Wang Lab

UNIVERSITY OF MICHIGAN

JOINT INSTITUTE

交大密西根学院

Machine Learning Algorithms

Peng Shi

2016.11.15

Wireless and Networking Lab

Overview(1/2)

Part 1: Introduction and Basics

- What is learning
- The Categories of Machine Learning
- Capacity, Overfitting, Underfitting
- Regularization
- Stochastic Gradient Descent
- Bias and Variance

Part 2: Supervised Learning

- Support Vector Machine
- Neural Networks
- Decision Tree
- Random Forests
- AdaBoost

Part 3: Unsupervised Learning

- Principal Components Analysis
- K-means Clustering
- Auto-Encoder
- Compressive Sensing

Part 4: Semi-supervised Learning

- Self-Training
- Generative Models
- Transductive SVMs
- Graph-based Algorithm
- Multi-view Algorithm

Overview(2/2)

Part 5: Reinforcement Learning

- Background: Markov Decision Process
- Reinforcement Learning: Actor Critic
- Reinforcement Learning: Q-Learning

Part 6: Deep Learning

- Intuition
- Convolutional Neural Network
- Recurrent and Recursive Nets

Part 7: More Topics

- How to Choose
- The history of RBF

Part 1
Introduction and Basics

- What is learning
- The Categories of Machine Learning
- Capacity, Overfitting, Underfitting
- Regularization
- Stochastic Gradient Descent
- Bias and Variance

Introduction

0.1 What is Learning: Definition

A computer program is said to learn from experience **E** with respect to some class of tasks **T** and performance measure **P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E**. see Mitchell (1997)

Example 1: A checkers learning problem

Task T: playing checkers

Performance measure P: percent of games won against opponents

Training experience E: playing practice games against itself

Example 2: A robot driving learning problem

Task T: driving on public four-lane highways using vision sensors

Performance measure P: average distance traveled before an error(as judged by human overseer)

Training experience E: a sequence of images and steering commands recorded while observing a human driver

[1] Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville, http://www.deeplearningbook.org, Book in preparation for MIT Press,2016

Introduction

0.1 What is Learning: Examples of Task(1/3)

- Classification: An example of a classification task is object recognition, where the input is an image (usually described as a set of pixel brightness values), and the output is a numeric code identifying the object in the image. For example, the Willow Garage PR2 robot is able to act as a waiter that can recognize different kinds of drinks and deliver them to people on command (Good-fellow et al., 2010). Modern object recognition is best accomplished with deep learning (Krizhevsky et al., 2012; Ioffe and Szegedy, 2015). Object recognition is the same basic technology that allows computers to recognize faces (Taigman et al., 2014), which can be used to automatically tag people in photo collections and allow computers to interact more naturally with their users.
- Classification with missing inputs: When some of the inputs may be missing, rather than providing a single classification function, the learning algorithm must learn a set of functions. One way to efficiently define such a large set of functions is to learn a probability distribution over all of the relevant variables, then solve the classification task by marginalizing out the missing variables.

Introduction

0.1 What is Learning: Examples of Task(2/3)

- Regression: In this type of task, the computer program is asked to predict a numerical value given some input.
- Transcription: In this type of task, the machine learning system is asked to observe a relatively unstructured representation of some kind of data and transcribe it into discrete, textual form. Google Street View uses deep learning to process address numbers in this way (Goodfellow et al.,2014d). Another example is speech recognition, where the computer program is provided an audio waveform and emits a sequence of characters or word ID codes describing the words that were spoken in the audio recording. Deep learning is a crucial component of modern speech recognition systems used at major companies including Microsoft, IBM and Google (Hinton et al.,2012b).
- Machine translation: In a machine translation task, the input already consists of a sequence of symbols in some language, and the computer program must convert this into a sequence of symbols in another language.

Introduction

0.1 What is Learning: Examples of Task(3/3)

- Structured output: Structured output tasks involve any task where the output is a vector (or other data structure containing multiple values) with important relationships between the different elements.
- Anomaly detection: In this type of task, the computer program sifts through a set of events or objects, and flags some of them as being unusual or atypical.
- Synthesis and sampling: In this type of task, the machine learning algorithm is asked to generate new examples that are similar to those in the training data.
- Imputation of missing values: In this type of task, the machine learning algorithm is given a new example, but with some entries missing. The algorithm must provide a prediction of the values of the missing entries.
- Denoising: In this type of task, the machine learning algorithm is given in input a corrupted example obtained by an unknown corruption process from a clean example. The learner must predict the clean example from its corrupted version, or more generally predict the conditional probability distribution.

[1] *Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville, http://www.deeplearningbook.org, Book in preparation for MIT Press,2016*

Introduction

0.1 What is Learning: Performance Measure

- For tasks such as classification, classification with missing inputs, and transcription, we often measure the accuracy of the model. Accuracy is just the proportion of examples for which the model produces the correct output.
- Usually we are interested in how well the machine learning algorithm performs on data that it has not seen before. We therefore evaluate these performance measures using a test set of data that is separate from the data used for training the machine learning system.

Difficult to choose a performance measure

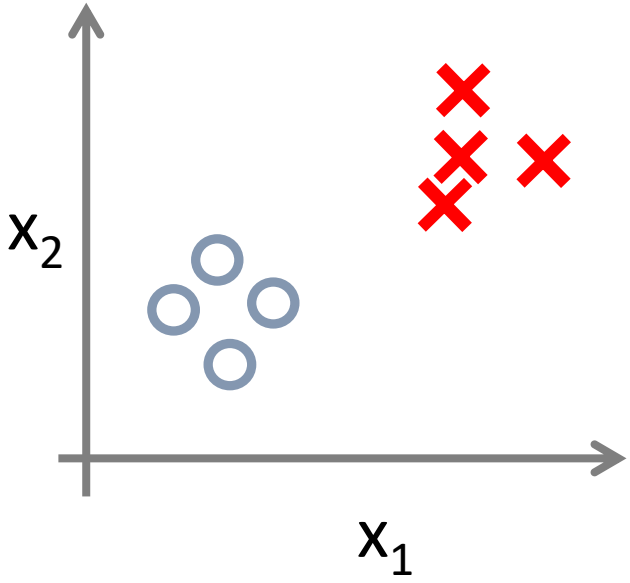
- In some cases(transcription task, regression task), this is because it is difficult to decide what should be measured. These kinds of design choices depend on the application.
- In other cases(density estimation), we know what quantity we would ideally like to measure, but measuring it is impractical. In these cases, one must design an alternative criterion that still corresponds to the design objectives, or design a good approximation to the desired criterion.

Introduction

0.2 The Categories of Machine Learning(1/4)

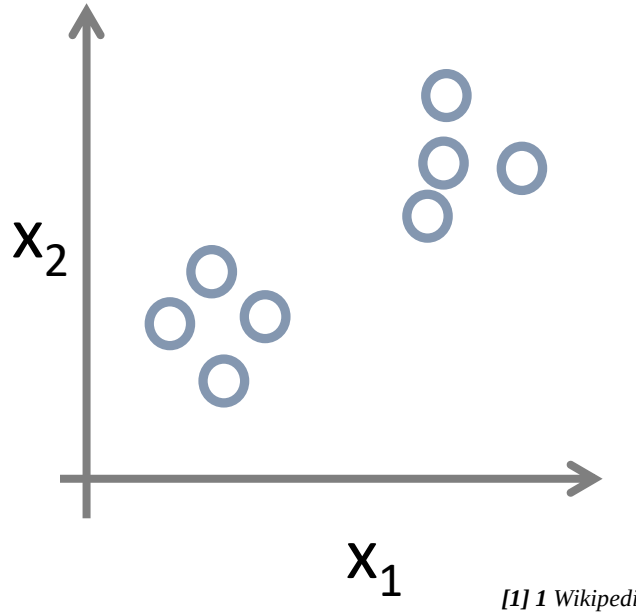
Supervised Learning

- Supervised learning is the machine learning task of inferring a function from labeled training data. The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal).



Unsupervised Learning

- Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labeled responses. The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data.



[1] 1 Wikipedia

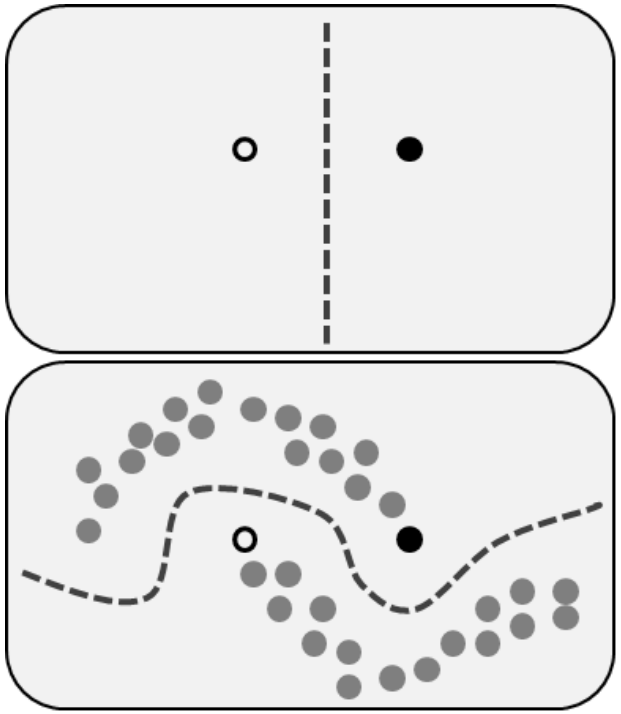
[1] 2 https://azure.microsoft.com/en-us/documentation/articles/machine-learning-algorithm-choice/

Introduction

0.2 The Categories of Machine Learning(2/4)

Semi-supervised Learning

- Semi-supervised learning is a class of supervised learning tasks and techniques that also make use of unlabeled data for training – typically a small amount of labeled data with a large amount of unlabeled data.



An example of the influence of unlabeled data in semi-supervised learning. The top panel shows a decision boundary we might adopt after seeing only one positive (white circle) and one negative (black circle) example. The bottom panel shows a decision boundary we might adopt if, in addition to the two labeled examples, we were given a collection of unlabeled data (gray circles). This could be viewed as performing clustering and then labeling the clusters with the labeled data, pushing the decision boundary away from high-density regions, or learning an underlying one-dimensional manifold where the data reside.

[1] Wikipedia

[2] https://azure.microsoft.com/en-us/documentation/articles/machine-learning-algorithm-choice/

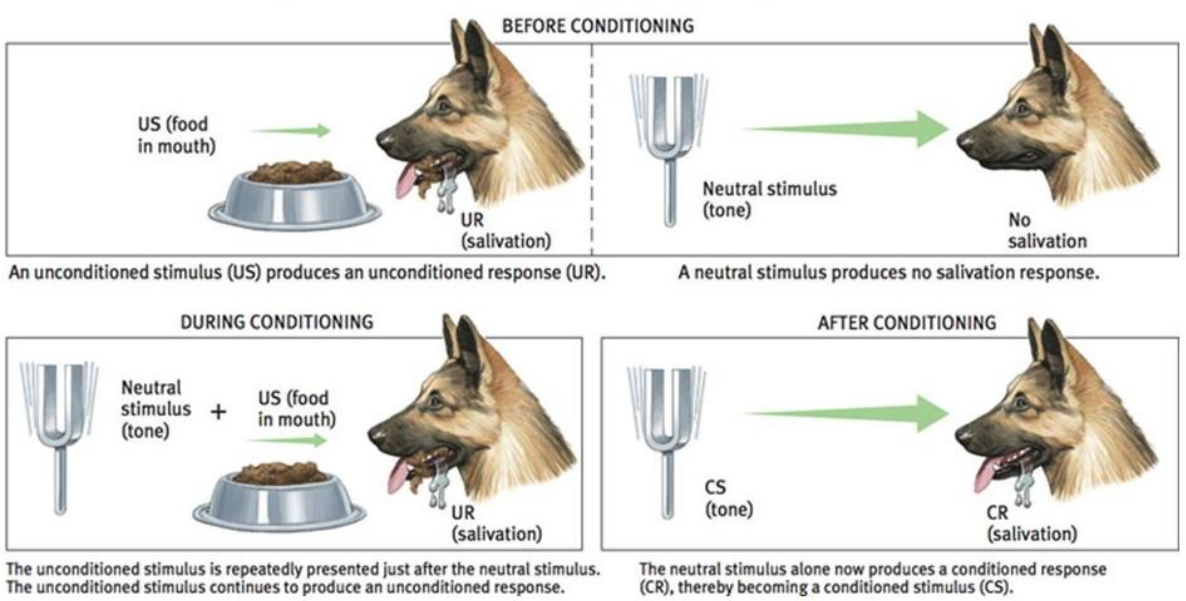
Introduction

0.2 The Categories of Machine Learning(3/4)

Reinforcement Learning

- In reinforcement learning, the algorithm gets to choose an action in response to each data point. The learning algorithm also receives a reward signal a short time later, indicating how good the decision was. Based on this, the algorithm modifies its strategy in order to achieve the highest reward.

- Reinforcement learning is common in robotics, where the set of sensor readings at one point in time is a data point, and the algorithm must choose the robot's next action. It is also a natural fit for Internet of Things applications.



[1] 1 Wikipedia

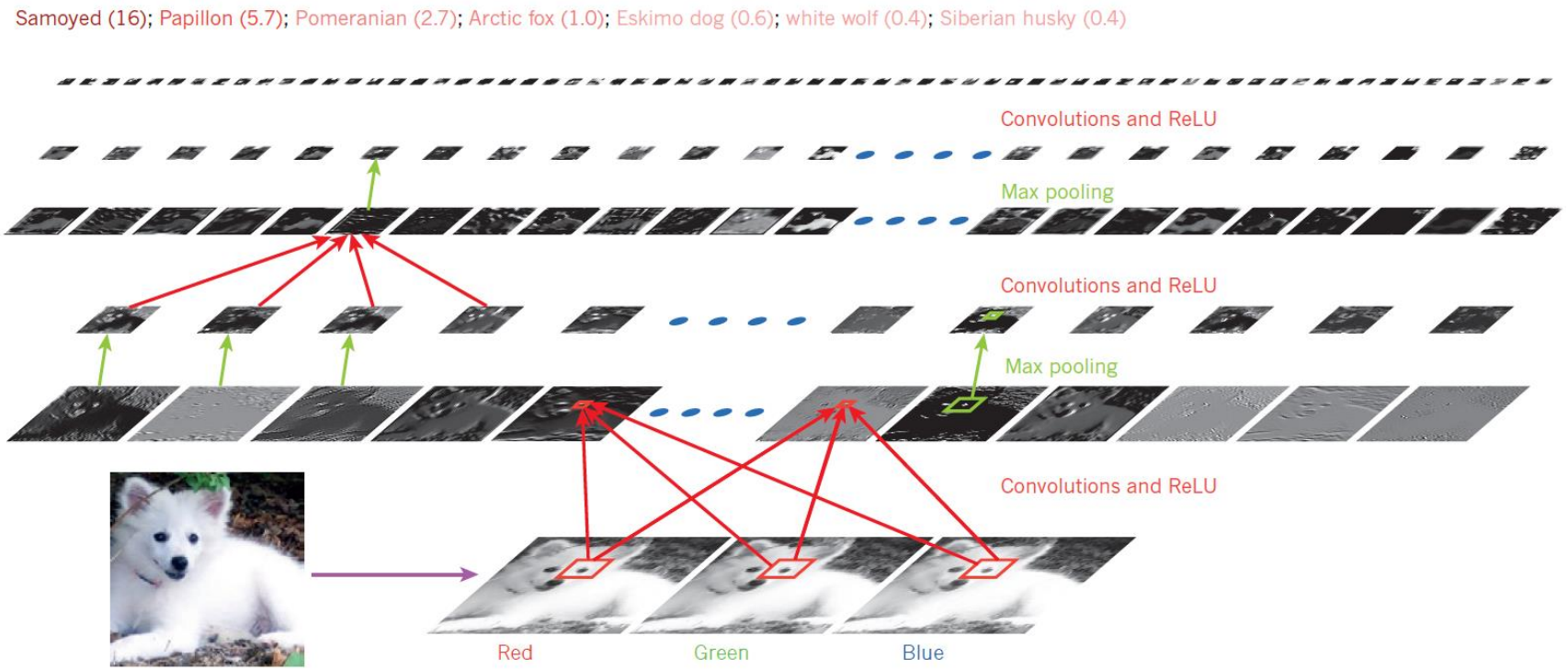
[1] 2 https://azure.microsoft.com/en-us/documentation/articles/machine-learning-algorithm-choice/

Introduction

0.2 The Categories of Machine Learning(4/4)

Deep Learning

- Deep learning (also known as deep structured learning, hierarchical learning or deep machine learning) is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using multiple processing layers, with complex structures or otherwise, composed of multiple non-linear transformations.



[1] 1 Wikipedia

[1] 2 https://azure.microsoft.com/en-us/documentation/articles/machine-learning-algorithm-choice/

Basics

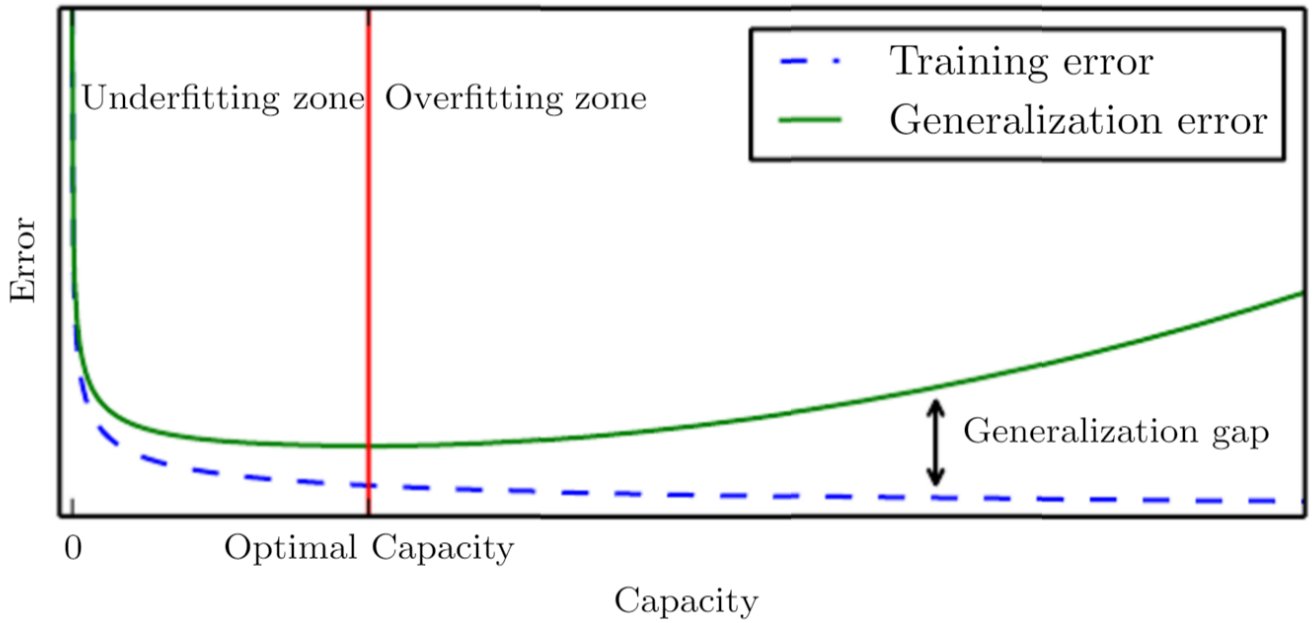
1.1 Capacity, Overfitting, Underfitting(1/2)

- The central challenge in machine learning is that we must perform well on new, previously unseen inputs—not just those on which our model was trained. The ability to perform well on previously unobserved inputs is called generalization.
- A model’s capacity is its ability to fit a wide variety of functions. Models with low capacity may struggle to fit the training set. Models with high capacity can overfit by memorizing properties of the training set that do not serve them well on the test set. One way to control the capacity of a learning algorithm is by choosing its hypothesis space, the set of functions that the learning algorithm is allowed to select as being the solution.
- The factors determining how well a machine learning algorithm will perform are its ability to:(1) Make the training error small. (2) Make the gap between training and test error small.

