Deep Fool Phenomenon

Smart AI or Clever Hans

Machine Learning

When to apply Machine Learning?

Types of Machine Learning and which one did we focus on?

Learning Algorithm

Loss

Risk

A smart AI genuinely understands and solves problems, while a "Clever Hans" AI appears intelligent but relies on deceptive correlations or unintended cues rather than true learning

Adverserial attack method that iteratively pertubs an input to find the minimal pertubation needed to misclassify it in a deep neural network

- No explicitly known rules (speech recognition)
- Beyond human capabilities (100k rows of data)
- Adaptivity required (handwriting recognition)

A subset of AI that involves training algorithms on data to identify patterns and make predictions or decisions without explicit programming

$A: D^m \to Y^X$ → set of all (measurable) mappings $h: X \to Y$

Given a data sample $s \in D^m$, the learning algorithm outputs a learned hypothesis $h = A(s)$

- supervised vs unsupervised
- active vs passive
- helpful vs neutral vs adversarial
- online vs batch

Given a loss function ℓ, an unknown data distribution μ on D we define the (expected) risk of a hypothesis $h: X \to Y$ by

$$R_\mu(h) := E_\mu[\ell(h,(X,Y))] = \int_{X \times Y} \ell(h,(x,y))\mu(dxdy)$$

To evaluate how well a hypothesis $h: X \to Y$ fits the given data (x_i, y_i) we use a loss function

$$\ell: Y^X \times D \to [0, \infty)$$

Empirical Risk

Empirical Risk Minimization (ERM)

Biased ERM Rule

Approximation Error

Estimation Error

Idealized Error Decomposition

Optimization Error

Realistic Error Decomposition

Given training data $s=((x_1,y_1),\dots,(x_m,y_m)) \in D^m$, loss function $\ell: Y^X \times D \to [0,\infty)$,

$$ERM(s) := \operatorname*{argmin}_{h \in Y^X} R_s(h) = \operatorname*{argmin}_{h \in Y^X} \frac{1}{m} \sum_{i=1}^{m} \ell(h,(x_i,y_i))$$

* Also known as naive ERM rule

Given a loss function ℓ, a data sample $s=((x_1,y_1),\dots,(x_m,y_m)) \in D^m$ the empirical risk of a hypothesis $h: X \to Y$ is given by

$$R_s(h) := \frac{1}{m} \sum_{i=1}^{m} \ell(h,(x_i,y_i))$$

Given a loss function ℓ, data distribution μ, the approximation error of a hypothesis class H is given by

$$\varepsilon_{app}(H) := \min_{h \in H} R_\mu(h)$$

Given training data $s=((x_1,y_1),\dots,(x_m,y_m)) \in D^m$, Hypothesis class $H \subseteq Y^X$, loss function $\ell: H \times D \to [0,\infty)$

$$ERM_H(s) := \operatorname*{argmin}_{h \in H} R_s(h) = \operatorname*{argmin}_{h \in H} \frac{1}{m} \sum_{i=1}^{m} \ell(h,(x_i,y_i))$$

$$\varepsilon_{gen}(s,H) = \varepsilon_{app}(H) + \varepsilon_{est}(s)$$

Given a loss function ℓ, data distribution μ, training data $s \in D^m$, a learning algorithm A, we define the estimation error by

$$\varepsilon_{est}(s) := R_\mu(A(s)) - \min_{h \in H} R_\mu(h)$$

$$\varepsilon_{gen}(s,H,k) = \varepsilon_{app}(H) + \varepsilon_{est}(s) + \varepsilon_{opt}(k)$$

We typically use iterative optimization methods for solving ERM. These recursively compute iterates $h_s^{(k)} \in H$, $k \in \mathbb{N}$, such that (hopefully) $h_s^{(k)} \to h_s$ as $k \to \infty$. However, we have to stop the iteration at some point, say after $k \in \mathbb{N}$ steps. This yields the optimization error

$$\varepsilon_{opt}(k) := R_\mu(h_s^{(k)}) - R_\mu(h_s)$$

Approximation vs Estimation (Bias - Variance Trade off)

