Course number: 80240743

Deep Learning

Xiaolin Hu (胡晓林)
Dept. of Computer Science and Technology
Tsinghua University

Topic 2: Basic Knowledge

Xiaolin Hu
Department of Computer Science and Technology
Tsinghua University

Outline

- Math basics
- Machine learning basics
- Neural networks in the early stage

Math basics

LINEAR ALGEBRA

Math objects

But I sometimes may not follow these conventions

- Scalar
  - A single number, often denoted by a lower case letter without boldface, e.g., a, b, x
- Vector
  - An array of numbers, often denoted by a lowercase letter with boldface, e.g., $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{x}$

$$\boldsymbol{a} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

- Matrix
  - A 2D array of numbers, often denoted by an uppercase letter with boldface, e.g., $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{X}$

$$\boldsymbol{A} = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}$$

- Tensor
  - An n-D array of numbers, often denoted like this: e.g., $\mathbf{A}$, $\mathbf{B}$, $\mathbf{X}$

$$\mathbf{A} = \left(\begin{pmatrix} A_{1,1,1} & A_{1,2,1} \\ A_{2,1,1} & A_{2,2,1} \end{pmatrix}, \begin{pmatrix} A_{1,1,2} & A_{1,2,2} \\ A_{2,1,2} & A_{2,2,2} \end{pmatrix} \right)$$

Simple operations

- Matrix transpose: $\boldsymbol{A}^\top$

$$\boldsymbol{A} = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \qquad \boldsymbol{A}^\top = \begin{pmatrix} A_{1,1} & A_{2,1} \\ A_{1,2} & A_{2,2} \end{pmatrix}$$

- A vector can be viewed as a special matrix

$$\boldsymbol{a} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \qquad \boldsymbol{a}^\top = (a_1, a_2, a_3)$$

- Matrix product: if $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{B} \in \mathbb{R}^{n \times p}$, then $\boldsymbol{C} = \boldsymbol{AB}$ with shape $m \times p$ and $C_{i,j} = \sum_k A_{i,k} B_{k,j}$
- Elementwise product (Hadamard product): $\boldsymbol{C} = \boldsymbol{A} \odot \boldsymbol{B}$ where the 3 matrices are of the same shape and $C_{i,j} = A_{i,j} B_{i,j}$

Simple operations

- Properties
  - $\boldsymbol{A}(\boldsymbol{B}+\boldsymbol{C}) = \boldsymbol{A}\boldsymbol{B}+\boldsymbol{A}\boldsymbol{C}$
  - $\boldsymbol{A}(\boldsymbol{B}\boldsymbol{C}) = (\boldsymbol{A}\boldsymbol{B})\boldsymbol{C}$
  - $\boldsymbol{A}\boldsymbol{B} \neq \boldsymbol{B}\boldsymbol{A}$ but $\boldsymbol{x}^\top\boldsymbol{y} = \boldsymbol{y}^\top\boldsymbol{x}$
  - $(\boldsymbol{A}\boldsymbol{B})^\top = \boldsymbol{B}^\top\boldsymbol{A}^\top$
- Identity matrix: $\boldsymbol{I} \in \mathbb{R}^{n\times n}$
  - For $x \in \mathbb{R}^n$, we have $\boldsymbol{I}\boldsymbol{x} = \boldsymbol{x}$
  - For $\boldsymbol{A} \in \mathbb{R}^{n\times m}$, we have $\boldsymbol{I}\boldsymbol{A} = \boldsymbol{A}$

$$\boldsymbol{I} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

- Matrix inverse of a square matrix $\boldsymbol{A}$ is a matrix denoted by $\boldsymbol{A}^{-1}$ such that $\boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}$

Linear equations

- A system of linear equations

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

where $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\boldsymbol{b} \in \mathbb{R}^{m}$ and $\boldsymbol{x} \in \mathbb{R}^{n}$, or equivalently

$$A_{1,1}x_1 + A_{1,2}x_2 + \cdots + A_{1,n}x_n = b_1$$
$$A_{2,1}x_1 + A_{2,2}x_2 + \cdots + A_{2,n}x_n = b_2$$
$$\cdots$$
$$A_{m,1}x_1 + A_{m,2}x_2 + \cdots + A_{m,n}x_n = b_m$$

  - If $m = n$ and $\boldsymbol{A}^{-1}$ exits, then there is exactly one solution $\boldsymbol{x}^*$ to the above equations
  - If $m < n$, there are infinite number of solutions underdetermined
  - If $m > n$, the solution may not exists! overdetermined

Norms

- Norm is used to measure the "size" of a vector or matrix
- The L_p norm of a vector is defined as

$$||\boldsymbol{x}||_p = \left(\sum_i |x_i|^p\right)^{1/p}$$

for $p \in \mathbb{R}, p > 1$

  - The L_2 norm is known as the Euclidean norm, and $||\boldsymbol{x}||_2 = \sqrt{\boldsymbol{x}^\top \boldsymbol{x}}$
  - The L_1 norm is simplified to $||\boldsymbol{x}||_1 = \sum_i |x_i|$
  - The L_∞ norm is simplified to $||\boldsymbol{x}||_\infty = \max_i |x_i|$

Norms

- A norm is any function f that satisfies
  - $f(\boldsymbol{x}) \geq 0$
  - $f(\boldsymbol{x}) = 0 \Rightarrow \boldsymbol{x} = 0$
  - $f(\boldsymbol{x} + \boldsymbol{y}) \leq f(\boldsymbol{x}) + f(\boldsymbol{y})$ (the triangle inequality)
  - $f(\alpha\boldsymbol{x}) = |\alpha| f(\boldsymbol{x}), \forall \alpha \in \mathbb{R}$
- Frobenius norm of a matrix

$$\left|\left|\boldsymbol{A}\right|\right|_F = \sqrt{\sum_{i,j} A_{i,j}^2}$$

Math basics

PROBABILITY THEORY

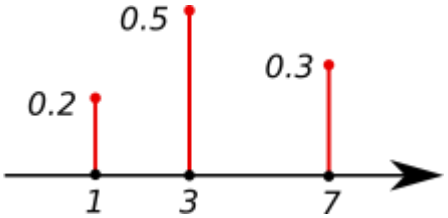
Random variable

- A random variable is a variable that can take on different values randomly
  - Denote the random variable by x and its two possible values by x_1 and x_2
  - For vectors, we write the random variable as $\mathbf{x}$ and one of its values as $\boldsymbol{x}$
  - Discrete versus continuous

Probability distribution

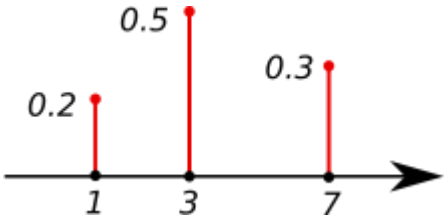
A probability distribution is a distribution of how likely a random variable or a set of random variables is to take on each of its possible states

- A probability distribution over discrete variables may be described using a probability mass function (PMF)
  - The prob that $\mathrm{x} = x$ is denoted as $P(x)$ or $P(\mathrm{x} = x)$
  - $\mathrm{x} \sim P(x)$ specify which distribution x follows
- Joint probability
  - $P(\mathrm{x} = x, \mathrm{y} = y)$ or $P(x, y)$ denotes the prob that $\mathrm{x} = x$ and $\mathrm{y} = y$ simultaneously

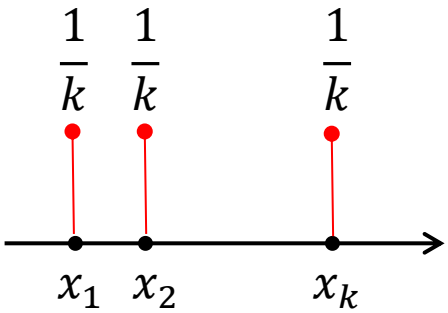


Probability mass function

- To be a PMF of a random variable x, a function P must satisfy:
  - The domain of P must be the set of all possible states of x
  - $\forall x \in \mathrm{x}, 0 \leq P(x) \leq 1$
  - $\sum_{x \in \mathrm{x}} P(x) = 1$

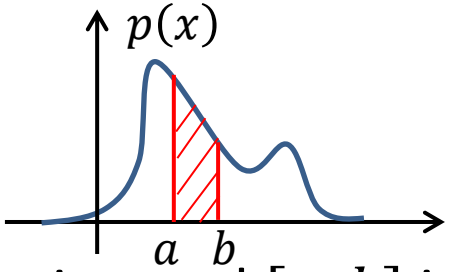


- Uniform distribution
  - Consider a single discrete random variable x with k different states
  - $P(\mathrm{x} = x_i) = \frac{1}{k}, \forall\, i$



Probability density function

- A probability distribution over continuous variables may be described using a probability density function (PDF)
- To be a PDF, a function p must satisfy the following properties
  - The domain of p must be the set of all possible states of x
  - $\forall x \in \mathrm{x}, p(x) \geq 0$
  - $\int p(x)dx = 1$



Can $p(x) > 1$?

