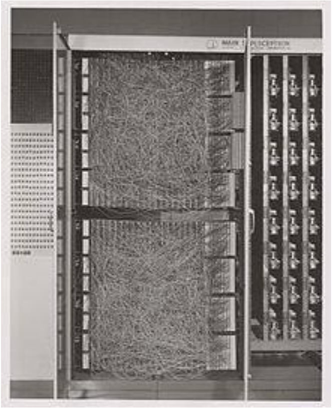
Perceptron, XOR Problem, Multi-layer Perceptron (MLP), Cost Functions, Activation functions, Output units

EE 5179

Instructor: Kaushik Mitra

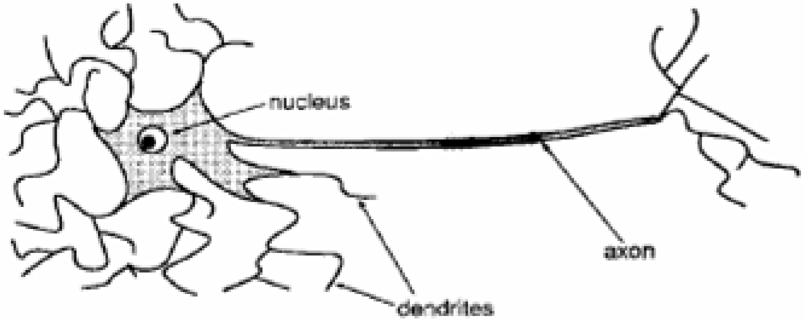
History

- 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

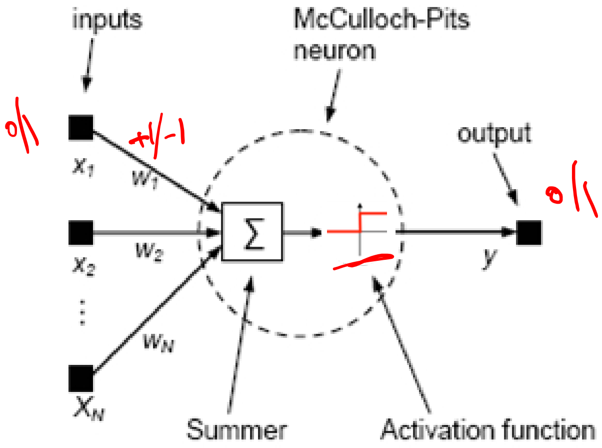


The Mark 1 Perceptron
By Rosenblat

McCulloch - Pits model



Biological neuron



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

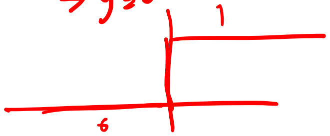
$w_1x_1 + w_2x_2 + w_N x_N > \mu$

$\Rightarrow y = 1$

$w_1x_1 + \dots < \mu$

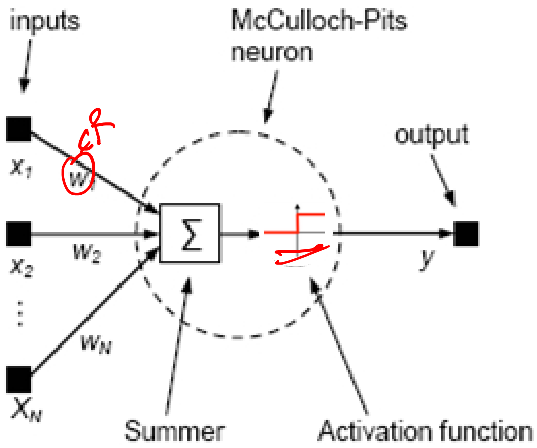
$\Rightarrow y = 0$

$y = f(\underbrace{\sum w_i x_i - \mu}_{>0})$

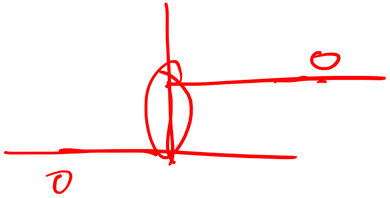


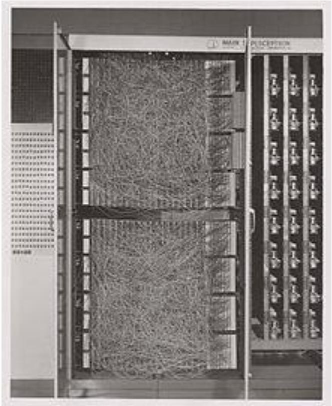
The input and output are binary and the weights are either excitatory (+1) or inhibitory (-1).

