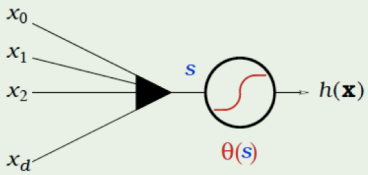
CSE4088 Introduction to Machine Learning

Neural Networks

Slides are adopted from lecture notes of Yaser Abu-Mostafa

Review of last lecture

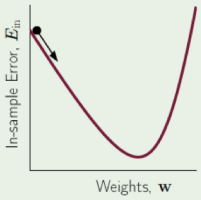
- Logistic regression



- Likelihood measure

$$\prod_{n=1}^{N} P(y_n \mid \mathbf{x}_n) = \prod_{n=1}^{N} \theta(y_n \mathbf{w}^{\mathsf{T}} \mathbf{x}_n)$$

- Gradient descent



- Initialize $\mathbf{w}(0)$
- For $t = 0, 1, 2, \cdots$ [to termination]

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \, \nabla E_{\text{in}}(\mathbf{w}(t))$$

- Return final $\mathbf{w}$

Outline

- Stochastic Gradient Descent
- Neural network model
- Backpropagation model

Stochastic gradient descent

GD minimizes: $E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \underbrace{\mathbf{e}\left(h(\mathbf{x}_n), y_n\right)}_{\ln\left(1+e^{-y_n \mathbf{w}^{\mathsf{T}}\mathbf{x}_n}\right)}$ ⟵ in logistic regression

by iterative steps along $-\nabla E_{\text{in}}$:

$$\Delta\mathbf{w} = -\eta\,\nabla E_{\text{in}}(\mathbf{w})$$

∇E_{in} is based on all examples $(\mathbf{x}_n, y_n)$

"batch" GD

The stochastic aspect

Pick one $(\mathbf{x}_n, y_n)$ at a time. Apply GD to $\mathbf{e}\left(h(\mathbf{x}_n), y_n\right)$

"Average" direction:

$$\mathbb{E}_n\left[-\nabla \mathbf{e}\left(h(\mathbf{x}_n), y_n\right)\right] = \frac{1}{N}\sum_{n=1}^{N} -\nabla \mathbf{e}\left(h(\mathbf{x}_n), y_n\right)$$

$$= -\nabla E_{\text{in}}$$

randomized version of GD

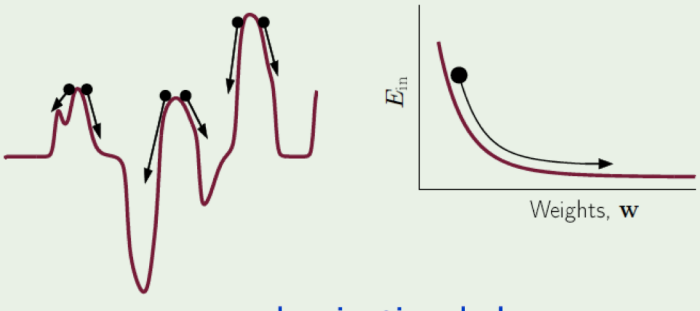
stochastic gradient descent (SGD)

Benefits of SGD

1. cheaper computation
2. randomization
3. simple

Rule of thumb:

$\eta = 0.1$ works



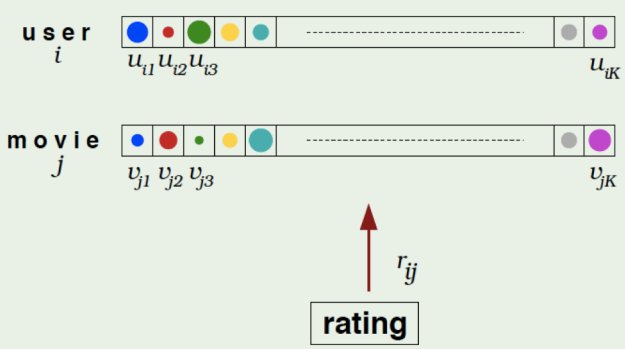
randomization helps

Randomization helps to:
1) Escape shallow minima (e.g. Leftmost minima above)
2) Avoid early termination in flat error regions (right figure)

