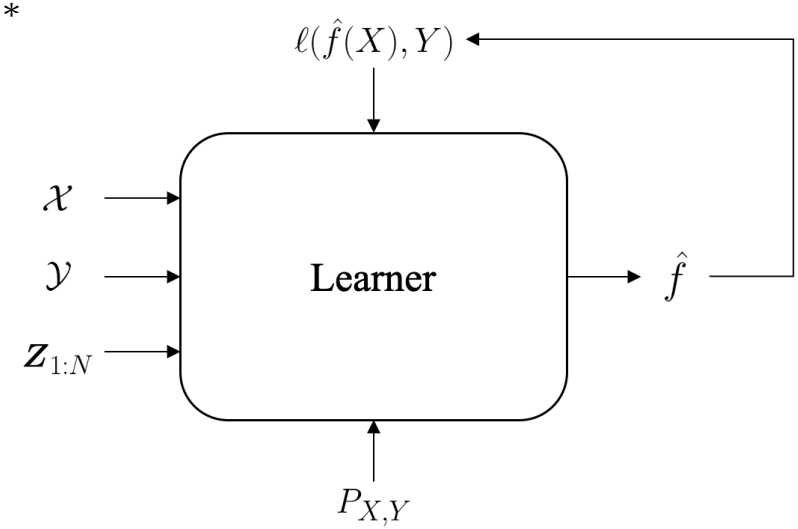
Topic 1: Learning

Learning: A Formal Introduction

- **Domain Set:** An arbitrary set $\mathcal{X}$ which we may wish to label.
- **Label Set:** Set of possible labels $\mathcal{Y} = f^*(\mathcal{X})$, where $f^* : \mathcal{X} \to \mathcal{Y}$ is the true labeling function.
- **Training Data:** A finite sequence of pairs in $\mathcal{X} \times \mathcal{Y}$, denoted as $\boldsymbol{z}_{1:N} = (z_1, \cdots, z_N) = \Big((x_1, y_1), \cdots, (x_N, y_N)\Big)$[1]
- **Learner's Output:** A prediction rule $f : \mathcal{X} \to \mathcal{Y}$ (also called a predictor, hypothesis, classifier) that predicts the labels of a new domain point from a predictor class $\mathcal{F}$. Technically, f is implemented using an algorithm $\mathcal{A}(\boldsymbol{z}_{1:N})$.
- **Data Generation Model:** A probability distribution $P_{X,Y}$ derived from a family of probability distributions $\mathcal{P}$, from which the training data $\boldsymbol{z}_{1:N}$ is generated.
- **Measure of Success:** Loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ (also called risk, generalized error, true error) that quantifies how bad the prediction rule is, when compared to the true labeling function f^*

Goal: Choose the best predictor, i.e.

$$\hat{f} = \underset{f \in \mathcal{F}}{\text{minimize}}\ \mathbb{E}_{P_{X,Y}}\left[\ell(f(X), Y)\right]$$



[1] Sometimes, training data does not come with labels. In such a case, training data is $\boldsymbol{x}_{1:N} = (x_1, \cdots, x_N)$.

Example: Bias Estimation in Coin Tossing

- Goal: Given a (biased) coin with unknown probability θ of turning heads, determine θ as accurately as possible.
- Outcomes of N coin tosses: $\boldsymbol{x}_{1:N} = (x_1, \cdots, x_N)$, where $x_i = \begin{cases} 1, & \text{if Heads,} \\ 0, & \text{otherwise.} \end{cases}$
- Given θ, N and $\hat{\theta}_N(\cdot)$, we can partition $\{0,1\}^N$, the set of all N coin tosses, as

$$\text{Good Data:} \quad G_{N,\epsilon} \triangleq \left\{ \boldsymbol{x}_{1:N} \in \{0,1\}^N \;\middle|\; \left\| \hat{\theta}_N(\boldsymbol{x}_{1:N}) - \theta \right\| \leq \epsilon \right\}$$

$$\text{Bad Data:} \quad B_{N,\epsilon} \triangleq \left\{ \boldsymbol{x}_{1:N} \in \{0,1\}^N \;\middle|\; \left\| \hat{\theta}_N(\boldsymbol{x}_{1:N}) - \theta \right\| > \epsilon \right\}$$

Claim 1

For any true value θ of the coin bias, given any $\epsilon, \delta > 0$, it suffices to collect $N \geq \frac{1}{2\epsilon^2} \log\left(\frac{2}{\delta}\right)$ samples to guarantee

$$\mathbb{P}_\theta\left(G_{N,\epsilon}\right) = \mathbb{P}_\theta\left(\left\| \hat{\theta}_N(\boldsymbol{x}_{1:N}) - \theta \right\| \leq \epsilon\right) > 1 - \delta.$$

Example: Bias Estimation in Coin Tossing (cont...)

Proof of Claim 1:

Main Essence of Learning

Our main wish is to learn something about a phenomenon of interest, via observing random samples of a quantity that is relevant to the phenomenon.

Two basic questions to ask:

- **Statistical Learning:** How many samples are needed to achieve a given accuracy (ϵ) with a given confidence (δ)?
- **Computational Learning:** How efficient is the learning algorithm?

Typical learning frameworks:

- Estimation (e.g. coin tossing)
- Prediction (e.g. classification)
- Clustering
- Representation (Feature) Learning
- Density Estimation...

All the frameworks can be broadly generalized into two learning problems:

- Concept Learning (Binary Outcomes)
- Function Learning (Generalized Outcomes)

Generalization 1: Concept Learning

- *Concept class:* A class $\mathscr{C}$ of subsets of $\mathcal{X}$
- *Unknown target concept:* $C^* \in \mathscr{C}$ picked by Nature
- Binary Label: $Y_i = \mathbb{1}_{\{X_i \in C^*\}}$
- The N feature-label pairs form the training set

$$Z_1 = (X_1, Y_1) = \left(X_1, \mathbb{1}_{\{X_1 \in C^*\}}\right), \cdots, Z_N = (X_N, Y_N) = \left(X_N, \mathbb{1}_{\{X_n \in C^*\}}\right).$$

The objective is to approximate target concept C^* as accurately as possible.

