# Introduction to Neural Networks and Deep Learning

## Multilayer Perceptron

Andres Mendez-Vazquez

September 28, 2019

# Outline



Cinvestav

# Do you remember?

**The Perceptron has the following problem**

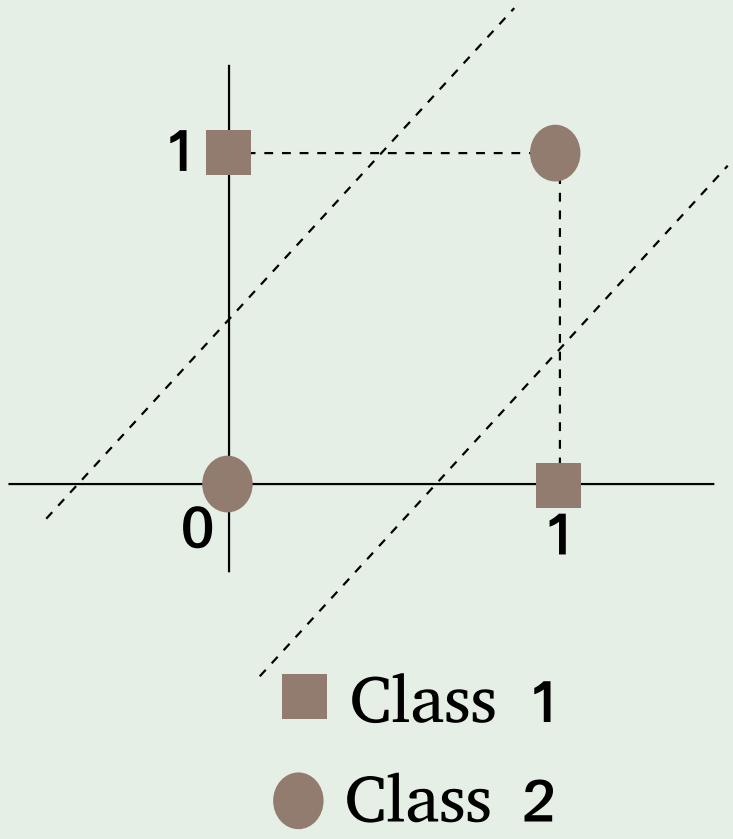
Given that the perceptron is a linear classifier

**It is clear that**

It will never be able to classify stuff that is not linearly separable

# Example: XOR Problem

## The Problem




Cinvestav

# The Perceptron cannot solve it

**Because**

The perceptron is a linear classifier!!!

**Thus**

Something needs to be done!!!

**Maybe**

Add an extra layer!!!

# A little bit of history

## It was first cited by Vapnik

Vapnik cites (Bryson, A.E.; W.F. Denham; S.E. Dreyfus. Optimal programming problems with inequality constraints. I: Necessary conditions for extremal solutions. AIAA J. 1, 11 (1963) 2544-2550 [1]) as the first publication of the backpropagation algorithm in his book "Support Vector Machines."

## It was first used by

Arthur E. Bryson and Yu-Chi Ho described it as a multi-stage dynamic system optimization method in 1969.

## However

It was not until 1974 and later, when applied in the context of neural networks and through the work of Paul Werbos, David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams that it gained recognition [2, 3, 4].

# Then

## Something Notable

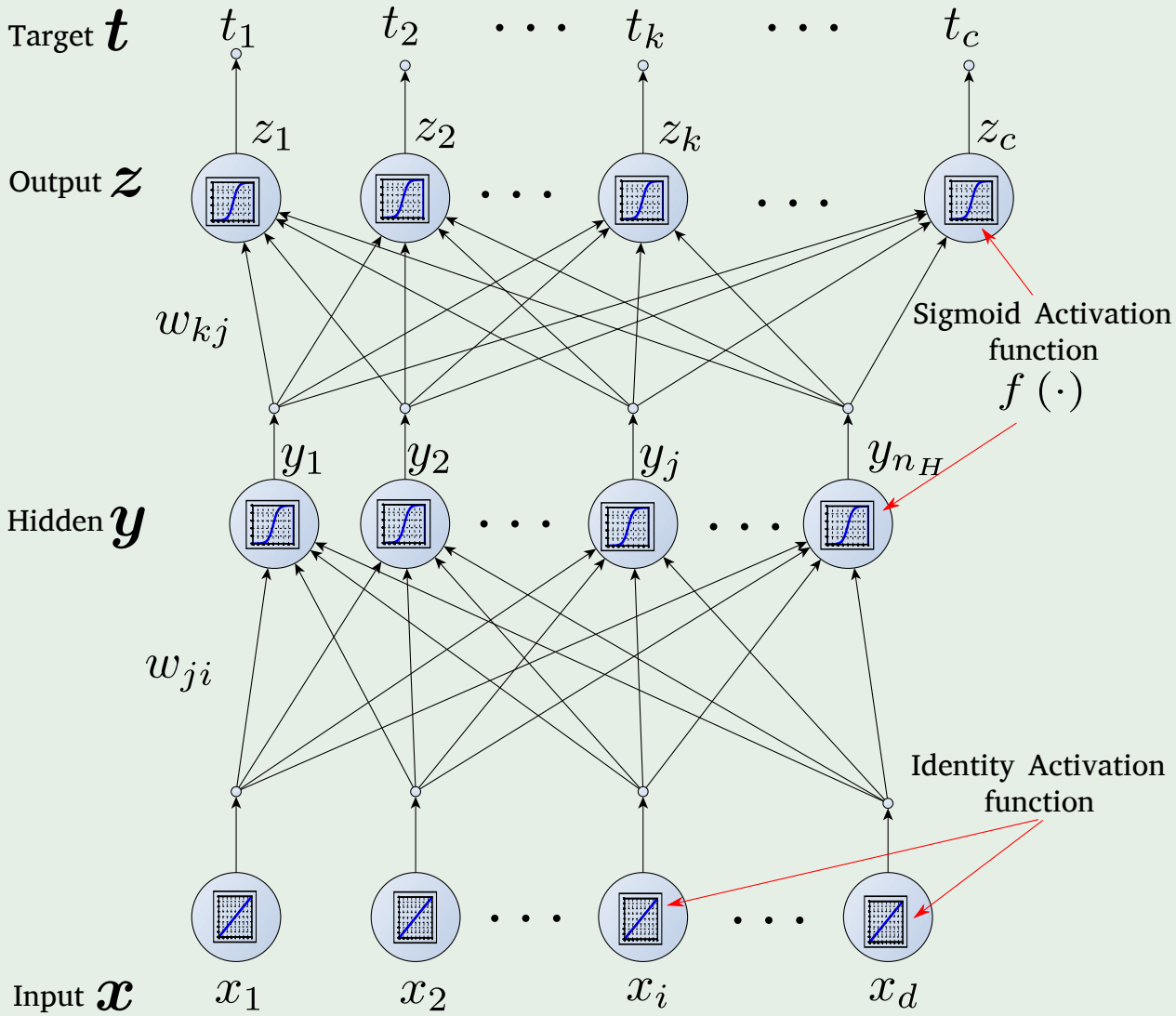
It led to a "renaissance" in the field of artificial neural network research.

## Nevertheless

During the 2000s it fell out of favor but has returned again in the 2010s, now able to train much larger networks using huge modern computing power such as GPUs.

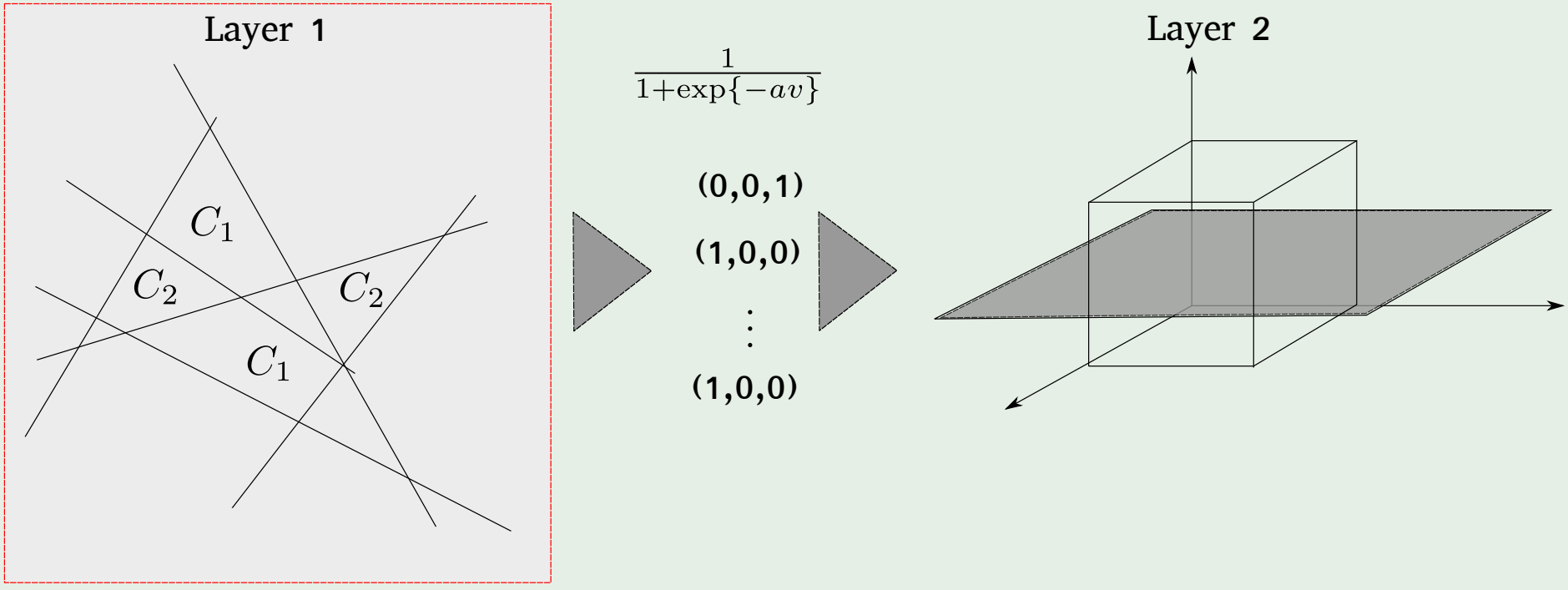
# Multi-Layer Perceptron (MLP) [5]

