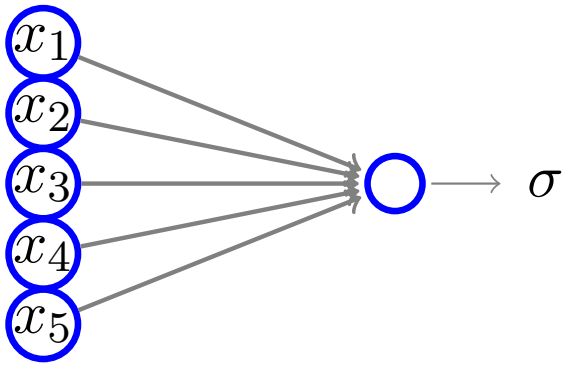
Introduction to Machine Learning (67577)
Lecture 7

School of CS and Engineering,
The Hebrew University of Jerusalem

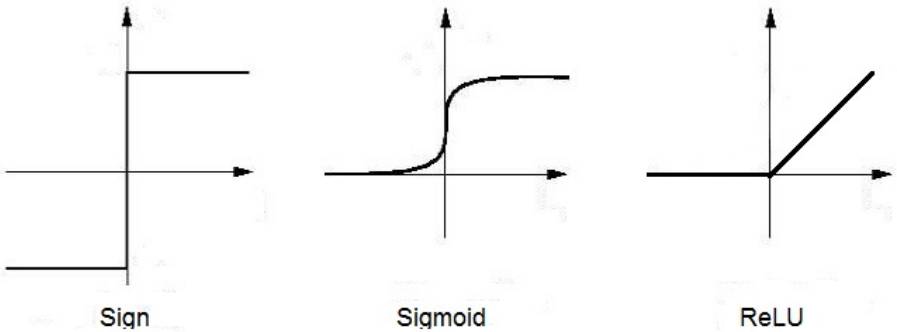
Heuristics: Neural Networks, Decision Trees and Nearest Neighbour

