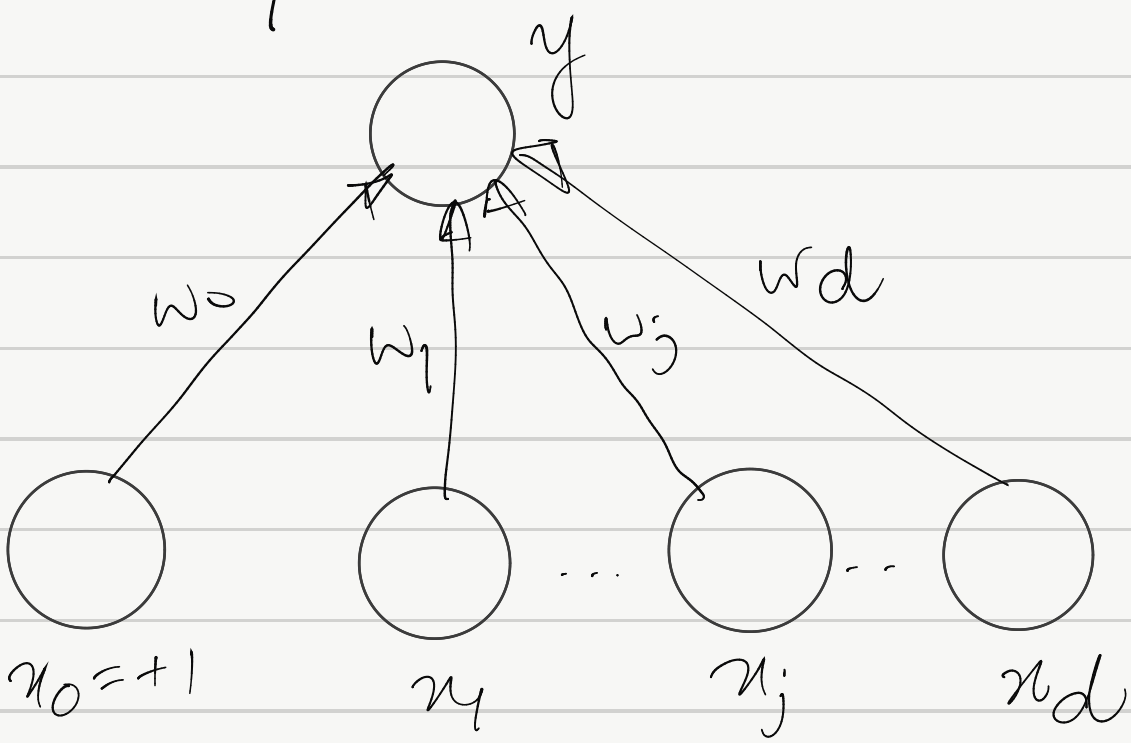
MLFA-Neural-Net-1

06-Mar-2023 at 3:38 PM

Multi Layer Perceptron

Perceptron Recap:



$x_j \in \mathbb{R} \quad j = 1, 2 \ldots, d$

$w_j \in \mathbb{R}$

$$y = \sum_{j=1}^{d} w_j x_j + w_0$$

$w_0 \rightarrow$ intercept value

$\rightarrow$ weight of extra "bias" unit x_0

$x_0 = +1$

$$y = w^T x$$

$$w = [w_0 \; w_1 \ldots w_d]^T \qquad w \in \mathbb{R}^{(d+1)\times 1}$$

$$x = [1 \; x_1 \ldots x_d]^T \qquad x \in \mathbb{R}^{(d+1)\times 1}$$

$y = w^T x$ defines a hyperplane that divides the input space into two halves:
$y > 0$ in one space and $y < 0$ in the other space.

This can be used to define a linear discriminant function by checking the sign of y.

$$s(a) = \begin{cases} 1 & \text{if } a > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{choose} \begin{cases} C_1 & \text{if } s(w^T x) > 0 \\ C_2 & \text{otherwise} \end{cases}$$

linear discriminant assumes that the classes are linearly separable.

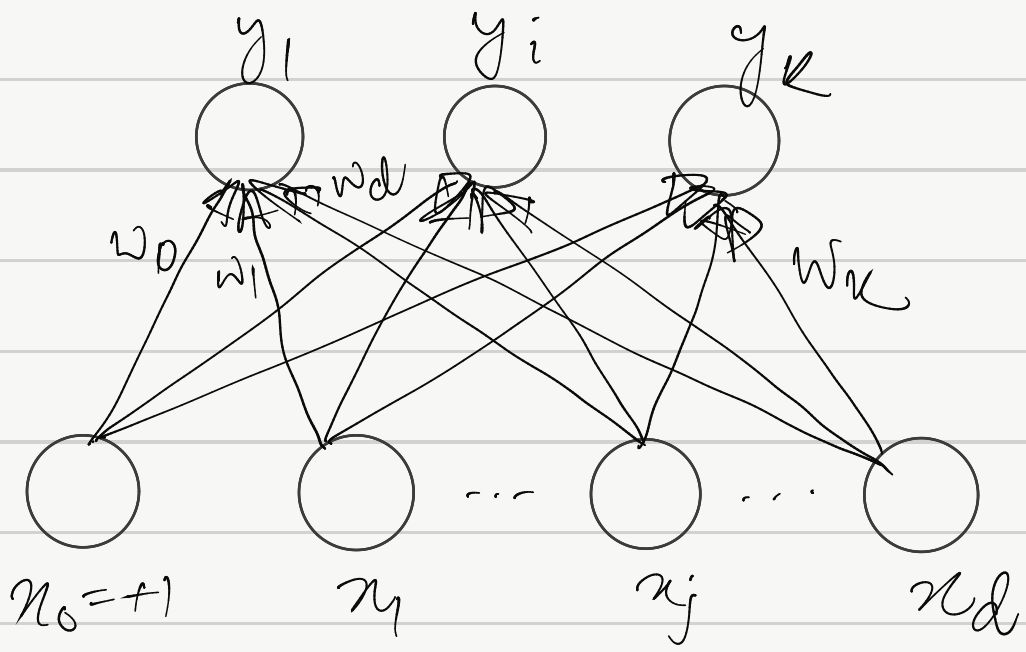
In reality, the classes may not be exactly linearly separable
→ Calculate risk
→ need posterior probability $P(C_i | x_t)$

Sigmoid (logistic)
$O = w^T x$

$$P(C_1 | x^t) = y = \text{sigmoid}(O) = \frac{1}{1 + \exp[-w^T x]}$$

$$P(C_2 | x^t) = 1 - P(C_1 | x^t)$$

The case $K>2$



$$x \in \mathbb{R}^{(d+1)\times 1} \quad w_i \in \mathbb{R}^{(d+1)\times 1}$$

$$y_i = \sum_{j=1}^{d} w_{ij} x_j + w_{i0} = w_i^T x$$

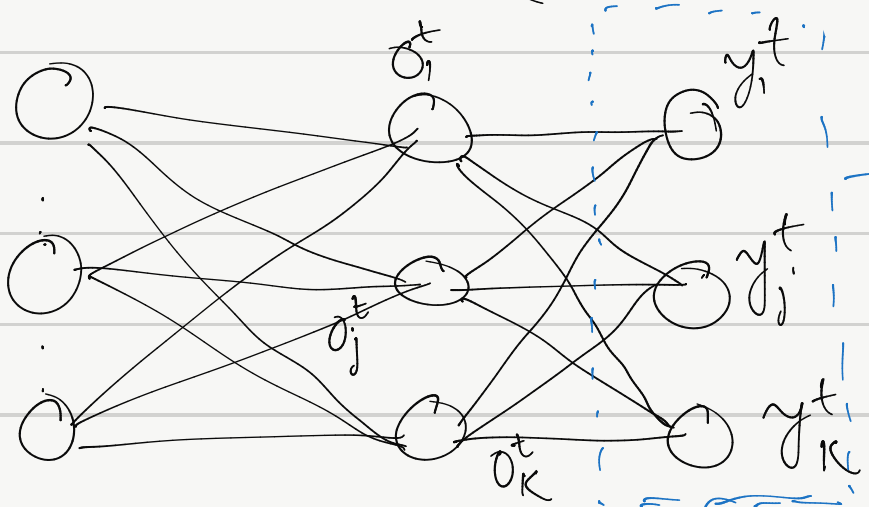
$$y = Wx \qquad W \in \mathbb{R}^{K\times(d+1)}$$

