Pattern Classification

10. Linear Perceptron, Least Squares & Multi-layer NNs

AbdElMoniem Bayoumi, PhD

Fall 2021

Recap: Linear Perceptron Algorithm

1. Initialize the weights and threshold (bias) randomly

2. Present the augmented input (or feature) vector of the m^th training $\underline{u}(m)$ and its corresponding desired output $\mathrm{d}(m)$

$$\boldsymbol{d(m)} = \begin{cases} \mathbf{1}, & \boldsymbol{if\ \underline{u}(m) \in C_1} \\ \mathbf{0}, & \boldsymbol{if\ \underline{u}(m) \in C_2} \end{cases}$$

3. Calculate the actual output for pattern m:

$$\boldsymbol{y(m) = f\left(\underline{W}^T \underline{u}(m)\right)}$$

4. Adapt the weights according to the following rule (called Widrow-Hoff rule):

$$\boldsymbol{\underline{W}(new) = \underline{W}(old) + \alpha[d(m) - y(m)]\underline{u}(m)}$$

where α is a constant called the learning rate

5. Go to step 2 until all patterns are classified correctly, i.e., d(m)=y(m) for m=1, … , M

Note: the algorithm is sequential w.r.t. the patterns $\underline{u}(m)$

Recap: Understanding Widrow-Hoff Update

- If d(m)=y(m) then no change is needed in the weights, i.e., $\underline{W}(new) = \underline{W}(old)$, because d(m)-y(m)=0

- If d(m)≠y(m) then weights get updated

$$\boldsymbol{\underline{W}}(\boldsymbol{new}) = \boldsymbol{\underline{W}}(\boldsymbol{old}) + \boldsymbol{\alpha}[\boldsymbol{d}(\boldsymbol{m}) - \boldsymbol{y}(\boldsymbol{m})]\boldsymbol{\underline{u}}(\boldsymbol{m})$$

Linear Perceptron Algorithm

$$\underline{\boldsymbol{W}}(\boldsymbol{new}) = \underline{\boldsymbol{W}}(\boldsymbol{old}) + \boldsymbol{\alpha}[\boldsymbol{d}(\boldsymbol{m}) - \boldsymbol{y}(\boldsymbol{m})]\underline{\boldsymbol{u}}(\boldsymbol{m})$$

- To show that each iteration corrects errors:
  - Let actual class $\boldsymbol{d}(\boldsymbol{m}) = \mathbf{1}$
  - If neuron classification $\boldsymbol{y}(\boldsymbol{m}) = \boldsymbol{f}\left(\underline{\boldsymbol{W}}^{\boldsymbol{T}}\underline{\boldsymbol{u}}(\boldsymbol{m})\right) = \mathbf{0}$
  - So, $\underline{\boldsymbol{W}}^{\boldsymbol{T}}\underline{\boldsymbol{u}}(\boldsymbol{m}) < \mathbf{0}$
  - However, we want $\boldsymbol{y}(\boldsymbol{m}) = \mathbf{1}$, i.e., $\mathbf{y}(\boldsymbol{m}) = \boldsymbol{d}(\boldsymbol{m})$
  - We need to correct wrong classification by making what is inside $\boldsymbol{f}(\cdot)$ more positive, which will make $\mathbf{y}(\boldsymbol{m})$ move likely to be 1

Linear Perceptron Algorithm

$$\underline{\boldsymbol{W}}(\boldsymbol{new}) = \underline{\boldsymbol{W}}(\boldsymbol{old}) + \boldsymbol{\alpha}[\boldsymbol{d}(\boldsymbol{m}) - \boldsymbol{y}(\boldsymbol{m})]\underline{\boldsymbol{u}}(\boldsymbol{m})$$

$$\boldsymbol{y}(\boldsymbol{new}) = \boldsymbol{f}\left(\underline{\boldsymbol{W}}^{\boldsymbol{T}}(\boldsymbol{new}).\underline{\boldsymbol{u}}(\boldsymbol{m})\right)$$

$$= \boldsymbol{f}\left(\underline{\boldsymbol{W}}^{\boldsymbol{T}}(\boldsymbol{old}).\underline{\boldsymbol{u}}(\boldsymbol{m}) + \boldsymbol{\alpha}[\boldsymbol{d}(\boldsymbol{m}) - \boldsymbol{y}(\boldsymbol{old})]\underline{\boldsymbol{u}}^{\boldsymbol{T}}(\boldsymbol{m})\underline{\boldsymbol{u}}(\boldsymbol{m})\right)$$

$$= \boldsymbol{f}\left(\underline{\boldsymbol{W}}^{\boldsymbol{T}}(\boldsymbol{old}).\underline{\boldsymbol{u}}(\boldsymbol{m}) + \boldsymbol{\alpha}[\mathbf{1} - \mathbf{0}]\underline{\boldsymbol{u}}^{\boldsymbol{T}}(\boldsymbol{m})\underline{\boldsymbol{u}}(\boldsymbol{m})\right)$$

$$= \boldsymbol{f}\left(-\boldsymbol{ve} \quad + \quad \boldsymbol{\alpha}\left\|\underline{\boldsymbol{u}}(\boldsymbol{m})\right\|^{2}\right) \qquad \boldsymbol{\alpha} > \mathbf{0}$$

+ve

- Tries to make what is inside $\boldsymbol{f}(\cdot)$ more positive, which will make $\mathbf{y}(\boldsymbol{m})$ move likely to be 1

Theorem (Rosenblatt)

- For a linearly separable problem the perceptron algorithm is guaranteed to converge, leading to a solution that classifies all points correctly

- If the problem is not linearly separable, then the algorithm will not converge & will keep cycling forever

- How to deal with not linearly separable problems?

Least Square Classifier

- We try to have the neuron produce positive numbers for patterns from class 1 & negative numbers for patterns from class 2

$$\underline{\boldsymbol{X}}(\boldsymbol{m}) \rightarrow \boldsymbol{b_m}$$

- $b_m > 0$ if $\underline{X}(m) \in C_1$
- $b_m < 0$ if $\underline{X}(m) \in C_2$

- $y(m) = f\left(\underline{W}^T \underline{u}(m)\right) = \underline{W}^T \underline{u}(m)$

linear activation fn.

Least Square Classifier

- Example:
  - $\underline{X}(1) = \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix}, \quad \underline{X}(1) \in C_1, \quad b_1 = 1$
  - $\underline{X}(2) = \begin{bmatrix} 3 \\ -2 \\ 5 \end{bmatrix}, \quad \underline{X}(2) \in C_1, \quad b_2 = 2$
  - $\underline{X}(3) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad \underline{X}(3) \in C_2, \quad b_3 = -3$

    $\vdots$

- We want:
  - $\underline{W}^T \underline{u}(1) \approx b_1 \equiv 1$
  - $\underline{W}^T \underline{u}(2) \approx b_2 \equiv 2$
  - $\underline{W}^T \underline{u}(3) \approx b_3 \equiv -3$

Find $\underline{W}$ that satisfy these equations!

