Introduction to Data Mining

Chapter 9. Classification: Advanced Methods

Jiawei Han, Computer Science, Univ. Illinois at Urbana-Champaign, 2017

1

Chapter 9. Classification: Advanced Methods

- Bayesian Networks
- Support Vector Machines
- Neural Networks and Deep Learning
- Lazy Learners and K-Nearest Neighbors

2

Generative vs. Discriminative Classifiers

- X: observed variables (features)
- Y: target variables (class labels)
- A generative classifier models p(Y, X) models how the data was "generated“. "what is the likelihood this or that class generated this instance?" and pick the one with higher probability
  - Naïve Bayes
  - Bayesian Networks
- A discriminative classifier models p(Y|X) uses the data to create a decision boundary
  - Logistic Regression
  - Support Vector Machines

3

Generative Classifier

- **Definition**: Models the joint probability P(X,Y)P(X, Y)P(X,Y), where X is the input (features) and Y is the output (label).
- **How It Works**:
  - Learns the distribution P(X|Y)P(X|Y)P(X|Y) (how the features are distributed given the class).
  - Learns the prior P(Y)P(Y)P(Y) (the probability of each class).
  - Combines these using Bayes' theorem to calculate P(Y|X)P(Y|X)P(Y|X) (posterior probability) for classification.
- **Example**: Naïve Bayes classifier, Bayesian Networks, Gaussian Mixture Models
- **Advantages**:
  - Can be used to generate new data by sampling from P(X,Y)P(X, Y)P(X,Y).
  - Handles missing data well by using P(X|Y)P(X|Y)P(X|Y).
  - Provides interpretable probabilistic outputs.
- **Disadvantages**:
  - Assumes a specific distribution for P(X|Y)P(X|Y)P(X|Y), which may be incorrect.
  - Typically requires more computational resources.

4

4

Discriminative Classifier

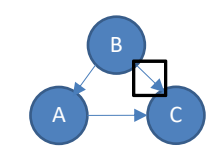
- **Definition**: Models the conditional probability P(Y|X)P(Y|X)P(Y|X) directly or learns a decision boundary between classes.
- **How It Works**:
  - Focuses only on the relationship between inputs XXX and outputs YYY.
  - Does not explicitly model P(X|Y)P(X|Y)P(X|Y) or P(X)P(X)P(X).
- **Example**: Logistic regression, support vector machines (SVM), neural networks.
- **Advantages**:
  - Often achieves higher accuracy on classification tasks, as it focuses directly on the decision boundary.
  - Requires fewer assumptions about the underlying data distribution.
- **Disadvantages**:
  - Cannot be used to generate new data.
  - Does not handle missing data as effectively as generative models.

5

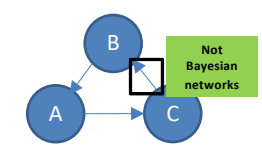
5

From Naïve Bayes to Bayesian Networks

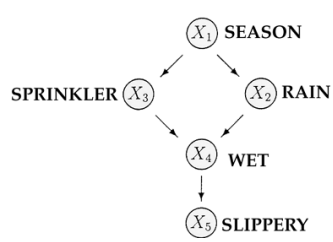
- Naïve Bayes classifiers assume *that the value of a particular feature is independent of the value of any other feature*, given the class variable
  - This assumption is often too simple to model the real world well
- Bayesian network (or Bayes network, belief network, Bayesian model or probabilistic directed acyclic graphical model) is a probabilistic graphical model.
- Represented by a set of random variables and their conditional dependencies via a directed acyclic graph (DAG)



directed **acyclic** graphical model



directed **cyclic** graphical model



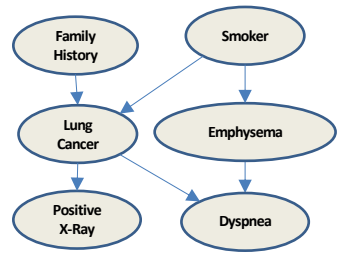
6

Bayesian network (Bayesian belief network or probabilistic network)

- Allows *class conditional independencies* between *subsets* of variables.
- Ex. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases

Nodes: random variables Links: dependency

- Two components:
  - A *directed acyclic graph* (called a structure)
  - A set of *conditional probability tables* (CPTs)



- *Directed Acyclic Graph* (DAG):
  - Represents dependency among the variables (*causal influence* relationship)
  - Gives a specification of joint probability distribution

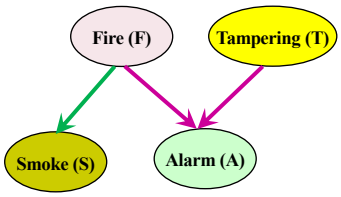
$$p(A, B, C) = p(B) \cdot p(A|B) \cdot p(C|A, B)$$

	FH, S	*FH, ~S*	*~FH, S*	*~FH, ~S*
LC	0.8	0.5	0.7	0.1
~LC	0.2	0.5	0.3	0.9

7

A Bayesian Network and Its CPTs

E.g. If there is a fire alarm, it may have been caused by a fire or by tampering. If there is a fire, there may be smoke.



Conditional **P**robability **T**ables (CPT)

Fire	Smoke	$\Theta_{s\|f}$
True	True	.90
False	True	.01

Fire	Tampering	Alarm	$\Theta_{a\|f,t}$
True	True	True	.5
True	False	True	.99
False	True	True	.85
False	False	True	.0001

CPT shows the conditional probability for each possible combination of its parents:

$$p(X) = \prod_k p(x_k|\text{Parents}(x_k))$$

$$p(F, S, A, T) = p(F) \cdot p(T) \cdot p(S|F) \cdot p(A|F, T)$$

Interactive example:
https://www.bayesserver.com/docs/introduction/bayesian-networks
https://www.bayesserver.com/examples/networks/asia

8

How Are Bayesian Networks Constructed?

- **Subjective construction**: Identification of (direct) causal structure
  - People are quite good at identifying direct causes from a given set of variables & whether the set contains all relevant direct causes
  - Markovian assumption: Each variable becomes independent of its non-effects once its direct causes are known
    - E.g., S ← F → A ← T, path S → A is blocked once we know F → A
  - HMM (Hidden Markov Model): often used to model dynamic systems whose states are not observable, yet their outputs are

- **Synthesis from other specifications**
  - E.g., from a formal system design: block diagrams & info flow
- **Learning from data** (e.g., from medical records or student admission record)
  - Learn parameters give its structure or learn both structure and parms
  - Maximum likelihood principle: favors Bayesian networks that maximize the probability of observing the given data set

9

Training Bayesian Networks: Several Scenarios

- Scenario 1: Given both the network structure and all variables observable: *compute only the CPT entries*
- Scenario 2: Network structure known, some variables hidden: *gradient descent* (greedy hill-climbing) method, i.e., search for a solution along the steepest descent of a criterion function
  - Weights are initialized to random probability values
  - At each iteration, it moves towards what appears to be the best solution at the moment, without backtracking
  - Weights are updated at each iteration & converge to local optimum
- Scenario 3: Network structure unknown, all variables observable: search through the model space to *reconstruct network topology*
- Scenario 4: Unknown structure, all hidden variables: No good algorithms known for this purpose
- D. Heckerman. A Tutorial on Learning with Bayesian Networks. In *Learning in Graphical Models,* M. Jordan, ed. MIT Press, 1999

10

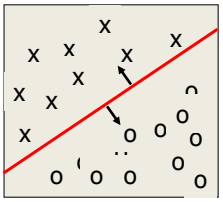
Chapter 9. Classification: Advanced Methods