Choose C_i if $y_i = \max_k y_k$

If we need posterior probability

$$o_i = w_i^T x$$

$$P(C_i|x^t) \quad y_i = \frac{\exp[w_i^T x]}{\sum_k \exp[w_k^T x]} \qquad \text{SOFTMAX}$$



Training a perceptron:

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j}$$

Update function: $w_j = w_j + \Delta w_j$

Training data: $X = \{x^t, r^t\}_{i=1}^N$

Error or Loss Function

Regression

$$E^t(w|x^t, r^t) = \frac{1}{2}(r^t - y^t)^2$$

$$= \frac{1}{2}[r^t - (w^T x^t)]^2$$

$$\Delta w_j^t = (r^t - y^t) x_j^t$$

$$j = 0, 1, \ldots, d$$

classification

Binary classification

$$E^t(w|x^t, r^t) = -r^t \log y^t - (1 - r^t) \log (1 - y^t)$$

$$y^t = \text{Sigmoid}(w^T x^t)$$

$$\Delta w_j^t = (r^t - y^t) x_j^t$$

Multiclass classification

$$y_i^t = \frac{\exp(w_i^T x^t)}{\sum_k \exp(w_k^T x^t)}$$

$r^t \rightarrow$ one-hot vector

r is an one-hot vector the label C_i is represented by setting the i^{th} entry to be '1' others '0'

$$E^t(\{w_i\}_i | x^t, r^t) = -\sum_i r_i^t \log y_i^t$$

$$\Delta w_{ij}^t = \eta (r_i^t - y_i^t) x_j$$

Derivation:

Logistic Func:

$$y^t = \frac{1}{1+e^{-ot}} = \frac{e^{ot}}{1+e^{ot}}$$

$$\frac{\partial y^t}{\partial ot} = \frac{(1-e^{ot}).e^{ot} - e^{ot}.e^{ot}}{(1+e^{ot})^2}$$

$$= \frac{e^{ot}}{1+e^{ot}} - \frac{e^{ot}}{1+e^{ot}} \cdot \frac{e^{ot}}{1+e^{ot}}$$

$$= y^t - y^t.\, y^t = y^t(1-y^t)$$

Softmax $\left[\frac{\partial y_i}{\partial z_j}\right]$ with respect to input

$$i=j: \frac{\partial}{\partial z_i}\left[\frac{e^{z_i}}{\Sigma_c}\right] = \frac{\Sigma_c e^{z_i} - e^{z_i}e^{z_i}}{\Sigma_c^2}$$

$$= \frac{e^{z_i}}{\Sigma_c} \cdot \frac{\Sigma_c}{\Sigma_c} - \frac{e^{z_i}}{\Sigma_c}$$

$$= y_i - y_i \cdot y_i = y_i(1-y_i)$$

$$i \neq j \quad \frac{\partial}{\partial z_j}\left[\frac{e^{z_i}}{\Sigma_c}\right] = \frac{0 - e^{z_i}.e^{z_j}}{\Sigma_c^2} = -y_i y_j$$

Binary classification:

$$E = -r^t \log y^t - (1-r^t)\log(1-y^t)$$

$$o^t = W^T x^t \qquad y^t = \text{Sigmoid}(o^t)$$

$$\frac{\partial E}{\partial w_j} = \frac{\partial E}{\partial y^t} \cdot \frac{\partial y^t}{\partial o^t} \cdot \frac{\partial o^t}{\partial w_j}$$

$$= \left[-\frac{r^t}{y^t} + \frac{(1-r^t)}{1-y^t} \right] \cdot y^t(1-y^t) \cdot x_j$$

$$= (y^t - r^t)\, x_j$$

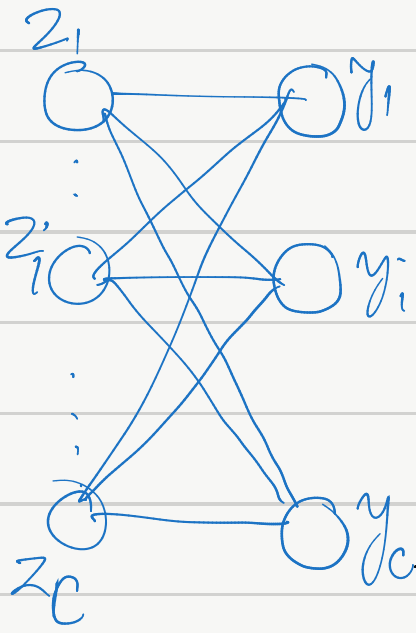
$$\Delta w_j = -\eta (y^t - r^t) x_j = \eta (r^t - y^t) x_j$$

Multiclass

$$E = -\sum_{k=1}^{C} r_k^t \log y_k^t \qquad y_k^t = \frac{e^{o_k}}{\sum_C} \cdot \quad o_k^t = w_k^T x^t$$

$$\frac{\partial E}{\partial w_{ij}} = -\sum_{k=1}^{C} \frac{\partial}{\partial w_{ij}} \left[r_k^t \log y_k^t \right] = -\sum_{k=1}^{C} r_k^t \frac{\partial}{\partial w_{ij}} \log y_k^t$$

$$= -\sum_{k=1}^{C} \frac{r_k^t}{y_k^t} \cdot \frac{\partial y_k^t}{\partial w_{ij}}$$



$$= -\sum_{k=1}^{C} \frac{r_k^t}{y_k^t} \cdot \frac{\partial y_k^t}{\partial o_i^t} \cdot \frac{\partial o_i^t}{w_{ij}}$$

$$= -\frac{r_i^t}{y_i^t} \cdot \frac{\partial y_i^t}{\partial o_i^t} \cdot \frac{\partial o_i^t}{\partial w_{ij}} - \sum_{i \neq k}^{C} \frac{r_k}{y_k} \cdot \frac{\partial y_k^t}{\partial o_i^t} \cdot \frac{\partial o_i^t}{\partial w_{ij}}$$

$$= -\frac{r_i^t}{y_i^t} \cdot y_i^t (1-y_i^t) \cdot x_j + \sum_{i \neq k}^{C} \frac{r_k}{y_k^t} \cdot y_i^t y_k^t \cdot x_j$$