The Perceptron - Rosenblatt (1953)



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



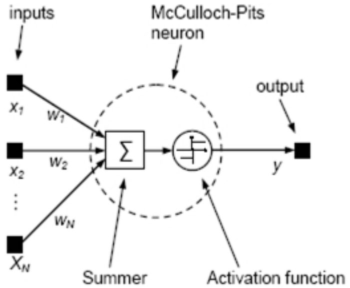


*Pic courtesy, wikipedia

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



$f(\sum w_i x_i - \mu)$

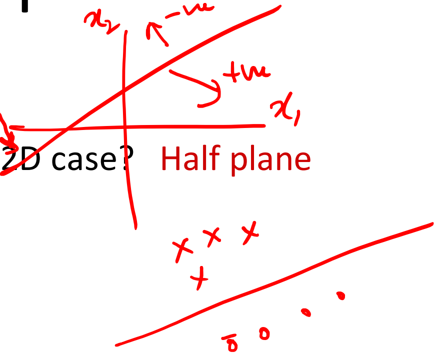
$w_1x_1 + w_2x_2 > \mu$

$w_1x_1 + w_2x_2 = \mu$

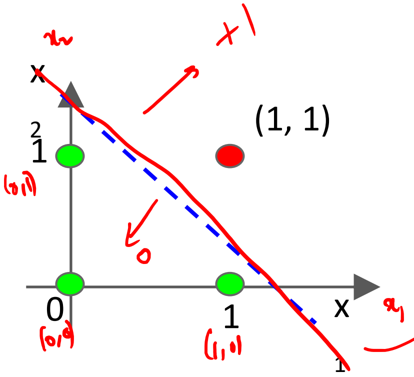
$w_1x_1 + w_2x_2 - \mu > 0$

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

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



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



Solve for W, μ:

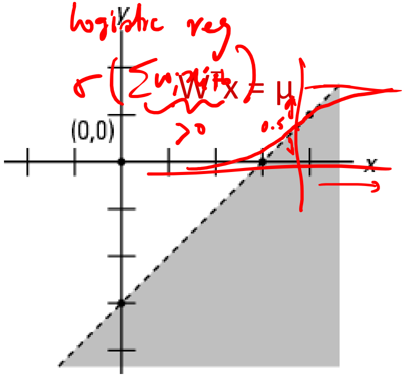
$x_1 + x_2 = 1.5$

$x_1 + x_2 > 1.5$

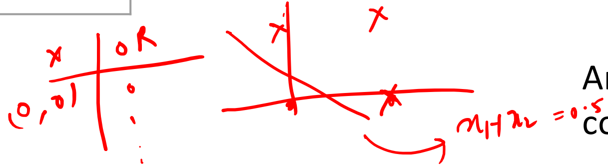
$w_1 = 1$, $w_2 = 1$ and $\mu = 1.5$

logistic reg

$\sigma(\sum w_i x_i) > 0$



*Pic courtesy, cliffsnotes



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

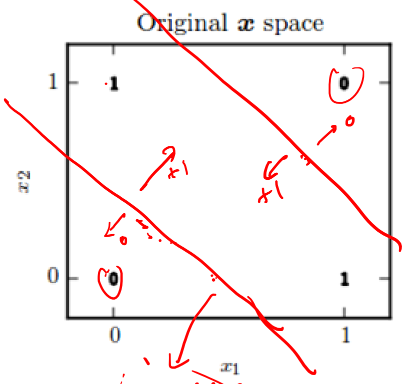
Perceptron - Limitations

$y = f(\sum w_i x_i - \mu)$

$\min_{w_i, \mu} L$

Goal: learn the XoR function (f^*)

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



(0,0) → 0, 1 → 0

(0,1) → 1, 1 → 1

(1,0) → 1, 1 → 1

(1,1) → 1, 0 → 0

o/p network

AND

NAND

OR

hidden layer

i/p layer

How to tackle this problem?

– Can we use more than one line?

– Yes, but how?

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

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

x_i, y_i

Lets use our perceptron for f, ϑ = {w,b}

$f(x; w,b) = w^T x + b$ $\qquad \sum_i (y_i - (w^T x_i + b))^2$

Solve for {w,b}

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

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

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

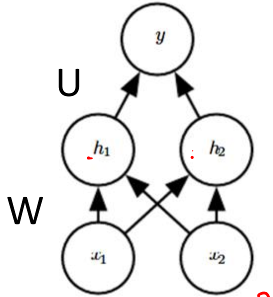
Perceptron - Limitations

$h = Ax$
$y = Bh$
$y = BAx$
$y = Cx$

How to tackle this problem?

