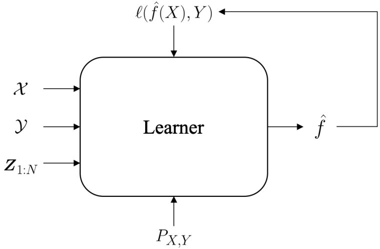
Topic 1: Learning

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

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

Claim 1

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

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

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

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

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

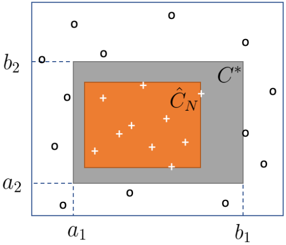
- Let $\mathcal{X} = [0,1]^2$ and $\mathscr{P}$ denote the set of all probability distributions on $\mathcal{X}$
- Let $\mathscr{C}$ denote the collection of all axis-parallel rectangles in $\mathcal{X}$, i.e., C is in $\mathscr{C}$ if it takes the form

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

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

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:

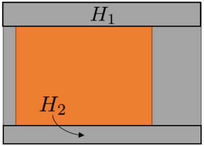
This problem is tangentially similar to estimating bounding boxes in images. The number of samples in $\mathcal{X}$ is similar to the resolution of an image.

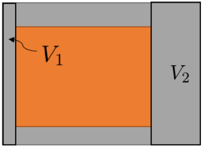
Example: Axis-Parallel Rectangles (cont...)

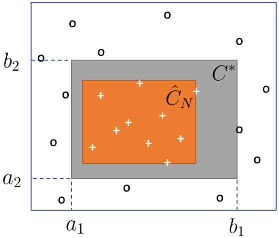
Theorem 1

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 $f : \mathcal{X} \rightarrow \mathcal{U}$.

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 $A_N : \mathcal{Z}^N \rightarrow \mathcal{F}$.

[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:** $\hat{f}_N = \underset{f \in \mathcal{F}}{\arg\min}\ L_N(f) \triangleq \frac{1}{N}\sum_{i=1}^{N} \ell\big(Y_i, f(X_i)\big)$

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 $\mathscr{C}$: $\quad \mathbb{S}_N(\mathscr{C}) = \sup\limits_{S \subset \mathcal{Z};\ |S| = n} \left\{S \cap C \;\middle|\; C \in \mathscr{C}\right\}.$

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 $f : \mathcal{Z} \rightarrow \{0, 1\}$. 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

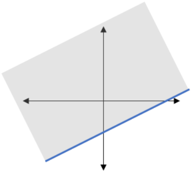
Let $\mathcal{A} = \{A_N\}_{N \in \mathbb{N}}$ be any binary classification algorithm with $0-1$ binary loss on a domain $\mathcal{X}$. Let the training set be of size $m \leq \frac{|\mathcal{X}|}{2}$. Then, there always exists a distribution P over $\mathcal{X} \times \{0,1\}$ such that

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)

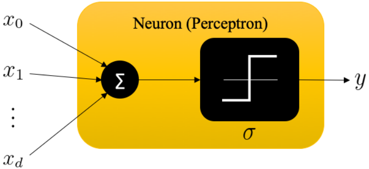
$$y = \sigma(\boldsymbol{w}^T \boldsymbol{x}) = \sigma\left(\sum_{i=0}^{d} w_i x_i\right) \in \{-1, +1\},$$

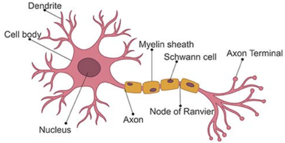


- $x_0 = 1$ (bias neuron)
- w_0 is the threshold
- σ is the *activation/squashing* function.
- **Example:** Heavyside step function

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

- Mimics synapses in biological neural networks

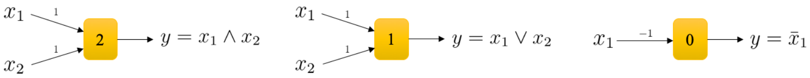




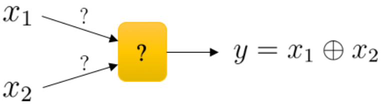
[2] Frank Rosenblatt, “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain,” *Psychological Review*, vol. 65, no. 6, pp: 386, 1958.

Perceptrons and Boolean Functions

Can perceptrons model boolean functions?