$$= -r_i^t (1-y_i^t) x_j + \sum_{i \neq k}^{C} r_k y_i^t \cdot x_j$$

$$= \left[-r_i^t + r_i^t y_i^t + \sum_{i \neq k}^{C} r_k y_i^t \right] x_j$$

$$= \left[-r_i^t + \sum_{k=1}^{C} r_k^t \cdot y_i^t \right] x_j$$

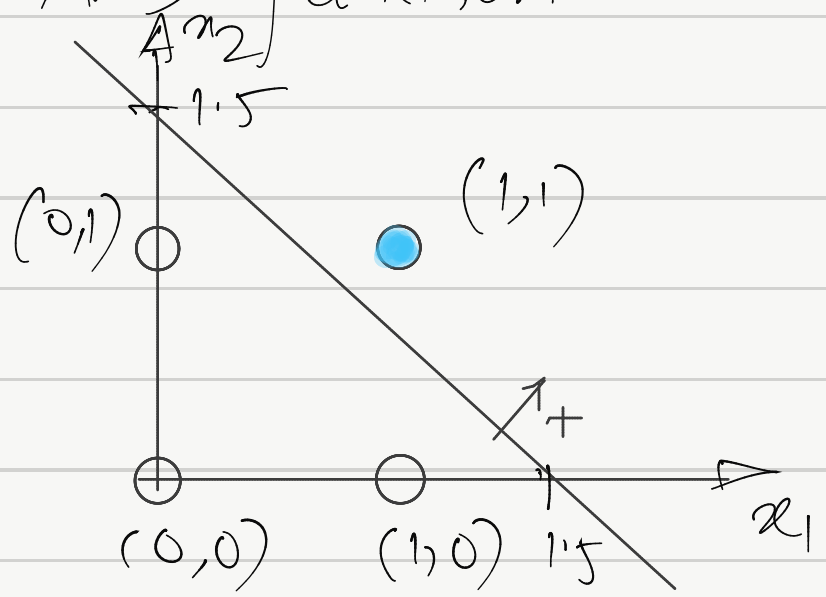
$$= \left(y_i^t - r_i^t \right) x_j \qquad \sum_{k=1}^{C} r_k^t = 1$$

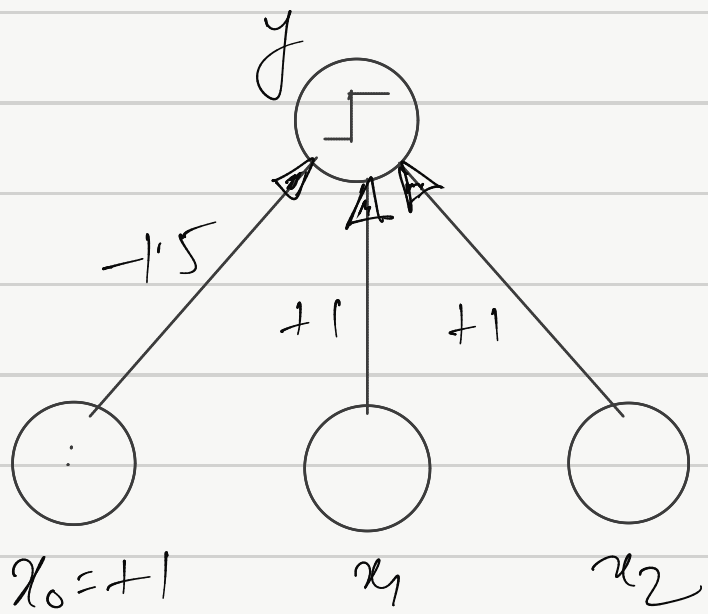
→ one-hot vector

Learning Boolean Function:

Perceptron to model AND function

x_1	x_2	r
0	0	0
0	1	0
1	0	0
1	1	1





$$y = s(x_1 + x_2 - 1.5)$$

$x_1 = 0,\ x_2 = 1 \Rightarrow y = s(-0.5)$
$\Rightarrow$ -ve class

$x_1 = 1,\ x_2 = 0 \Rightarrow y = s(-0.5)$
$\Rightarrow$ -ve class

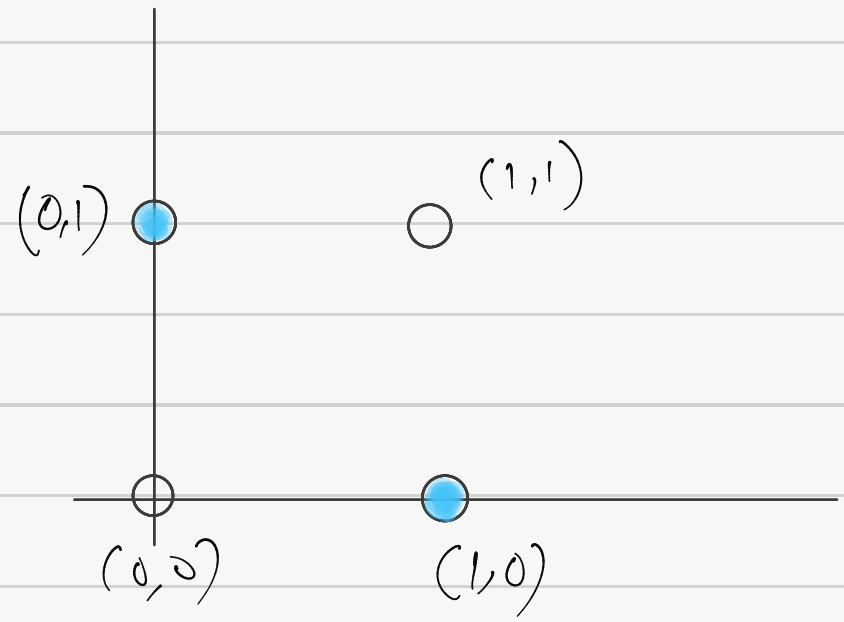
$x_1 = 1,\ x_2 = 1 \Rightarrow y = s(0.5)$
$\Rightarrow$ +ve class

Similarly, we can design a perceptron for learning Boolean OR function.

However, Single layer perceptron fails to model XOR function as XOR is not linearly separable.

XOR

x_1	x_2	r
0	0	0
0	1	1
1	0	1
1	1	0



we cannot draw a line where the empty circles are on one side while the shaded circles are on the other side of the line.

Mathematically:

○ -ve examples

● +ve examples

w_0, w_1 and w_2 should satisfy the following:

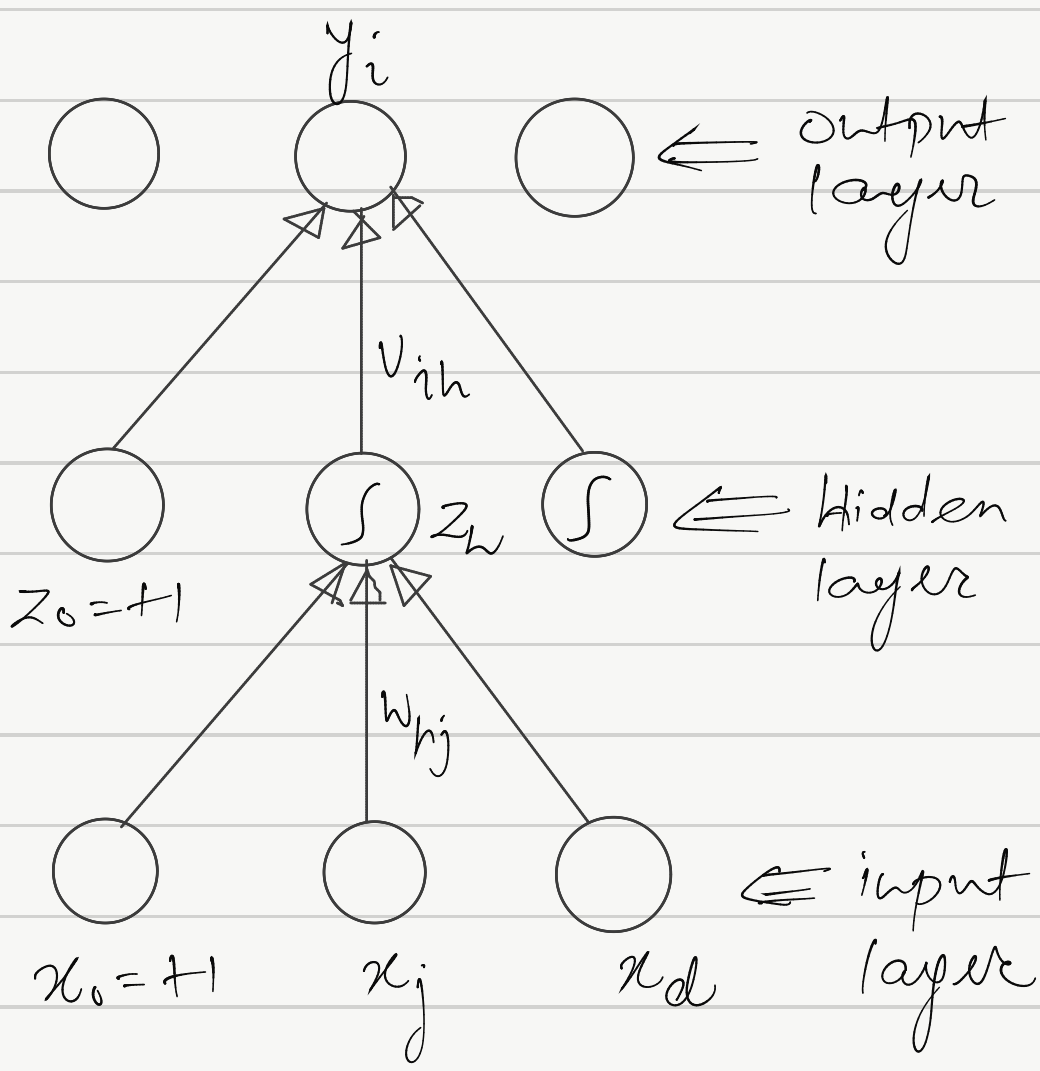
$$w_0 \leq 0$$
$$w_2 + w_0 > 0$$
$$w_1 + w_0 > 0$$
$$w_2 + w_1 + w_0 \leq 0$$