Linear Hypotheses

Halfspaces

Perceptron Algorithm

Convergence of Perceptron

Likelihood

Maximum Likelihood Estimation

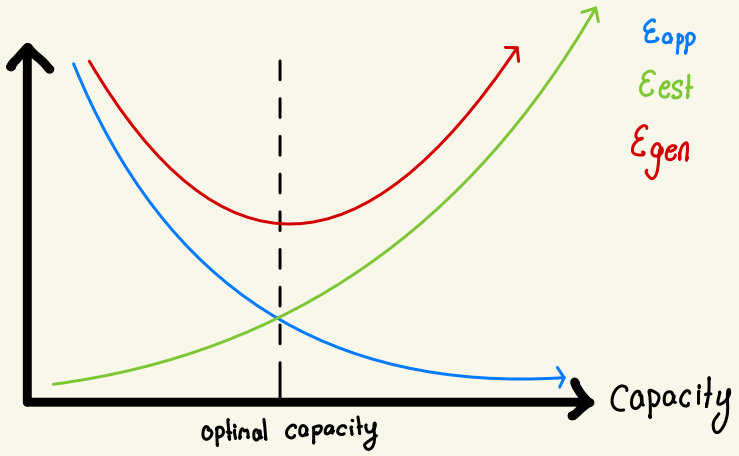
Log Loss

A hypothesis $h: X \to Y$, $X \subseteq \mathbb{R}^d$, is a **linear hypothesis** if there exists an affine-linear function $f_{w,b} \in L_d$ and an **activation function** $\phi: \mathbb{R} \to Y$ such that

$$h(x) = \phi \circ f_{w,b}(x) = \phi(w \cdot x + b), \quad x \in X$$

The set of all linear hypotheses with activation function ϕ

$$L_{d,\phi} := \{h_{w,b}(x) := \phi(wx+b) \mid w \in \mathbb{R}^d, b \in \mathbb{R}\}$$



input: Sample s with m data points $(x_i, y_i) \in \mathbb{R}^d \times \{-1, 1\}$

start: Set $t = 0$ and $w_t' = (w_t, b_t) = 0$

iteration: For $t = 1, 2, \ldots$ do

If there exists an $i = 1, \ldots, m$ with

$$y_i(w_t' \cdot x_i') \leq 0 \quad x_i' := (x_i, 1)$$

then set

$$w_{t+1}' = w_t' + y_i x_i'$$

else stop and output w_t'

A linear hypothesis $h_{w,b}(x) = \operatorname{sgn}(wx+b)$, $x \in X$ divides the features space $X = \mathbb{R}^d$ in two halfspaces

$$H_{w,b}^+ := \{x \in \mathbb{R}^d : wx + b \geq 0\},$$

$$H_{w,b}^- := \{x \in \mathbb{R}^d : wx + b < 0\},$$

by a **separating hyperplane**

$$H_{w,b} := \{x \in \mathbb{R}^d : wx + b = 0\}$$

$$\mathbb{P}_{w,b}(Y = y \mid X = x) := \frac{1}{1 + e^{-y(wx+b)}}$$

The recursive rule $w_{t+1}' = w_t' + y_i x_i'$ for i with $y_i(w_t' \cdot x_i') \leq 0$ points the vector w_{t+1}' in the right direction, since

$$y_i(w_{t+1}' \cdot x_i') = y_i(w_t' \cdot x_i') + y_i^2(x_i' \cdot x_i') > y_i(w_t' x_i')$$

Algorithm stops after $T \leq R^2 B^2$ steps where $R := \max_{i=1,\ldots,m} \|x_i'\|$

$$B := \min\{\|w'\| : y_i(w' \cdot x_i) \geq 1 \quad \forall i = 1, \ldots, m\}$$

Since $(w_s, b_s) = \operatorname{argmin}_{w,b} \sum_{i=1}^m \ln(1 + e^{-y(wx+b)})$

$$\ell(h_{w,b}, (x, y)) := \ln(1 + e^{-y(wx+b)})$$

$$L_{w,b}(s) := \mathbb{P}_{w,b}(Y_1 = y_1, \ldots, Y_m = y_m \mid X_1 = x_1, \ldots, X_m = x_m)$$

$$= \prod_{i=1}^m \mathbb{P}_{w,b}(Y = y_i \mid X = x_i) = \prod_{i=1}^m \frac{1}{1 + e^{-y(wx+b)}}$$