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



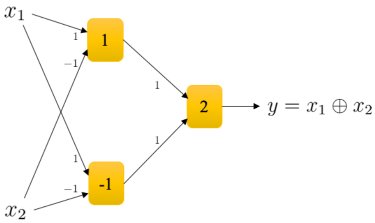
Lemma 1: Minsky and Papert, 1969

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.

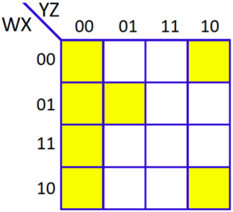
Caveat:

- 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)

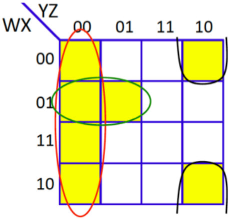
Can we reduce the number of neurons?

Need for Deep Neural Networks (DNNs)

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



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

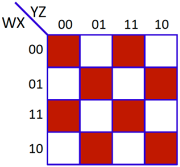


$$\bar{Y}\bar{Z} + \bar{W}X\bar{Y} + \bar{X}Y\bar{Z}$$

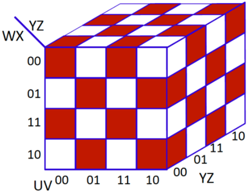
Largest Irreducible DNF:

Lemma 2

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



4 variables ⇒ 8 terms



6 variables ⇒ 32 terms

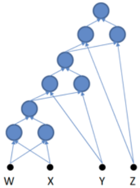
[3] This example was borrowed from the lecture notes of “Deep Learning” course offered in Spring 2022 by Bhiksha Raj and Rita Singh at Carnegie Melon University

Need for DNNs (cont...)

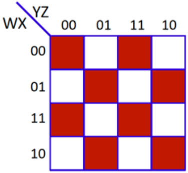
- XOR of two variables needs 3 neurons
- XOR of N variables $\Rightarrow$ $(N-1)$ XORs of two variables
- N variables needs $3(N-1)$ neurons

Lemma 3

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

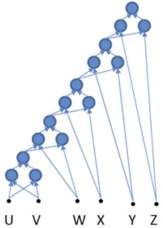


$3 \times 3 = 9$ neurons

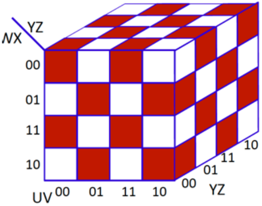


	00	01	11	10
00	🟥	⬜	🟥	⬜
01	⬜	🟥	⬜	🟥
11	🟥	⬜	🟥	⬜
10	⬜	🟥	⬜	🟥

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



$3 \times 5 = 15$ 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...

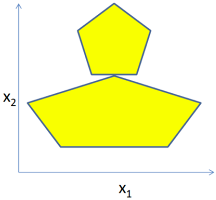
Claim 3: Shannon's Theorem

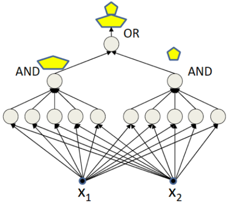
For $N > 2$, there is a Boolean function of N variables that requires at least $\dfrac{2^N}{N}$ Boolean gates.

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

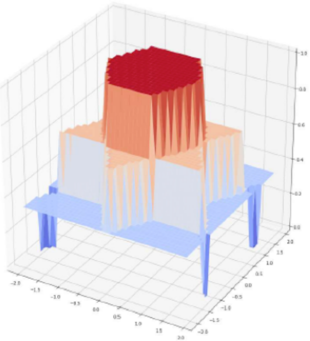
NNs as Universal Classifiers

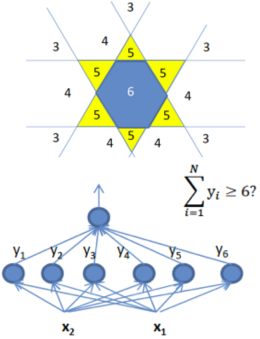
- Each boundary line is a neuron.
- Convex region ⇒ AND of all boundary neurons
- Union of all convex regions ⇒ OR gate.