But there are no way to satisfy the above inequalities

* A perceptron that has a single layer of weights can only model linear functions of inputs but will fail to learn functions that require non-linear discriminators (like XOR)

Learning non-linear functions with MLP:

MLP (stacking of multiple layers of perceptrons) may overcome the limitation of single layer perceptron by implementing non-linear discriminants (for classification) and approximating non-linear function of input (for regression)



⇒ x is fed to the input layer

⇒ Activation propagates in the forward direction

⇒ values of the hidden units (z_h) computed

⇒ Compute output layer value y_i from the hidden layer values.

$$z_h = \text{Sigmoid}(w_h^T x) = \frac{1}{1 + \exp\left[-\left(\sum_{j=1}^{d} w_{hj} x_j + w_{h0}\right)\right]}$$

$$w_h \in \mathbb{R}^{(d+1)\times 1}$$

$$h = 1, 2, \ldots, H$$

$$y_i = v_i^T z = \sum_{h=1}^{H} v_{ih} z_h + v_{i0} \qquad v_i \in \mathbb{R}^{H}$$

(scalar)

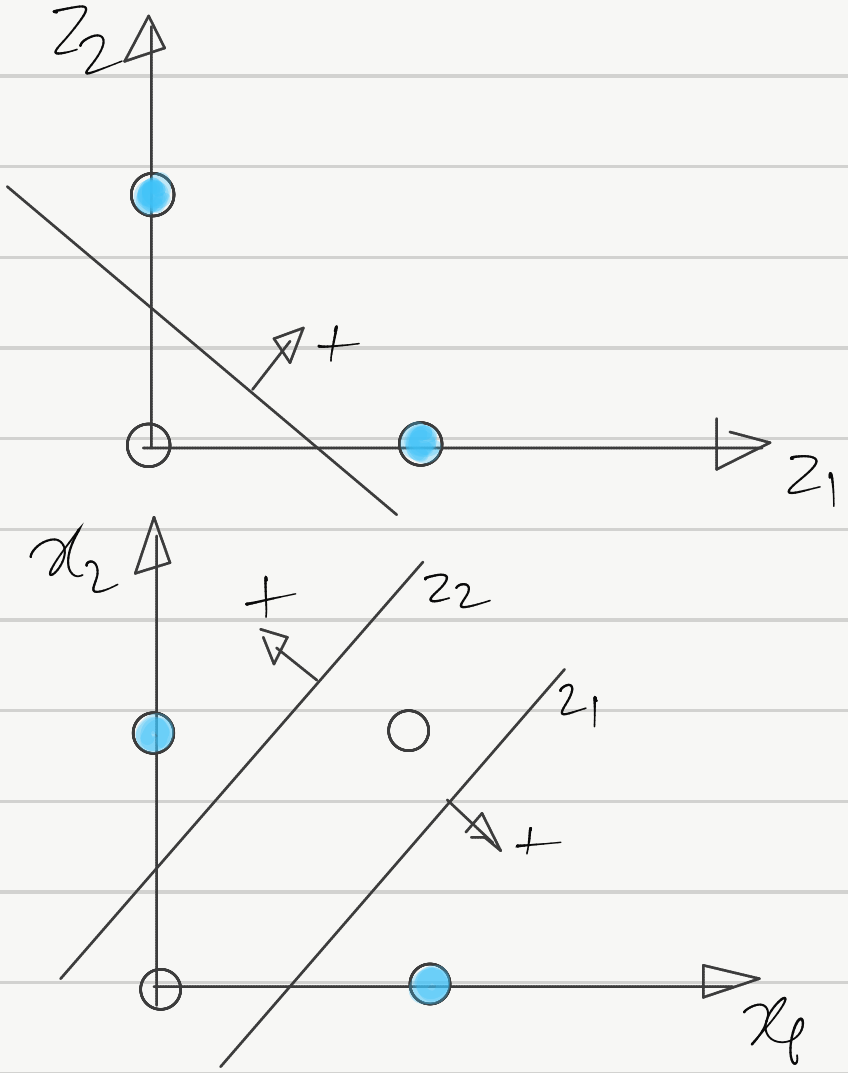
$$W \in \mathbb{R}^{H\times(d+1)} \qquad V \in \mathbb{R}^{K\times(H+1)}$$

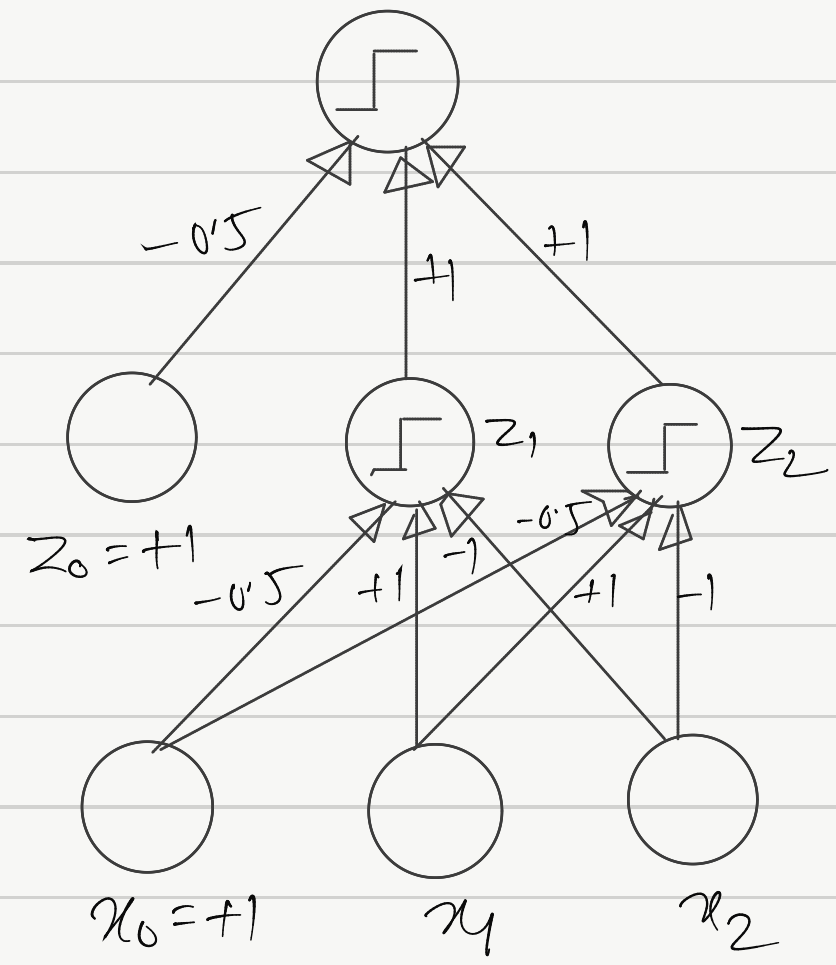
How MLP solves the XOR problem?