Basics

1.1 Capacity, Overfitting, Underfitting(2/2)

- Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set. Overfitting occurs when the gap between the training error and test error is too large.



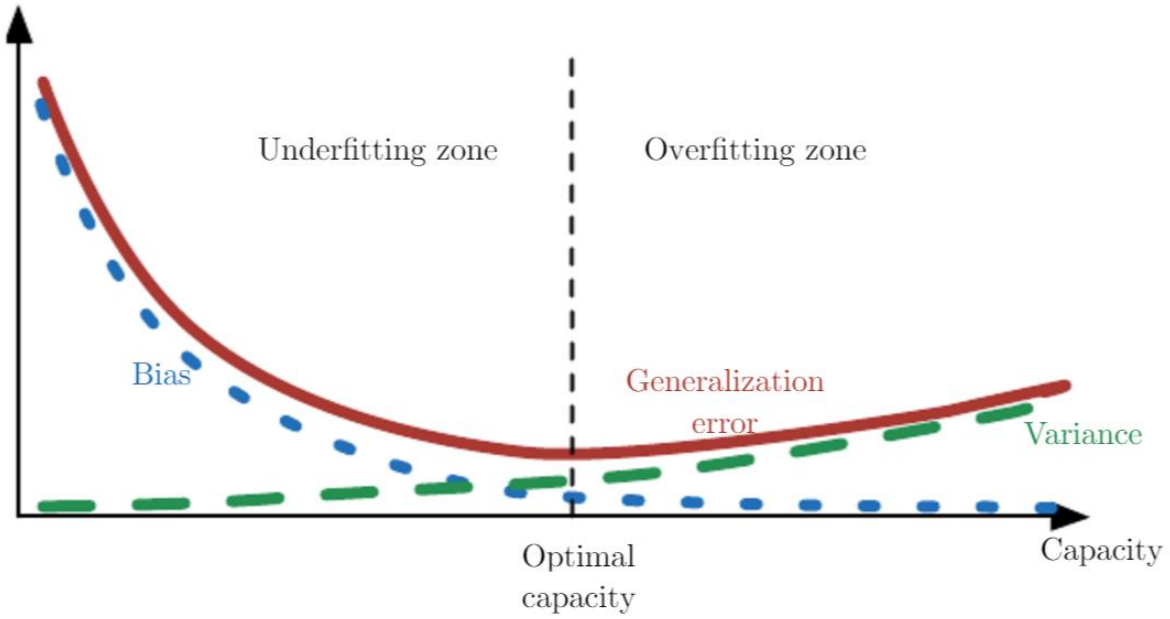
Typical relationship between capacity and error.

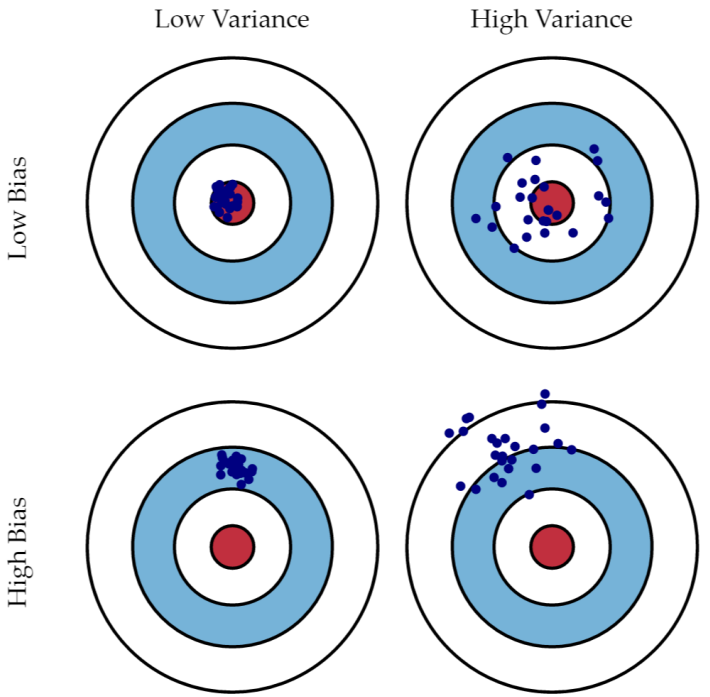
[1] Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville, http://www.deeplearningbook.org, Book in preparation for MIT Press,2016

Basics

1.2 Bias and Variance

- Bias measures the expected deviation from the true value of the function or parameter.
- Variance on the other hand, provides a measure of the deviation from the expected estimator value that any particular sampling of the data is likely to cause.
- The most common way to negotiate this trade-off is to use cross-validation.
- $\mathrm{MSE} = \mathrm{E}[(\hat{\theta}_m - \theta)^2] = Bias(\hat{\theta}_m)^2 + Var(\hat{\theta}_m)$





[1] Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville, http://www.deeplearningbook.org, Book in preparation for MIT Press,2016

Basics

1.3 Regularization

- We must design our machine learning algorithms to perform well on a specific task. We do so by building a set of preferences into the learning algorithm. When these preferences are aligned with the learning problems we ask the algorithm to solve, it performs better.
- In general, a regularization term $R(f)$ is introduced to a general loss function:

$$\min_f \sum_{i=1}^{n} V(f(\widehat{x_i}), \widehat{y_i}) + \lambda R(f)$$

for a loss function V that describes the cost of predicting $f(x)$ when the label is y, such as the square loss or hinge loss, and for the term λ which controls the importance of the regularization term. $R(f)$ is typically a penalty on the complexity of f, such as restrictions for smoothness or bounds on the vector space norm.

- there is no best machine learning algorithm, and, in particular, no best form of regularization. Instead we must choose a form of regularization that is well-suited to the particular task we want to solve.

[1] *Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville, http://www.deeplearningbook.org, Book in preparation for MIT Press,2016*

Basics

1.4 Stochastic Gradient Descent(1/3)

Description

- Nearly all of deep learning is powered by one very important algorithm: stochastic gradient descent or SGD.
- Both statistical estimation and machine learning consider the problem of minimizing an objective function that has the form of a sum:

$$\mathrm{Q}(w) = \sum_{i=1}^{n} \mathrm{Q_i}(w)$$

Where the parameter w which minimizes Q(w) is to be estimated. Each summand function $\mathrm{Q_i}(w)$ is typically associated with the i-th observation in the data set (used for training).

[1] *Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville, http://www.deeplearningbook.org, Book in preparation for MIT Press,2016*

Basics

1.4 Stochastic Gradient Descent(2/3)

Description

- When used to minimize the above function, a standard (or "batch") gradient descent method would perform the following iterations :

$$w := w - \eta \nabla Q(w) = w - \eta \sum_{i=1}^{n} \nabla \mathrm{Q_i}(w)$$

Where η is a step size (sometimes called the learning rate in machine learning).

Iterative method

- Choose an initial vector of parameters w and learning rate η.
- Repeat until an approximate minimum is obtained:

  Randomly shuffle examples in the training set. For i=1, 2, ..., n, do:

$$w := w - \eta \nabla Q_i(w)$$

[1] *Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville, http://www.deeplearningbook.org, Book in preparation for MIT Press,2016*

Basics

1.4 Stochastic Gradient Descent(3/3)

Comments

- The insight of stochastic gradient descent is that the gradient is an expectation. The expectation may be approximately estimated using a small set of samples. Specifically, on each step of the algorithm, we can sample a mini batch of examples drawn uniformly from the training set.
- The machine learning models work very well when trained with gradient descent. The optimization algorithm may not be guaranteed to arrive at even a local minimum in a reasonable amount of time, but it often finds a very low value of the cost function quickly enough to be useful.

Applications

- Stochastic gradient descent is a popular algorithm for training a wide range of models in machine learning, including (linear) support vector machines, logistic regression and graphical models. When combined with the backpropagation algorithm, it is the standard algorithm for training artificial neural networks.

[1] *Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville, http://www.deeplearningbook.org, Book in preparation for MIT Press,2016*

Part 2
Supervised Learing

- Support Vector Machine
- Neural Networks
- Decision Tree
- Random Forests
- AdaBoost

Supervised Learning

2.1 Support Vector Machine: History

- The original SVM algorithm was invented by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in 1963.
- In 1992, Bernhard E. Boser, Isabelle M. Guyon and Vladimir N. Vapnik suggested a way to create nonlinear classifiers by applying the kernel trick to maximum-margin hyperplanes.
- The current standard incarnation (soft margin) was proposed by Corinna Cortes and Vapnik in 1993 and published in 1995.

[1] *Wikipedia , https://en.wikipedia.org/wiki/Support_vector_machine*

Supervised Learning

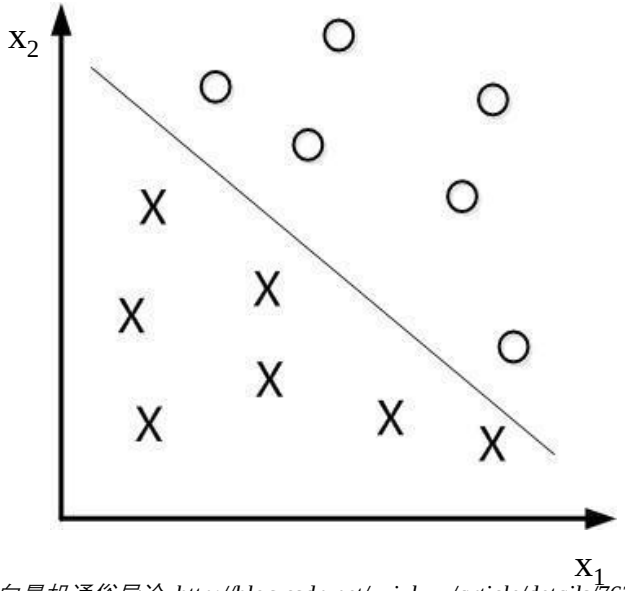
2.1 Support Vector Machine: Linear Classifier

Start from a simple case: **the data is linearly separable**. The method is also known as **Linear Classifier**.

- Given a set of training examples, each marked for belonging to one of two categories.
- Use **x** to represent the points. **x** is an m-dimensional vector. In the right picture, **x** is a two-dimensional vector.
- Use y to represent the categories. E.g. y can be 1 and -1 to denote the two categories.

- The goal is to find a hyperplane in the n-dimension space to split the data of different categories.
- The equation of the hyperplane can be written by

$$\boldsymbol{w}^T\boldsymbol{x} + b = 0$$



[1] 支持向量机通俗导论, http://blog.csdn.net/v_july_v/article/details/7624837

Supervised Learning

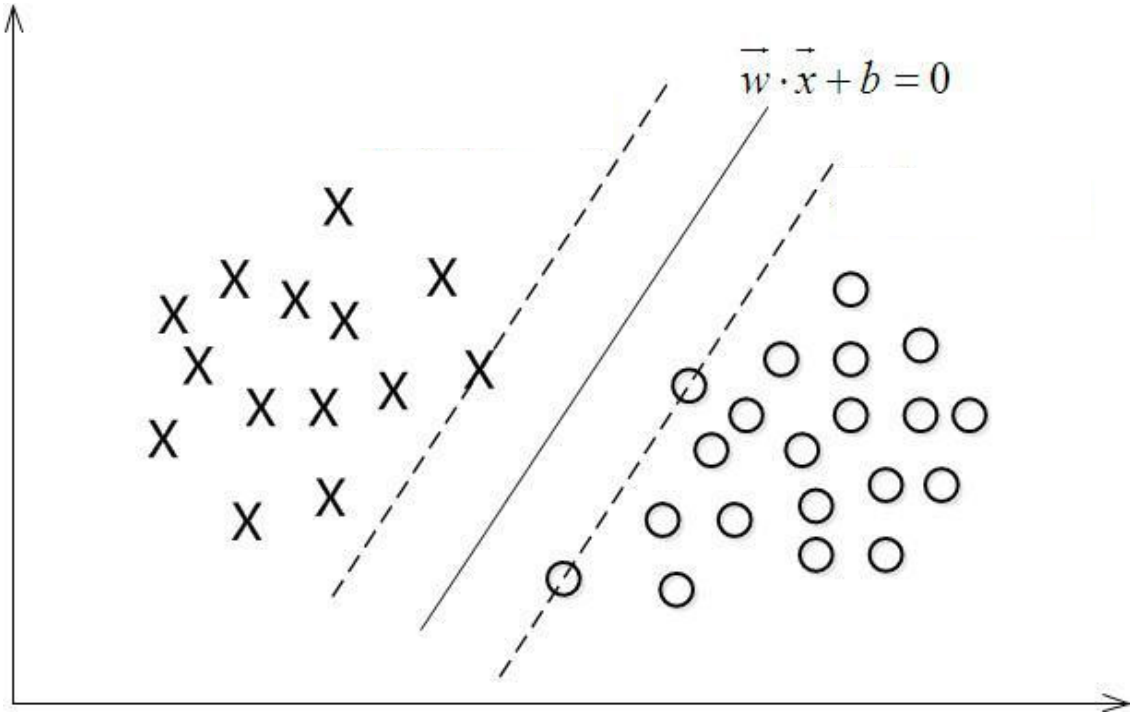
2.1 Support Vector Machine: Linear Classifier

Example:

- Two dimension, two category
- Classification Function

$$f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$$

- If $f(\boldsymbol{x})>0$, y=1. If $f(\boldsymbol{x})<0$, y=-1.



If we got the hyperplane and classification function, when it comes a new data point **x**, we could classify it by calculating $f(\boldsymbol{x})$.

So, the question is, how to get the hyperplane and classification function?

[1] 支持向量机通俗导论, http://blog.csdn.net/v_july_v/article/details/7624837

Supervised Learning

2.1 Support Vector Machine: Linear Classifier

If the training data are linearly separable, we can select two hyperplanes in a way that they separate the data and there are no points between them, and then try to maximize their distance. The region bounded by them is called "the margin".

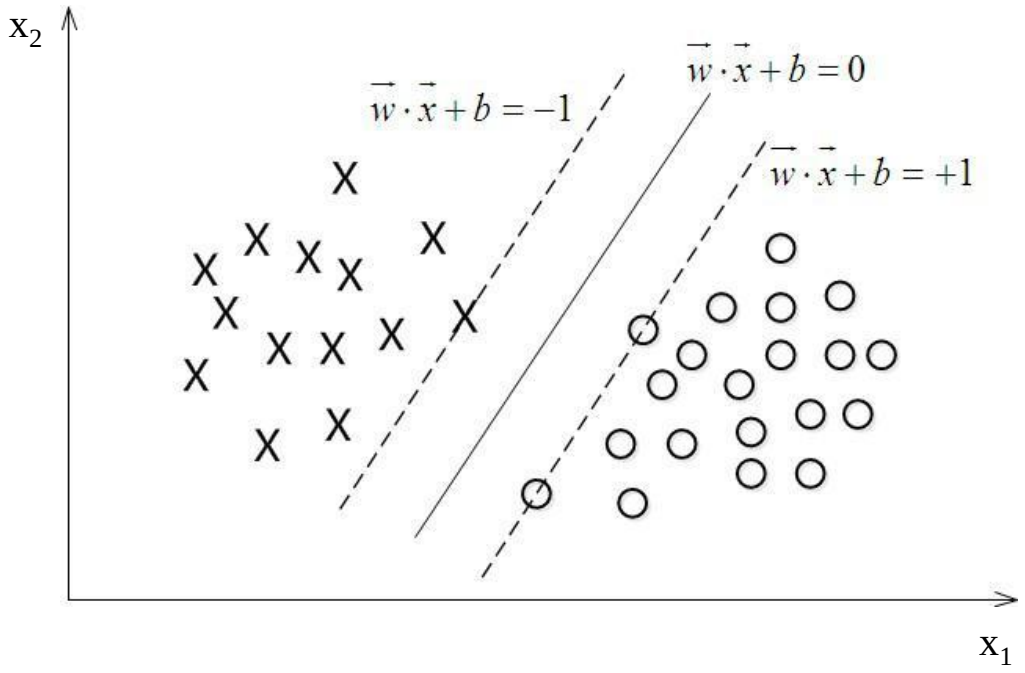
As $\boldsymbol{w}$ and b could scale at any rate, and it doesn't change the separate function $\boldsymbol{w}^T\boldsymbol{x} + b = 0$

So, Normalize the two hyperplanes as

$$\boldsymbol{w}^T\boldsymbol{x} + b = 1 \quad and \quad \boldsymbol{w}^T\boldsymbol{x} + b = -1$$

And we could get the constraint:

$$y^{(i)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b\right) \geq 1, i = 1,2, \dots, m$$



[1] 支持向量机通俗导论, http://blog.csdn.net/v_july_v/article/details/7624837

Supervised Learning

2.1 Support Vector Machine: Linear Classifier

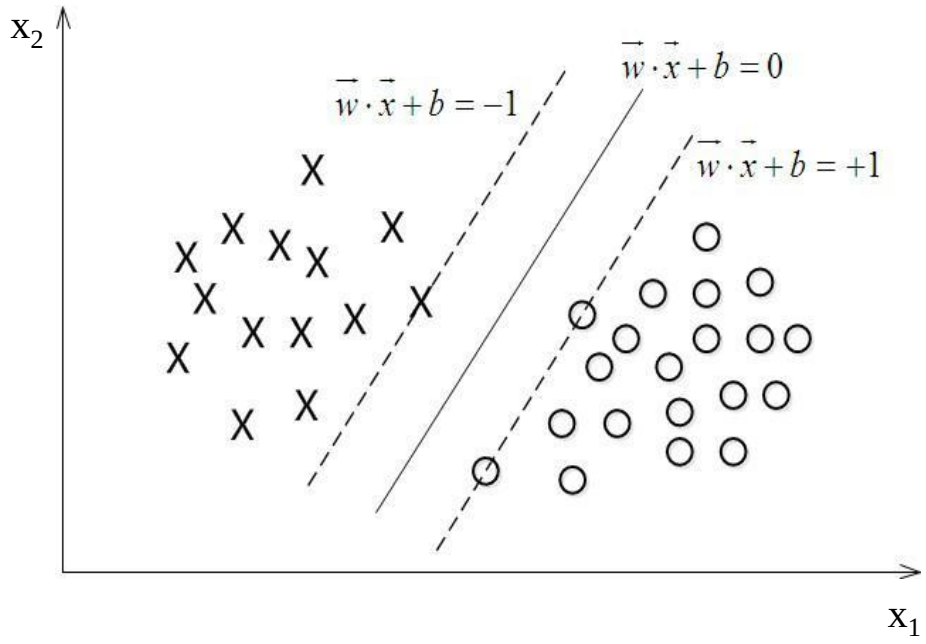
Geometrically, the distance between these two hyperplanes is $\frac{2}{||\boldsymbol{w}||^2}$, so to maximize the distance between the planes is to minimize $\frac{1}{2}||\boldsymbol{w}||^2$

The question turns into:

$$\min \frac{1}{2}||\boldsymbol{w}||^2$$

$$s.t.\ \ y^{(i)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(i)}+b\right) \geq 1, i = 1,2,\dots,m$$

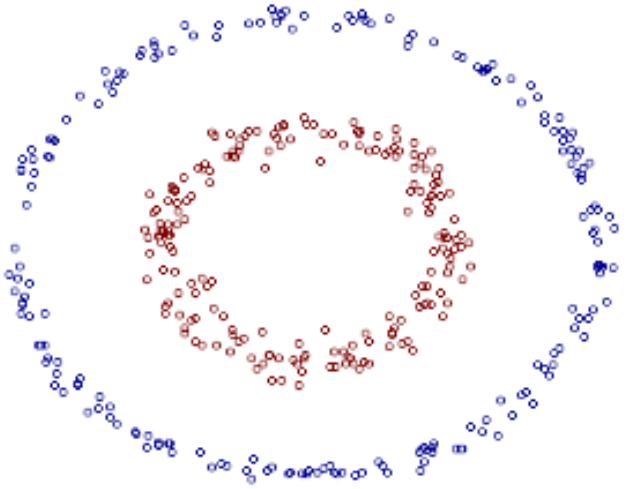
It can be solved by convex quadratic programming.