- Bayesian Networks
- Support Vector Machines
- Neural Networks and Deep Learning
- Lazy Learners and K-Nearest Neighbors

11

Classification: A Mathematical Mapping

- **Classification:** Predicts categorical class labels
  - E.g., Personal homepage classification
    - $x_i = (x_1, x_2, x_3, \ldots)$, $y_i = +1$ or -1
    - x_1 : # of word "homepage"
    - x_2 : # of word "welcome"
- Mathematically, $x \in X = \Re^n$, $y \in Y = \{+1, -1\}$,
  - We want to derive a function $f: X \rightarrow Y$
- Linear Classification
  - Binary Classification problem
  - Data above the red line belongs to class 'x'
  - Data below red line belongs to class 'o'
  - Examples: SVM, Perceptron, Probabilistic Classifiers



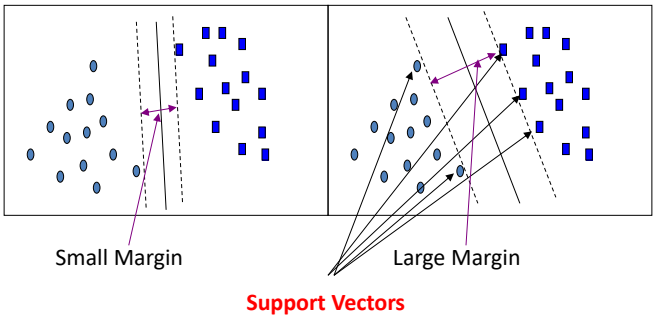
12

SVM—Support Vector Machines

- A classification method for both linear and nonlinear data
  - Vapnik and colleagues (1992)—groundwork from Vapnik & Chervonenkis' statistical learning theory in 1960s
- It uses a nonlinear mapping to transform the original training data into a higher dimension
- With the new dimension, it searches for the linear optimal separating **hyperplane** (i.e., "decision boundary")
- With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane
- SVM finds this hyperplane using **support vectors** ("essential" training tuples) and **margins** (defined by the support vectors)

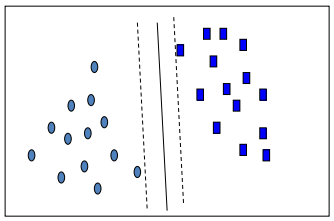
13

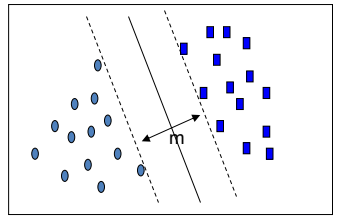
SVM—General Philosophy



14

SVM—When Data Is Linearly Separable





Let data D be ($\mathbf{X}_1$, y_1), …, ($\mathbf{X}_{|D|}$, $y_{|D|}$), where $\mathbf{X}_i$ is the set of training tuples associated with the class labels y_i

There are infinite lines (hyperplanes) separating the two classes but we want to find the best one (the one that minimizes classification error on unseen data)

SVM searches for the hyperplane with the largest margin, i.e., **maximum marginal hyperplane** (MMH)

15

SVM—Linearly Separable

- A separating hyperplane can be written as

  $\mathbf{W} \bullet \mathbf{X} + b = 0$

  where $\mathbf{W}=\{w_1, w_2, \ldots, w_n\}$ is a weight vector and b a scalar (bias)
- For 2-D, it can be written as: $w_0 + w_1 x_1 + w_2 x_2 = 0$
- The hyperplane defining the sides of the margin:

  H_1: $w_0 + w_1 x_1 + w_2 x_2 \geq 1$ for $y_i = +1$, and

  H_2: $w_0 + w_1 x_1 + w_2 x_2 \leq -1$ for $y_i = -1$
- Any training tuples that fall on hyperplanes H_1 or H_2 (i.e., the sides defining the margin) are **support vectors**
- This becomes a **constrained (convex) quadratic optimization** problem:
  - Quadratic objective function and linear constraints → *Quadratic Programming (QP)* → Lagrangian multipliers

16

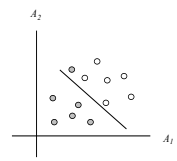
SVM—Linearly Inseparable

- Transform the original input data into a higher dimensional space

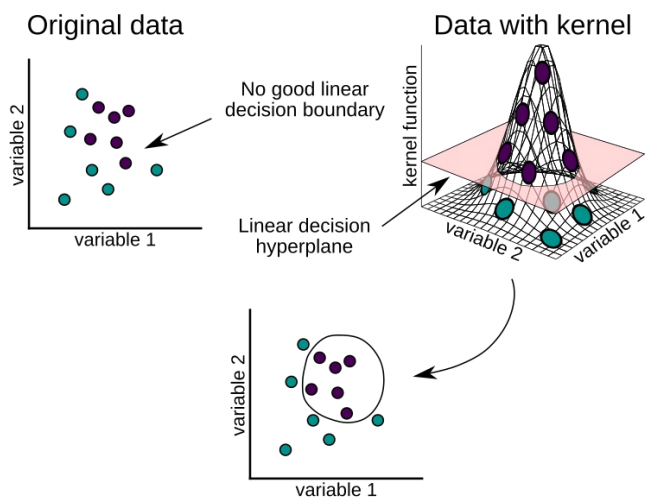
Example 6.8 Nonlinear transformation of original input data into a higher dimensional space. Consider the following example. A 3D input vector $\mathbf{X} = (x_1, x_2, x_3)$ is mapped into a 6D space Z using the mappings $\phi_1(X) = x_1, \phi_2(X) = x_2, \phi_3(X) = x_3$, $\phi_4(X) = (x_1)^2, \phi_5(X) = x_1x_2$, and $\phi_6(X) = x_1x_3$. A decision hyperplane in the new space is $d(\mathbf{Z}) = \mathbf{WZ} + b$, where $\mathbf{W}$ and $\mathbf{Z}$ are vectors. This is linear. We solve for $\mathbf{W}$ and b and then substitute back so that we see that the linear decision hyperplane in the new ($\mathbf{Z}$) space corresponds to a nonlinear second order polynomial in the original 3-D input space,

$$\begin{aligned} d(Z) &= w_1x_1 + w_2x_2 + w_3x_3 + w_4(x_1)^2 + w_5x_1x_2 + w_6x_1x_3 + b \\ &= w_1z_1 + w_2z_2 + w_3z_3 + w_4z_4 + w_5z_5 + w_6z_6 + b \end{aligned}$$

- Search for a linear separating hyperplane in the new space



17



https://www.r-bloggers.com/2019/10/support-vector-machines-with-the-mlr-package/

18

18

Kernel Functions for Nonlinear Classification

- Instead of computing the dot product on the transformed data, it is mathematically equivalent to applying a kernel function K(**X**$_i$, **X**$_j$) to the original data, i.e.,
  - K(**X**$_i$, **X**$_j$) = Φ(**X**$_i$) Φ(**X**$_j$)
- Typical Kernel Functions

$$\textbf{Polynomial kernel of degree } h: \quad K(\boldsymbol{X}_i, \boldsymbol{X}_j) = (\boldsymbol{X}_i \cdot \boldsymbol{X}_j + 1)^h$$

$$\textbf{Gaussian radial basis function kernel}: \quad K(\boldsymbol{X}_i, \boldsymbol{X}_j) = e^{-\|\mathbf{X}_i - \mathbf{X}_j\|^2/2\sigma^2}$$

$$\textbf{Sigmoid kernel}: \quad K(\boldsymbol{X}_i, \boldsymbol{X}_j) = \tanh(\kappa \boldsymbol{X}_i \cdot \boldsymbol{X}_j - \delta)$$