SGD in action

- Movie ratings

$$e_{ij} = \left(r_{ij} - \sum_{k=1}^{K} u_{ik} v_{jk} \right)^2$$



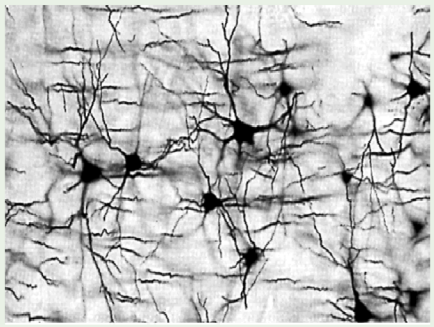
Outline

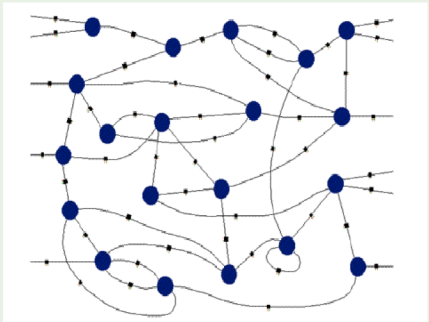
- Stochastic Gradient Descent
- Neural network model
- Backpropagation model

Biological inspiration

- We want to replicate the biological function of learning

biological function ⟶ biological structure



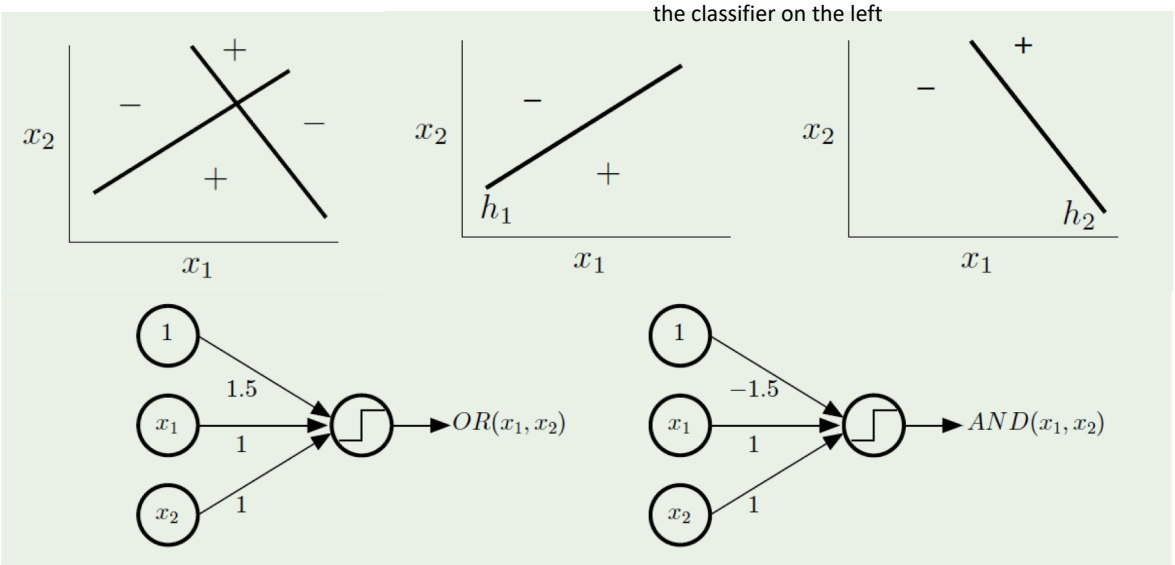


Biological inspiration

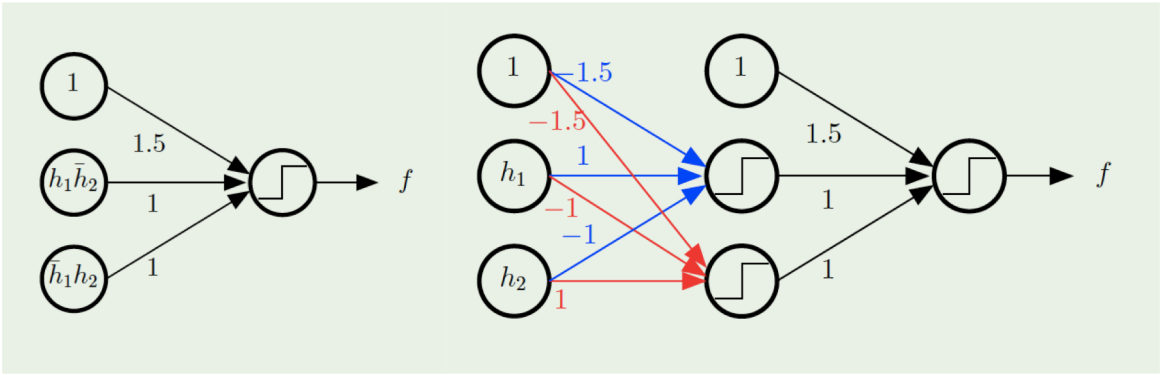
biological function ⟶ biological structure