Examples: Classification

Problem 1: Concept Learning

- A concept learning problem is a triple $(\mathcal{X}, \mathcal{P}, \mathscr{C})$, where $\mathcal{X}$ is the feature space, $\mathcal{P}$ is a family of probability distributions on $\mathcal{X}$, and $\mathscr{C}$ is a concept class.
- A learning algorithm for $(\mathcal{X}, \mathcal{P}, \mathscr{C})$ is a sequence $\mathcal{A} = \{A_n\}_{n=1}^{\infty}$ of mappings $A_N : (\mathcal{X} \times \{0,1\})^N \to \mathscr{C}$.

Given a training set $\boldsymbol{Z}_{1:N} = (Z_1, \cdots, Z_N) \in \mathcal{Z}^N$ and a learning algorithm $\mathcal{A}$, the approximation to C^* is

$$\hat{C}_N = A_N(\boldsymbol{Z}_{1:N}) = A_N(Z_1, \cdots, Z_N) = A_N\left(\left(X_1, \mathbb{1}_{\{X_1 \in C^*\}}\right), \cdots, \left(X_N, \mathbb{1}_{\{X_N \in C^*\}}\right)\right).$$

Generalization 1: Concept Learning (cont...)

Two types of errors: (i) $X \in C^* \cap \hat{C}_N^c$, (ii) $X \in (C^*)^c \cap \hat{C}_N$.

Combining the two, misclassification happens when X lies in the symmetric difference

$$C^* \Delta \hat{C}_N = \left(C^* \cap \hat{C}_N^c\right) \cup \left((C^*)^c \cap \hat{C}_N\right).$$

Performance measure of $\mathcal{A}$: $L(C^*, \hat{C}_N) = \mathbb{P}(C^* \Delta \hat{C}_N) = \mathbb{P}(X \in C^* \Delta \hat{C}_N)$.

Good Algorithm $\Rightarrow L(C^*, \hat{C}_N) \to \infty$ as $N \to \infty$.

- Let $X \sim P$, and $(X, \mathbb{1}_{X_N \in C}) \sim P_C$ for any $C \in \mathscr{C}$.
- Since $\hat{C}_N$ is a random element in $\mathscr{C}$, the above convergence can only be achieved in a stochastic sense.
- Define "worst case" size of set of "bad" samples as

$$\phi_{\mathcal{A}}(N, \epsilon, P) = \sup_{C \in \mathscr{C}} P_C^N \left(L\Big(C, A_N(\boldsymbol{Z}_{1:N})\Big) > \epsilon \right)$$

- Since we do not know P, consider the worst case distribution as shown below:

$$\Phi_{\mathcal{A}}(N, \epsilon, \mathcal{P}) = \sup_{P \in \mathcal{P}} \phi_{\mathcal{A}}(N, \epsilon, P).$$

Generalization 1: Concept Learning (cont...)

Definition 1: PAC for Concept Learning

A learning algorithm $\mathcal{A} = \{A_N\}$ is probably approximately correct[a] (or PAC) to accuracy ϵ if

$$\lim_{N \to \infty} \Phi_{\mathcal{A}}(N, \epsilon, \mathcal{P}) = 0.$$

- We say that $\mathcal{A}$ is PAC, if it is PAC to accuracy ϵ for every $\epsilon > 0$.
- The concept class $\mathscr{C}$ is called PAC-learnable to accuracy ϵ w.r.t. P, if there exists an algorithm that is PAC to accuracy ϵ.
- Finally, we say that $\mathscr{C}$ is PAC-learnable, if there exists an algorithm that is PAC.

[a] D. Angluin. Queries and concept learning. Machine Learning, 2:319–342, 1988.

Equivalently, a learning algorithm $\mathcal{A} = \{A_N\}$ is PAC, if for any $\epsilon > 0$ and $\delta > 0$, there exists some $N^*(\epsilon, \delta) \in \mathbb{N}$ such that, for all $N \geq N^*(\epsilon, \delta)$, $C \in \mathscr{C}$ and $P \in \mathscr{P}$, we have

$$P_C^N \left(L\Big(C, A_N(\boldsymbol{Z}_{1:N})\Big) > \epsilon \right) \leq \delta.$$

Note: $N^*(\epsilon, \delta)$ is called the ***sample complexity*** of the learning algorithm $\mathcal{A}$.

Example: Axis-Parallel Rectangles

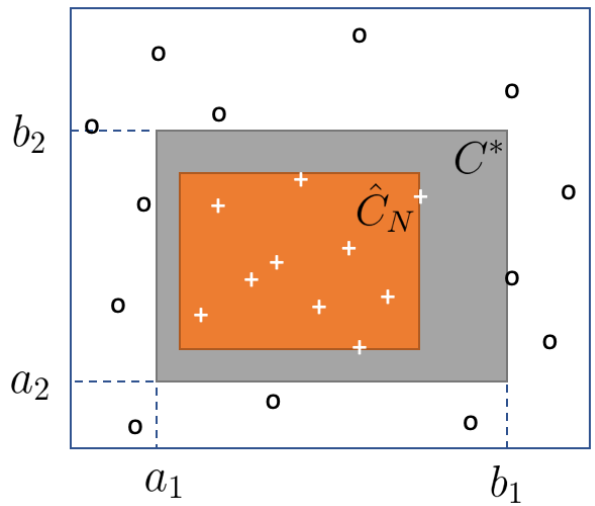
- Let $\mathcal{X} = [0,1]^2$ and $\mathscr{P}$ denote the set of all probability distributions on $\mathcal{X}$
- Let $\mathscr{C}$ denote the collection of all axis-parallel rectangles in $\mathcal{X}$, i.e., C is in $\mathscr{C}$ if it takes the form

$$C = [a_1, b_1] \times [a_2, b_2] = \left\{ (x_1, x_2) \in [0,1]^2 \;\middle|\; a_1 \leq x_1 \leq b_1,\ a_2 \leq x_2 \leq b_2 \right\},$$

for some $0 \leq a_1 \leq b_1 \leq 1$ and $0 \leq a_2 \leq b_2 \leq 1$.