– Add a hidden layer with two units

$y_{1\times1} = f(U_{1\times2} h_{2\times1} + c_{1\times1})$



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

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

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

$\begin{pmatrix} h_1 \\ h_2 \end{pmatrix} = f(W_{2\times2}\, x_{2\times1} + b_{2\times1})$

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

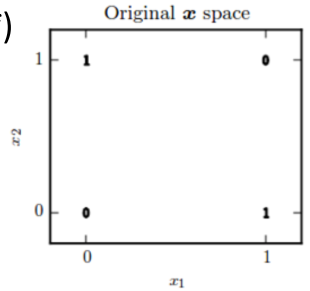
$h_1 = f(w_{11}x_1 + w_{12}x_2 + b_1)$
$h_2 = f(w_{21}x_1 + w_{22}x_2 + b_2)$
$h = f(W$

– If its linear again the composition still remains linear

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

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

Goal: learn the XoR function (f^*)



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

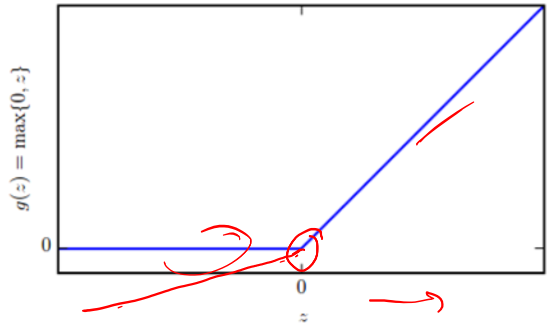
$h = f^{(1)}(x; W, b) = g(Wx+b)$

– g is referred as activation function commonly

– We will use ReLU here

❑ Rectified Linear Unit (widely used)

❑ $g(z) = \max\{0,z\}$



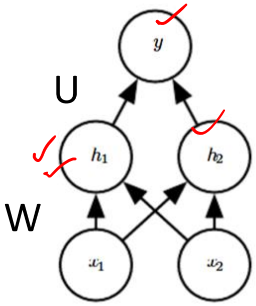
*Slide courtesy, Ian Goodfellow et al., deep learning book

Perceptron - Limitations

Goal: learn the XoR function (f^*)

How to tackle this problem?

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



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

$h = g(Wx+b)$

$h_1 = \text{ReLU}(W_{11}x_1 + W_{12}x_2 + b_1)$

$h_1 = (x_1 + x_2 + 0)$

$h_2 = \text{ReLU}(x_1 + x_2 - 1)$

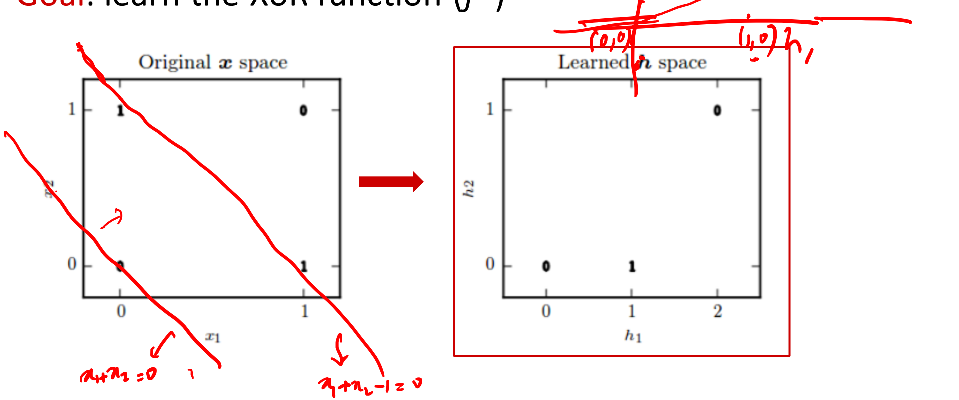
Let,

$W = \begin{array}{cc} W_{11} & W_{12} \\ \end{array} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{array}{cc} W_{21} & W_{22} \end{array}$, $b = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$, $U = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$,

c = 0

$y = (h_1 - 2h_2)$

$h_1 - 2h_2 = 0$ $h_2 = \frac{1}{2}h_1$



$X = \begin{array}{c} x_1 \\ x_2 \end{array} \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$ WX = $\begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 1 & 1 & 2 \end{bmatrix}$