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

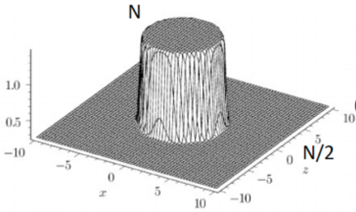


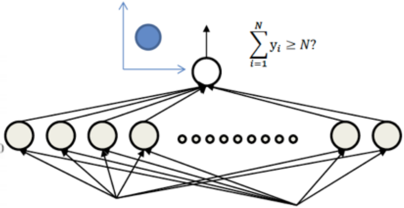


$$\sum_{i=1}^{N} y_i \geq 6?$$

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

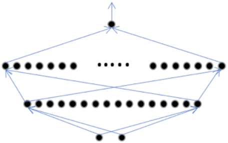
Neural networks are universal classifiers

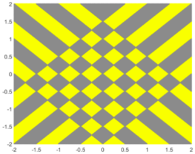
Note: Deeper networks require far fewer neurons in most classifiers.

Architecture Design for Complete Representation[4]

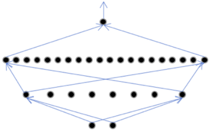
When is a NN architecture sufficiently broad/deep to represent a function?

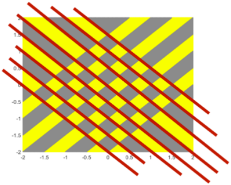
- The adjacent pattern is composed of 16 lines
- So, a network with 16 or more neurons in the first layer is sufficient to represent the region.
- Also, need 40 neurons in the second layer.





- What if, we only have 8 neurons in the first layer?

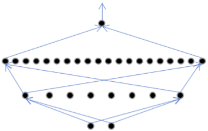


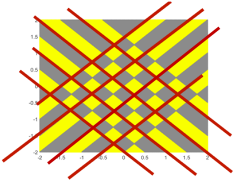


[4] This example was borrowed from the lecture notes of “Deep Learning” course offered in Spring 2022 by Bhiksha Raj and Rita Singh at Carnegie Melon University

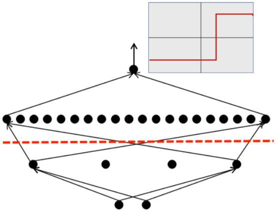
Architecture Design for Complete Representation (cont...)

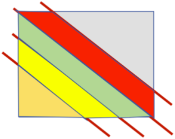
- What if, we consider a different combination of lines?



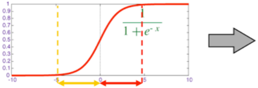


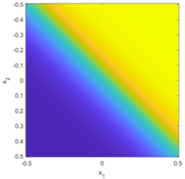
- **Note:** That information, which is not captured by existing neurons, is lost forever in all subsequent layers in the NN.
- This is mainly because of the heavyside step function!



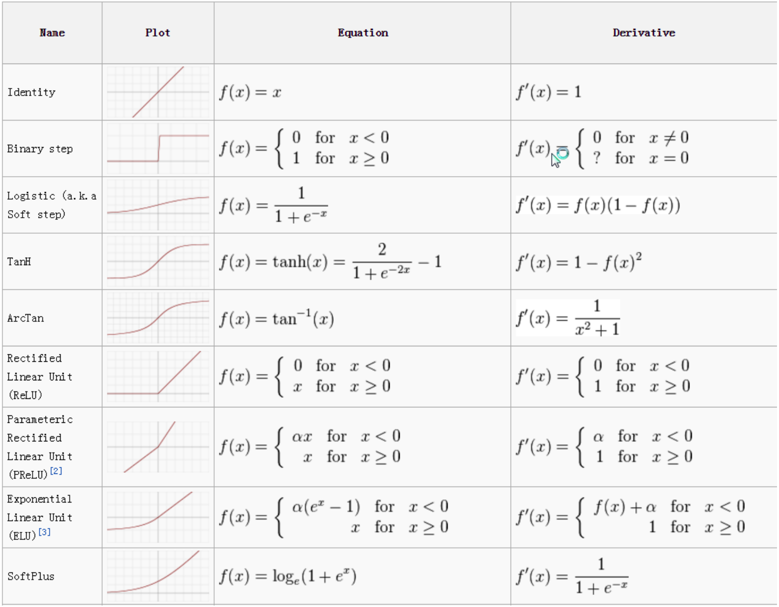


- What if, we have a different activation function?





Activation Functions to our Rescue!



Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x \neq 0 \\ ? & \text{for} \quad x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \dfrac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \dfrac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \dfrac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \dfrac{1}{1 + e^{-x}}$

Source: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

More about Activation Functions...