## Multi-Layer Architecture·



# What do we believe happens in the MLP

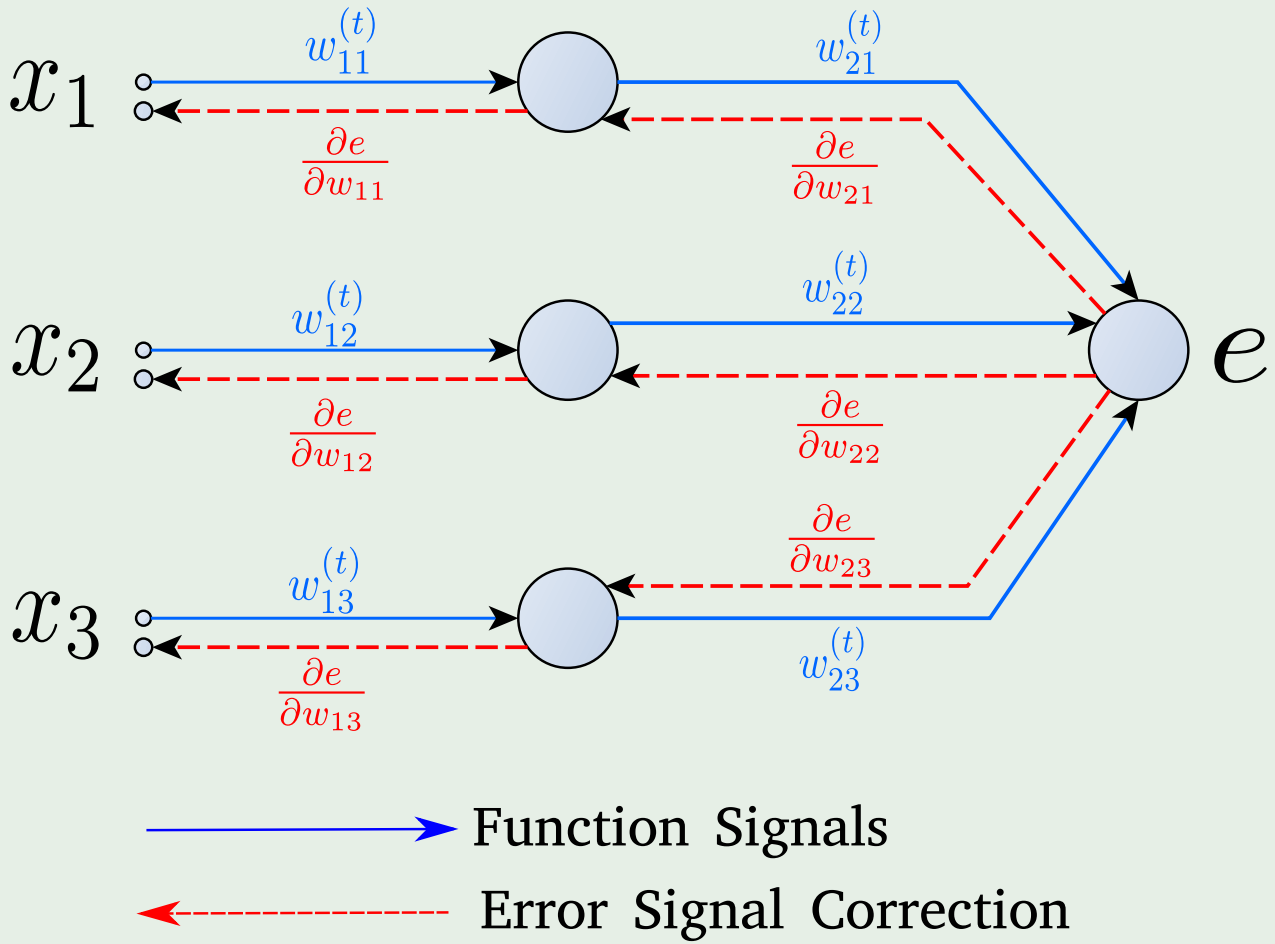
## We have something like this




Cinvestav

# Information Flow

## We have the following information flow



# Based in the Previous Idea

## People noticed that you require a way to get the info

- In order to build the error signal...

## This can be done by simply

- Evaluating the function composition to get the error

$$\boldsymbol{x} \longrightarrow f_n \circ f_{n-1} \circ \cdots f_1(\boldsymbol{x}) = y \longrightarrow e = t - y$$

Cinvestav

# Now, if we want to use the gradient descent

## We have a small problem

- If you have the derivative of the cost function by weights that are deep into the network from the cost function

$$\frac{\partial F(\boldsymbol{x})}{\partial w} = \frac{\partial \left(t - f_n \circ f_{n-1} \circ \cdots f_1(\boldsymbol{x})\right)^2}{\partial w}$$

## Therefore, we need to use the chain rule of derivatives

- To reach those functions to build the gradient descent for weights deep into the network

Cinvestav

# Chain Rule

## Definition

- Given a composition of differentiable functions

$$F(\boldsymbol{x}) = f_n \circ f_{n-1} \circ \cdots f_1(\boldsymbol{x})$$

Then

$$\frac{\partial F(\boldsymbol{x})}{\partial \boldsymbol{x}} = \frac{\partial f_n}{\partial f_{n-1}} \times \frac{\partial f_{n-1}}{\partial f_{n-2}} \times \frac{\partial f_{n-2}}{\partial f_{n-3}} \times \cdots \times \frac{\partial f_1}{\partial \boldsymbol{x}}$$

Cinvestav

# Therefore

## If we derivate with respect to weights at each level, we are actually back-propagating the error

$$J(\boldsymbol{W}) = \frac{1}{2}(t-z)^2$$
$$z = f\left(\boldsymbol{w}_1^T \boldsymbol{y}\right)$$
$$\boldsymbol{y} = g\left(\boldsymbol{w}_2^T \boldsymbol{x}\right)$$

## Therefore, for the first layer

$$\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{w}_1} = \frac{\partial J(\boldsymbol{W})}{\partial z} \times \frac{\partial f\left(\boldsymbol{w}_1^T \boldsymbol{y}\right)}{\partial \boldsymbol{w}_1^T \boldsymbol{y}} \times \frac{\partial \boldsymbol{w}_1^T \boldsymbol{y}}{\partial \boldsymbol{w}_1}$$

## What about the second layer?

- We go to the blackboard!!!

# The Quadratic Learning Error Function

## Cost Function our well know error at **pattern** $m$

$$J(m) = \frac{1}{2} e_k^2(m) \tag{1}$$

## Delta Rule or Widrow-Hoff Rule

$$\Delta w_{kj}(m) = -\eta e_k(m) x_j(m) \tag{2}$$

## Actually this is know as Gradient Descent

$$w_{kj}(m+1) = w_{kj}(m) + \Delta w_{kj}(m) \tag{3}$$

Cinvestav

# Back-propagation

## Setup

Let $t_k$ be the $k$-th target (or desired) output and $z_k$ be the $k$-th computed output with $k = 1, \ldots, d$ and $\boldsymbol{w}$ represents all the weights of the network

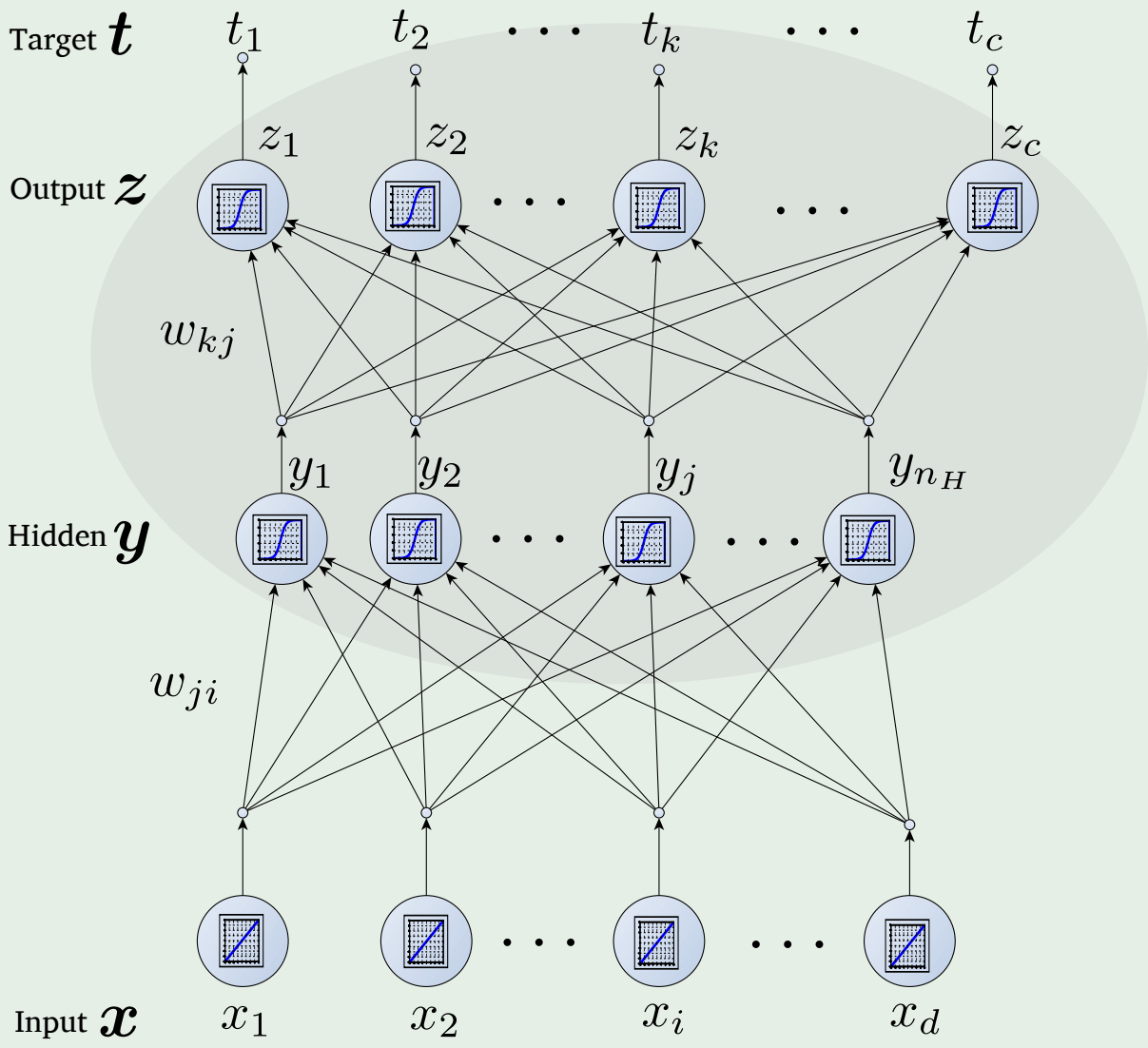
## Training Error for a single Pattern or Sample!!!

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2}\|\boldsymbol{t} - \boldsymbol{z}\|^2 \tag{4}$$

Cinvestav

# Multilayer Architecture

## Multilayer Architecture: hidden–to-output weights



# Observation about the activation function

## Hidden Output is equal to

$$y_j = f\left(\sum_{i=1}^{d} w_{ji}x_i\right)$$

## Output is equal to

$$z_k = f\left(\sum_{j=1}^{y_{n_H}} w_{kj}y_j\right)$$

Cinvestav

# Hidden–to-Output Weights

## $net_k$

- It describes how the overall error changes with the activation of the unit's net:

$$net_k = \sum_{j=1}^{y_{n_H}} w_{kj} y_j = \boldsymbol{w}_k^T \cdot \boldsymbol{y} \tag{5}$$

## Which is composed with an activation function $f$

$$z_k = f\left(net_k\right) \tag{6}$$

## Thus

$$\frac{\partial z_k}{\partial net_k} = f'\left(net_k\right) \tag{7}$$

Cinvestav

# Now, we have the cost function $J$

## Thus, we can apply the chain rule to the cost function

$$\frac{\partial J(\boldsymbol{z}_k)}{\partial w_{kj}} = \frac{\partial J(\boldsymbol{z}_k)}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{kj}} = -\delta_k \cdot \frac{\partial net_k}{\partial w_{kj}} \tag{8}$$

## Still, we need to apply the same for $\frac{\partial J(\boldsymbol{z}_k)}{\partial net_k}$

$$\frac{\partial J(\boldsymbol{z}_k)}{\partial net_k} = \frac{\partial J(\boldsymbol{z}_k)}{\partial z_k} \cdot \frac{\partial z_k}{\partial net_k} = -(t_k - z_k) f'(net_k) \tag{9}$$

Cinvestav

# Then

## We create a new variable

$$\delta_k = -\frac{\partial J(\boldsymbol{z}_k)}{\partial net_k} = (t_k - z_k) f'(net_k)$$