- If **M>N+1**, then we cannot get exact equality because

#unknowns < #equations

$\underline{W} \rightarrow N + 1$ $\qquad M$

Least Square Classifier

- Define error function:

$$E = \sum_{m=1}^{M} (\underline{W}^T \underline{u}(m) - b_m)^2$$

- It measures how close the obtained solution is to the desired one.

- We then seek to minimize the error fn.

- Thus, we try to find $\underline{W}$ that minimizes E

Least Square Classifier

- Define error function:

$$E = \sum_{m=1}^{M} (\underline{W}^T \underline{u}(m) - b_m)^2$$

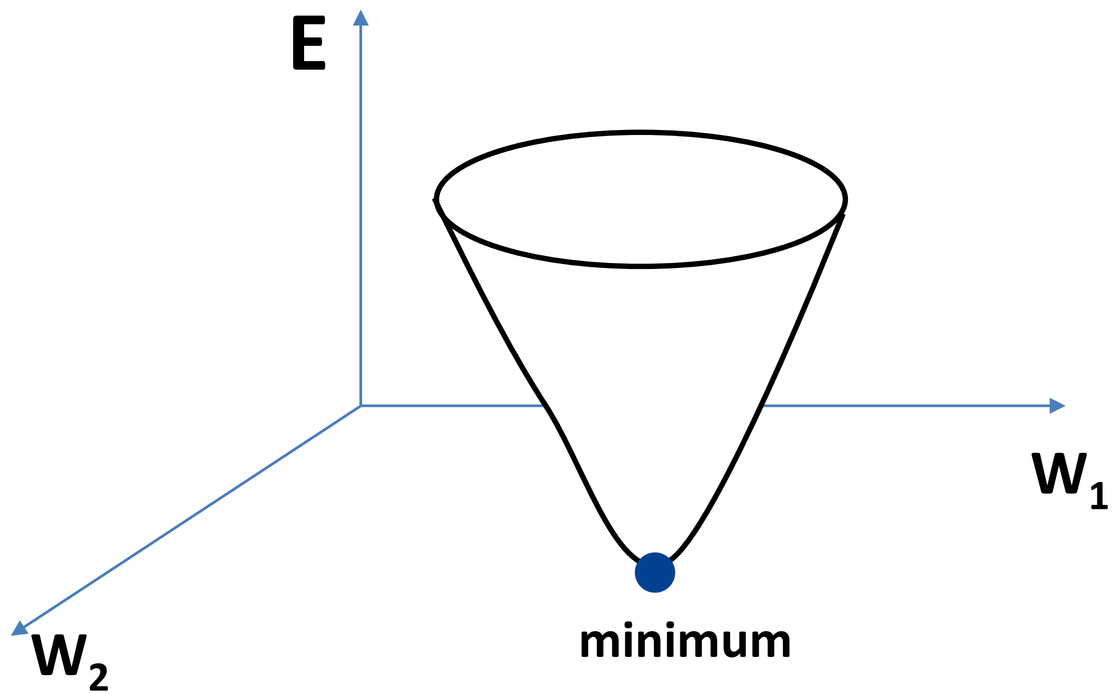
At the minimum:

$$\frac{\partial E}{\partial W_0} = 0$$

$$\frac{\partial E}{\partial W_1} = 0$$

$$\vdots$$

$$\frac{\partial E}{\partial W_N} = 0$$



Least Square Classifier

- Define the gradient vector:

$$\frac{\partial E}{\partial \underline{W}} = \begin{bmatrix} \frac{\partial E}{\partial W_0} \\ \frac{\partial E}{\partial W_1} \\ \vdots \\ \frac{\partial E}{\partial W_N} \end{bmatrix}$$

- Set $\frac{\partial \boldsymbol{E}}{\partial \underline{\boldsymbol{W}}} = \mathbf{0}$ and solve for $\underline{\boldsymbol{W}}$
- **Advantage:**
  - Can converge if the problem is not linearly separable
- **Disadvantage:**
  - Linear classifier → not suitable for most applications

Least Square Classifier

- $E = \sum_{m=1}^{M}(\underline{W}^T\underline{u}(m) - b_m)^2 = \sum_{m=1}^{M}\underbrace{(\underline{u}^T(m)\underline{W} - b_m)}_{Z_m}{}^2$

- Let $Y = \begin{bmatrix} \underline{u}^T(1) \\ \underline{u}^T(2) \\ \vdots \\ \underline{u}^T(M) \end{bmatrix}$ Matrix M x N

- Then, $Y\,\underline{W} - \underline{b} = \begin{bmatrix} \underline{u}^T(1) \\ \underline{u}^T(2) \\ \vdots \\ \underline{u}^T(M) \end{bmatrix}\underline{W} - \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_M \end{bmatrix} = \begin{bmatrix} \underline{u}^T(1)\underline{W} - b_1 \\ \underline{u}^T(2)\underline{W} - b_2 \\ \vdots \\ \underline{u}^T(M)\underline{W} - b_M \end{bmatrix} \equiv \underline{Z}$

- $E = \sum_{m=1}^{M} Z_m^2 = \left\|\underline{Z}\right\|^2 = \underline{Z}^T\underline{Z}$
$$\begin{aligned} &= \left[Y\,\underline{W} - \underline{b}\right]^T\left[Y\,\underline{W} - \underline{b}\right] \\ &= \left[\underline{W}^T Y^T - \underline{b}^T\right]\left[Y\,\underline{W} - \underline{b}\right] \\ &= \underline{W}^T Y^T Y\,\underline{W} - \underline{W}^T Y^T \underline{b} - \underline{b}^T Y\,\underline{W} + \underline{b}^T\underline{b} \\ &= \underline{W}^T Y^T Y\,\underline{W} - 2\underline{b}^T Y\,\underline{W} + \underline{b}^T\underline{b} \end{aligned}$$

Least Square Classifier

- $E = \underline{W}^T Y^T Y \, \underline{W} - 2\underline{b}^T Y \, \underline{W} + \underline{b}^T \underline{b}$
- $\dfrac{\partial \boldsymbol{E}}{\partial \underline{\boldsymbol{W}}} = \mathbf{0}$
- $\dfrac{\partial}{\partial \underline{\boldsymbol{W}}} \left[\underline{W}^T \underbrace{Y^T Y}_{\boldsymbol{A}} \, \underline{W} - 2 \underbrace{\underline{b}^T Y}_{\boldsymbol{C}^{\boldsymbol{T}}} \, \underline{W} + \underline{b}^T \underline{b} \right] = \mathbf{0}$
- $$\begin{aligned} \underline{W}^T A \, \underline{W} &= W_1^2 A_{11} + W_1 W_2 A_{12} + \cdots \\ &\quad + W_1 W_2 A_{21} + W_2^2 A_{22} + \cdots \\ &\quad + \cdots \end{aligned}$$

Quadratic form

- $\dfrac{\partial}{\partial \underline{\boldsymbol{W}}} \left[\underline{W}^T A \, \underline{W} \right] = 2A\underline{W}$ $\qquad \dfrac{\partial}{\partial \underline{\boldsymbol{W}}} \left[C^T \underline{W} \right] = C$

Exercise!