- SVMs can efficiently perform a non-linear classification using kernel functions, implicitly mapping their inputs into high-dimensional feature spaces

19

SVM: Applications

- Features: training can be slow but accuracy is high owing to their ability to model complex nonlinear decision boundaries (margin maximization)
- Used for: classification and numeric prediction
  - SVM can also be used for classifying multiple (> 2) classes and for regression analysis (with additional parameters)
- Applications:
  - handwritten digit recognition, object recognition, speaker identification, benchmarking time-series prediction tests

20

SVM Related Links

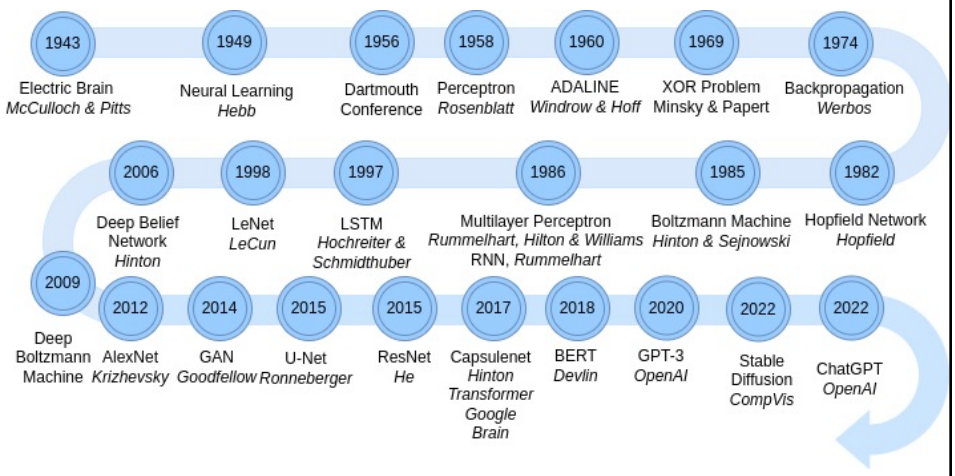
- SVM Website: http://www.kernel-machines.org/
- Representative implementations
  - **LIBSVM**: an efficient implementation of SVM, multi-class classifications, nu-SVM, one-class SVM, including also various interfaces with java, python, etc.
  - **SVM-light**: simpler but performance is not better than LIBSVM, support only binary classification and only in C
  - **SVM-torch**: another recent implementation also written in C

21

Chapter 9. Classification: Advanced Methods

- Bayesian Networks
- Support Vector Machines
- Neural Networks and Deep Learning
- Lazy Learners and K-Nearest Neighbors

22



https://pub.towardsai.net/a-brief-history-of-neural-nets-472107bc2c9c

23

23

Inspired by a biological neuron

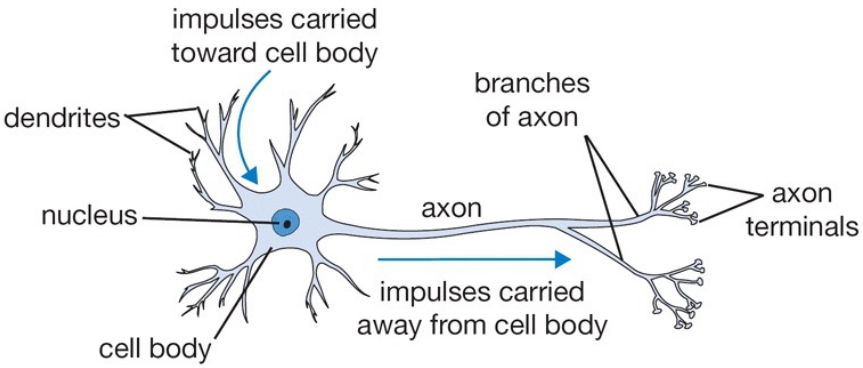
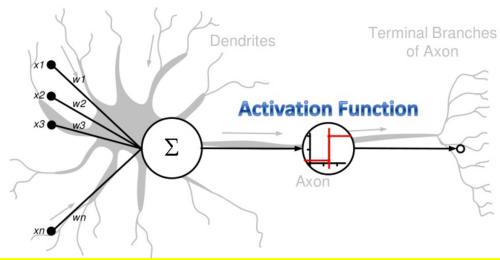


Image credit:
http://cs231n.github.io/neural-networks-1/

24

Neural Network for Classification

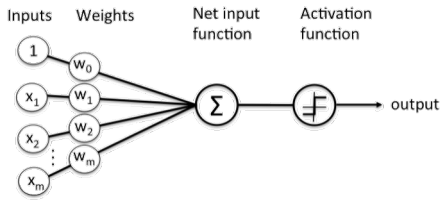
- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- A neural network: A set of connected input/output units where each connection has a **weight** associated with it
  - During the learning phase, the **network learns by adjusting the weights** so as to be able to predict the correct class label of the input tuples



Artificial Neural Networks as an analogy of Biological Neural Networks

25

Perceptron: Predecessor of a Neural Network



$$\hat{y} = f(\boldsymbol{W}^T\boldsymbol{x}) = f(\sum W_i x_i + b)$$

Activation Function
Adding non-linearity to the model

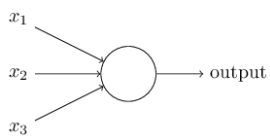
Weights

Bias
A measure of how easy it is to get the perceptron to output a 1

- Computes a weighted sum of inputs
- Invented in 1957 by Frank Rosenblatt. The original perceptron model does not have a non-linear activation function

26

Perceptron: Predecessor of a Neural Network



$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

- The perceptron algorithm: invented in 1957 by Frank Rosenblatt
- Input: An *n*-dimensional input vector **x** (with n variables)
- Output: 1 or 0 depending on if the weighted sum passes a threshold
- Perceptron: A device that makes decisions by weighing up evidence
- Often written in the vector form, using bias (*b*) instead of threshold, as

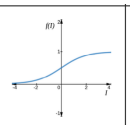
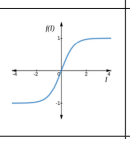
$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

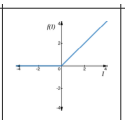
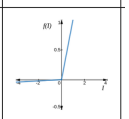
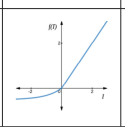
Bias: a measure of how easy it is to get the perceptron to output a 1