## Not only that, but we need

$$\frac{\partial net_k}{\partial w_{kj}}$$

## Since $net_k = \boldsymbol{w}_k^T \cdot \boldsymbol{y}$ therefore:

$$\frac{\partial net_k}{\partial w_{kj}} = y_j \tag{10}$$

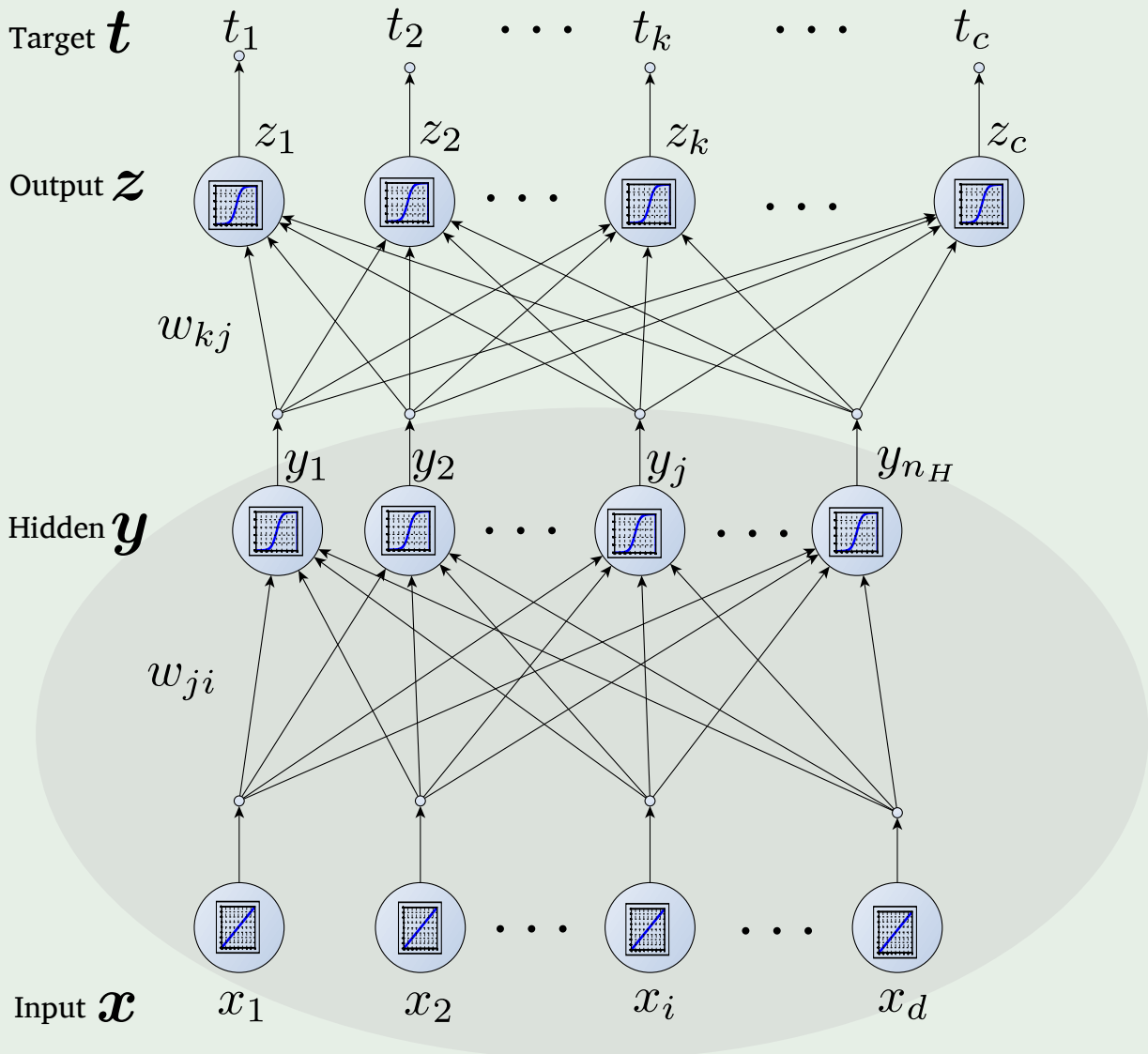
# Finally

**The weight update (or learning rule) for the hidden-to-output weights is:**

$$\triangle w_{kj} = \eta \delta_k y_j = \eta \left( t_k - z_k \right) f' \left( net_k \right) y_j \tag{11}$$

# Multi-Layer Architecture

## Going deeper into the network



# Input–to-Hidden Weights

## Chain rule on the Input–to-Hidden weights

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \cdot \frac{\partial y_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}} \tag{12}$$

## Thus

$$\begin{aligned}
\frac{\partial J}{\partial y_j} &= \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{c}\left(t_k - z_k\right)^2\right] \\
&= -\sum_{k=1}^{c}\left(t_k - z_k\right)\frac{\partial z_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}\left(t_k - z_k\right)\frac{\partial z_k}{\partial net_k}\cdot\frac{\partial net_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}\left(t_k - z_k\right)\frac{\partial f\left(net_k\right)}{\partial net_k}\cdot w_{kj}
\end{aligned}$$

# Input–to-Hidden Weights

## Finally

$$\frac{\partial J}{\partial y_j} = -\sum_{k=1}^{c} (t_k - z_k) f'(net_k) \cdot w_{kj} \tag{13}$$

## Remember

$$\delta_k = -\frac{\partial J}{\partial net_k} = (t_k - z_k) f'(net_k) \tag{14}$$

Cinvestav

# What is $\frac{\partial y_j}{\partial net_j}$?

## First

$$net_j = \sum_{i=1}^{d} w_{ji} x_i = \boldsymbol{w}_j^T \cdot \boldsymbol{x} \tag{15}$$

## Then

$$y_j = f\left(net_j\right)$$

## Then

$$\frac{\partial y_j}{\partial net_j} = \frac{\partial f\left(net_j\right)}{\partial net_j} = f'\left(net_j\right)$$

# Then, we can define $\delta_j$

**By defying the sensitivity for a hidden unit:**

$$\delta_j = f'(net_j) \sum_{k=1}^{c} w_{kj}\delta_k \tag{16}$$

**Which means that:**

"The sensitivity at a hidden unit is simply the sum of the individual sensitivities at the output units weighted by the **hidden-to-output weights** $w_{kj}$; all multiplied by $f'(net_j)$"

Cinvestav

# What about $\frac{\partial net_j}{\partial w_{ji}}$?

## We have that

$$\frac{\partial net_j}{\partial w_{ji}} = \frac{\partial \boldsymbol{w}_j^T \cdot \boldsymbol{x}}{\partial w_{ji}} = \frac{\partial \sum_{i=1}^{d} w_{ji} x_i}{\partial w_{ji}} = x_i$$

# Finally

**The learning rule for the input-to-hidden weights is:**

$$\Delta w_{ji} = \eta x_i \delta_j = \eta \left[ \sum_{k=1}^{c} w_{kj} \delta_k \right] f'(net_j) x_i \tag{17}$$

Cinvestav

# Basically, the entire training process has the following steps

## Initialization

Assuming that no prior information is available, pick the synaptic weights and thresholds

## Forward Computation

Compute the induced function signals of the network by proceeding forward through the network, layer by layer.

## Backward Computation

Compute the local gradients of the network.

## Finally

Adjust the weights!!!

Cinvestav

# However, you have a problem

## Problems with Hidden Layers

1. Increase complexity of Training
2. It is necessary to think about "Long and Narrow" network vs "Short and Fat" network.

## Intuition for a One Hidden Layer

1. For every input case of region, that region can be delimited by hyperplanes on all sides using hidden units on the first hidden layer.
2. A hidden unit in the second layer than ANDs them together to bound the region.

## Advantages

It has been proven that an MLP with one hidden layer can learn any nonlinear function of the input.

# Now, Calculating Total Change

## We have for that

The Total Training Error is the sum over the errors of $N$ individual patterns

## The Total Training Error

$$J = \sum_{p=1}^{N} J_p = \frac{1}{2} \sum_{p=1}^{N} \sum_{k=1}^{d} \left( t_k^p - z_k^p \right)^2 = \frac{1}{2} \sum_{p=1}^{n} \left\| \boldsymbol{t}^p - \boldsymbol{z}^p \right\|^2 \tag{18}$$

Cinvestav

# About the Total Training Error

## Remarks

- A weight update may reduce the error on the single pattern being presented but can increase the error on the full training set.
- However, given a large number of such individual updates, the total error of equation (18) decreases.

# Now, we want the training to stop

## Therefore

It is necessary to have a way to stop when the change of the weights is enough!!!

## A simple way to stop the training

- The algorithm terminates when the change in the criterion function $J(\boldsymbol{w})$ is smaller than some preset value $\Theta$.

$$\Delta J\left(\boldsymbol{w}\right) = \left|J\left(\boldsymbol{w}\left(t+1\right)\right) - J\left(\boldsymbol{w}\left(t\right)\right)\right| \tag{19}$$

- There are other stopping criteria that lead to better performance than this one.

Cinvestav

# Other Stopping Criteria

## Norm of the Gradient

The back-propagation algorithm is considered to have converged when the Euclidean norm of the gradient vector reaches a sufficiently small gradient threshold.

$$\|\nabla_{\boldsymbol{w}} J(m)\| < \Theta \tag{20}$$

## Rate of change in the average error per epoch

The back-propagation algorithm is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small.

$$\left| \frac{1}{N} \sum_{p=1}^{N} J_p \right| < \Theta \tag{21}$$

# About the Stopping Criteria

## Observations

1. Before training starts, the error on the training set is high.
   - Through the learning process, the error becomes smaller.
2. The error per pattern depends on the amount of training data and the expressive power (such as the number of weights) in the network.
3. The average error on an independent test set is always higher than on the training set, and it can decrease as well as increase.
4. A validation set is used in order to decide when to stop training.
   - We do not want to over-fit the network and decrease the power of the classifier generalization "we stop training at a minimum of the error on the validation set"

# Some More Terminology

## Epoch

As with other types of backpropagation, 'learning' is a supervised process that occurs with each cycle or 'epoch' through a forward activation flow of outputs, and the backwards error propagation of weight adjustments.

## In our case

I am using the batch sum of all correcting weights to define that epoch.