Learning Algorithm:
Consider an intuitive algorithm $\mathcal{A} = \{A_N\}_{N=1}^{\infty}$ where, for each N, we have

$$\begin{aligned} \hat{C}_N &= A_N(\boldsymbol{Z}_{1:N}) \\ &= \text{smallest rectangle } C \in \mathscr{C} \text{ that contains all positive samples in } \boldsymbol{Z}_{1:N}. \end{aligned}$$



Connections to Computer Vision:

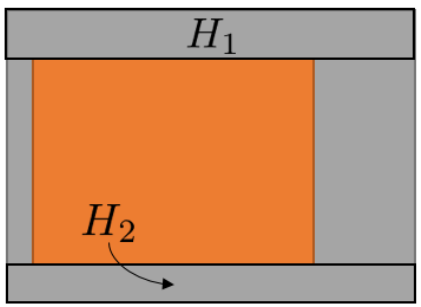
This problem is tangentially similar to estimating bounding boxes in images. The number of samples in $\mathcal{X}$ is similar to the resolution of an image.

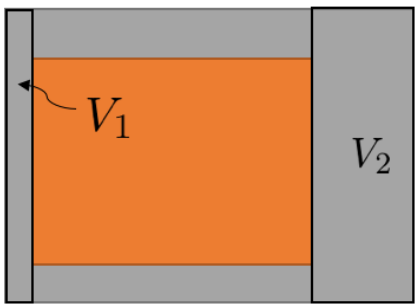
Example: Axis-Parallel Rectangles (cont...)

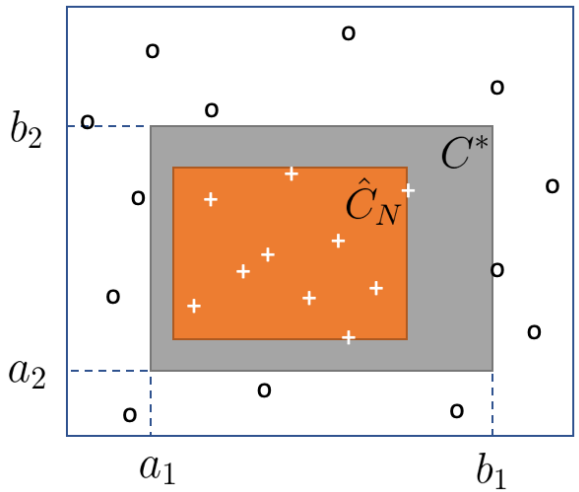
Theorem 1

The above tightest rectangle algorithm $\mathcal{A}$ is PAC, and therefore, the class $\mathscr{C}$ is PAC-learnable, since

$$\Phi_{\mathcal{A}}(N, \epsilon, \mathcal{P}) \leq 4\left(1 - \frac{\epsilon}{4}\right)^N .$$







Proof of Theorem 1:

Example: Axis-Parallel Rectangles (cont...)

Generalization 2: Function Learning

- *Function class:* A class $\mathcal{F}$ defined on $\mathcal{X}$
- *Target function:* $f^* \in \mathcal{F}$ picked by nature
- Real-valued output: $Y_i = f^*(X_i)$
- The N input-output pairs

$$Z_1 = (X_1, Y_1) = (X_1, f^*(X_1)), \cdots, Z_N = (X_N, Y_N) = (X_N, f^*(X_n)).$$

The objective is to approximate target function f^* as accurately as possible.

Examples: Estimation

Problem 2: Function Learning

- A function learning problem is a triple $(\mathcal{X}, \mathcal{P}, \mathcal{F})$, where $\mathcal{X}$ is the feature space, $\mathcal{P}$ is a family of probability distributions on $\mathcal{X}$, and $\mathcal{F}$ is a class of functions $f : \mathcal{X} \to [0, 1]$.
- A learning algorithm for $(\mathcal{X}, \mathcal{P}, \mathcal{F})$ is a sequence $\mathcal{A} = \{A_n\}_{n=1}^{\infty}$ of mappings $A_N : (\mathcal{X} \times \{0, 1\})^N \to \mathcal{F}$.

Given a training set $\boldsymbol{Z}_{1:N} = (Z_1, \cdots, Z_N) \in \mathcal{Z}^N$ and a learning algorithm $\mathcal{A}$, the approximation of f^* is

$$\hat{f}_N = A_N(\boldsymbol{Z}_{1:N}) = A_N\left(\left(X_1, \mathbb{1}_{\{X_1 \in C^*\}}\right), \cdots, \left(X_N, \mathbb{1}_{\{X_N \in C^*\}}\right)\right).$$

Generalization 2: Function Learning (cont...)

Performance of $\mathcal{A}$: $L_P(\hat{f}_N, f^*) = \mathbb{E}_P\left[\left|\hat{f}_N - f^*\right|^2\right] = \int_{\mathcal{X}} |\hat{f}_N(x) - f^*(x)|^2 P(dx)$

Remark: Concept learning is a special case of function learning.

- Define "worst case" size of set of "bad" samples as

$$\phi_{\mathcal{A}}(N, \epsilon, P) = \sup_{f \in \mathcal{F}} P_f^N \left(L\Big(A_N(\boldsymbol{Z}_{1:N}), f\Big) > \epsilon \right)$$

- Since we do not know P, consider the worst case distribution as shown below:

$$\Phi_{\mathcal{A}}(N, \epsilon, \mathcal{P}) = \sup_{P \in \mathcal{P}} \gamma_{\mathcal{A}}(N, \epsilon, P).$$

Definition 2: PAC for Function Learning

A learning algorithm $\mathcal{A} = \{A_N\}$ is probably approximately correct (or PAC) to accuracy ϵ if

$$\lim_{N \to \infty} \Phi_{\mathcal{A}}(N, \epsilon, \mathcal{P}) = 0.$$

- The function class $\mathcal{F}$ is called PAC-learnable to accuracy ϵ w.r.t. P, if there exists an algorithm that is PAC to accuracy ϵ.
- Finally, $\mathcal{F}$ is called PAC-learnable, if there exists an algorithm that is PAC.