Outline

A Single Artificial Neuron

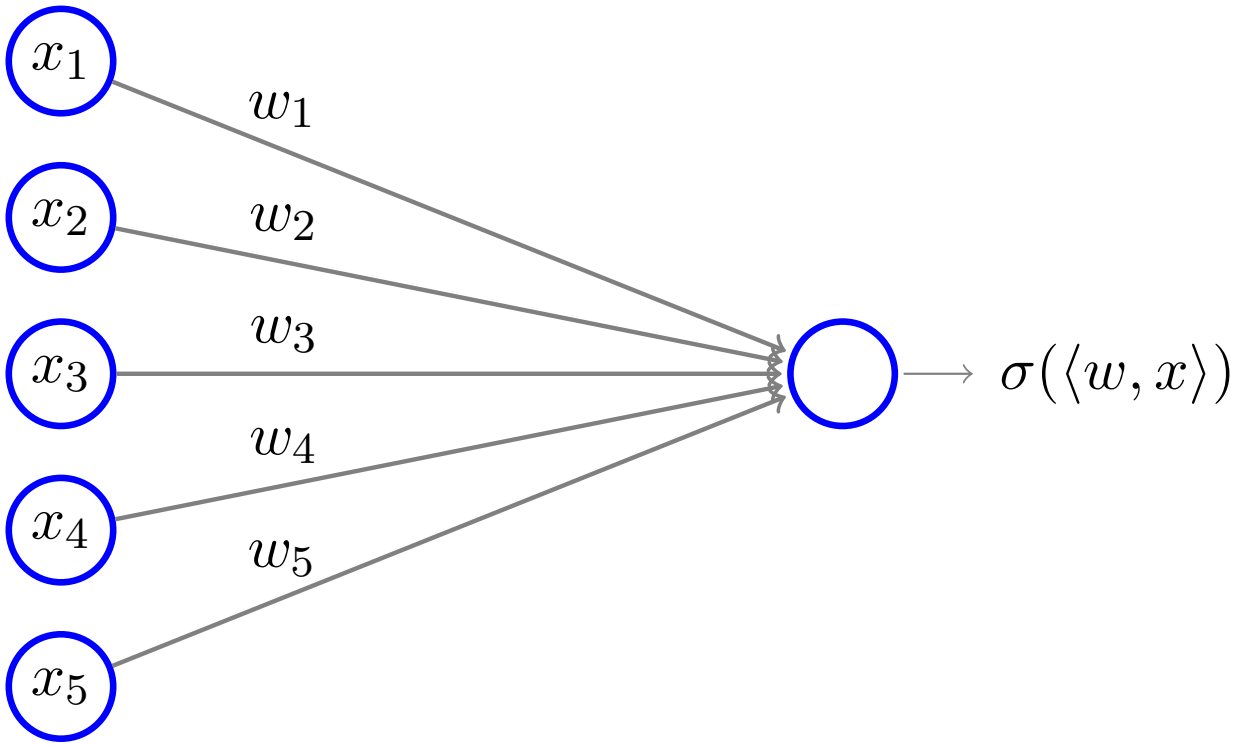


- **Neuron:** Computational device for computing functions $\mathbb{R}^n \to \mathbb{R}$.
- Composed of
  - n input nodes.
  - Output node.
  - "Wires" from the input to the output node.
  - Activation function $\sigma : \mathbb{R} \to \mathbb{R}$. E.g:

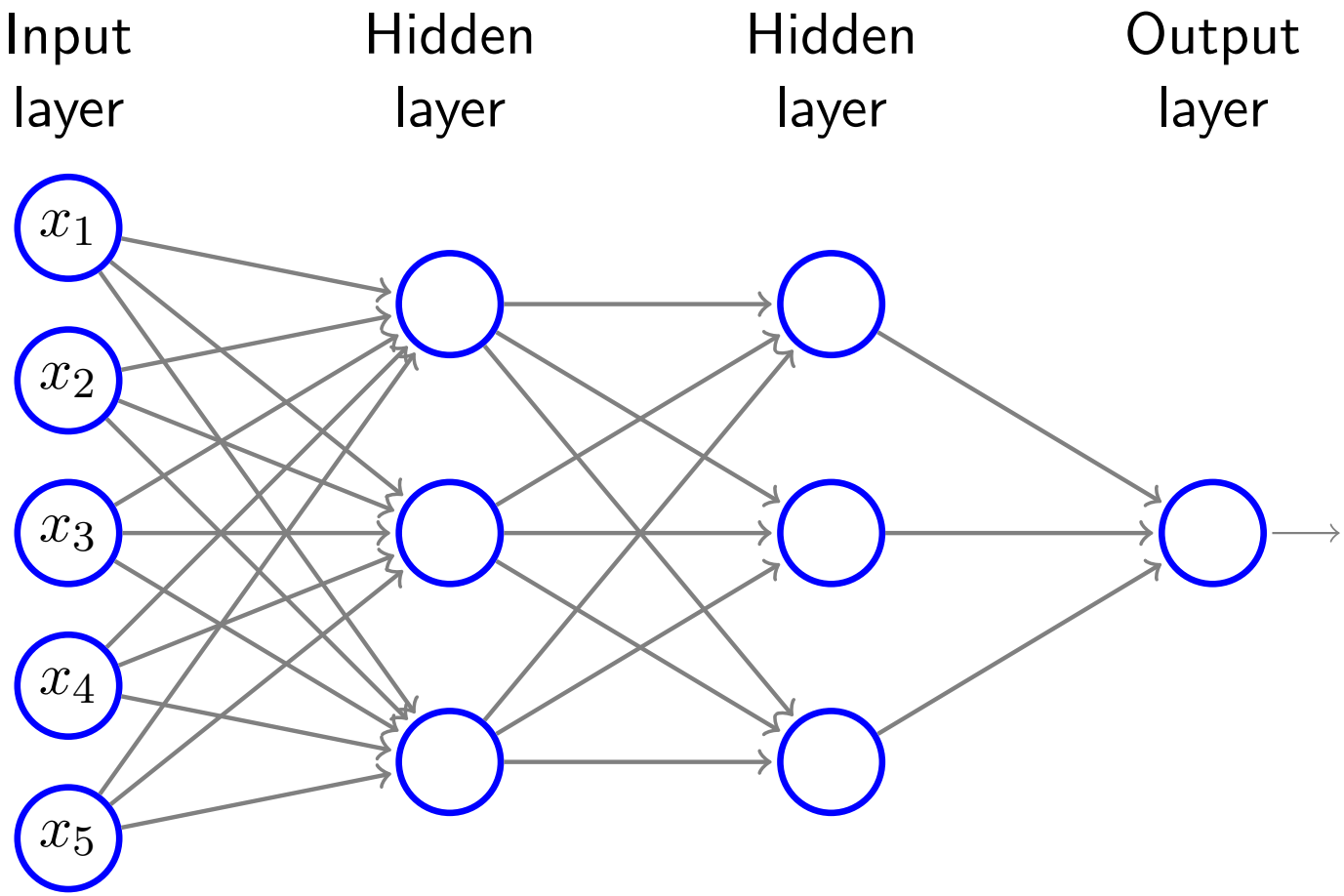


A Single Artificial Neuron

- Given weights $\mathbf{w} \in \mathbb{R}^n$, computes the function $\mathbf{x} \mapsto \sigma(\langle \mathbf{w}, \mathbf{x} \rangle)$.

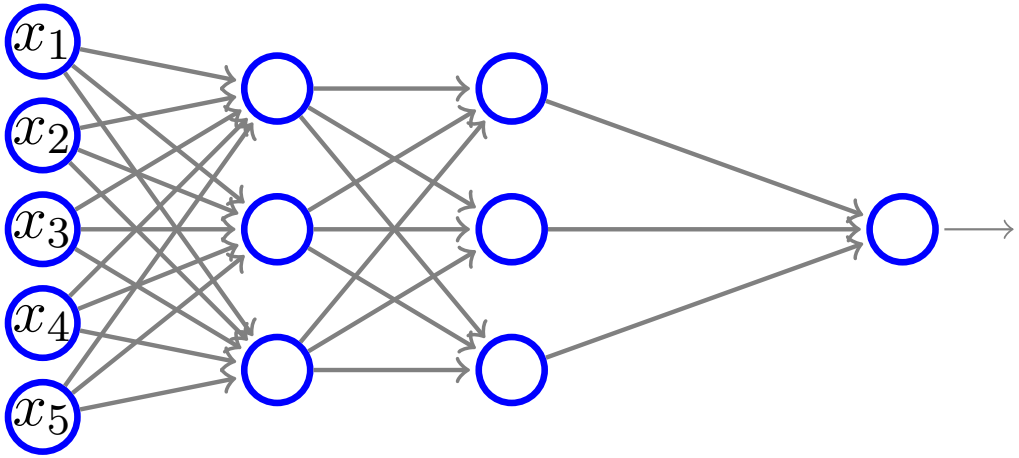


Neural Networks



- A neural network is obtained by connecting many neurons together
- We focus on layered feedforward networks with single output