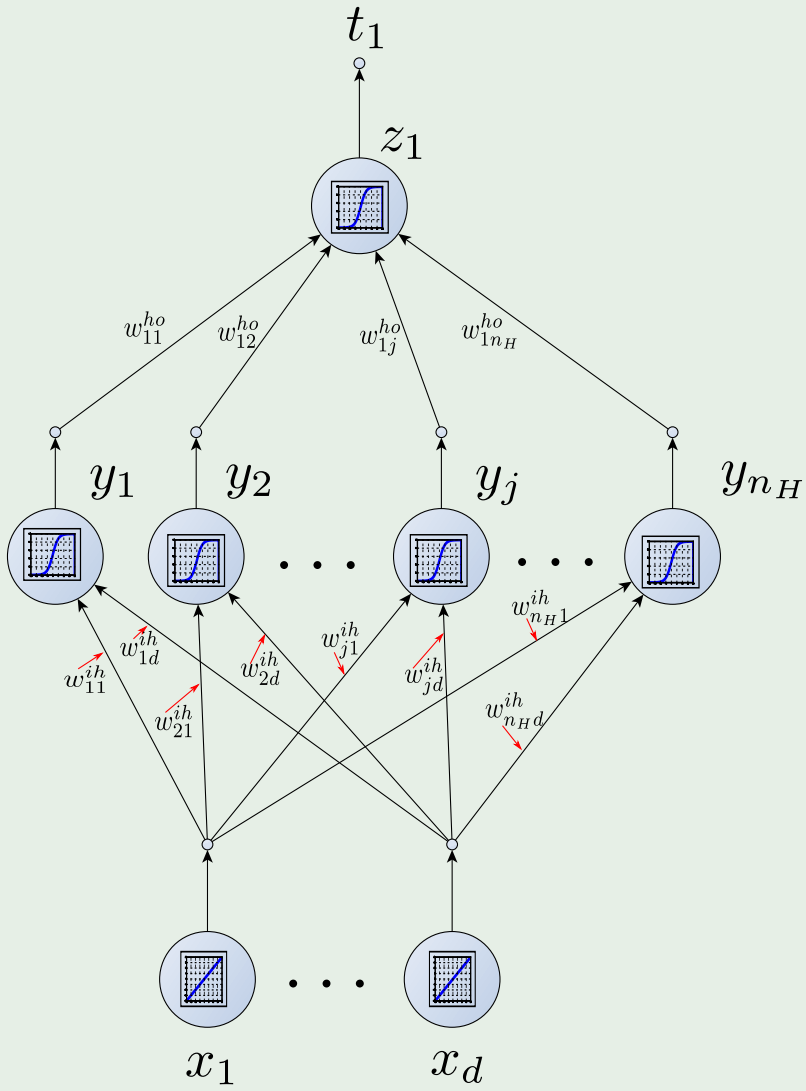
# Final Basic Batch Algorithm

**Perceptron**$(\boldsymbol{X})$

1. **Initialize random** $\boldsymbol{w}$, **number of hidden units** $n_H$, **number of outputs** $\boldsymbol{z}$, **stopping criterion** $\Theta$, **learning rate**$\eta$, **epoch** $m = 0$
2. **do**
3. $m = m + 1$
4. **for** $s = 1$ **to** $N$
5. $\boldsymbol{x}(m) = \boldsymbol{X}(:, s)$
6. **for** $k = 1$ **to** $c$
7. $\delta_k = (t_k - z_k) f'\left(\boldsymbol{w}_k^T \cdot \boldsymbol{y}\right)$
8. **for** $j = 1$ **to** $n_H$
9. $net_j = \boldsymbol{w}_j^T \cdot \boldsymbol{x}; y_j = f\left(net_j\right)$
10. $w_{kj}(m) = w_{kj}(m) + \eta \delta_k y_j(m)$
11. **for** $j = 1$ **to** $n_H$
12. $\delta_j = f'\left(net_j\right) \sum_{k=1}^{c} w_{kj} \delta_k$
13. **for** $i = 1$ **to** $d$
14. $w_{ji}(m) = w_{ji}(m) + \eta \delta_j x_i(m)$
15. **until** $\|\nabla_{\boldsymbol{w}} J(m)\| < \Theta$
16. **return** $\boldsymbol{w}(m)$

# Example of Architecture to be used

## Given the following Architecture and assuming $N$ samples



# Generating the output $z_k$

## Given the input

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_N \end{bmatrix} \tag{22}$$

## Where

$\boldsymbol{x}_i$ is a vector of features

$$\boldsymbol{x}_i = \begin{pmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{di} \end{pmatrix} \tag{23}$$

Cinvestav

# Therefore

## We must have the following matrix for the input to hidden inputs

$$\boldsymbol{W}_{IH} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_H 1} & w_{n_H 2} & \cdots & w_{n_H d} \end{pmatrix} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_{n_H}^T \end{pmatrix} \tag{24}$$

Given that $\boldsymbol{w}_j = \begin{pmatrix} w_{j1} \\ w_{j2} \\ \vdots \\ w_{jd} \end{pmatrix}$

## Thus

We can create the $\boldsymbol{net}_j$ for all the inputs by simply

$$\boldsymbol{net}_j = \boldsymbol{W}_{IH}\boldsymbol{X} = \begin{pmatrix} \boldsymbol{w}_1^T\boldsymbol{x}_1 & \boldsymbol{w}_1^T\boldsymbol{x}_2 & \cdots & \boldsymbol{w}_1^T\boldsymbol{x}_N \\ \boldsymbol{w}_2^T\boldsymbol{x}_1 & \boldsymbol{w}_2^T\boldsymbol{x}_2 & \cdots & \boldsymbol{w}_2^T\boldsymbol{x}_N \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{w}_{n_H}^T\boldsymbol{x}_1 & \boldsymbol{w}_{n_H}^T\boldsymbol{x}_2 & \cdots & \boldsymbol{w}_{n_H}^T\boldsymbol{x}_N \end{pmatrix} \tag{25}$$

# Now, we need to generate the $\boldsymbol{y}_k$

## We apply the activation function element by element in $\boldsymbol{net}_j$

$$\boldsymbol{y}_1 = \begin{pmatrix} f\left(\boldsymbol{w}_1^T\boldsymbol{x}_1\right) & f\left(\boldsymbol{w}_1^T\boldsymbol{x}_2\right) & \cdots & f\left(\boldsymbol{w}_1^T\boldsymbol{x}_N\right) \\ f\left(\boldsymbol{w}_2^T\boldsymbol{x}_1\right) & f\left(\boldsymbol{w}_2^T\boldsymbol{x}_2\right) & \cdots & f\left(\boldsymbol{w}_2^T\boldsymbol{x}_N\right) \\ \vdots & \vdots & \ddots & \vdots \\ f\left(\boldsymbol{w}_{n_H}^T\boldsymbol{x}_1\right) & f\left(\boldsymbol{w}_{n_H}^T\boldsymbol{x}_2\right) & \cdots & f\left(\boldsymbol{w}_{n_H}^T\boldsymbol{x}_N\right) \end{pmatrix} \tag{26}$$

## IMPORTANT about overflows!!!

- Be careful about the numeric stability of the activation function.
- I the case of python, we can use the ones provided by scipy.special

Cinvestav

# However, We can create a Sigmoid function

## It is possible to use the following pseudo-code

Sigmoid$(x)$

1. if try $\left\{ \frac{1.0}{1.0+\exp\{-\alpha x\}} \right\}$ catch $\{OVERFLOW\}$ ◁ I will use a
   ◁ try-and-catch to catch
   ◁ the overflow
2. if $x < 0$
3. return 0
4. else
5. return 1
6. else
7. return $\frac{1.0}{1.0+\exp\{-\alpha x\}}$ ◁ 1.0 refers to the floating point (Rationals
   ◁ trying to represent Reals)

Cinvestav

# For this, we get $\boldsymbol{net}_k$

## For this, we obtain the $\boldsymbol{W}_{HO}$

$$\boldsymbol{W}_{HO} = \begin{pmatrix} w_{11}^o & w_{12}^o & \cdots & w_{1n_H}^o \end{pmatrix} = \begin{pmatrix} \boldsymbol{w}_o^T \end{pmatrix} \tag{27}$$

## Thus

$$\boldsymbol{net}_k = \begin{pmatrix} w_{11}^o & w_{12}^o & \cdots & w_{1n_H}^o \end{pmatrix} \begin{pmatrix} f\left(\boldsymbol{w}_1^T \boldsymbol{x}_1\right) & f\left(\boldsymbol{w}_1^T \boldsymbol{x}_2\right) & \cdots & f\left(\boldsymbol{w}_1^T \boldsymbol{x}_N\right) \\ f\left(\boldsymbol{w}_2^T \boldsymbol{x}_1\right) & f\left(\boldsymbol{w}_2^T \boldsymbol{x}_2\right) & \cdots & f\left(\boldsymbol{w}_2^T \boldsymbol{x}_N\right) \\ \vdots & \vdots & \ddots & \vdots \\ \underbrace{f\left(\boldsymbol{w}_{n_H}^T \boldsymbol{x}_1\right)}_{\boldsymbol{y}_{k1}} & \underbrace{f\left(\boldsymbol{w}_{n_H}^T \boldsymbol{x}_2\right)}_{\boldsymbol{y}_{k2}} & \cdots & \underbrace{f\left(\boldsymbol{w}_{n_H}^T \boldsymbol{x}_N\right)}_{\boldsymbol{y}_{kN}} \end{pmatrix} \tag{28}$$

## In matrix notation

$$\boldsymbol{net}_k = \begin{pmatrix} \boldsymbol{w}_o^T \boldsymbol{y}_{k1} & \boldsymbol{w}_o^T \boldsymbol{y}_{k2} & \cdots & \boldsymbol{w}_o^T \boldsymbol{y}_{kN} \end{pmatrix} \tag{29}$$

# Now, we have

**Thus, we have $z_k$ (In our case $k = 1$, but it could be a range of values)**

$$z_k = \begin{pmatrix} f\left(w_o^T y_{k1}\right) & f\left(w_o^T y_{k2}\right) & \cdots & f\left(w_o^T y_{kN}\right) \end{pmatrix} \quad (30)$$

**Thus, we generate a vector of differences**

$$d = t - z_k = \begin{pmatrix} t_1 - f\left(w_o^T y_{k1}\right) & t_2 - f\left(w_o^T y_{k2}\right) & \cdots & t_N - f\left(w_o^T y_{kN}\right) \end{pmatrix} \quad (31)$$

where $t = \begin{pmatrix} t_1 & t_2 & \cdots & t_N \end{pmatrix}$ is a row vector of desired outputs for each sample.

Cinvestav

# Now, we multiply element wise

## We have the following vector of derivatives of $net$

$$\boldsymbol{D}_f = \left( \begin{array}{cccc} \eta f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k1}\right) & \eta f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k2}\right) & \cdots & \eta f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{kN}\right) \end{array} \right) \tag{32}$$

where $\eta$ is the step rate.

## Finally, by element wise multiplication (Hadamard Product)

$$\begin{aligned} \boldsymbol{d} = ( & \eta \left[t_1 - f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k1}\right)\right] f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k1}\right) \quad \eta \left[t_2 - f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k2}\right)\right] f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k2}\right) \quad \cdots \\ & \eta \left[t_N - f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{kN}\right)\right] f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{kN}\right)) \end{aligned}$$

# Tile $\boldsymbol{d}$

## Tile downward

$$\boldsymbol{d}_{tile} = n_H \text{ rows } \begin{Bmatrix} \begin{pmatrix} \boldsymbol{d} \\ \boldsymbol{d} \\ \vdots \\ \boldsymbol{d} \end{pmatrix} \end{Bmatrix} \tag{33}$$

## Finally, we multiply element wise against $\boldsymbol{y}_1$ (Hadamard Product)

$$\Delta \boldsymbol{w}_{1j}^{temp} = \boldsymbol{y}_1 \circ \boldsymbol{d}_{tile} \tag{34}$$

# We obtain the total $\Delta \boldsymbol{w}_{1j}$

## We sum along the rows of $\Delta \boldsymbol{w}_{1j}^{temp}$

$$\Delta \boldsymbol{w}_{1j} = \begin{pmatrix} \eta \left[ t_1 - f\left( \boldsymbol{w}_o^T \boldsymbol{y}_{k1} \right) \right] f'\left( \boldsymbol{w}_o^T \boldsymbol{y}_{k1} \right) y_{11} + \eta \left[ t_1 - f\left( \boldsymbol{w}_o^T \boldsymbol{y}_{k1} \right) \right] f'\left( \boldsymbol{w}_o^T \boldsymbol{y}_{k1} \right) y_{1N} \\ \vdots \\ \eta \left[ t_1 - f\left( \boldsymbol{w}_o^T \boldsymbol{y}_{k1} \right) \right] f'\left( \boldsymbol{w}_o^T \boldsymbol{y}_{k1} \right) y_{n_H 1} + \eta \left[ t_1 - f\left( \boldsymbol{w}_o^T \boldsymbol{y}_{k1} \right) \right] f'\left( \boldsymbol{w}_o^T \boldsymbol{y}_{k1} \right) y_{n_H N} \end{pmatrix} \tag{35}$$

where $y_{hm} = f\left( \boldsymbol{w}_h^T \boldsymbol{x}_m \right)$ with $h = 1, 2, ..., n_H$ and $m = 1, 2, ..., N$.