27

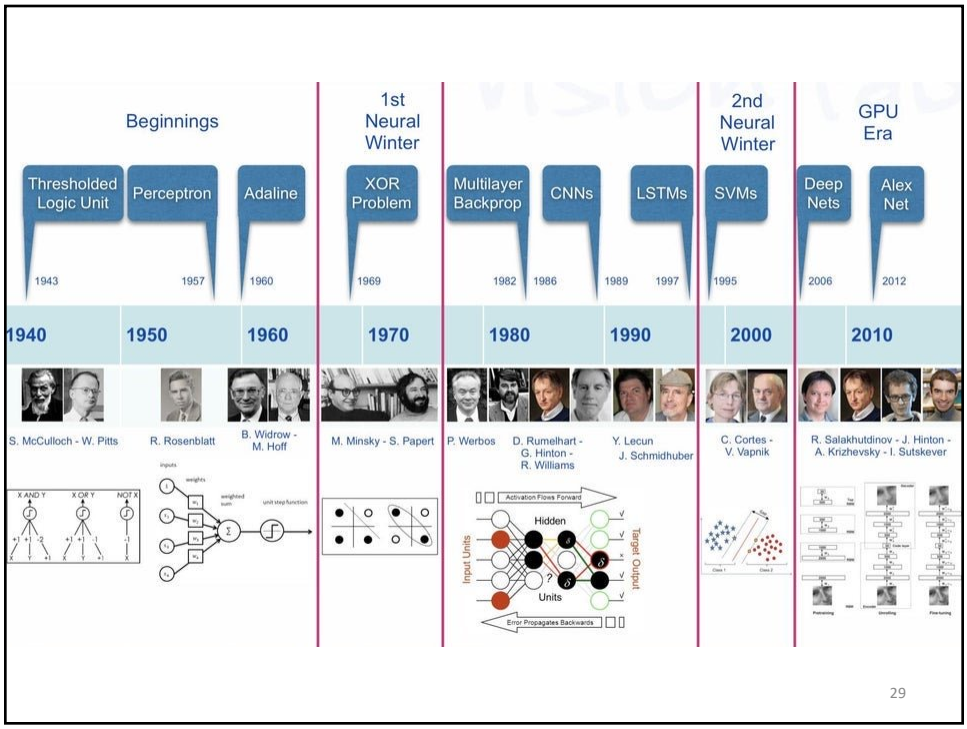
Perceptron: Predecessor of a Neural Network

- Examples of activation functions

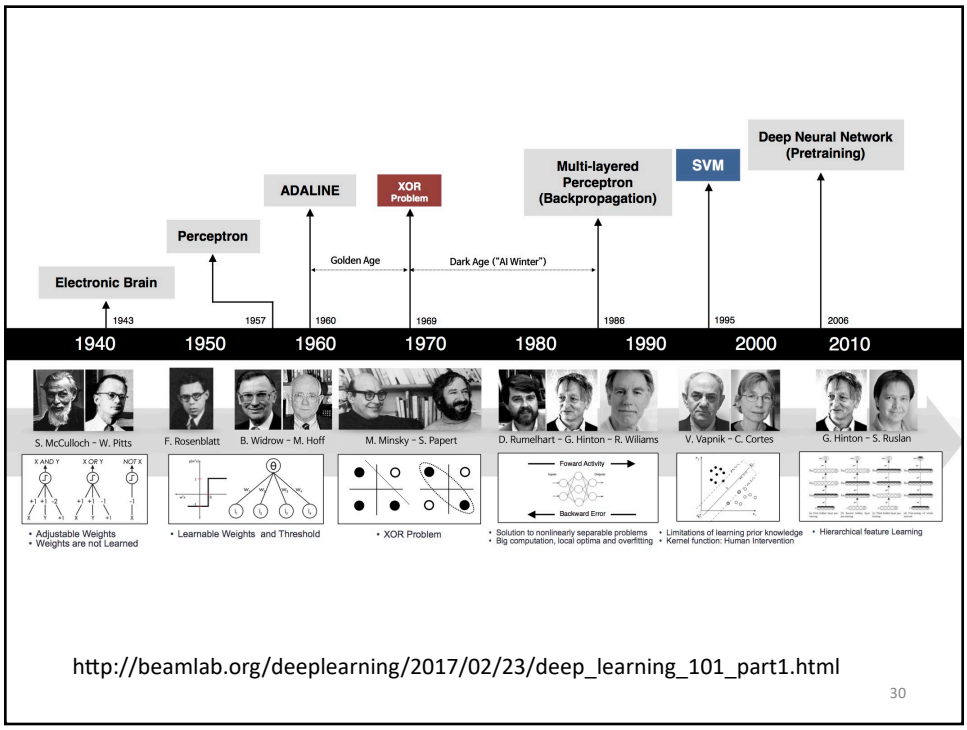
Sigmoid	$\frac{1}{1+e^{-I}}$	
Tanh	$\frac{e^I - e^{-I}}{e^I + e^{-I}}$	

ReLU	$\begin{cases} 0, I \leq 0 \\ I, I > 0 \end{cases}$	
Leaky ReLU	$\begin{cases} 0.01 \times I, I < 0 \\ I, I \geq 0 \end{cases}$	
ELU	$\begin{cases} \alpha(e^I - 1), I \leq 0 \\ I, I > 0 \end{cases}$	

28

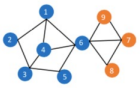
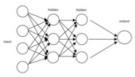
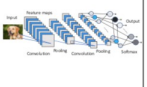
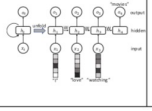
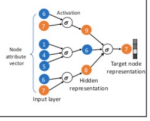

Beginnings
1st Neural Winter
2nd Neural Winter
GPU Era
Thresholded Logic Unit
Perceptron
Adaline
XOR Problem
Multilayer Backprop
CNNs
LSTMs
SVMs
Deep Nets
Alex Net
1943
1957
1960
1969
1982
1986
1989
1997
1995
2006
2012
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch - W. Pitts
R. Rosenblatt
B. Widrow - M. Hoff
M. Minsky - S. Papert
P. Werbos
D. Rumelhart - G. Hinton - R. Williams
Y. Lecun J. Schmidhuber
C. Cortes - V. Vapnik
R. Salakhutdinov - J. Hinton - A. Krizhevsky - I. Sutskever
Activation Flows Forward
Hidden
Units
Input Units
Target Output
Error Propagates Backwards
29

29


Deep Neural Network (Pretraining)
SVM
Multi-layered Perceptron (Backpropagation)
ADALINE
XOR Problem
Perceptron
Electronic Brain
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch – W. Pitts
F. Rosenblatt
B. Widrow – M. Hoff
M. Minsky – S. Papert
D. Rumelhart – G. Hinton – R. Wiliams
V. Vapnik – C. Cortes
G. Hinton – S. Ruslan
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning
Foward Activity
Backward Error
http://beamlab.org/deeplearning/2017/02/23/deep_learning_101_part1.html
30

30

Overview of Typical Deep Learning Architectures

Data type	Multi-dimensional	Grid	Sequence	Graph
	Features: credit rating, account balance $x = (4.5, 500, 3, 5)$ #deposits, #withdraws		x = "I love watching movies."	
DL Archi-tecture	Feed-forward Network	CNN	RNN	GNN
				

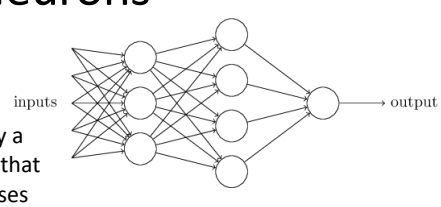
31

Sigmoid Neurons

- A many-layer network of perceptrons can engage in sophisticated decision making
- Instead of assigning weights of the edges by a person, we can devise *learning algorithms* that can automatically tune the weights and biases of a network of artificial neurons
- Use sigmoid neuron instead of perceptron: Output is not 0/1 but a simoid function: σ(*w*●*x* + b) , i.e.,
- The smoothness of σ means that small changes in the Δw_j weights and in the Δb bias will produce a small change Δ_{output} in the output from the neuron