- The prob that x lies in the interval $[a, b]$ is given by $\int_a^b p(x)dx$
- Note $p(x)$ does not give the prob of a specific state directly

Marginal probability

Suppose we know the prob distribution over a set of variables. The prob distribution over just a subset of them is known as the marginal prob distribution

- Let $P(\mathrm{x}, \mathrm{y})$ denote the prob distribution of discrete random variables x and y, then

$$P(\mathrm{x} = x) = \sum_{y} P(\mathrm{x} = x, \mathrm{y} = y)$$

- Let $p(\mathrm{x}, \mathrm{y})$ denote the PDF of continuous random variables x and y, then

$$p(x) = \int P(x, y) dy$$

Conditional probability

The conditional probability is the probability of some event, given that some other event has happened

- The conditional prob that $\mathrm{y} = y$ given $\mathrm{x} = x$ is denoted by $P(\mathrm{y} = y|\mathrm{x} = x)$, which can be calculated as
$$P(\mathrm{y} = y|\mathrm{x} = x) = P(\mathrm{y} = y, \mathrm{x} = x)/P(\mathrm{x} = x)$$
- The chain rule
$$P\left(\mathrm{x}^{(1)}, \dots, \mathrm{x}^{(n)}\right) = P\left(\mathrm{x}^{(1)}\right)\Pi_{i=2}^{n} P\left(\mathrm{x}^{(i)}\middle|\mathrm{x}^{(1)}, \dots, \mathrm{x}^{(i-1)}\right)$$
- Exercise: With this definition, is the following correct?
$$P(\mathrm{a}, \mathrm{b}, \mathrm{c}) = P(\mathrm{a}|\mathrm{b}, \mathrm{c})P(\mathrm{b}|\mathrm{c})P(\mathrm{c})$$

Independence and conditional independence

Two random variables x and y are independent if their joint prob distribution can be expressed as a product of two factors, one involving x and one involving y :

Denoted by $\mathrm{x} \perp \mathrm{y}$

$$\forall x \in \mathrm{x}, y \in \mathrm{y},$$
$$p(\mathrm{x} = x, \mathrm{y} = y) = p(\mathrm{x} = x)p(\mathrm{y} = y)$$

Two random variables x and y are conditionally independent given a random variable z if the conditional prob distribution over x and y factorizes in this way:

Denoted by $\mathrm{x} \perp \mathrm{y}|\mathrm{z}$

$$\forall x \in \mathrm{x}, y \in \mathrm{y}, z \in \mathrm{z},$$
$$p(\mathrm{x} = x, \mathrm{y} = y|\mathrm{z} = z) = p(\mathrm{x} = x|\mathrm{z} = z)p(\mathrm{y} = y|\mathrm{z} = z)$$

Expectation

The expectation, or expected value, of some function $f(x)$ w.r.t. a prob distribution $P(\mathrm{x})$ is the average value that f takes on when x is drawn from P

- For discrete variables

$$\mathbb{E}_{\mathrm{x}\sim P}[f(x)] = \sum_{x} P(x)f(x)$$

- For continuous variables

$$\mathbb{E}_{\mathrm{x}\sim P}[f(x)] = \int p(x)f(x)dx$$

- If the identity of the distribution is clear, we may write $\mathbb{E}_{\mathrm{x}}[f(x)]$
- Expectation is linear: if α and β do not depend on x, then

$$\mathbb{E}_{\mathrm{x}}[\alpha f(x) + \beta g(x)] = \alpha\mathbb{E}_{\mathrm{x}}[f(x)] + \beta\mathbb{E}_{\mathrm{x}}[g(x)]$$

Variance and covariance

f and g can be identity functions

- The variance of a function $f(x)$

$$\mathrm{Var}\big(f(x)\big) = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2]$$

  - It measures how much the value of the function f varies when x is sampled from its distribution
- The covariance of two functions $f(x)$ and $g(x)$

$$\mathrm{Cov}(f(x), g(y)) = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])(g(y) - \mathbb{E}[g(y)])]$$

  - It measures how much two values are linearly related to each other, as well as the scale of these variables
  - High absolute values of the covariance mean that the values change very much and are both far from their respective means
- The covariance matrix of a random vector $\boldsymbol{x} \in \mathbb{R}^n$ is an $n \times n$ matrix

$$\mathrm{Cov}(\mathbf{x})_{i,j} = \mathrm{Cov}(\mathrm{x}_i, \mathrm{x}_j)$$

Common prob distributions

- Bernoulli distribution: over a single binary random variable

$$P(\mathrm{x}=1)=\phi, P(\mathrm{x}=0)=1-\phi$$
$$P(\mathrm{x}=x)=\phi^{x}(1-\phi)^{1-x}$$
$$\mathbb{E}_{\mathrm{x}}[\mathrm{x}]=\phi, \mathrm{Var}(\mathrm{x})=\phi(1-\phi)$$

- Multinoulli or categorical distribution: over a single discrete variable with k different states where k is finite

$$P(\mathrm{x}=i|\boldsymbol{p})=p_i$$

where $\boldsymbol{p} \in [0,1]^k$ and $\sum_{i=1}^{k} p_i = 1$

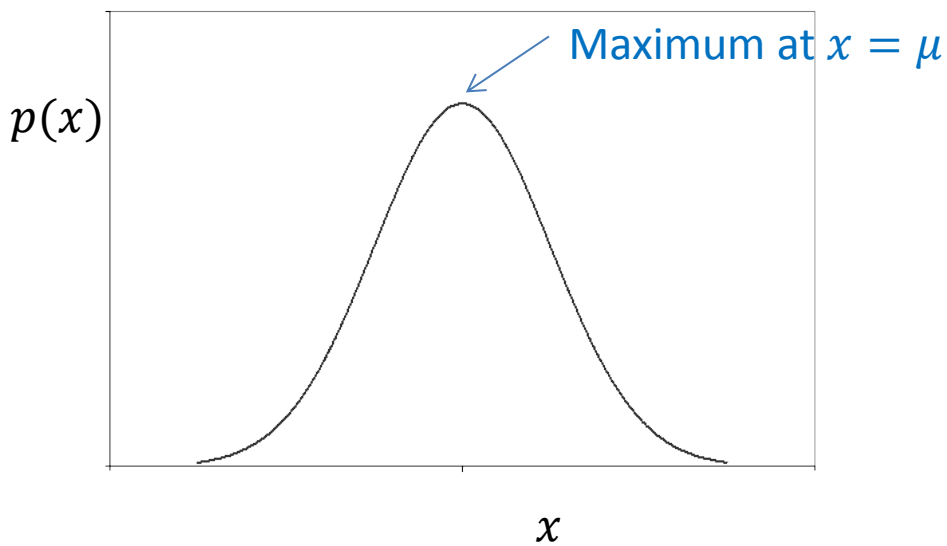
Common prob distributions

- Gaussian distribution or normal distribution: over a continuous variable

$$\mathcal{N}(x;\mu,\sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

  - Mean: $\mathbb{E}[\mathrm{x}] = \mu$
  - Variance: $\mathrm{Var}[\mathrm{x}] = \sigma^2$
  - Standard deviation: σ

The *central limit theorem* shows that the sum of many independent variables is approximately normally distributed

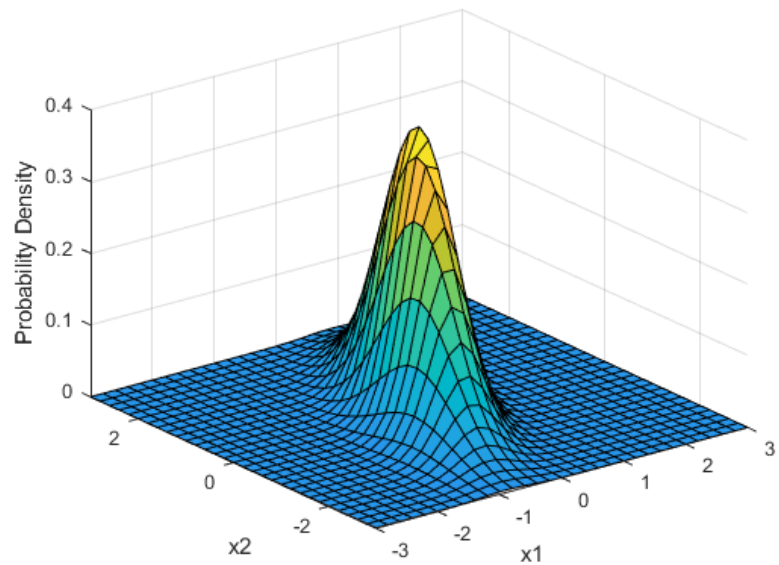


Common prob distributions

- Multivariate normal distribution: over a continuous vector

$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) = \sqrt{\frac{1}{(2\pi)^2 \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right)$$

  - Mean: $\mathbb{E}[\mathbf{x}] = \boldsymbol{\mu}$
  - Covariance: $\mathrm{Cov}[\mathbf{x}] = \boldsymbol{\Sigma}$



Isotropic Gaussian distribution: $\boldsymbol{\Sigma} = \alpha \boldsymbol{I}$ where α is a scalar

Bayes' rule

Thomas Bayes (1702~1761)

Prior probability of x

Posterior probability of x

Probability of y given x; likelihood

$$P(\mathrm{x}|\mathrm{y}) = \frac{P(\mathrm{x})P(\mathrm{y}|\mathrm{x})}{P(\mathrm{y})}$$