Maximum Likelihood Estimate

$$(w_s, b_s) = \operatorname*{argmax}_{w,b} L_{w,b}(s)$$

It should be clear that

$$\operatorname*{argmax}_{w,b} L_{w,b}(s) = \operatorname*{argmin}_{w,b} -\ln(L_{w,b}(s))$$

Multinomial Logistic Regression

Cross Entropy Loss

Consistency

PAC Condition
(Probably Approximately Correct)

PAC Learnability

PAC Learnability and Consistency

No Free Lunch Theorem

Learnability via Uniform Convergence

$$\ell(h_{w,b},(x,y)) = -\sum_{j=1}^{n} 1_{\{y_j\}}(y) \ln\left(\frac{e^{w_j \cdot x + b_j}}{\sum_{k=1}^{n} e^{w_k \cdot x + b_k}}\right)$$

$$h(x) \approx \begin{bmatrix} P(Y=y_1 \mid X=x) \\ \vdots \\ P(Y=y_n \mid X=x) \end{bmatrix} \in \mathbb{R}^n$$

output of the hypothesis h is a vector $p \in [0,1]^n$ such that $p_1 + \dots + p_n = 1$

Activation function is softmax $\sigma(z) = \left[\frac{e^{z_j}}{\sum_{k=1}^{n} e^{z_k}}\right]$

For a tolerance bound $\varepsilon > 0$ and failure probability $\delta \in (0,1)$ it holds

$$\mathbb{P}_{\mu^m}(R_\mu(h_S) \leq \inf_{h \in H} R_\mu(h) + \varepsilon) \geq 1 - \delta$$

or in short $\mathbb{P}_{\mu^m}(\varepsilon_{est}(H,S) \leq \varepsilon) \geq 1 - \delta$

Given a hypothesis class $H \subseteq Y^X$, a loss ℓ we call a learning algorithm $A: \bigcup_{m \in \mathbb{N}} D^m \to H$ (universally) consistent if for any distribution μ on D we have with $S \sim \mu^m$

$$\varepsilon_{est}(H,S) \xrightarrow[m \to \infty]{\mathbb{P}} 0 \iff \lim_{m \to \infty} \mathbb{P}_{\mu^m}(R_\mu(A(S)) - \inf_{h \in H} R_\mu(h) > \varepsilon) = 0$$

$$\forall \varepsilon > 0$$

If a hypothesis class H is PAC learnable w.r.t. a given loss by a learning algorithm A, then this learning algorithm is consistent for H and ℓ. Converse is not true, because the sample complexity m_H has to apply to any distribution μ

A hypothesis class $H \subseteq Y^X$ is called (agnostic) PAC learnable w.r.t. a given loss ℓ, if there exists a mapping $m_H: (0,1)^2 \to \mathbb{N}$, a learning algorithm A, such that for any data μ on $X \times Y$ and any $\varepsilon \in (0,1)$ and $\delta \in (0,1)$ we satisfy the PAC condition for $h_S = A(S)$ and $m \geq m_H(\varepsilon, \delta)$

Smallest such mapping m_H is called sample complexity of H

Given $H \subseteq Y^X$ we have for $A = ERM_H$ almost surely

$$\varepsilon_{est}(H,S) = R_\mu(A(S)) - \inf_{h \in H} R_\mu(h) \leq 2 \sup_{h \in H} |R_S(h) - R_\mu(h)|$$

Let X be finite and $|Y| = 2$. Further, let ℓ be the 0-1 loss and A be an arbitrary learning algorithm. Then for any sample size $m < |X|/2$, there exists a distribution μ on $X \times Y$ and a hypothesis $h^*: X \to Y$ such that

$$R_\mu(h^*) = 0 \quad \text{but} \quad \mathbb{P}_{\mu^m}(R_\mu(A(S)) \geq 1/8) \geq 1/7$$

- There is no learning algorithm which succeeds on all tasks.
- We need to restrict to suitable hypotheses classes $H \subset Y^X$ for learnability