$$x_1 \otimes x_2 = \underbrace{(x_1 \wedge \bar{x}_2)}_{z_1} \vee \underbrace{(\bar{x}_1 \wedge x_2)}_{z_2}$$

x_1	x_2	z_1	z_2	y
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	0	0	0

Point 2 <----- Point 3 <----- Map to a single point (Point 1)





$$z_1 = S(x_1 - x_2 - 0.5)$$
$$z_2 = S(-x_1 + x_2 - 0.5)$$
$$y = S(z_1 + z_2 - 0.5)$$

Points (0,0) and (1,1) are mapped to (0,0) in Z-space, allowing linear separability in Z-space.

Strategy:

⇒ For every input combination where the output is 1, we define a hidden unit

↳ checks conjunction of inputs

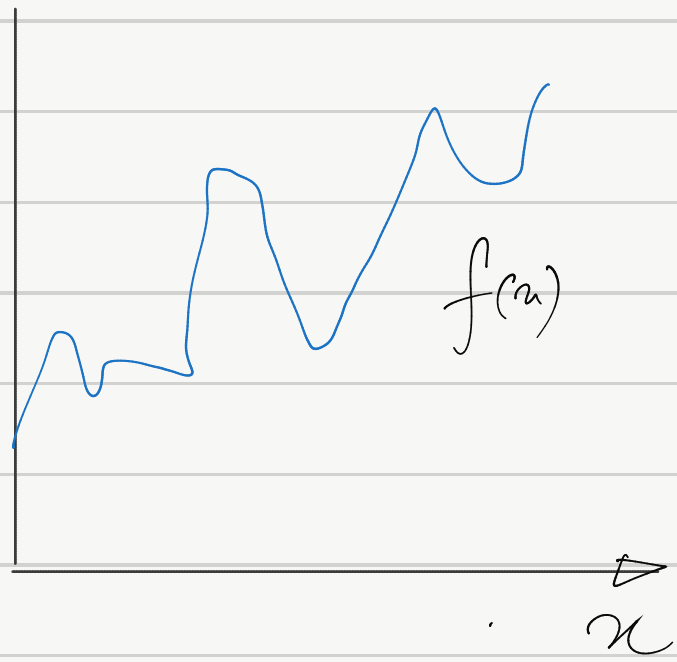
⇒ The output layer implements a disjunction.

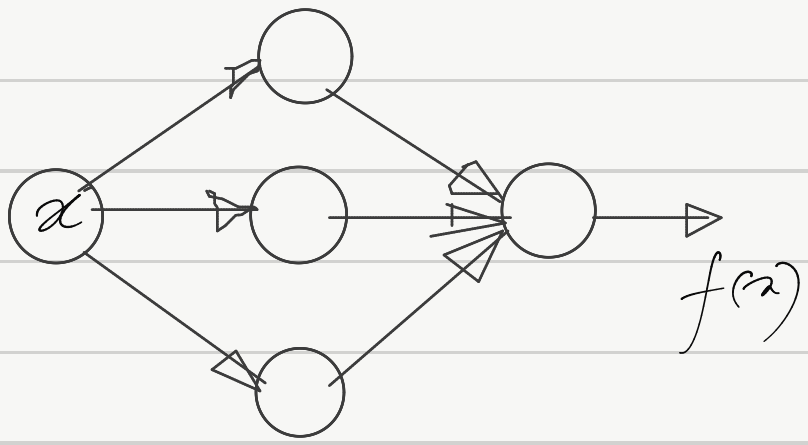
** This is just an existence proof.. Such networks are practically infeasible

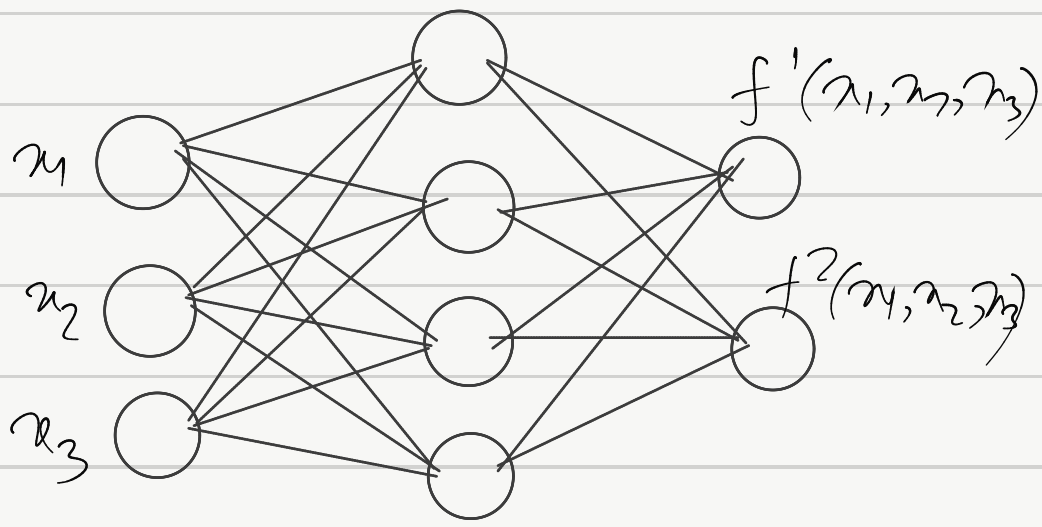
⇒ for d size input no of hidden units $O(2^d)$

Universal Approximation

An MLP can learn any function







The universal approximation theorem is true even for a very simple single layer MLP with only one hidden layer.

Two caveats:

1) A neural network can compute any function (But not exactly)
   ⇒ It can "approximate" with an arbitrary accuracy
   ⇒ By increasing the number of neurons in the hidden layer, we can increase accuracy.

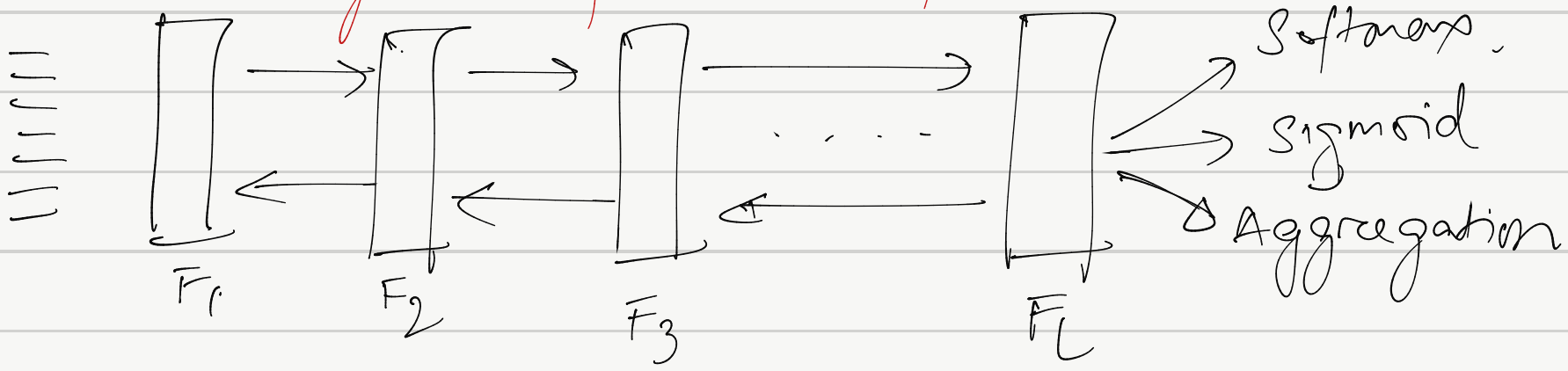
   We would like to compute a function $f(x)$ within some bound on error $\epsilon > 0$. It is guaranteed that using enough hidden neurons we can always find a neural network with output $g(x)$ that satisfies $|g(x) - f(x)| < \epsilon$

2) Class of functions which can be approximated are continous functions.