Prior probability of y

Math basics

OPTIMIZATION

Gradient-based optimization

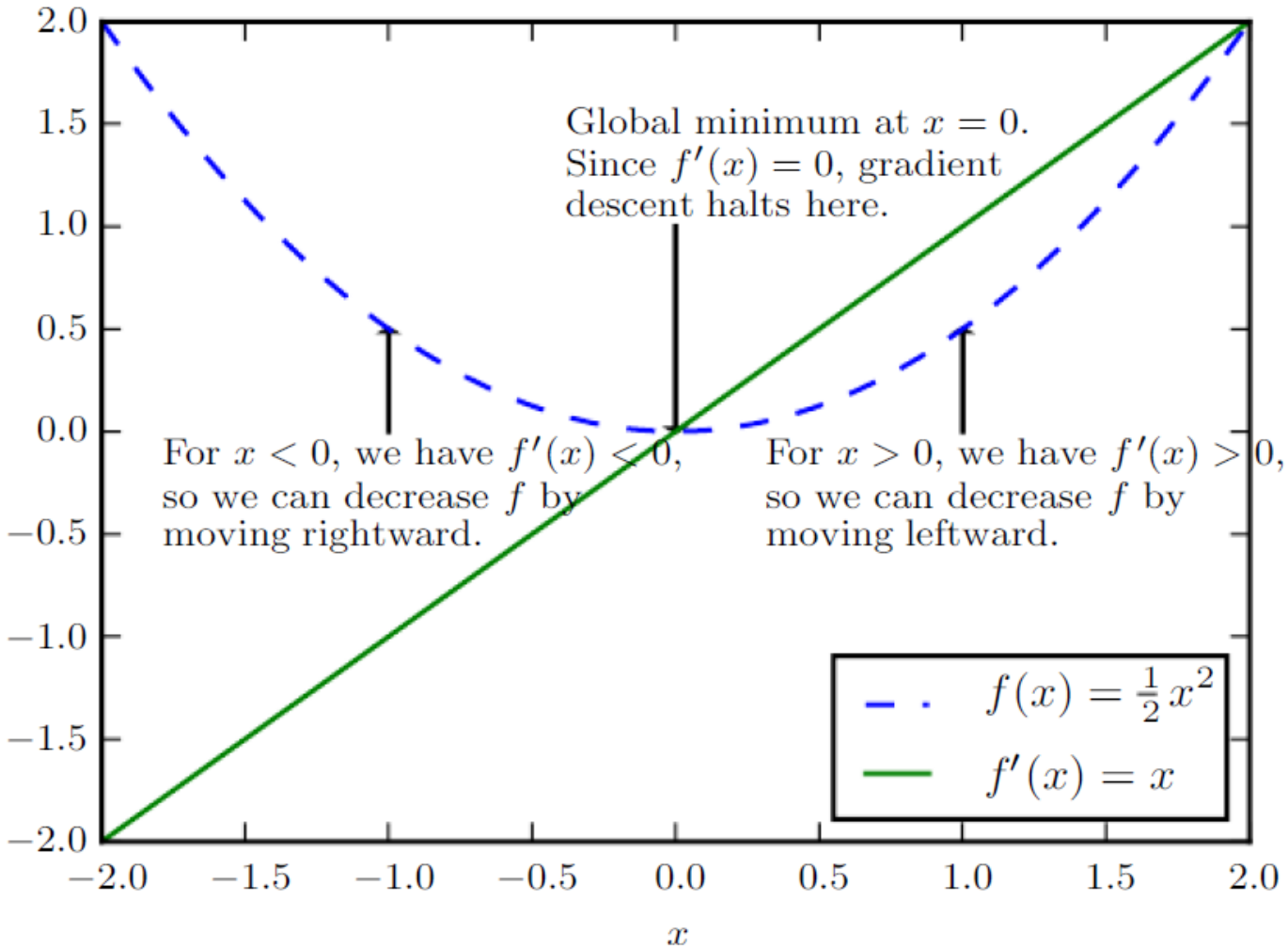
- The function we want to minimize or maximize is called objective function
- When we are minimizing it, we may also call it the cost function, loss function, or error function
- The derivative of a function $y = f(x)$, denoted by $f'(x)$ or $\frac{dy}{dx}$, gives the slope, or gradient, of f at the point x
- Gradient descent

$$x' = x - \eta f'(x)$$

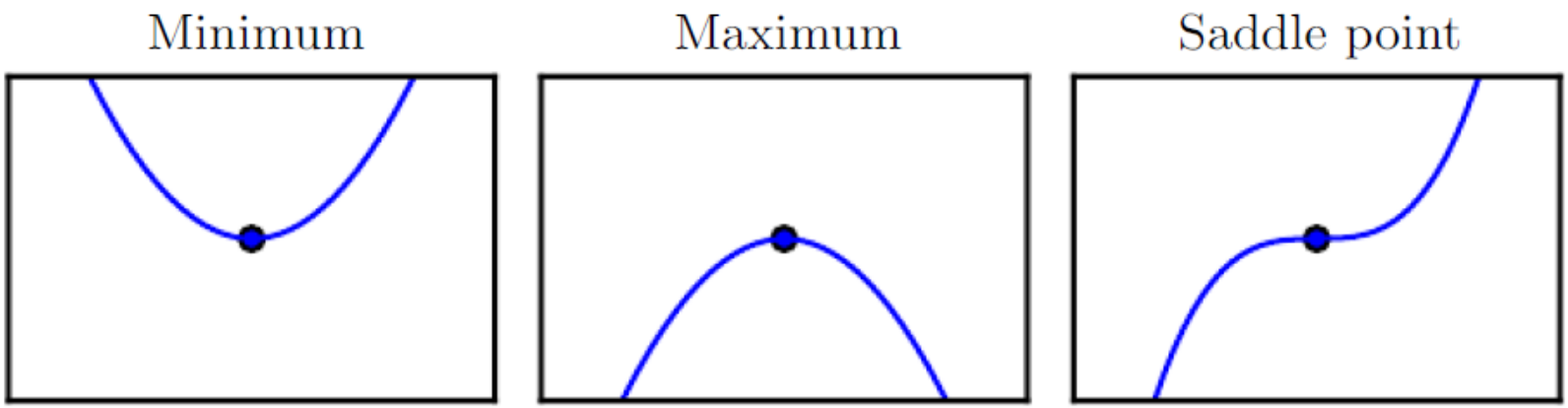
where $\eta > 0$ is the learning rate

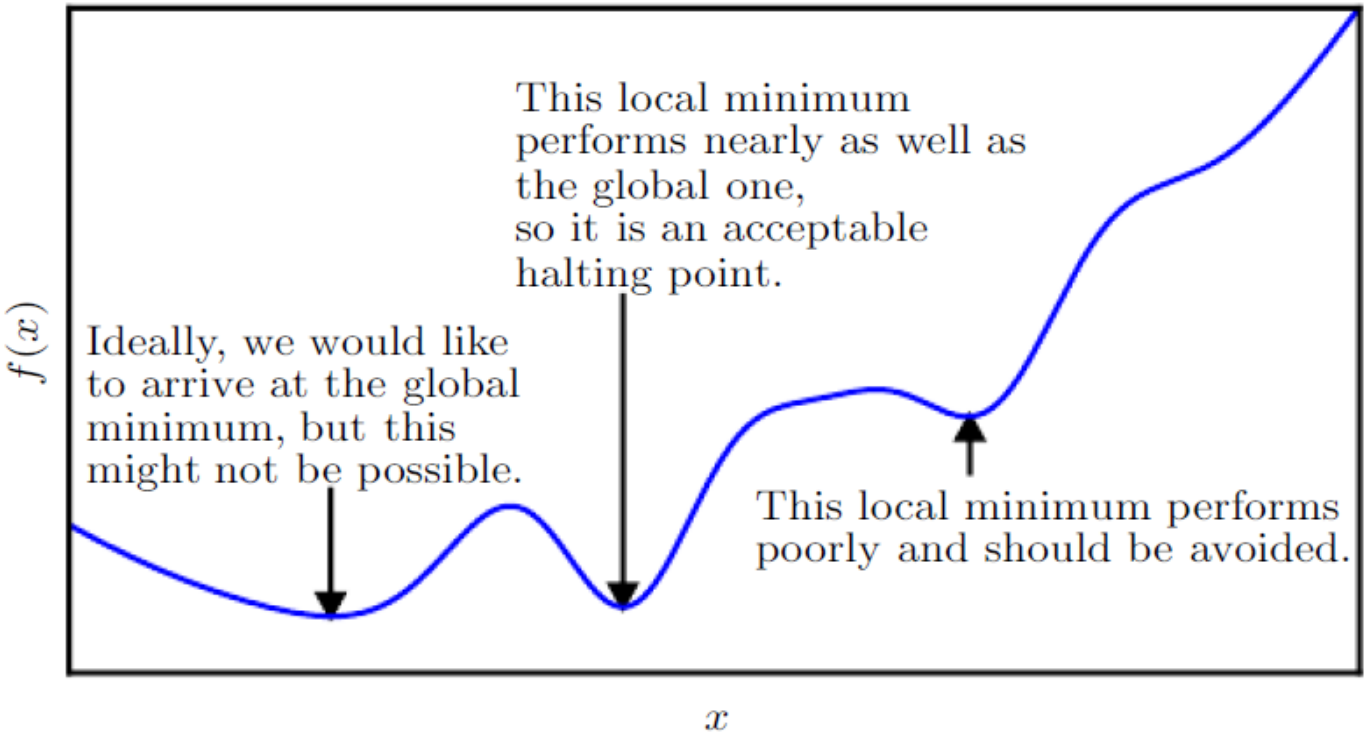
Gradient descent

$$x' = x - \eta f'(x)$$



Critical points





Gradient decent for multivariate functions

- For a function of a single variable $y = f(x)$, the gradient decent method is

$$x' = x - \eta f'(x)$$

- For a function $y = f(\boldsymbol{x})$, the partial derivative is denoted by $\partial f / \partial x_i$
- The gradient decent method becomes

$$\boldsymbol{x}' = \boldsymbol{x} - \eta \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$$

where $\eta > 0$ and $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \begin{pmatrix} \partial f / \partial x_1 \\ \partial f / \partial x_2 \\ \dots \\ \partial f / \partial x_n \end{pmatrix}$

Rules in calculus

- Chain rule: the derivative of the composition function $f(g(x))$ is
$$\left[f\left(g(x)\right)\right]' = f'\left(g(x)\right)g'(x)$$
or in Leibniz's notation
$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$
- Product rule: the derivative of product of two functions
$$(f \cdot g)' = f' \cdot g + f \cdot g'$$
or in Leibniz's notation
$$\frac{d}{dx}(u \cdot v) = \frac{du}{dx} \cdot v + u \cdot \frac{dv}{dx}$$
- Quotient rule
$$\frac{d}{dx}\left(\frac{f(x)}{g(x)}\right) = \frac{f'g - fg'}{g^2}$$

Derivative of two-step composition

Suppose we have:

- Independent input variables $x_1, x_2, \dots, x_n$
- Dependent intermediate variables, $u_1, u_2, \dots, u_m$, each of which is a function of $x_1, x_2, \dots, x_n$
- Dependent output variables $w_1, w_2, \dots, w_p$, each of which is a function of $u_1, u_2, \dots, u_m$

Then for any $i \in \{1,2, \dots, p\}$ and $j \in \{1,2, \dots, n\}$ we have

$$\frac{\partial w_i}{\partial x_j} = \sum_{k=1}^{m} \frac{\partial w_i}{\partial u_k} \frac{\partial u_k}{\partial x_j}$$