[1] 支持向量机通俗导论, http://blog.csdn.net/v_july_v/article/details/7624837

Supervised Learning

2.1 Support Vector Machine: Linear Classifier



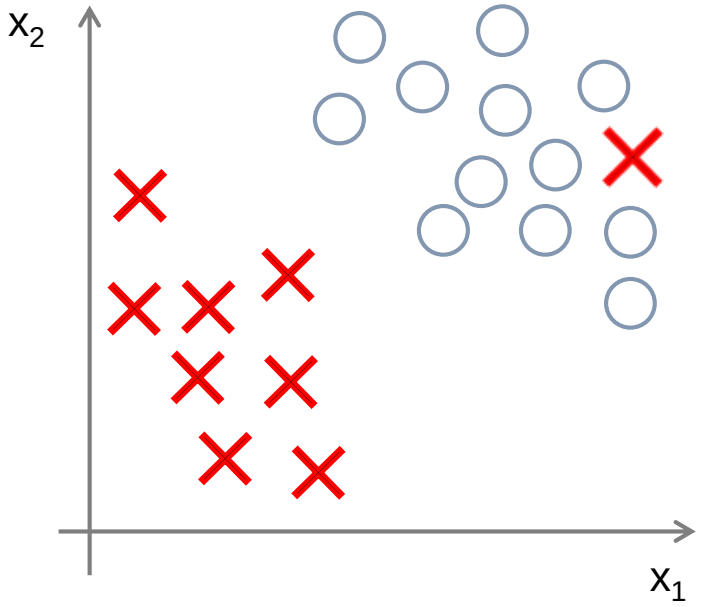
The question turns into:

$$\min \frac{1}{2}||\boldsymbol{w}||^2$$

$$s.t.\ \ y^{(i)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(i)}+b\right)\geq 1, i=1,2,\ldots,n$$

Based on the assumption: **linearly separable**

If the data set is not linearly separable, we got two problems.



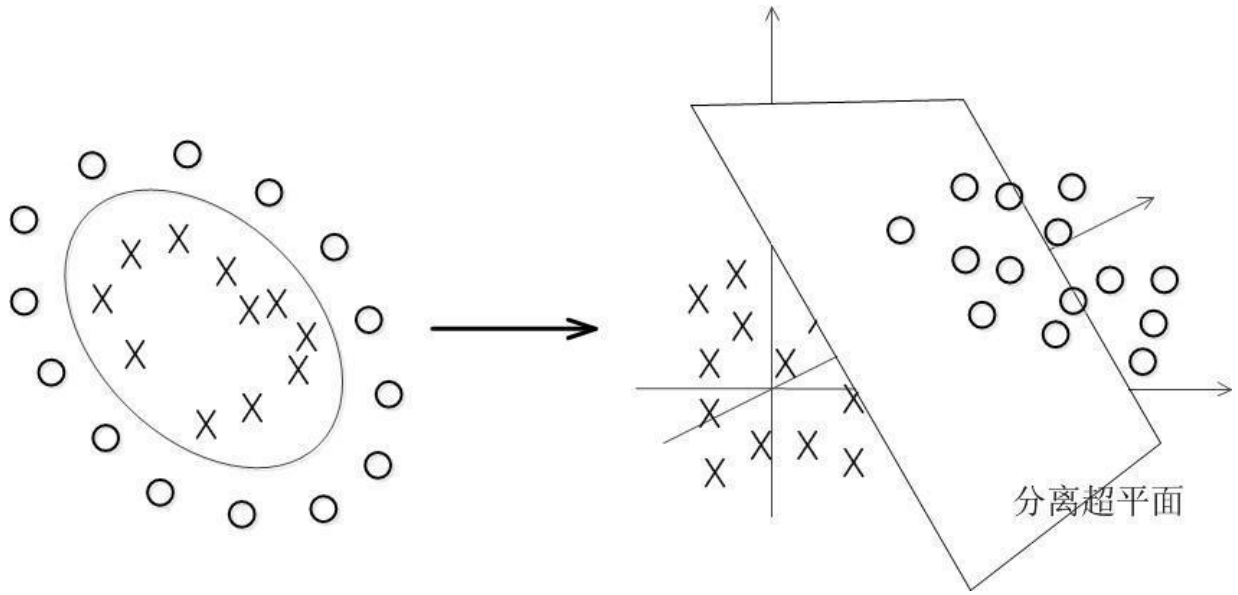
[1] 支持向量机通俗导论, http://blog.csdn.net/v_july_v/article/details/7624837

Supervised Learning

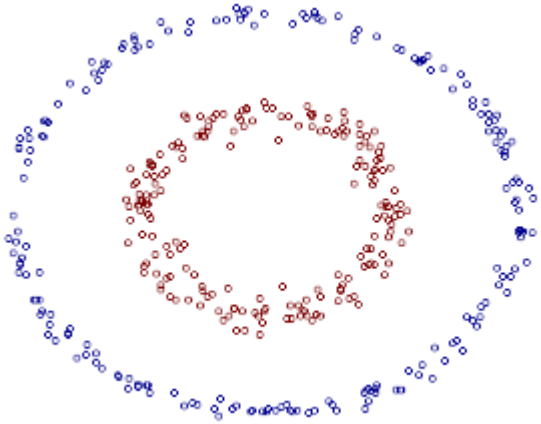
2.1 Support Vector Machine: Kernel

The basic idea is to map the data into the high-dimensional feature space.

The classifier is a hyperplane in the high-dimensional feature space, it may be nonlinear in the original input space.



[1] 支持向量机通俗导论, http://blog.csdn.net/v_july_v/article/details/7624837



Example

The equation for a quadratic curve:

$$a_1x_1 + a_2x_1^2 + a_3x_2 + a_4x_2^2 + a_5x_1x_2 + a_6 = 0$$

Do the map: $R^2 \rightarrow R^5$:

$$f_1 = x_1, f_2 = x_1^2, f_3 = x_2, f_4 = x_2^2, f_5 = x_1x_2$$

The equation turns into hyperplane function:

$$a_1f_1 + a_2f_2 + a_3f_3 + a_4f_4 + a_5f_5 + a_6 = 0$$

The data is linearly separable in the 5-dimension space.

[1] 支持向量机通俗导论, http://blog.csdn.net/v_july_v/article/details/7624837

A simpler example

As we cannot draw a 5-dimention space, to make it clear, I use $R^2 \rightarrow R^3$ as an example

Consider a quadratic curve equation:

$$a_1 x_1^2 + a_2 x_2 + a_3 x_2^2 + a_4 = 0$$

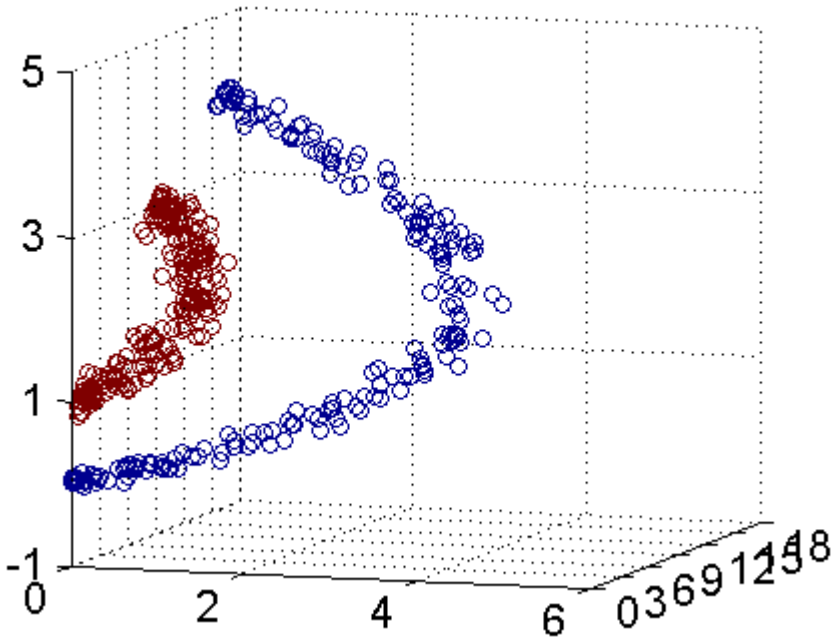
Do the map: $R^2 \rightarrow R^3$:

$$f_1 = x_1^2, f_2 = x_2, f_3 = x_2^2$$

The equation turns into hyperplane function:

$$a_1 f_1 + a_2 f_2 + a_3 f_3 + a_4 = 0$$

The data is linearly separable in the 3-dimension.



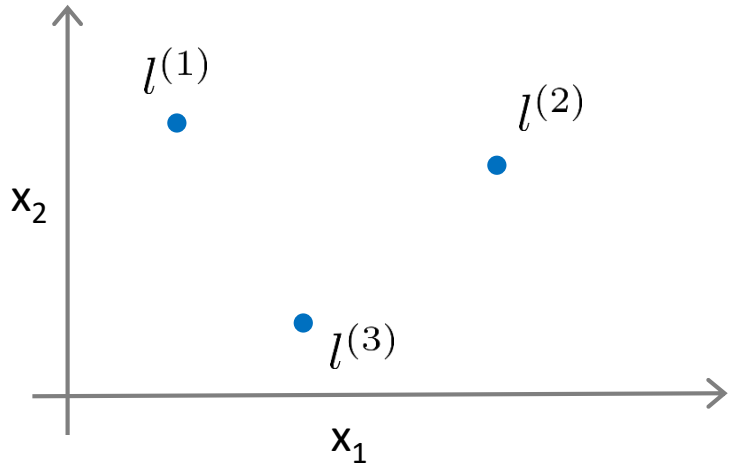
Is there a different / better choice of the features f_1, f_2, f_3?

[1] 支持向量机通俗导论, http://blog.csdn.net/v_july_v/article/details/7624837

Supervised Learning

2.1 Support Vector Machine: Kernel

Given $\boldsymbol{x}$, compute new feature depending on proximity to landmarks $l^{(1)}, l^{(2)}, l^{(3)}$

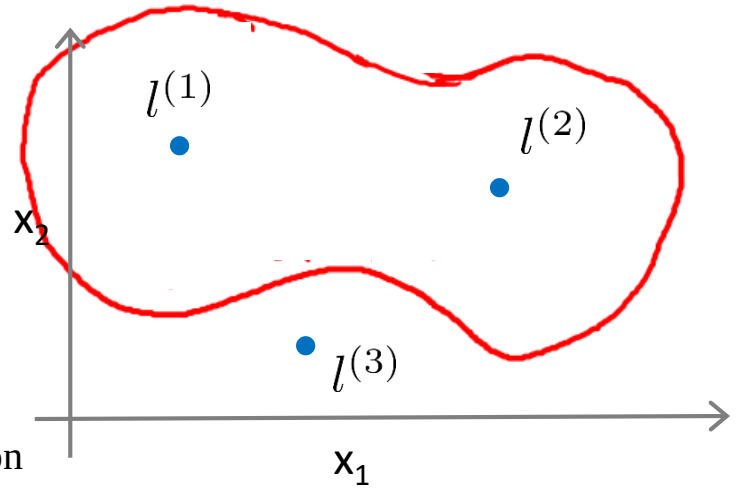


For any **x**, Do the map, got the three features :

$$f_i = \exp(-\frac{\left|\left|\boldsymbol{x} - l^{(i)}\right|\right|^2}{2\sigma^2})$$

If $w_1 f_1 + w_2 f_2 + w_3 f_3 + b > 0, y = 1$

If $w_1 f_1 + w_2 f_2 + w_3 f_3 + b < 0, y = -1$



For example:

$$w_1 = 1, w_2 = 1, w_3 = 0, b = 0.5$$

The Separate hyperplane would be a non-linear curve in the two-dimension space.

[1] 支持向量机通俗导论, http://blog.csdn.net/v_july_v/article/details/7624837

Supervised Learning

2.1 Support Vector Machine: Kernel

Given Training Data, $(\boldsymbol{x}^{(1)}, y^{(1)})$, $(\boldsymbol{x}^{(2)}, y^{(2)})$,…, $(\boldsymbol{x}^{(m)}, y^{(m)})$

- For $\boldsymbol{x}^{(i)}$, Calculate the m-dimensional $\boldsymbol{f}^{(i)}$**:**

$$f_1{}^{(i)} = K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(1)}), f_2{}^{(i)} = K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(2)}),\ldots, f_m{}^{(i)} = K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(m)})$$

y=1, if $\boldsymbol{w}^T\boldsymbol{f} + b > 0$

y =-1, if $\boldsymbol{w}^T\boldsymbol{f} + b < 0$

The original question turns into:

$$\min \frac{1}{2}||\boldsymbol{\alpha}||^2$$

$$s.t.\ \ y^{(i)}\left(\boldsymbol{\alpha}^T\boldsymbol{f}^{(i)} + b\right) \geq 1, i = 1,2,\ldots,m$$

[1] 支持向量机通俗导论, http://blog.csdn.net/v_july_v/article/details/7624837

Common Kernels

- Linear: $K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \boldsymbol{x}^{(i)^T}\boldsymbol{x}^{(j)}$
- Polynomial: $K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = (\gamma \boldsymbol{x}^{(i)^T}\boldsymbol{x}^{(j)} + r)^d, \gamma > 0$
- Radial Basis Function(RBF): $K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \exp\left(-\gamma \left\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\right\|^2\right), \gamma > 0$
- Sigmoid: $K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \tanh(\gamma \boldsymbol{x}^{(i)^T}\boldsymbol{x}^{(j)}+\mathrm{r})$

γ, r and d are kernel parameters

[1] A Practical Guide to Support Vector Classification, http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

The Reasons of Kernel Trick

- It allows us to learn models that are nonlinear as a function of $\boldsymbol{x}$ using convex optimization techniques that are guaranteed to converge efficiently. The optimization algorithm can view the decision function as being linear in a different space.
- The kernel function K often admits an implementation that is significantly more computational efficient than naively constructing two $\varphi(*)$ vectors and explicitly taking their dot product.

Drawbacks to Kernel Machines

- Kernel machines also suffer from a high computational cost of training when the dataset is large.
- Kernel machines with generic kernels struggle to generalize well.

[1] *Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville, http://www.deeplearningbook.org, Book in preparation for MIT Press,2016*

Supervised Learning

2.1 Support Vector Machine: Slack Variable

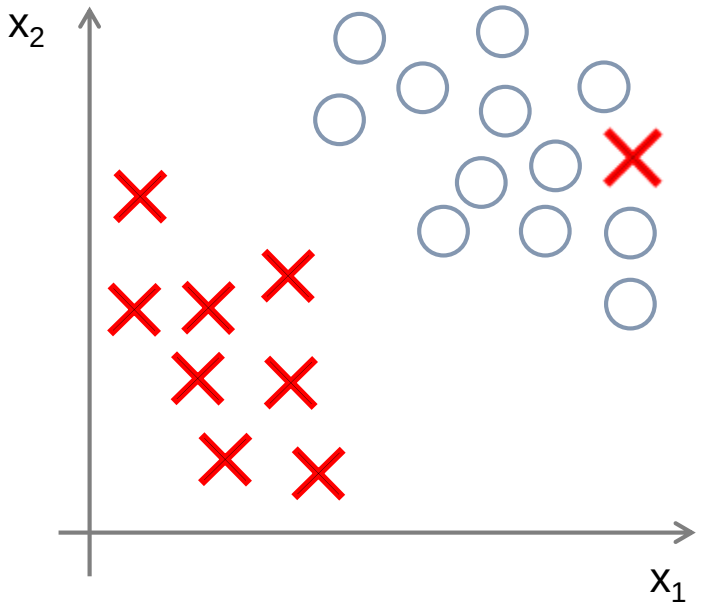
Change the original constraint from

$$y^{(i)}\left(\boldsymbol{\alpha}^T \boldsymbol{f}^{(i)} + b\right) \geq 1, \quad i = 1,2,\dots,m$$

to

$$y^{(i)}\left(\boldsymbol{\alpha}^T \boldsymbol{f}^{(i)} + b\right) \geq 1\text{-}\xi_i, \quad i = 1,2,\dots,m$$

Where $\xi_i \geq 0$, called Slack Variable



If ξ_i is a large number, then any hyperplane will meet the requirement, so we need to change the optimization objective from

$$\min \frac{1}{2}||\boldsymbol{\alpha}||^2$$

to

$$\min \frac{1}{2}||\boldsymbol{\alpha}||^2 + \mathrm{C}\sum_{i=1}^{n}\xi_i$$

C > 0 is the penalty parameter of the error term

[1] 支持向量机通俗导论, http://blog.csdn.net/v_july_v/article/details/7624837

Supervised Learning

2.1 Support Vector Machine: Problem Statement

- Given a training set of instance-label pairs $(\boldsymbol{x}^{(i)}, y^{(i)})$, $i = 1, \ldots, \mathrm{m}$, where $\boldsymbol{x}^{(i)} \in R^n$ and $y^{(i)} \in \{1, -1\}$
- For $\boldsymbol{x}^{(i)}$, Calculate the m-dimensional $\boldsymbol{f}^{(i)}$**:**

$$f_1{}^{(i)} = K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(1)}), f_2{}^{(i)} = K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(2)}), \ldots, f_m{}^{(i)} = K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(m)})$$

- the SVM require the solution of the following optimization problem:

$$\min_{\boldsymbol{\alpha},\mathrm{b},\xi} \frac{1}{2}||\boldsymbol{\alpha}||^2 + \mathrm{C}\sum_{\mathrm{i}=1}^{\mathrm{m}} \xi_i$$

$$\text{subject to } y^{(i)}\left(\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{f}^{(i)} + \mathrm{b}\right) \geq 1\text{-}\xi_i, \text{ where i} = 1,2, \ldots, \mathrm{m}$$

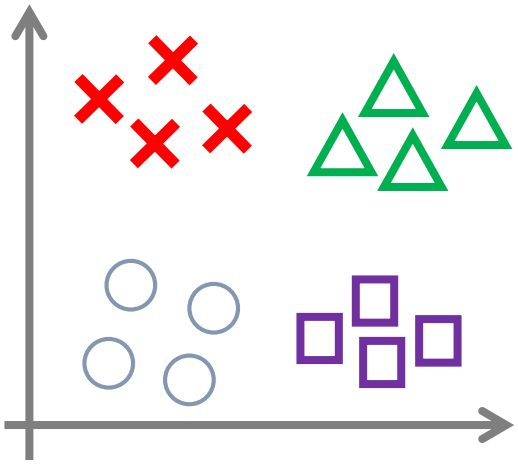
$$\text{and } \xi_i \geq 0$$

- Need to specify choice of penalty parameter C and choice of kernel before optimization.
- SVM finds a linear separating hyperplane with the maximal margin in this higher dimensional space.