Least Square Classifier

- $$\frac{\partial \boldsymbol{E}}{\partial \underline{\boldsymbol{W}}} = \frac{\partial}{\partial \underline{\boldsymbol{W}}}\left[\underline{W}^T Y^T Y\,\underline{W} - 2\underline{b}^T Y\,\underline{W} + \underline{b}^T\underline{b}\right]$$
  $$= 2A\underline{W} - \boldsymbol{C} + \boldsymbol{0}$$
  $$= 2Y^T Y\,\underline{W} - 2Y^T b = \boldsymbol{0}$$

- $\underline{W} = (Y^T Y)^{-1} Y^T b$ **Weights for the least square classifier**

- Advantage:
  – Can converge if the problem is not linearly separable

Multi-Layer Networks

- We have seen that a single neuron, or a single layer network is capable of only producing linear classifiers

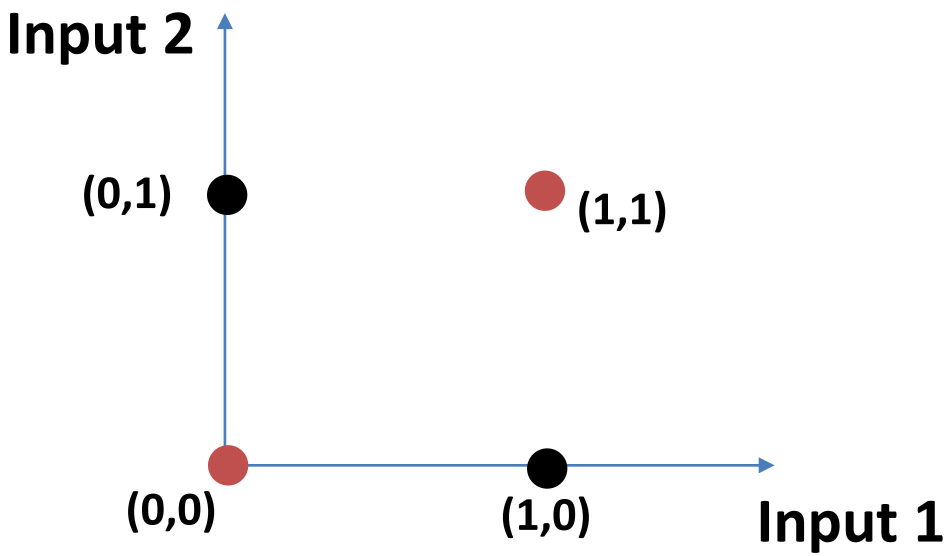
- Hence, it is not adequate for most applications

- Some very simple fn.'s like XOR fn. Cannot be implemented with a single neuron

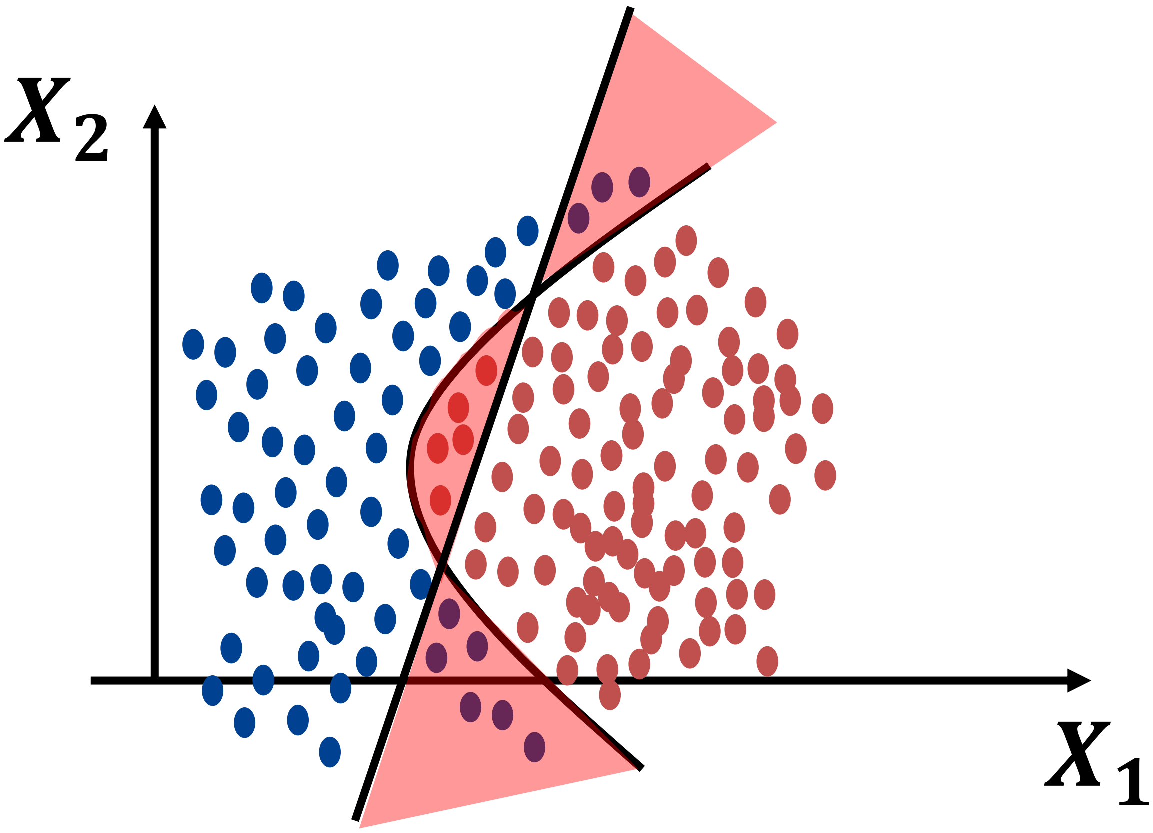
Multi-Layer Networks

- XOR function:

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

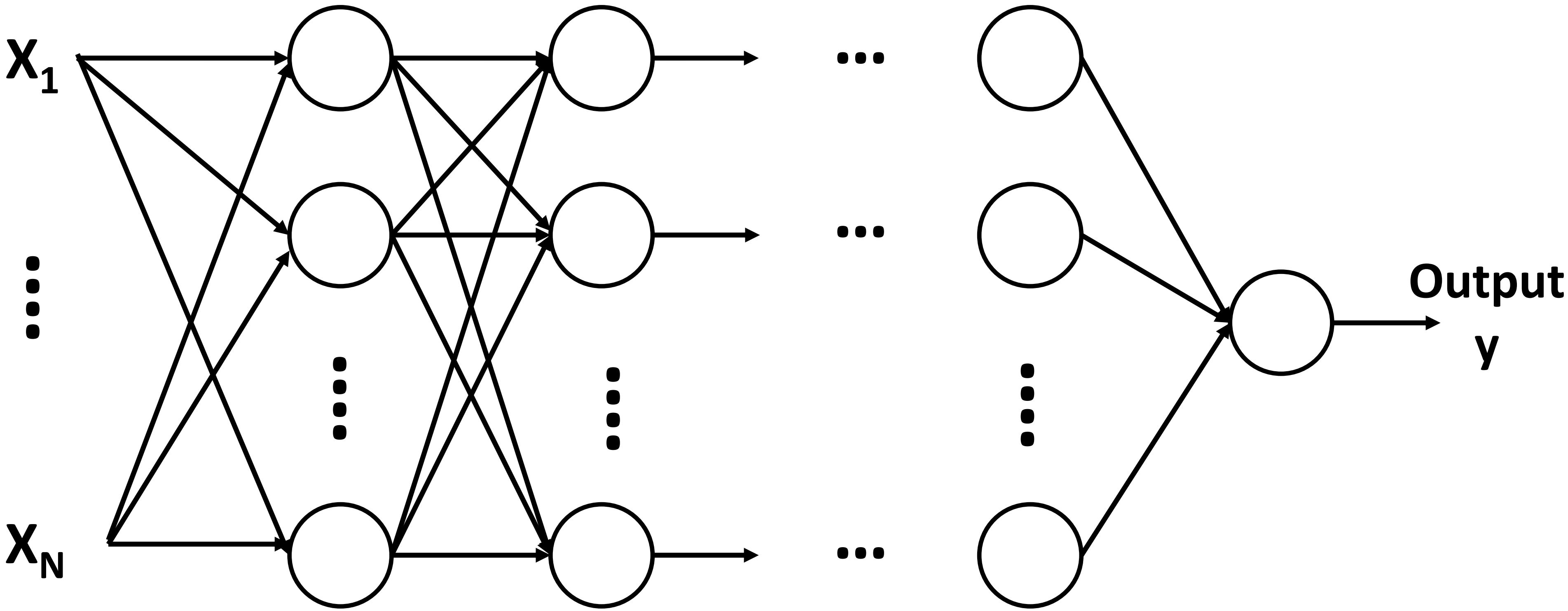


Multi-Layer Networks

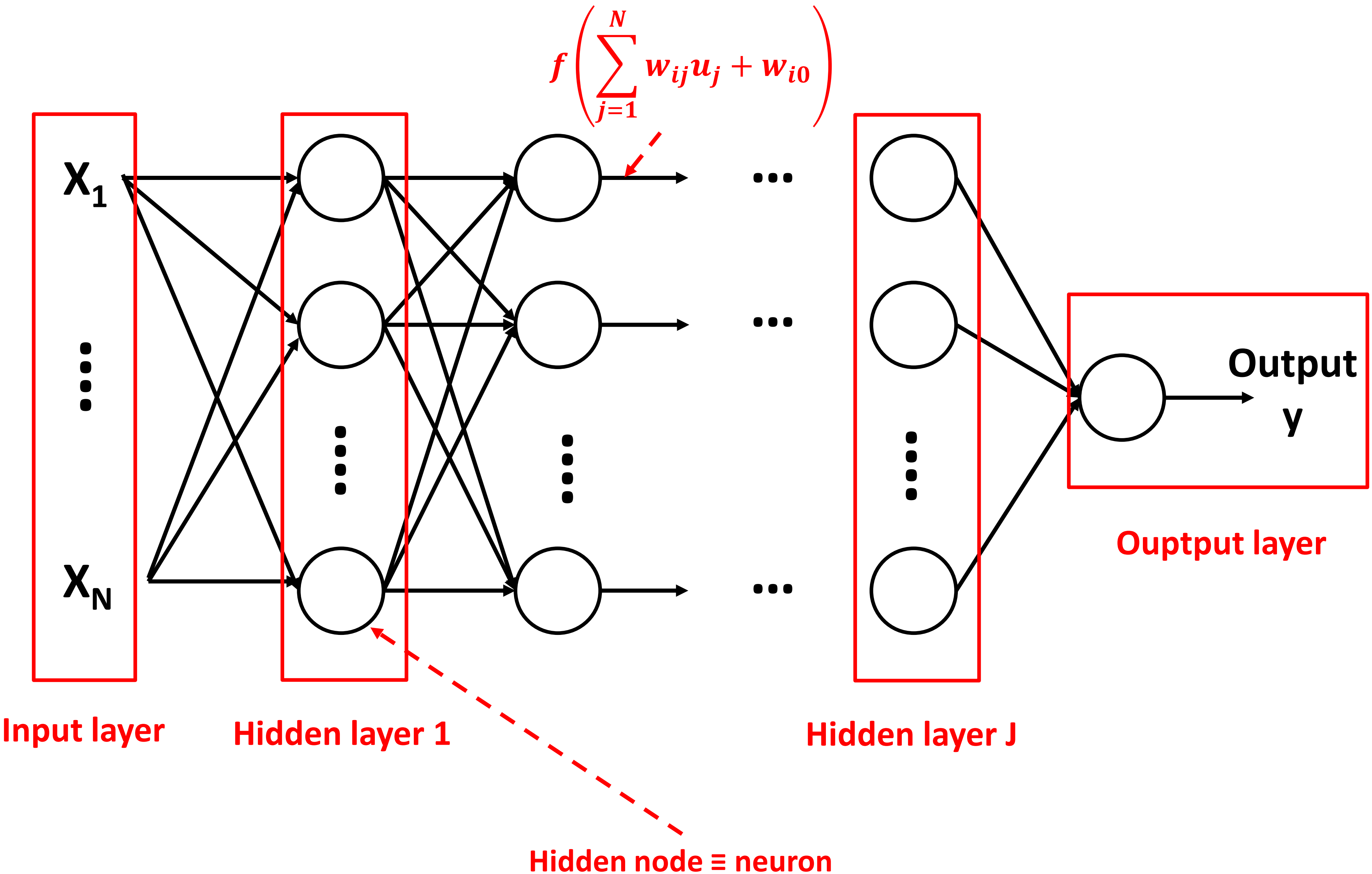


- We need non-linear decision region → use multi-layer network

Multi-Layer Networks



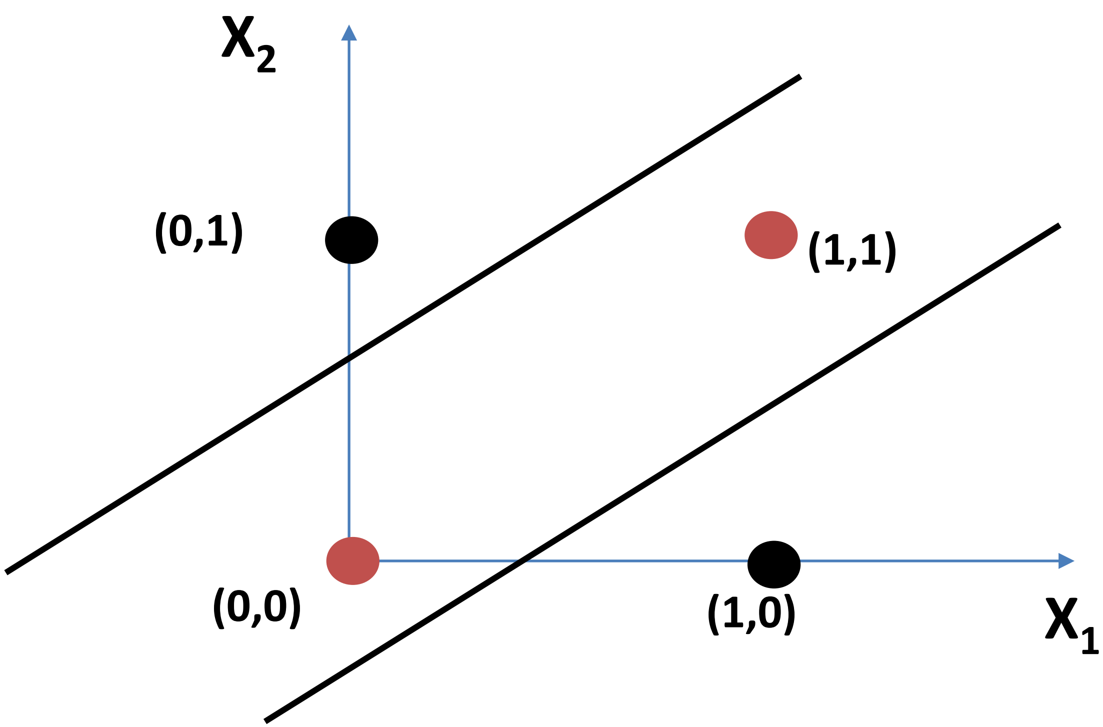
Multi-Layer Networks



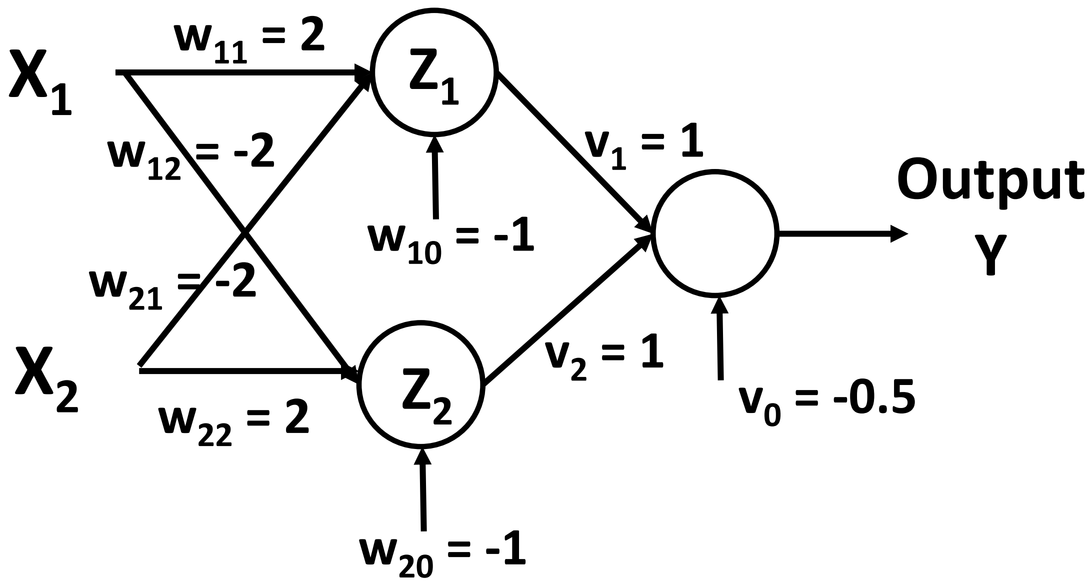
Example on Multi-Layer Networks

- A one-hidden-layer network can implement the XOR problem

- Note: We need two linear classifiers to solve the problem



Example on Multi-Layer Networks



$$Z_1 = f(w_{11}X_1 + w_{21}X_2 + w_{10}) = f(2X_1 - 2X_2 - 1)$$

$$Z_2 = f(w_{12}X_1 + w_{22}X_2 + w_{20}) = f(-2X_1 + 2X_2 - 1)$$

Hidden neurons output

$$Y = f(v_1Z_1 + v_2Z_2 + v_0) = f(Z_1 + Z_2 - 0.5)$$