Sum over the intermediate variables

From wikipedia

Summary so far

- Linear algebra
  - Math objects: Scalars, vectors, matrices, tensors
  - Simple operations: matrix transpose, inverse, product
  - Norms: L_p norm
- Probability theory
  - Random variables: discrete, continuous
  - Prob distribution: PMF and PDF
  - Marginal probability
  - Conditional probability
  - Independence and conditional independence
  - Expectation, variance and covariance
  - Common prob distributions
  - Bayes' rule
- Optimization
  - Gradient descent
  - Critical points
  - Rules in calculus

Outline

- Math basics
- Machine learning basics
- Neural networks in the early stage

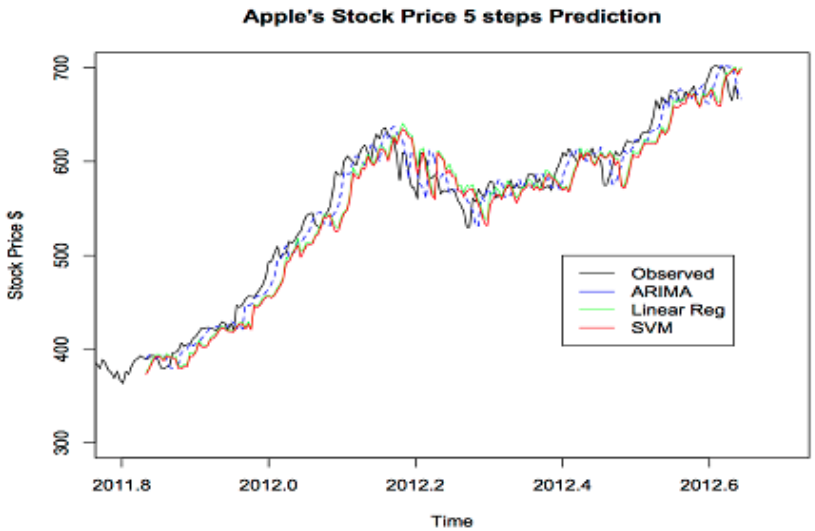
Learning algorithms

"A computer program is said to learn from experience E w.r.t. some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." ---Tom Mitchell, 1997

- Machine learning (ML) tasks are usually described in terms of how the ML system should process an example
- An example is a collection of features that have been quantitatively measured from some object or event
  - Features of a bucket: color, diameter, height, material, etc
  - Features of an animal: size, shape, number of legs, , etc

The tasks T

- Classification
  - Suppose there are k categories. Find a function $f: \mathbb{R}^n \to \{1, \dots, k\}$



- Regression
  - Find a function $f: \mathbb{R}^n \to \mathbb{R}^m$, and m is often 1

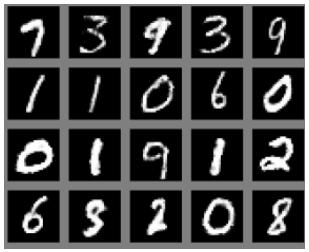
Regression results might be converted to classification results

The tasks T

- Synthesis and sampling

dataset

Synthesized using GAN



- Denoising

The tasks T

- Transcription

Input

Output

Deep learning is cool

- Machine translation

The tasks, T

- Structured output
- Anomaly detection
- Synthesis and sampling
- Imputation of missing values
- Density estimation
- Etc.

The performance measure, P

- A performance measure is required to quantitatively evaluate the performance of a ML system
- Usually this measure P is specific to the task T being carried out by the system
  - Classification and transcription: accuracy or error rate
  - Regression and denoising: distance between the ground-truth and prediction
  - Synthesis, machine translation: difficult and sometimes need human evaluation
- What we are more interested in is the performance measure on a test set of data that is separated from the data used for training the system

The experience, E

- ML algorithms including deep learning algorithms can be broadly categorized as unsupervised or supervised by what kind of experience they are allowed to have during the learning process
- The algorithms experience a dataset, which is a collection of many examples or data points denoted by $\boldsymbol{x}$
  - We can view examples as samples of a random variable $\mathbf{x}$
- Unsupervised learning algorithms

  learn $p(\mathbf{x})$
  - experience a dataset containing many features, then learn useful properties or structures of this dataset
- Supervised learning algorithms

  learn $p(\mathbf{y}|\mathbf{x})$
  - experience a dataset containing many features, but each example is also associated with a label or target denoted by $\boldsymbol{y}$

Example: linear regression

x_i: feature
w_i: weight

- Task T: to predict y from x by outputting $\hat{y} = \boldsymbol{w}^\top \boldsymbol{x}$
- Performance P: mean squared error of the model on the test with m test samples $\{(\boldsymbol{x}_i, y_i)\}^{\text{test}}$

$$\text{MSE}_{\text{test}} = \frac{1}{m}\sum_i (\hat{y}_i - y)^{\text{test}}$$

- Experience E: minimize the MSE on the training set of q samples $\{(\boldsymbol{x}_i, y_i)\}^{\text{train}}$

$$\text{MSE}_{\text{train}} = \frac{1}{q}\sum_i (\hat{y}_i - y)^{\text{train}}$$

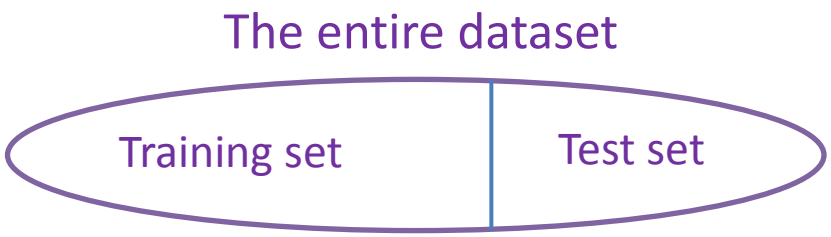
  – Denote $\{(\boldsymbol{x}_i, y_i)\}^{\text{train}}$ collectively by $\left(\boldsymbol{X}^{\text{train}}, \boldsymbol{y}^{\text{train}}\right)$, then

$$\nabla_w \text{MSE}_{\text{train}} = \nabla_w \frac{1}{q}\left|\left|\hat{\boldsymbol{y}}^{\text{train}} - \boldsymbol{y}^{\text{train}}\right|\right|_2^2 = 0$$

$$\Rightarrow \boldsymbol{w} = \left(\boldsymbol{X}^{\text{train}\top}\boldsymbol{X}^{\text{train}}\right)^{-\mathbf{1}} \boldsymbol{X}^{\text{train}\top}\boldsymbol{y}^{\text{train}}$$

Capacity, overfitting and underfitting

- A ML algorithm must perform well on new, previously unseen inputs—not just on which it was trained
  - This ability is called generalization



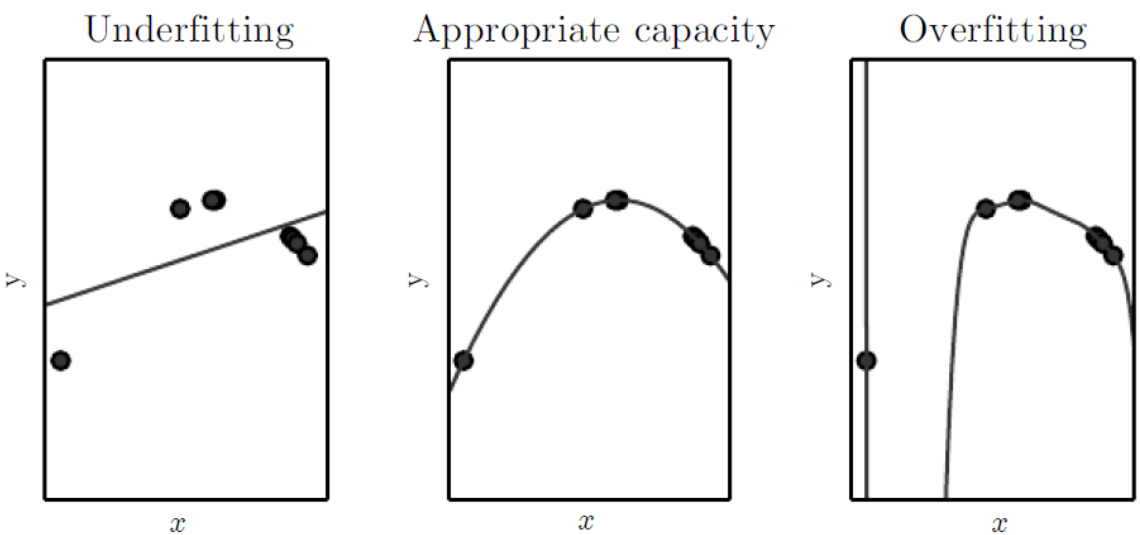
- Smaller training error → higher model capacity
  - If the training error is too large, the model is underfitting the training set
- Smaller test error or generalization error → higher generalization ability
  - If the training error is very small but the test error is very large, the model is overfitting the training set

Example: polynomial regression

- Consider a regression problem in which the input x and output y are both scalars. Find a function $f: \mathbb{R} \to \mathbb{R}$ to fit the data
  - $f(x) = b + wx$
  - $f(x) = b + w_1 x + w_2 x^2$
  - $f(x) = b + \sum_{i=1}^{9} w_i x^i$

MSE training:

$$\min_{w} \frac{1}{N} \sum_{n=1}^{N} \left\| f\left(x^{(n)}\right) - y^{(n)} \right\|_2^2$$



General principles

- Increase the model capacity
  - Make the training error small
- Increase the generalization ability
  - Make the gap between training error and test error small

