Predictive Learning from Data

LECTURE SET 5

Nonlinear Optimization Strategies

Cherkassky, Vladimir, and Filip M. Mulier. *Learning from data: concepts, theory, and methods*. John Wiley & Sons, 2007.

Source: Dr. Vladimir Cherkassky (revised by Dr. Hsiang-Han Chen)

OUTLINE

Objectives

- describe common optimization strategies used for nonlinear (adaptive) learning methods
- introduce terminology
- comment on implementation of model complexity control

- **Nonlinear Optimization in learning**
- Optimization basics (Appendix A)
- Stochastic approximation (gradient descent)
- Iterative methods
- Greedy optimization
- Summary and discussion

Optimization

- Optimization is concerned with the problem of determining extreme values (maxima or minima) of a function on a given domain.
- Let *f*(x) be a real-valued function of real variables x_1, x_2, …, x_m,
- If x* minimizes unconstrained function *f*(x), then the gradient of *f*(x) evaluated at x* is zero

$$\nabla f(\mathbf{x}) = 0$$

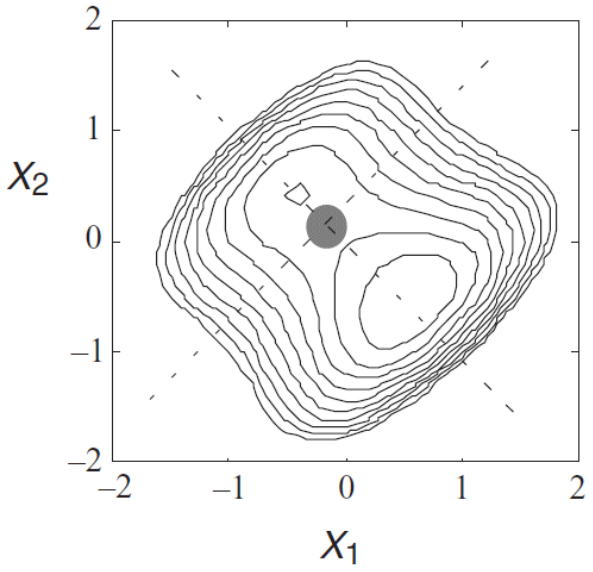
- That is, x* is a solution of the system of equations

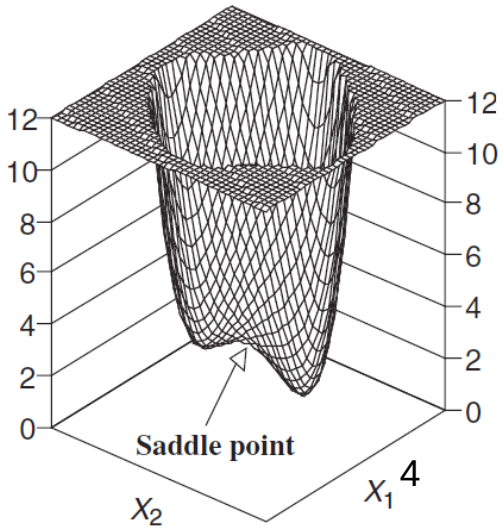
$$\frac{\partial f(\mathbf{x})}{\partial x_i} = 0 \qquad (i = 1, 2, \ldots, m)$$

Optimization

- The point where $\nabla f(\mathbf{x}) = 0$ is called a stationary or critical point;
- It can be a (local) minimum, a maximum, or a saddle point of *f*(x)

The saddle point is a local minimum along the line x2 = -x1 and a local maximum in the orthogonal direction.





Optimization

A critical point x* can be further checked for optimality by considering the Hessian matrix of second partial derivatives:

$$\{\mathbf{H}_f(\mathbf{x})\}_{ij} = \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j}$$

evaluated at x*. At a critical point x* where the gradient is zero:

- If $\mathbf{H}_f(\mathbf{x})$ is positive definite, then $\mathbf{x}^*$ is a minimum of f
- If $\mathbf{H}_f(\mathbf{x})$ is negative definite, then $\mathbf{x}^*$ is a maximum of f
- If $\mathbf{H}_f(\mathbf{x})$ is indefinite, then $\mathbf{x}^*$ is a saddle point of f

Optimization

- With nonlinear optimization, there is always a possibility of several local minima and saddle points.
- This has two important implications:
  1. An optimization algorithm can find, at best, only a local minimum.
  2. The local minimum found by an algorithm is likely to be close to an initial point x_0.

The chances for obtaining globally optimal solution can be improved (but not assured) by brute-force computational techniques.

EX: Restarting optimization with many (randomized) initial points and/or using simulated annealing to escape from local minima.

Optimization in predictive learning

- Optimization is used in learning methods for parameter estimation.
- Recall implementation of SRM:

  - fix complexity (VC-dimension)

  - minimize empirical risk
- Two related issues:

  - parameterization (of possible models)

  - optimization method
- Many learning methods use dictionary parameterization

$$f_m(\mathbf{x}, \mathbf{w}, \mathbf{V}) = \sum_{i=0}^{m} w_i \, g(\mathbf{x}, \mathbf{v}_i)$$

- Optimization methods vary

Nonlinear Optimization

- The ERM approach

$$R_{emp}(\mathbf{V},\mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n} L\left(y_i, f\left(\mathbf{x}_i, \mathbf{V}, \mathbf{W}\right)\right)$$

where the model

$$f(\mathbf{x},\mathbf{V},\mathbf{W}) = \sum_{j=1}^{m} w_j \, g_j\left(\mathbf{x}, \mathbf{v}_j\right)$$

Two factors contributing to nonlinear optimization

- nonlinear basis functions $g_j\left(\mathbf{x}, \mathbf{v}_j\right)$

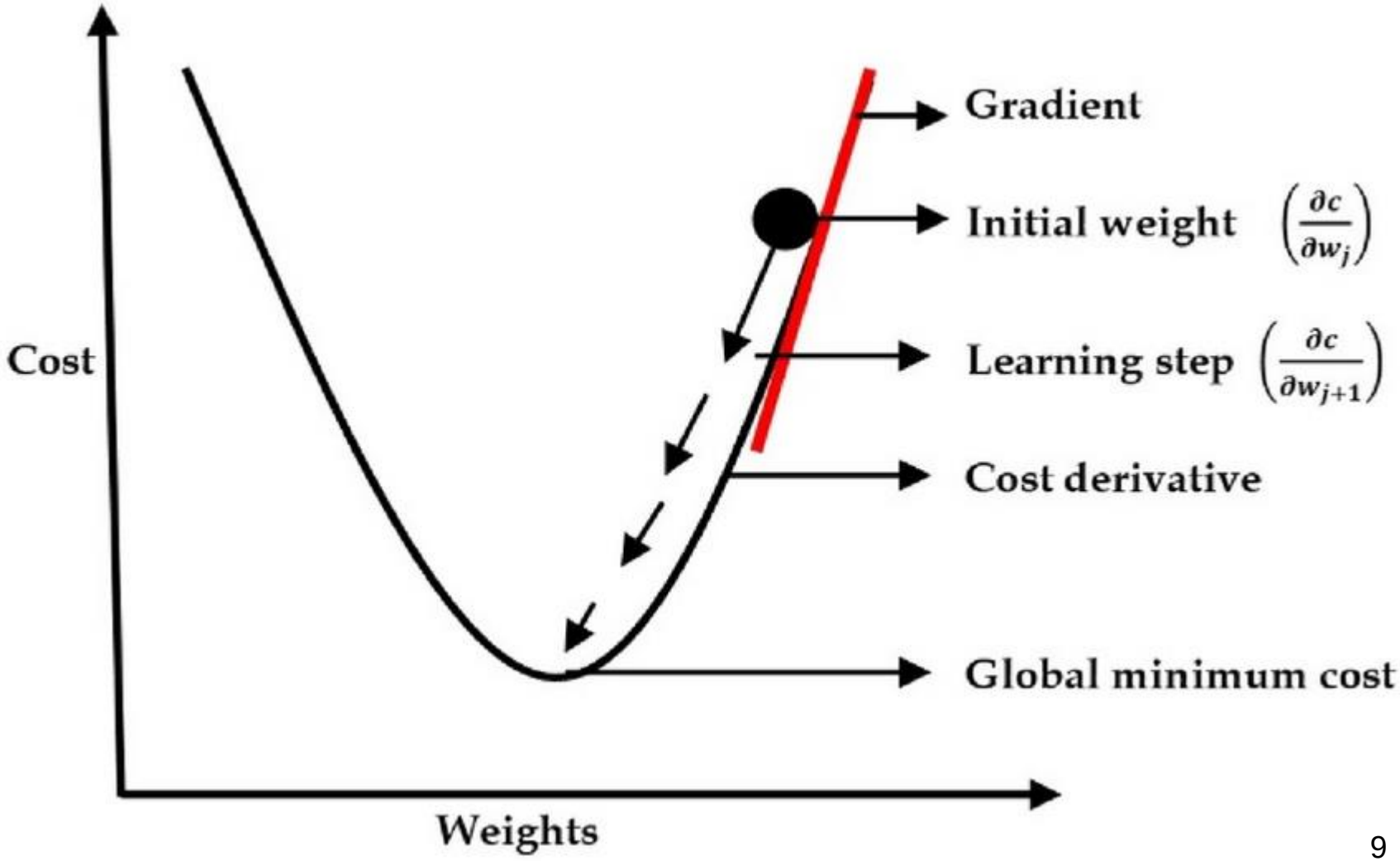
- non-convex loss function *L*

  Examples

  convex loss: squared error, least-modulus

  non-convex loss: 0/1 loss for classification

Nonlinear Optimization - convex



https://www.researchgate.net/figure/The-convex-optimization-function-Friedman-2002_fig4_365210227

Unconstrained Minimization

- ERM or SRM (with squared loss) lead to unconstrained convex minimization
- Minimization of a real-valued function of many input variables

  → Optimization theory
- We discuss only *popular* optimization strategies developed in statistics, machine learning and neural networks
- These optimization methods have been introduced in various fields and usually use specialized terminology

- **Dictionary representation** $f_m(\mathbf{x}, \mathbf{w}, \mathbf{V}) = \sum_{i=0}^{m} w_i \, g(\mathbf{x}, \mathbf{v}_i)$

Two possibilities

- **Linear (non-adaptive) methods**

  ~ predetermined (fixed) basis functions $g_i(\mathbf{x})$

  → only parameters w_i have to be estimated

  via standard optimization methods (linear least squares)