Combining perceptrons

We can XOR h_1 and h_2 to get the classifier on the left



Creating layers

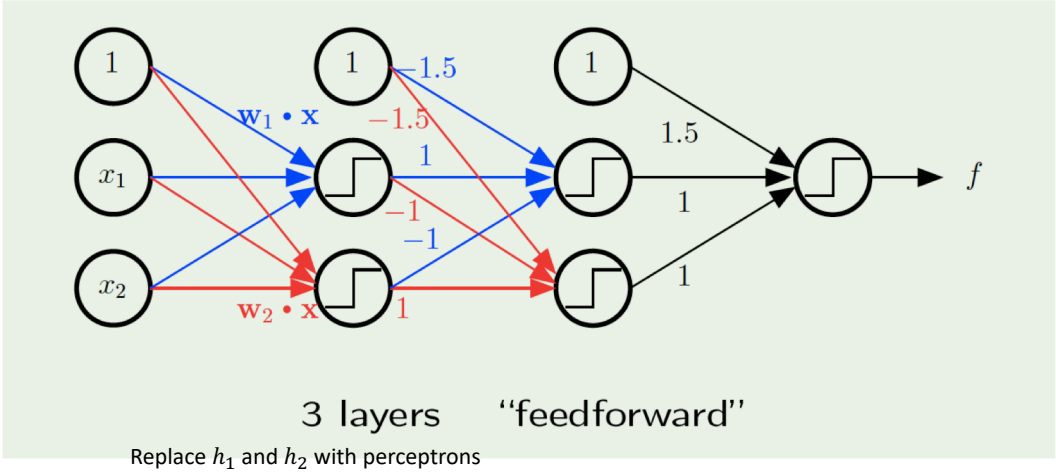


XOR(h_1,h_2)=$OR(h_1\overline{h_2}, \overline{h_1}h_2)$

Blue lines implement $h_1\overline{h_2}$
Red lines implement $\overline{h_1}h_2$

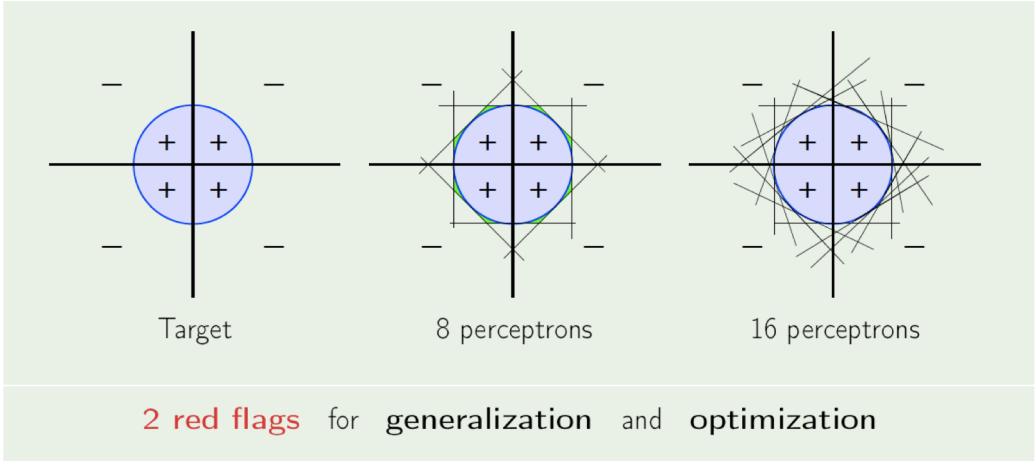
XOR(h_1,h_2)=$OR(h_1\overline{h_2}, \overline{h_1}h_2)$

The multilayer perceptron



Replace h_1 and h_2 with perceptrons that use the inputs .