- **Rectified Linear Unit (ReLU):** $f(x) = \begin{cases} x, & \text{if } x \geq 0, \\ 0, & \text{otherwise.} \end{cases}$
  - Good for intermediate layers
  - *Concern:* No information on one side of hyperplane

- **Leaky Rectified Linear Unit (Leaky ReLU):** $f(x) = \begin{cases} x, & \text{if } x \geq 0, \\ \alpha x, & \text{otherwise,} \end{cases}$, for any $0 < \alpha \ll 1$.
  - Ideal for intermediate layers

- **Sigmoid (or logistic) function:** $f(x) = \dfrac{1}{1 + e^{-x}}$
  - Suits to estimate probability of an outcome in binary classification settings

- **Softmax (or generalized logistic) function:** $f_i(\boldsymbol{x}) = \dfrac{e^{x_i}}{\sum_{j=1}^{M} e^{x_j}}$, for all $i = 1, \cdots, M$
  - Suits to estimate probability of an outcome in M-ary classification settings

- **Hyperbolic tangent function:** $f(x) = \tanh x$
  - Suits to predict the outcome in binary classification settings

NNs for Universal Function Approximation

Definition 7: Universal Approximator

A class of functions $\mathcal{F}$ is a universal approximator over a compact set S, if for every continuous function g and target accuracy $\epsilon > 0$, there exists $f \in \mathcal{F}$ with

$$\sup_{x \in S} |f(x) - g(x)| \leq \epsilon.$$

Let $\mathcal{F}_{\sigma,d}$ denote the set of all multilayer feedforward NNs $\mathcal{F}$, that is restricted to a d-dimensional input and uses $\sigma(\cdot)$ as an activation function in all layers.

Theorem 6: Hornik, Stinchcombe and White, 1989

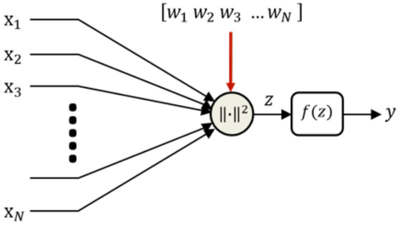
Suppose $\sigma : \mathbb{R} \to \mathbb{R}$ denote a continuous sigmoidal function such that

$$\lim_{z \to -\infty} \sigma(z) = 0, \quad \text{and} \quad \lim_{z \to \infty} \sigma(z) = 1.$$

Then, for any $d \in \mathbb{R}$, $\mathcal{F}_{\sigma,d}$ is universal.

Radial Basis Function Networks for Function Approximation

Why always compare a point to some hyperplane?

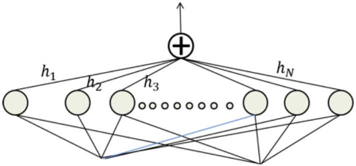


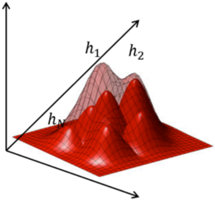
$z = \|\mathbf{x} - \mathbf{w}\|^2$

$y = f(z)$

Typical activation

$f(z) = \exp(-\beta z)$





*

* Source: S. Cuomo, A. Galletti, G. Giunta and A. Starace, ”Surface reconstruction from scattered point via RBF interpolation on GPU,” 2013 Federated Conference on Computer Science and Information Systems, pp. 433-440, 2013.

Feedforward NN: A Formal Model

Definition 8: Feedforward NN

A feedforward neural network is a directed acyclic graph $\mathcal{G} = (V, E)$, and a weight function $w : E \to \mathbb{R}$. Each node is a single neuron (LTU) which is modeled by an activation function $\sigma : \mathbb{R} \to \mathbb{R}$.

Let the set of nodes be decomposed into a union of disjoint nodes, i.e.

$$V = \bigcup_{\ell=0}^{L} V_\ell,$$

such that each edge $e \in E$ connects some node in $V_{\ell-1}$ to V_ℓ for some $\ell = 1, \cdots, L$.