Examples: linear regression, polynomial regression

linear classifiers, quadratic classifiers

- **Nonlinear (adaptive) methods**

  ~ basis functions $g(\mathbf{x}, \mathbf{v}_i)$ depend on the training data

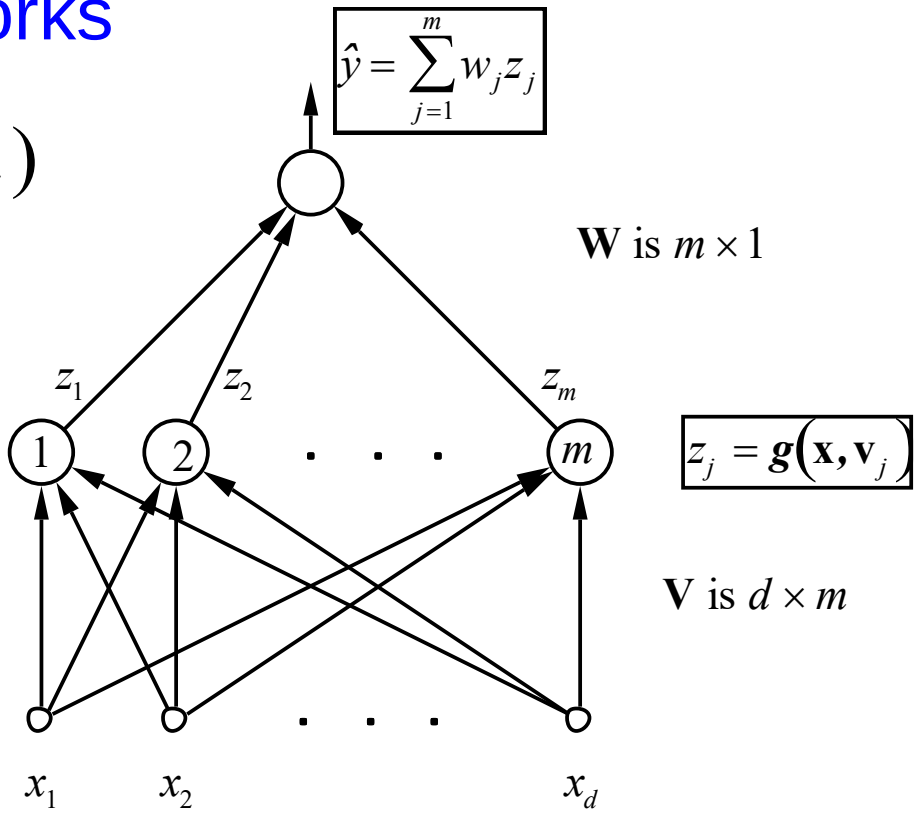
Possibilities: nonlinear b.f. (in parameters $\mathbf{v}_i$)

feature selection (i.e. wavelet denoising), etc.

Nonlinear Parameterization: Neural Networks

- MLP or RBF networks

$$f_m(\mathbf{x}, \mathbf{w}, \mathbf{V}) = \sum_{i=0}^{m} w_i \, g(\mathbf{x}, \mathbf{v}_i)$$



- dimensionality reduction
- universal approximation property
- possibly multiple ‘hidden layers’ – aka “Deep Learning”

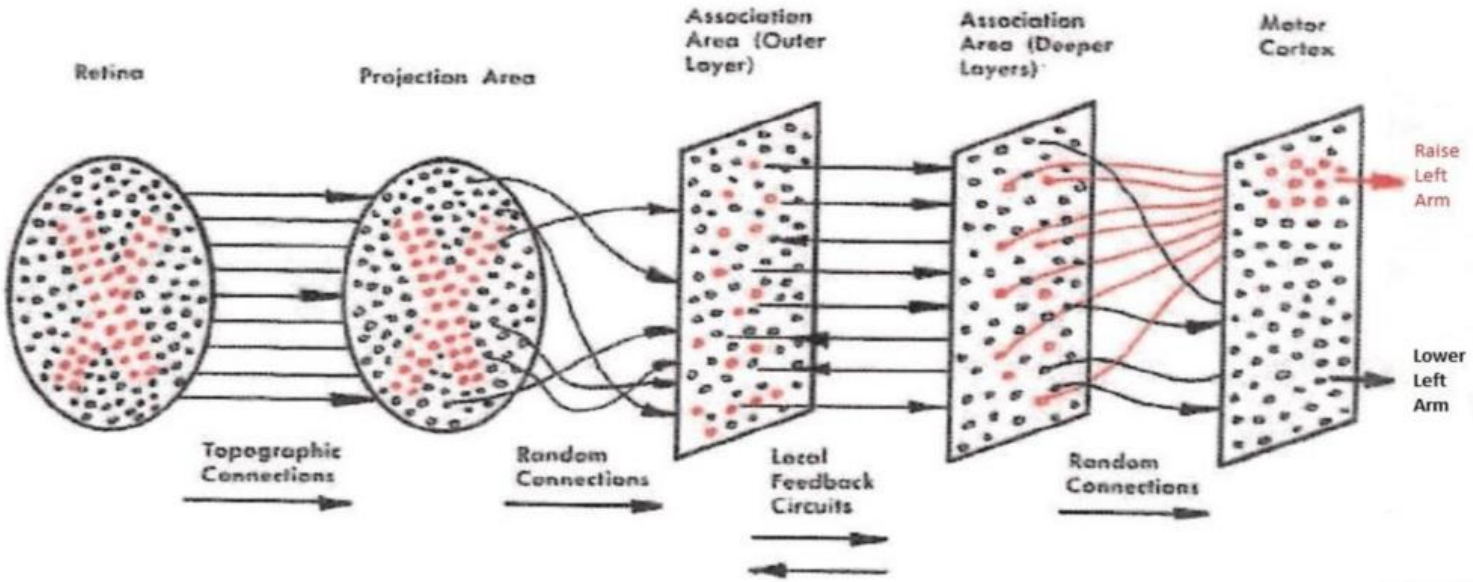


FIG. 1 — Organization of a biological brain. (Red areas indicate active cells, responding to the letter X.)

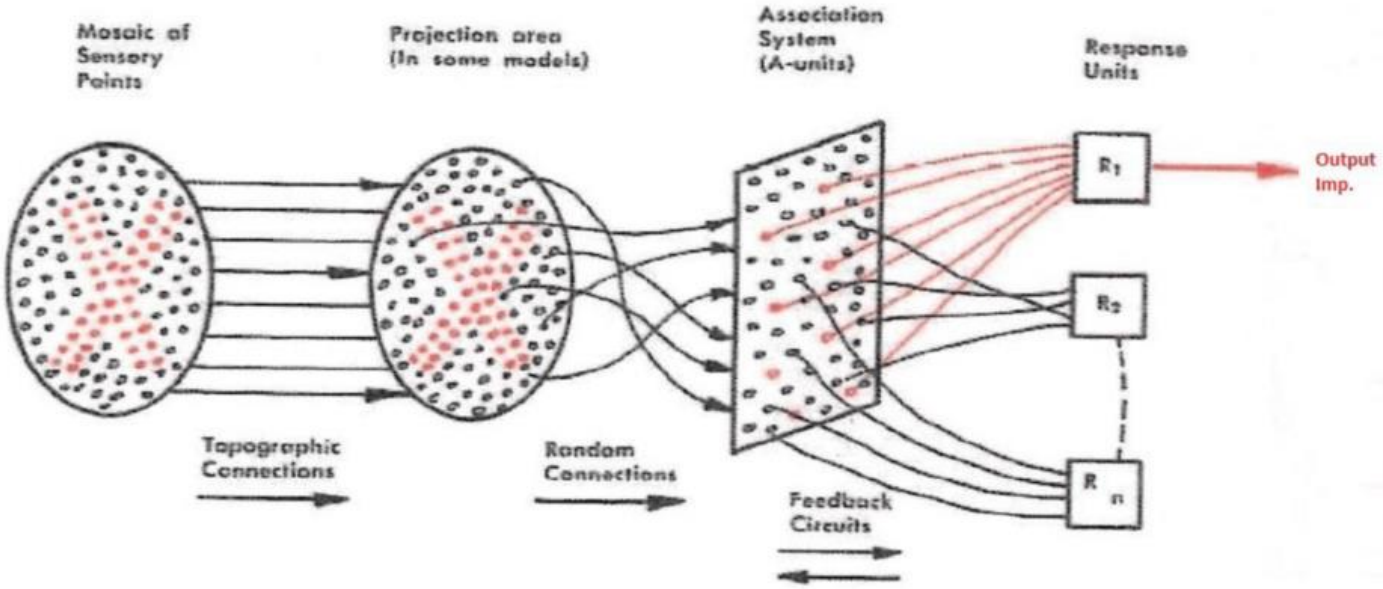


FIG. 6 — Organization of a perceptron.

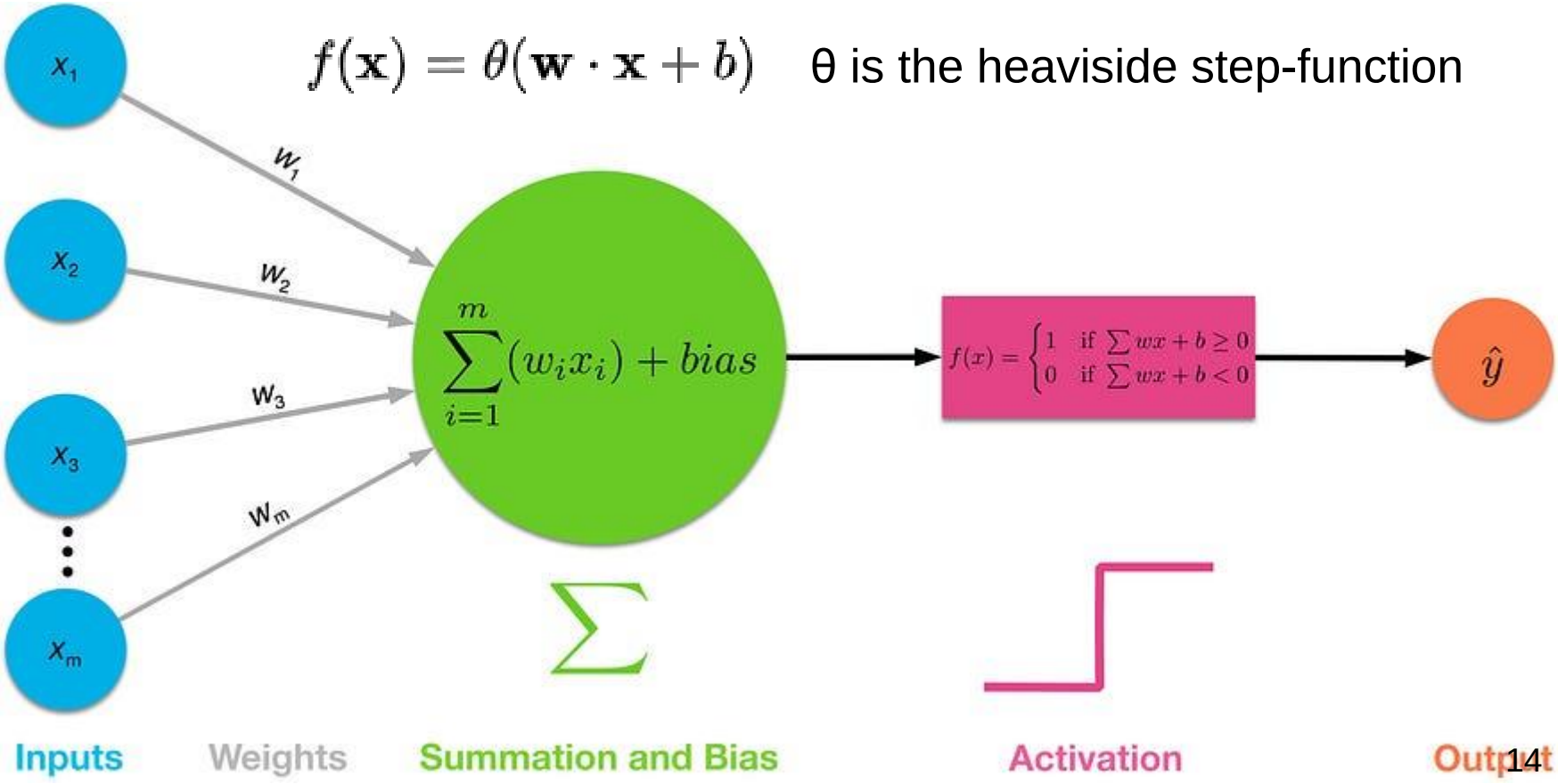
https://en.wikipedia.org/wiki/Perceptron