[1] A Practical Guide to Support Vector Classification, http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

Supervised Learning

2.1 Support Vector Machine: Multiclass Classification



$$y \in \{1, 2, 3, \ldots, K\}$$

Use one-vs.-all method. (Train K SVMs, one to distinguish $y = i$ from the rest, for $i = 1,2,..K$)

[1] Machine Learning, Andrew Ng, https://class.coursera.org/ml-005/lecture

Compare SVM with NN

Similarity

- SVM + sigmoid kernel ~ two-layer feedforward NN
- SVM + Gaussian kernel ~ RBF network
- For most problems, SVM and NN have similar performance

[1] https://goo.gl/UBi0O8

Supervised Learning

2.1 Support Vector Machine: Advantages and Disadvantages

Advantages

- prediction accuracy is generally high
- robust works when training examples contain errors robust, works when training examples contain errors
- fast evaluation of the learned target function

Disadvantages

- long training time
- It only considers two classes

[1] *https://goo.gl/UBi0O8*

Supervised Learning

2.1 Support Vector Machine: More Topics

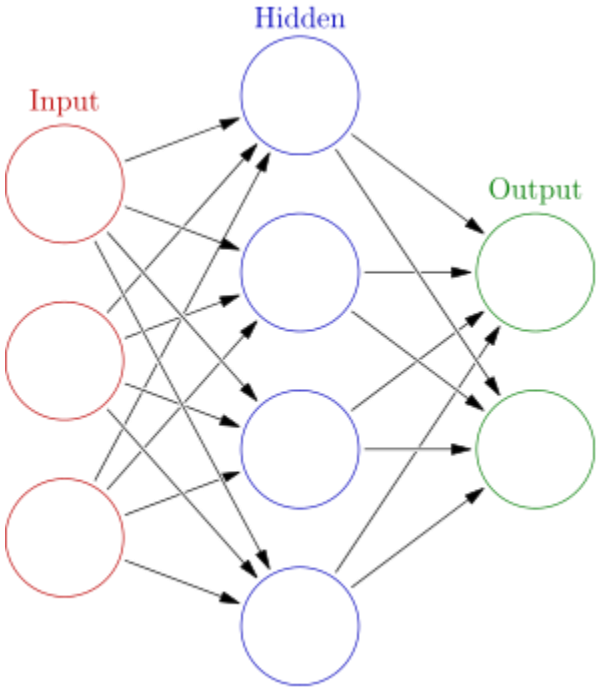
- Regression, clustering, semi-supervised learning and other domains.
- Lots of other kernels, e.g. string kernels to handle text.
- Lots of research in modifications, e.g. to improve generalization ability, or tailoring to a particular task.
- Lots of research in speeding up training.

[1] *Support Vector Machine (and Statistical Learning Theory) Tutorial, NEC Labs of Columbia University*

Supervised Learning

2.2 Neural Networks: Introduction

- An artificial neutral network (ANN) is a system that is based on the biological neural network.
- An ANN is comprised of a network of artificial neurons (also known as "nodes"). These nodes are connected to each other, and the strength of their connections to one another is assigned a value based on their strength: inhibition or excitation. If the value of the connection is high, then it indicates that there is a strong connection.
- Within each node's design, a transfer function is built in.
- There are three types of neurons in an ANN, input nodes, hidden nodes, and output nodes.

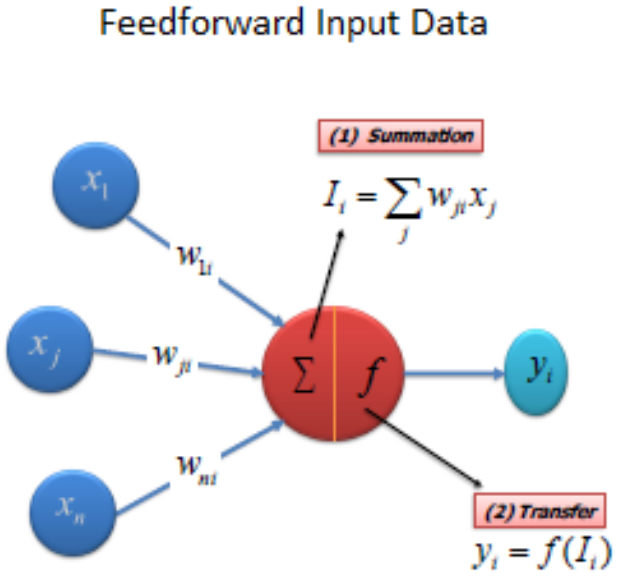


[1] http://www.saedsayad.com/artificial_neural_network.htm

Supervised Learning

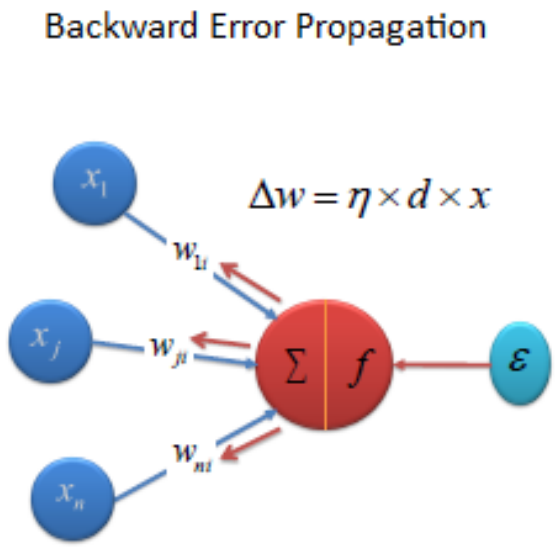
2.2 Neural Networks: Introduction

- The input nodes take in information, in the form which can be numerically expressed. The information is presented as activation values, where each node is given a number.
- This information is then passed throughout the network. Based on the connection strengths (weights), inhibition or excitation, and transfer functions, the activation value is passed from node to node. Each of the nodes sums the activation values it receives; it then modifies the value based on its transfer function.
- The activation flows through the network, through hidden layers, until it reaches the output nodes. The output nodes then reflect the input in a meaningful way to the outside world.



[1] http://www.saedsayad.com/artificial_neural_network.htm

- The difference between predicted value and actual value (error) will be propagated backward by apportioning them to each node's weights according to the amount of this error the node is responsible for (e.g., gradient descent algorithm).



[1] http://www.saedsayad.com/artificial_neural_network.htm

Supervised Learning

2.2 Neural Networks: History

First Attempts

- McCulloch and Pitts (1943) developed models of neural networks based on their understanding of neurology.

Promising & Emerging Technology

- Rosenblatt (1958) stirred considerable interest and activity in the field when he designed and developed the Perceptron.
- Another system was the ADALINE (ADAptive LInear Element) which was developed in 1960 by Widrow and Hoff (of Stanford University). The ADALINE was an analogue electronic device made from simple components.

Period of Frustration & Disrepute

- Neural network research stagnated after the publication of machine learning research by Marvin Minsky and Seymour Papert (1969), who discovered two key issues with the computational machines that processed neural networks.

[1] *https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#Appendix A - Historical background in detail*
[2] *https://en.wikipedia.org/wiki/Artificial_neural_network#History*

Resurgence

- A key advance that came later was the backpropagation algorithm which effectively solved the exclusive-or problem, and more generally the problem of quickly training multi-layer neural networks (Werbos 1975).
- Support vector machines and other, much simpler methods such as linear classifiers gradually overtook neural networks in machine learning popularity.
- As computing power increased through the use of GPUs and distributed computing, image and visual recognition problems came to the forefront, neural networks were deployed again, on larger scales. This became called "deep learning" which is simply a re-branding of neural networks, though emphasizing the use of modern parallel hardware implementations.

[1] https://en.wikipedia.org/wiki/Artificial_neural_network#History

Improvements since 2006

- Computational devices have been created in CMOS, for both biophysical simulation and neuromorphic computing.
- Between 2009 and 2012, the recurrent neural networks and deep feedforward neural networks developed in the research group of Jürgen Schmidhuber at the Swiss AI Lab IDSIA have won eight international competitions in pattern recognition and machine learning
- Fast GPU-based implementations of this approach by Dan Ciresan and colleagues at IDSIA have won several pattern recognition contests.

[1] *https://en.wikipedia.org/wiki/Artificial_neural_network#History*

Supervised Learning

2.2 Neural Networks: Algorithms

There are different types of neural networks, but they are generally classified into feed-forward and feed-back networks.

A feed-forward network

- a non-recurrent network which contains inputs, outputs, and hidden layers; the signals can only travel in one direction.
- Input data is passed onto a layer of processing elements where it performs calculations. Each processing element makes its computation based upon a weighted sum of its inputs. The new calculated values then become the new input values that feed the next layer. This process continues until it has gone through all the layers and determines the output.
- A threshold transfer function is sometimes used to quantify the output of a neuron in the output layer.
- Feed-forward networks include Perceptron (linear and non-linear) and Radial Basis Function networks.
- Feed-forward networks are often used in data mining.
- Common Examples are Perceptrons, Back Propagation Network, and Radial Basis Function Network.

[1] *http://www.saedsayad.com/artificial_neural_network.htm*
[2] *http://baike.baidu.com/view/1986922.htm*

There are different types of neural networks, but they are generally classified into feed-forward and feed-back networks.

A feed-back network

- they can have signals traveling in both directions using loops.
- All possible connections between neurons are allowed.
- Since loops are present in this type of network, it becomes a non-linear dynamic system which changes continuously until it reaches a state of equilibrium.
- Feed-back networks are often used in associative memories and optimization problems where the network looks for the best arrangement of interconnected factors.
- Common Examples are Boltzmann machine and Hopfield network.

[1] *http://www.saedsayad.com/artificial_neural_network.htm*
[2] *http://baike.baidu.com/view/1986922.htm*

Supervised Learning

2.2 Neural Networks: Backpropagation Neural Networks

The backpropagation learning algorithm can be divided into two phases: propagation and weight update.

- Phase 1: Propagation

(1) Forward propagation of a training pattern's input through the neural network in order to generate the propagation's output activations.

(2) Backward propagation of the propagation's output activations through the neural network using the training pattern target in order to generate the deltas (the difference between the targeted and actual output values) of all output and hidden neurons.

[1] *https://en.wikipedia.org/wiki/Backpropagation*

Supervised Learning

2.2 Neural Networks: Backpropagation Neural Networks

The backpropagation learning algorithm can be divided into two phases: propagation and weight update.

- Phase 2: Weight update

(1) Multiply its output delta and input activation to get the gradient of the weight.

(2) Subtract a ratio (percentage) of the gradient from the weight.

Repeat phase 1 and 2 until the performance of the network is satisfactory.

This ratio (percentage) influences the speed and quality of learning; it is called the learning rate. The greater the ratio, the faster the neuron trains; the lower the ratio, the more accurate the training is. The sign of the gradient of a weight indicates where the error is increasing, this is why the weight must be updated in the opposite direction.

[1] *https://en.wikipedia.org/wiki/Backpropagation*

Algorithm

initialize network weights (often small random values)

do

forEach training example named ex

prediction = neural-net-output(network, ex) // forward pass

actual = teacher-output(ex)

compute error (prediction - actual) at the output units

compute Δw_h for all weights from hidden layer to output layer // backward pass

compute Δw_i for all weights from input layer to hidden layer // backward pass continued

update network weights // input layer not modified by error estimate

until all examples classified correctly or another stopping criterion satisfied

return the network

[1] *https://en.wikipedia.org/wiki/Backpropagation*

Derivation(1/7)

Assuming one output neuron, the squared error function is:

$$E = \frac{1}{2}(t - y)^2$$

Where E is the squared error, t is the target output for a training sample, and y is the actual output of the output neuron. The factor of $\frac{1}{2}$ is included to cancel the exponent when differentiating.

[1] https://en.wikipedia.org/wiki/Backpropagation

Derivation(2/7)

For each neuron a$j(a^{th}\ layer, j^{th}\ neuron)$, its output o_{aj} is defined as

$$o_{aj} = \varphi(net_{aj}) = \varphi(\sum_{k=1}^{n} w_{(a-1)kaj} o_{(a-1)k})$$

The input net_{aj} to a neuron is the weighted sum of outputs $o_{(a-1)k}$ of previous neurons. If the neuron is in the first layer after the input layer, the $o_{(a-1)k}$ of the input layer are simply the inputs x_k to the network. The variable $w_{(a-1)iaj}$ denotes the weight between neurons $i(a-1^{th}\ layer)$ and $j(a^{th}\ layer)$.

[1] *https://en.wikipedia.org/wiki/Backpropagation*

Derivation(3/7)

The activation function φ is in general non-linear and differentiable. A commonly used activation function is the **logistic function**:

$$\varphi(z) = \frac{1}{1 + e^{-z}}$$

which has a nice derivative of:

$$\frac{d\varphi}{dz}(z) = \varphi(z)(1 - \varphi(z))$$

Calculating the partial derivative of the error with respect to a weight w_{ij} is done using the chain rule twice:

$$\frac{\partial E}{\partial w_{(a-1)iaj}} = \frac{\partial E}{\partial o_{aj}} \frac{\partial o_{aj}}{\partial net_{aj}} \frac{\partial net_{aj}}{\partial w_{(a-1)iaj}}$$

[1] https://en.wikipedia.org/wiki/Backpropagation

Derivation(4/7)

$$\frac{\partial E}{\partial w_{(a-1)\mathrm{i}aj}} = \frac{\partial E}{\partial o_{aj}} \frac{\partial o_{aj}}{\partial net_{aj}} \frac{\partial net_{aj}}{\partial w_{(a-1)iaj}}$$

(1)

$$\frac{\partial net_{aj}}{\partial w_{(a-1)iaj}} = \frac{\partial}{\partial w_{(a-1)iaj}} \left(\sum_{k=1}^{n} w_{(a-1)kaj} o_{(a-1)k} \right) = o_{(a-1)i}$$

(2)

$$\frac{\partial \mathrm{o}_{aj}}{\partial \mathrm{net}_{aj}} = \frac{\partial}{\partial net_{aj}} \varphi\left(net_{aj}\right) = \varphi\left(net_{aj}\right)\left(1 - \varphi\left(net_{aj}\right)\right)$$

This is the reason why backpropagation requires the activation function to be differentiable.

[1] https://en.wikipedia.org/wiki/Backpropagation

Derivation(5/7)

$$\frac{\partial E}{\partial w_{(a-1)iaj}} = \frac{\partial E}{\partial o_{aj}} \frac{\partial o_{aj}}{\partial net_{aj}} \frac{\partial net_{aj}}{\partial w_{(a-1)iaj}}$$

(3)

if the neuron o_{aj} is in the output layer,

$$\frac{\partial E}{\partial o_{aj}} = \frac{\partial E}{\partial y} = \frac{\partial}{\partial y} \frac{1}{2} (t - y)^2 = y - t$$

if o_{aj} is in the inner layer of the network,

$$\frac{\partial E}{\partial o_{aj}} = \sum_{l \in L} \left(\frac{\partial E}{\partial net_{(a+1)l}} \frac{\partial net_{(a+1)l}}{\partial o_{aj}} \right) = \sum_{l \in L} \left(\frac{\partial E}{\partial o_{(a+1)l}} \frac{\partial o_l}{\partial net_{(a+1)l}} w_{aj(a+1)l} \right)$$

[1] https://en.wikipedia.org/wiki/Backpropagation

Derivation(6/7)

Overall,

$$\frac{\partial E}{\partial w_{(a-1)iaj}} = \delta_{aj} o_{(a-1)i}$$

With

$$\delta_{aj} = \frac{\partial E}{\partial o_{aj}} \frac{\partial o_{aj}}{\partial net_{aj}} = \begin{cases} (o_{aj} - t_{aj}) o_{aj} (1 - o_{aj}) & if\ j\ is\ an\ output\ neuron \\ \left(\sum_{l \in L} \delta_{(a+1)l}\, w_{aj(a+1)l} \right) o_{aj} (1 - o_{aj}) & if\ j\ is\ an\ inner\ neuron \end{cases}$$

[1] https://en.wikipedia.org/wiki/Backpropagation

Derivation(7/7)

To update the weight $w_{(a-1)iaj}$ using gradient descent, one must choose a learning rate, α. The change in weight, which is added to the old weight, is equal to the product of the learning rate and the gradient, multiplied by -1:

$$\Delta w_{(a-1)iaj} = -\alpha \frac{\partial E}{\partial w_{(a-1)iaj}}$$

The -1 is required in order to update in the direction of a minimum, not a maximum, of the error function.

[1] https://en.wikipedia.org/wiki/Backpropagation

Limitation

Gradient descent with backpropagation is not guaranteed to find the global minimum of the error function, but only a local minimum

[1] *https://en.wikipedia.org/wiki/Backpropagation*

Example(1/3)

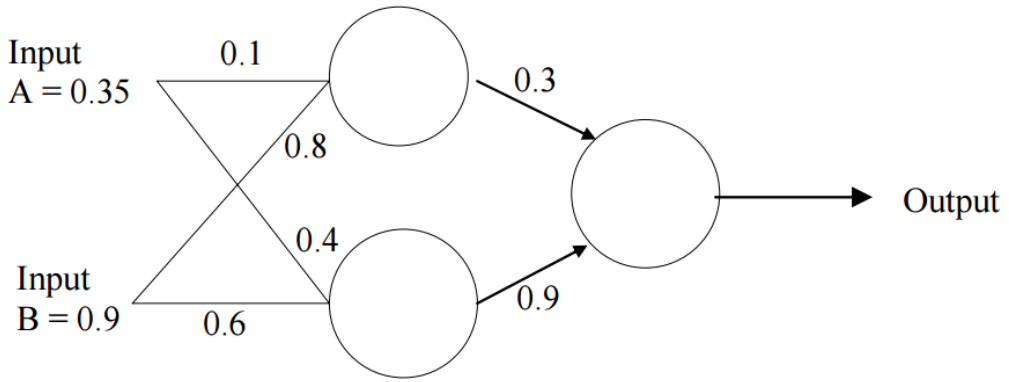
Consider the simple network. Assume that the neurons have a Sigmoid activation function

(1) Perform a forward pass on the network

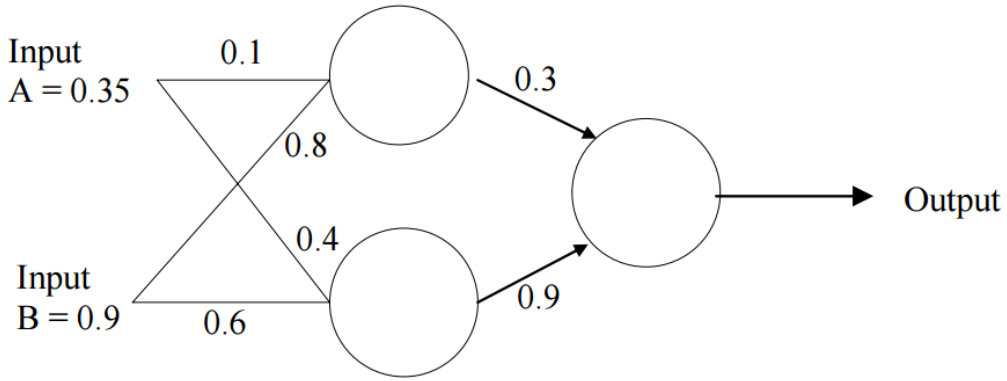
Input to top neuron $= (0.35 * 0.1) + (0.9 * 0.8) = 0.755. Out = 0.68.$