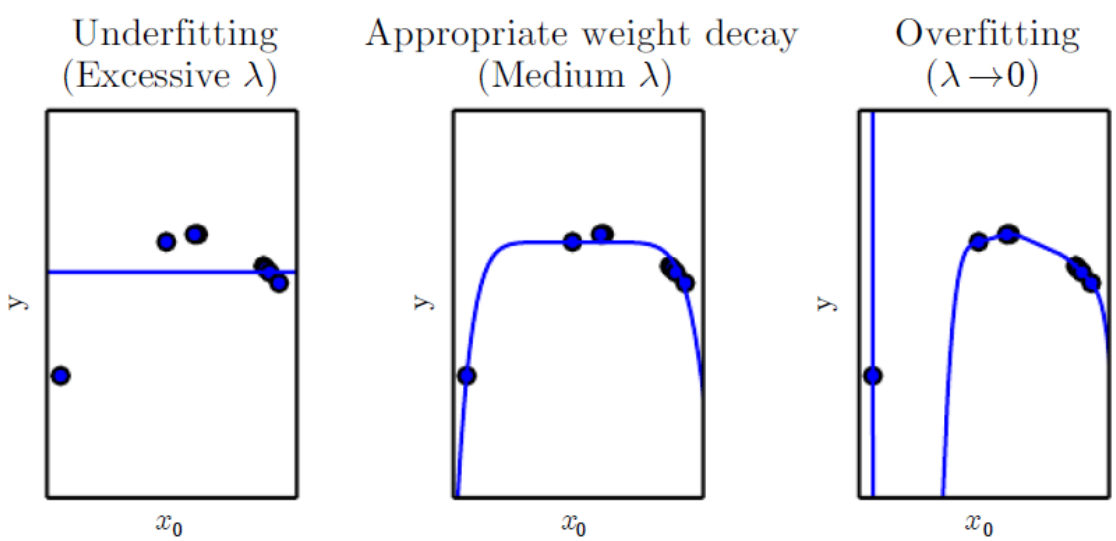
Regularization

- To carry out a specific task, we often build a set of preferences into the learning algorithm, which is embodied by a regularizer Ω
- E.g., for polynomial regression, the total cost function becomes

$$J(\boldsymbol{w}) = \mathrm{MSE}_{\mathrm{train}} + \lambda \boldsymbol{w}^\top \boldsymbol{w}$$

Weight decay

where $\lambda > 0$ is a constant.



A high-degree polynomial regression example

- Here $\Omega(\boldsymbol{w}) = \boldsymbol{w}^\top \boldsymbol{w}$
- There are many regularizers

Regularization

Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error

Hyperparameters

- Many machine learning algorithms have two sets of parameters:
  - Hyperparameters: control the algorithm's behavior and are not adapted by the algorithm itself. They often determines the capacity of the model
  - Learnable parameters ("learnable" is often omitted): can be learned from data
- The polynomial regression algorithm $J(\boldsymbol{w}) = \mathrm{MSE}_{\mathrm{train}} + \lambda \boldsymbol{w}^\top \boldsymbol{w}$
  - Hyperparameters: λ
  - Learnable parameters: $\boldsymbol{w}$
- A neural network
  - Hyperparameters: the number of layers, the number of neurons per layer, etc.
  - Learnable parameters: weights and bias in each layer

Question

- How to choose the hyperparameters considering that we cannot see the test set?

Maximum likelihood estimation

Problem definition

- Given a set of N examples $\mathbb{X} = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(N)}\}$ drawn independently from the true but unknown data-generating distribution $p_{\text{data}}(\mathbf{x})$
- Find a prob distribution $p_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})$ to approximate $p_{\text{data}}(\mathbf{x})$
- The task is to find optimal $\boldsymbol{\theta}$

- The maximum likelihood estimator for $\boldsymbol{\theta}$ is defined as

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} p_{\text{model}}(\mathbb{X}; \boldsymbol{\theta}) = \arg\max_{\theta} \Pi_{i=1}^{N} p_{\text{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

- We usually use

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log p_{\text{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

$$= \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})$$

Log-likelihood

  - where $\hat{p}_{\text{data}}$ is the empirical distribution

Conditional log-likelihood

- Estimate a conditional probability $P(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$ in order to predict $\mathbf{y}$ given $\mathbf{x}$
  - E.g. For classification, $\mathbf{y}$ is a (discrete) random variable representing label of an input $\mathbf{x}$
- If $\boldsymbol{X}$ represents all inputs and $\boldsymbol{Y}$ all observed targets, then the conditional maximum likelihood estimator is

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} P_{\mathrm{model}}(\boldsymbol{Y}|\boldsymbol{X};\boldsymbol{\theta})$$

- If the examples are assumed to be i.i.d., then this can be decomposed into

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log P_{\mathrm{model}}(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta})$$

Stochastic gradient decent

- Minimizing the cost function over the entire training set is computationally expensive
- We often decompose the training set into smaller subsets or minibatches and optimize the cost function defined over individual minibatches $(\boldsymbol{X}^{(i)}, \boldsymbol{y}^{(i)})$ and take the average

$$J(\boldsymbol{\theta}) = \frac{1}{N'} \sum_{i=1}^{N'} L(\boldsymbol{X}^{(i)}, \boldsymbol{y}^{(i)}, \boldsymbol{\theta})$$

$$\boldsymbol{g} = \frac{1}{N'} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{N'} L(\boldsymbol{X}^{(i)}, \boldsymbol{y}^{(i)}, \boldsymbol{\theta})$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \boldsymbol{g}$$

- A total of N' minibatches
- The batchsize ranges from 1 to a few hundreds

Summary so far

- Math basics
  - Linear algebra
  - Probability theory
  - Optimization
- Machine learning basics
  - Learning algorithms, task T, performance P and experience E
  - Capacity versus generalization
  - Maximum likelihood estimation
  - SGD

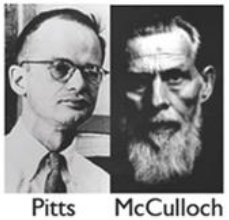
Outline

- Math basics
- Machine learning basics
- Neural networks in the early stage

Neural networks in the early stage

MCCULLOCH AND PITTS NEURON

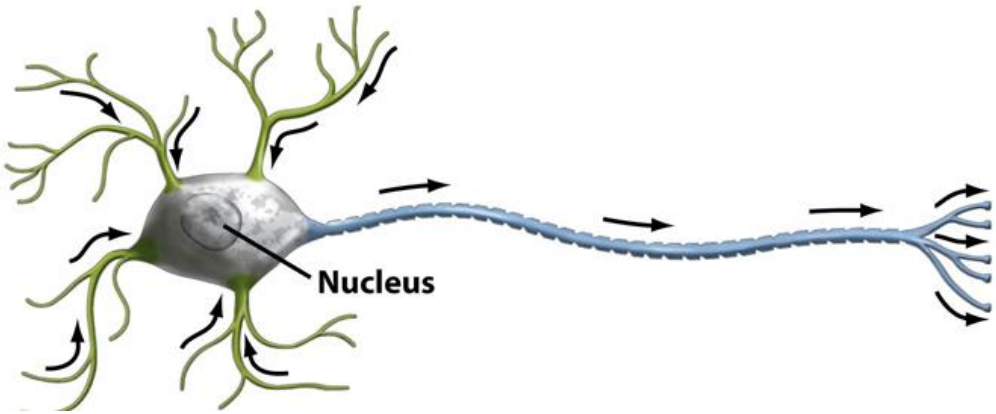
Threshold Logic Unit (TLU)

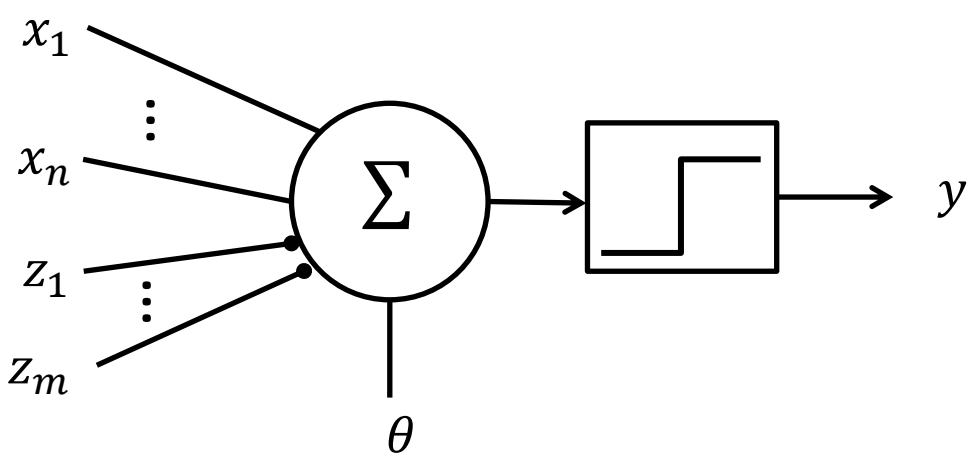


- The first artificial neuron proposed by Warren McCulloch and Walter Pitts in 1943

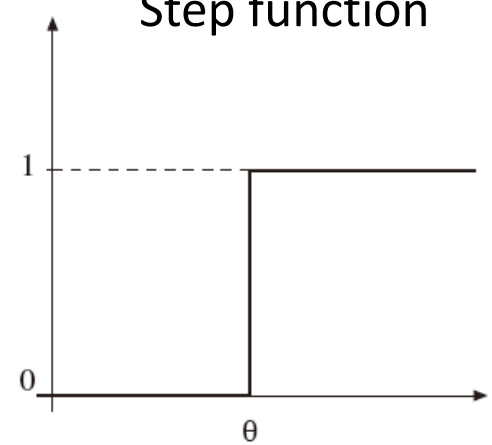
  A Logical Calculus of Ideas Immanent in Nervous Activity, Bulletin of Mathematical Biophysics, 1943
- The model was specifically targeted as a computational model of the "nerve net" in the brain
- Pitts believed that "the brain computing information digital neuron by digital neuron using the exacting implement of mathematical logic", but the experiments on frog's eyes opposed this belief
- Pitts burned his unpublished PhD dissertation

Threshold Logic Unit (TLU)