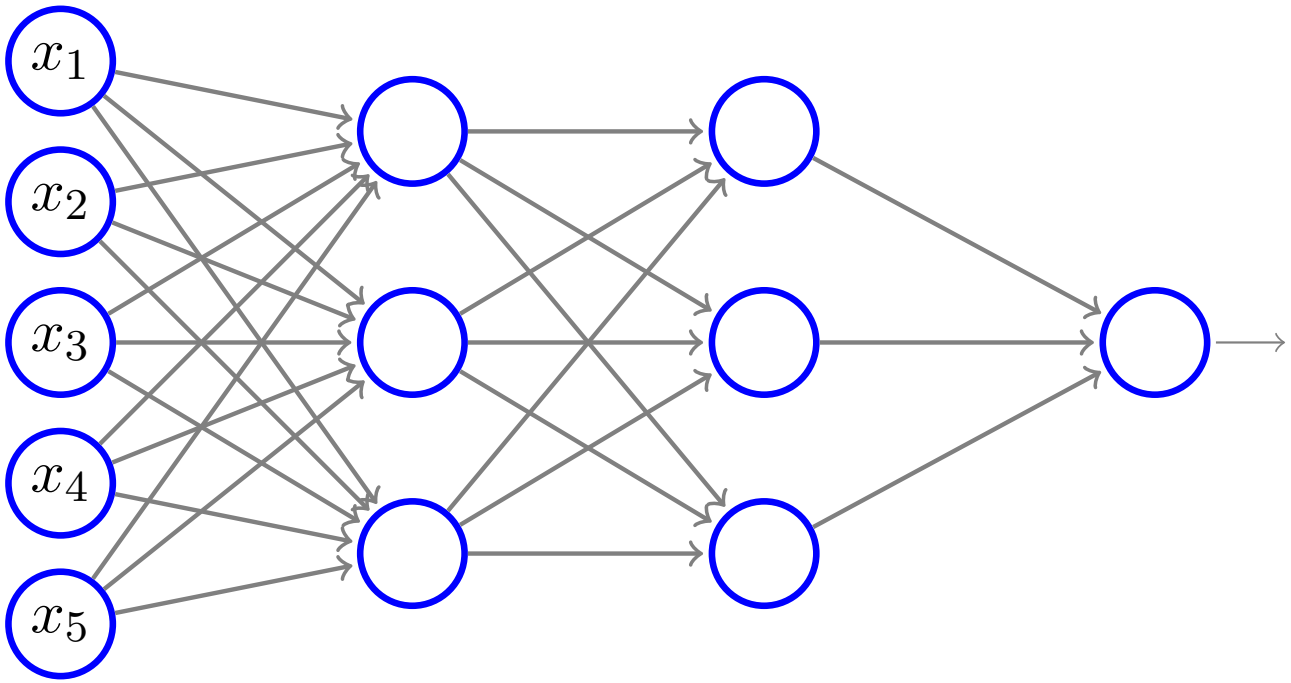
Neural Networks – Definitions



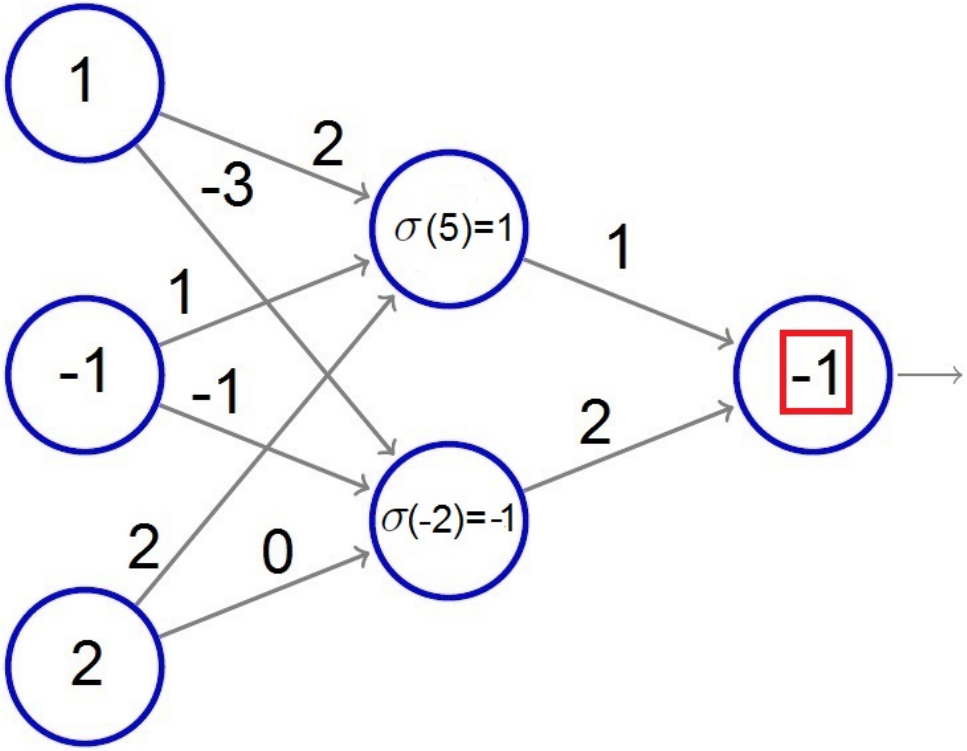
- **Neural Net:** A pair $\mathcal{N} = (G, \sigma)$ where
  - $\sigma : \mathbb{R} \to \mathbb{R}$ is a function called the activation function
  - $G = (V, E)$ is a depth d layered directed graph with single output. I.e.:
    - $V = \uplus_{t=0}^{d} V_t$
    - All edges are from V_{t-1} to V_t
    - A single output node: $|V_d| = 1$
- **Terminology:**
  - The size of the net is $|\mathcal{N}| := |E|$
  - The inputs nodes are V_0
  - The inputs size is $n = |V_0|$

Example – Fully Connected Neural Net

- **Fully connected net** with input size n, width l, depth d, and activation function σ is the net $\mathcal{N}_{l,d,\sigma} = (G, \sigma)$ where
  - $V = \dot{\cup}_{t=0}^{d} V_t$
  - $|V_t| = l$, for all $1 \leq t \leq d-1$
  - $|V_0| = n$
  - E consists of all edges from V_{t-1} to V_t.
- $\mathcal{N}_{3,3,\sigma}$:



- Fix a net $\mathcal{N}$ with input size n
- Given weights $w : E \to \mathbb{R}$, the net computes $h_{\mathcal{N},w} : \mathbb{R}^n \to \mathbb{R}$
- $h_{\mathcal{N},w}(x)$ is the value obtained by propagating x thru the net (in the following example $\sigma = \mathrm{sign}$)



Learning with Neural Networks

- Fix a neural network $\mathcal{N}$
- Learning algorithms try to find a good hypothesis $h_{\mathcal{N},w}$.
- To analyse such algorithms we let

$$\mathcal{H}_{\mathcal{N}} \;=\; \{h_{\mathcal{N},w} \;:\; w : E \to \mathbb{R}\} \;.$$

- We can now study
  - Estimation error (sample complexity)
  - Approximation error (expressivenss)
  - Optimization error (computational complexity)

Outline

Sample Complexity

- Fix a net $\mathcal{N} = (G, \sigma)$.
- Each $h_{\mathcal{N},w}$ is specified by $|\mathcal{N}|$ parameters
- **Corollary:** If weights are represented using k bits, $\mathrm{VC}(\mathcal{H}_{\mathcal{N}}) \leq k \cdot |\mathcal{N}|$
- **Theorem:** For $\sigma = \mathrm{sign}$, $\mathrm{VC}(\mathcal{H}_{\mathcal{N}}) \leq C \cdot |\mathcal{N}| \cdot \log(|\mathcal{N}|)$
- **Theorem:** For $\sigma = \mathrm{SIGMOID}$, $\mathrm{VC}(\mathcal{H}_{\mathcal{N}}) \leq C \cdot |\mathcal{N}|^2$
- Sample complexity decreases when using various regularizations

Outline

What can be expressed with neural networks?

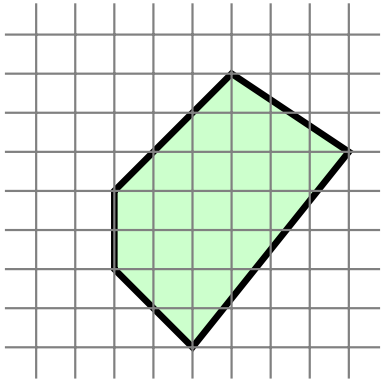
- For simplicity, focus on boolean inputs and $\sigma = \mathrm{sign}$.
- What functions from $\{\pm 1\}^n$ to $\{\pm 1\}$ are in $\mathcal{H}_{\mathcal{N}}$?
- **Theorem:** $\mathcal{H}_{\mathcal{N}_{2^n,2,\mathrm{sign}}}$ contains all functions from $\{\pm 1\}^n$ to $\{\pm 1\}$!
- **Theorem:** If $\mathcal{H}_{\mathcal{N}}$ contains all functions from $\{\pm 1\}^n$ to $\{\pm 1\}$ then $|\mathcal{N}|$ is exponential in n
- What functions can be implemented by small $\mathcal{N}$?
- **Theorem:** $\mathcal{H}_{\mathcal{N}_{T,T,\mathrm{sign}}}$ contains all functions computed in time T!

The ultimate hypothesis class! (ignoring training time)

If we only care about functions computed in time T, we can use size T^3 neural network, and the sample complexity is also bounded by T^3!

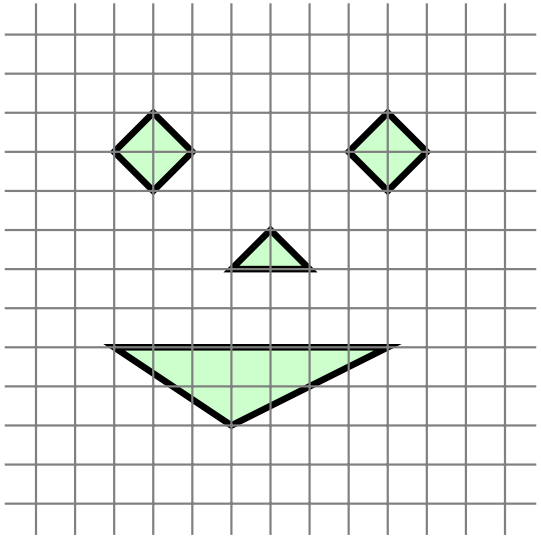
Geometric Intuition

- 2 layer networks can express intersection of halfspaces



Geometric Intuition

- 3 layer networks can express unions of intersection of halfspaces



Outline

Runtime of learning neural networks

ERM problem:

$$\mathrm{ERM}(S) = \operatorname*{argmin}_{h \in \mathcal{H}_{\mathcal{N}}} L_S(h) = \operatorname*{argmin}_{w} L_S(h_{\mathcal{N},w})$$