Limitations of Model-Based Approaches

- We assume $C^* \in \mathscr{C}$ (or equivalently, $f^* \in \mathcal{F}$) $\Rightarrow$ Fit data regarding a well-studied phenomenon to some ***a priori*** **known hypothesis class**
- Labels $y = \mathbf{1}_{x \in C^*}$ (or equivalently, $y = f^*(x)$) are assumed to be **noiseless**.

Such limitations will lead us naturally towards a new framework called ***model-agnostic learning*** (also called model-free learning).

The main goal is to find the best possible hypothesis (concept/function) within a chosen hypothesis class $\mathcal{F}$.

Problem 3: Model-Agnostic Learning

A model-agnostic learning problem is a tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{P}, \mathcal{F})$, where

- Sets: $\mathcal{X}$ (input feature space), $\mathcal{Y}$ (label space) and $\mathcal{U}$[a] (hypothesis space)
- A class $\mathcal{P}$ of probability distributions on $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$.
- A class $\mathcal{F}$ of functions $f : \mathcal{X} \to \mathcal{U}$.

A learning algorithm for $(\mathcal{X}, \mathcal{Y}, \mathcal{U}, \mathcal{P}, \mathcal{F}, \ell)$ is a sequence of mappings $\mathcal{A} = \{A_N\}_{N=1}^{\infty}$, where $A_N : \mathcal{Z}^N \to \mathcal{F}$.

[a] $\mathcal{U} \neq \mathcal{Y}$, since the true-hypothesis labels are corrupted by noise.

PAC Learnability for Model-Agnostic Learning

- Given a learning algorithm $\mathcal{A} = \{A_N\}_{N=1}^{\infty}$ with $A_N : \mathcal{Z}^N \to \mathcal{F}$, if $\hat{f}_N = A_N(\boldsymbol{Z}_{1:N})$, then the performance can be measured as

$$L_P(\hat{f}_N) = \mathbb{E}_P\left[\ell(Y, \hat{f}_N(X))\right] = \int_{\mathcal{X}\times\mathcal{Y}} \ell(y, \hat{f}_(x))P(dx, dy)$$

- **Minimum risk:** $L_P^*(\mathcal{F}) = \inf_{f\in\mathcal{F}} L_P(f)$ for an induced function class $\mathcal{F}$.

  i.e., given any algorithm $\mathcal{A}$, we have $0 \leq L_P^*(\mathcal{F}) \leq L_P(\hat{f}_N) \leq 1$.
- Given any $\epsilon > 0$, let the worst case probability of getting a bad sample be

$$\Phi_{\mathcal{A}}(N, \epsilon) = \sup_{P\in\mathcal{P}} P^N\left(L_P(\hat{f}_N) > L_P^*(\mathcal{F}) + \epsilon\right)$$

Definition 3: PAC for Model-Agnostic Learning

A learning algorithm $\mathcal{A} = \{A_N\}$ for a problem $(\mathcal{X}, \mathcal{Y}, \mathcal{U}, \mathcal{P}, \mathcal{F}, \ell)$ is PAC to accuracy ϵ if

$$\lim_{N\to\infty} \Phi_{\mathcal{A}}(N, \epsilon) = 0.$$

- An algorithm that is PAC to accuracy ϵ for every $\epsilon > 0$ is said to be PAC.
- Finally, a learning problem $(\mathcal{X}, \mathcal{Y}, \mathcal{U}, \mathcal{P}, \mathcal{F}, \ell)$ is *model-agnostically learnable* if there exists an algorithm for it, which is PAC.

Empirical Risk Minimization

But, we do not always know the input distribution $P \in \mathcal{P}$.

- $L_P(f)$ is unknown. Can we replace this with some surrogate?
- **ERM Algorithm:** $\hat{f}_N = \arg\min_{f \in \mathcal{F}} L_N(f) \triangleq \frac{1}{N} \sum_{i=1}^{N} \ell\big(Y_i, f(X_i)\big)$

Theorem 2: Fundamental Theorem of Learning

Consider an agnostic learning problem $(\mathcal{X}, \mathcal{P}, \mathcal{F})$ and let $\delta > 0$. For any $P \in \mathcal{P}$, the ERM algorithm satisfies

$$L_P(\hat{f}_N) \leq L_P^*(\mathcal{F}) + 8\sqrt{\frac{V(\mathcal{F}) \log(n+1)}{n}} + \sqrt{\frac{2\log(1/\delta)}{n}}$$

with probability at least $1 - \delta$.

Moreover, there is a universal constant C so that for any distribution P on $\mathcal{Z}$ and $\delta \in (0, 1)$, the ERM algorithm satisfies

$$L_P(\hat{f}_N) \leq L_N(\hat{f}_N) + C\sqrt{\frac{V(\mathcal{F})}{n}} + \sqrt{\frac{2\log(1/\delta)}{n}}$$

with probability at least $1 - \delta$.

Vapnik-Chervonenkis Dimension

Definition 4: Shattering

Let $\mathscr{C}$ be a class of (measurable) subsets of some space $\mathcal{Z}$. We say that a finite set $S = \{z_1, , \cdots, z_N\} \subset \mathcal{Z}$ is shattered by $\mathscr{C}$, if for every subset $S' \subseteq S$, there exists some $C \in \mathscr{C}$ such that $S' = S \cap C$.

i.e. $S = \{z_1, , \cdots, z_N\} \subset \mathcal{Z}$ is shattered by $\mathscr{C}$ if, for any binary N-tuple $b = (b_1, \cdots, b_N) \in \{0, 1\}^N$, there exists some $C \in \mathscr{C}$ such that

$$\left(\mathbb{1}_{\{z_1 \in C\}}, \cdots, \mathbb{1}_{\{z_N \in C\}}\right) = b.$$

N^{th} Shatter Coefficient of $\mathscr{C}$: $$\mathbb{S}_N(\mathscr{C}) = \sup_{S \subset \mathcal{Z};\ |S|=n} \left\{S \cap C \;\middle|\; C \in \mathscr{C}\right\}.$$

Definition 5: VC Dimension

The Vapnik-Chervonenkis dimension (or the VC dimension) of $\mathscr{C}$ is

$$V(\mathscr{C}) = \sup\left\{|S|, \text{ such that } S \text{ is shattered by } \mathscr{C}\right\}.$$

If $V(\mathscr{C}) < \infty$, we say that $\mathscr{C}$ is a VC class (of sets).