Cinvestav

# Finally, we update the first weights

**We have then**

$$\boldsymbol{W}_{HO}(t+1) = \boldsymbol{W}_{HO}(t) + \Delta \boldsymbol{w}_{1j}^{T}(t) \tag{36}$$

# First

## We multiply element wise the $\boldsymbol{W}_{HO}$ and $\Delta \boldsymbol{w}_{1j}$

$$\boldsymbol{T} = \Delta \boldsymbol{w}_{1j}^T \circ \boldsymbol{W}_{HO}^T \tag{37}$$

## Now, we obtain the element wise derivative of $\boldsymbol{net}_j$

$$\boldsymbol{Dnet}_j = \begin{pmatrix} f'\left(\boldsymbol{w}_1^T \boldsymbol{x}_1\right) & f'\left(\boldsymbol{w}_1^T \boldsymbol{x}_2\right) & \cdots & f'\left(\boldsymbol{w}_1^T \boldsymbol{x}_N\right) \\ f'\left(\boldsymbol{w}_2^T \boldsymbol{x}_1\right) & f'\left(\boldsymbol{w}_2^T \boldsymbol{x}_2\right) & \cdots & f'\left(\boldsymbol{w}_2^T \boldsymbol{x}_N\right) \\ \vdots & \vdots & \ddots & \vdots \\ f'\left(\boldsymbol{w}_{n_H}^T \boldsymbol{x}_1\right) & f'\left(\boldsymbol{w}_{n_H}^T \boldsymbol{x}_2\right) & \cdots & f'\left(\boldsymbol{w}_{n_H}^T \boldsymbol{x}_N\right) \end{pmatrix} \tag{38}$$

# Thus

## We tile to the right $T$

$$\boldsymbol{T}_{tile} = \underbrace{\begin{pmatrix} \boldsymbol{T} & \boldsymbol{T} & \cdots & \boldsymbol{T} \end{pmatrix}}_{N \text{ Columns}} \tag{39}$$

## Now, we multiply element wise together with $\eta$

$$\boldsymbol{P}_t = \eta \left( \boldsymbol{Dnet}_j \circ \boldsymbol{T}_{tile} \right) \tag{40}$$

where $\eta$ is constant multiplied against the result the Hadamar Product (Result a $n_H \times N$ matrix)

# Finally

**We get use the transpose of $\boldsymbol{X}$ which is a $N \times d$ matrix**

$$\boldsymbol{X}^T = \begin{pmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{pmatrix} \tag{41}$$

**Finally, we get a $n_H \times d$ matrix**

$$\Delta \boldsymbol{w}_{ij} = \boldsymbol{P}_t \boldsymbol{X}^T \tag{42}$$

**Thus, given $\boldsymbol{W}_{IH}$**

$$\boldsymbol{W}_{IH}(t+1) = \boldsymbol{W}_{HO}(t) + \Delta \boldsymbol{w}_{ij}^T(t) \tag{43}$$

Cinvestav

# Maximizing information content

## Two ways of achieving this, LeCun 1993

- The use of an example that results in the largest training error.
- The use of an example that is radically different from all those previously used.

## For this

Randomized the samples presented to the multilayer perceptron when not doing batch training.

## Or use an emphasizing scheme

By using the error identify the difficult vs. easy patterns:

- Use them to train the neural network

Cinvestav

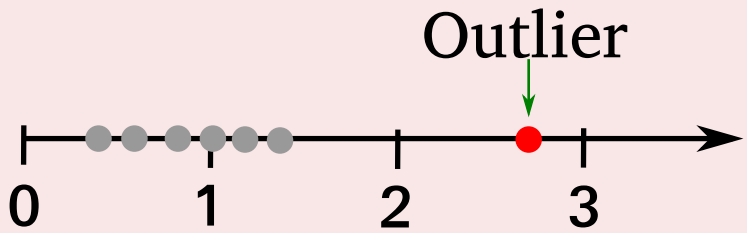
# However

## Be careful about emphasizing scheme

- The distribution of examples within an epoch presented to the network is distorted.
- The presence of an outlier or a mislabeled example can have a catastrophic consequence on the performance of the algorithm.

## Definition of Outlier

An outlier is an observation that lies outside the overall pattern of a distribution (Moore and McCabe 1999).




Cinvestav

# Activation Functions [5]

## We say that

An activation function $f(v)$ is antisymmetric if $f(-v) = -f(v)$

## It seems to be

That the multilayer perceptron learns faster using an antisymmetric function.

- For example, the hyperbolic tangent function

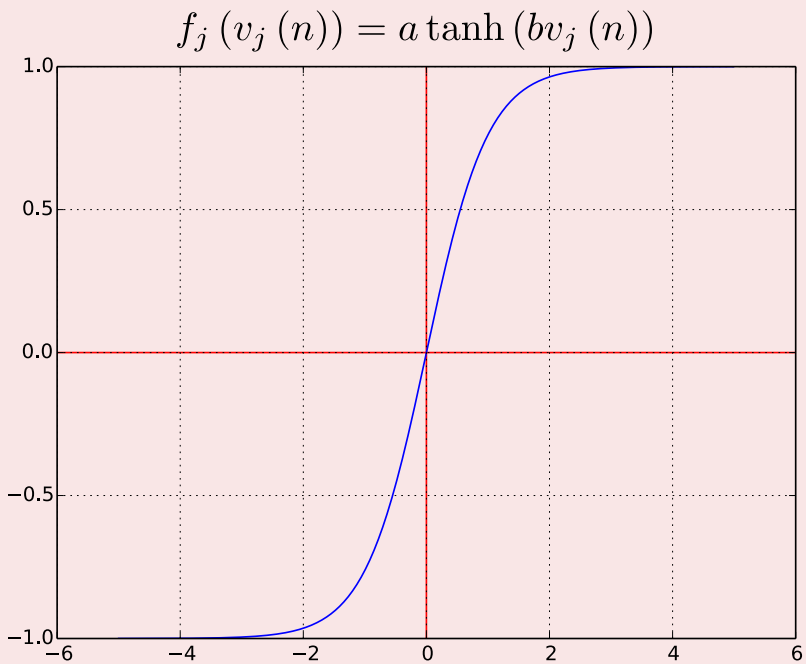
Cinvestav

# Hyperbolic tangent function

## Another commonly used form of sigmoid non linearity is the hyperbolic tangent function

$$f_j\left(v_j\left(n\right)\right) = a\tanh\left(bv_j\left(n\right)\right) \tag{44}$$

## Example



# The differential of the hyperbolic tangent

## We have

$$\begin{aligned} f_j\left(v_j\left(n\right)\right) =& ab\text{sech}^2\left(bv_j\left(n\right)\right) \\ =& ab\left(1-\tanh^2\left(bv_j\left(n\right)\right)\right) \end{aligned}$$

## BTW

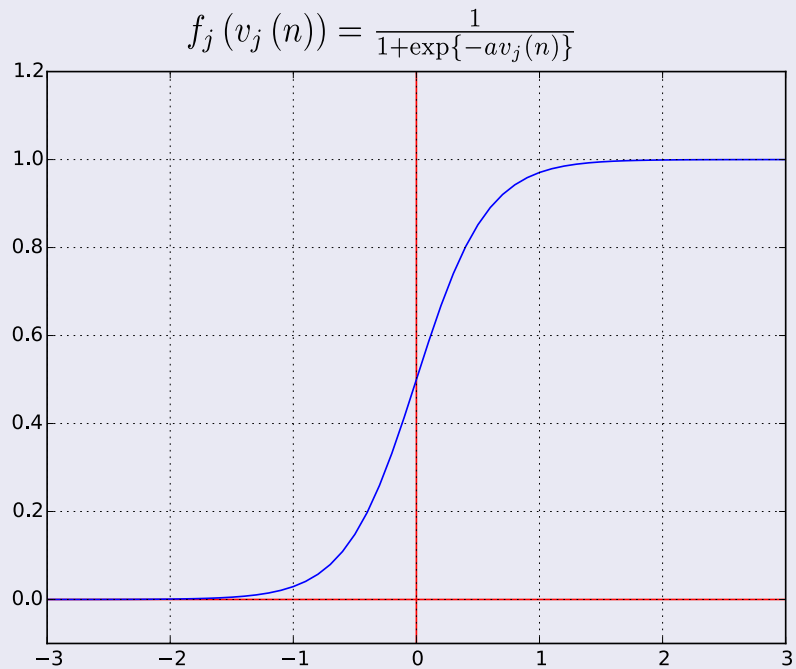
I leave to you to figure out the outputs.

# Logistic Function

## This non-linear function has the following definition for a neuron $j$

$$f_j\left(v_j\left(n\right)\right) = \frac{1}{1+\exp\left\{-av_j\left(n\right)\right\}} \quad a > 0 \text{ and } \quad -\infty < v_j\left(n\right) < \infty \tag{45}$$

## Example



# The differential of the sigmoid function

## Now, if we differentiate it, we have

$$f_j'(v_j(n)) = \left[\frac{1}{1+\exp\{-av_j(n)\}}\right]\left[1-\frac{1}{1+\exp\{-av_j(n)\}}\right]$$
$$= \frac{\exp\{-av_j(n)\}}{(1+\exp\{-av_j(n)\})^2}$$

# The outputs finish as

## For the output neurons

$$\begin{aligned}\delta_k &= (t_k - z_k) f'(net_k) \\ &= (t_k - f_k(v_k(n))) f_k(v_k(n)) (1 - f_k(v_k(n)))\end{aligned}$$

## For the hidden neurons

$$\delta_j = f_j(v_j(n)) (1 - f_j(v_j(n))) \sum_{k=1}^{c} w_{kj}\delta_k$$

Cinvestav

# Problems

## It is clear from our first class

- The problems of the sigmoid activation function, the vanishing gradient

## Not only that, we want activation functions

- That accelerate the learning
- That can control the previous problem
- An any other possible properties

Cinvestav

# ReLU

## Something Notable

- This activation function was first introduced to a dynamical network by Hahnloser et al. [6].

## Finally eleven years after

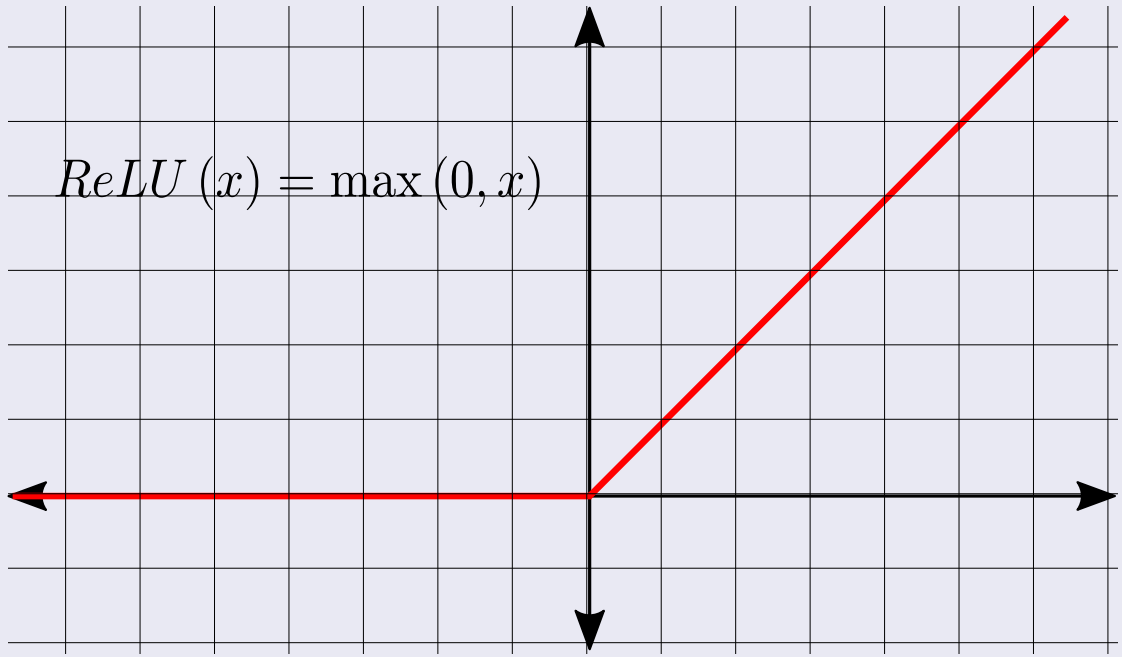
- It was proved that the ReLU function could accelerate the training of deep networks [7].