- **Theorem:** NP hard to implement ERM already for $\mathcal{N}_{2,2,\mathrm{sign}}$.
- Maybe some other algorithm?
- **Theorem:** Probably hard to return h with $L_{\mathcal{D}}^{0-1}(h) \leq \frac{1}{2} - \frac{1}{n^{20}}$ even when $L_{\mathcal{D}}^{0-1}(\mathcal{H}_{\mathcal{N}_{\log^2(n),2,\mathrm{sign}}}) = 0$

How to train neural network?

- So, neural networks can form an excellent hypothesis class, but it is intractable to train it.
- Can be still trained using heuristics

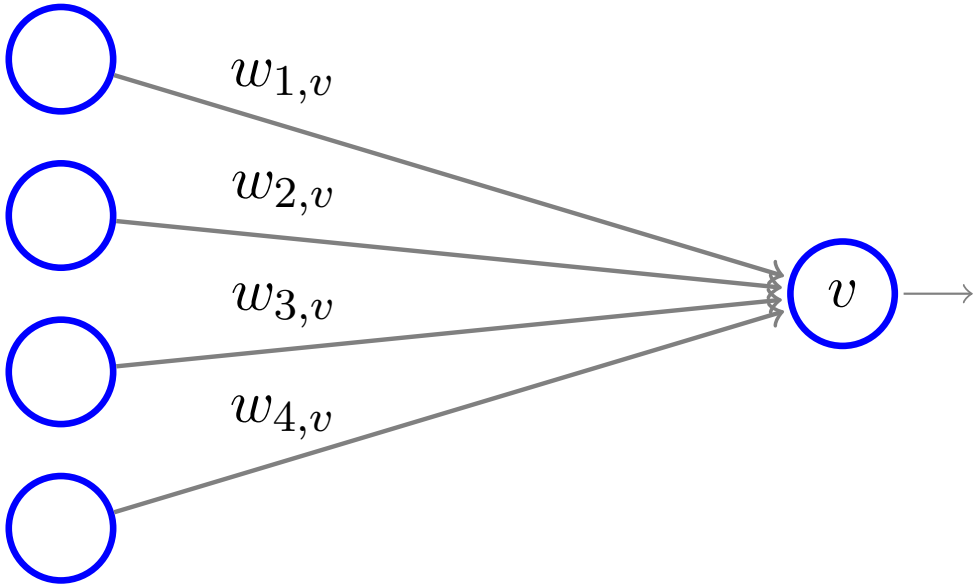
The basic heuristic

1. Start with random weights w
2. At each step, change w a bit, in a direction that decreases the loss.

- Not convex! No guarantees! Can take a long time!
- But, often still works fine!
- Many ways to implement the basic heuristic.
- We will see a few popular ones next.

Step I: Initializing the weight

- Fix neuron v with incoming weights $w_{1,v}, \ldots, w_{k,v}$



- **Rule of thumb:** Chose the weights independently in a way that

$$E\left[\sum_{i=1}^{k} w_{i,v}^2\right] = 1$$

E.g., $w_{i,v} \sim U\left[-\sqrt{\frac{3}{k}}, \sqrt{\frac{3}{k}}\right]$

Step II: Locally improving w – SGD for Neural Networks

- Fix a loss $l : \mathbb{R} \times Y \to \mathbb{R}_+$ (say, the hinge loss)
- For $(x, y) \in X \times Y$, let $l_{(x,y)}(w) = l(h_{\mathcal{N},w}(x), y))$
- For a sub-sample $S' \subset S$ let $L_{S'}(w) = \frac{1}{|S'|} \sum_{(x,y)\in S'} l_{(x,y)}(w)$
- Update rule:

$$w_{t+1} = w_t - \eta_t \nabla L_{S'}(w)$$

  where:
  - η_t is learning rate (e.g. $\eta_t = 0.01$ for all t)
  - S' is a random subset of the training examples (called a "minibatch")
    - GD: $S' = S$
    - SGD: $|S'| = 1$
- It is left to show how to calculate the gradient $\nabla l_{(x,y)}(w)$