In other words,

$$V(\mathscr{C}) = \sup\ \left\{n \in \mathcal{N} \;\middle|\; \mathbb{S}_n(\mathscr{C}) = 2^n\right\}.$$

Vapnik-Chervonenkis Dimension (cont...)

Definition 6: VC Dimension of $\mathcal{F}$

Let $\mathcal{F}$ be a class of functions $f : \mathcal{Z} \to \{0, 1\}$. We say that a finite set $S = \{z_1, \cdots, z_n\} \subset \mathcal{Z}$ is shattered by $\mathcal{F}$ if it is shattered by the class $\mathcal{C}_{\mathcal{F}} = \{C_f : f \in \mathcal{F}\}$, where $C_f = \{z \in Z \mid f(z) = 1\}$.

Example 1: Semi-Infinite Intervals

Let $\mathcal{Z} = \mathbb{R}$ and $\mathscr{C}$ is a class of all intervals of the form (∞, t). Then, $V(\mathscr{C}) = 1$.

Example 2: Closed Intervals

Let $\mathcal{Z} = \mathbb{R}$ and $\mathscr{C}$ is a class of all intervals of the form (s, t). Then, $V(\mathscr{C}) = 2$.

Example 3: Closed Half-Spaces

Let $\mathcal{Z} = \mathbb{R}^2$ and $\mathscr{C}$ is a class of all closed half-spaces, i.e. the sets of the form $\{z = (z_1, z_2) \in \mathbb{R}^2 \mid w_1 z_1 + w_2 z_2 \geq b\}$ for all choices of w_1, w_2, b such that $(w_1, w_2) \neq (0, 0)$. Then, $V(\mathscr{C}) = 3$.

Example 4: Axis-Parallel Rectangles

Let $\mathcal{Z} = \mathbb{R}^2$ and $\mathscr{C}$ is a class of all axis-parallel rectangles, i.e. the sets of the form $C = [a_1, b_1] \times [a_2, b_2]$ for all $a_1, a_2, b_1, b_2 \in \mathbb{R}$. Then, $V(\mathscr{C}) = 4$.

No-Free-Lunch Theorem

$$\mathbb{P}\left[L_P(\hat{f}_N) \leq L_N(\hat{f}_N) + C\sqrt{\frac{V(\mathcal{F})}{n}} + \sqrt{\frac{2\log(1/\delta)}{n}}\right] \geq 1 - \delta.$$

The converse of Theorem 2 is also true, i.e. *there is no universal learner*.

Theorem 3: No-Free-Lunch Theorem

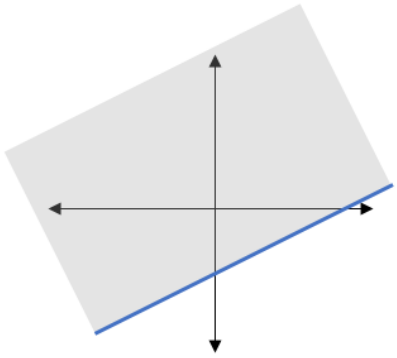
Let $\mathcal{A} = \{A_N\}_{N \in \mathbb{N}}$ be any binary classification algorithm with $0-1$ binary loss on a domain $\mathcal{X}$. Let the training set be of size $m \leq \frac{|\mathcal{X}|}{2}$. Then, there always exists a distribution P over $\mathcal{X} \times \{0, 1\}$ such that

1. there exists a function $f : \mathcal{X} \to \{0, 1\}$ such that $L_P(f) = 0$,
2. with probability at least $1/7$ over some training set $S \sim P^m$, we have $L_P(A_m(S)) \geq 1/8$.

Interpretation: Design ML algorithm that performs well on a specific task on a practical data distribution space.

Perceptron[2]: A Linear Threshold Unit (LTU)

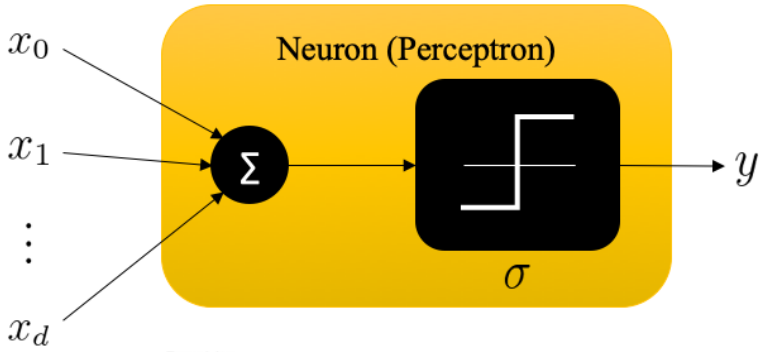
$$y = \sigma(\boldsymbol{w}^T \boldsymbol{x}) = \sigma\left(\sum_{i=0}^{d} w_i x_i\right) \in \{-1, +1\},$$

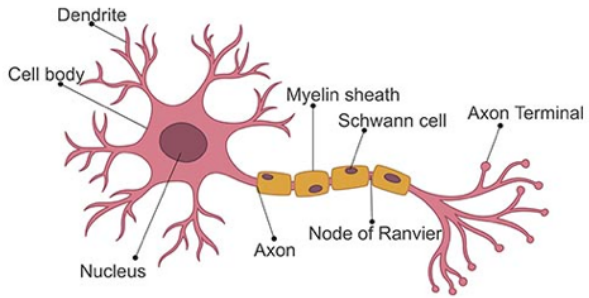


- $x_0 = 1$ (bias neuron)
- w_0 is the threshold
- σ is the *activation/squashing* function.
- **Example:** Heavyside step function

$$\sigma(x) = \begin{cases} 1, & \text{if } x \geq 0, \\ 0, & \text{otherwise.} \end{cases}$$