Let $\boldsymbol{f}_\ell(\boldsymbol{x})$ denote the output of V_ℓ, when the NN is fed with some input $\boldsymbol{x}$. Then,

$$\boldsymbol{f}_\ell(\boldsymbol{x}) = \sigma_\ell\Big(W_\ell \cdot \boldsymbol{f}_{\ell-1}(\boldsymbol{x})\Big)$$

In other words,

$$\mathcal{F}_L = \left\{ \boldsymbol{x} \to \sigma_L\Big(W_L \cdot \sigma_{L-1}\big(\cdots \sigma_1(W_1 \boldsymbol{x}) \cdots\big)\Big) \;\middle|\; ||W_\ell|| \leq B \text{ for all } \ell = 1, \cdots, L \right\}$$

VC Dimension for Neural Networks

Theorem 7

VC dimension of perceptrons ($\mathcal{F}_1$ with a heavyside-step activation function) on $\mathbb{R}^d$ is $d+1$.

Theorem 8: (Bartlett, Harvey, Liaw, Mehrabian 2019)

Let σ be a piecewise linear function, W be the number of weight parameters, L be the number of layers. Then,

$$V(\mathcal{F}_L) \leq O(WL \log W).$$

Furthermore, there also exist networks with $V(\mathcal{F}_L) \geq \Omega(WL \log(W/L))$.

Training Neural Networks: ERM Algorithm

- Training data: $z_1 = (\boldsymbol{x}_1, y_1), \cdots, z_N = (\boldsymbol{x}_N, y_N)$
- Function space: Set of all L-layered neural networks $\mathcal{F}_L$
  - Activation functions $\sigma_1(\cdot), \cdots, \sigma_L(\cdot)$ are fixed.
  - M outputs in one-hot form, for M-ary classification.
- Empirical Loss function: $L_N\left(\mathbb{W}|\boldsymbol{z}_{1:N}\right) = \dfrac{1}{N}\displaystyle\sum_{i=1}^{N} \ell\Big(y_i, f(\boldsymbol{x}_i|\mathbb{W})\Big)$
  - L_p-norm: $\ell(y, f(\boldsymbol{x}|\mathbb{W})) = \left(\displaystyle\sum_{m=1}^{M}\Big(y_m - f_m(\boldsymbol{x}|\mathbb{W})\Big)^p\right)^{\frac{1}{p}}$
  - Binary Cross Entropy: $\ell(y, f(\boldsymbol{x}|\mathbb{W})) = -y \log f(\boldsymbol{x}|\mathbb{W}) - (1-y)\log\Big(1 - f(\boldsymbol{x}|\mathbb{W})\Big)$
  - Cross Entropy (or KL Divergence): $\ell(y, f(\boldsymbol{x}|\mathbb{W})) = \displaystyle\sum_{m=1}^{M} y_m \log f_m(\boldsymbol{x}|\mathbb{W})$ for M classes, where y and $f(\cdot)$ are represented in one-hot form.

ERM Algorithm: $\underset{W}{\text{minimize}}\ L_N\left(W|\boldsymbol{z}_{1:N}\right)$

How do we solve the above optimization problem?

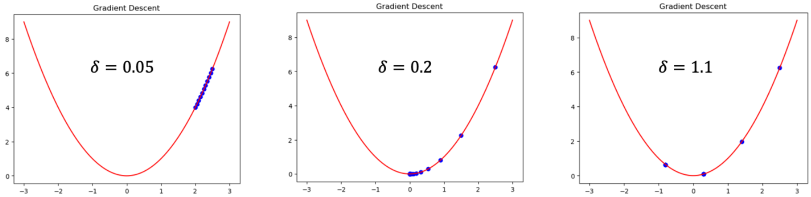
Gradient Descent

- Change $\mathbb{W}$ in the opposite direction of gradient of L_N.

$$\mathbb{W}^{(r)} = \mathbb{W}^{(r-1)} - \delta \cdot \text{sgn}\left(\nabla L_N\left(\mathbb{W}^{(r-1)}\right)\right)$$

```
GD-SIGN$\left(\nabla L_N, \mathbb{W}^{(0)}, R\right)$
for $r = 1$ to $R$
    $\mathbb{W}^{(r)} = \mathbb{W}^{(r-1)} - \delta \cdot \text{sgn}\left(\nabla L_N\left(\mathbb{W}^{(r-1)}\right)\right)$
return $\mathbb{W}^{(R)}$
```

Toy Example: Gradient descent with fixed step size on $f(x) = x^2$ looks like...