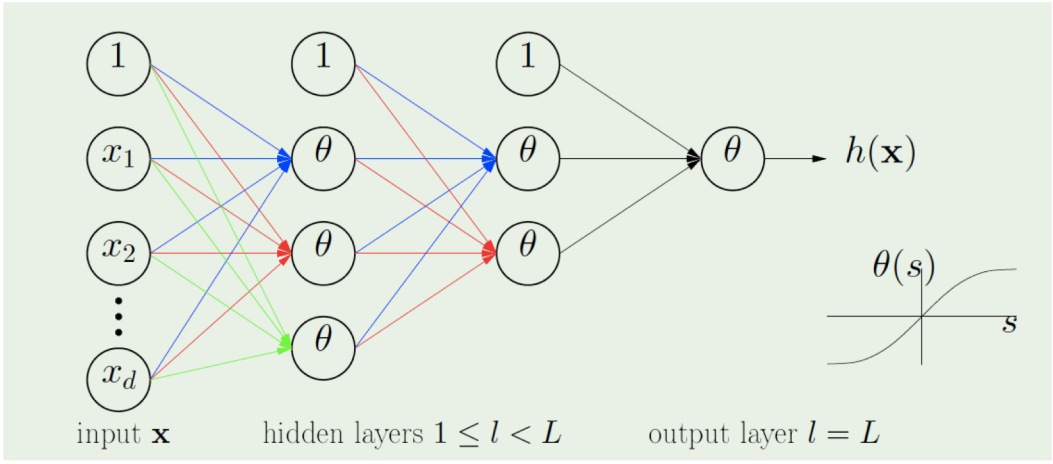
A powerful model



A sufficient number of neurons **can approximate any function**. Two red flags:

- **Generalization**: Since VC dimension increases as the number of neurons increase, we must provide more data.
- **Optimization**: It becomes a difficult combinatorial optimization problem. Solution: we can use a soft threshold and differentiate to optimize (we can use the gradient descent!).

The neural network

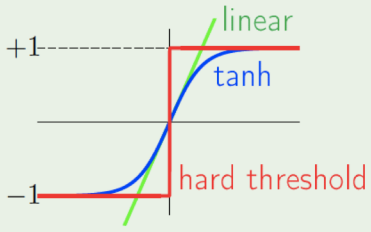


How the network operates

$$w_{ij}^{(l)} \quad \begin{cases} 1 \le l \le L & \text{layers} \\ 0 \le i \le d^{(l-1)} & \text{inputs} \\ 1 \le j \le d^{(l)} & \text{outputs} \end{cases}$$

$$x_j^{(l)} = \theta(s_j^{(l)}) = \theta\left(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)}\, x_i^{(l-1)}\right)$$

Apply $\mathbf{x}$ to $x_1^{(0)} \cdots x_{d^{(0)}}^{(0)} \rightarrow\rightarrow x_1^{(L)} = h(\mathbf{x})$



$$\theta(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

Given the input x, apply the recursive definition through the layers until you get the output h(x)

Outline

- Stochastic Gradient Descent
- Neural network model
- Backpropagation model (to learn the weights)

Applying SGD

All the weights $\mathbf{w} = \{w_{ij}^{(l)}\}$ determine $h(\mathbf{x})$

Error on example $(\mathbf{x}_n, y_n)$ is

$$\mathbf{e}\left(h(\mathbf{x}_n), y_n\right) = \mathbf{e}(\mathbf{w})$$

To implement SGD, we need the gradient

$$\nabla \mathbf{e}(\mathbf{w}): \quad \frac{\partial\, \mathbf{e}(\mathbf{w})}{\partial\, w_{ij}^{(l)}} \quad \text{for all} \quad i, j, l$$

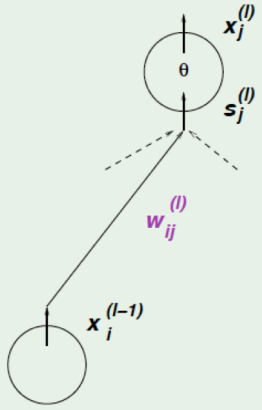
Computing $\frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, w_{ij}^{(l)}}$

We can evaluate $\frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, w_{ij}^{(l)}}$ one by one: analytically or numerically

A trick for efficient computation:

$$\frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, w_{ij}^{(l)}} = \frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, s_j^{(l)}} \times \frac{\partial\, s_j^{(l)}}{\partial\, w_{ij}^{(l)}}$$

We have $\frac{\partial\, s_j^{(l)}}{\partial\, w_{ij}^{(l)}} = x_i^{(l-1)}$ We only need: $\frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, s_j^{(l)}} = \delta_j^{(l)}$