- Mimics synapses in biological neural networks

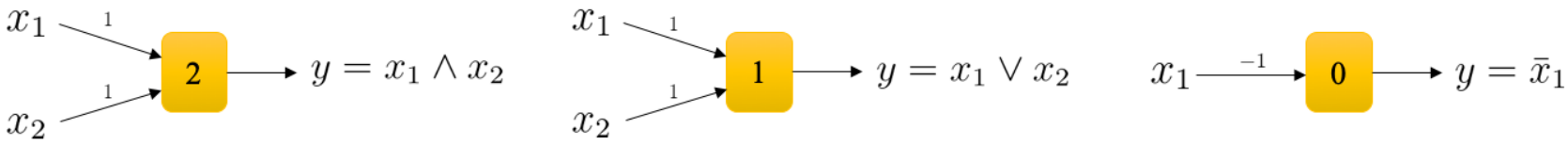




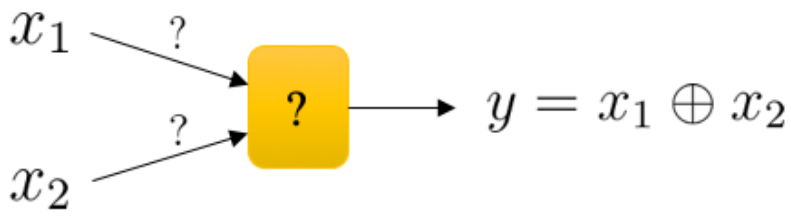
[2] Frank Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review*, vol. 65, no. 6, pp: 386, 1958.

Perceptrons and Boolean Functions

Can perceptrons model boolean functions?



Perceptrons can model several generalized boolean expressions (e.g. majority rule).



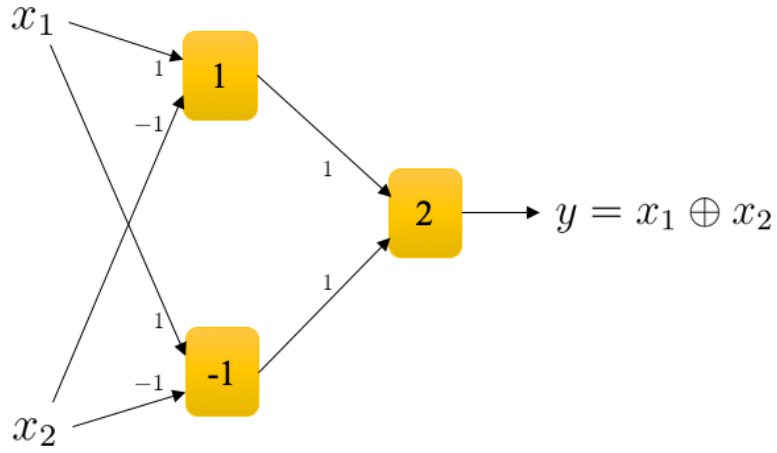
Lemma 1: Minsky and Papert, 1969

A perceptron cannot model a XOR gate.

In other words, there is no combination of weights and threshold in the perceptron model such that the output resembles an XOR.

Need for Multilayer Perceptrons (or, Neural Networks)

- Stack perceptrons to model complex functions
- A simple two-layer neural network (one hidden layer, one output layer) can model XOR function.



Let $\mathcal{F}_L$ denote the set of all neural networks with L layers.

Theorem 4

$\mathcal{F}_2$ is a universal Boolean function space.

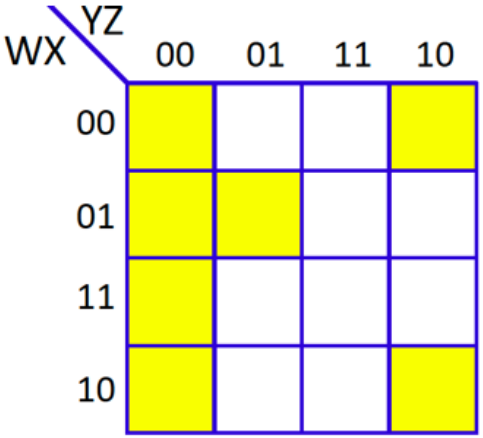
Caveat:

- An arbitrary Boolean function can be represented as a truth table.
- A truth table can be represented in disjunctive normal form (DNF)
- DNF expressions may need a very large number of neurons in the hidden layer.
- Reduce the number of neurons with the help of Karnaugh maps (K-Maps)

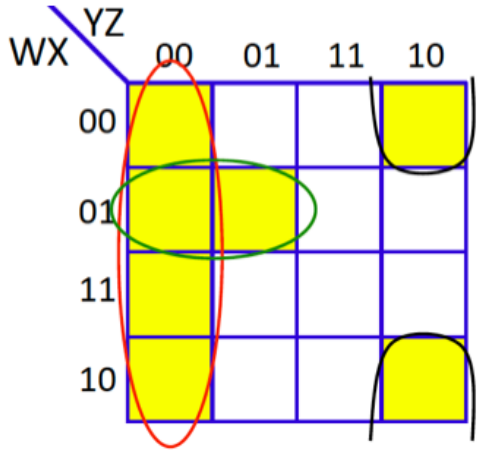
Can we reduce the number of neurons?