# Definition

## Rectified Linear Unit (ReLU)

$$ReLU: \mathbb{R} \longrightarrow \mathbb{R}$$
$$ReLU\left(x\right) = \max\left(0, x\right), \ x \in \mathbb{R}$$

## Plotting ReLU



# Noisy ReLUs

## Definition

$$f(x) = \max(0, x + Y) \text{ with } Y \sim N(0, \sigma(x))$$

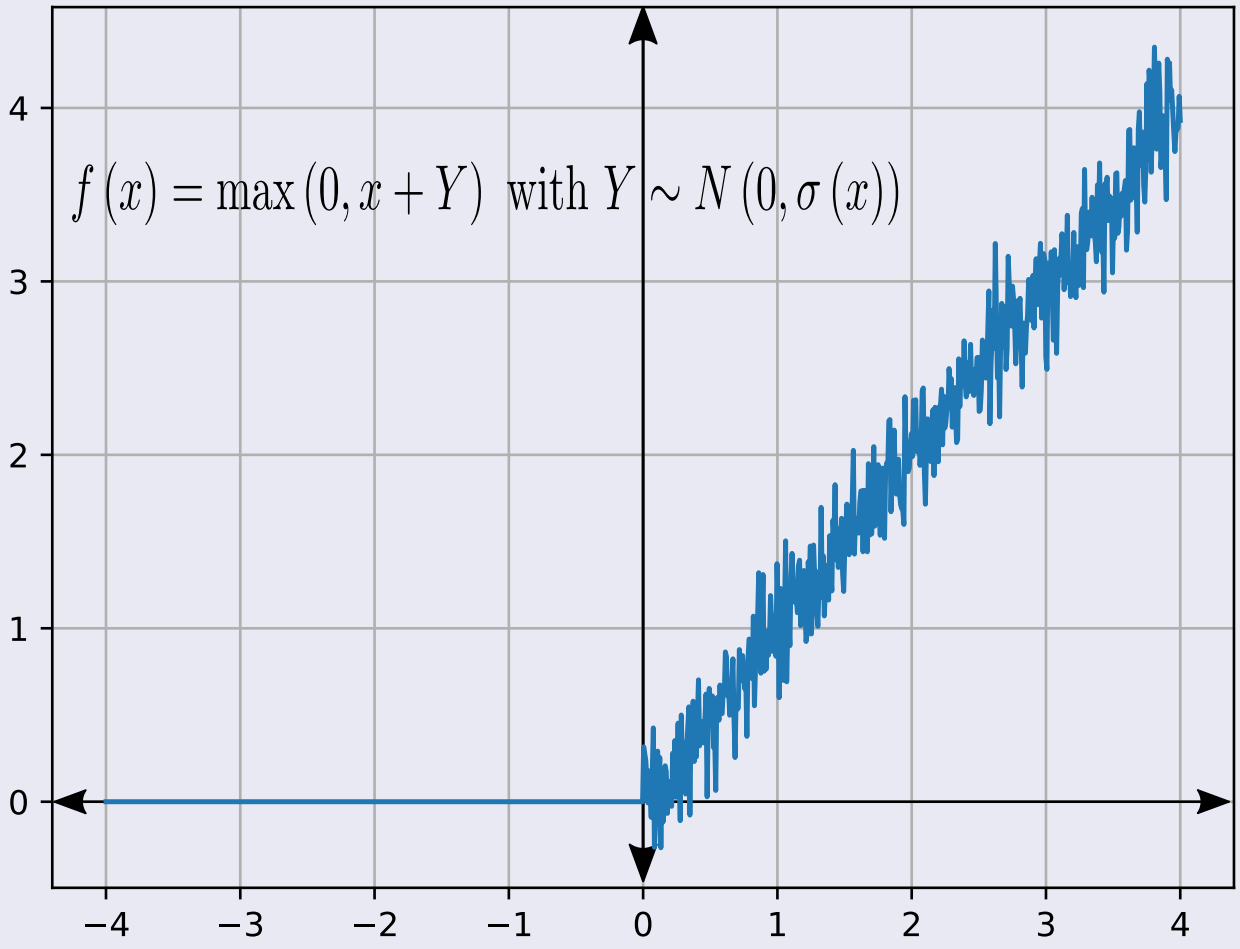
## They have been used

- In restricted Boltzmann machines for computer-vision tasks.

Cinvestav

# We have

## Plot

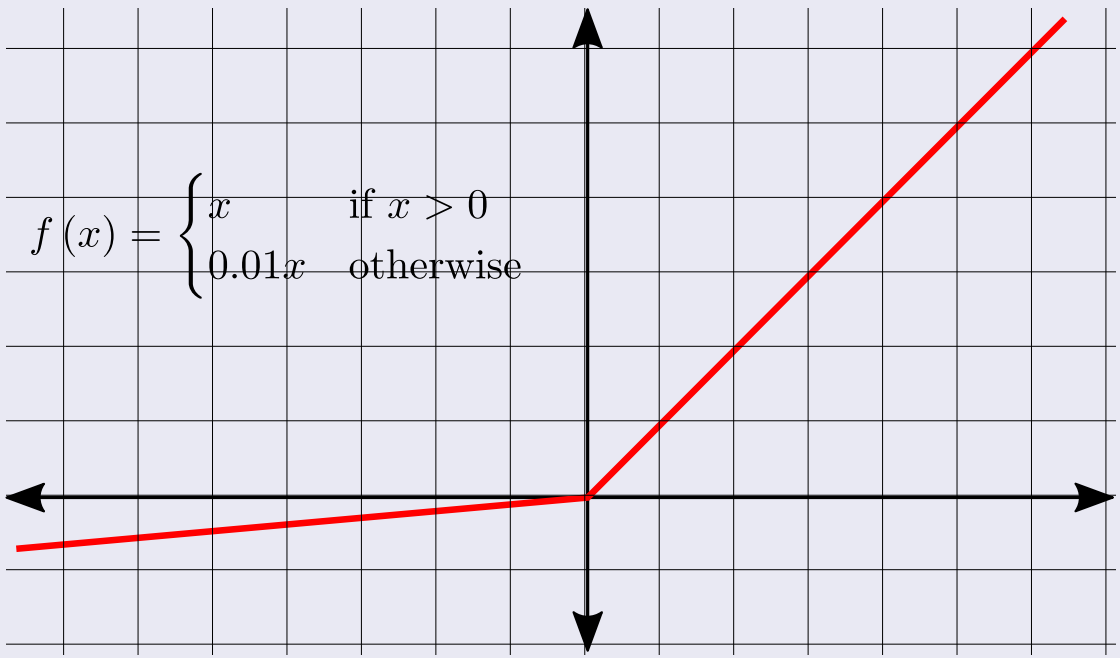


# Leaky ReLU

## Definition

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$

## Plotting



# Furthermore

## The Parametric ReLU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{otherwise} \end{cases}$$

- with $a \leq 1$

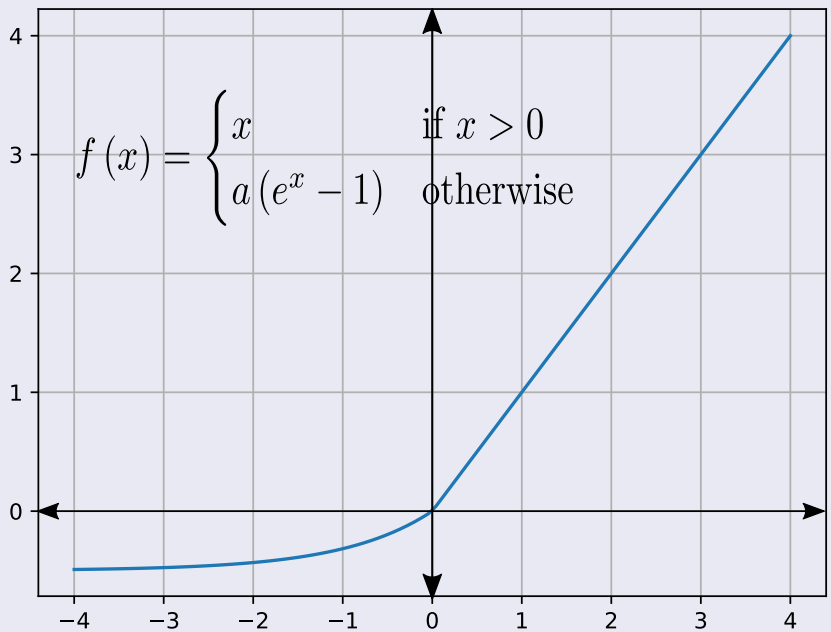
# ELU

## Definition

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ a\left(e^x - 1\right) & \text{otherwise} \end{cases}$$

- where $a$ is a hyper-parameter to be tuned, and $a \geq 0$.

## Plotting



# Target Values

**Important**

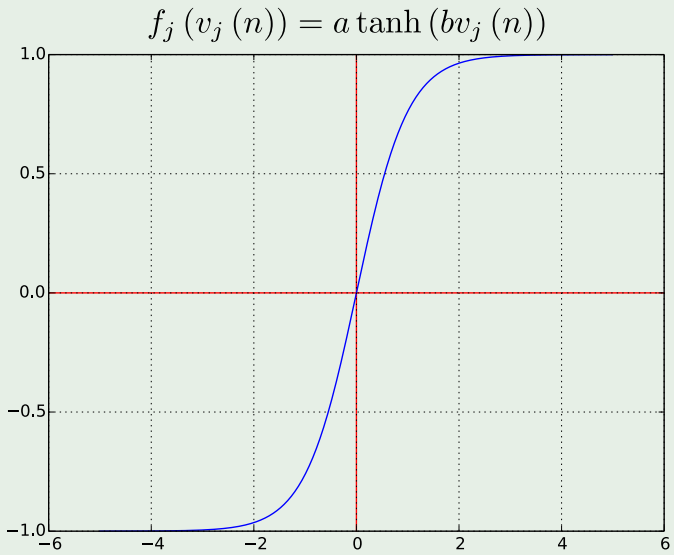
It is important that the target values be chosen within the range of the sigmoid activation function.

**Specifically**

The desired response for neuron in the output layer of the multilayer perceptron should be offset by some amount $\epsilon$

# For example

## Given the $a$ limiting value




## We have then

- If we have a limiting value $+a$, we set $t = a - \epsilon$.
- If we have a limiting value $-a$, we set $t = -a + \epsilon$.