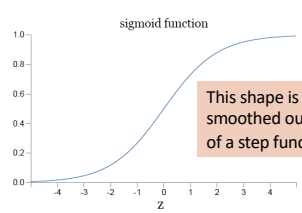
$$\Delta \text{output} \approx \sum_j \frac{\partial \,\text{output}}{\partial w_j} \Delta w_j + \frac{\partial \,\text{output}}{\partial b} \Delta b$$

i.e., Δ_{output} is a *linear function* of the changes Δw_j and Δb



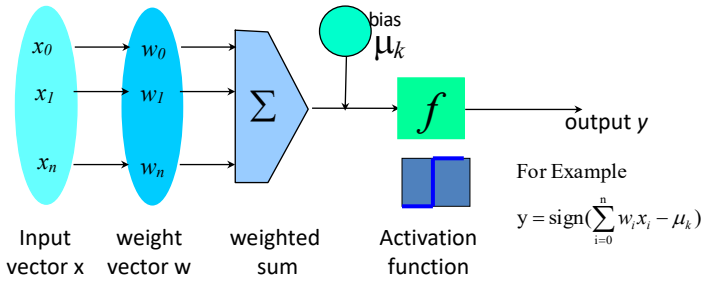
Sigmoid function: $\sigma(z) \equiv \frac{1}{1+e^{-z}}$

$$\frac{1}{1+exp(-\sum_j w_j x_j - b)}$$



This shape is a smoothed out version of a step function

32



For Example

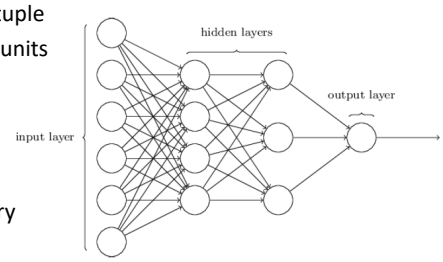
$$y = \text{sign}(\sum_{i=0}^{n} w_i x_i - \mu_k)$$

- An *n*-dimensional input vector **x** is mapped into variable y by means of the scalar product and a nonlinear function mapping
- The inputs to unit are outputs from the previous layer. They are multiplied by their corresponding weights to form a weighted sum, which is added to the bias associated with unit. Then a nonlinear activation function is applied to it.

33

(Feed-Forward) Neural Network (NN)

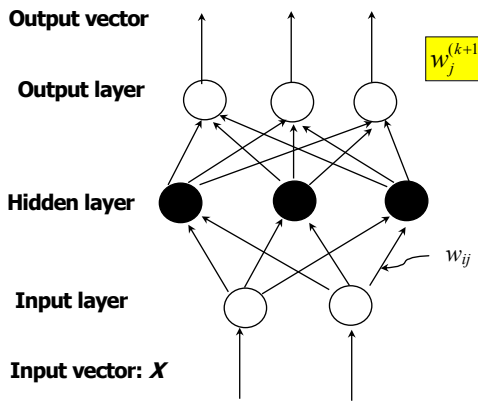
- **Input layer**
  - The **inputs** to NN correspond to the attributes measured for each training tuple
  - Inputs are fed simultaneously into the units making up the **input layer**
- **Hidden layer(s)**
  - Inputs are weighted and fed simultaneously to a hidden layer
  - The number of hidden layers is arbitrary



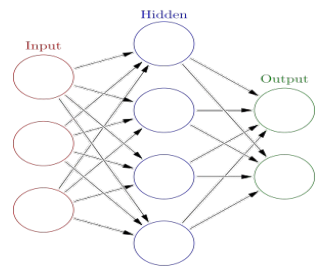
- **Output layer**
  - The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction

34

A Multi-Layer Feed-Forward Neural Network



$$w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \hat{y}_i^{(k)})x_{ij}$$



35

Feed-Forward vs. Recurrent

- **Feed-Forward Neural Network:** Typical neural network architecture
  - The output from one layer is used as input to the next layer (no loops)
  - Information is always fed forward, never fed back
  - From a statistical point of view, networks perform **nonlinear regression**
  - Given enough hidden units and enough training samples, they can closely approximate any function

- **Recurrent neural network**: Feedback loops are possible (cascade of neurons firing)
  - Some neurons fire for some limited duration of time, before becoming quiescent
  - That firing can stimulate other neurons, which may fire a little while later, also for a limited duration, which causes still more neurons to fire, and so on
  - Loops do not cause problems since a neuron's output only affects its input at some later time, not instantaneously

36

Learning with Gradient Descent

- A quadratic cost (objective) function C (or mean square error, MSE)

$$C(w,b) \equiv \frac{1}{2n}\sum_x \|y(x)-a\|^2$$

where w: the collection of all weights in the network, b: all the biases, n: the total # of training inputs, a: the vector of outputs from the network when *x* is input

- **Goal of training a network**: Find weights and biases which minimize the cost *C*(*w*, *b*)

- That is, choose Δv_1 and Δv_2 to make ΔC negative; i.e., the ball is rolling down into the valley:

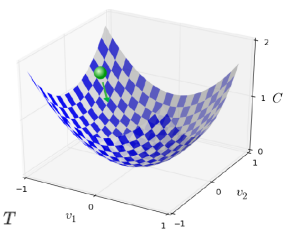
$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2$$

- The change ΔC *in C by a small change in v,* Δv*:*

$$\Delta C \approx \nabla C \cdot \Delta v$$

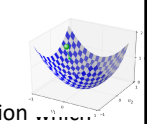
where ∇C is the gradient vector:

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_m}\right)^T$$



37

Stochastic Gradient Descent



- Gradient descent can be viewed as a way of taking small steps in the direction which does the most to immediately decrease *C*
- To compute gradient ∇C, we need to compute the gradients ∇C_x separately for each training input, x, and then average them: slow when the # of training inputs is large
- *Stochastic gradient descent* (SGD): Speed up learning
  - Computing for a small sample of randomly chosen training inputs and *averaging over them*, we can quickly get a good estimate of the true gradient
  - Method: Randomly pick out a small number (***mini-batch***) *m* of randomly chosen training inputs. Provided the sample size is large enough, we expect that the average value will be roughly equal to the average over all, that is,

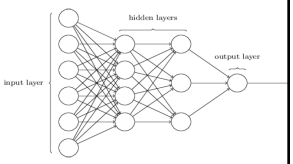
$$\nabla C \approx \frac{1}{m}\sum_{j=1}^{m}\nabla C_{X_j}$$

- Stochastic gradient descent in neural networks:
  - Pick out a randomly chosen minibatch of training inputs and train with them; then pick out another minibatch, until inputs exhausted—complete an *epoch* of training
  - Then we start over with a new training epoch

38

Backpropagation for Fast Gradient Computation

- **Backpropagation**: Reset weights on the "front" neural units and this is sometimes done in combination with training where the correct result is known
- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value. For each training tuple, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value
- Modifications are made in the "**backwards**" direction
  - From the output layer, through each hidden layer back to the first hidden layer, hence "**backpropagation**"



- Steps
  - Initialize weights to small random numbers, associated with biases
  - Propagate the inputs forward (by applying activation function)
  - Backpropagate the error (by updating weights and biases)
    - Terminating condition (when error is very small, etc.)