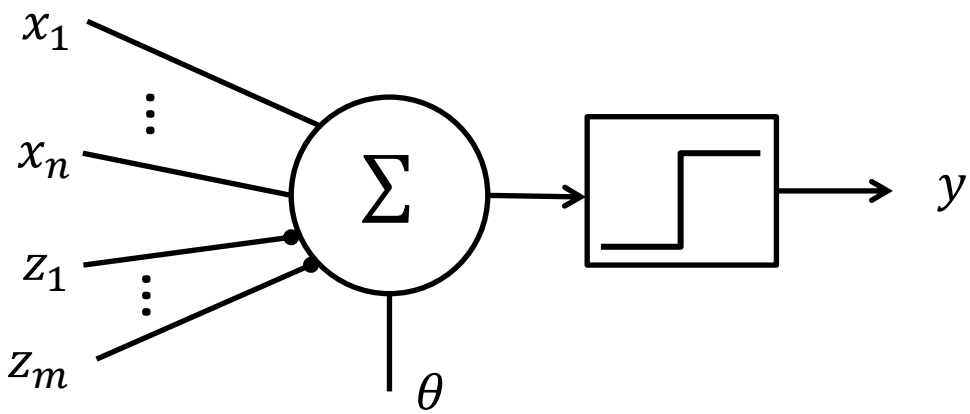


Step function



- Excitatory input x_i
- Inhibitory input z_i
- Binary output y_i
- Threshold θ

McCulloch–Pitts unit (M-P unit)



- If at least one of $z_1, z_2, \ldots, z_m$ is 1, the unit is inhibited and $y = 0$
- Otherwise the total excitation $T = \sum_{i=1}^{n} x_i$ is computed and compared with the threshold θ of the unit (if $n = 0$ then $x = 0$)
  - If $T \geq \theta$ the unit fires a 1
  - If $T < \theta$ the result is 0.
- The MP unit can be inactivated by a single inhibitory signal, as is the case with some real neurons

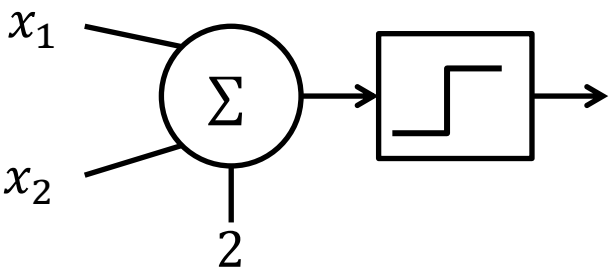
Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

Synthesis of Boolean functions

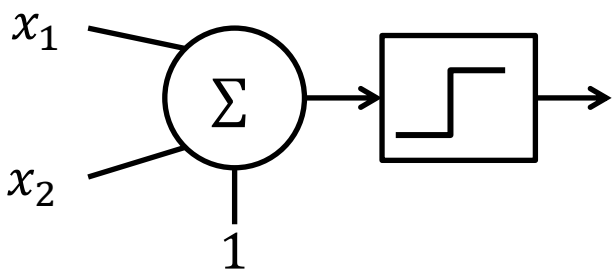
Boolean function: $\{0,1\}^n \rightarrow \{0,1\}$

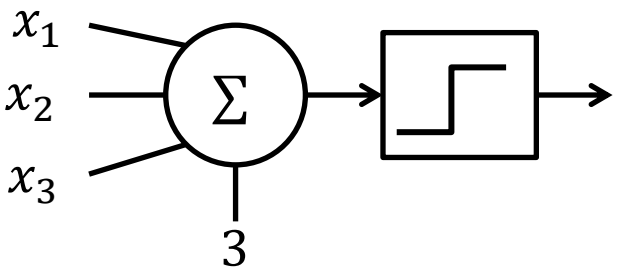
- Conjunction, disjunction, negation

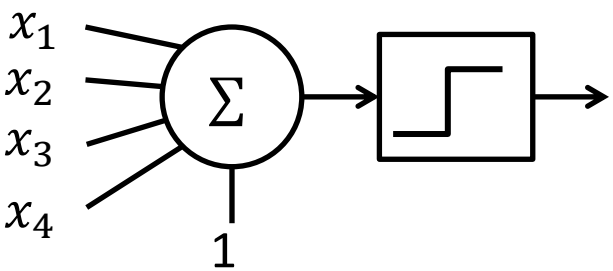
AND



OR







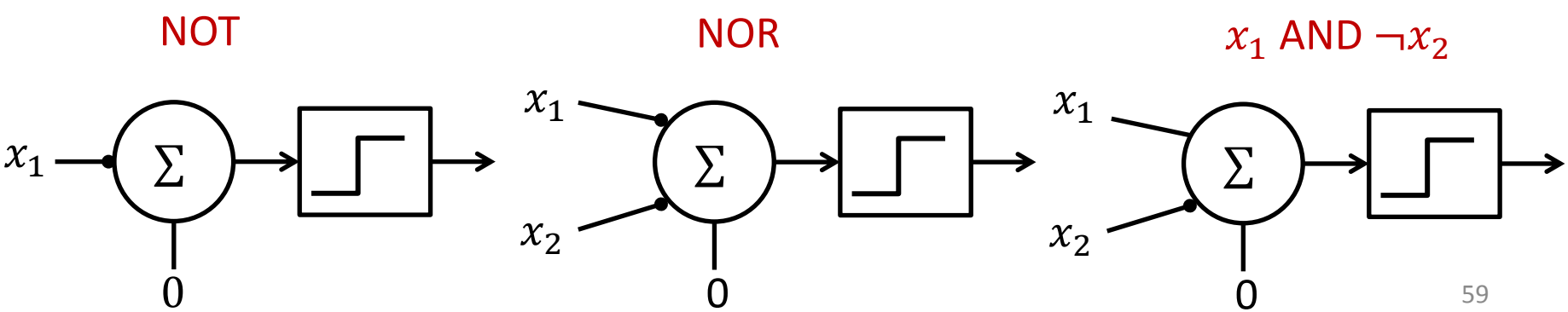
Can you implement negation using the M-P units?

Monotonic logical function

A monotonic logical function f of n arguments is one whose value at two given n-dimensional points $\boldsymbol{x} = (x_1, \dots, x_n)$ and $\boldsymbol{y} = (y_1, \dots, y_n)$ is such that $f(\boldsymbol{x}) \geq f(\boldsymbol{y})$ whenever the number of ones in the input $\boldsymbol{y}$ is a subset of the ones in the input $\boldsymbol{x}$.

Proposition 1. *Uninhibited threshold logic elements of the McCulloch–Pitts type can only implement monotonic logical functions.*

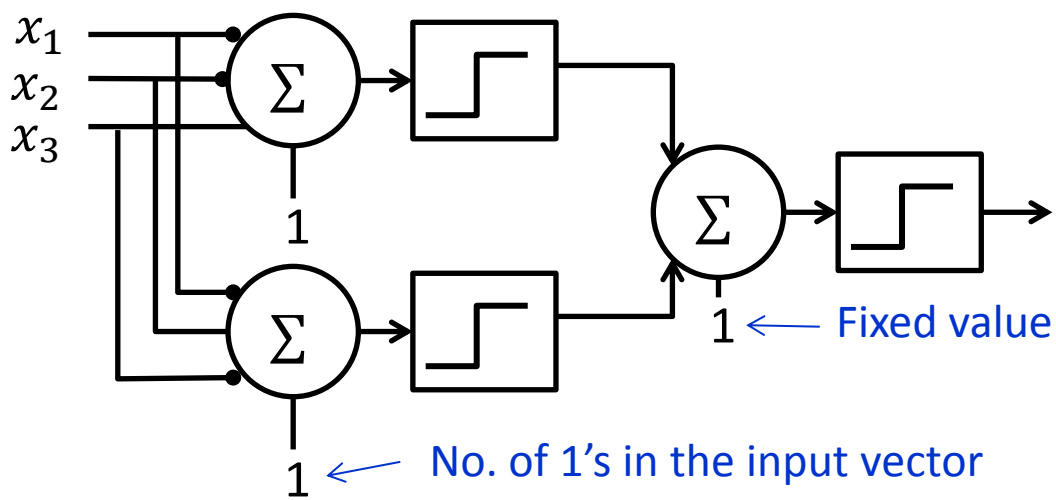
Proof. See (Rojas: Neural Networks,1996)



Constructive synthesis

- Every logical function of n variables can be written in tabular form. Suppose there are K rows that have results 1
  ① Use a M-P unit to represent n values which lead to the result of 1
  ② Use a disjunction unit to connect the K M-P unit
- For example, let $n = 3$ and the table is as follows

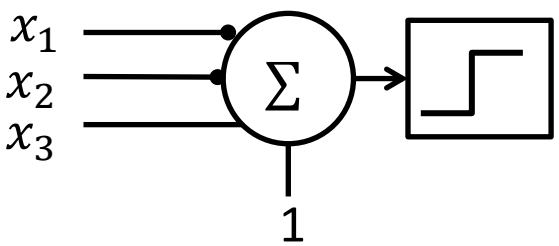
input vectors	F
(0,0,1)	1
(0,1,0)	1
all others	0



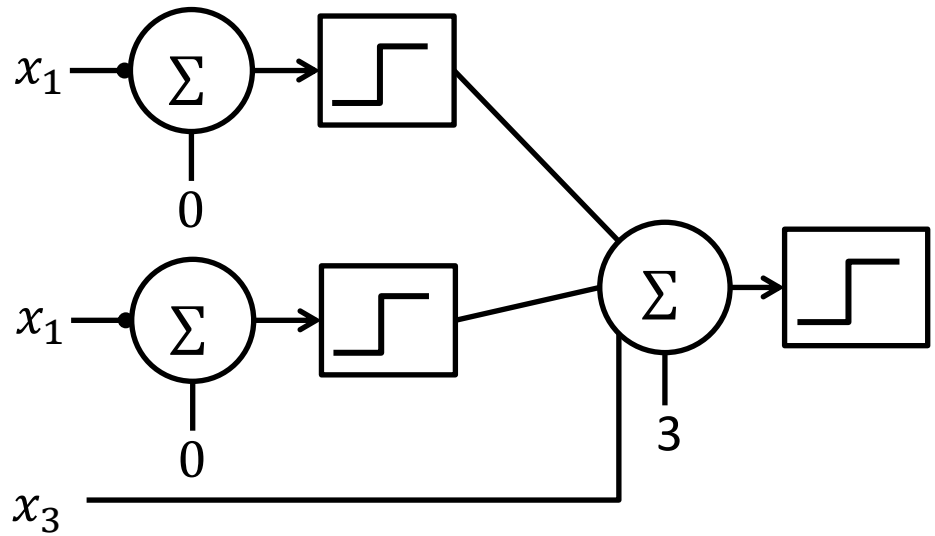
Proposition 2. *Any logical function $F : \{0,1\}^n \to \{0,1\}$ can be computed with a M-P network of two layers.*

Constructive synthesis

What are basic components to construct all logical functions?