Cinvestav

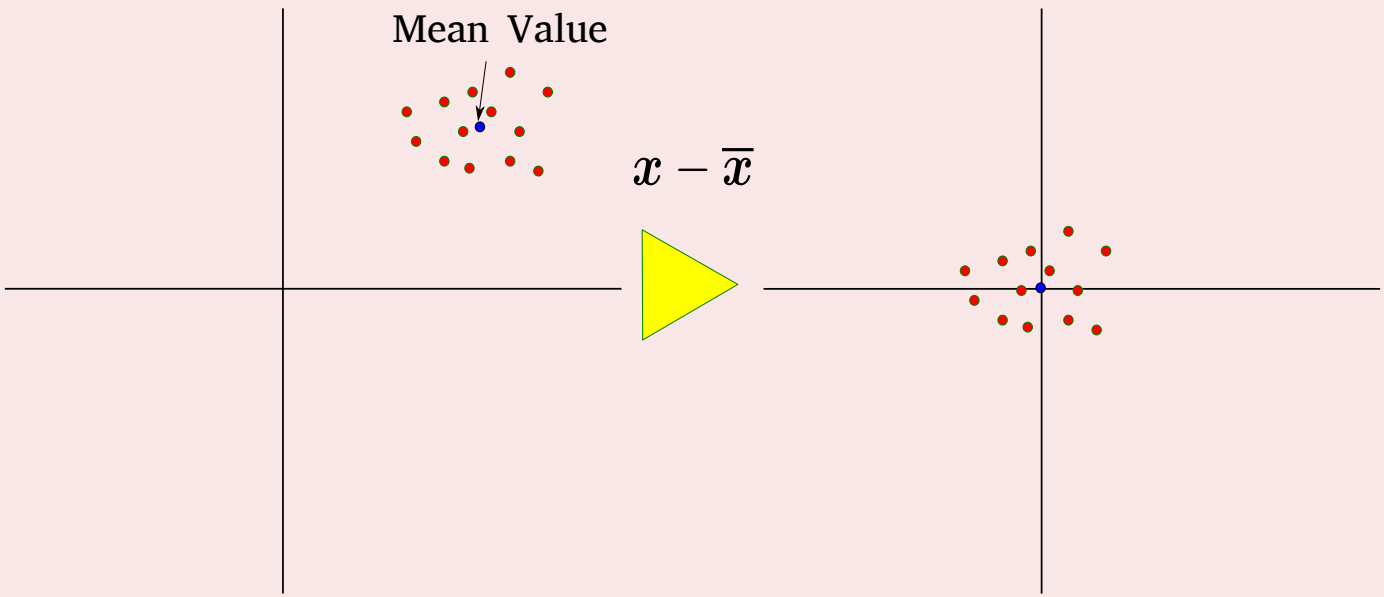
# Normalizing the inputs

## Something Important (LeCun, 1993)

Each input variable should be preprocessed so that:

- The mean value, averaged over the entire training set, is close to zero.
- Or it is small compared to its standard deviation.

## Example

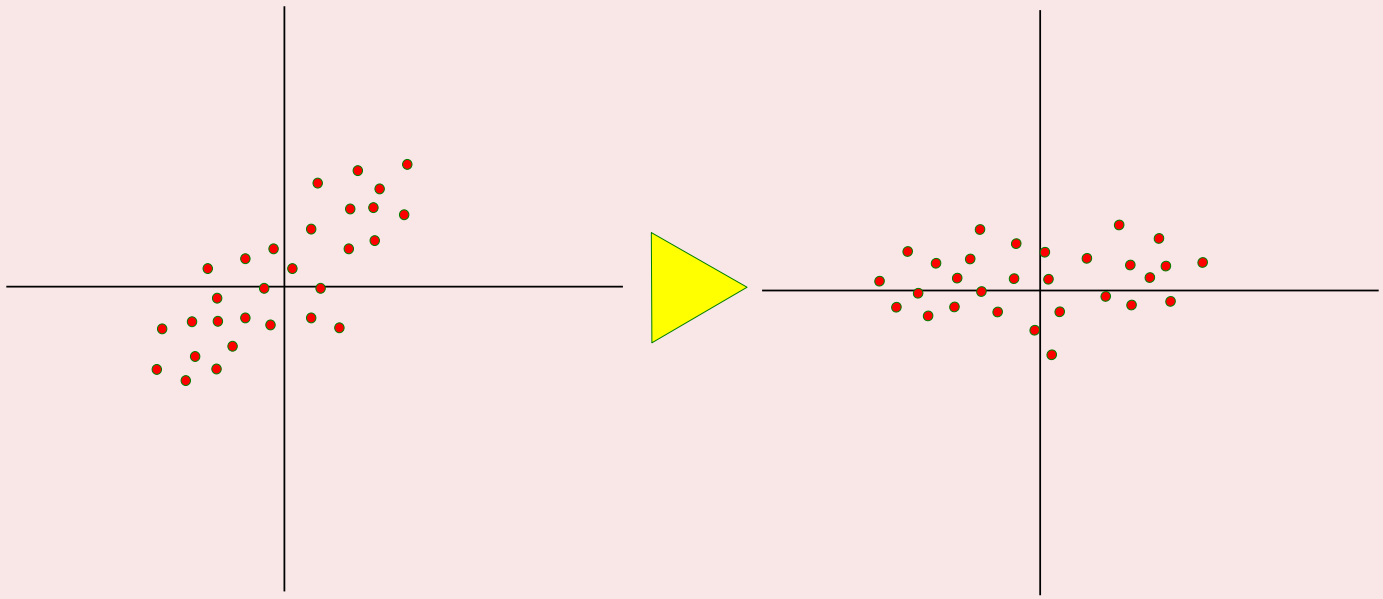


# The normalization must include two other measures

## Uncorrelated

We can use the principal component analysis

## Example



Cinvestav

# In addition

## Quite interesting

- The decorrelated input variables should be scaled so that their covariances are approximately equal.

## Why

- Ensuring that the different synaptic weights learn at approximately the same speed.

# There are other heuristics

## As

- Initialization
- Learning form hints
- Learning rates
- etc

# In addition

## In section 4.15, Simon Haykin

We have the following techniques:

- Network growing
  - You start with a small network and add neurons and layers to accomplish the learning task.
- Network pruning
  - Start with a large network, then prune weights that are not necessary in an orderly fashion.

# Virtues and limitations of Back-Propagation Algorithm

## Something Notable

The back-propagation algorithm has emerged as the most popular algorithm for the training of multilayer perceptrons.

## It has two distinct properties

- It is simple to compute locally.
- It performs stochastic gradient descent in weight space when doing pattern-by-pattern training

# Connectionism

## Back-propagation

t is an example of a connectionist paradigm that relies on local computations to discover the processing capabilities of neural networks.

## This form of restriction

It is known as the locality constraint

# Why this is advocated in Artificial Neural Networks

**First**

Artificial neural networks that perform local computations are often held up as metaphors for biological neural networks.

**Second**

The use of local computations permits a graceful degradation in performance due to hardware errors, and therefore provides the basis for a fault-tolerant network design.

**Third**

Local computations favor the use of parallel architectures as an efficient method for the implementation of artificial neural networks.

# However, all this has been seriously questioned [8, 9, 10]

## First

- The reciprocal synaptic connections between the neurons of a multilayer perceptron may assume weights that are excitatory or inhibitory.
- In the real nervous system, neurons usually appear to be the one or the other.

## Second

In a multilayer perceptron, hormonal and other types of global communications are ignored.

# However, all this has been seriously questioned [8, 9, 10]

## Third

- In back-propagation learning, a synaptic weight is modified by a presynaptic activity and an error (learning) signal independent of postsynaptic activity.
- There is evidence from neurobiology to suggest otherwise.

## Fourth

- In a neurobiological sense, the implementation of back-propagation learning requires the rapid transmission of information backward along an axon.
- It appears highly unlikely that such an operation actually takes place in the brain.

# However, all this has been seriously questioned [8, 9, 10]

## Fifth

- Back-propagation learning implies the existence of a "teacher," which in the con text of the brain would presumably be another set of neurons with novel properties.
- The existence of such neurons is biologically implausible.

# Computational Efficiency

## Something Notable

The computational complexity of an algorithm is usually measured in terms of the number of multiplications, additions, and storage involved in its implementation.

- This is the electrical engineering approach!!!

## Taking in account the total number of synapses, $W$ including biases

We have $\triangle w_{kj} = \eta \delta_k y_j = \eta \left( t_k - z_k \right) f' \left( net_k \right) y_j$ (Backward Pass)

## We have that for this step

1. We need to calculate $net_k$ linear in the number of weights.
2. We need to calculate $y_j = f \left( net_j \right)$ which is linear in the number of weights.

Cinvestav

# Computational Efficiency

## Now the Forward Pass

$$\Delta w_{ji} = \eta x_i \delta_j = \eta f'(net_j) \left[ \sum_{k=1}^{c} w_{kj} \delta_k \right] x_i$$

## We have that for this step

$[\sum_{k=1}^{c} w_{kj}\delta_k]$ takes, because of the previous calculations of $\delta_k$'s, linear on the number of weights

## Clearly all this takes to have memory

In addition the calculation of the derivatives of the activation functions, but assuming a constant time.

# We have that

**The Complexity of the multi-layer perceptron is**

$O(W)$ Complexity

# Exercises

**We have from NN by Haykin**

4.2, 4.3, 4.6, 4.8, 4.16, 4.17, 3.7

# Introduction

## Representation of functions

The main result in multi-layer perceptron is its power of representation.

## Furthermore

After all, it is quite striking if we can represent continuous functions of the form $f : \mathbb{R}^n \longmapsto \mathbb{R}$ as a finite sum of simple functions.

# Therefore

## Our main goal

We want to know under which conditions the sum of the form:

$$G(\boldsymbol{x}) = \sum_{j=1}^{N} \alpha_j f\left(\boldsymbol{w}^T \boldsymbol{x} + \theta_j\right) \tag{46}$$

can represent continuous functions in a specific domain.

Cinvestav

# Setup of the problem

## Definition of $I_n$

It is an $n$-dimensional unit cube $[0,1]^n$

## In addition, we have the following set of functions

$$C(I_n) = \{f : I_n \to \mathbb{R} | f \text{ is a continous function}\} \tag{47}$$

Cinvestav

# Now, some important definitions

## Definition (Topological Space)

A topological space is then a set $X$ together with a collection of subsets of $X$, called **open sets** and satisfying the following axioms:

1. The empty set and $X$ itself are open.
2. Any union of open sets is open.
3. The intersection of any finite number of open sets is open.

## Examples

Given the set $\{1, 2, 3, 4\}$ we have that the following set is a topology $\{\emptyset, \{1\}, \{1, 2\}, \{1, 2, 3, 4\}\}$.

## Remark

- This is quite axiomatic... because any set in the collection of $X$ is open...

# Then

## We are interested in defining open and close sets in metric spaces

- After all this will allow to define the concept of closed set

## Definition

- A subset $U$ of a metric space $(M, d)$ is called open if, given any point $x \in U$, there exists a real number $\epsilon > 0$ such that, given any point $y \in M$ with $d(x, y) < \epsilon$, $y$ also belongs to $U$.

## Therefore

- A set $V \subset M$ is closed if $M - V$ is open.

Cinvestav

# Now

**Theorem**

A compact set is closed and bounded.

**Thus**

$I_n$ is a compact set in $\mathbb{R}^n$.

**Definition (Continuous functions)**

A function $f : X \to Y$ where the pre-image of every open set in $Y$ is open in $X$.

# Thus

## We have the following statement

Let $K$ be a nonempty subset of $\mathbb{R}^n$, where $n > 1$. Then If $K$ is compact, then every continuous real-valued function defined on $K$ is **bounded.**

## Definition (Supremum Norm)

Let $X$ be a topological space and let $F$ be the space of all bounded complex-valued continuous functions defined on $K$. The supremum norm is the norm defined on $F$ by

$$\|f\| = \sup_{x \in X} |f(x)| \tag{48}$$

# Limit Points

## Definition

If $X$ is a topological space and $p$ is a point in $X$, a neighborhood of $p$ is a subset $V$ of $X$ that includes an open set $U$ containing $p$, $p \in U \subseteq V$.

- This is also equivalent to $p \in X$ being in the interior of $V$.

## Example in a metric space

In a metric space $(X, d)$, a set $V$ is a neighborhood of a point $p$ if there exists an open ball with center at $p$ and radius $r > 0$, such that

$$B_r(p) = B(p; r) = \{x \in X | d(x, p) < r\} \tag{49}$$

is contained in $V$.

# Limit Points

## Definition of a Limit Point

Let $S$ be a subset of a topological space $X$. A point $x \in X$ is a limit point of $S$ if every neighborhood of $x$ contains at least one point of $S$ different from $x$ itself.