39

More on Backpropagation

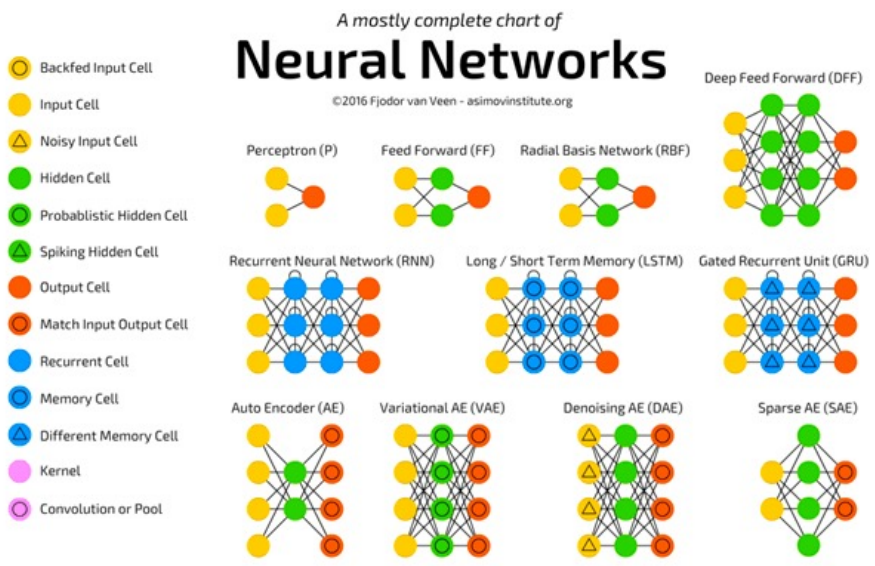
- With backpropagation, we distribute the "blame" backward through the network
  - Each hidden node sending input to the current node is somewhat "responsible" for some portion of the error in each neuron to which it has forward connection
- Local minima and backpropagation
  - Backpropagation can be stuck at local minima
  - But in practice it generally performs well
- Is backpropagation too slow?
  - Historically, backpropagation has been considered slow
  - Recent advances in computer power through parallelism and GPUs (graphics processing units) have reduced time substantially for training neural networks

40

From Neural Networks to Deep Learning

- Train networks with many layers (vs. shallow nets with just a couple of layers)
  - More neurons than previous networks
  - More complex ways to connect layers
  - Tremendous computing power to train networks
  - Automatic feature extraction
- Multiple layers work to build an improved feature space
  - Analogy: Signals passing through regions of the visual cortex
    - Example: For face recognition: edge → nose → face, layer-by-layer
- Popular Deep Learning Frameworks for Classification
  - Deep Feedforward Neural Networks
  - Convolutional Neural Networks
  - Recurrent Neural Networks