Inhibitory connections in the decoders can be replaced with a negation gate



Proposition 3. *All logical functions can be implemented with a network composed of units which exclusively compute the AND, OR, and NOT functions*

Neural networks in the early stage

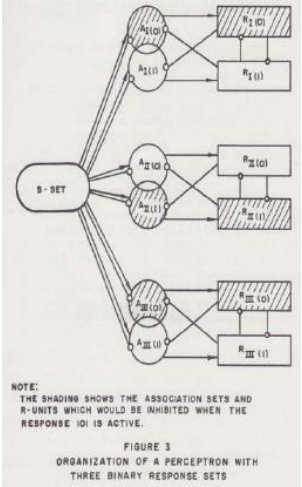
PERCEPTRON

Frank Rosenblatt and Perceptron

- The quest
  - How is information about the physical world sensed by the biological system?
  - In what form is information stored and retrieved?
  - How does remembered information influence recognition and behavior?
- In 1957, Rosenblatt proposed the Perceptron algorithm

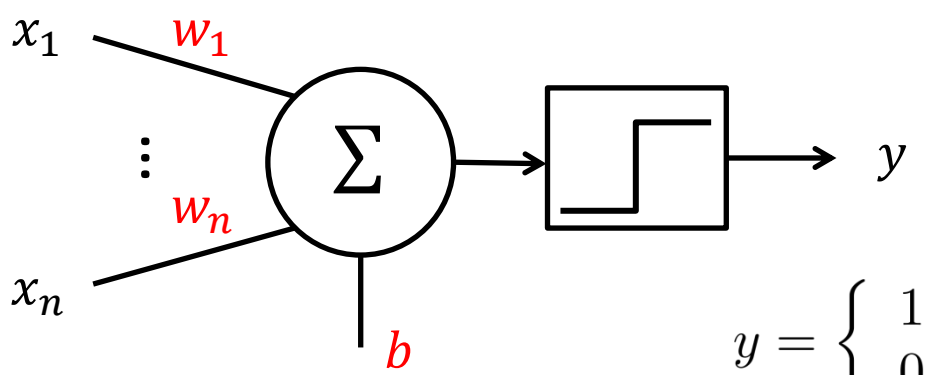
  Rosenblatt (1957) The Perceptron--a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory.
- Variants, applications and theoretical results were developed in 1960s
- A machine was built

1928 -1971



By George Nagy in 2011, Rosenblatt's PhD student

Perceptron

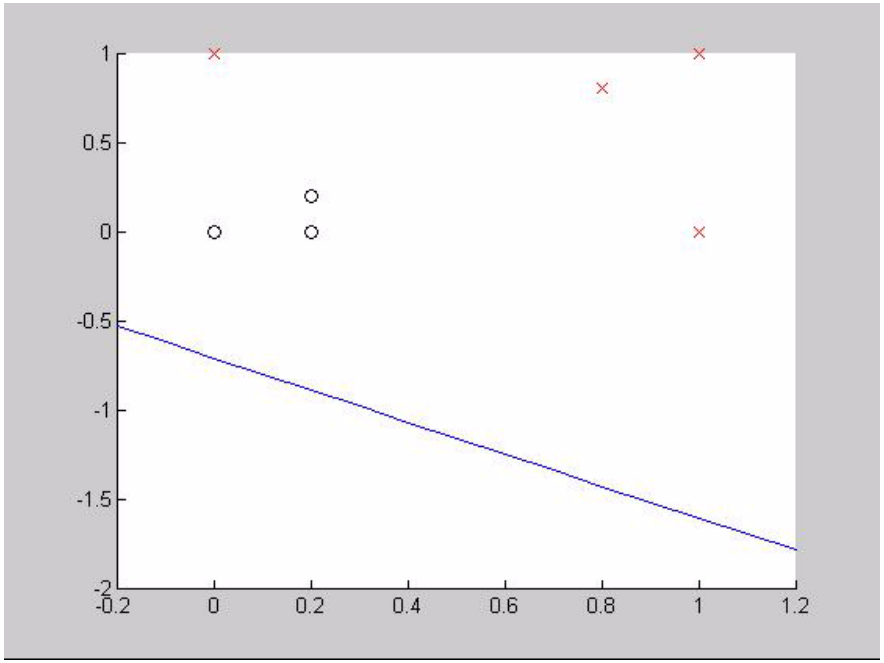


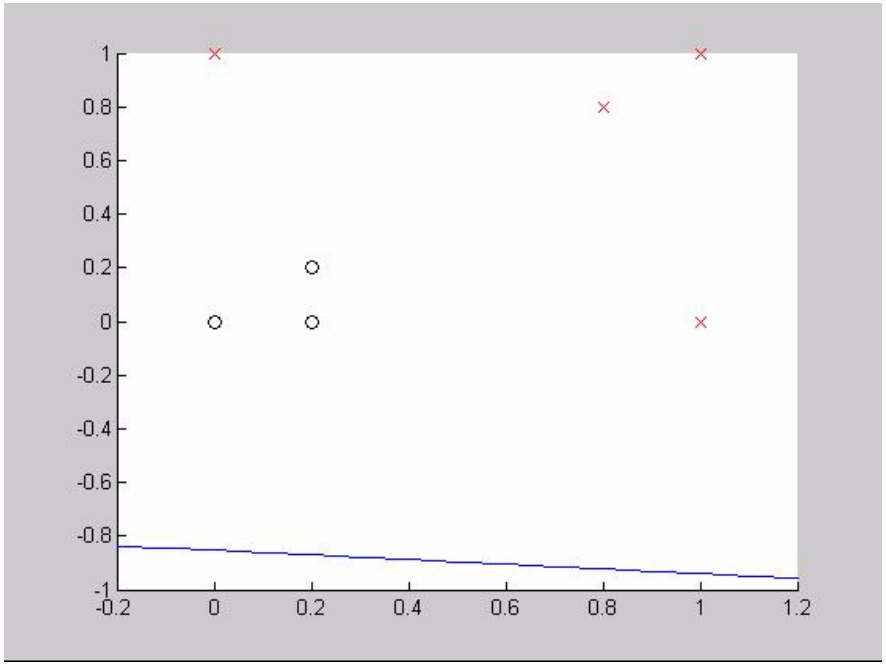
$$y = \begin{cases} 1 & \boldsymbol{w}^\top \boldsymbol{x} + b > 0 \\ 0 & \text{else} \end{cases}$$

- Add weights to the input connections of the M-P unit
- Propose a supervised learning algorithm: For each data points $\boldsymbol{x}^{(j)} \in R^m$ and the corresponding labels $t^{(j)}$
  - Calculate the actual output $y^{(j)}$
  - Update the weights: $\boldsymbol{w}^{\text{new}} = \boldsymbol{w}^{old} + \eta\left(t^{(j)} - y^{(j)}\right)\boldsymbol{x}^{(j)}$;
    $$b^{\text{new}} = b^{old} + \eta\left(t^{(j)} - y^{(j)}\right)$$

  where $\eta > 0$ is the learning rate

Example



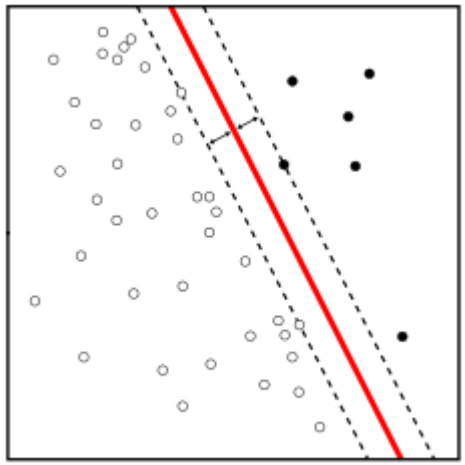


From two different sets of initial weights

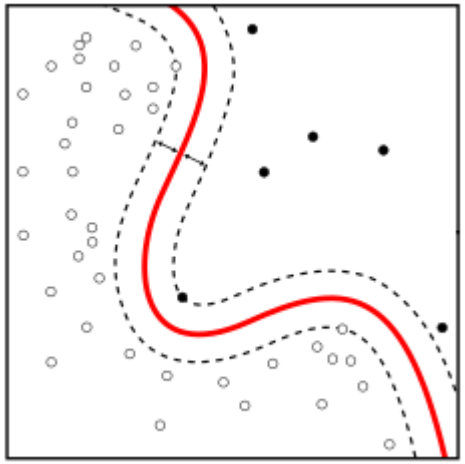
Convergence

Proposition 4: If the training set is linearly separable, then the perceptron is guaranteed to converge. Furthermore, there is an upper bound on the number of times the perceptron will adjust its weights during the training.