Input to bottom neuron $= (0.9 * 0.6) + (0.35 * 0.4) = 0.68. Out = 0.6637.$

Input to final neuron $= (0.3 * 0.68) + (0.9 * 0.6637) = 0.80133. Out = 0.69.$



[1] https://www.fer.unizg.hr/_download/repository/BP_chapter3_-_bp.pdf



Example(2/3)

(2) Perform a reverse pass (training) once (target = 0.5)

Output error: $\delta = (t - o)(1 - o)o = (0.5 - 0.69) \cdot (1 - 0.69) \cdot 0.69 = -0.0406$

New weights for output layer

$w_1^+ = w_1 + (\delta \cdot input) = 0.3 + (-0.0406 \cdot 0.68) = 0.272392$, $w_2^+ = w_2 + (\delta \cdot input) = 0.9 + (-0.0406 \cdot 0.6637) = 0.87305$

Errors for hidden layers:

$\delta_1 = \delta \cdot w_1 \cdot o \cdot (1 - o) = -0.0406 \cdot 0.272392 \cdot o \cdot (1 - o) = -2.406 \cdot 10^{-3}$, $\delta_2 = \delta \cdot w_2 \cdot o \cdot (1 - o) = -0.0406 \cdot 0.87305 \cdot o \cdot (1 - o) = -7.916 \cdot 10^{-3}$

New hidden layer weights:

$w_3^+ = 0.1 + (-2.406 \cdot 10^{-3} \cdot 0.35) = 0.09916$, $w_4^+ = 0.8 + (-2.406 \cdot 10^{-3} \cdot 0.9) = 0.7978$,

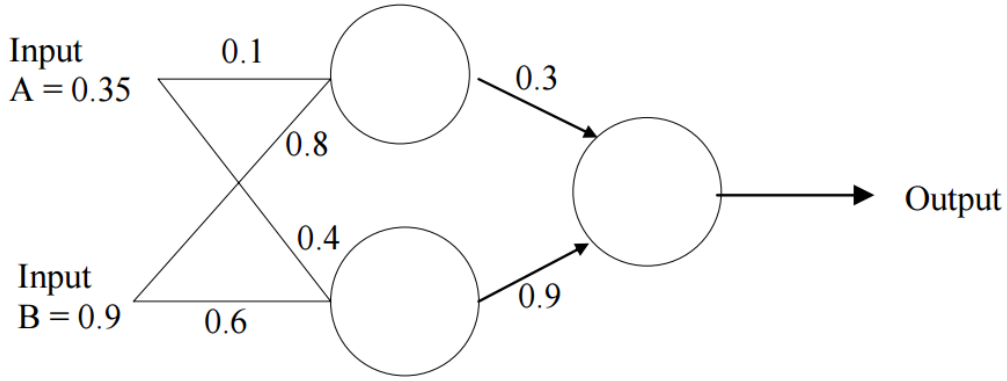
$w_5^+ = 0.4 + (-7.916 \cdot 10^{-3} \cdot 0.35) = 0.3972$, $w_6^+ = 0.6 + (-7.916 \cdot 10^{-3} \cdot 0.9) = 0.5928$c

[1] https://www.fer.unizg.hr/_download/repository/BP_chapter3_-_bp.pdf

Example(3/3)

(3) Perform a further forward pass and comment on the result

Old error was -0.19. New error is -0.18205. Therefore error has reduced.



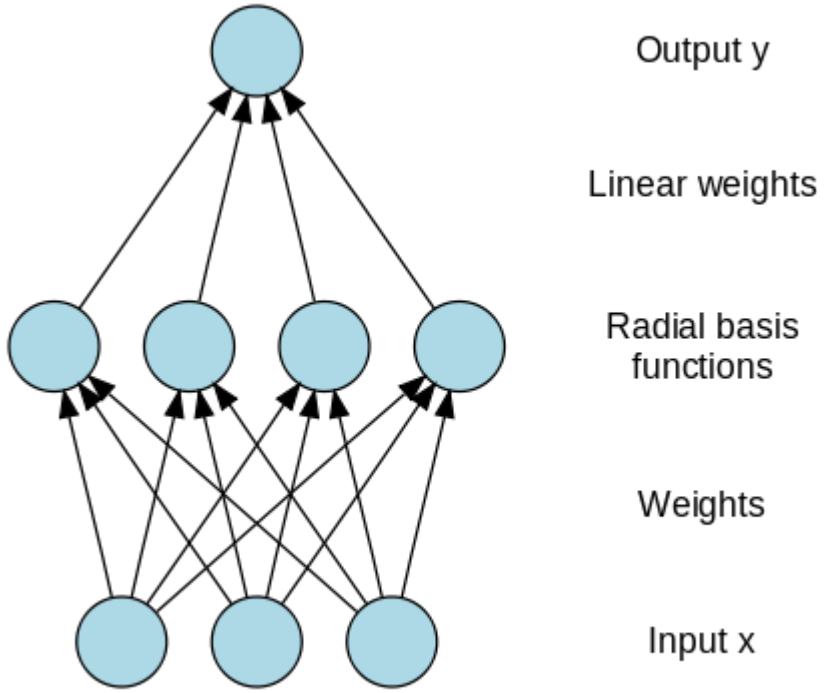
[1] https://www.fer.unizg.hr/_download/repository/BP_chapter3_-_bp.pdf

Supervised Learning

2.2 Neural Networks: Radial Basis Neural Networks

Introduction

- A radial basis function network is an artificial neural network that uses radial basis functions as activation functions. The output of the network is a linear combination of radial basis functions of the inputs and neuron parameters.
- Radial basis function (RBF) networks typically have three layers: an input layer, a hidden layer with a non-linear RBF activation function and a linear output layer.



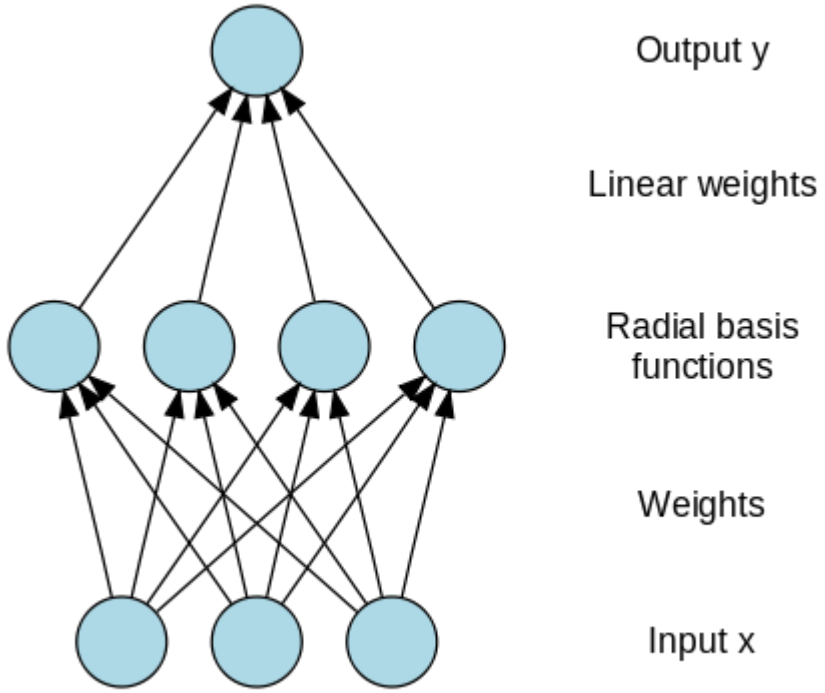
[1] https://en.wikipedia.org/wiki/Radial_basis_function_network

Supervised Learning

2.2 Neural Networks: Radial Basis Neural Networks

Training

- RBF networks are typically trained by a two-step algorithm. In the first step, the center vectors of the RBF functions in the hidden layer are chosen. This step can be performed in several ways; centers can be randomly sampled from some set of examples, or they can be determined using k-means clustering. Note that this step is unsupervised.
- The second step simply fits a linear model with coefficients to the hidden layer's outputs with respect to some objective function.



[1] https://en.wikipedia.org/wiki/Radial_basis_function_network

Supervised Learning

2.2 Neural Networks: Advantages and Limitations

Advantages

- Can be applied to many problems, as long as there is some data.
- Can be applied to problems, for which analytical methods do not yet exist
- Can be used to model non-linear dependencies.
- If there is a pattern, then neural networks should quickly work it out, even if the data is ‘noisy’.
- Always gives some answer even when the input information is not complete.
- Networks are easy to maintain.

[1] *http://www.eis.mdx.ac.uk/staffpages/rvb/teaching/BIS3226/hand11.pdf*

Limitations

- Like with any data-driven models, they cannot be used if there is no or very little data available.
- There are many free parameters, such as the number of hidden nodes, the learning rate, minimal error, which may greatly influence the final result.
- Not good for arithmetic and precise calculations.
- Neural networks do not provide explanations. If there are many nodes, then there are too many weights that are difficult to interpret (unlike the slopes in linear models, which can be seen as correlations). In some tasks, explanations are crucial (e.g. air traffic control, medical diagnosis).

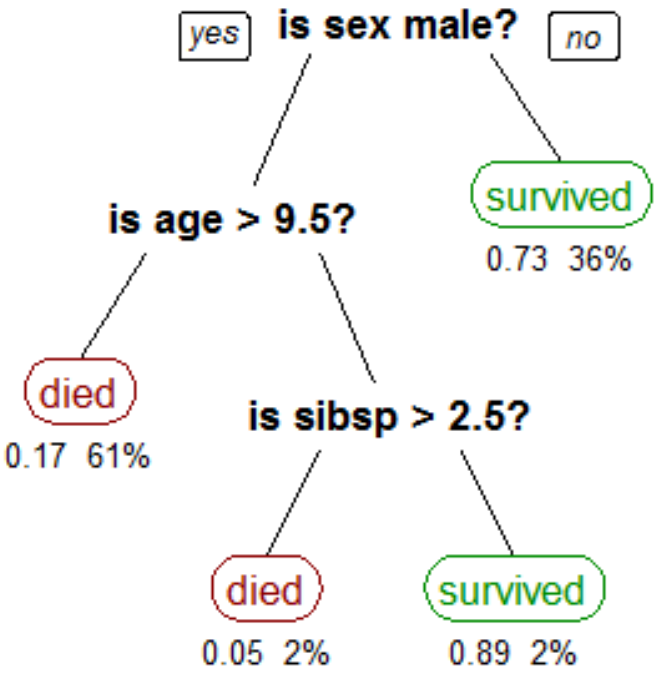
[1] *http://www.eis.mdx.ac.uk/staffpages/rvb/teaching/BIS3226/hand11.pdf*

Supervised Learning

2.3 Decision Tree: Introduction

Decision tree

- Decision tree builds classification or regression models in the form of **a tree structure**.
- It breaks down a dataset into smaller and smaller **subsets** while at the same time an associated decision tree is incrementally developed.
- The final result is **a tree with decision nodes and leaf nodes**.



A tree showing survival of passengers on the Titanic ("sibsp" is the number of spouses or siblings aboard). The figures under the leaves show the probability of survival and the percentage of observations in the leaf.

Decision tree learning

It is the **construction** of a decision tree from **class-labeled** training tuples.

[1] Wikipedia, https://en.wikipedia.org/wiki/Decision_tree_learning
[2] http://www.saedsayad.com/decision_tree.htm

Supervised Learning

2.3 Decision Tree: History

- Hunt and colleagues use **exhaustive search decision-tree methods** (CLS) to model human concept learning in the 1960's.
- In the late 70's, Quinlan developed **ID3** with the information gain heuristic to learn expert systems from examples.
- Simultaneously, Breiman and Friedman and colleagues develop **CART (Classification and Regression Trees)**, similar to ID3.
- In the 1980's a variety of improvements are introduced to **handle noise**, **continuous features**, **missing features**, and **improved splitting criteria**. Various expert-system development tools results.
- Quinlan's updated decision-tree package (**C4.5**) released in 1993.

[1] University of Texas at Austin, CS 391L slides, https://www.cs.utexas.edu/~mooney/cs391L/slides/dtrees.pdf

There are many specific decision-tree algorithms. Notable ones include:

- ID3 (Iterative Dichotomiser 3)
- C4.5 (successor of ID3)
- CART (Classification And Regression Tree)
- CHAID (CHi-squared Automatic Interaction Detector). Performs multi-level splits when computing classification trees.
- MARS: extends decision trees to handle numerical data better.
- Conditional Inference Trees. Statistics-based approach that uses non-parametric tests as splitting criteria, corrected for multiple testing to avoid overfitting. This approach results in unbiased predictor selection and does not require pruning.

[1] *Wikipedia, https://en.wikipedia.org/wiki/Decision_tree_learning*

Supervised Learning

2.3 Decision Tree: ID3 Algorithm(Classification)

ID3 by J. R. Quinlan

- employs a **top-down**, **greedy search** through the space of possible branches with no backtracking.
- uses **Entropy and Information Gain** to construct a decision tree.

ID3 Algorithm Procedure for Classification

Step 1: Calculate entropy of the target.

Step 2: The dataset is then split on the different attributes. The entropy for each branch is calculated. Then it is added proportionally, to get total entropy for the split. The resulting entropy is subtracted from the entropy before the split. The result is the Information Gain, or decrease in entropy.

Look at an example first

Step 3: Choose attribute with the largest information gain as the decision node.

Step 4a: A branch with entropy of 0 is a leaf node.

Step 4b: A branch with entropy more than 0 needs further splitting.

Step 5: The ID3 algorithm is run recursively on the non-leaf branches, until all data is classified.

[1] http://www.saedsayad.com/decision_tree.htm

Supervised Learning

2.3 Decision Tree: ID3 Algorithm(Classification)

Step 0: Example Data

Predictors | Target

Outlook	Temp.	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

Step 1: Calculate entropy of the target.

Play Golf	
Yes	No
9	5

Entropy(PlayGolf) = Entropy (5,9)
= Entropy (0.36, 0.64)
= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)
= 0.94

[1] http://www.saedsayad.com/decision_tree.htm

Supervised Learning

2.3 Decision Tree: ID3 Algorithm(Classification)

Step 2: The dataset is then **split on the different attributes**. The entropy for each branch is calculated. Then it is added proportionally, to get total entropy for the split. The resulting entropy is subtracted from the entropy before the split. The result is the **Information Gain**, or decrease in entropy.

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14

E(PlayGolf, Outlook) = **P**(Sunny)***E**(3,2) + **P**(Overcast)***E**(4,0) + **P**(Rainy)***E**(2,3)

= (5/14)*0.971 + (4/14)*0.0 + (5/14)*0.971

= 0.693

$$Gain(T,X) = Entropy(T) - Entropy(T,X)$$

G(PlayGolf, Outlook) = **E**(PlayGolf) – **E**(PlayGolf, Outlook)

= 0.940 – 0.693 = 0.247

Predictors | Target

Outlook	Temp.	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1
Gain = 0.029			

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1
Gain = 0.152			

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3
Gain = 0.048			

[1] http://www.saedsayad.com/decision_tree.htm

Supervised Learning

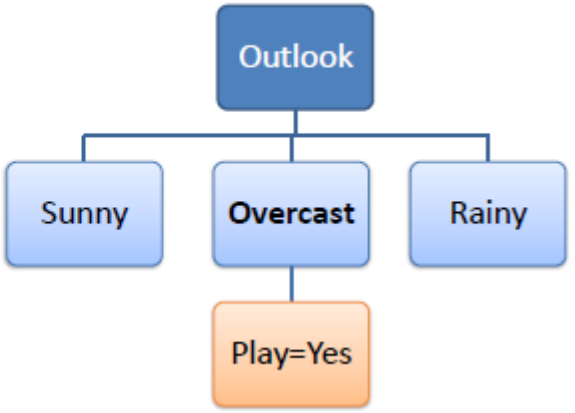
2.3 Decision Tree: ID3 Algorithm(Classification)

Step 3: Choose attribute with **the largest information gain** as the decision node.

★		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

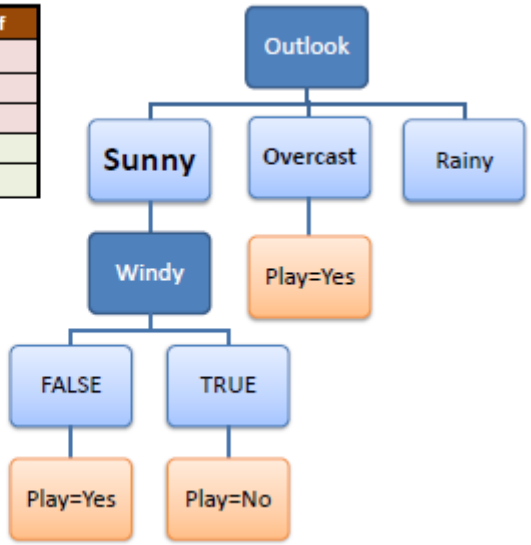
Step 4a: A branch with entropy of 0 is a **leaf node**.

Temp	Humidity	Windy	Play Golf
Hot	High	FALSE	Yes
Cool	Normal	TRUE	Yes
Mild	High	TRUE	Yes
Hot	Normal	FALSE	Yes
Hot	High	FALSE	Yes



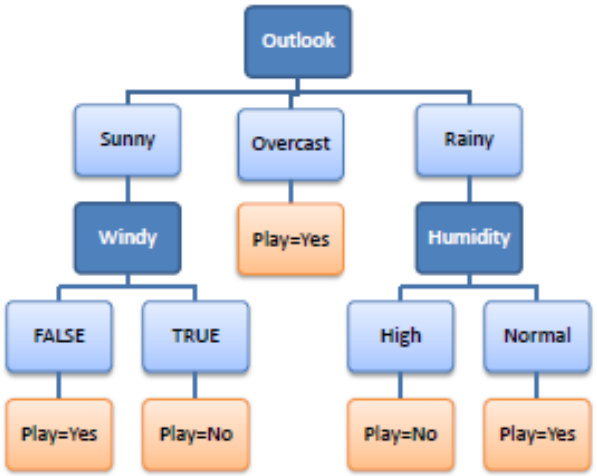
Step 4b: A branch with entropy more than 0 needs **further splitting**.

Temp	Humidity	Windy	Play Golf
Mild	High	FALSE	Yes
Cool	Normal	FALSE	Yes
Mild	Normal	FALSE	Yes
Cool	Normal	TRUE	No
Mild	High	TRUE	No



[1] http://www.saedsayad.com/decision_tree.htm

Step 5: The ID3 algorithm is **run recursively** on the non-leaf branches, until all data is classified.



[1] http://www.saedsayad.com/decision_tree.htm

Supervised Learning

2.3 Decision Tree: ID3 Algorithm(Classification)

ID3 Algorithm Procedure for Classification

Step 1: Calculate **entropy** of the target.

Step 2: The dataset is then **split** on the different attributes. The entropy for each branch is calculated. Then it is added proportionally, to get total entropy for the split. The resulting entropy is subtracted from the entropy before the split. The result is the **Information Gain**, or decrease in entropy.

Step 3: Choose attribute with the **largest information gain** as the decision node.

Step 4a: A branch with entropy of 0 is a **leaf node**.

Step 4b: A branch with entropy more than 0 needs **further splitting**.

Step 5: The ID3 algorithm is **run recursively** on the non-leaf branches, until all data is classified.

[1] *http://www.saedsayad.com/decision_tree.htm*

Supervised Learning

2.3 Decision Tree: ID3 Algorithm(Regression)

The ID3 algorithm can be used to construct a decision tree for regression by replacing Information Gain with **Standard Deviation Reduction**.

ID3 Algorithm Procedure for Regression

Step 1: The standard deviation of the target is calculated.

Step 2: The dataset is then split on the different attributes. The standard deviation for each branch is calculated. The resulting standard deviation is subtracted from the standard deviation before the split. The result is the standard deviation reduction.