Network output

f(.) is a step function

Example on Multi-Layer Networks

$$Z_1 = f(w_{11}X_1 + w_{12}X_2 + w_{10}) = f(2X_1 - 2X_2 - 1)$$

$$Z_2 = f(w_{21}X_1 + w_{22}X_2 + w_{20}) = f(-2X_1 + 2X_2 - 1)$$

$$Y = f(v_1Z_1 + v_2Z_2 + v_0) = f(Z_1 + Z_2 - 0.5)$$

X1	X2	Z1	Z2	Y	d target
0	0	0	0	0	0
0	1	0	1	1	1
1	0	1	0	1	1
1	1	0	0	0	0

This network can implement the XOR fn. correctly

Training Multi-Layer Networks

- We have seen that multi-layer networks can implement non-linear fn.'s /decision regions

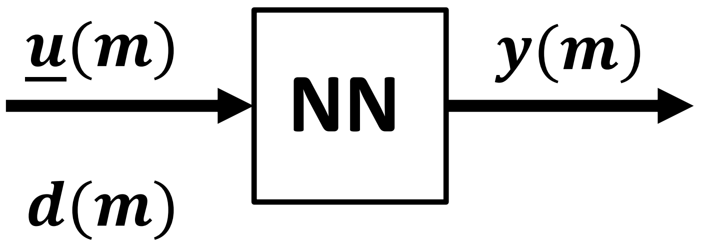
- The powerful feature of multilayer networks (aka. feed-forward networks) is its ability to learn

- Again, we use a training set collected from the problem we wish to solve, i.e., $\underline{u}(m)$ & $d(m)$