Uniform Convergence

Restriction

Growth Function

Shattering

VC Dimension

Fundamental Theorem of Learning

Margin

Hard SVM Rule

Given a hypothesis class $H \subseteq \{0,1\}^X$ and a finite set $M = \{x_1, \dots, x_m\} \subseteq X$ we define the restriction of H to M by

$$H_M := \{[h(x_1), \dots, h(x_m)] : h \in H\}$$

i.e., the set of all m-bits $b \in \{0,1\}^m$ generated by an $h \in H$ on M

A class $H \subseteq Y^X$ satisfies the uniform convergence condition (w.r.t. a loss ℓ) if there exists a mapping $m_H^{UC} : (0,1)^2 \to \mathbb{N}$ such that for any data distribution μ on $X \times Y$, any $\varepsilon \in (0,1)$, $\delta \in (0,1)$ we have

$$|P_{\mu^m}(\sup_{h \in H} |R_\mu(h) - R_S(h)| \leq \varepsilon) \geq 1 - \delta \quad \forall m \geq m_H^{UC}(\varepsilon, \delta)$$

- H satisfies UC $\Rightarrow$ H is PAC-learnable with $A = ERM_H$

Let $M \subseteq X$ be finite and $H \subseteq Y^X$, $|Y| = 2$, be a class of binary hypotheses. Then we say H shatters the set M if its restriction H_M to M satisfies

$$H_M = Y^{|M|} \iff |H_M| = 2^{|M|}$$

For a binary hypothesis class $H \subseteq \{0,1\}^X$ its growth function $\tau_H : \mathbb{N} \to \mathbb{N}$ is given by

$$\tau_H(m) := \sup_{M \subset X : |M| = m} |H_M|$$

For a class $H \subseteq Y^X$, $|Y| = 2$, the following statements are equivalent given the 0-1 loss:

1) H satisfies uniform convergence (UC).
2) H is (agnostic) PAC-learnable by $A = ERM_H$.
3) H is (agnostic) PAC-learnable
4) H has finite VC dimension.

The VC (Vapnik-Chervonenkis) dimension of a hypothesis class $H \subseteq \{0,1\}^X$ is

$$VCD(H) := \sup\{|M| : H \text{ shatters } M \subseteq X\}$$

Given linearly separable sample s with m pairs of data $(x_i, y_i) \in \mathbb{R}^d \times \{-1, +1\}$, compute $h_{w,b}(s) \in \mathcal{L}_d$ given by

$$(w_s, b_s) = \underset{(w,b) \in \mathbb{R}^{d+1}}{\operatorname{argmin}} \|w\|^2 \quad \text{subject to} \quad y_i(w \cdot x_i + b) \geq 1 \ \forall i$$

The margin of a sample $s = ((x_1, y_1), \dots, (x_m, y_m))$ to a hyperplane $H_{w,b} := \{x \in \mathbb{R}^d : wx + b = 0\}$ is the smallest distance of a point $x_i \in \mathbb{R}^d$ to $H_{w,b}$.

$$\gamma_{w,b}(s) := \frac{1}{\|w\|} \min_{i = 1, \dots, m} |w \cdot x_i + b|$$

Why is it called "Support Vector" Machine?

Karush-kuhn-Tucker Conditions

Soft SVM Rule

Hinge Loss

Kernel SVM

Hard SVM Rule in F

Soft SVM Rule in F

Representer Theorem

Let $f:\mathbb{R}^d \to \mathbb{R}$ be differentiable, $g_i(w) = a_i^T w + c_i$, $a_i \in \mathbb{R}^d$, $c_i \in \mathbb{R}$ for $i = 1,\dots,m$ and consider

$$w^* \in \operatorname{argmin}_{w \in \mathbb{R}^d} f(w) \quad \text{subject to:} \quad g_i(w) \leq 0 \quad \forall i = 1,\dots,m$$

Then there exists coefficients $\alpha_i \geqslant 0$, $i = 1,\dots,m$, such that

$$\nabla f(w^*) + \sum_{i=1}^m \alpha_i \nabla g_i(w^*) = 0 \quad \text{and} \quad \alpha_i g_i(w^*) = 0 \quad \forall i = 1,\dots,m$$

Consider now the SVM rule as a special case of the optimization task:

$$\left.\begin{aligned} f(w,b) &= \|w\|^2, \Rightarrow \nabla f(w,b) = (2w, 0) \\ g_i(w,b) &= 1 - y_i(\langle w, x_i\rangle + b) \Rightarrow \nabla g_i(w,b) = -y_i(x_i, 1)^T \end{aligned}\right\} \text{KKT yields } w_S = \sum_{s \in J} \alpha_s x_s$$

The weight vector $w_S \in \mathbb{R}^d$ learned by the hard SVM rule is composed of special data points $x_J \in \mathbb{R}^d$:

$$w_S = \sum_{J \in \mathcal{J}} \alpha_J x_J \qquad J \in \mathcal{J} := \{i : y_i(w_S \cdot x_i + b_S) = 1\}, \ \alpha_J \in \mathbb{R}$$

the vectors x_J with $y_J(w_S \cdot x_J + b_S) = 1$ are called support vectors of w_S.