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

$U^T h + c$

After layer2

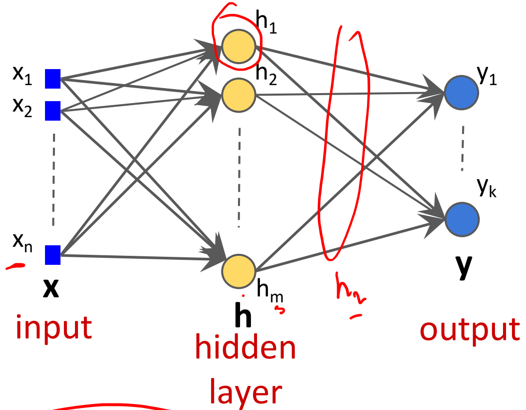
$[0\ 1\ 1\ 0]$

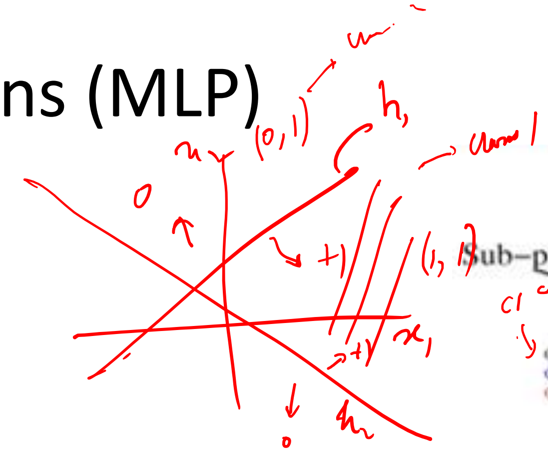
	h_1	h_2	y
(0,0) →	0	0 →	0
(1,0) →	1	0 →	1
(0,1) →	1	0 →	1
(1,1) →	2	1 →	0

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

Multi-layer Perceptrons (MLP)

A typical feed forward neural network





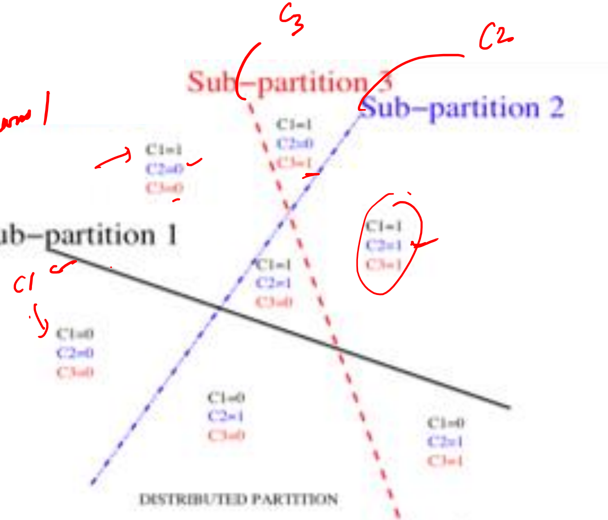
$\mathbf{h} = f(W\mathbf{x} + \mathbf{b}_1)$; $\mathbf{y} = g(U\mathbf{h} + \mathbf{b}_2)$

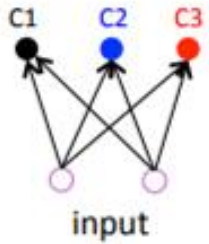
$m \times 1 \quad m \times n \quad n \times 1 \quad m \times 1$

$h_2 = f(W'h + b'_2)$

With more hidden units network is more expressible

$$\begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_m \end{bmatrix} = f\begin{pmatrix} W_{11}x_1 + W_{12}x_2 + \dots + b_{11} \\ \\ W_{m1}x_1 + \dots \ b_{1m} \end{pmatrix}$$





Non-mutually exclusive features/ attributes create a combinatorially large set of distinguiable configurations

*Pic courtesy, Yoshua Bengio

Feedforward Neural Networks - Cost functions

For regression,

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

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

training data (x_i, y_i)

min $J(\theta)$

$\theta = \{w, b\}$

cross-entropy

$$\sum_x p(x) \ln \frac{1}{q(x)}$$

tiger $= p(x)$

$$\text{entropy} = \sum_x p(x) \ln \frac{1}{p(x)}$$

For classification,

$q(x) = f(x, \theta)$

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

$$D_{KL}(p(x)||q(x)) = \sum_{x\in X} p(x)\ln\frac{p(x)}{q(x)} = \underbrace{\sum p(x) \ln \frac{1}{q(x)}}_{\text{cross-entropy}} - \underbrace{p(x) \ln \frac{1}{p(x)}}_{H(p)}$$