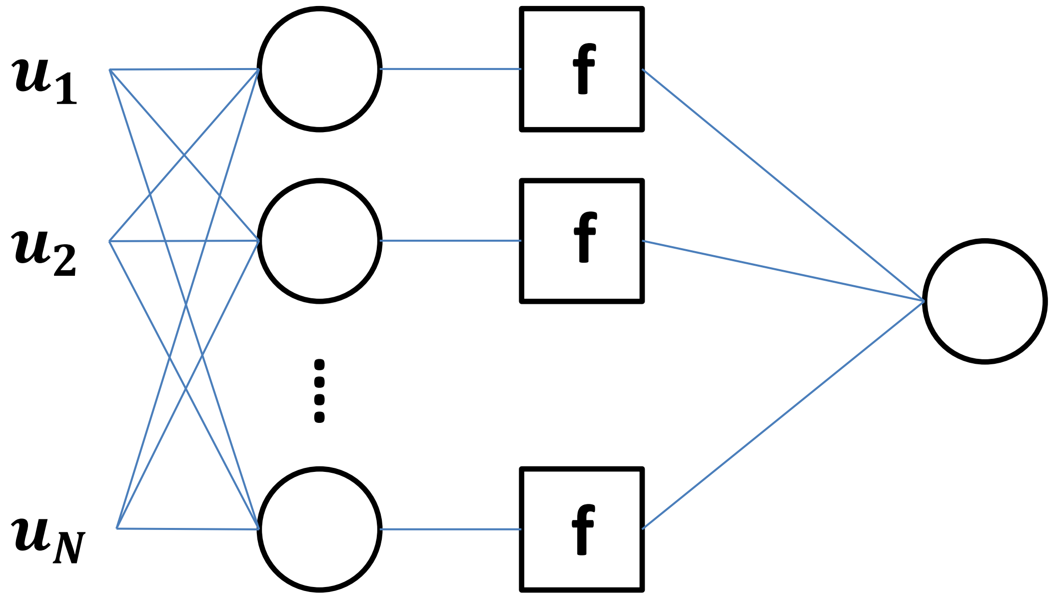
Training Multi-Layer Networks

- The target output $d(m)$ could be the classification of a pattern in case of pattern classification problem

- Alternatively, the target output could be the actual value to be predicted by the NN



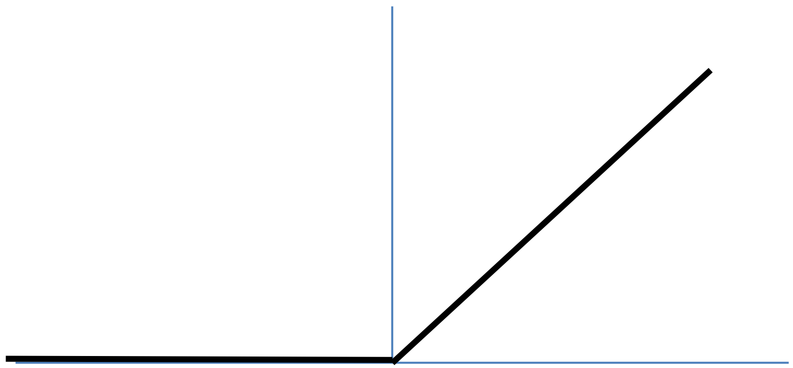
Training Multi-Layer Networks



- We usually use hidden node fn's that are continuous (like the ReLu fn.)

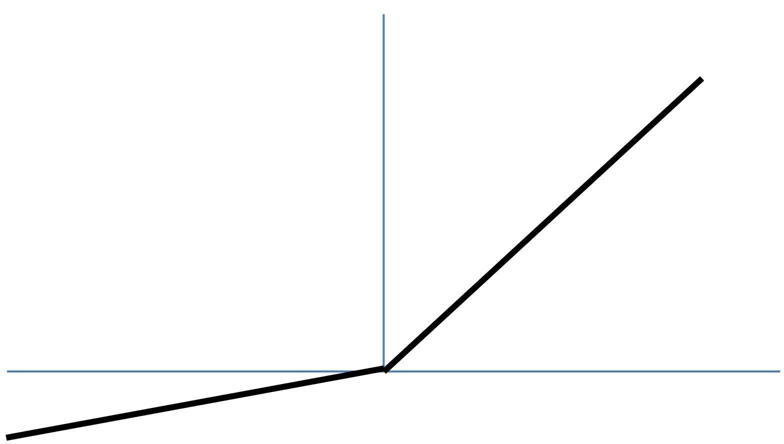
Activation Functions

- ReLU (Rectified Linear Unit)



$$f(x) = \max(x, 0)$$

- Leaky ReLU

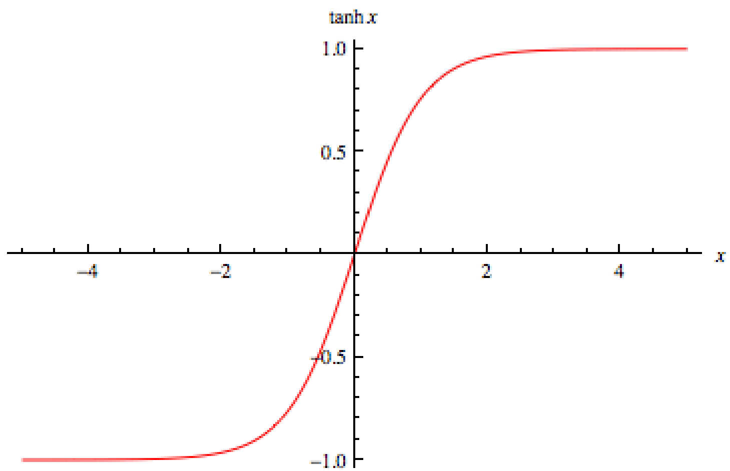


$$f(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{otherwise} \end{cases}$$

$$\text{where } 0 < a < 1$$

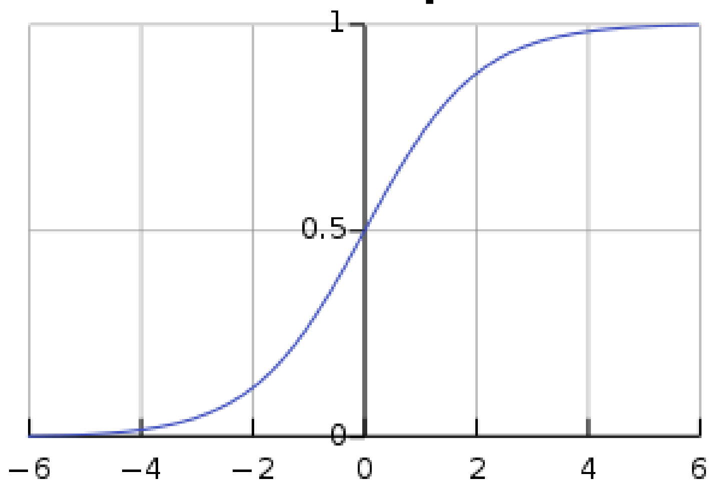
Activation Functions

- Tanh



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

- Sigmoid (used only in o/p layer in binary classification problems)