Step 3: The attribute with the largest standard deviation reduction is chosen for the decision node.

Look at an example first

Step 4a: Dataset is divided based on the values of the selected attribute.

Step 4b: A branch set with standard deviation more than 0 needs further splitting. In practice, we need some termination criteria. For example, when standard deviation for the branch becomes smaller than a certain fraction (e.g., 5%) of standard deviation for the full dataset OR when too few instances remain in the branch (e.g., 3).

Step 5: The process is run recursively on the non-leaf branches, until all data is processed. When the number of instances is more than one at a leaf node we calculate the average as the final value for the target.

[1] http://www.saedsayad.com/decision_tree.htm

Supervised Learning

2.3 Decision Tree: ID3 Algorithm(Regression)

Step 0: Example Data

Predictors				Target
Outlook	Temp.	Humidity	Windy	Hours Played
Rainy	Hot	High	False	25
Rainy	Hot	High	True	30
Overcast	Hot	High	False	46
Sunny	Mild	High	False	45
Sunny	Cool	Normal	False	52
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	35
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	46
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	52
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30

Step 1: The standard deviation of the target is calculated.

Hours Played
25
30
46
45
52
23
43
35
38
46
48
52
44
30

$$S = \sqrt{\frac{\sum (x-\mu)^2}{n}}$$

Standard Deviation

S = 9.32

[1] http://www.saedsayad.com/decision_tree.htm

Supervised Learning

2.3 Decision Tree: ID3 Algorithm(Regression)

Step 2: The dataset is then split on the different attributes.

$$S(T,X) = \sum_{c \in X} P(c)S(c)$$

		Hours Played (StDev)	Count
Outlook	Overcast	3.49	4
	Rainy	7.78	5
	Sunny	10.87	5
			14

S(Hours, Outlook) = **P**(Sunny)***S**(Sunny) + **P**(Overcast)***S**(Overcast) + **P**(Rainy)***S**(Rainy)

= (4/14)*3.49 + (5/14)*7.78 + (5/14)*10.87

= 7.66

$$SDR(T,X) = S(T) - S(T,X)$$

SDR(Hours , Outlook) = **S**(Hours) – **S**(Hours, Outlook)

= 9.32 – 7.66 = 1.66

		Hours Played (StDev)
Outlook	Overcast	3.49
	Rainy	7.78
	Sunny	10.87
SDR=1.66		

		Hours Played (StDev)
Temp.	Cool	10.51
	Hot	8.95
	Mild	7.65
SDR=0.17		

		Hours Played (StDev)
Humidity	High	9.36
	Normal	8.37
SDR=0.28		

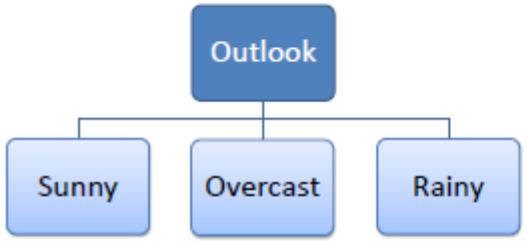
		Hours Played (StDev)
Windy	False	7.87
	True	10.59
SDR=0.29		

[1] http://www.saedsayad.com/decision_tree.htm

Step 3: The attribute with the largest standard deviation reduction is chosen for the decision node.

★		Hours Played (StDev)
Outlook	Overcast	3.49
	Rainy	7.78
	Sunny	10.87
SDR=1.66		

Step 4a: Dataset is divided based on the values of the selected attribute.



Outlook	Temp	Humidity	Windy	Hours Played
Sunny	Mild	High	FALSE	45
Sunny	Cool	Normal	FALSE	52
Sunny	Cool	Normal	TRUE	23
Sunny	Mild	Normal	FALSE	46
Sunny	Mild	High	TRUE	30
Rainy	Hot	High	FALSE	25
Rainy	Hot	High	TRUE	30
Rainy	Mild	High	FALSE	35
Rainy	Cool	Normal	FALSE	38
Rainy	Mild	Normal	TRUE	48
Overcast	Hot	High	FALSE	46
Overcast	Cool	Normal	TRUE	43
Overcast	Mild	High	TRUE	52
Overcast	Hot	Normal	FALSE	44

[1] http://www.saedsayad.com/decision_tree.htm

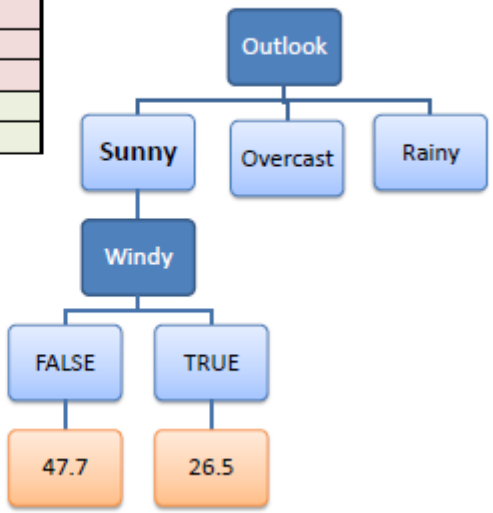
2.3 Decision Tree: ID3 Algorithm(Regression)

Step 4b: A branch set with standard deviation more than 0 needs further splitting.

Temp.	Humidity	Windy	Hours Played
Mild	High	FALSE	45
Cool	Normal	FALSE	52
Mild	Normal	FALSE	46
Cool	Normal	TRUE	23
Mild	High	TRUE	30

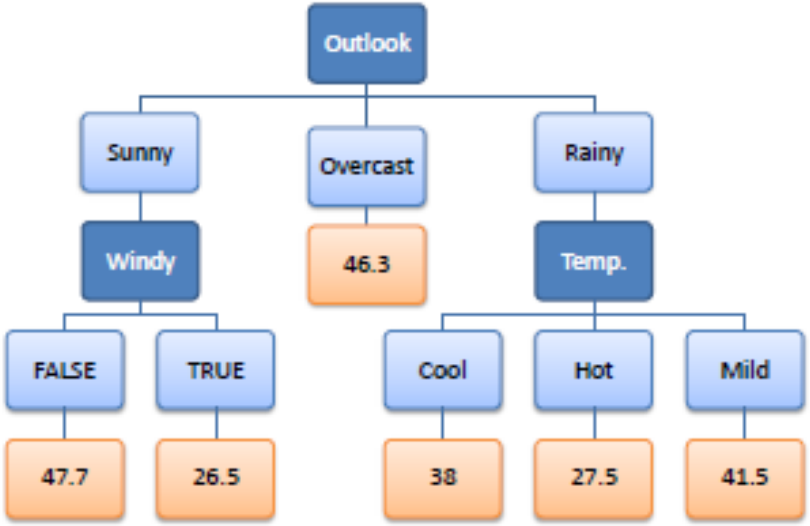
★		Hours Played (StDev)
Windy	False	3.09
	True	3.50
SDR= 7.62		

SDR = 10.87-((3/5)*3.09 + (2/5)*3.5)



[1] http://www.saedsayad.com/decision_tree.htm

Step 5: The process is run recursively on the non-leaf branches, until all data is processed.



[1] http://www.saedsayad.com/decision_tree.htm

Supervised Learning

2.3 Decision Tree: ID3 Algorithm(Regression)

ID3 Algorithm Procedure for Regression

Step 1: The **standard deviation** of the target is calculated.

Step 2: The dataset is then **split** on the different attributes. The standard deviation for each branch is calculated. The resulting standard deviation is subtracted from the standard deviation before the split. The result is the standard deviation reduction.

Step 3: The attribute with the **largest standard deviation reduction** is chosen for the decision node.

Step 4a: Dataset is **divided** based on the values of the selected attribute.

Step 4b: A branch set with standard deviation more than 0 needs **further splitting**. In practice, we need some termination criteria. For example, when standard deviation for the branch becomes smaller than a certain fraction (e.g., 5%) of standard deviation for the full dataset OR when too few instances remain in the branch (e.g., 3).

Step 5: The process is **run recursively** on the non-leaf branches, until all data is processed. When the number of instances is more than one at a leaf node we calculate the average as the final value for the target.

[1] http://www.saedsayad.com/decision_tree.htm

Overfitting is a significant practical difficulty for decision tree models

Several approaches:

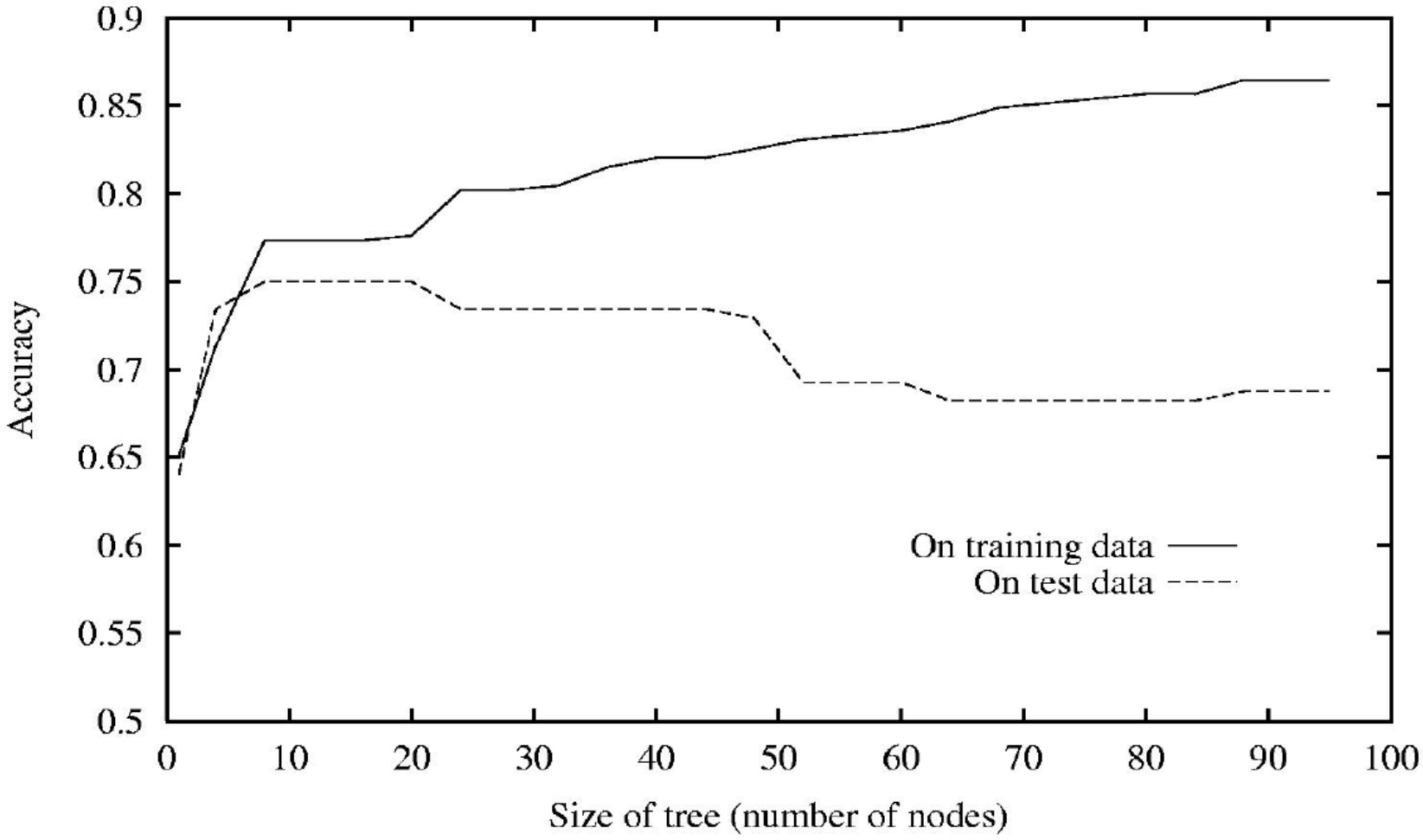
- **Pre-pruning** that stop growing the tree earlier, before it perfectly classifies the training set.
- **Post-pruning** that allows the tree to perfectly classify the training set, and then post prune the tree.

Practically, the second approach of **post-pruning** overfit trees is **more successful** because it is not easy to precisely estimate when to stop growing the tree.

[1] *http://www.saedsayad.com/decision_tree_overfitting.htm*

Supervised Learning

2.3 Decision Tree: Avoid Overfitting



[1] David Sontag, Slides, Decision Trees, http://cs.nyu.edu/~dsontag/courses/ml12/slides/lecture11.pdf

Supervised Learning

2.3 Decision Tree: Avoid Overfitting

The important step of tree pruning is to define a **criterion** be used to determine the correct final tree size:

- Use a distinct dataset from the training set (called **validation set**), to evaluate the effect of post-pruning nodes from the tree.
- Build the tree by using the training set, then apply a **statistical test** to estimate whether pruning or expanding a particular node is likely to produce an improvement beyond the training set.
  - Error estimation
  - Significance testing (e.g., Chi-square test)
- **Minimum Description Length principle** : Use an explicit measure of the complexity for encoding the training set and the decision tree, stopping growth of the tree when this cost **cost(tree) + cost(misclassifications(tree)** is minimized.

[1] http://www.saedsayad.com/decision_tree_overfitting.htm

Supervised Learning

2.3 Decision Tree: Advantages and Disadvantages

Advantages

- Requires **little data preparation**. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed.
- Able to handle both **numerical and categorical data**. Other techniques are usually specialized in analyzing datasets that have only one type of variable.
- Uses **a white box model**. If a given situation is observable in a model the explanation for the condition is easily explained by boolean logic.
- **Robust** to noisy data.
- Performs well with **large datasets**. Large amounts of data can be analyzed using standard computing resources in reasonable time.

[1] Wikipedia, https://en.wikipedia.org/wiki/Decision_tree_learning
[2] http://web.cs.hacettepe.edu.tr/~ilyas/Courses/BIL712/lec02-DecisionTree.pdf

Limitations(1/2)

- The problem of learning an optimal decision tree is known to be **NP-complete** under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristics such as the greedy algorithm where locally-optimal decisions are made at each node. Such algorithms **cannot guarantee to return the globally-optimal decision tree**. To reduce the greedy effect of local-optimality some methods such as the dual information distance (DID) tree were proposed.
- Decision-tree learners can create over-complex trees that do not generalize well from the training data. (This is known as **overfitting**.) Mechanisms such as pruning are necessary to avoid this problem (with the exception of some algorithms such as the Conditional Inference approach, that does not require pruning).

[1] Wikipedia, https://en.wikipedia.org/wiki/Decision_tree_learning

Limitations(2/2)

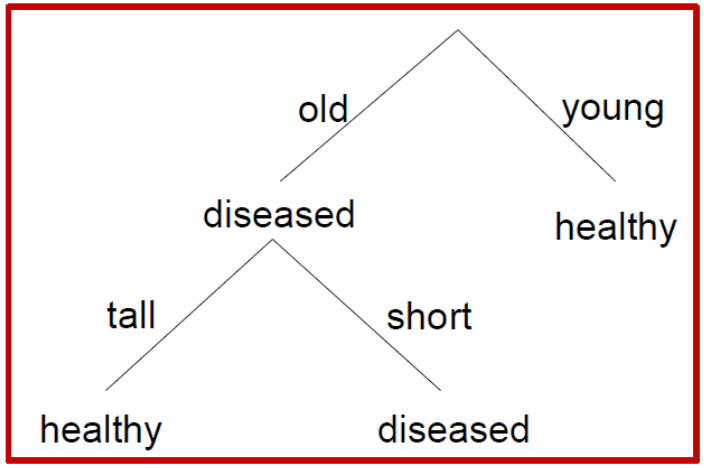
- There are **concepts that are hard to learn** because decision trees do not express them easily, such as XOR, parity or multiplexer problems. In such cases, the decision tree becomes prohibitively large. Approaches to solve the problem involve either changing the representation of the problem domain (known as propositionalisation) or using learning algorithms based on more expressive representations (such as statistical relational learning or inductive logic programming).

[1] Wikipedia, https://en.wikipedia.org/wiki/Decision_tree_learning

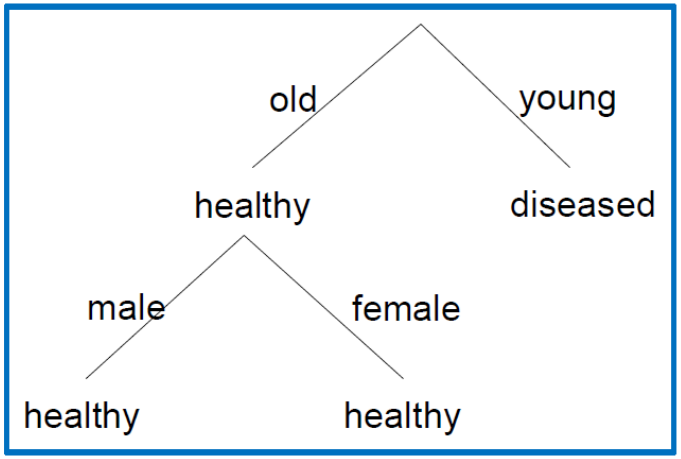
Supervised Learning

2.4 Random Forests: Intuition

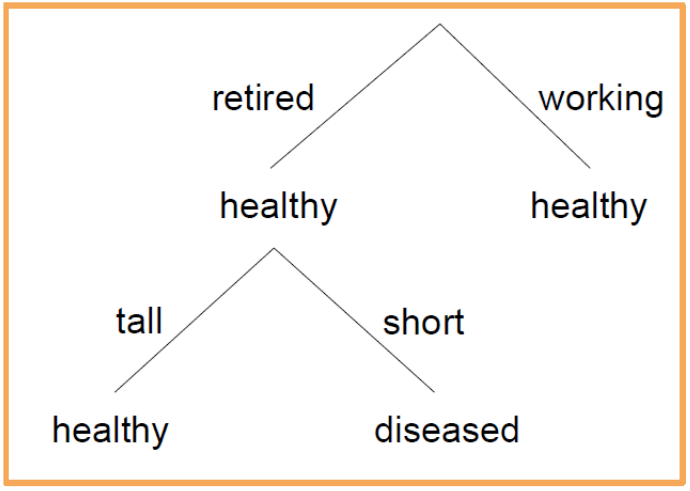
Tree 1



Tree 2



Tree 3



New sample:
old, retired, male, short

Tree predictions:
diseased, healthy, diseased

Majority rule:
diseased

[1] https://stat.ethz.ch/education/semesters/ss2012/ams/slides/v10.2.pdf

Supervised Learning

2.4 Random Forests: Algorithm

Bootstrap resample of data set with N samples:
Make new data set by drawing with replacement N samples; i.e., some samples will probably occur multiple times in new data set

1. For b=1 to B
   - **Draw a bootstrap sample** Z^* of size N from the training data.
   - **Grow a random-forest tree** T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until **the minimum node size** n_{min} is reached.
     - **Select m variables** at random from the p variables.
     - Pick the best variable/Split-point among the m
     - Split the node into two daughter nodes.
2. Output the **ensemble of trees** $\{T_b\}_1^B$.
3. To **make a prediction** at a new point x:
   - Regression: $\hat{f}_{rf}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$
   - Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{rf}^B(x) = mojority\ vote\ \{\hat{C}_b(x)\}_1^B$