Back-Propagation

- Recall that $\nabla l_{(x,y)}(w) = \left(\frac{\partial l_{(x,y)}}{\partial w_e}(w)\right)_{e \in E}$
- Let $e \in E$ be an edge whose output neuron is in the i'th layer.
- Fix (x, y) and all weights but w_e
- Let $l_e(t) = l_{(x,y)}(w^e|t)$, where $w^e|t$ is obtained by changing w_e to t.
- We have $\frac{\partial l_{(x,y)}}{\partial w_e}(w) = l'_e(w_e)$. Moreover

$$l_e(t) = l_y \circ h_d \circ \ldots, h_{i+1} \circ h_i(t)$$

  Where
  - $h_i(t)$ is the vector with the values of the neurons in the i'th layer.
  - For $j > i$, h_j computes the j'th layer given the $(j-1)$'th layer.
  - $l_y(\hat{y}) = l(\hat{y}, y)$
- Hence, $\frac{\partial l_{(x,y)}}{\partial w_e}(w)$ can be calculated using the (multivariate) chain rule:

$$(f \circ g)'(x) = f'(g(x))g'(x)$$

- The back-propagation algorithm is an efficient way to do that.
- Details in the Tirgul

Neural Networks: Current Trends

- *Design Issues:*
  - Deep nets
  - ReLU activation: $\sigma(a) = \max\{0, a\}$
  - Very large networks: More parameters than examples!
  - May cause overfitting. Partially avoided by various regularizations
- *Algorithmic Issues:*
  - Dropout: Some neurons are "muted" at random during training
  - Training on GPU

Outline

Summary

- Neural nets can be used to construct the ultimate hypothesis class
- Computationally, it's impossible to train neural networks
- ... but, empirically, it works reasonably well
- Leads to state-of-the-art on many real world problems

Historical Remarks

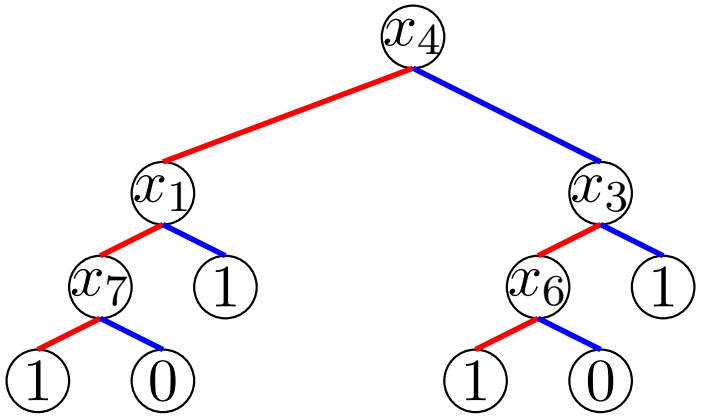
- 1940s-70s:
  - Inspired by learning/modeling the brain (Pitts, Hebb, and others)
  - Perceptron Rule (Rosenblatt)
  - Backpropagation (Werbos 1975)
- 1980s – early 1990s:
  - SGD (Bottou)
  - Initial empirical success
- 1990s-2000s:
  - Lost favor to SVM and Boosting
- 2006 –:
  - Computational advances and several new tricks allow training HUGE networks. Empirical success leads to renewed interest

A fundamental question

When does it work and why ?

Outline

Learning Decision Trees



- A decision tree over $\{\pm 1\}^n$ is defined by a
  - Rooted binary tree in which every node has 2 or 0 children
  - Internal nodes are labelled by $x_i, \ 1 \leq i \leq n$
  - Leafs are labelled by 0 or 1
  - Edges are labelled by -1 or 1
- A decision tree T naturally defines a function $h_T : \{\pm 1\}^n \to \{0, 1\}$:
  - To calculate $h_T(x)$, start from the root and traverse the tree according to x until a leaf is reached. $h_T(x)$ is the label of that leaf
- We will study algorithms that learn a decision tree

Sample Complexity

Theorem: Let $\mathcal{T}_k$ be the class of decision trees with $k \leq 2^n$ leaves. Then,

$$k \leq \mathrm{VC}(\mathcal{T}_k) \leq 2k\lceil \log_2(4nk) \rceil$$

Proof:

- $\mathrm{VC}(\mathcal{T}_k) \leq 2k\lceil \log_2(4nk) \rceil$:
  - Each decision tree can be described using $2k\lceil \log_2(4nk) \rceil$ bits:
  - A tree with k leaves have $2k-1$ nodes (prove it by induction!).
  - Each node can be described by $\lceil \log_2(n+2) + \log_2(2k) \rceil$ bits, encoding
    - A description (node of the form '$x_i = 1$?' / leaf with value $0/1$)
    - The identity of its parent
- $\mathrm{VC}(\mathcal{T}_k) \geq k$:
  - *Every* $A \subset \{\pm 1\}^n$ of size k is shattered! (Targil)

Computational Complexity

- NP hard problem ...
- Tree algorithms follow greedy heuristics

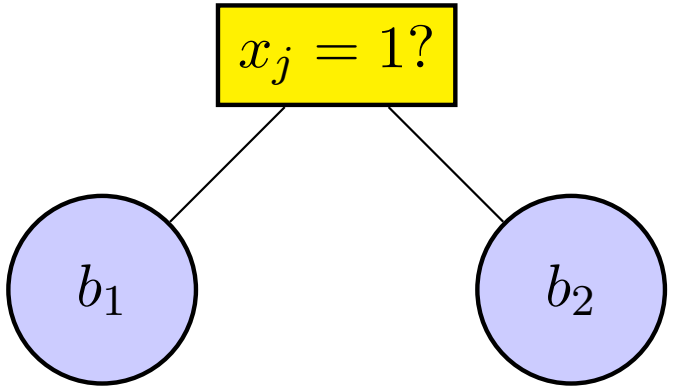
The basic heuristic

1. Start with a very simple tree (say, a single leaf)
2. At each step, improve the tree by a small modification

A Basic Greedy Tree Algorithm

INPUT: training set $S \subset \{\pm 1\}^n \times \{\pm 1\}$

- Start with a single leaf
- At each step, replace one of the leaves by a tree of the form:



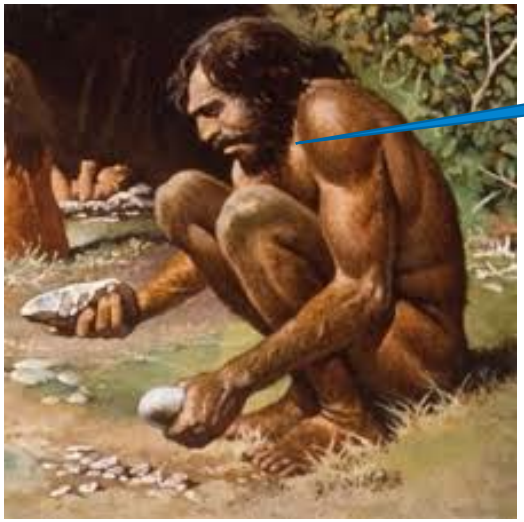
  for $b_1, b_2 \in \{\pm 1\}$ and $j \in [n]$, in a way that minimizes the loss among all such replacements.
- Stop when no improvement is possible

Variants and Extensions

- Algorithms differ by the measures they greedily optimize
  - The presented algorithm uses the empirical loss
  - Other algorithms optimize different measures
- Greedy algorithms might produce large trees. Can be tackled by:
  - Early stopping
  - Pruning
- Extensions to real-valued features and real-valued/multiclass output

Outline

k-Nearest Neighbor

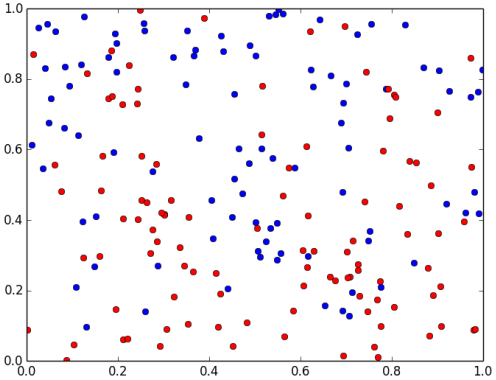


"Things that look alike must be alike"

- Memorize the training set $S = (x_1, y_1), \ldots, (x_m, y_m)$
- Given new x, find the k closest points in S and return majority vote among their labels

- Very simple
- Works well when $\mathcal{X}$ is low dimensional.
- Problematic in high dimension.

k-Nearest Neighbor: Bias-Complexity Tradeoff

S= 

h^*=