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

Activation Functions

- ReLU & Leaky ReLU are most famous now
  - Especially with learning from images
  - Faster learning

- Tanh
  - comes next after ReLU
  - More famous with sequence problems, e.g., speech recognition

- Sigmoid
  - Slow learning
  - Used only in output layer in binary classification problems

Supervised Learning

- Learning in case of input/output training examples
- We would like the network to produce an output y(m) when inputting u(m) as close as possible to the target output d(m)
- We define an error function, i.e., cost function, as a measure of how close the network outputs to the target outputs
  - Choice depends on the problem
  - E.g. MSE (Mean Square Error):

$$\boldsymbol{E} = \frac{\boldsymbol{1}}{\boldsymbol{M}} \sum_{\boldsymbol{m}=\boldsymbol{1}}^{\boldsymbol{M}} (\boldsymbol{y}(\boldsymbol{m}) - \boldsymbol{d}(\boldsymbol{m}))^{\boldsymbol{2}}$$

Supervised Learning

- How learning is done is that we adjust the weights in small steps, so that each step decreases the error function a little
- We keep repeating this process until the error reaches its lowest value, i.e., at which y(m) will be as close as possible to d(m)
- Our goal is to minimize E → find weights that give minimum E

Supervised Learning

- Denote $\underline{W} = \begin{bmatrix} w_{11} \\ w_{12} \\ \vdots \end{bmatrix}$ → get min $\mathrm{E}(\underline{W})$
- At the minimum $\frac{\partial E}{\partial w_{11}} = 0$, $\frac{\partial E}{\partial w_{12}} = 0$ …
  - $\frac{\partial E}{\partial \underline{W}} = \begin{bmatrix} \frac{\partial E}{\partial w_{11}} \\ \frac{\partial E}{\partial w_{12}} \\ \vdots \end{bmatrix}$ gradient vector
  - $\frac{\partial E}{\partial \underline{W}} = 0$ → has no analytical or closed form solution, i.e., cannot be algebraically solved

Gradient Descent

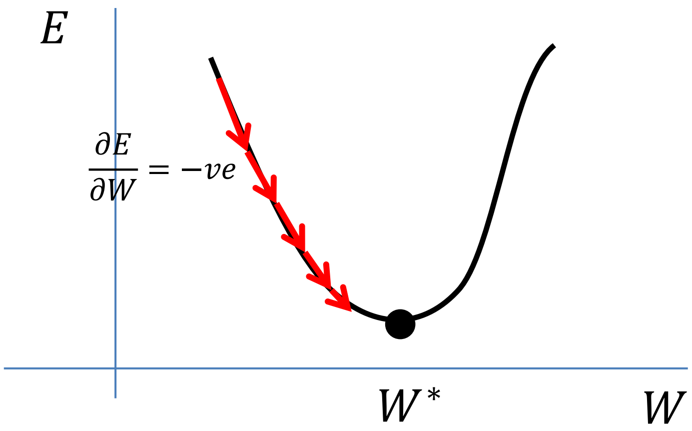
- We use the concept of steepest descent, aka. gradient descent
- It is a general algorithm for minimizing fn's
- We update the weights as:

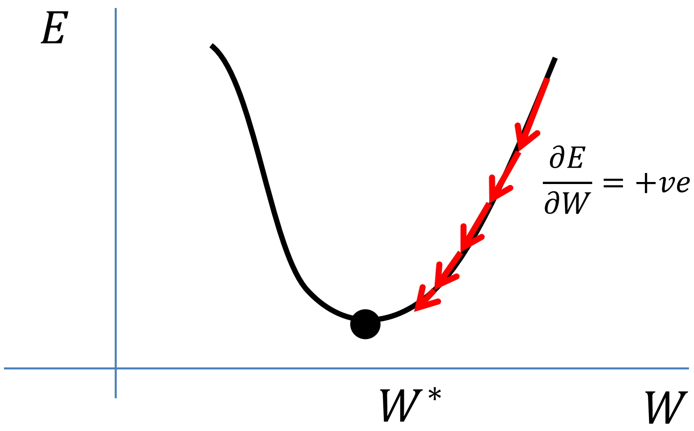
$$\underline{W}(new) = \underline{W}(old) - \alpha \frac{\partial E}{\partial \underline{W}}$$

- α is a constant, i.e., learning rate that determines the size of the move

Gradient Descent

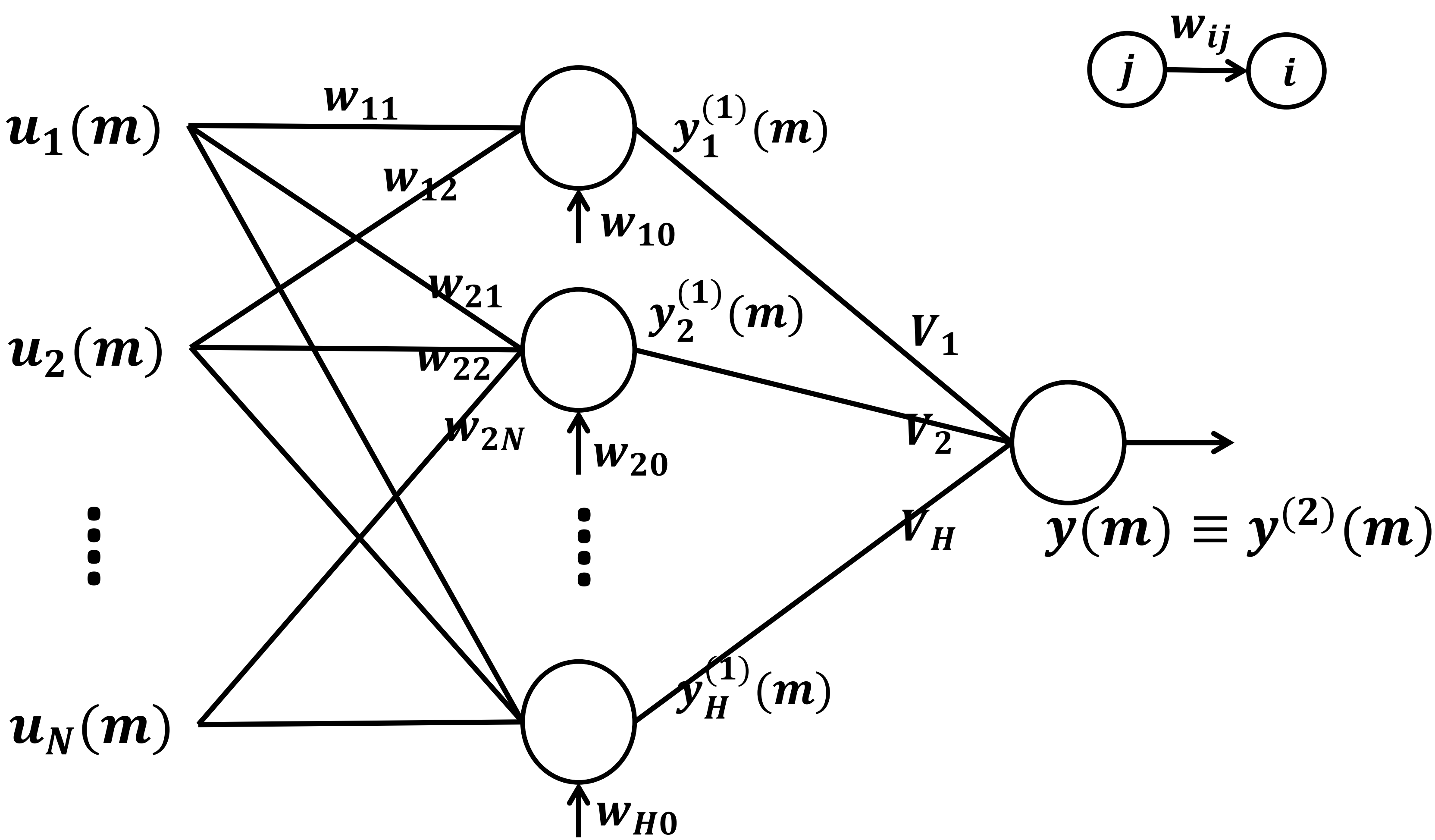
- It can be shown that the negative direction of the gradient gives the steepest descent