δ for the final layer

$\delta_j^{(l)} = \frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, s_j^{(l)}}$

For the final layer $l = L$ and $j = 1$:

$$\delta_1^{(L)} = \frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, s_1^{(L)}}$$

Output $h(x_n)$

Target

$$\mathrm{e}(\mathbf{w}) = (x_1^{(L)} - y_n)^2$$

$$x_1^{(L)} = \theta(s_1^{(L)})$$

$$\theta'(s) = 1 - \theta^2(s) \quad \text{for the tanh}$$

$$\delta_1^{(L)} = \frac{\partial e(w)}{\partial s_1^{(L)}} = \frac{\partial e(w)}{\partial x_1^{(L)}} \times \frac{\partial x_1^{(L)}}{\partial s_1^{(L)}}$$
$$= 2\left(x_1^{(L)} - y_n\right) \times \theta'\left(s_1^{(L)}\right)$$
$$= 2\left(x_1^{(L)} - y_n\right)\left(1 - \theta^2\left(s_1^{(L)}\right)\right)$$

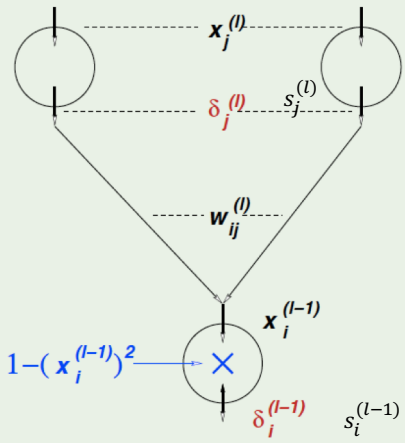
Back propagation of δ

$$\delta_i^{(l-1)} = \frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, s_i^{(l-1)}}$$

$$= \sum_{j=1}^{d^{(l)}} \frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, s_j^{(l)}} \times \frac{\partial\, s_j^{(l)}}{\partial\, x_i^{(l-1)}} \times \frac{\partial\, x_i^{(l-1)}}{\partial\, s_i^{(l-1)}}$$

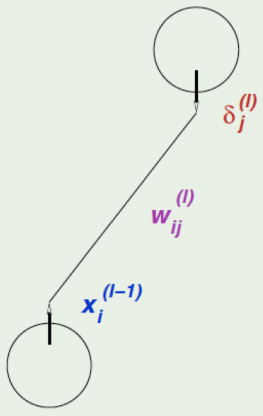
$$= \sum_{j=1}^{d^{(l)}} \delta_j^{(l)} \times w_{ij}^{(l)} \times \theta'(s_i^{(l-1)})$$

$$\delta_i^{(l-1)} = (1 - (x_i^{(l-1)})^2) \sum_{j=1}^{d^{(l)}} w_{ij}^{(l)}\, \delta_j^{(l)}$$



Backpropagation algorithm

1: Initialize all weights $w_{ij}^{(l)}$ at random
2: for $t = 0, 1, 2, \ldots$ do
3: Pick $n \in \{1, 2, \cdots, N\}$
4: *Forward:* Compute all $x_j^{(l)}$
5: *Backward:* Compute all $\delta_j^{(l)}$
6: Update the weights: $w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta\, x_i^{(l-1)} \delta_j^{(l)}$
7: Iterate to the next step until it is time to stop
8: Return the final weights $w_{ij}^{(l)}$

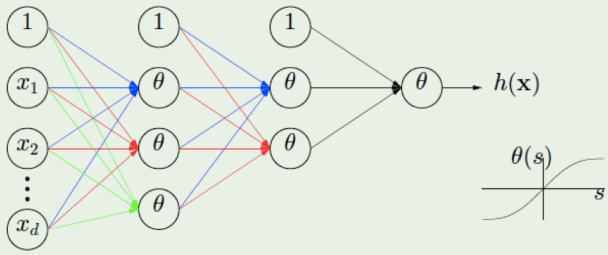


Note: If we initialize the weights to zero, we will be not be able to update the weights.

Final remark: hidden layers

learned nonlinear transform

interpretation?



Note: Hidden layers are performing nonlinear transformations on the outputs of the previous layer. Hence, they are like non-linear features (of features). But they are LEARNED in the neural network.

-- Can we interpret the outputs of the hidden layers? --> Not very easy.

-- The goal of a neural network is to produce the right hypothesis. Not to explain to a human what the hidden layers are doing.