[1] https://stat.ethz.ch/education/semesters/ss2012/ams/slides/v10.2.pdf

Differences from standard tree

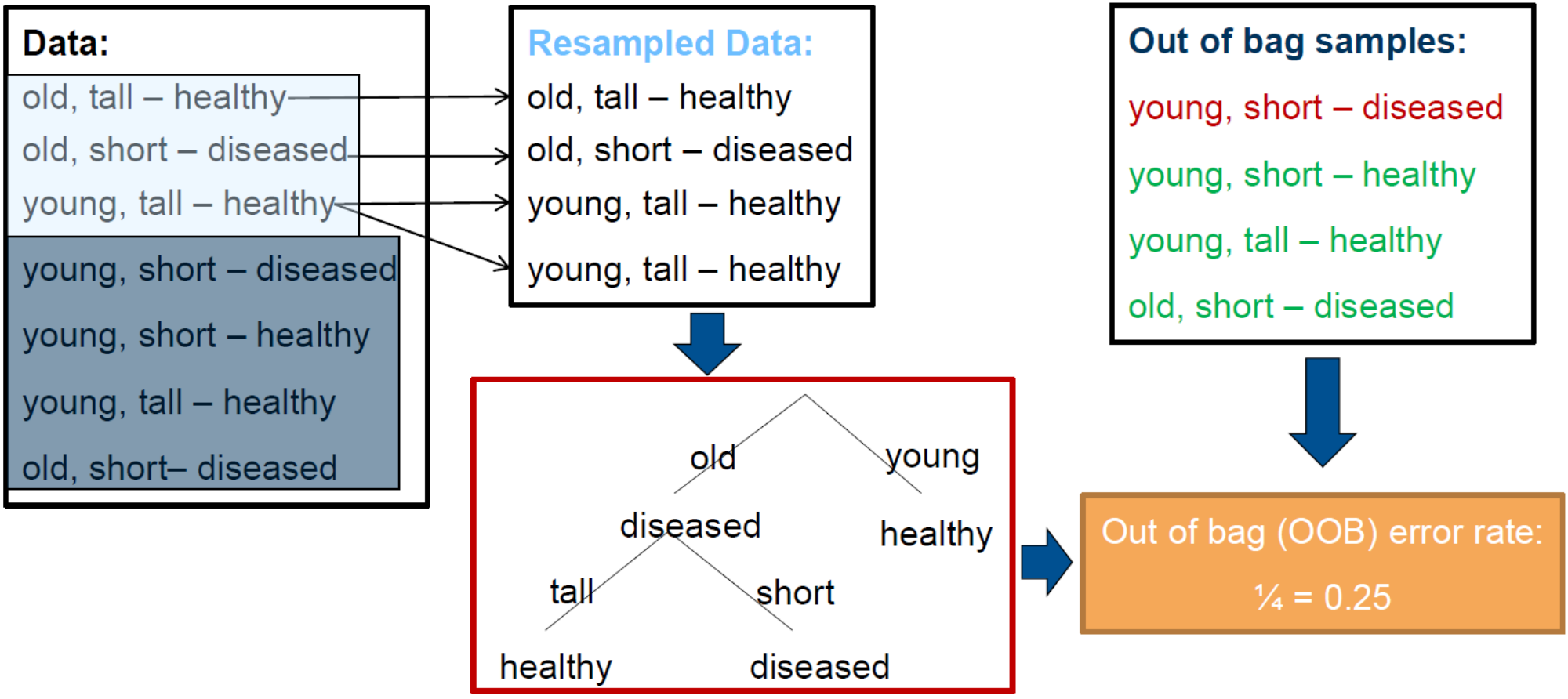
- Train each tree on bootstrap resample of data
- For each split, consider only m randomly selected variables
- Don’t prune
- Use average or majority voting to aggregate results

[1] *https://stat.ethz.ch/education/semesters/ss2012/ams/slides/v10.2.pdf*

Supervised Learning

2.4 Random Forests: Variable importance

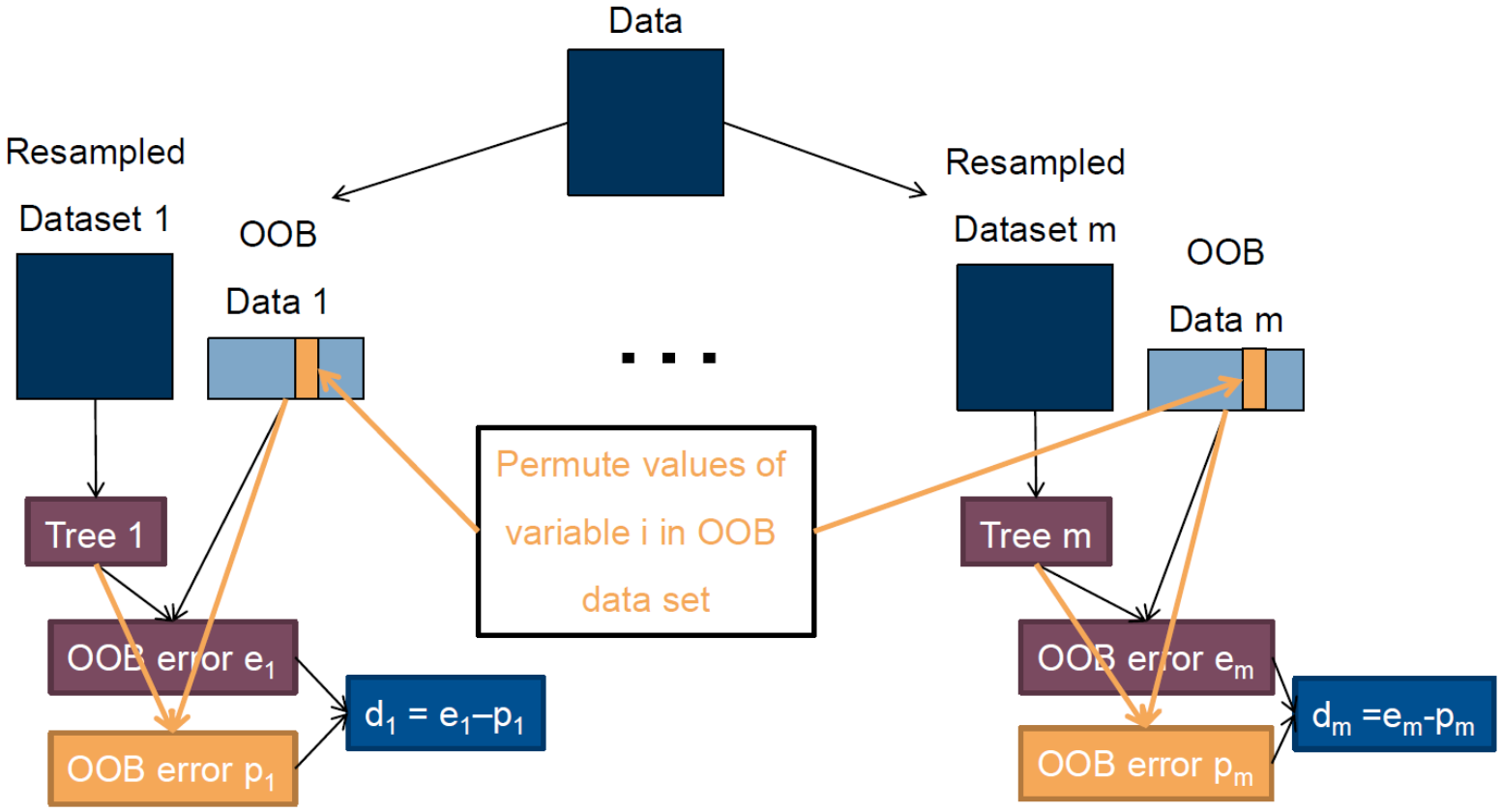
Out of bag(OOB) error



[1] https://stat.ethz.ch/education/semesters/ss2012/ams/slides/v10.2.pdf

Supervised Learning

2.4 Random Forests: Variable importance



$$\bar{d} = \frac{1}{m}\sum_{i=1}^{m} d_i$$

$$s_d^2 = \frac{1}{m-1}\sum_{i=1}^{m}(d_i - \bar{d})^2$$

$$v_i = \frac{\bar{d}}{s_d}$$

[1] https://stat.ethz.ch/education/semesters/ss2012/ams/slides/v10.2.pdf

Drawbacks

- For data including categorical variables with different number of levels, random forests are **biased in favor of those attributes with more leve**ls.
- If the data contain groups of correlated features of similar relevance for the output, then **smaller groups are favored over larger groups**.

[1] *https://en.wikipedia.org/wiki/Random_forest*

Supervised Learning

2.4 Random Forests: Advantages and Disadvantages

Advantages

- Easy to tune parameters
- Can model nonlinear class boundaries
- Works on continuous and categorical responses (regression / classification)
- Gives variable importance
- Very good performance

Disadvantages

- Rather slow
- “Black Box”: Rather hard to get insights into decision rules

[1] https://stat.ethz.ch/education/semesters/ss2012/ams/slides/v10.2.pdf

Supervised Learning

2.5 AdaBoost: Introduction

- **Boosting** is a powerful technique for **combining multiple 'base' classifiers** to produce a form of committee whose performance can be significantly better than that of any of the base classifiers.

- Here we describe the most widely used form of boosting algorithm called **AdaBoost**, short for 'adaptive boosting', developed by Freund and Schapire (1996).

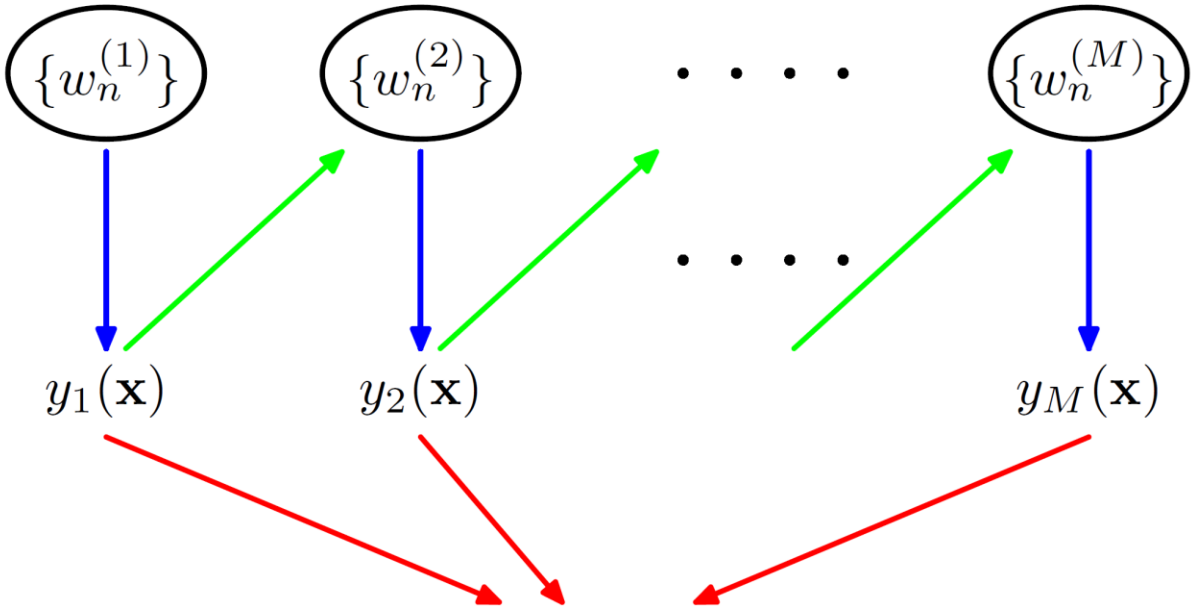
- Boosting can give good results even if the base classifiers have a performance that is only slightly better than random, and hence sometimes the base classifiers are known as **weak learners**.

[1] Bishop, Christopher M. "Pattern recognition." Machine Learning 128 (2006).

Supervised Learning

2.5 AdaBoost: Schematic illustration of the boosting framework

- Each **base classifier** $y_m(\boldsymbol{x})$ is trained on a weighted form of the training set (blue arrows)
- **The weights** $w_n^{(m)}$ depend on the performance of the previous base classifier $y_{m-1}(\boldsymbol{x})$ **(**green arrows).
- Once all base classifiers have been trained, they are combined to give the **final classifier** $Y_M(\boldsymbol{x})$(red arrows).



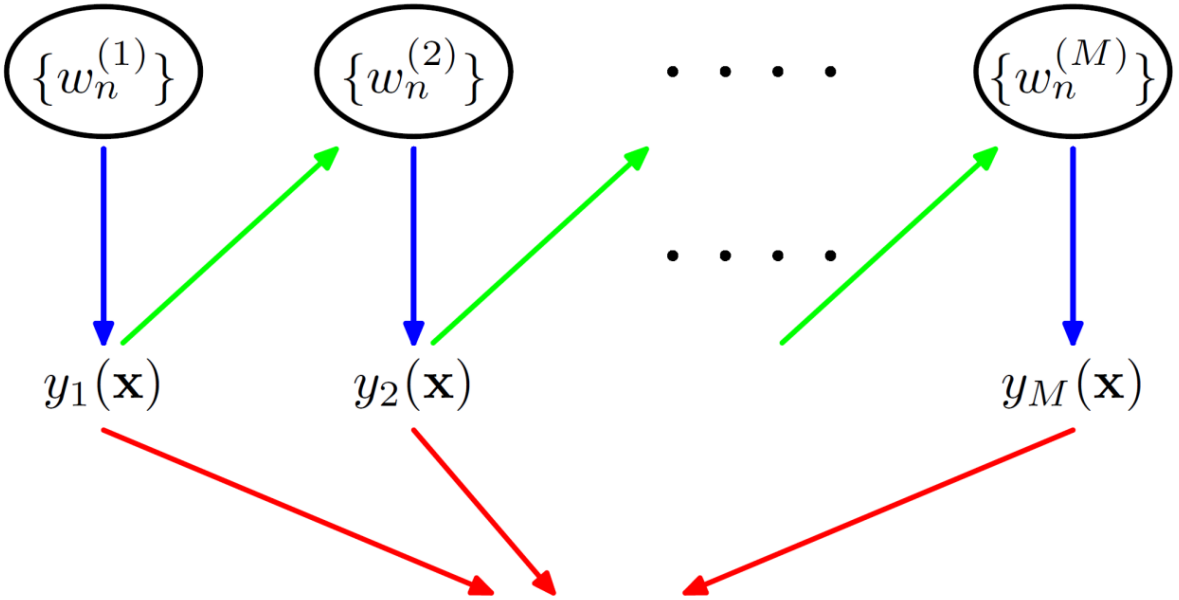
$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_m^M \alpha_m y_m(\mathbf{x})\right)$$

[1] Bishop, Christopher M. "Pattern recognition." Machine Learning 128 (2006).

Supervised Learning

2.5 AdaBoost: Algorithm

Step 1: Initialize the data weighting coefficients $\{w_n\}$ by setting $w_n^{(1)} = 1/N$ for $n = 1,2 \ldots, N$



$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_m^M \alpha_m y_m(\mathbf{x})\right)$$

[1] Bishop, Christopher M. "Pattern recognition." Machine Learning 128 (2006).

Supervised Learning

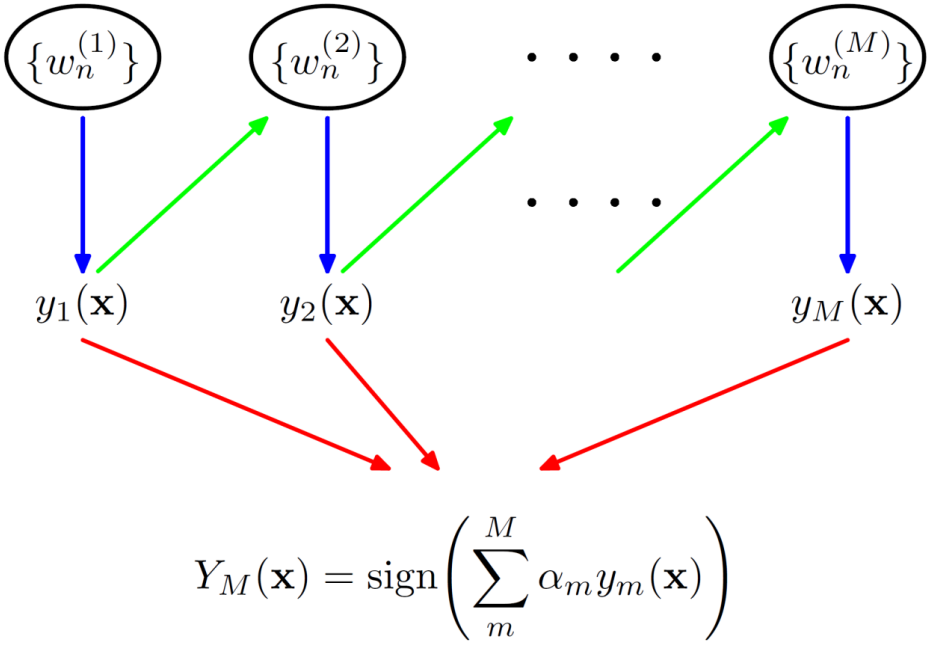
2.5 AdaBoost: Algorithm

Step 2: For m=1,…,M:

2.1 Fit a classifier $\boldsymbol{y_m(x)}$ to the training data by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(\boldsymbol{x}_n) \neq t_n)$$

$I(y_m(\boldsymbol{x}_n) \neq t_n)$ is the indicator function and equals 1 when $y_m(\boldsymbol{x}_n) \neq t_n$ and 0 otherwise.



$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_m^M \alpha_m y_m(\mathbf{x})\right)$$

2.2 **Evaluate the quantities**

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(\boldsymbol{x}_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}$$

And then use these to evaluate

$$\alpha_m = \ln\{\frac{1-\epsilon_m}{\epsilon_m}\}$$

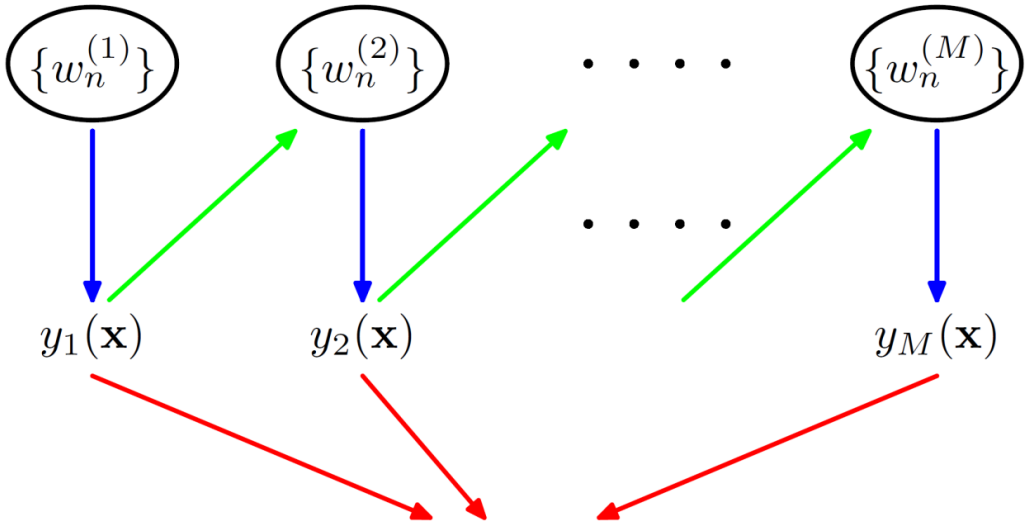
2.3 **Update the data weighting coefficients**

$$w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m I(y_m(\boldsymbol{x}_n) \neq t_n)\}$$

[1] Bishop, Christopher M. "Pattern recognition." Machine Learning 128 (2006).