## Example in $\mathbb{R}$

Which are the limit points of the set $\left\{\frac{1}{n}\right\}_{n=1}^{\infty}$?

# This allows to define the idea of density

## Something Notable

A subset $A$ of a topological space $X$ is dense in $X$, if for any point $x \in X$, any neighborhood of $x$ contains at least one point from $A$.

## Classic Example

The real numbers with the usual topology have the rational numbers as a countable dense subset.

- Why do you believe the floating-point numbers are rational?

## In addition

Also the irrational numbers.

# From this, you have the idea of closure

**Definition**

The closure of a set $S$ is the set of all points of closure of $S$, that is, the set $S$ together with all of its limit points.

**Example**

The closure of the following set $(0,1) \cup \{2\}$

**Meaning**

Not all points in the closure are limit points.

Cinvestav

# Hausdorff Space

## Definition of Separation

Points $x$ and $y$ in a topological space $X$ can be separated by neighborhoods if there exists a neighborhood $U$ of $x$ and a neighborhood $V$ of $y$ such that $U$ and $V$ are disjoint.

## Definition

$X$ is a Hausdorff space if any two distinct points of $X$ can be separated by neighborhoods.

Cinvestav

# We have then

**Look at what we have**

1. $C(I_n)$ is compact
2. The continuous functions there are bounded!!!

# Now, the Measure Concept

## Definition of $\sigma-$algebra

Let $\mathcal{A} \subset \mathcal{P}(X)$, we say that $\mathcal{A}$ to be an algebra if

1. $\emptyset$, $X \in \mathcal{A}$.
2. $A, B \in \mathcal{A}$ then $A \cup B \in \mathcal{A}$.
3. $A \in \mathcal{A}$ then $A^c \in \mathcal{A}$.

## Definition

An algebra $\mathcal{A}$ in $\mathcal{P}(X)$ is said to be a $\sigma-$algebra, if for any sequence $\{A_n\}$ of elements in $\mathcal{A}$, we have $\cup_{n=1}^{\infty} A_n \in \mathcal{A}$

## Example

In $X = [0, 1)$, the class $\mathcal{A}_0$ consisting of $\emptyset$, and all finite unions $A = \cup_{i=1}^{n} [a_i, b_i)$ with $0 \leq a_i < b_i \leq a_{i+1} \leq 1$ is an algebra.

Cinvestav

# Now, the Measure Concept

## Definition of additivity

Let $\mu : \mathcal{A} \to [0, +\infty]$ be such that $\mu(\emptyset) = 0$, we say that $\mu$ is $\sigma-$additive if for any $\{A_i\}_{i \in I} \subset \mathcal{A}$ (Where $I$ can be finite of infinite countable) of mutually disjoint sets such that $\cup_{i \in I} A_i \in \mathcal{A}$, we have that

$$\mu\left(\cup_{i \in I} A_i\right) = \sum_{i \in I} \mu\left(A_i\right) \tag{50}$$

## Definition of Measurability

Let $\mathcal{A}$ be a $\sigma-$algebra of subsets of $X$, we say that the [air $(X, \mathcal{A})$ is a measurable space where a $\sigma-$additive function $\mu : \mathcal{A} \to [0, +\infty]$ is called a measure on $(X, \mathcal{A})$.

Cinvestav

# A Borel Measure

## Definition

The Borel $\sigma$-algebra is defined to be the $\sigma$-algebra generated by the open sets (or equivalently, by the closed sets).

## Definition of a Borel Measure

If $\mathcal{F}$ is the Borel $\sigma$-algebra on some topological space, then a measure $\mu : \mathcal{F} \to \mathbb{R}$ is said to be a Borel measure (or Borel probability measure). For a Borel measure, all continuous functions are measurable.

## Definition of a signed Borel Measure

A signed Borel measure $\mu : \mathcal{B}(X) \to$ is a measure such that

1. $\mu(\emptyset) = 0$.
2. $\mu$is $\sigma$-additive.
3. $\sup_{A \in \mathcal{B}(X)} |\mu(A)| < \infty$

# A Borel Measure

## Regularity

A measure $\mu$ is Borel regular measure:

1. For every Borel set $B \subseteq \mathbb{R}^n$ and $A \subseteq \mathbb{R}^n$, $\mu(A) = \mu(A \cap B) + \mu(A - B)$.
2. For every $A \subseteq \mathbb{R}^n$, there exists a Borel set $B \subseteq \mathbb{R}^n$ such that $A \subseteq B$ and $\mu(A) = \mu(B)$.

Cinvestav

# Discriminatory Functions

## Definition

Given the set $M(I_n)$ of signed regular Borel measures, a function $f$ is discriminatory if for a measure $\mu \in M(I_n)$

$$\int_{I_n} f\left(\boldsymbol{w}^T \boldsymbol{x} + \theta\right) d\mu = 0 \tag{51}$$

for all $\boldsymbol{w} \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ implies that $\mu = 0$

## Definition

We say that $f$ is sigmoidal if

$$f(t) \to \begin{cases} 1 & \text{as } t \to +\infty \\ 0 & \text{as } t \to -\infty \end{cases}$$

Cinvestav

# The Important Theorem

## Theorem 1

Let $f$ a be any continuous discriminatory function. Then finite sums of the form

$$G(\boldsymbol{x}) = \sum_{j=1}^{N} \alpha_j f\left(\boldsymbol{w}_j^T \boldsymbol{x} + \theta_j\right), \tag{52}$$

where $\boldsymbol{w}_j \in \mathbb{R}^n$ and $\alpha_j$, $\theta_j \in \mathbb{R}$ are fixed, are dense in $C(I_n)$

## Meaning

Basically given any function $g \in C(I_n)$ and any neighborhood $V$ of $g$, you have a $G \in V$.

# Furthermore

## In other words

Given any $g \in C(I_n)$ and $\epsilon > 0$, there is a sum, $G(\boldsymbol{x})$, of the above form, for which

$$|G(\boldsymbol{x}) - g(\boldsymbol{x})| < \epsilon \; \forall \boldsymbol{x} \in I_n \tag{53}$$

# Proof

**Let $S \subset C(I_n)$ be the set of functions of the form $G(\boldsymbol{x})$**

First, $S$ is a linear subspace of $C(I_n)$

**Definition**

A subset $V$ of $\mathbb{R}^n$ is called a linear subspace of $\mathbb{R}^n$ if $V$ contains the zero vector, and is closed under vector addition and scaling. That is, for $X, Y \in V$ and $c \in \mathbb{R}$, we have $X + Y \in V$ and $cX \in V$.

**We claim that the closure of $S$ is all of $C(I_n)$**

Assume that the closure of $S$ is not all of $C(I_n)$

# Proof

## Then

- The closure of $S$, say $R$, is a closed proper subspace of $C(I)$

## We use the Hahn-Banach Theorem

- If $p : V \to \mathbb{R}$ is a sublinear function
  - $p(x+y) \leq p(x) + p(y)$
  - $p(\alpha x) = \alpha p(x)$

## And $\varphi : U \to \mathbb{R}$ is a linear functional on a linear subspace $U \subseteq V$

- It is dominated by $p$ on $U$, i.e. $\varphi(x) \leq p(x)\ \forall x \in U$.

Cinvestav

# Hahn-Banach Theorem

## Then

There exists a **linear extension** $\psi : V \to \mathbb{R}$ of $\varphi$ to the whole space $V$, i.e., there exists a linear functional $\psi$ such that

1. $\psi(x) = \varphi(x)\ \forall x \in U$.
2. $\psi(x) \leq p(x)\ \forall x \in V$.

Cinvestav

# Proof

## It is possible to construct sublinear function defined as follow

We define the following linear functional

$$T(f) = \begin{cases} f & \text{if } f \in C(I_n) - R \\ 0 & \text{if } f \in R \end{cases} \tag{54}$$

## Then

We have there is a bounded linear functional (Using $T$ as $p$ and $\varphi$, and $V \in C(I_n)$ and $U = R$) called $L \neq 0$ with $L(R) = L(S) = 0$.

Cinvestav

# Proof

## Now, we use the Riesz Representation Theorem

Let $X$ be a locally compact Hausdorff space. For any positive linear functional $\psi$ on $C(X)$, there is a unique regular Borel measure $\mu$ on $X$ such that

$$\psi = \int_X f(x)\, d\mu(x) \tag{55}$$

for all $f$ in $C(X)$

## We can then do the following

$$L(h) = \int_{I_n} h(\boldsymbol{x})\, d\mu(\boldsymbol{x}) \tag{56}$$

## Where?

For some $\mu \in M(I_n)$, for all $h \in C(I_n)$

# Proof

## In particular

Given that $f\left(\boldsymbol{w}^T\boldsymbol{x}+\theta\right)$ is in $R$ for all $\boldsymbol{w}$ and $\theta$

## We must have that

$$\int_{I_n} f\left(\boldsymbol{w}^T\boldsymbol{x}+\theta\right)d\mu\left(\boldsymbol{x}\right)=0 \tag{57}$$

for all $\boldsymbol{w}$ and $\theta$

## But we assumed that $f$ is discriminatory!!!

Then... $\mu=0$ contradicting the fact that $L\neq 0$!!! We have a contradiction!!!

# Proof

**Finally**

The subspace $S$ of sums of the form $G$ is dense!!!

# Now, we deal with the sigmoidal function

**Lemma 1**

Any bounded, measurable sigmoidal function, $f$, is discriminatory. In particular, any continuous sigmoidal function is discriminatory.

**Proof**

I will leave this to you... it is possible I will get a question from this proof for the firs midterm.

# We have the theorem finally!!!

## Universal Representation Theorem for the multi-layer perceptron

Let $f$ be any continuous sigmoid function. Then finite sums of the form

$$G(\boldsymbol{x}) = \sum_{j=1}^{N} \alpha_j f\left(\boldsymbol{w}^T \boldsymbol{x} + \theta_j\right) \tag{58}$$

are dense in $C(I_n)$.

## In other words

Given any $g \in C(I_n)$ and $\epsilon > 0$, there is a sum $G(\boldsymbol{x})$ of the above form, for which

$$|G(\boldsymbol{x}) - g(\boldsymbol{x})| < \epsilon \; \forall \boldsymbol{x} \in I_n \tag{59}$$

Cinvestav

# Poof

## Simple

Combine the theorem and lemma 1... and because the continuous sigmoid satisfy the conditions of the lemma... we have our representation!!!


A. E. Bryson, W. F. Denham, and S. E. Dreyfus, "Optimal programming problems with inequality constraints," *AIAA journal*, vol. 1, no. 11, pp. 2544–2550, 1963.


P. J. Werbos *et al.*, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.


D. E. Rumelhart and D. Zipser, "Feature discovery by competitive learning," *Cognitive science*, vol. 9, no. 1, pp. 75–112, 1985.


D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Neurocomputing: Foundations of research," ch. Learning Representations by Back-propagating Errors, pp. 696–699, Cambridge, MA, USA: MIT Press, 1988.


S. Haykin, *Neural Networks and Learning Machines*. No. v. 10 in Neural networks and learning machines, Prentice Hall, 2009.


Cinvestav

- R. H. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *Nature*, vol. 405, no. 6789, p. 947, 2000.
- X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, 2011.
- G. M. Shepherd, "The dendritic spine: a multifunctional integrative unit," *Journal of neurophysiology*, vol. 75, no. 6, pp. 2197–2210, 1996.
- F. Crick, "The recent excitement about neural networks.," *Nature*, vol. 337, no. 6203, pp. 129–132, 1989.
- D. G. Stork, "Is backpropagation biologically plausible," in *International Joint Conference on Neural Networks*, vol. 2, pp. 241–246, IEEE Washington, DC, 1989.


Cinvestav