For details:
neuralnetworksanddeeplearning.com/chap4.html

Functional view of MLP

Each layer transforms a representation into another



$$F_N = F_L(F_{L-1} \cdots (F_3(F_2(F_1(input))))$$

$$E = f(F_N)$$

$$\frac{\partial E}{\partial w} \Rightarrow \frac{\partial f(F_N)}{\partial w_i} \Rightarrow \frac{\partial E}{\partial F_L} \cdot \frac{\partial F_L}{\partial F_{L-1}} \cdots \frac{\partial F_2}{\partial F_1} \cdot \frac{\partial F_1}{\partial w}$$

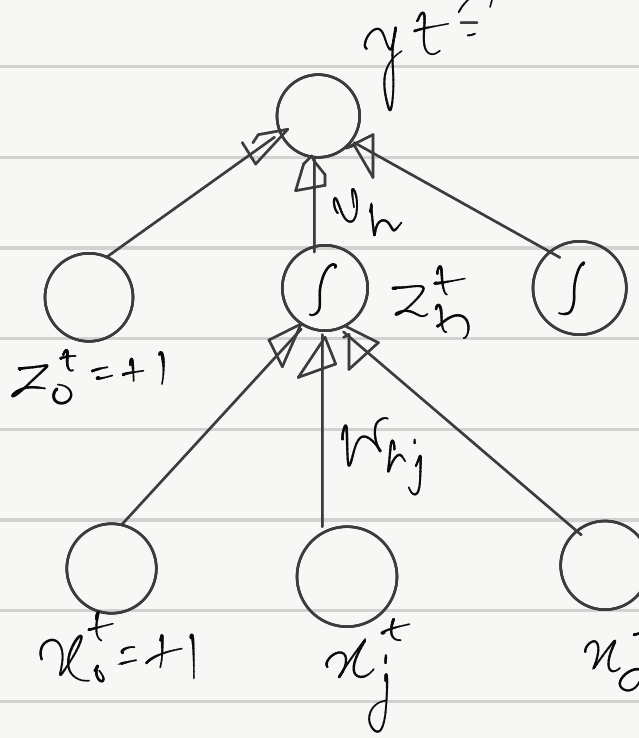
Learning strategy:

- **Online learning:**
  - ⇒ Process one data at a time
  - ⇒ update the weights
  - ⇒ Epoch: iteration that uses all the training data points

- **Batch learning:**
  - ⇒ Process one data point at a time
  - ⇒ update the weights after all datapoints were processed.
  - ⇒ Epoch: a complete batch

- **Minibatch learning:**
  - ⇒ Update the weights after processing a subset of datapoints

Regression With multilayer Perceptron:



$i \Rightarrow$ index of output layer nodes

$h \Rightarrow$ index of the hidden layer nodes

$j \Rightarrow$ index of the input layer nodes.

This is a batch or mini batch - based update $\Leftarrow$

$$E(W, v \mid X) = \frac{1}{2} \sum_t (r^t - y^t)^2$$

FORWARD

$$y^t = v^T z^t = \sum_{h=0}^{H} v_h z_h^t$$

$$z_h^t = \text{Sigmoid}(w_h^T x^t)$$

w_h

x

$$v_h = v_h + \Delta v_h$$

$$w_{hj} = w_{hj} + \Delta w_{hj}$$

BACKWARD

$$\Delta v_h = -\eta \frac{\partial E}{\partial v_h} = \eta \sum_t (r^t - y^t) z_h^t$$

$$\Delta w_{hj} = -\eta \frac{\partial E}{\partial w_{hj}}$$

$$= -\eta \sum_t \frac{\partial E}{\partial y^t} \cdot \frac{\partial y^t}{\partial z_h^t} \cdot \frac{\partial z_h^t}{\partial w_h^t} \cdot \frac{\partial w_h^t}{\partial w_{hj}}$$

$$= -\eta \sum_t -(r^t - y^t) \cdot v_h \cdot z_h^t (1 - z_h^t) x_j^t$$

$$= \eta \sum_t (r^t - y^t) v_h \cdot z_h^t (1 - z_h^t) x_j^t$$

Backpropagation

Algorithm : Backpropagation for Perceptron

Initialize all v_{ih} and w_{hj}

Repeat

For all $(x^t, r^t) \in \mathcal{X}$ in random order

For $h = 1, \ldots H$

$$z_h = \text{sigmoid}(w_h^T x^t)$$

$w_h \in \mathbb{R}^{H \times (d+1)}$, $x^t \in \mathbb{R}^{(d+1)\times 1}$

Scalar

$$y^t = v^T z$$

$v \in \mathbb{R}^{(H+1)\times 1}$, $z \in \mathbb{R}^{(H+1)\times 1}$

$$\Delta v_h = \eta (r^t - y^t) z_h^t$$

For $h = 1, \ldots H$

$$\Delta w_h = \eta (r^t - y^t) . v_h z_h^t (1 - z_h^t) x^t$$

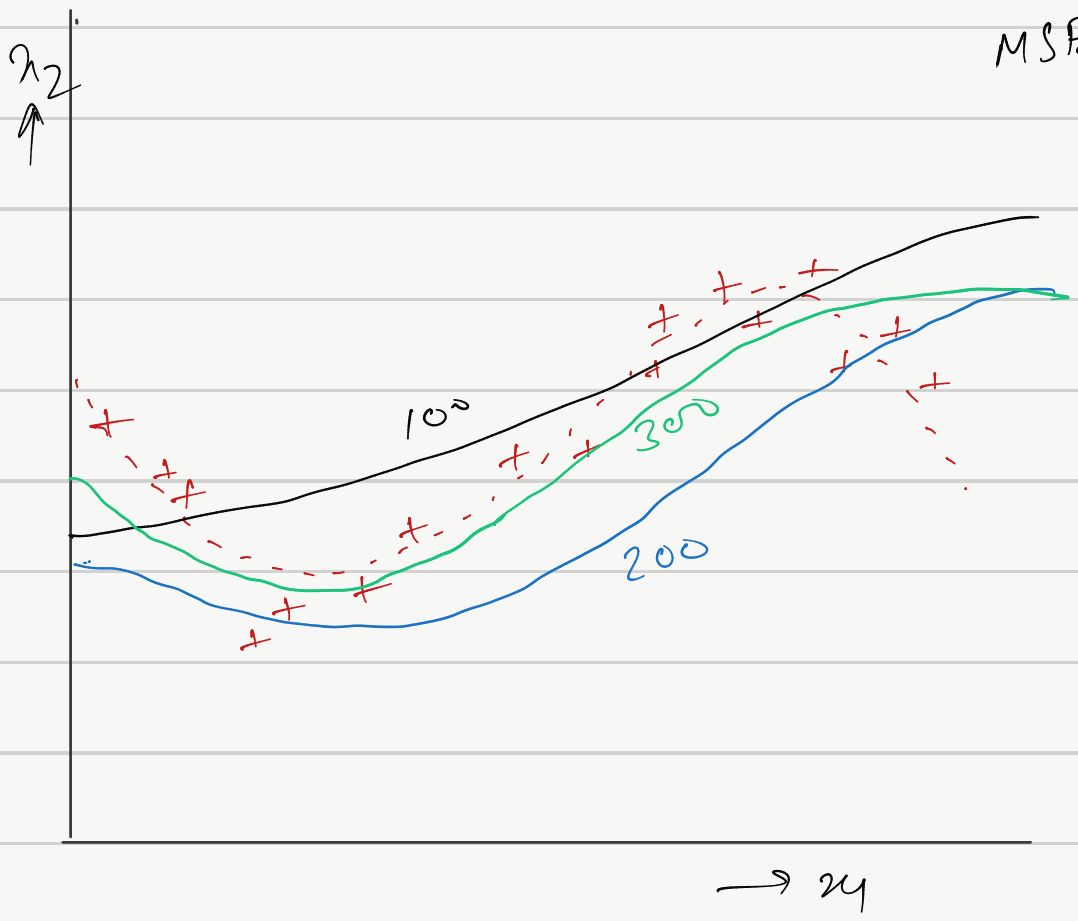
$\Delta w_h \in \mathbb{R}^{(d+1)\times 1}$, $x^t \in \mathbb{R}^{(d+1)\times 1}$