Perceptron

The perceptron is an algorithm for learning a binary classifier called a threshold function: a function that maps its input *x* (a real-valued vector) to an output value

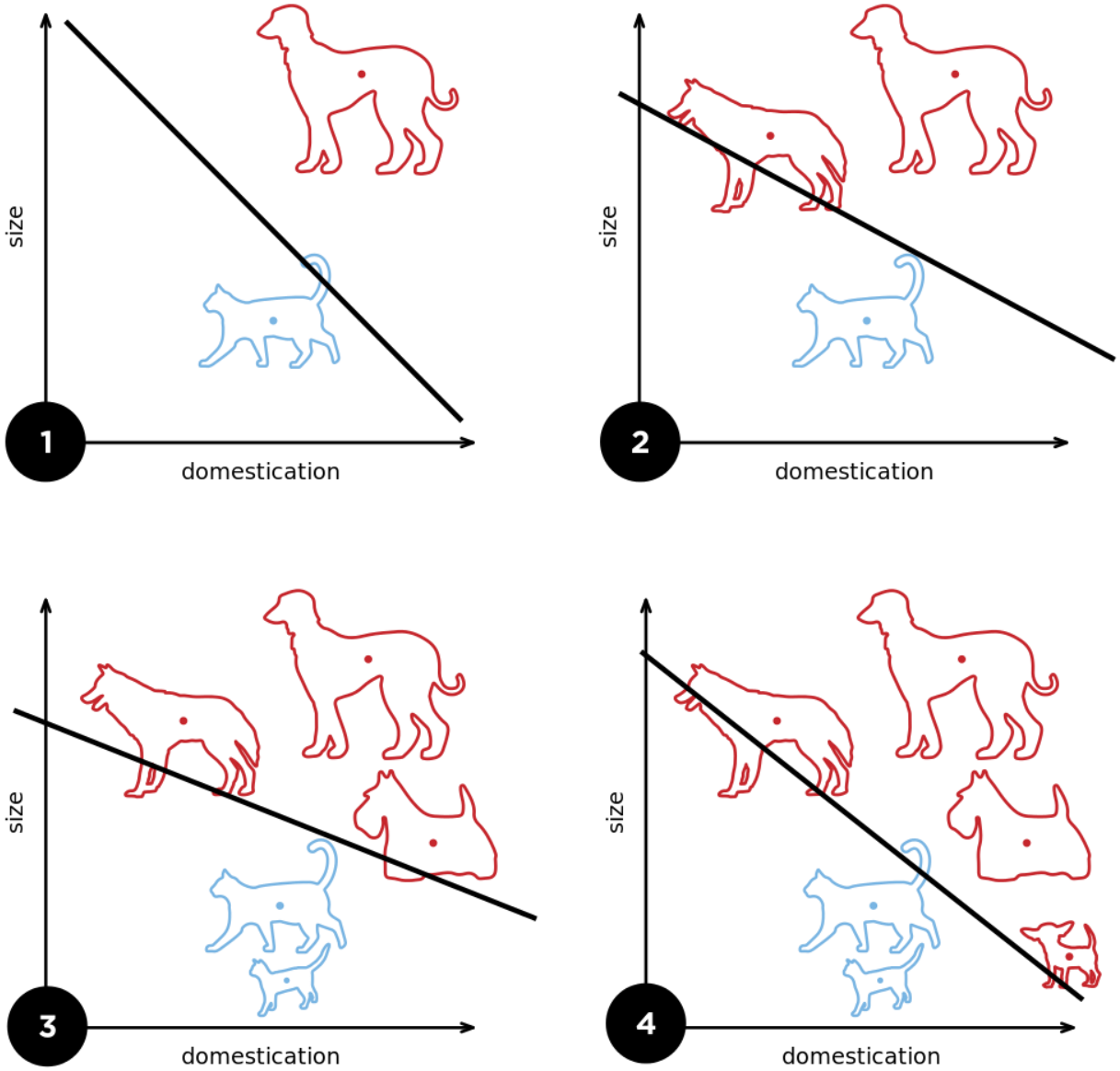
$$f(\mathbf{x}) = \theta(\mathbf{w} \cdot \mathbf{x} + b)$$

θ is the heaviside step-function



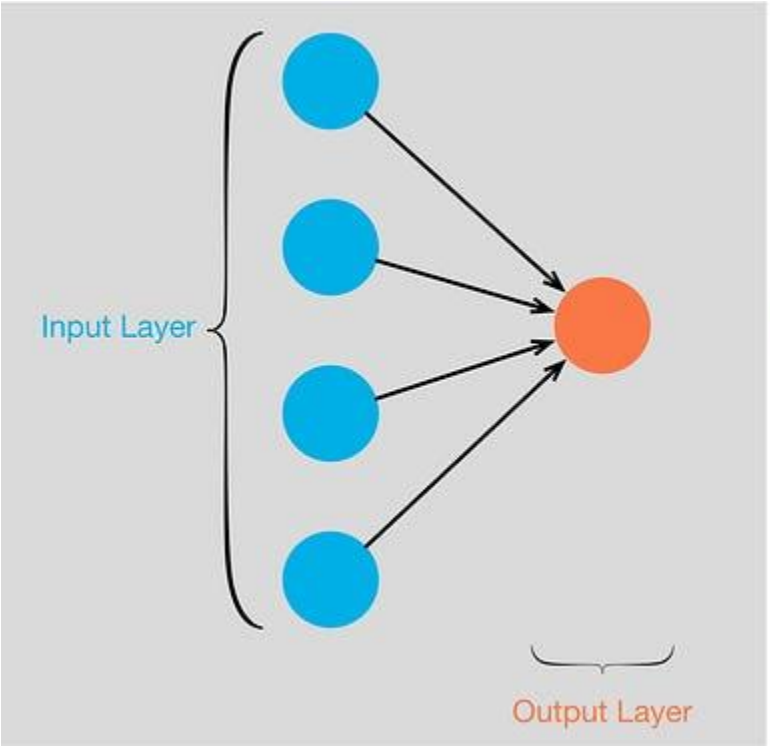
https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

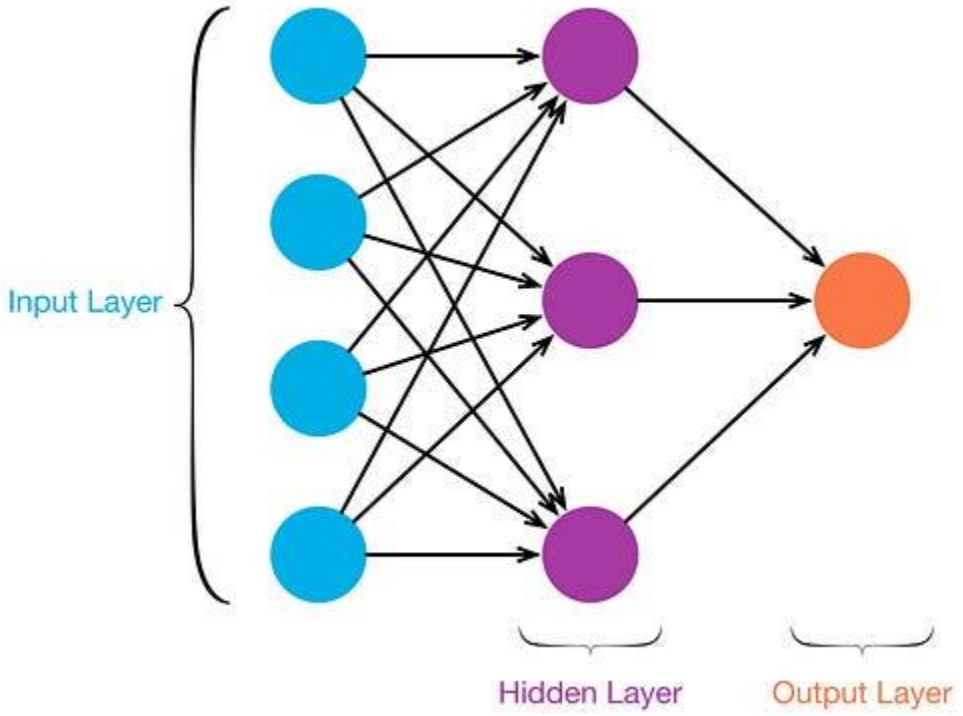
Perceptron



https://en.wikipedia.org/wiki/Perceptron

Single layer to multi-layer





Step function has no useful derivative (its derivative is 0 everywhere or undefined at the 0 point on x-axis). It doesn't work for backpropagation.

Hence, we need a better activation function!

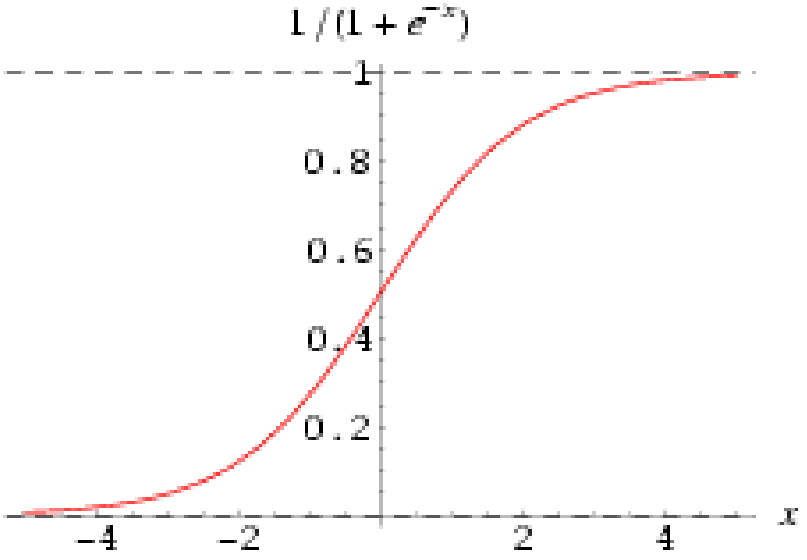
https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