$a \to 0.5 \to p(a)$, $q(a)$
$b \to 0.2 \to p(b)$, $q(b)$
$c \to 0.2 \to p(c)$
$d \to 0.1 \to p(d)$

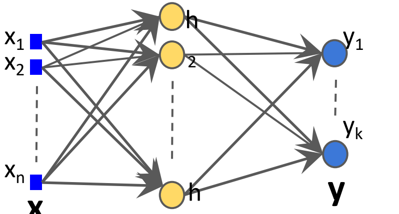
$$= \sum p(x) \ln p(x) - p(x) \ln q(x)$$

w, b

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

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

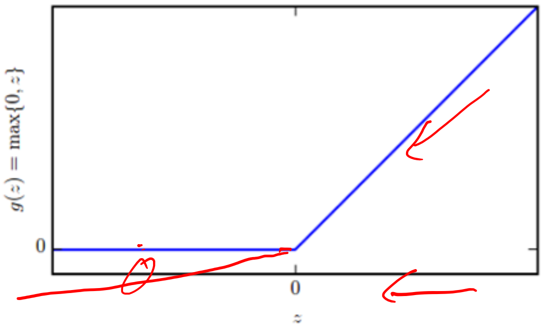
Activation functions



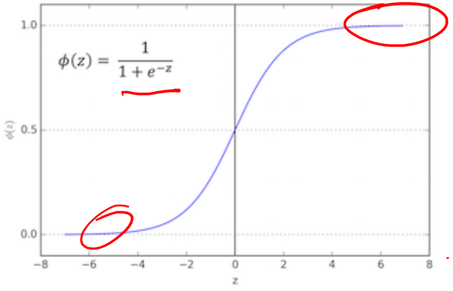
h = *g*(Wx+b); Affine transformation followed by activation function, *g*

– Very important factor in learning features

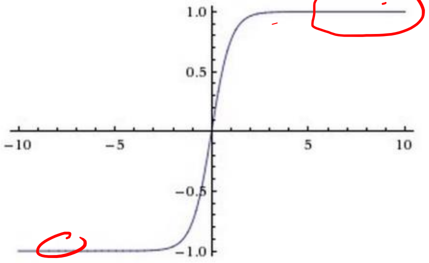
g(z) = max{0,z}, ReLU

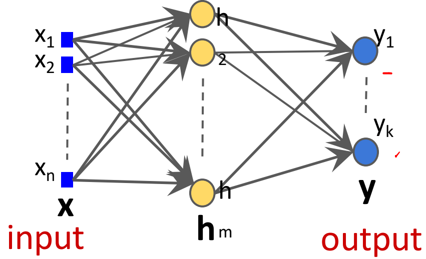


g(z) = σ(z), sigmoid



tanh(z) = 2σ(2z) - 1





Output units

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

    $$y' = Wh+b$$

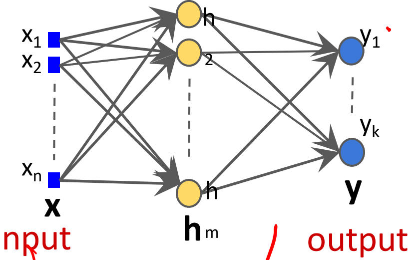
  - Most commonly used with regression tasks

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

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

    $\in [0,1]$
  - 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

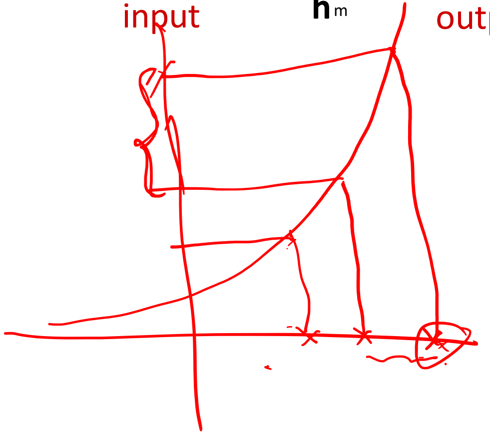
– Now, say we want to do multi-class classification (K classes)

❑ 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



$\frac{\sigma(z_i)}{\sum_j \sigma(z_j)} = 1$ ≥ 0

softmax

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

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

≥ 0

$\sum \frac{\exp(z_i)}{\sum \exp(z)} = 1$

z_1, z_2, z_3

$\frac{1}{1+e^{z_2-z_1}}$

exp

$\frac{e^{z_1}/e^{z_1}}{e^{z_1}+e^{z_2}/e^{z_1}}$

$\frac{e^{z_2}}{e^{z_1}+e^{z_2}}$

$\frac{1}{1+e^{-z_1}}$