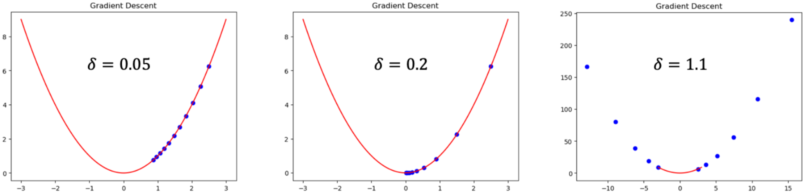


Gradient Descent (cont...)

- Faster convergence ⇒ Adapt step size according to the gradient.

```
GD($\nabla L_N, \mathbb{W}^{(0)}$)
for $r = 1$ to $R$
    $\mathbb{W}^{(r)} = \mathbb{W}^{(r-1)} - \delta \cdot \nabla L_N \left(\mathbb{W}^{(r-1)}\right)$
return $\mathbb{W}^{(R)}$
```

- **Toy Example:** Gradient descent algorithm on $f(x) = x^2$ looks like...



Convergence of Gradient Descent

Theorem 9

Suppose $L_N : \mathbb{R}^W \to \mathbb{R}$ is convex and Lipschitz continuous with constant $\eta > 0$, i.e. we have $||\nabla L_N(\mathbb{W}_1) - \nabla L_N(\mathbb{W}_2)||^2 \le \eta||\mathbb{W}_1 - \mathbb{W}_2||^2$ for any $\mathbb{W}_1, \mathbb{W}_2 \in \mathbb{R}^W$.
Then, the gradient descent algorithm after r rounds with a fixed step size $\delta \le 1/\eta$ will yield a solution $\mathbb{W}^{(r)}$ which satisfies

$$L_N\left(\mathbb{W}^{(r)}\right) - L_N(\mathbb{W}^*) \le \frac{||\mathbb{W}^{(0)} - \mathbb{W}^*||^2}{2r\delta},$$

where $\mathbb{W}^* = \arg\min_{\mathbb{W} \in \mathcal{F}_L} L_N(\mathbb{W})$ is the optimal solution to ERM.

Proof:

Gradient Descent: Variants and Limitations

- Rather than iteratively repeat the update step R times, define a convergence criteria based on loss function...

```
GD2($\nabla L_N, \mathbb{W}^{(0)}, \epsilon$)
r = 1
while $|L_N(\mathbb{W}^r) - L_N(\mathbb{W}^{r-1})| > \epsilon$
    $\mathbb{W}^{(r+1)} = \mathbb{W}^{(r)} - \delta \cdot \nabla L_N \left(\mathbb{W}^{(r)}\right)$
    $r = r + 1$
return $\mathbb{W}^{(r)}$
```

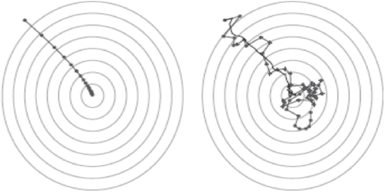
Can we achieve faster convergence?

- Note that most practical neural network architectures (wide or deep) have a large number of weight parameters.
- Computing the gradient is a challenging task.
- What if, we only compute the gradient in just one random direction?

Stochastic Gradient Descent

- Compute gradient only in one random direction (say $j \in \{1, \cdots, W\}$)
- Let $\left[\nabla_{\mathbb{W}} L_N\left(\mathbb{W}^{(r-1)}\right)\right]_j = \begin{bmatrix} 0 & \cdots & 0 & \frac{\partial L_N\left(\mathbb{W}^{(r-1)}\right)}{\partial \mathbb{W}_j} & 0 & \cdots & 0 \end{bmatrix}^T.$
- Requires significantly more rounds to converge
- However, each round is extremely fast!

```
SGD($\nabla L_N, \mathbb{W}^{(0)}, \epsilon$)
$r = 1$
while  $|L_N(\mathbb{W}^r) - L_N(\mathbb{W}^{r-1})| > \epsilon$
    Pick some $j \in \{1, \cdots, W\}$ at random
    $\mathbb{W}^{(r)} = \mathbb{W}^{(r-1)} - \delta \cdot \left[\nabla_{\mathbb{W}} L_N\left(\mathbb{W}^{(r-1)}\right)\right]_j$
    $r = r + 1$
return $\mathbb{W}^{(r)}$
```