Multilayer Perceptron (MLP) network

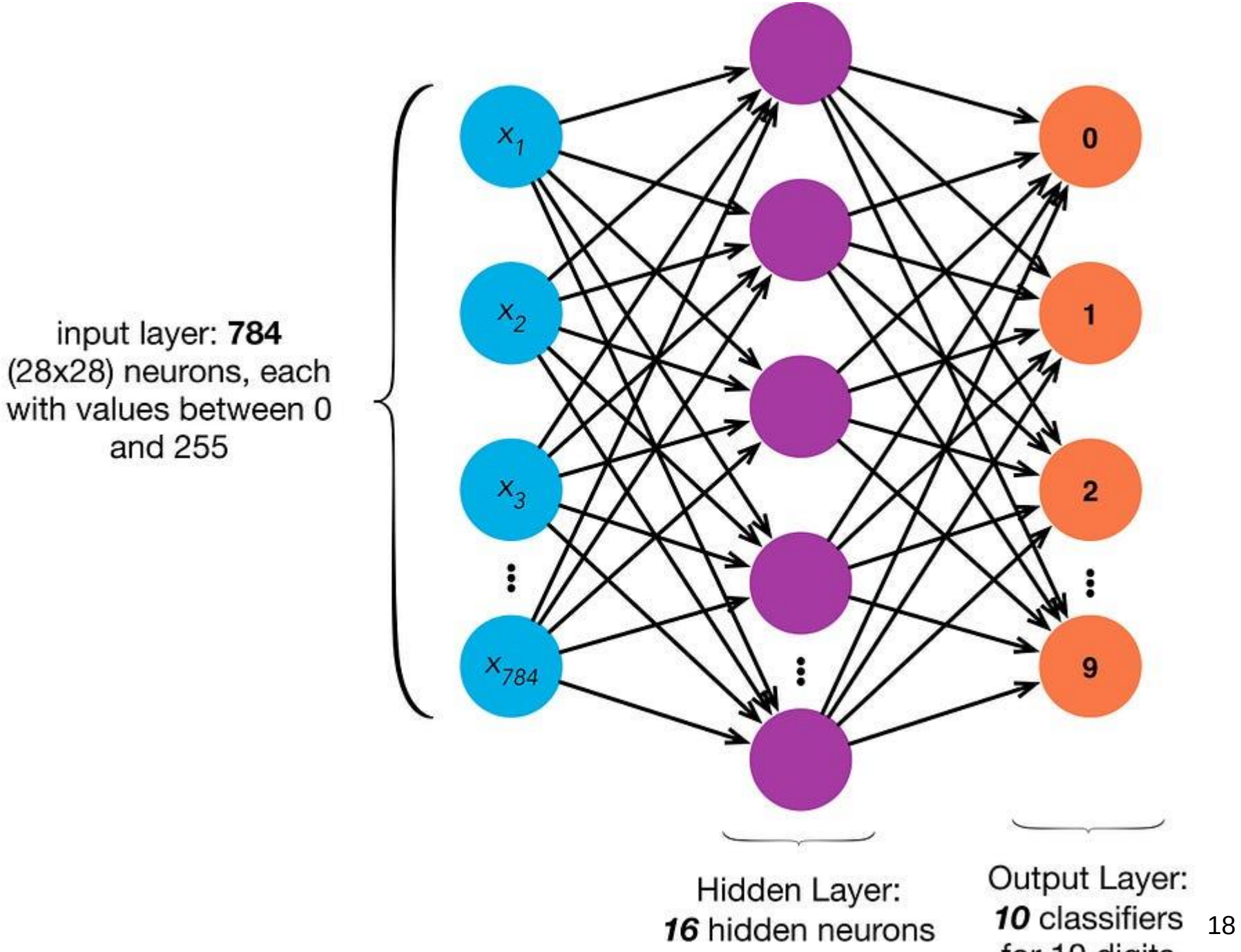
- Basis functions of the form $g_i(t) = g\left((\mathbf{x}\mathbf{v}_i + b_i)\right)$

  i.e. sigmoid aka logistic function

$$s(t) = \frac{1}{1+\exp(-t)}$$



- commonly used in artificial neural networks
- combination of sigmoids ~ universal approximator



https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

Radial basis function

In mathematics a radial basis function (RBF) is a real-valued function whose value depends only on the distance between the input and some fixed point, either the origin or some other fixed point *c* called a center,

$$g_i(t) = g\left(\left\|\mathbf{x} - \mathbf{v}_i\right\|\right)$$

https://en.wikipedia.org/wiki/Radial_basis_function

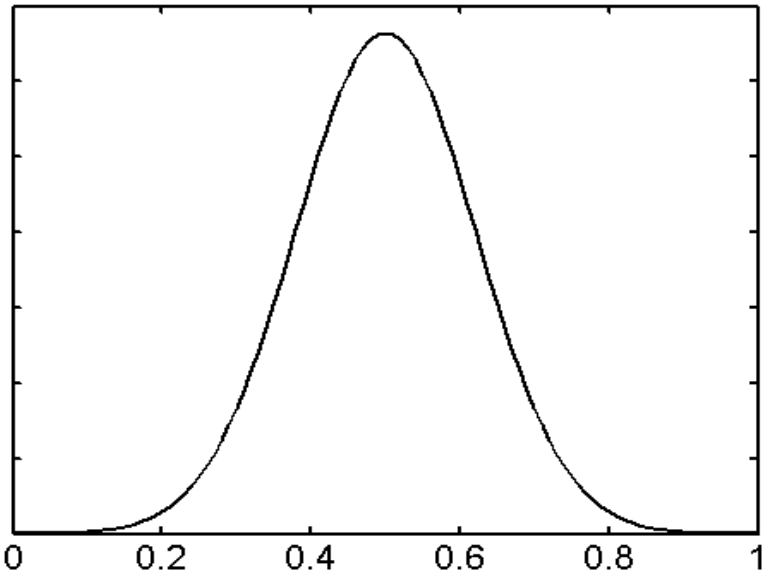
RBF Networks

- Basis functions of the form $g_i(t) = g\left(\left\|\mathbf{x} - \mathbf{v}_i\right\|\right)$
  i.e. Radial Basis Function(RBF)

$$g(t) = \exp\left(-\frac{t^2}{2\alpha^2}\right)$$

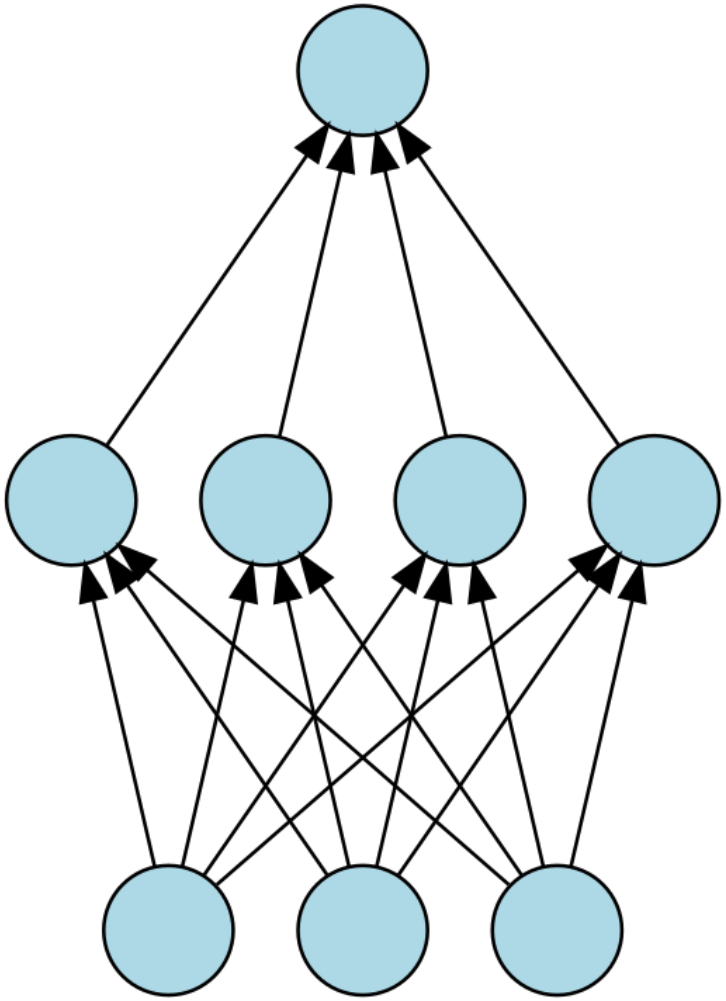
$$g(t) = \left(t^2 + b^2\right)^{-2}$$

$$g(t) = t$$



- RBF adaptive parameters: center, width
- commonly used in artificial neural networks
- combination of RBF's ~ universal approximator

RBF Networks



Output y

Linear weights a_i

Radial basis functions

Parameters c_i

Input x

https://en.wikipedia.org/wiki/Radial_basis_function_network

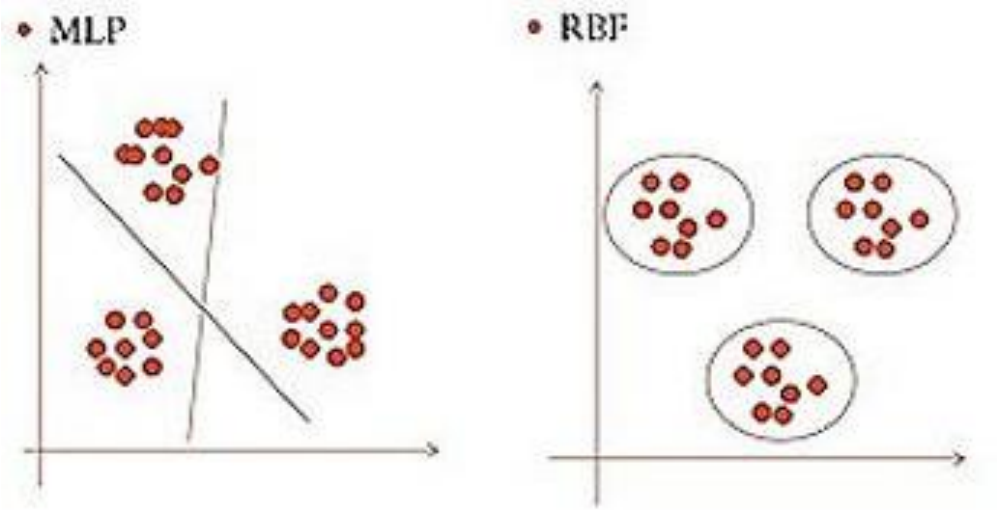
MLP vs. RBF

- MLPs > RBFs

When the underlying characteristics feature of data is embedded deep inside very high dimensional sets for example, in image recognition etc.