The support vectors are exactly those data points x_s which have the smallest distance to the hyperplane H_{w_S,b_S}:

$$y_i(w_S \cdot x_i + b_S) = 1 \iff d(x_i, H_{w_S,b_S}) = \gamma_{w_S,b_S}(S)$$

For $w' = (w,b) \in \mathbb{R}^{d+1}$ and $x \in \mathbb{R}^d$ and $y \in \{-1,+1\}$ let

$$\ell_{\text{hinge}}(w', (x,y)) := \max\{0, 1 - y(w' \cdot x')\}, \quad x' = (x,1)$$

Given a sample S with m data pairs $(x_i, y_i) \in \mathbb{R}^d \times \{-1,+1\}$, parameter $\lambda > 0$

Compute $h_{w_S,b_S} = \text{SVM}_{\text{soft}}(S;\lambda) \in \mathcal{L}_d$ given by

$$(w_S, b_S, \xi_S) \in \operatorname*{argmin}_{(w,b,\xi) \in \mathbb{R}^{d+1+m}} \lambda\|w\|^2 + \frac{1}{m}\sum_{i=1}^m \xi_i$$

subject to: $y_i(w \cdot x_i + b) \geqslant 1 - \xi_i$ and $\xi_i \geqslant 0 \quad \forall i = 1,\dots,m$

Compute $h_S(x) := \text{sgn}(w_S \cdot \psi(x) + b_S)$ by

$$(w_S, b_S) = \operatorname*{argmin}_{w \in F,\, b \in \mathbb{R}} \|w\|^2$$

subject to

$$y_i(w \cdot \psi(x_i) + b) \geqslant 1 \quad \forall i$$

The idea is to map the original inputs $x \in X$ into a (much) larger feature space F, and apply linear hypotheses $h_F : F \to \{-1,+1\}$ in F for classification

The corresponding embedding $\psi : X \to F$ is called feature map and the resulting nonlinear hypotheses on X are then $h_F \circ \psi$.

The outcome $w_S \in F$ of the hard and soft SVM rule in F as well as any

$$(w_S, b_S) \in \operatorname*{argmin}_{w \in F,\, b \in \mathbb{R}} f(w \cdot \psi(x_1) + b, \dots, w \cdot \psi(x_m) + b) + R(\|w\|)$$

for arbitrary $f : \mathbb{R}^m \to \mathbb{R} \cup \{+\infty\}$ and strictly increasing $R : [0,\infty) \to \mathbb{R}$ can be represented as

$$w_S = \sum_{i=1}^m \alpha_{S,i}\, \psi(x_i), \quad \alpha_{S,i} \in \mathbb{R}$$

Compute $h_S(x) := \text{sgn}(w_S \cdot \psi(x) + b_S)$ by

$$(w_S, b_S) = \operatorname*{argmin}_{w \in F,\, b \in \mathbb{R}} \frac{1}{m}\sum_{i=1}^m \max\{0, 1 - y_i[w \cdot \psi(x_i) + b]\} + \lambda\|w\|^2$$

Kernel Trick

Hard Kernel SVM Rule

Soft Kernel SVM Rule

Polynomial Kernel

Gaussian Kernel

Train, Validation, Test

Cross Validation

Gradient Descent

Compute $h_S(x) := \operatorname{sgn}\left(\sum_{i=1}^{m} \alpha_{S,i}\, k(x_i, x) + b_S\right)$ by

$$(\alpha_S, b_S) = \operatorname*{argmin}_{(\alpha,b)\in\mathbb{R}^{m+1}} \alpha^T K \alpha$$

subject to

$$y_i(K_i.\alpha + b) \geq 1 \quad \forall i$$

Given a feature map $\psi: X \to F$ we define a Kernel $K: X \times X \to \mathbb{R}$ by

$$k(x,y) := \psi(x).\psi(y)$$

Thus we can express everything by K, e.g.,

$$w_S.\psi(x_i) = \sum_{s=1}^{m} \alpha_{S,s}\, k(x_i, x_s), \quad h_S(x) = \operatorname{sgn}\left(\sum_{i=1}^{m} \alpha_{S,i}\, k(x_i, x) + b_S\right)$$