Need for Deep Neural Networks (DNNs)

Example[3]: Reduction from 7 hidden neurons to 3 hidden neurons.



$$\bar{W}\bar{X}\bar{Y}\bar{Z} + \bar{W}X\bar{Y}\bar{Z} + WX\bar{Y}\bar{Z} + W\bar{X}\bar{Y}\bar{Z} + \bar{W}X\bar{Y}Z + \bar{W}\bar{X}YZ + W\bar{X}Y\bar{Z}$$

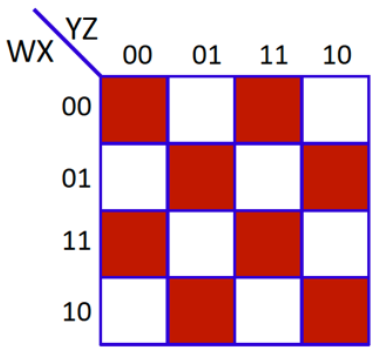


$$\bar{Y}\bar{Z} + \bar{W}X\bar{Y} + \bar{X}YZ$$

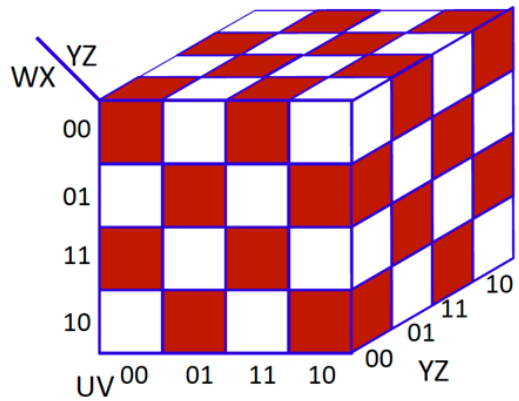
Largest Irreducible DNF:

Lemma 2

A perceptron may require $O(2^N)$ hidden neurons to represent a Boolean function of N variables.



4 variables $\Rightarrow$ 8 terms

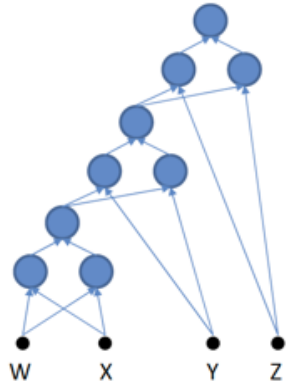


6 variables $\Rightarrow$ 32 terms

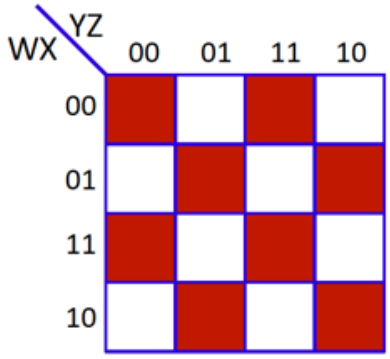
[3] This example was borrowed from the lecture notes of "Deep Learning" course offered in Spring 2022 by Bhiksha Raj and Rita Singh at Carnegie Melon University

Need for DNNs (cont...)

- XOR of two variables needs 3 neurons
- XOR of N variables $\Rightarrow$ $(N-1)$ XORs of two variables
- N variables needs $3(N-1)$ neurons



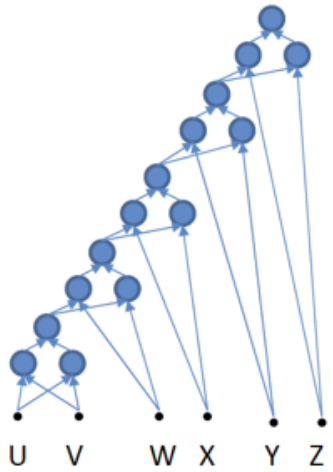
$3 \times 3 = 9$ neurons



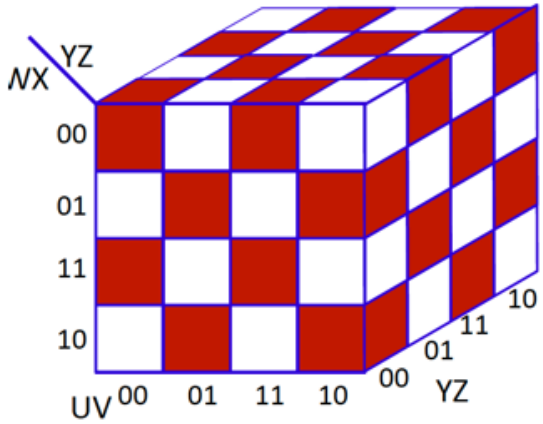
$W \oplus X \oplus Y \oplus Z$

Lemma 3

A perceptron requires $O(N)$ hidden neurons to represent an XOR function of N variables.



$3 \times 5 = 15$ neurons



$U \oplus V \oplus W \oplus X \oplus Y \oplus Z$

- Neurons can be arranged in $2\log_2(N)$ layers.

Claim 2

Reducing the number of layers below the minimum will result in an exponentially sized network for full representation.

An Important Note...

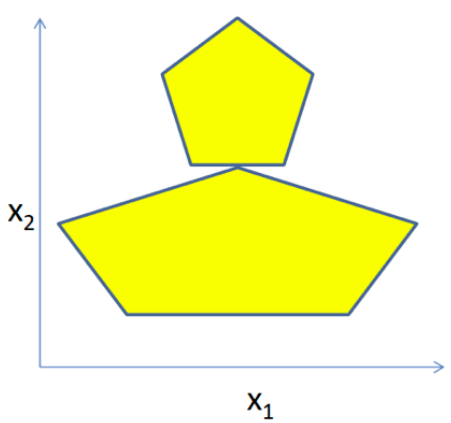
Claim 3: Shannon's Theorem

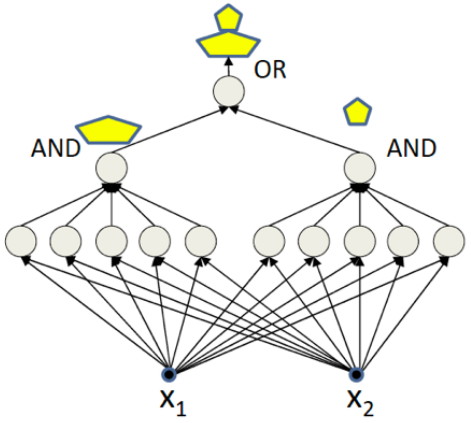
For $N > 2$, there is a Boolean function of N variables that requires at least $\frac{2^N}{N}$ Boolean gates.