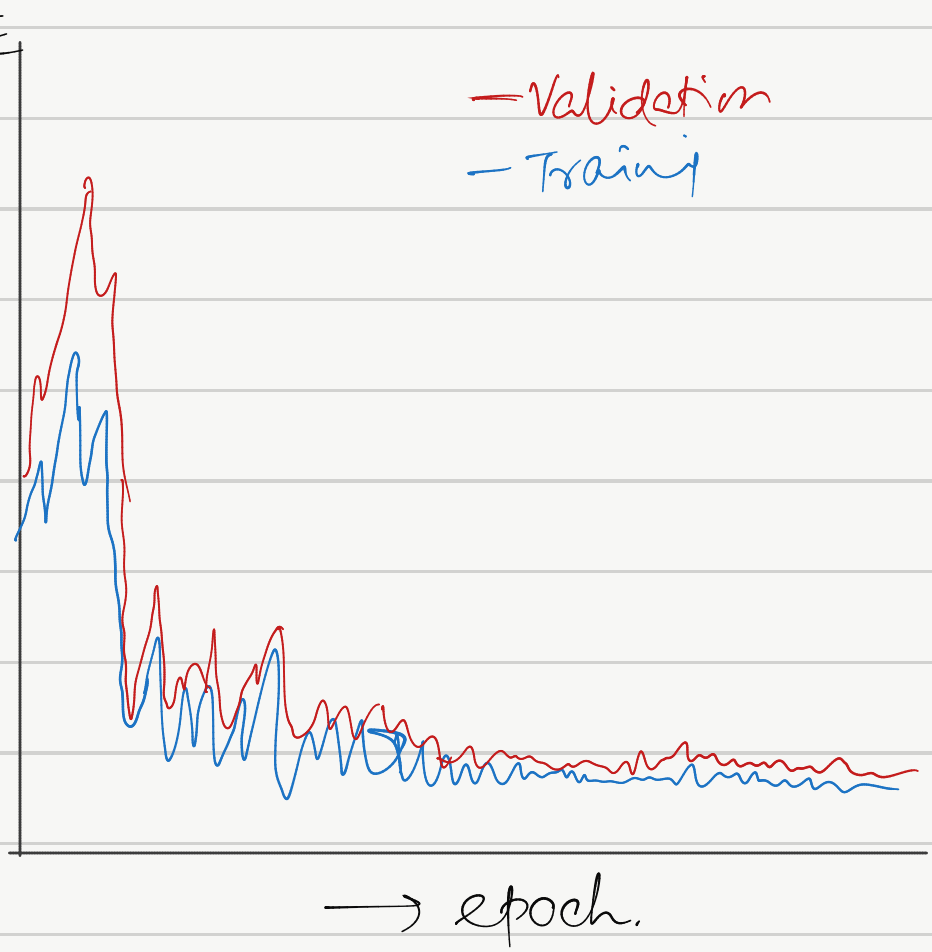
For $h = 1, \ldots H$

$w_h \in \mathbb{R}^{(d+1)\times 1}$

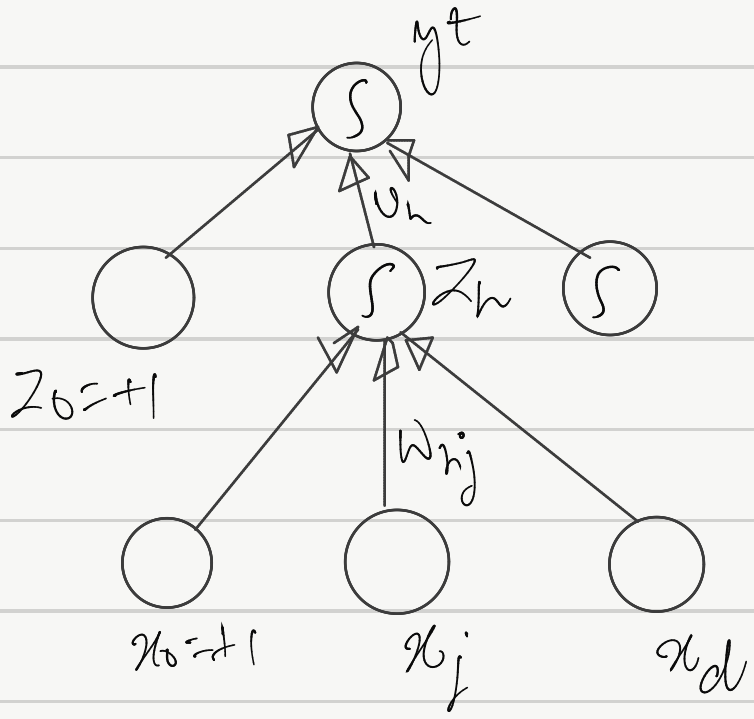
$$w_h = w_h + \Delta w_h$$

Until convergence





Two class Discrimination :



$$E(W,V|X) = -\left[\sum_t r^t \log y^t + (1-r^t)\log(1-y^t)\right]$$

o^t

$$y^t = \text{Sigmoid}(V^T z)$$

p^t

$$z_h^t = \text{Sigmoid}(w_h^T x^t)$$

x^t

$$\Delta v_h = \eta \sum_t (r^t - y^t) z_h^t$$

$$\Delta w_{hj} = \eta \sum_t \frac{\partial E}{\partial y^t} \cdot \frac{\partial y^t}{\partial z_h^t} \cdot \frac{\partial z_h^t}{\partial x^t}$$