- When we approach the min, the steps become very small because close to the min we find $\frac{\partial E}{\partial \underline{W}} \approx 0$

Example



Let $\underline{y}^{(0)}(m) \equiv \underline{u}(m)$

$H \equiv$ no. of hidden nodes

Example

- $E = \sum_{m=1}^{M}\left[y^{(2)}(m) - d(m)\right]^2 = \sum_{m=1}^{M} e^2(m)$

- To get gradient, compute:
  - $\frac{\partial E}{\partial W_{IJ}}$ & $\frac{\partial E}{\partial V_I}$

- $y_i^{(1)}(m) = f\left(\sum_{j=1}^{N} w_{ij}\, u_j(m) + w_{i0}\right)$

- $y^{(2)}(m) = f\left(\sum_{j=1}^{H} V_j y_j^{(1)}(m) + V_0\right)$

Example

- $E=\sum_{m=1}^{M}\left[y^{(2)}(m)-d(m)\right]^2=\sum_{m=1}^{M}e^2(m)$

- $\frac{\partial E}{\partial V_I}=\sum_{m=1}^{M} 2\,e(m)\frac{\partial e(m)}{\partial V_I}$

- $\frac{\partial e(m)}{\partial V_I}=\frac{\partial\left[y^{(2)}(m)-d(m)\right]}{\partial V_I}=\frac{\partial\, y^{(2)}(m)}{\partial V_I}$

$$=f'\left(\sum_{j=1}^{H}V_j y_j^{(1)}(m)+V_0\right)\frac{\partial\left[V_1 y_1^{(1)}(m)+\cdots+V_H y_H^{(1)}(m)+V_0\right]}{\partial V_I}$$

$$=f'\left(\sum_{j=1}^{H}V_j y_j^{(1)}(m)+V_0\right)\frac{\partial V_I y_I^{(1)}(m)}{\partial V_I}=f'\left(\sum_{j=1}^{H}V_j y_j^{(1)}(m)+V_0\right)y_I^{(1)}(m)$$

- $$\boxed{\begin{aligned}\frac{\partial E}{\partial V_I}&=\sum_{m=1}^{M} 2\,e(m)f'\left(\sum_{j=1}^{H}V_j y_j^{(1)}(m)+V_0\right)y_I^{(1)}(m)\\&=\sum_{m=1}^{M} 2\,e(m)f'\left(\sum_{j=1}^{H}V_j y_j^{(1)}(m)+V_0\right)f\left(\sum_{j=1}^{N}w_{Ij}\,u_j(m)+w_{I0}\right)\end{aligned}}$$

Example

- $E = \sum_{m=1}^{M}\left[y^{(2)}(m) - d(m)\right]^2 = \sum_{m=1}^{M} e^2(m)$

- $\frac{\partial E}{\partial w_{IJ}} = \sum_{m=1}^{M} 2\, e(m) \frac{\partial e(m)}{\partial w_{IJ}}$

- $\frac{\partial e(m)}{\partial w_{IJ}} = \frac{\partial\left[y^{(2)}(m) - d(m)\right]}{\partial w_{IJ}} = \frac{\partial\, y^{(2)}(m)}{\partial w_{IJ}}$

$$= f'\left(\sum_{j=1}^{H} V_j y_j^{(1)}(m) + V_0\right) \frac{\partial\left[V_1 y_1^{(1)}(m) + \cdots + V_H y_H^{(1)}(m) + V_0\right]}{\partial w_{IJ}}$$

$$= f'\left(\sum_{j=1}^{H} V_j y_j^{(1)}(m) + V_0\right) \frac{\partial\left[V_1 f\left(\sum_{j=1}^{N} w_{1j}\, u_j(m) + w_{10}\right) + \cdots + V_H f\left(\sum_{j=1}^{N} w_{Hj}\, u_j(m) + w_{H0}\right) + V_0\right]}{\partial w_{IJ}}$$

$$= f'\left(\sum_{j=1}^{H} V_j y_j^{(1)}(m) + V_0\right) \frac{\partial V_I f\left(\sum_{j=1}^{N} w_{Ij}\, u_j(m) + w_{I0}\right)}{\partial w_{IJ}}$$

$$= f'\left(\sum_{j=1}^{H} V_j y_j^{(1)}(m) + V_0\right) V_I f'\left(\sum_{j=1}^{N} w_{Ij}\, u_j(m) + w_{I0}\right) \frac{\partial\left[\sum_{j=1}^{N} w_{Ij}\, u_j(m) + w_{I0}\right]}{\partial w_{IJ}}$$

$$= f'\left(\sum_{j=1}^{H} V_j y_j^{(1)}(m) + V_0\right) V_I f'\left(\sum_{j=1}^{N} w_{Ij}\, u_j(m) + w_{I0}\right) u_J(m)$$

- $\boxed{\frac{\partial E}{\partial w_{IJ}} = \sum_{m=1}^{M} 2\, e(m) f'\left(\sum_{j=1}^{H} V_j y_j^{(1)}(m) + V_0\right) V_I f'\left(\sum_{j=1}^{N} w_{Ij}\, u_j(m) + w_{I0}\right) u_J(m)}$

Acknowledgment

- These slides have been created relying on lecture notes of Prof. Dr. Amir Atiya