Note: If we could represent every Boolean function in $O(N)$ hidden neurons, then $P = NP$!

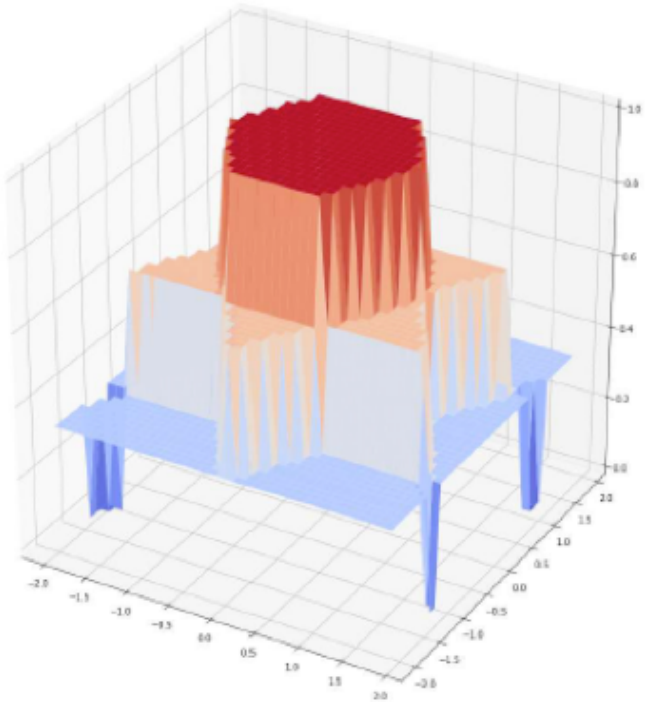
NNs as Universal Classifiers

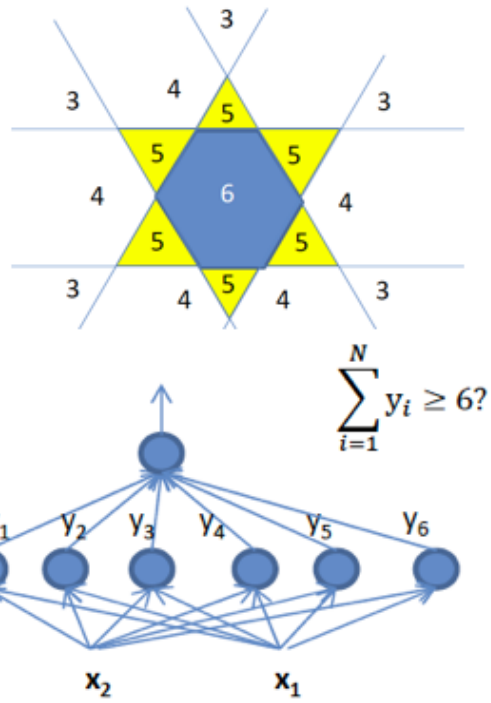
- Each boundary line is a neuron.
- Convex region $\Rightarrow$ AND of all boundary neurons
- Union of all convex regions $\Rightarrow$ OR gate.





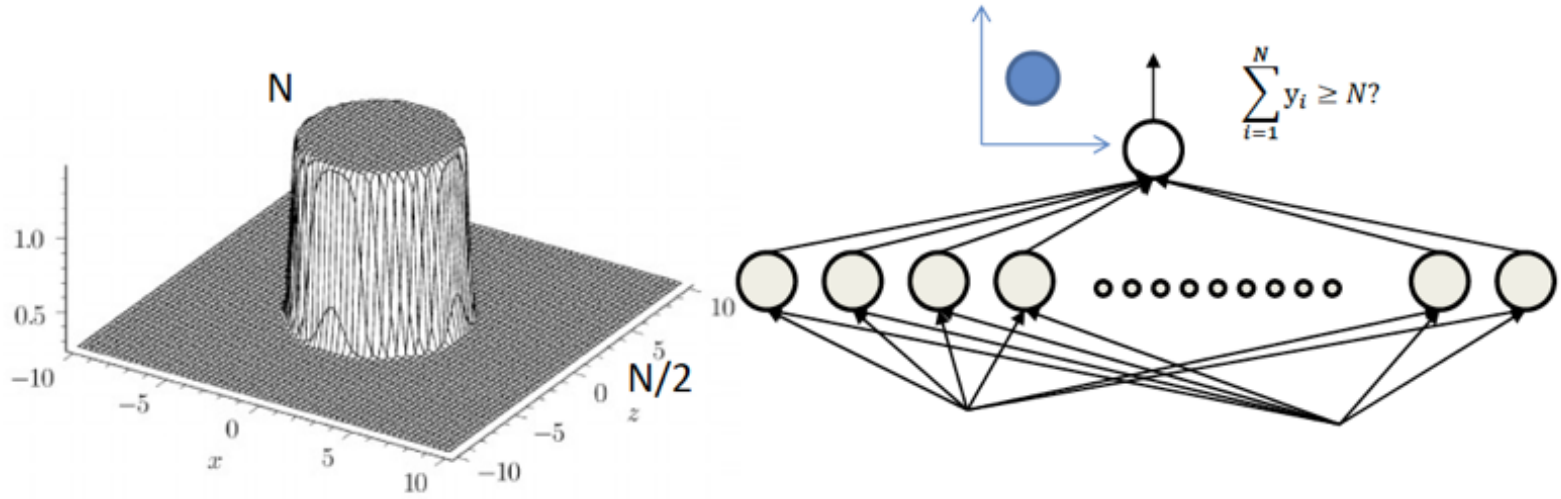
Circular regions can be approximated with higher-order polygons – *Polygon Nets*





NNs as Universal Classifiers (cont...)

A good approximation of a circle needs asymptotic number of neurons!



Any arbitrary region is a union/intersection of circular regions (topology on $\mathcal{X}$).

Theorem 5

Neural networks are universal classifiers

Note: Deeper networks require far fewer neurons in most classifiers.