Let $X = \mathbb{R}^d$ and for a $q \in \mathbb{N}$ set $k(x,y) := (1 + x.y)^q$

As a feature mapping $\psi: \mathbb{R}^d \to \mathbb{R}^{(1+d)^q}$ we then define

$$\psi(x) := \left(\prod_{i=1}^{q} x_{J_i} : J = (J_1, \dots, J_q) \in \{0, \dots, d\}^q\right)$$

It then holds with Euclidean inner product in $\mathbb{R}^d$ of $F = \mathbb{R}^{(1+d)^q}$

$$k(x,y) = (1 + x.y)^q = \sum_{J\in\{0,\dots,d\}} \prod_{i=1}^{q} x_{J_i} y_{J_i} = \psi(x).\psi(y)$$

Compute $h_S(x) := \operatorname{sgn}\left(\sum_{i=1}^{m} \alpha_{S,i}\, K(x_i, x) + b_S\right)$ by

$$(\alpha_S, b_S) \in \operatorname*{argmin}_{(\alpha,b)\in\mathbb{R}^{m+1}} \lambda \alpha^T K \alpha + \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - y_i(K_i.\alpha + b)\}$$

Training: Determine hypothesis h_S via numerical computation of a learning rule $h_S \approx A(S)$, e.g. ERM rule $A = ERM_H$

Validation: Choose the most appropriate class H or hyperparameter

Test: Estimate the generalization error of the learned hypothesis

$k(x,y) := e^{-\gamma \|x-y\|^2}$ with scaling parameter $\gamma > 0$

Gaussian kernel evaluates the similarity of the features with respect to their distance $\|x-y\|$, where the distance enters nonlinearly

The Gaussian kernel is also called the RBF kernel, where RBF stands for radial basis function

To small γ may not allow a good fit and too large γ will lead to overfitting

Given a starting vector $w_0 \in \mathbb{R}^p$, calculate for $k = 0, 1, 2, \dots$

$$w_{k+1} = w_k - \eta_k \nabla F(w_k)$$

with corresponding step sizes $\eta_k > 0$

1. Divide the dataset into k equal-sized subsets (folds)
2. For each fold:
   - Use k-1 folds for training
   - Use the remaining fold for validation
3. Rotate the validation fold and repeat the process k times
4. Compute the average performance metric

Lipschitz - smooth

λ-strongly convex

Stochastic Gradient Descent

Subgradient Descent

Artificial Neural Networks

Characteristics of FNN

Structure of CNN

ε-table

A differentiable function $F: \mathbb{R}^p \to \mathbb{R}$ is called λ-strongly convex if for $\lambda > 0$ we have

$$F(v) \geq F(w) + \nabla F(w)^T (v-w) + \frac{\lambda}{2} \|v-w\|^2 \quad \forall w, v \in W$$

Strongly convex functions possess at most one minimum

A differentiable function $F: \mathbb{R}^p \to \mathbb{R}$ is called L-smooth if for $L > 0$ we have

$$\|\nabla F(w) - \nabla F(v)\| \leq L \|w - v\| \quad \forall v, w \in W$$

L-smoothness guarantees a decrease in the objective function value for gradient descent

Given initial state $w_0 \in \mathbb{R}^p$ compute for $k = 0, 1, 2, \ldots$

$$w_{k+1} = w_k - \eta_k v_k,$$
$$v_k \in \partial F(w_k)$$

with suitable stepsizes $\eta_k > 0$

Given a starting vector $w_0 \in \mathbb{R}^p$ and an objective function $F(w) = \frac{1}{m} \sum_{i=1}^{m} f_i(w)$, $f_i: \mathbb{R}^p \to \mathbb{R}$, calculate for $k = 1, 2, \ldots$ iterates w_{k+1} as follows