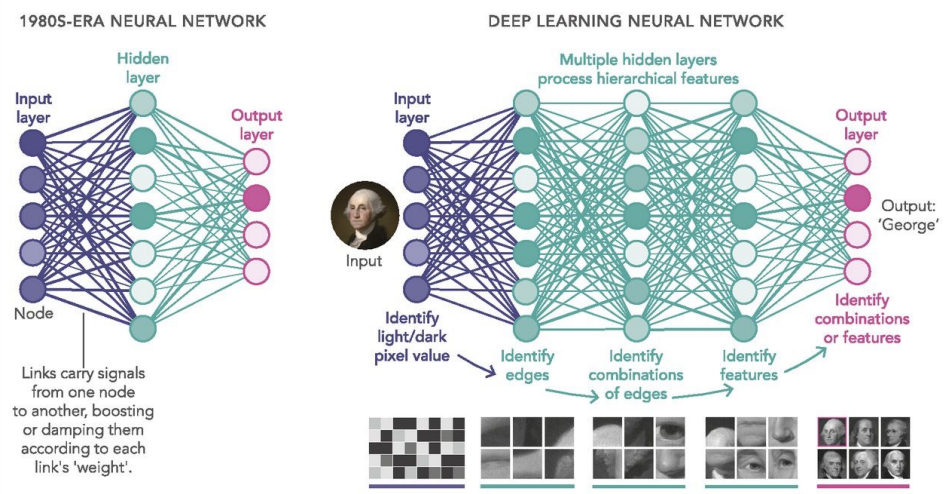
41



https://www.guru99.com/deep-learning-tutorial.html

42

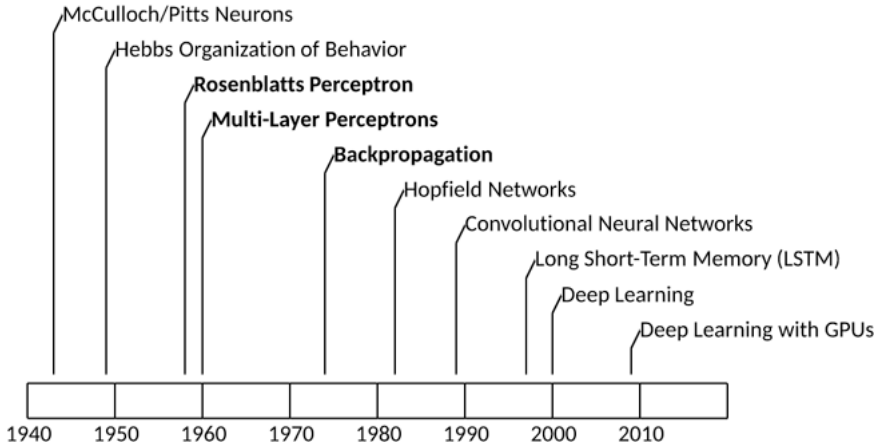
42



https://www.pnas.org/content/116/4/1074

43

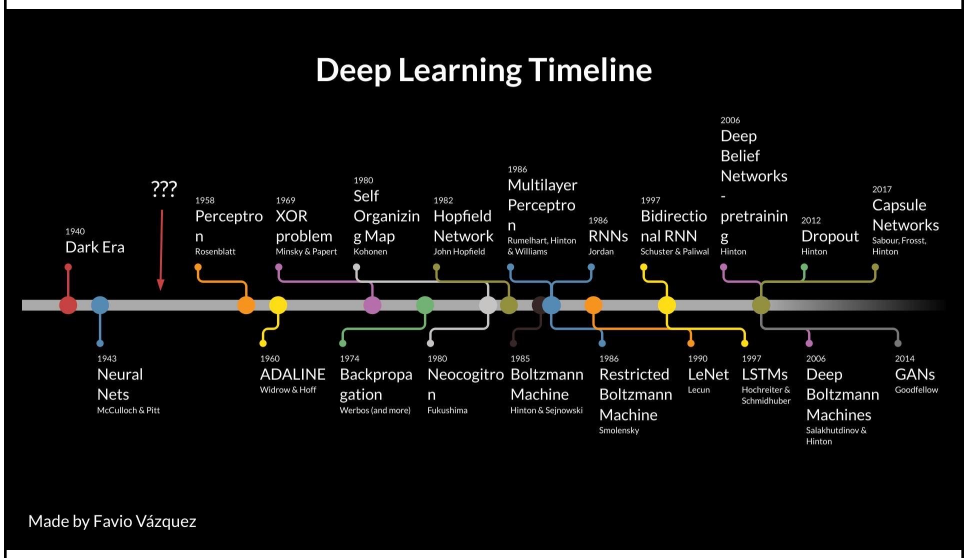
43



https://developer.ibm.com/articles/cc-cognitive-neural-networks-deep-dive/

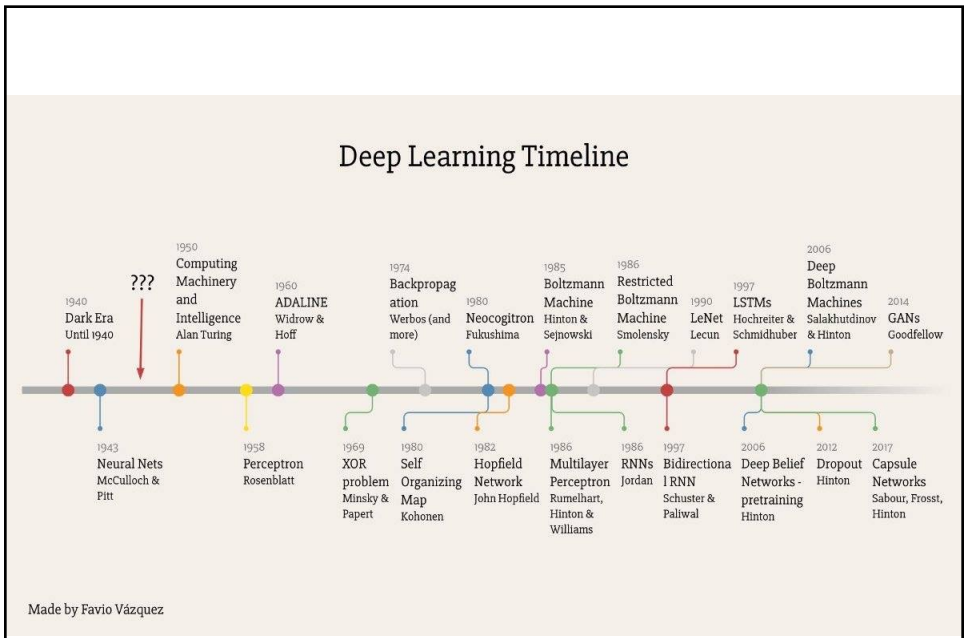
44

44


Deep Learning Timeline
1940
Dark Era
???
1943
Neural Nets
McCulloch & Pitt
1958
Perceptron
Rosenblatt
1960
ADALINE
Widrow & Hoff
1969
XOR problem
Minsky & Papert
1974
Backpropagation
Werbos (and more)
1980
Self Organizing Map
Kohonen
1980
Neocogitron
Fukushima
1982
Hopfield Network
John Hopfield
1985
Boltzmann Machine
Hinton & Sejnowski
1986
Multilayer Perceptron
Rumelhart, Hinton & Williams
1986
Restricted Boltzmann Machine
Smolensky
1986
RNNs
Jordan
1990
LeNet
Lecun
1997
Bidirectional RNN
Schuster & Paliwal
1997
LSTMs
Hochreiter & Schmidhuber
2006
Deep Belief Networks - pretraining
Hinton
2006
Deep Boltzmann Machines
Salakhutdinov & Hinton
2012
Dropout
Hinton
2014
GANs
Goodfellow
2017
Capsule Networks
Sabour, Frosst, Hinton
Made by Favio Vázquez

45

45


Deep Learning Timeline
1940
Dark Era
Until 1940
???
1943
Neural Nets
McCulloch & Pitt
1950
Computing Machinery and Intelligence
Alan Turing
1958
Perceptron
Rosenblatt
1960
ADALINE
Widrow & Hoff
1969
XOR problem
Minsky & Papert
1974
Backpropagation
Werbos (and more)
1980
Self Organizing Map
Kohonen
1980
Neocogitron
Fukushima
1982
Hopfield Network
John Hopfield
1985
Boltzmann Machine
Hinton & Sejnowski
1986
Multilayer Perceptron
Rumelhart, Hinton & Williams
1986
Restricted Boltzmann Machine
Smolensky
1986
RNNs
Jordan
1990
LeNet
Lecun
1997
Bidirectional RNN
Schuster & Paliwal
1997
LSTMs
Hochreiter & Schmidhuber
2006
Deep Belief Networks - pretraining
Hinton
2006
Deep Boltzmann Machines
Salakhutdinov & Hinton
2012
Dropout
Hinton
2014
GANs
Goodfellow
2017
Capsule Networks
Sabour, Frosst, Hinton
Made by Favio Vázquez

46

46

Deep (Feed Forward) Neural Networks

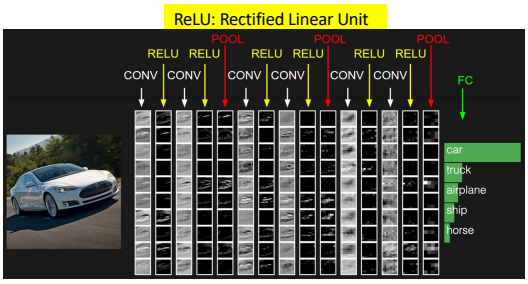
- How multiple layers work to build an improved feature space?
  - First layer learns 1st order features (e.g., edges, …)
  - 2nd layer learns higher order features (combinations of first layer features, combinations of edges, etc.)
  - In Deep Belief Networks (DBNs), layers often learn in an unsupervised mode and discover general features of the input space—serving multiple tasks related to the unsupervised instances (image recognition, etc.)
  - Then final layer features are fed into supervised layer(s)
    - And entire network is often subsequently tuned using supervised training of the entire net, using the initial weightings learned in the unsupervised phase
  - Could also do fully supervised versions (back-propagation)

47

Convolutional Neural Networks: General Architecture

- Learn high-order features in the data via convolutions
  - Well suited to object recognition with images (e.g., computer vision)
  - Build position- and (somewhat) rotation-invariant features from raw image data
- CNN leverages learnable visual filters and globally shared local features
  - Specifics: high dimensional, 2D topology of pixels, invariance to translations, etc.

- High-level general CNN architecture
  - Input layer
  - Feature-extraction layers (Convolution—ReLU—Pool)
  - Classification layers
- CNN properties
  - Local connectivity
  - Parameter sharing
  - Subsampling

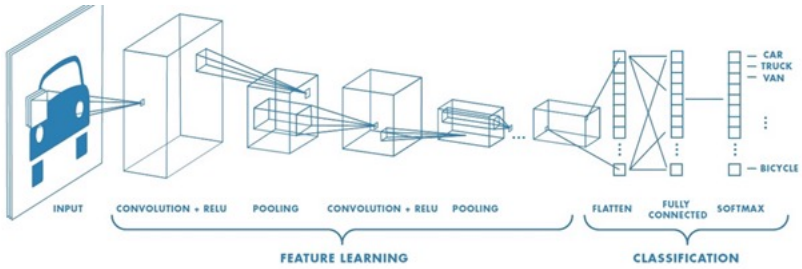


Larret et al 2009 from nyu http://cs231n.github.io/convolutional-networks/

48

Convolutional Neural Networks

CNN is a multi-layered neural network with a unique architecture designed to extract increasingly complex features of the data at each layer to determine the output. CNN's are well suited for perceptual tasks.