- RBFs > MLPs

When low dimensional data where deep feature extraction is not required and results are directly correlated with the components of input vector.



https://www.madrasresearch.org/post/radial-basis-functions-neural-networks

Piecewise-constant Regression (CART)

- Adaptive Partitioning (CART) $f(\mathbf{x}) = \sum_{j=1}^{m} w_j I(\mathbf{x} \in \mathbf{R}_j)$

  each b.f. is a rectangular region in **x**-space

$$I(\mathbf{x} \in \mathbf{R}_j) = \prod_{l=1}^{d} I(a_{jl} \le x_l \le b_{jl})$$

- Each b.f. depends on *2d* parameters $\mathbf{a}_j, \mathbf{b}_j$
- Since the regions $\mathbf{R}_j$ are disjoint, parameters *w* can be easily estimated (for regression) as:

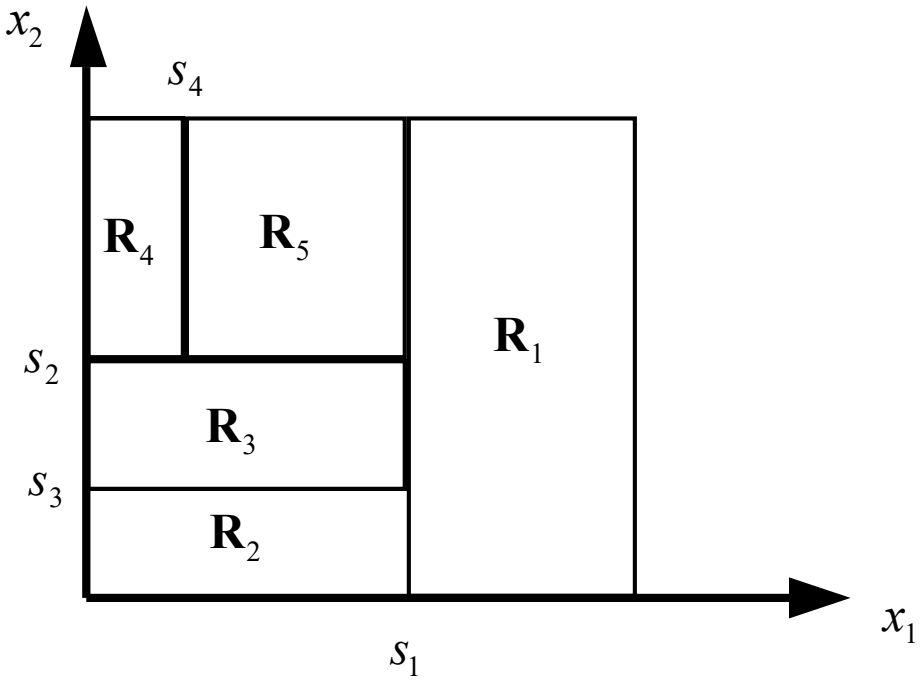
$$w_j = \frac{1}{n_j} \sum_{\mathbf{x}_i \in \mathbf{R}_j} y_i$$

- Estimating b.f.'s ~ adaptive partitioning of **x**-space

Example of CART Partitioning

- CART Partitioning in 2D space

  - each region ~ basis function
  - piecewise-constant estimate of *y* (in each region)
  - number of regions *m* ~ model complexity



OUTLINE

- Objectives
- Nonlinear Optimization in learning
- *Optimization basics (Appendix A)*
- **Stochastic approximation aka gradient descent**
- Iterative methods
- Greedy optimization
- Summary and discussion

Sequential estimation of model parameters

- Batch vs on-line (iterative) learning
  - Algorithmic (statistical) approaches ~ batch
  - Neural-network inspired methods ~ on-line

BUT the only difference is on the implementation level (so both types of methods should yield similar generalization)

- Recall ERM inductive principle (for regression):

$$R_{emp}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} L(\mathbf{x}_i, y_i, \mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \left(y_i - f(\mathbf{x}_i, \mathbf{w})\right)^2$$

- Assume dictionary parameterization with fixed basis fcts

$$\hat{y} = \boldsymbol{f}(\mathbf{x}, \mathbf{w}) = \sum_{j=1}^{m} w_j \boldsymbol{g}_j(\mathbf{x})$$

L = (y - y^) ^ 2 = g(y)
y = f(x, w)

Sequential (on-line) least squares minimization

- Training pairs $\mathbf{x}(k), y(k)$ presented sequentially
- On-line update equations for minimizing empirical risk (MSE) wrt parameters **w** are:

with ref to

finding the way to go based on the gradient

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \gamma_k \frac{\partial}{\partial \mathbf{w}} L(\mathbf{x}(k), y(k), \mathbf{w})$$

how large a leap should take

W*(global opt)

W0(local opt)

(gradient descent learning)

where the gradient is computed via the chain rule:

$$\frac{\partial}{\partial w_j} L(\mathbf{x}, y, \mathbf{w}) = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_j} = 2(\hat{y} - y) g_j(\mathbf{x})$$

the learning rate γ_k is a small positive value (decreasing with *k*)

On-line least-squares minimization algorithm

- Known as delta-rule (Widrow and Hoff, 1960):

Given initial parameter estimates **w**(0), update parameters during each presentation of *k*-th training sample **x**(*k*),*y*(*k*)

- Step 1: forward pass computation

$$z_j(k) = g_j(\mathbf{x}(k)) \qquad j = 1,\ldots,m$$

$$\hat{y}(k) = \sum_{j=1}^{m} w_j(k) z_j(k)$$ - estimated output

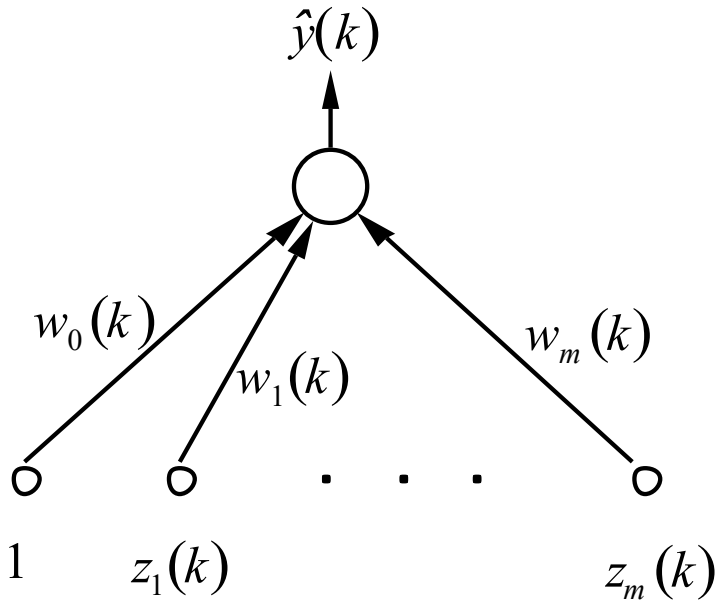
- Step 2: backward pass computation

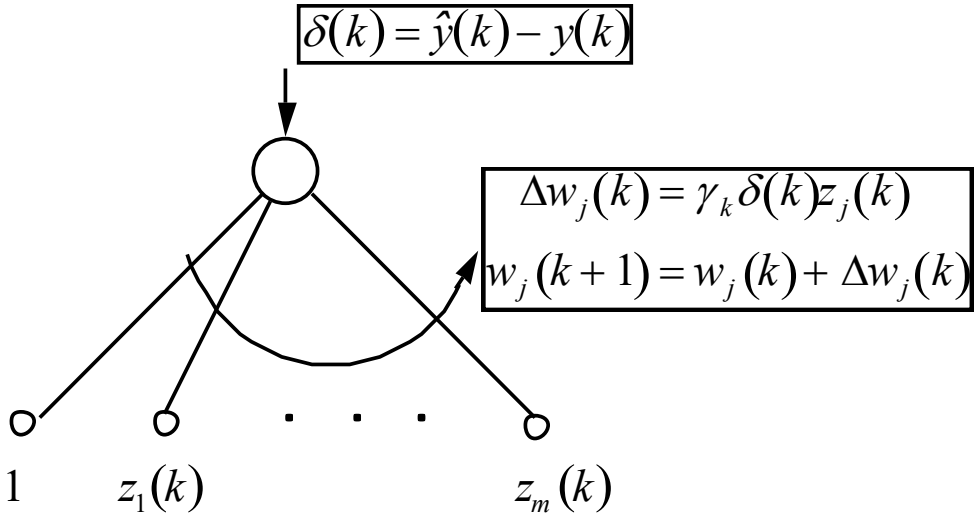
$$\delta(k) = \hat{y}(k) - y(k)$$ - error term (delta)

$$\mathbf{w}_j(k+1) = \mathbf{w}_j(k) - \gamma_k \delta(k) z_j(k), \; j = 1,\ldots,m$$

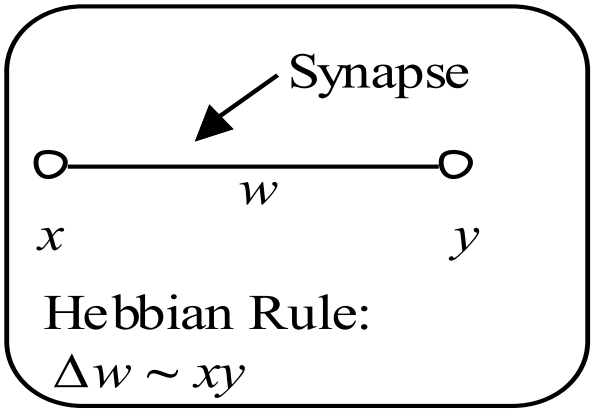
Neural network interpretation of delta rule

- ***Forward pass*** ***Backward pass***





- ***"Biological learning"***
  - parallel+ distributed comp.
  - can be extended to multiple-layer networks



Backpropagation Training of MLP Networks

Consider a set of approximating functions

$$f(\mathbf{x}, \mathbf{w}, \mathbf{V}) = w_0 + \sum_{j=1}^{m} w_j g\left(v_{0j} + \sum_{i=1}^{d} x_i v_{ij}\right)$$

The risk functional is

$$R_{\text{emp}} = \sum_{i=1}^{n} \left(f(\mathbf{x}_i, \mathbf{w}, \mathbf{V}) - y_i\right)^2$$

The stochastic approximation procedure for minimizing this risk with respect to the parameters V and w is

$$\mathbf{V}(k+1) = \mathbf{V}(k) - \gamma_k \text{grad}_{\mathbf{V}} L(\mathbf{x}(k), y(k), \mathbf{V}(k), \mathbf{w}(k)),$$

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \gamma_k \text{grad}_{\mathbf{w}} L(\mathbf{x}(k), y(k), \mathbf{V}(k), \mathbf{w}(k)), k = 1, \ldots, n,$$

Backpropagation Training of MLP Networks

The loss L is

$$L(\mathbf{x}(k), y(k), \mathbf{V}(k), \mathbf{w}(k)) = \tfrac{1}{2}(f(\mathbf{x}, \mathbf{w}, \mathbf{V}) - y)^2$$

The gradient of the loss L can be computed via the chain rule of derivatives if the approximating function is decomposed as

$$f(\mathbf{x}, \mathbf{w}, \mathbf{V}) = w_0 + \sum_{j=1}^{m} w_j g\left(v_{0j} + \sum_{i=1}^{d} x_i v_{ij}\right)$$

$$a_j = \sum_{i=0}^{d} x_i v_{ij}, \quad j = 1, \ldots, m,$$

$$z_j = g(a_j), \quad j = 1, \ldots, m,$$

$$z_0 = 1,$$

$$\hat{y} = \sum_{j=0}^{m} w_j z_j.$$

Backpropagation Training of MLP Networks

Based on the chain rule, the relevant gradients are

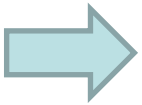
$$\frac{\partial R}{\partial v_{ij}} = \frac{\partial R}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_j} \frac{\partial a_j}{\partial v_{ij}}$$

$$\frac{\partial R}{\partial w_j} = \frac{\partial R}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_j}.$$

We can calculate these partial derivatives using functions in the previous slide.

$$\frac{\partial R}{\partial \hat{y}} = \hat{y} - y \qquad \frac{\partial \hat{y}}{\partial a_j} = g'(a_j) w_j$$

$$\frac{\partial a_j}{\partial v_{ij}} = x_i \qquad \frac{\partial \hat{y}}{\partial w_j} = z_j$$



$$\frac{\partial R}{\partial v_{ij}} = (\hat{y} - y) g'(a_j) w_j x_i$$

$$\frac{\partial R}{\partial w_j} = (\hat{y} - y) z_j.$$

Backpropagation Training of MLP Networks

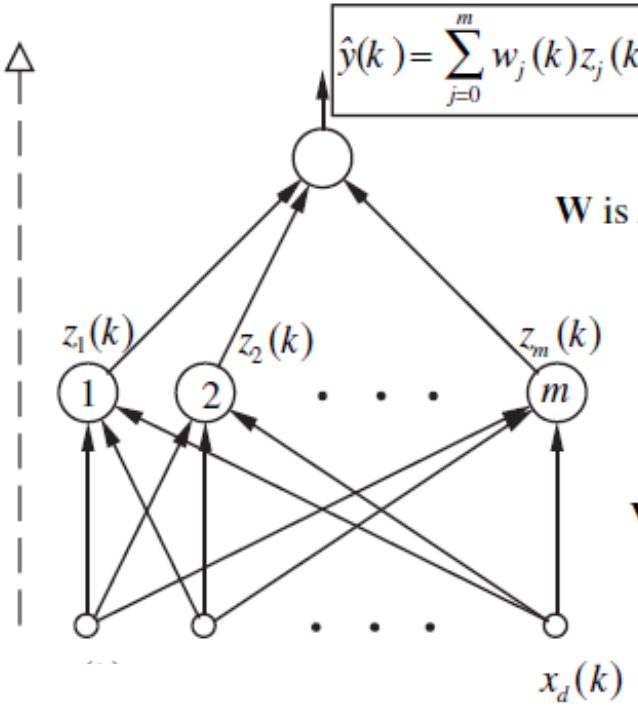
- With these gradients and the stochastic approximation updating equations, it is now possible to construct a computational procedure to find the local minimum of the empirical risk.