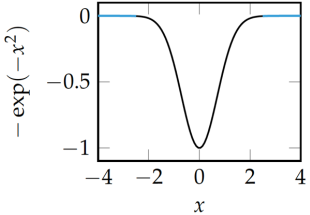


Accelerated Gradient Descent and Nesterov's Momentum[5]

- GD traverses very slowly on a nearly flat surface.
- **Note:** Mimic how a ball gains its momentum as it rolls down a nearly horizontal incline.
- **Gradient Descent with Momentum (GDM):**

$$\begin{aligned} \mathbb{W}^{(r+1)} &= \mathbb{W}^{(r)} + \boldsymbol{m}^{(r+1)} \\ \boldsymbol{m}^{(r+1)} &= \beta \cdot \boldsymbol{m}^{(r)} - \alpha \cdot \nabla L_N(\mathbb{W}^{(r)}) \end{aligned}$$

- If $\beta = 0$, we have GD.



Source: M. J. Kochenderfer and T. A. Wheeler, "Algorithms for Optimization," The MIT Press, 2019.

- **Problem:** GDM does not slow down enough at the bottom of the valley, and tends to overshoot the floor value.
- **Solution:** Gradient Descent with Nesterov's Momentum (GDNM)
  - Use the gradient at the projected future position instead...

$$\begin{aligned} \mathbb{W}^{(r+1)} &= \mathbb{W}^{(r)} + \boldsymbol{m}^{(r+1)} \\ \boldsymbol{m}^{(r+1)} &= \beta \cdot \boldsymbol{m}^{(r)} - \alpha \cdot \nabla L_N(\mathbb{W}^{(r)} + \beta \cdot \boldsymbol{m}^{(r)}) \end{aligned}$$

[5] Y. Nesterov, "A Method of Solving a Convex Programming Problem with Convergence Rate $O(1/k^2)$," *Soviet Mathematics Doklady*, vol. 27, no. 2, pp. 543-547, 1983.

First Order Methods with Adaptive Learning Rate

Can we update different components of $\boldsymbol{W}$ using a different learning rate?

AdaGrad[6]**:**

$$\begin{aligned} \mathbb{W}^{(r+1)} &= \mathbb{W}^{(r)} - \frac{\alpha}{\sqrt{s^{(r)}} + \epsilon} \cdot \nabla L_N(\mathbb{W}^{(r)}) \\ s^{(r)} &= s^{(r-1)} + \left[\nabla L_N(\mathbb{W}^{(r)})\right]^T \cdot \nabla L_N(\mathbb{W}^{(r)}) \end{aligned}$$

- Coordinates with routinely large gradients are scaled down significantly, while those with small gradients are treated more gently.
- But, learning rate decreases too quickly in the context of most learning settings.

RMSProp[7]**:**

$$\begin{aligned} \mathbb{W}^{(r+1)} &= \mathbb{W}^{(r)} - \frac{\alpha}{\sqrt{s^{(r)}} + \epsilon} \cdot \nabla L_N(\mathbb{W}^{(r)}) \\ s^{(r)} &= \beta \cdot s^{(r-1)} + (1-\beta) \left[\nabla L_N(\mathbb{W}^{(r)})\right]^T \cdot \nabla L_N(\mathbb{W}^{(r)}) \end{aligned}$$

- Forgets the gradients from the past $\Rightarrow$ learning rate is better controlled.

[6] J. Duchi, E. Hazan, and Y. Singer, “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization,” *JMLR*, vol. 12, pp. 2121-2159, 2011.

[7] Unpublished, proposed by Geoff Hinton in Lecture-6e in his *Neural Networks for Machine Learning* class on Coursera in Spring 2012

AdaM[8]

Combining the update equations of Nesterov acceleration and RMSProp,
we get ***Adaptive Momentum*** (in short, AdaM)

$$\mathbb{W}^{(r+1)} = \mathbb{W}^{(r)} - \frac{\alpha}{\sqrt{\hat{s}^{(r+1)}} + \epsilon} \hat{\boldsymbol{m}}^{(r+1)}$$

$$\begin{aligned} \boldsymbol{m}^{(r+1)} &= \beta_1 \cdot \boldsymbol{m}^{(r)} + (1 - \beta_1) \cdot \nabla L_N(\mathbb{W}^{(r)}) \\ s^{(r+1)} &= \beta_2 \cdot s^{(r)} + (1 - \beta_2) \cdot \left[\nabla L_N(\mathbb{W}^{(r)}) \right]^T \cdot \nabla L_N(\mathbb{W}^{(r)}) \end{aligned}$$

$$\hat{s}^{(r+1)} = \frac{s^{(r+1)}}{1 - \beta_2^r} \qquad \hat{\boldsymbol{m}}^{(r+1)} = \frac{\boldsymbol{m}^{(r+1)}}{1 - \beta_1^r}$$

- Well suited for problems with large data and/or large number of parameters.
- Works well with non-stationary objectives.
- Works well with very noisy/or sparse gradients.
- Requires little tuning of hyper-parameters.