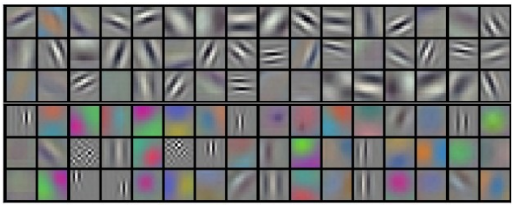


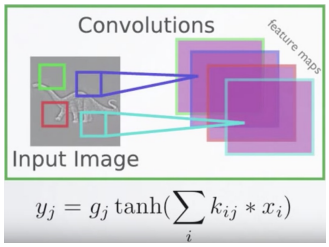
https://www.guru99.com/deep-learning-tutorial.html

49

Convolutional Neural Networks: Parameter Sharing

- Parameter sharing
  - Discrete convolution: share matrix of parameters across certain units
    - Reduces even more the number of parameters
    - Extract the same feature at every position





Larret et al 2009 from nyu http://cs231n.github.io/convolutional-networks/

50

Recurrent Neural Networks: General Concepts

- Modeling the time dimension: by creating cycles in the network (thus "recurrent")
  - Adding feedback loops connected to past decisions
  - Long-term dependencies: Use hidden states to preserve sequential information
- RNNs are trained to generate sequences: Output at each timestamp is based on both the current input and the inputs at all previous timestamps

$$\mathbf{h}_t = \phi\left(W\mathbf{x}_t + U\mathbf{h}_{t-1}\right),$$

  - Compute a gradient with the algorithm BPTT (backpropagation through time)
- Major obstacles of RNN: Vanishing and Exploding Gradients
  - When the gradient becomes too large or too small, it is difficult to model long-range dependencies (10 timestamps or more)
  - Solution: Use a variant of RNN: LSTM (1997, by Hochreiter and Schmidthuber)

https://deeplearning4j.org/lstm.html

51

Chapter 9. Classification: Advanced Methods

- Bayesian Networks
- Support Vector Machines
- Neural Networks and Deep Learning
- Lazy Learners and K-Nearest Neighbors

52

Lazy vs. Eager Learning

- Lazy vs. eager learning
  - **Lazy learning** (e.g., instance-based learning): Simply stores training data (or only minor processing) and waits until it is given instance must be classified
  - **Eager learning** (e.g. previously discussed methods): Given a set of training tuples, constructs a classification model before receiving new (e.g., test) data to classify
- Lazy: less time in training, but more time in predicting
  - Pros: Classification hypothesis is developed locally for each instance to be classified
  - Cons: Running time (no model is built, so each classification actually builds a local model from scratch)
- Eager: must commit to a single hypothesis that covers the entire instance space

53

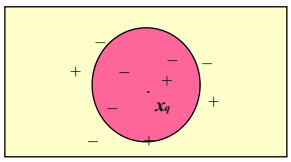
Lazy Learner: Instance-Based Methods

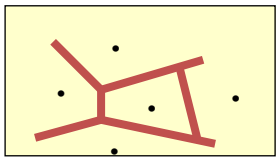
- Instance-based learning:
  - Store training examples and delay the processing ("lazy evaluation") until a new instance must be classified
- Typical approaches
  - *k*-nearest neighbor approach
    - Instances represented as points in a Euclidean space.
  - Case-based reasoning
    - Uses symbolic representations and knowledge-based inference

54

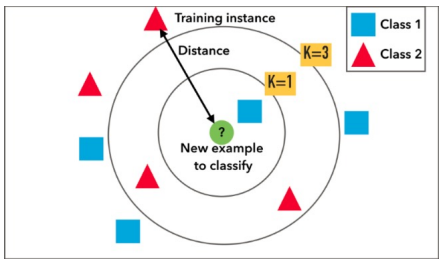
The *k*-Nearest Neighbor Algorithm

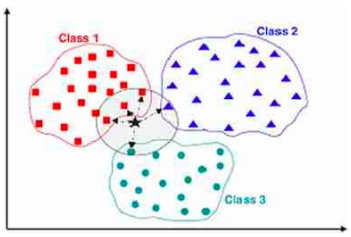
- All instances correspond to points in the n-D space
- The nearest neighbor are defined in terms of Euclidean distance, dist($\mathbf{X_1}$, $\mathbf{X_2}$)
- Target function could be discrete- or real- valued
- For discrete-valued, *k*-NN returns the most common value among the *k* training examples nearest to x_q
- Voronoi diagram: the decision surface induced by 1-NN for a typical set of training examples



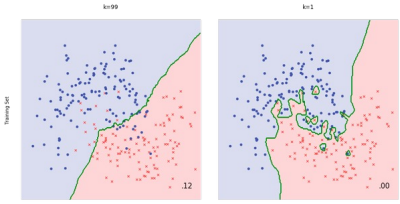


55





A small **value** of **k** means that noise will have a **higher** influence on the result and a large **value** make it computationally expensive.



K = 1 overfits the data to reduce overfitting we need to increase K value

56

56

Discussion on the *k*-NN Algorithm

- *k*-NN for real-valued prediction for a given unknown tuple
  - Returns the mean values of the *k* nearest neighbors
- Distance-weighted nearest neighbor algorithm
  - Weight the contribution of each of the *k* neighbors according to their distance to the query x_q $w \equiv \frac{1}{d(x_q, x_i)^2}$
    - Give greater weight to closer neighbors
- Robust to noisy data by averaging *k*-nearest neighbors
- Curse of dimensionality: distance between neighbors could be dominated by irrelevant attributes
  - To overcome it, axes stretch or elimination of the least relevant attributes

57

Case-Based Reasoning (CBR)

- **CBR**: Uses a database of problem solutions to solve new problems
- Store symbolic description (tuples or cases)—not points in a Euclidean space
- Applications: Customer-service (product-related diagnosis), legal ruling
- Methodology
  - Instances represented by rich symbolic descriptions (e.g., function graphs)
  - Search for similar cases, multiple retrieved cases may be combined
  - Tight coupling between case retrieval, knowledge-based reasoning, and problem solving
- Challenges
  - Find a good similarity metric
  - Indexing based on syntactic similarity measure, and when failure, backtracking, and adapting to additional cases

58

References

- C. M. Bishop, Neural Networks for Pattern Recognition. Oxford University Press, 1995
- L. Breiman, J. Friedman, R. Olshen, and C. Stone. Classification and Regression Trees. Wadsworth International Group, 1984
- C. J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2): 121-168, 1998
- N. Cristianini and J. Shawe-Taylor, Introduction to Support Vector Machines and Other Kernel-Based Learning Methods, Cambridge University Press, 2000
- H. Yu, J. Yang, and J. Han. Classifying large data sets using SVM with hierarchical clusters. KDD'03
- A. J. Dobson. An Introduction to Generalized Linear Models. Chapman & Hall, 1990
- R. O. Duda, P. E. Hart, and D. G. Stork. Pattern Classification, 2ed. John Wiley, 2001
- T. Hastie, R. Tibshirani, and J. Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer-Verlag, 2001
- S. Haykin, Neural Networks and Learning Machines, Prentice Hall, 2008
- D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. Machine Learning, 1995

59

Other Regression-Based Models

- Generalized linear model:
  - Foundation on which linear regression can be applied to modeling categorical response variables
  - Variance of y is a function of the mean value of y, not a constant
  - Logistic regression: models the prob. of some event occurring as a linear function of a set of predictor variables
  - Poisson regression: models the data that exhibit a Poisson distribution
- Log-linear models: (for categorical data)
  - Approximate discrete multidimensional prob. distributions
  - Also useful for data compression and smoothing
- Regression trees and model trees
  - Trees to predict continuous values rather than class labels

60

Regression Trees and Model Trees

- Regression tree: proposed in CART system (Breiman et al. 1984)
  - CART: Classification And Regression Trees
  - Each leaf stores a *continuous-valued prediction*
  - It is the *average value of the predicted attribute* for the training tuples that reach the leaf
- Model tree: proposed by Quinlan (1992)
  - Each leaf holds a regression model—a multivariate linear equation for the predicted attribute
  - A more general case than regression tree
- Regression and model trees tend to be more accurate than linear regression when the data are not represented well by a simple linear model

61