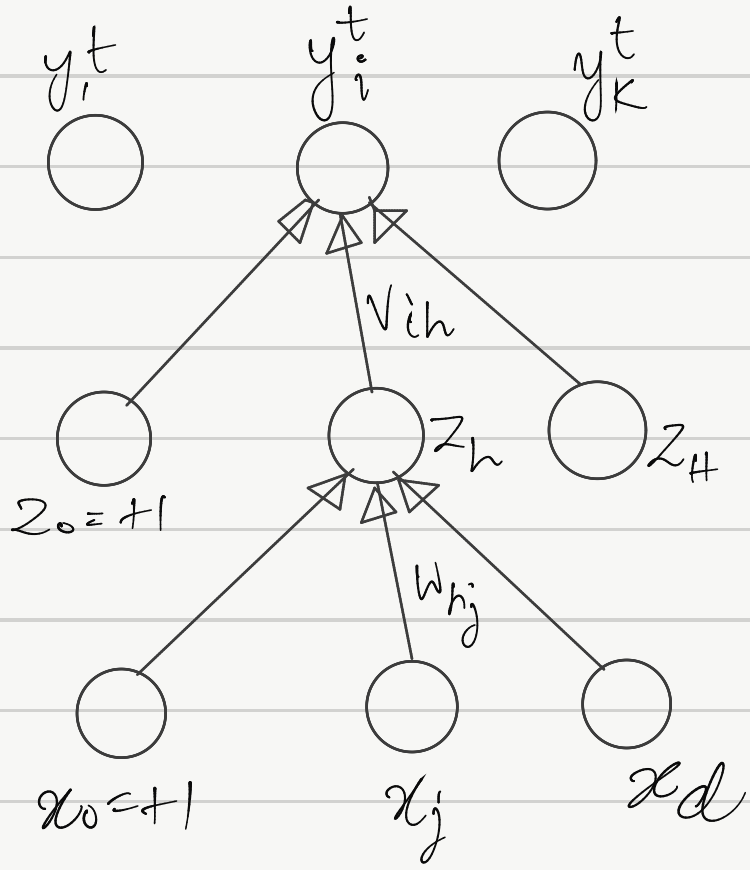
$$= \eta \sum_t (r^t - y^t) v_h z_h^t (1 - z_h^t) x_j^t$$

$$\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y^t} \cdot \frac{\partial y^t}{\partial o^t} \cdot \frac{\partial o^t}{\partial z_h^t} \cdot \frac{\partial z_h^t}{\partial p^t} \cdot \frac{\partial p^t}{\partial w_{hj}}$$

$$= -\left[\frac{r^t}{y^t} - \frac{(1-r^t)}{1-y^t}\right] \cdot y^t(1-y^t) \cdot v_h \cdot z_h^t(1-z_h^t) \cdot x_j^t$$

$$= -(r^t - y^t) \cdot v_h z_h^t (1 - z_h^t) x_j^t$$

Multiclass Discrimination :



$$E(W, V \mid X) = -\sum_t \sum_i r_i^t \log y_i^t$$

$$y_i^t = \frac{\exp(o_i^t)}{\sum_k \exp(o_k^t)}$$

$$o_i^t = v_i^t z_h^t$$

$$z_h^t = \text{Sigmoid}(w_h^T x_j^t)$$

$$x_j^t$$

$$\Delta v_{ih} = \eta \sum_t (r_i^t - y_i^t) z_h^t$$

$$\Delta w_{hj} = \eta \sum_t \left[\sum_i (r_i^t - y_i^t) v_{ih} \right] z_h^t (1 - z_h^t) x_j^t$$

Deep Learning

Universal approximation theorem says any function can be approximately computed by a neural network.

One may think about "long and thin" network or "short and fat" network to model the functions. It has been empirically observed that "long and thin" (deep) networks have fewer parameters and better generalization ability than the shallow networks.

Deep Neural Networks:

⇒ many hidden layers

⇒ each hidden layer combines the values in its preceding layer

⇒ learn complex functions.

⇒ Successive hidden layers correspond to more abstract representations.

Advantages:

⇒ learn features of increasing abstraction

⇒ less human effort in selecting useful features for a given application.

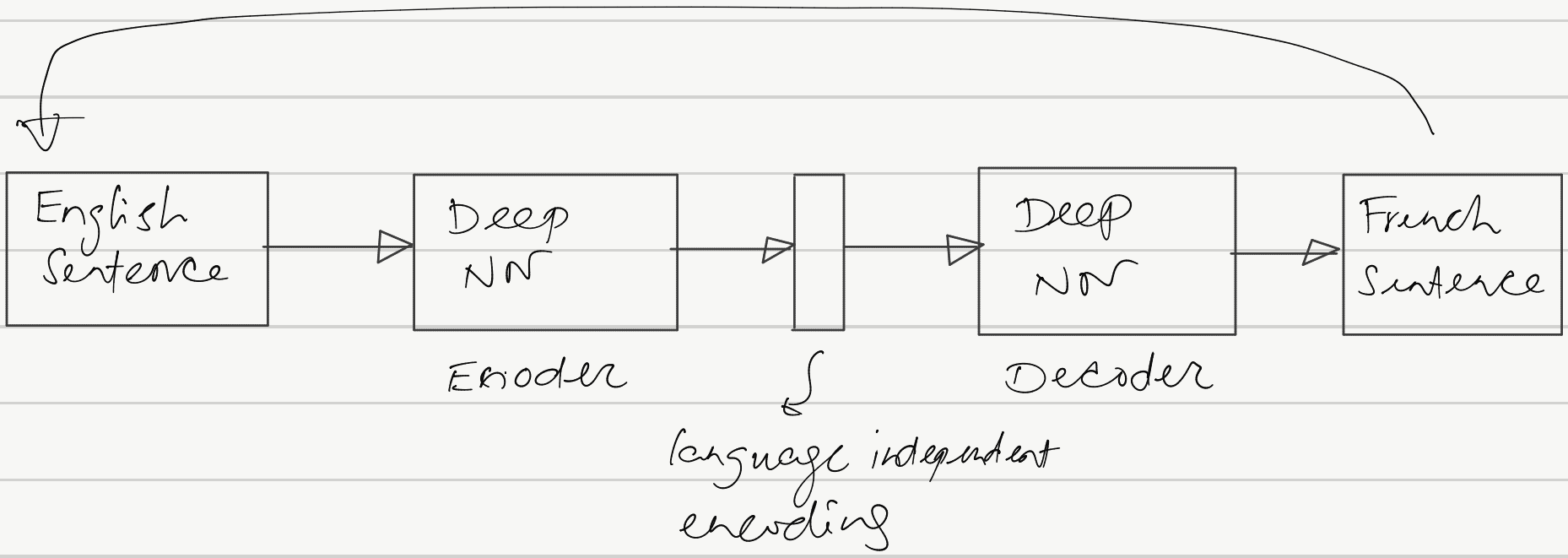
Issues:

⇒ Deep neural networks have large number of weights (parameters), processing units

⇒ Needs more memory and computation

⇒ Training a deep NN with back propagation needs multiplying derivatives
⇒ Vanishing and exploding gradients.

Example: Machine Translation

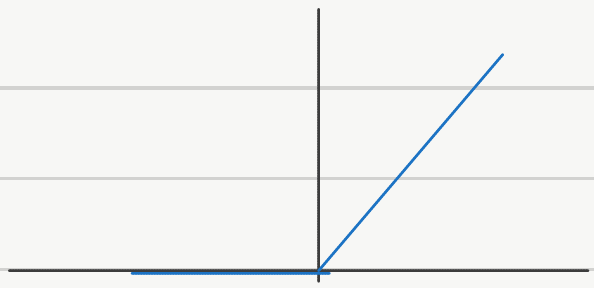


End-to-end Training:

⇒ Training data → pairs of sentences

⇒ Internal representations are learned automatically.

Rectified Linear Unit (ReLU):



$$ReLU(a) = \begin{cases} a & \text{if } a > 0 \\ 0, & \text{otherwise} \end{cases}$$

⇒ Not differentiable at $a = 0$

$$ReLU'(a) = \begin{cases} 1 & \text{if } a > 1 \\ 0 & \text{otherwise} \end{cases}$$

Advantages of ReLU:

⇒ Does not saturate like Sigmoid or tanh.
⇒ updates can be done for large +ve a.

⇒ For some inputs, some hidden unit activations will be zero
⇒ Sparse representation, leading to faster training.

Disadvantage:

⇒ Derivative is zero for $a \leq 0$
⇒ No further training.

$$\text{leaky-ReLU}(a) = \begin{cases} a & \text{if } a > 0 \\ \alpha a & \text{otherwise} \end{cases}$$