[8] Diederik P. Kingma and Jimmy Ba, “Adam: A Method for Stochastic Optimization,” In *Proceedings of 3rd International Conference on Learning Representations*, 2015.

Computing Gradients: Backpropagation

All the optimization algorithms rely on computing the gradient.

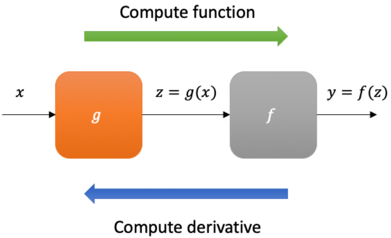
How can we compute the gradient of a neural network?

Backpropagation: Use chain rule to compute derivations...

Consider a simple function composition: $y = f(g(x))$

- Assume x is a scalar quantity
- Let $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ be real valued functions
- Then,

$$\frac{dy}{dx} = \left.\frac{df(z)}{dz}\right|_{z=g(x)} \cdot \frac{dg(x)}{dx}$$



Backpropagation on Neural Networks

Perceptron (NN-1): $Y = \sigma(W \cdot \boldsymbol{x})$

- If $Z = W \cdot \boldsymbol{x}$, we have $Y = \sigma(Z)$
- Empirical loss gradient wrt W: $\nabla_W L_N = \left((\nabla_Y L_N)^T \cdot \nabla_Z Y \right)^T \cdot \nabla_W Z$

Two-Layer Neural Network: $Y = \sigma_2 \left(W_2 \cdot \sigma_1(W_1 \cdot \boldsymbol{x}) \right)$

- If $Z_1 = W_1 \cdot \boldsymbol{x}$, $\tilde{Z}_1 = \sigma_1(Z_1)$ and $Z_2 = W_2 \cdot \tilde{Z}_1$, we have $Y = \sigma_2(Z_2)$
- Empirical loss gradient wrt W_1:
$$\nabla_{W_1} L_N = \left[\left\{ \left((\nabla_Y L_N)^T \cdot \nabla_{Z_2} Y \right)^T \cdot \nabla_{\tilde{Z}_1} Z_2 \right\}^T \cdot \nabla_{Z_1} \tilde{Z}_1 \right]^T \cdot \nabla_{W_1} Z_1$$
- Empirical loss gradient wrt W_2: $\nabla_{W_2} L_N = \left((\nabla_Y L_N)^T \cdot \nabla_{Z_2} Y \right)^T \cdot \nabla_{W_2} Z_2$

Advanced topics on backpropagation (e.g. automatic differentiation) will be covered in Topic 2.

Weight Initialization

No rigorous theory to provide guidance

However, we have some good rules of thumb!

- Do not initialize weights to zero – lower-layer gradients will vanish.
- Too large weight initialization ⇒ exploding gradients problem.
- $\mathbb{W}^{(0)} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$ – not very effective in practice!
- **Xavier Initialization**[9]**:** Choose $W_k \sim \mathcal{N}\left(0, \frac{1}{|W_{k-1}|} I\right)$, where $|W_k|$ is the number of weight parameters in the k^{th} layer.
  - The mean of the activations are maintained to be zero.
  - The variance of the activations would stay the same across every layer.
  - Works well for hyperbolic tangent activations
- **He Initialization**[10]**:** Choose $W_k \sim \mathcal{N}\left(0, \frac{2}{|W_{k-1}|} I\right)$, where $|W_k|$ is the number of weight parameters in the k^{th} layer.
  - Works well with ReLU activations

[9] X. Glorot and Y. Bengio, “Understanding the Difficulty of Training Deep Feedforward Neural Networks,” In *International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[10] K. He, X. Zhang, S. Ren, and J, Sun, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on Image-Net Classification,” In *IEEE International Conference on Computer Vision*, pp. 1026-1034. 2015.