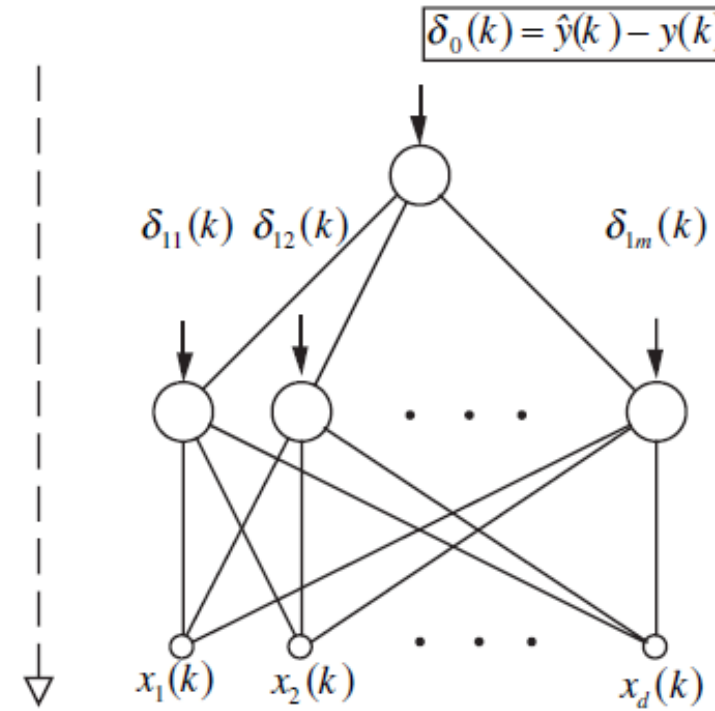
$$\frac{\partial R}{\partial v_{ij}} = (\hat{y} - y) g'(a_j) w_j x_i$$

$$\frac{\partial R}{\partial w_j} = (\hat{y} - y) z_j.$$

- Starting with an initial guess for values w(0) and V(0), the stochastic approximation procedure for parameter (weight) updating upon presentation of a sample (x(k), y(k)) at iteration step k with learning rate γ_k.

Backpropagation training:
(a) forward pass;
(b) backward pass.





$$\delta_{1j}(k) = \delta_0(k)\, g'(a_j(k))\, w_j(k+1)$$

$$w_j(k+1) = w_j(k) - \gamma_k \delta_0(k) z_j(k)$$

$$v_{ij}(k+1) = v_{ij}(k) - \gamma_k \delta_{1j}(k) x_i(k)$$

Theoretical basis for on-line learning

- Standard inductive learning: given training data $\mathbf{z}_1, \ldots, \mathbf{z}_n$ find the model providing *min* of prediction risk $R(\omega) = \int L(\mathbf{z}, \omega)\, \boldsymbol{p}(\mathbf{z}) d\mathbf{z}$

- **Stochastic Approximation** guarantees minimization of risk (asymptotically):

$$\omega(k+1) = \omega(k) - \gamma_k \, \boldsymbol{grad}_\omega \, L(\mathbf{z}_k, \omega(k))$$

under general conditions on the learning rate:

$$\boldsymbol{\lim_{k \to \infty}} \gamma_k = 0$$

$$\sum_{k=1}^{\infty} \gamma_k = \infty$$

$$\sum_{k=1}^{\infty} \gamma_k^2 < \infty$$

Practical issues for on-line learning

- Given finite training set (*n* samples): $\mathbf{z}_1, \ldots, \mathbf{z}_n$

  this set is presented sequentially to a learning algorithm many times. Each presentation of *n* samples is called an epoch, and the process of repeated presentations is called recycling (of training data)

- **Learning rate schedule**: initially set large, then slowly decreasing with *k (iteration number)*. Typically 'good' learning rate schedules are data-dependent.
- **Stopping conditions:**

  (1) monitor the gradient (i.e., stop when the gradient falls below some small threshold)

  (2) early stopping can be used for complexity control

OUTLINE

- Objectives
- Nonlinear Optimization in learning
- *Optimization basics (Appendix A)*
- Stochastic approximation aka gradient descent methods
- **Iterative methods**
- Greedy optimization
- Summary and discussion

- **Dictionary representation** with fixed *m :*

$$f(\mathbf{x}, \mathbf{w}, \mathbf{V}) = \sum_{j=0}^{m} w_i g_i(\mathbf{x}, \mathbf{v}_i)$$

Iterative strategy *for ERM (nonlinear optimization)*

- **Step 1**: for current estimates of $\hat{\mathbf{v}}_i$ update w_i
- **Step 2**: for current estimates of $\hat{w}_i$ update $\mathbf{v}_i$

Iterate Steps (1) and (2) above

Note: - this idea can be implemented for different problems:

e.g., unsupervised learning, supervised learning

Specific update rules depend on the type of problem & loss fct.

Iterative Methods

- Implement iterative parameter estimation.
- This leads to a generic parameter estimation scheme, where the two steps (expectation and maximization) are iterated until some convergence criterion is met.
- Expectation-Maximization (EM) developed/used in statistics.

EM Methods for Density Estimation

Motivating Example:

$$p_1 = P(\text{H on Coin 1})$$

$$p_2 = P(\text{H on Coin 2})$$

- Have two coins: Coin1 and Coin2

1. Select a coin at random and flip that one coin m times.
2. Repeat this process n times.

- We have

$$\begin{array}{cccc} X_{11} & X_{12} & \cdots & X_{1m} \\ X_{21} & X_{22} & \cdots & X_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ X_{n1} & X_{n2} & \cdots & X_{nm} \end{array} \qquad \begin{array}{c} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{array}$$

$$Y_i \text{ live in } \{1,2\} \qquad X_{ij} = \begin{cases} 1 & , \text{ if the jth flip for the ith coin chosen is H} \\ 0 & , \text{ if the jth flip for the ith coin chosen is T} \end{cases}$$

- Note that all the X's are independent and, in particular

$$X_{i1}, X_{i2}, \ldots, X_{im} | Y_i = j \overset{iid}{\sim} Bernoulli(p_j)$$

https://www.colorado.edu/amath/sites/default/files/attached-files/em_algorithm.pdf

EM Methods for Density Estimation

- We can write out the joint pdf of all nm+n random variables and formally come up with MLEs for p_1 and p_2.
- Call these MLEs p_1 and p_2. They will turn out as expected:

$$\widehat{p}_1 = \frac{\text{total \# of times Coin 1 came up H}}{\text{total \# times Coin 1 was flipped}}$$

$$\widehat{p}_2 = \frac{\text{total \# of times Coin 2 came up H}}{\text{total \# times Coin 2 was flipped}}$$

- Now suppose that the Y_i are not observed but we still want MLEs for p_1 and p_2. The data set now consists of only the X's and is "incomplete".
- The goal of the EM Algorithm is to find MLEs for p_1 and p_2 in this case.

https://www.colorado.edu/amath/sites/default/files/attached-files/em_algorithm.pdf

EM Methods for Density Estimation

- Let X be observed data, generated by some distribution depending on some parameters.
- Let Y be some “hidden” or “unobserved data” depending on some parameters.
- Let Z=(X,Y) represent the “complete” dataset.
- Assume we can write the pdf for Z as (depends on some parameter θ)

$$f(z|\theta) = f(x, y|\theta) = f(y|x, \theta)f(x|\theta)$$

- We have the complete likelihood function:

$$L(\theta|Z) = L(\theta|X, Y) = f(X, Y|\theta)$$

- We have the incomplete likelihood function:

$$L(\theta|X) = f(X|\theta)$$

https://www.colorado.edu/amath/sites/default/files/attached-files/em_algorithm.pdf

EM Methods for Density Estimation

- The EM Algorithm is a numerical iterative for finding an MLE of θ. The rough idea is to start with an initial guess for θ and to use this and the observed data X to "complete" the dataset by using X and the guessed θ to postulate a value for Y, at which point we can then find an MLE for θ in the usual way.

- We will use an initial guess for θ and postulate an entire distribution for Y, ultimately averaging out the unknown Y.

https://www.colorado.edu/amath/sites/default/files/attached-files/em_algorithm.pdf

EM Methods for Density Estimation

1. Let $k = 0$. Give an initial estimate for θ. Call it $\widehat{\theta}^{(k)}$.

2. Given observed data X and assuming that $\widehat{\theta}^{(k)}$ is correct for the parameter θ, find the conditional density $f(y|X, \widehat{\theta}^{(k)})$ for the completion variables.

3. Calculate the conditional expected log-likelihood or "Q-function":

$$Q(\theta|\widehat{\theta}^{(k)}) = \mathsf{E}[\ln f(X, Y|\theta)|X, \widehat{\theta}^{(k)}].$$

Here, the expectation is with respect to the conditional distribution of Y given X and $\widehat{\theta}^{(k)}$ and thus can be written as

$$Q(\theta|\widehat{\theta}^{(k)}) = \int \ln(f(X, y|\theta)) \cdot f(y|X, \widehat{\theta}^{(k)})\, dy.$$

(The integral is high-dimensional and is taken over the space where Y lives.)