1. draw realization $i_k \in [m]$ of the uniformly random index variable $I_k \sim U([m])$, $[m] := \{1, \ldots, m\}$ where the $I_k, k \in \mathbb{N}$ are stochastically independent,
2. for a given deterministic step size $\eta_k > 0$, calculate

$$w_{k+1} = w_k - \eta_k \nabla f_{i_k}(w_k)$$

$$h(x) = \rho \circ f_{W_L, b_L} \circ \sigma \circ f_{W_{L-1}, b_{L-1}} \circ \sigma \circ \cdots \circ \sigma \circ f_{W_1, b_1}(x)$$

Depth: L

Width: $B := \max_{k=0,\ldots,L} n_k$

Size: $n := n_0 + n_1 + \cdots + n_L$

Architecture: (V, E) with $V = (V_0, \ldots, V_L)$ and
$E \subseteq \{(v_{k,i}, v_{k+1,j}) : v_{k,i} \in V_k \text{ and } v_{k+1,j} \in V_{k+1}\}$

Number of parameters: $P_{V,E} := |E| + |V_1| + \cdots + |V_L| \leq L(B^2 + B)$

A feedforward neural network is a hypothesis $h: X \to Y$ of the form

$$h(x) = \rho \circ f_{W_L, b_L} \circ \phi \circ f_{W_{L-1}, b_{L-1}} \circ \phi \circ \cdots \circ \phi \circ f_{W_1, b_1}(x)$$

where

- $\phi: \mathbb{R} \to \mathbb{R}$ and $\rho: \mathbb{R} \to Y$ are chosen activation functions whose applications are to be understood componentwise,
- given layerwise weight matrices $W_k \in \mathbb{R}^{n_k \times n_{k-1}}$ and bias vectors $b_k \in \mathbb{R}^{n_k}$ $f_{W_k, b_k}(y) := W_k y + b_k$
- with $n_k \in \mathbb{N}$ denotes the size of the k-th layer V_k

Method	$\mathcal{E}_{app}$	$\mathcal{E}_{est}$	$\mathcal{E}_{opt}$
Perceptron			
Logistic Regression			
Hard/Soft SVM rule			
Kernel SVM rule			
FNN			

CNN typically consist of three kinds of layers

- Convolutional layer: extract new features by applying convolutional filters
- Pooling layer: Reduces information and downsamples new features
- Fully connected layer: Learns classification based on new features provided by previous layer

The Universal Apprimxation theorm or Cybenko theorm states that if we do have a NN specifically a Shallow FNN which has L=2, then it learn any kind of pattern or function mostly continous if we could give it enough neurons and a finite width.

1) here we do have a continous input activation function which is a sigmoid activation function. in the Shallow FFN formula we tend to set our learning NN to learn a continous other actvation function with respect to a threshold.

This threshold mean that the Maximum differences of the Shallow FNN and the learned continous function should not very low and should not be greater that the threshold. this means that the leanred continous function or pattern should be as close as possible to the main Shallow FNN.
We could also tell that bound of diffemeces, that can be the Approximation error too for any epsilon > 0

2) The Kigner and Lyons theorm says that even if we could have a deep narro NN, unlike the Cybenko Theorm, we could have a smooth term of the NN too that can approximate the ocntinous g function. again, the same as above it says that learned continous patter should be as close as possible to main NN that have learned. even with having the small fixed with : d+3 neurons per Layer!

so Going deeper is better than going wider, because it gives: Faster improvement in accuracy - Lower VC dimension not to tend to inflnity or even Estimation Error. according to Petersen theorm.
This is a lower bound on how close the neural network's output h(x) can get to the target g(x).
The error shrinks (gets better) as:

Depth L increases → very fast (exponentially)

Width B increases → much slower (polynomially)

$$\sup_{x \in [0,1]^d} |g(x) - f(x)| \geq C_g (2B+2)^{-2L-2}$$

$$① + ② = \sup_{x \in X} |g(x) - h(x)| \leq \epsilon$$

$$\rightarrow NN : y_{Ki} = \phi\left(\sum_{i \geq 1}^{m} w y_{K-1,i} + b\right)$$

w.r.t $\phi, \rho, \beta, \alpha$:

$$y_{Ki} = \rho\left(\beta + \sum_{i \geq 1}^{m} \alpha_i \phi\left(w_i^T x + b_i\right)\right)$$