Proof. See (Novikoff, 1962)



linearly separable

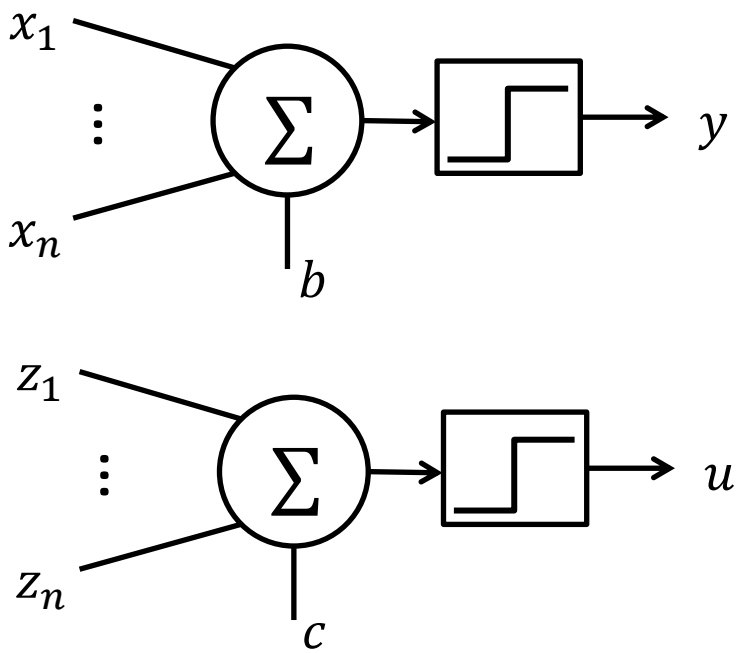


linearly non-separable

Disadvantage: the decision boundary is often close to the training samples, therefore sensitive to noise

Multiple Perceptrons in one layer

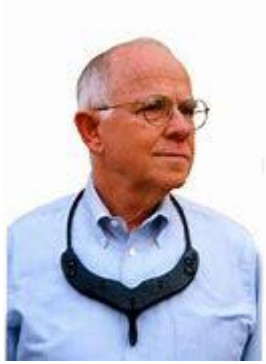
- When multiple Perceptrons are combined, each output neuron operates independently of all the others; thus, learning each output can be considered in isolation



Neural networks in the early stage

ADALINE

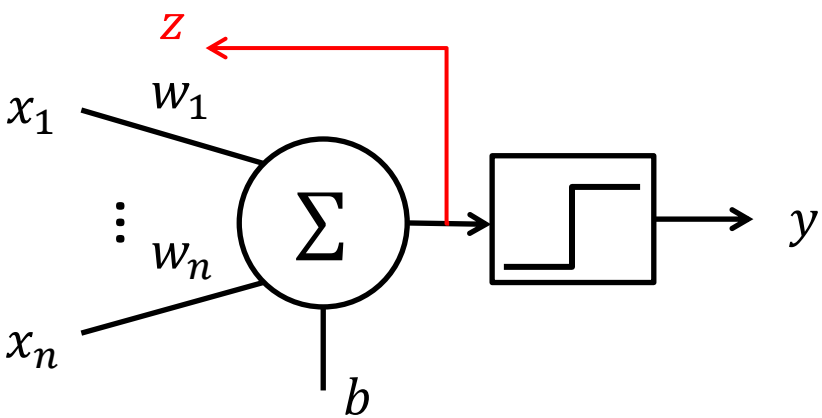
Bernard Widrow and ADALINE



Born in 1929

- Widrow and his doctoral student Ted Hoff invented the least mean squares filter (LMS) adaptive algorithm in 1960
- The LMS algorithm led to the ADALINE and MADALINE and to the backpropagation technique
- ADALINE (*Adaptive Linear Neuron* or *Adaptive Linear Element*): A single-layer model
- LMS algorithm minimizes the mean squared error (MSE), and is a stochastic gradient descent (SGD) method
  - It was proposed for signal processing and achieved great success in that field, but not so successful in training muli-layer neural networks
- In early 1960 Widrow turned to study signal processing, and returned to neural networks in 1980s

ADALINE



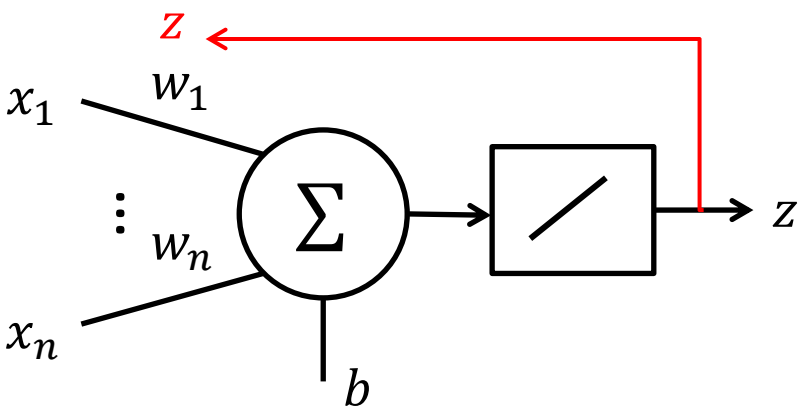
$$y = \begin{cases} 1 & \boldsymbol{w}^\top \boldsymbol{x} + b > 0 \\ 0 & \text{else} \end{cases}$$

- Same architecture as Perceptron; different training algorithm
  - $z = \boldsymbol{w}^\top \boldsymbol{x} + b$ instead of y is used to adjust the weights and bias
- Minimize MSE $E = \sum_j \left(t^{(j)} - z^{(j)}\right)^2$. The learning algorithm:

$$\boldsymbol{w}^{\text{new}} = \boldsymbol{w}^{old} + \eta\left(t^{(j)} - z^{(j)}\right)\boldsymbol{x}^{(j)}$$
$$b^{\text{new}} = b^{old} + \eta\left(t^{(j)} - z^{(j)}\right)$$

  where $\eta > 0$ is the learning rate
- Different names: LMS rule, Delta rule, Widrow-Hoff rule, actually SGD

Another view

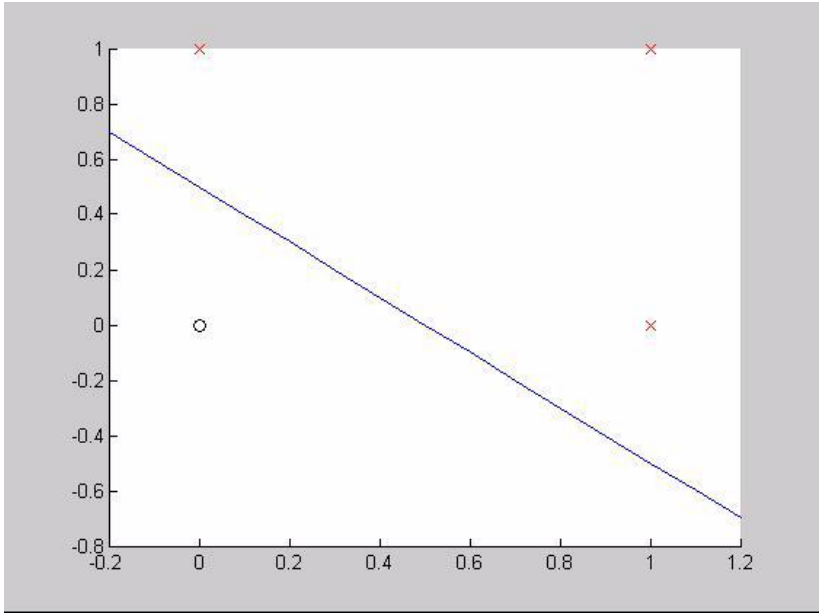


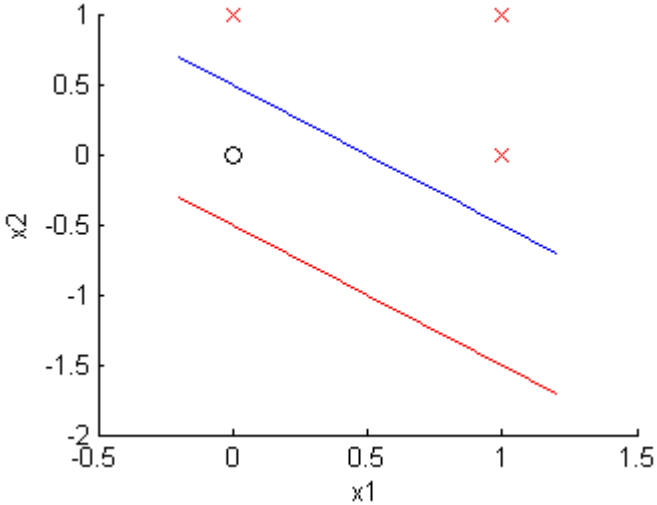
$$z = \boldsymbol{w}^\top \boldsymbol{x} + b$$

$$y = \begin{cases} 1 & z > 0 \\ 0 & \text{else} \end{cases}$$

- There is a linear activation function for the variable z
  - This is where the name *Adaptive Linear Neuron* comes
- The step function is only used for output y and the output is not involved in the learning process

Example





Blue: $\boldsymbol{w} = [1\ 1], b = -0.5$
Red (optimal): $\boldsymbol{w} = [0.5\ 0.5], b = 0.25$
Loss for blue: 0.25
Loss for Red: 0.0625

What's the problem?

Doesn't the algorithm converge to the minimum of the MSE?

MADALINE

- MADALINE: Many ADALINE
- A 3-layer (input, hidden, output), fully connected, feed-forward architecture for classification that uses ADALINE units in its hidden and output layers
- Three different training algorithms for MADALINE networks were proposed
  - The first dates back to 1962 and cannot adapt the weights of the hidden-output connection
  - The second training algorithm improved on Rule I and was described in 1988
  - The third "Rule" applied to a modified network with sigmoid activations instead of step function; it was later found to be equivalent to backpropagation

See Wikipedia and references therein

Summary

- Math basics
  - Linear algebra
  - Probability theory
  - Optimization
- Machine learning basics
  - Learning algorithms, task T, performance P and experience E
  - Capacity versus generalization
  - Maximum likelihood estimation
  - SGD
- Neural networks in the early stage
  - MP unit
    - Logic unit
  - Perceptron
    - Binary classification, convergence
  - Adaline
    - Widrow-Hoff rule

References

- Chapters 2-5 in Deep Learning by Goodfellow, Bengio and Courville, 2016, MIT Press
- MP unit material on Web Learning