4. Find the θ that maximizes $Q(\theta|\widehat{\theta}^{(k)})$. Call this $\widehat{\theta}^{(k+1)}$.

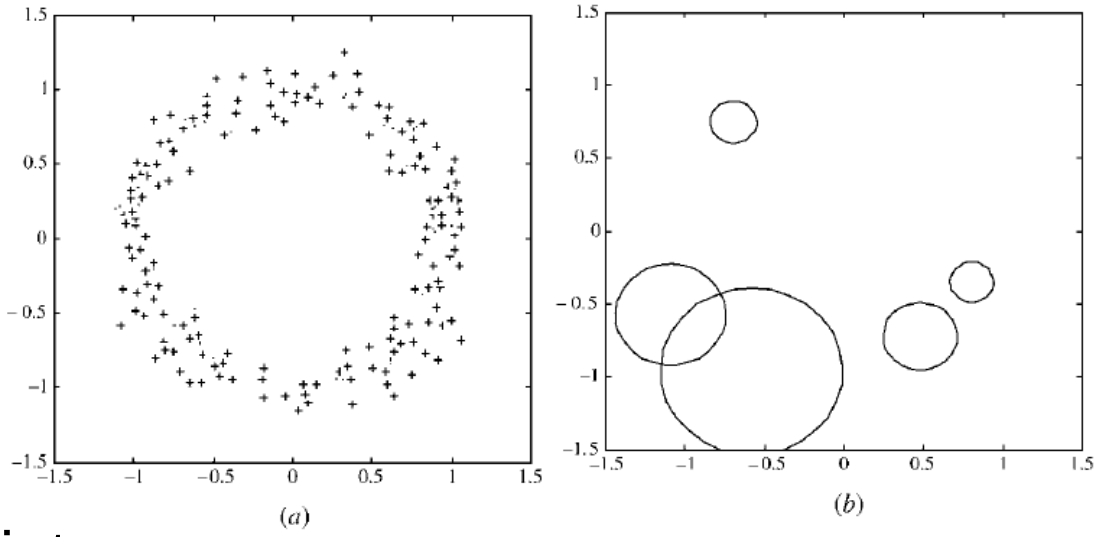
Let $k = k + 1$ and return to Step 2.

The EM Algorithm is iterated until the estimate for θ stops changing.

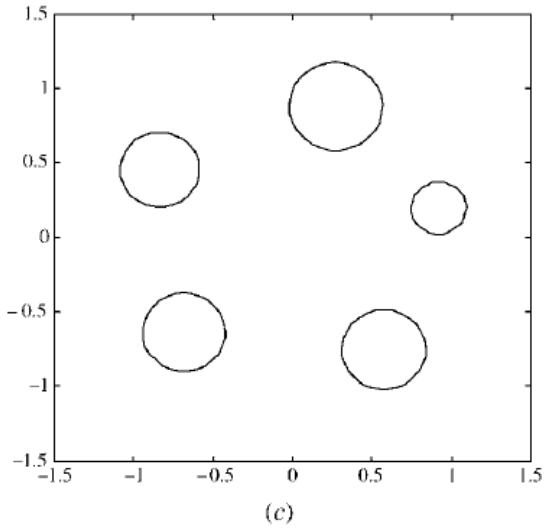
$$||\widehat{\theta}^{(k+1)} - \widehat{\theta}^{(k)}|| < \varepsilon$$

https://www.colorado.edu/amath/sites/default/files/attached-files/em_algorithm.pdf

EM Methods for Density Estimation



a) Two hundred data points drawn from a doughnut distribution.
b) Initial configuration of five Gaussian mixtures.
c) Configuration after 20 iterations of the EM algorithm



OUTLINE

- Objectives
- Nonlinear Optimization in learning
- Optimization basics (Appendix A)
- Stochastic approximation (gradient descent)
- Iterative methods
- **Greedy optimization**
- Summary and discussion

Greedy Optimization Methods

- Minimization of empirical risk (regression problems)

$$R_{emp}(\mathbf{V},\mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n} L(\mathbf{x}_i, y_i, \mathbf{V}, \mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n} (y_i - f(\mathbf{x}_i, \mathbf{V}, \mathbf{W}))^2$$

where the model $f(\mathbf{x}, \mathbf{V}, \mathbf{W}) = \sum_{j=1}^{m} w_j \, g_j(\mathbf{x}, \mathbf{v}_j)$

- Greedy Optimization Strategy

basis functions are estimated sequentially, one at a time,
i.e., the training data is represented as
structure (model fit) + noise (residual):

(1) DATA = (model) FIT 1 + RESIDUAL 1
(2) RESIDUAL 1 = FIT 2 + RESIDUAL 2
and so on. The final model for the data will be

MODEL = FIT 1 + FIT 2 + ...

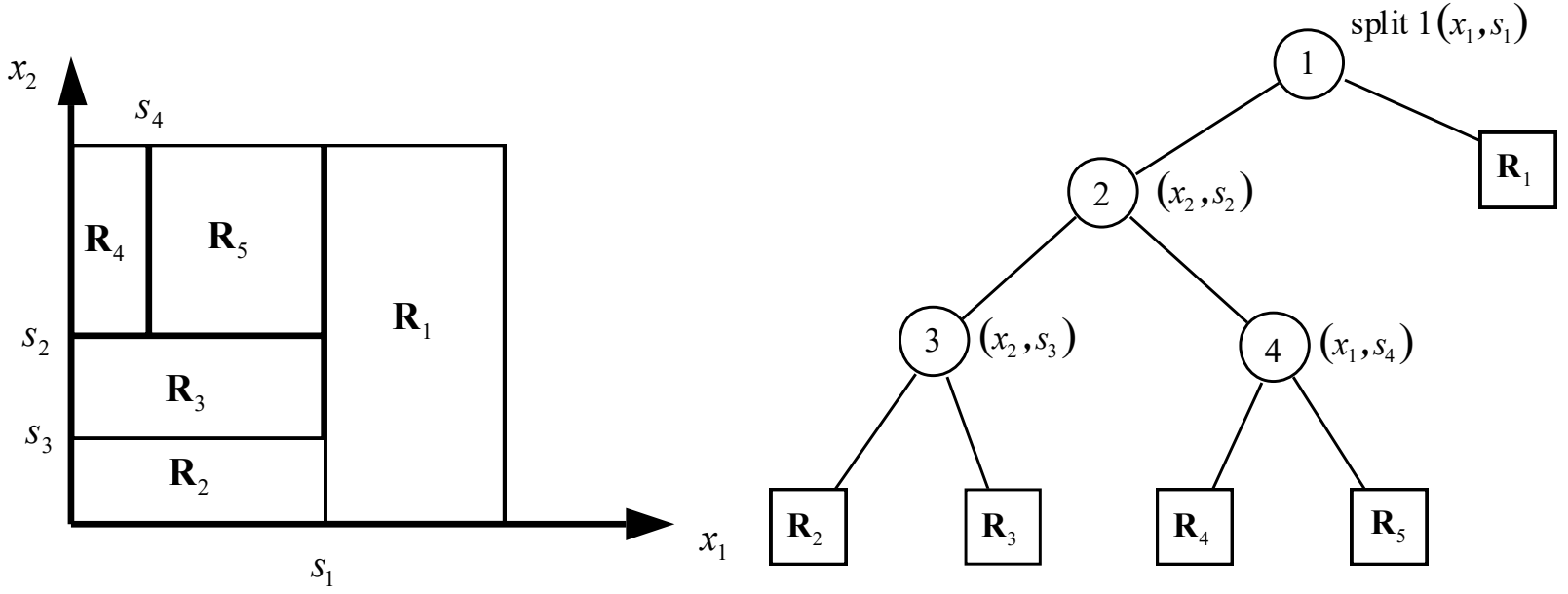
- Advantages: computational speed, interpretability

Classification and Regression Trees (CART)

- Minimization of empirical risk (squared error) via partitioning of the input space into regions

$$f(\mathbf{x}) = \sum_{j=1}^{m} w_j I\left(\mathbf{x} \in \mathbf{R}_j\right)$$

- Example of CART partitioning for a function of 2 inputs



Growing CART

- Recursive partitioning for estimating regions (via binary splitting)
- Initial Model ~ Region $\mathbf{R}_0$ (the whole input domain) is divided into two regions $\mathbf{R}_1$ and $\mathbf{R}_2$
- A split is defined by one of the inputs(*k*) and split point *s*
- Optimal values of *(k, s)* chosen so that splitting a region into two daughter regions minimizes empirical risk
- Issues:

  - efficient implementation (selection of opt. split)

  - optimal tree size ~ model selection(complexity control)

Growing CART

- Advantages:

1. Results are simplistic.
2. Classification and regression trees are Nonparametric and Nonlinear.
3. Classification and regression trees implicitly perform feature selection.
4. Outliers have no meaningful effect on CART.
5. It requires minimal supervision and produces easy-to-understand models.

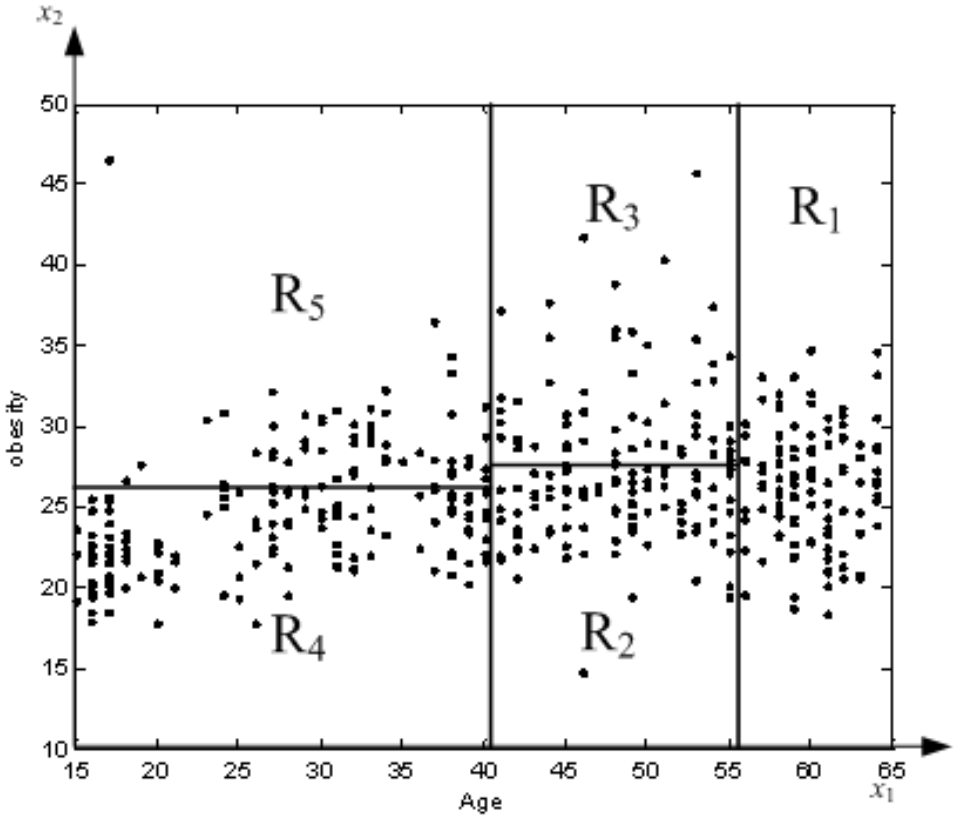
- Limitations:

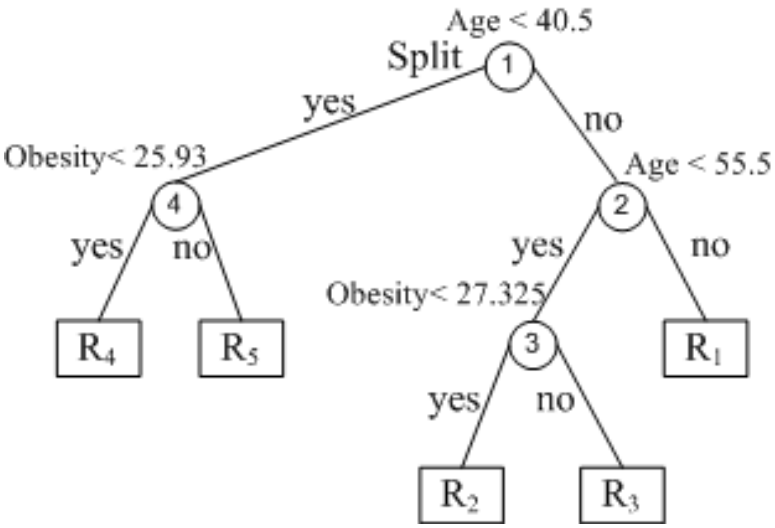
1. Overfitting.
2. High Variance.
3. low bias.
4. the tree structure may be unstable.

https://www.geeksforgeeks.org/cart-classification-and-regression-tree-in-machine-learning/

CART Example

- CART model for estimating *Systolic Blood Pressure (SBP)* as a function of *Age* and *Obesity* in a population of males in S. Africa





CART model selection

- Model selection strategy

  (1) Grow a large tree (subject to min leaf node size)

  (2) Tree pruning by selectively merging tree nodes

- The final model ~ minimizes penalized risk

$$R_{pen}(\omega, \lambda) = R_{emp}(\omega) + \lambda \|T\|$$

where empirical risk ~ MSE

number of leaf nodes ~ $\|T\|$

regularization parameter ~ λ

- *Note:* larger λ → smaller trees
- In practice: often user-defined (*splitmin* in Matlab)

Pitfalls of greedy optimization

- Greedy binary splitting strategy may yield:

  - sub-optimal solutions (regions $\mathbf{R}_i$)

  - solutions very sensitive to random samples (especially with correlated input variables)

- Counterexample for CART:

  estimate function *f*(a,b,c):

  - which variable for the first split?

  Choose a → 3 errors

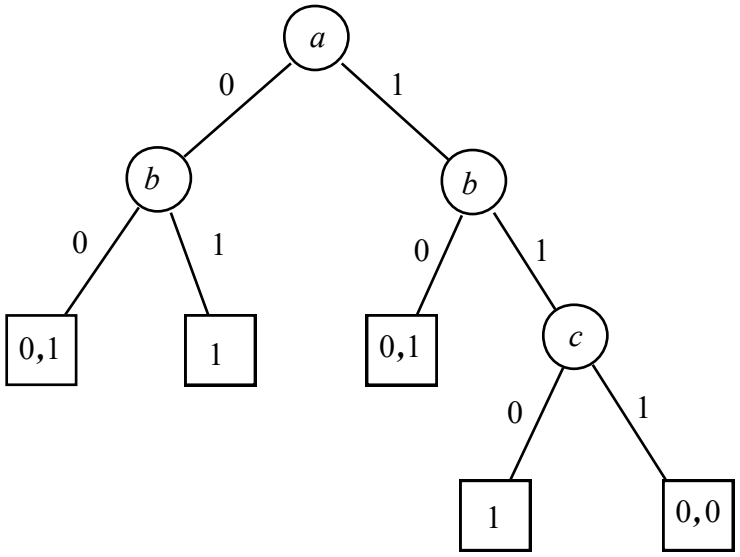
  Choose b → 4 errors

  Choose c → 4 errors

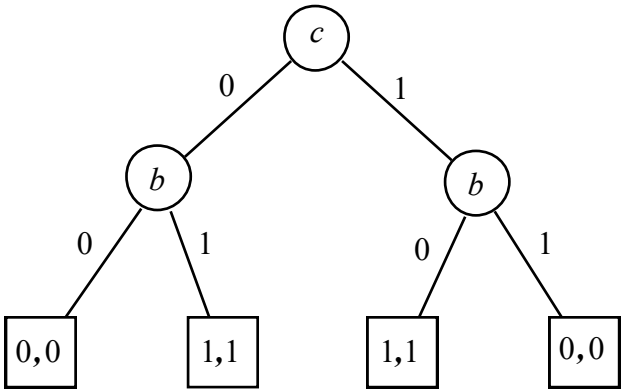
y	a	b	c
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
1	0	1	0
1	1	1	0
0	1	1	1
0	1	1	1

Counter example (cont'd)

(a) Suboptimal tree by CART



(b) Optimal binary tree

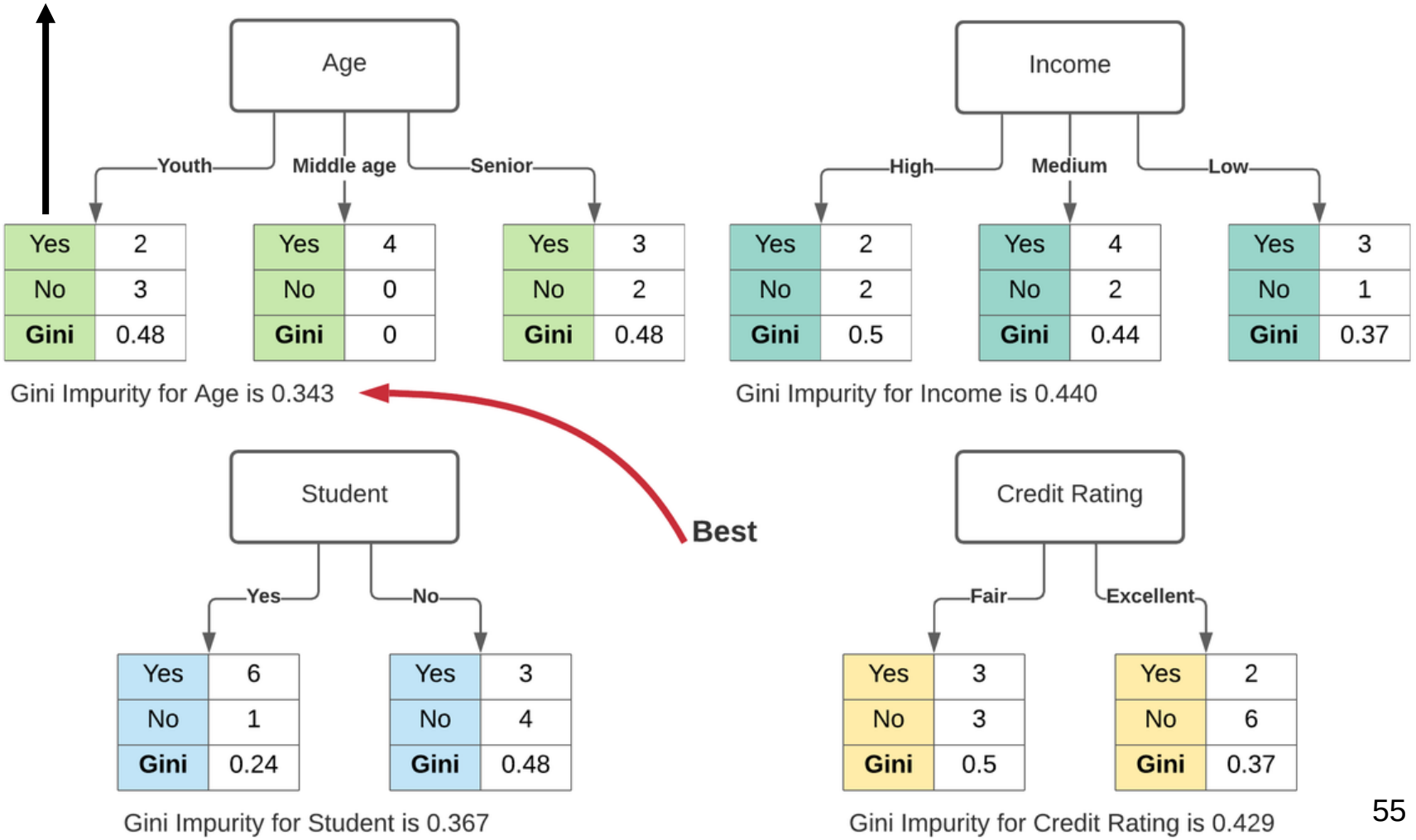


Gini score

$$Gini = 1 - \sum_{i=1}^{n} (pi)^2$$

pi is the probability of an object being classified to a particular class.

1-(2/5)^2-(3/5)^2= 1-4/25-9/25=12/25=0.48



https://www.learndatasci.com/glossary/gini-impurity/

Backfitting Algorithm

- Consider regression estimation of a function of two variables of the form $y = g_1(x_1) + g_2(x_2) + noise$

  from training data $(x_{1i}, x_{2i}, y_i) \quad i = 1,2,..., n$

  For example $t(x_1, x_2) = x_1^2 + \sin(2\pi x_2) \quad \mathbf{x} \in (0,1)^2$

Backfitting method: (1) estimate $g_1(x_1)$ for fixed g_2

(2) estimate $g_2(x_2)$ for fixed g_1

iterate above two steps

- Estimation via minimization of empirical risk

$$R_{emp}(g_1(x_1), g_2(x_2)) = \frac{1}{n}\sum_{i=1}^{n}(y_i - g_1(x_{1i}) - g_2(x_{2i}))^2$$

$$(first\_iteration) = \frac{1}{n}\sum_{i=1}^{n}([y_i - g_2(x_{2i})] - g_1(x_{1i}))^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}(r_i - g_1(x_{1i}))^2$$

Backfitting Algorithm(cont'd)

- Estimation of $g_1(x_1)$ via minimization of MSE

$$R_{emp}\left(g_1(x_1)\right) = \frac{1}{n}\sum_{i=1}^{n}\left(r_i - g_1(x_{1i})\right)^2 \rightarrow min$$

- This is a univariate regression problem of estimating $g_1(x_1)$ from *n* data points (x_{1i}, r_i) where $r_i = y_i - g_2(x_{2i})$
- Can be estimated by smoothing (kNN regression)
- Estimation of $g_2(x_2)$ (second iteration) proceeds in a similar manner, via minimization of

$$R_{emp}\left(g_2(x_2)\right) = \frac{1}{n}\sum_{i=1}^{n}\left(r_i - g_2(x_{2i})\right)^2 \quad \text{where } r_i = y_i - g_1(x_{1i})$$

- **Backfitting ~gradient descent in the function space**

OUTLINE

- Objectives
- Nonlinear Optimization in learning
- Optimization basics (Appendix A)
- Stochastic approximation aka gradient descent methods
- Iterative methods
- Greedy optimization
- **Summary and discussion**

Summary

- Different interpretation of optimization
- Consider dictionary parameterization

$$f_m(\mathbf{x}, \mathbf{w}, \mathbf{V}) = \sum_{i=0}^{m} w_i \, g(\mathbf{x}, \mathbf{v}_i)$$

- VC-theory: implementation of SRM
- Gradient descent + iterative optimization

  - SRM structure is specified a priori
  - selection of *m* is separate from ERM

- Greedy optimization strategy

  - no a priori specification of a structure
  - model selection is a part of optimization

Summary (cont'd)

- Interpretation of greedy optimization

$$f_m(\mathbf{x}, \mathbf{w}, \mathbf{V}) = \sum_{i=0}^{m} w_i \, g(\mathbf{x}, \mathbf{v}_i)$$

- Statistical strategy for iterative data fitting

  (1) Data = (model) Fit1 + Residual_1

  (2) Residual_1 = (model) Fit2 + Residual_2

  ……..

→ Model = Fit1 + Fit2 + …

- This has superficial similarity to SRM