Step 3: Make predictions using the final model, which is given by

$$Y_M(\boldsymbol{x}) = sign(\sum_{m=1}^{M} \alpha_m y_m(\boldsymbol{x}))$$



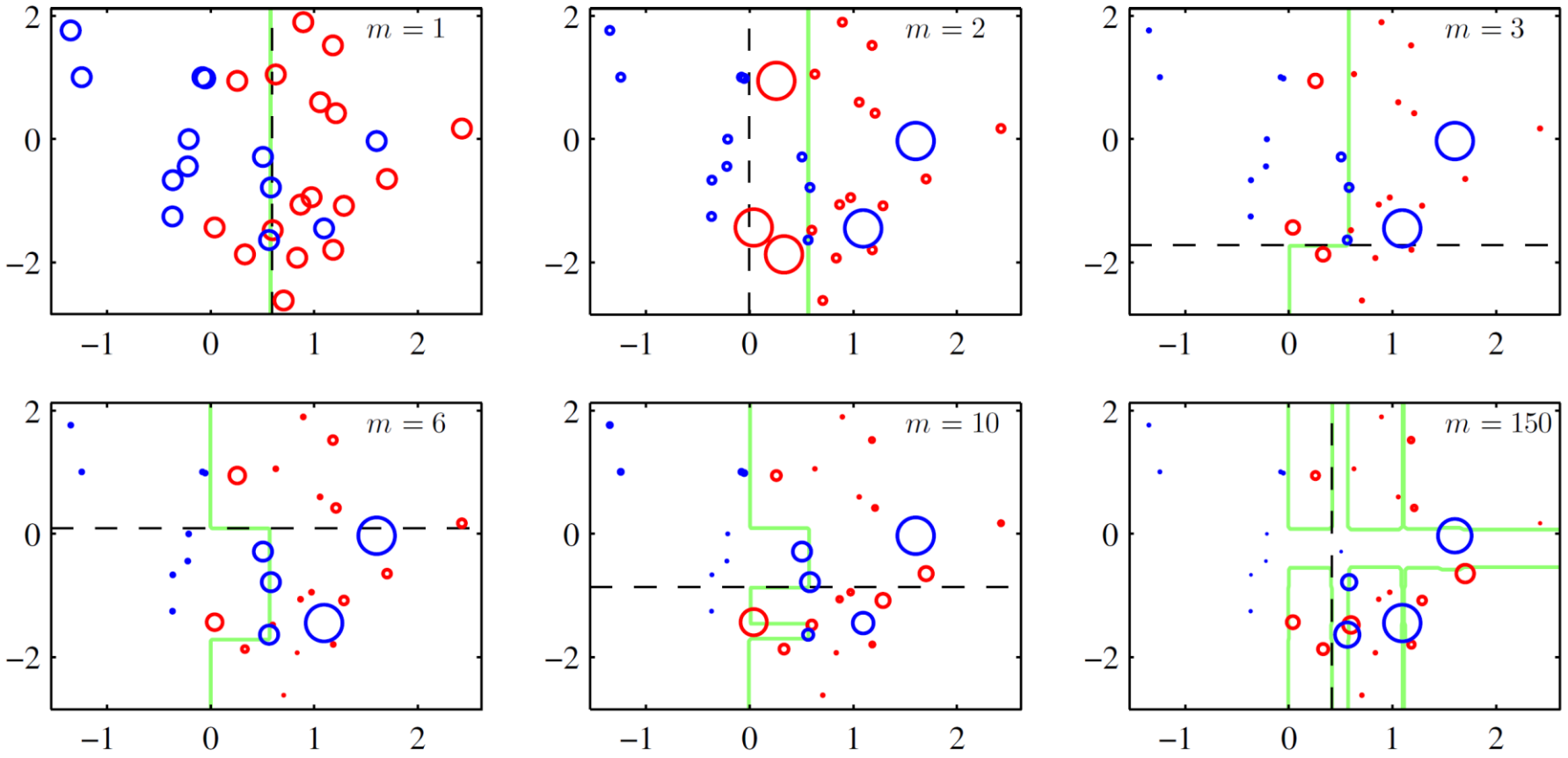
$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_{m}^{M} \alpha_m y_m(\mathbf{x})\right)$$

[1] Bishop, Christopher M. "Pattern recognition." Machine Learning 128 (2006).

Supervised Learning

2.5 AdaBoost: Example

Illustration of boosting in which the base learners consist of simple thresholds applied to one or other of the axes.



[1] Bishop, Christopher M. "Pattern recognition." Machine Learning 128 (2006).

Supervised Learning

2.5 AdaBoost: Advantages and Disadvantages

Pros

- Fast
- Simple and easy to program
- No parameters to tune
- No prior knowledge needed about weak learner
- Provably effective given weak learning assumption
- Versatile

Cons

- Weak classifiers too complex leads to overfitting.
- Weak classifiers too weak can lead to low margins, and can also lead to overfitting.
- From empirical evidence, AdaBoost is particularly vulnerable to uniform noise.

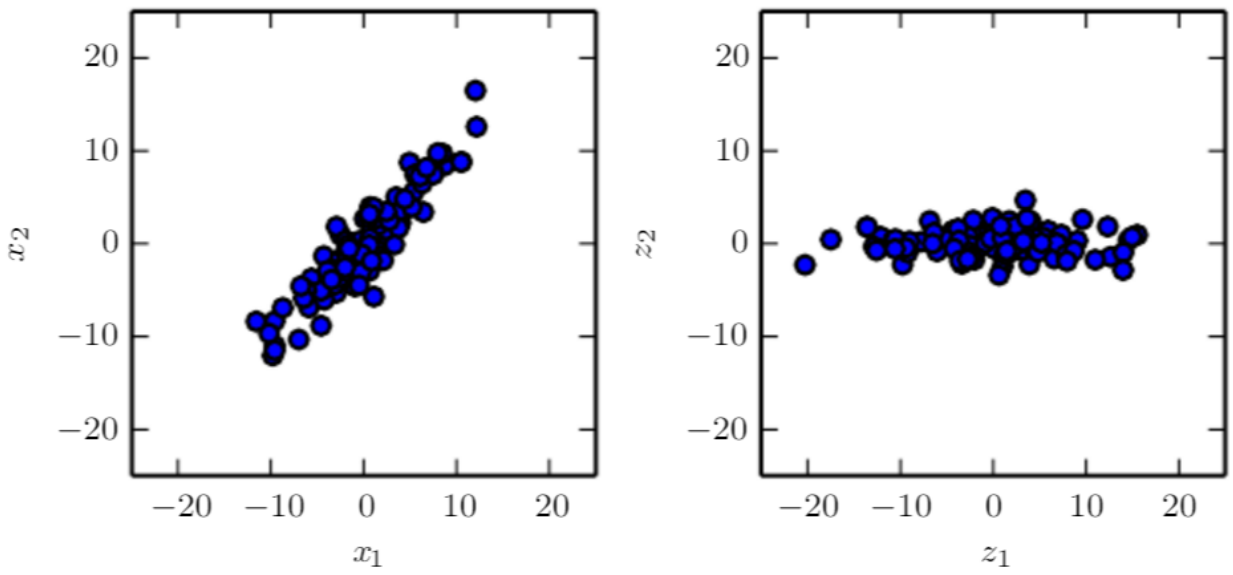
[1] http://math.mit.edu/~rothvoss/18.304.3PM/Presentations/1-Eric-Boosting304FinalRpdf.pdf

Part 3
Unsupervised Learning

- Principal Components Analysis
- K-means Clustering
- Auto-Encoder
- Compressive Sensing

Unsupervised Learning

3.1 Principal Components Analysis



- (Left) The original data consists of samples of $\boldsymbol{x}$. In this space, the variance might occur along directions that are not axis-aligned.
- (Right) The transformed data $\boldsymbol{z} = \boldsymbol{x}^T \boldsymbol{W}$ now varies most along the axis $\boldsymbol{z}_1$. The direction of second most variance is now along $\boldsymbol{z}_2$.

[1] Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville, http://www.deeplearningbook.org, Book in preparation for MIT Press,2016

Unsupervised Learning

3.1 Principal Components Analysis

- Principal components analysis (PCA) is a technique that can be used to **simplify a dataset**
- It is a linear transformation that chooses **a new coordinate system** for the data set such that greatest variance by any projection of the data set comes to lie on the first axis (then called the first principal component), the second greatest variance on the second axis, and so on.
- PCA can be used for **reducing dimensionality** by eliminating the later principal components.

[1] *http://www.cse.psu.edu/~rtc12/CSE586Spring2010/lectures/pcaLectureShort.pdf*

Unsupervised Learning

3.1 Principal Components Analysis

- By finding the **eigenvalues** and **eigenvectors** of the covariance matrix, we find that the eigenvectors with the **largest eigenvalues** correspond to the dimensions that have the strongest correlation in the dataset. This is **the principal component**.

- PCA is a useful statistical technique that has found application in:
  - ✓ fields such as face recognition and **image compression**
  - ✓ finding patterns in **data of high dimension**.

[1] *Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville, http://www.deeplearningbook.org, Book in preparation for MIT Press,2016*

Unsupervised Learning

3.2 K-means Clustering: Introduction

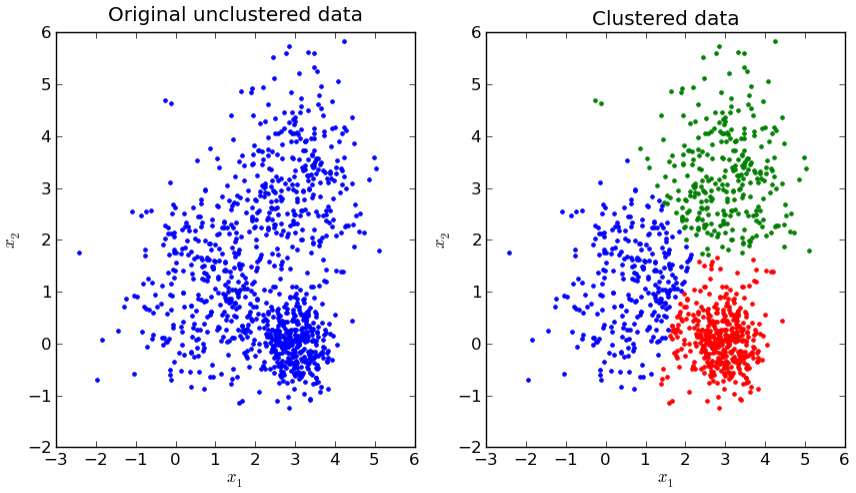
- Given a set of observations $(x_1, x_2, x_3 \ldots x_n)$, where each observation is a d-dimensional real vector, k-means clustering **aims to partition** the n observations into k ($\leq$ n) sets S = $\{S_1, S_2, S_3, \ldots, S_k\}$.

- The goal is to **minimize the within-cluster sum of squares** (WCSS) (sum of distance functions of each point in the cluster to the K center).

- In other words, its objective is to find:

$$\min_{\boldsymbol{S}} \sum_{i=1}^{k} \sum_{x \in S_i} ||\boldsymbol{x} - \boldsymbol{\mu_i}||^2$$

where $\boldsymbol{\mu_i}$ is the mean of points in S_i



[1] https://en.wikipedia.org/wiki/K-means_clustering

Unsupervised Learning

3.2 K-means Clustering: Algorithm

- The Forgy method randomly chooses k observations from the data set and uses these as the initial means.
- The Random Partition method first randomly assigns a cluster to each observation and then proceeds to the update step, thus computing the initial mean to be the centroid of the cluster's randomly assigned points.

Given **an initial set** of k means $m_1^{(1)}, m_2^{(1)}, \ldots, m_k^{(1)}$, the algorithm proceeds by alternating between two steps:

- Assignment step: **Assign each observation to the cluster** whose mean yields the least within-cluster sum of squares (WCSS). Since the sum of squares is the squared Euclidean distance, this is intuitively the "nearest" mean.

$$S_i^{(t)} = \{x_p : \left\| x_p - m_i^{(t)} \right\|^2 \leq \left\| x_p - m_j^{(t)} \right\|^2 \ \forall j, 1 \leq j \leq k\}$$

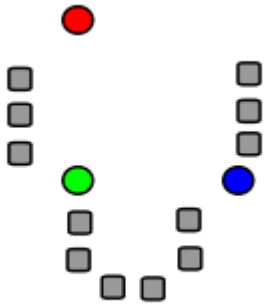
- Update step: **Calculate the new means to be the centroids** of the observations in the new clusters.

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$

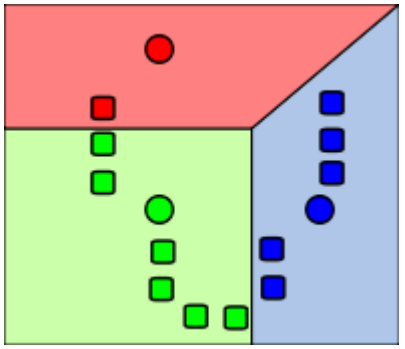
[1] https://en.wikipedia.org/wiki/K-means_clustering

Unsupervised Learning

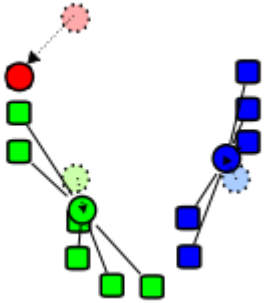
3.2 K-means Clustering: Algorithm



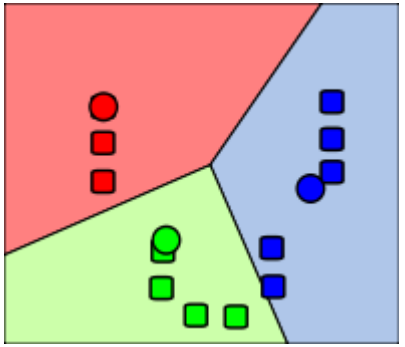
1. k initial "means" (in this case k=3) are randomly generated within the data domain (shown in color).



2. k clusters are created by associating every observation with the nearest mean.



3. The centroid of each of the k clusters becomes the new mean.



4. Steps 2 and 3 are repeated until convergence has been reached.

[1] *https://en.wikipedia.org/wiki/K-means_clustering*

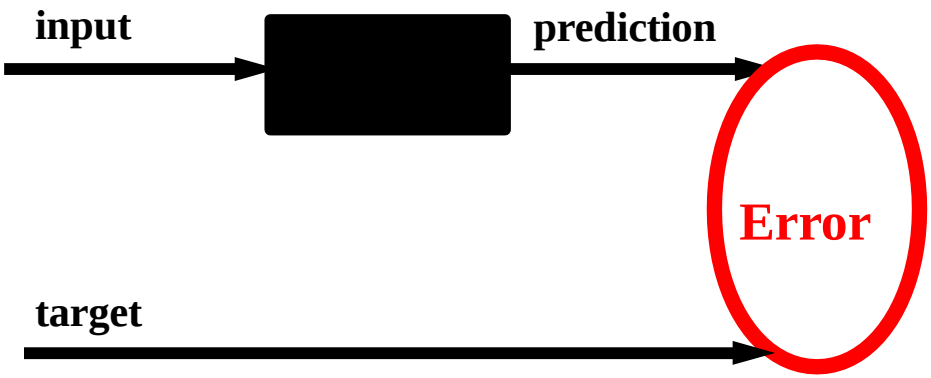
Limitation

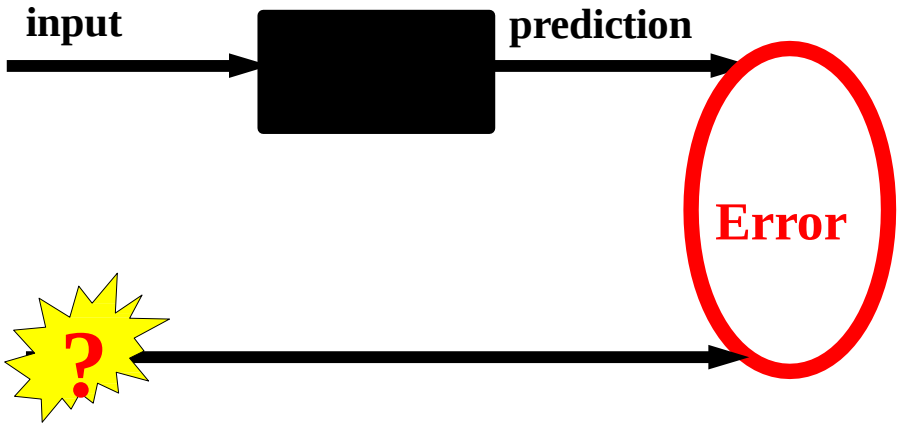
- The number of clusters **k is an input parameter**: an inappropriate choice of k may yield poor results. That is why, when performing k-means, it is important to run diagnostic checks for determining the number of clusters in the data set.
- **Convergence to a local minimum** may produce counterintuitive ("wrong") results.

[1] *https://en.wikipedia.org/wiki/K-means_clustering*

Unsupervised Learning

3.3 Auto-Encoder: Feature Representation





How should we train the input-output mapping if we do not have target values?

Code has to retain information from the input

[1] NEURAL NETS FOR VISION, Marc'Aurelio Ranzato from Google, CVPR 2012 Tutorial on Deep Learning http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/tutorial_p2_nnets_ranzato_short.pdf

[2] http://blog.sina.com.cn/s/blog_6a1b8c6b0101h9ho.html

Unsupervised Learning

3.3 Auto-Encoder: Feature Representation

Three of the most common ways of defining a simpler representation include

- lower dimensional representations
- sparse representations
- independent representations

[1] *NEURAL NETS FOR VISION, Marc'Aurelio Ranzato from Google, CVPR 2012 Tutorial on Deep Learning http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/tutorial_p2_nnets_ranzato_short.pdf*

[2] *http://blog.sina.com.cn/s/blog_6a1b8c6b0101h9ho.html*

Unsupervised Learning

3.3 Auto-Encoder: Feature Representation

How to constrain the model to represent training samples?

- Reconstruct the input from the code and make code compact (**Auto-encoder with Bottleneck**).
- Reconstruct the input from the code and make code sparse (**Sparse Auto-encoders**)

  Work in LeCun, Ng, Fergus, Lee, Yu's labs
- Input is partially corrupted(**Denoising Auto-encoders**)

  Work in Y. Bengio, Lee's lab
- Make sure that the model defines a distribution that normalizes to 1 (**Restricted Boltzmann Machine**).

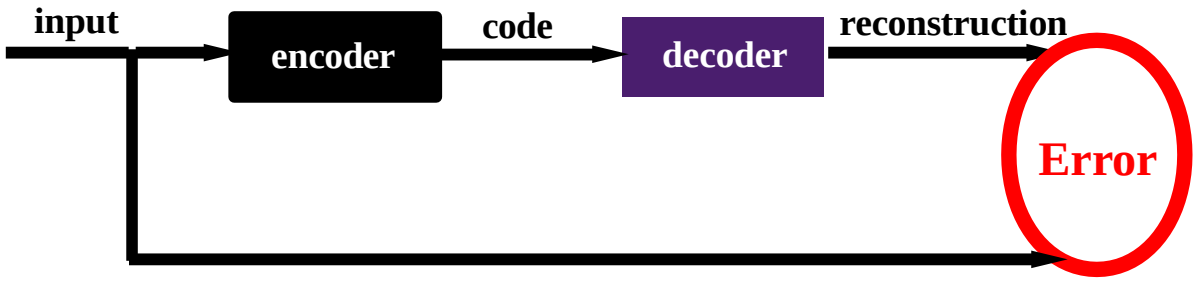
  Work in Y. Bengio, Hinton, Lee, Salakthudinov's lab

[1] *NEURAL NETS FOR VISION, Marc'Aurelio Ranzato from Google, CVPR 2012 Tutorial on Deep Learning http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/tutorial_p2_nnets_ranzato_short.pdf*

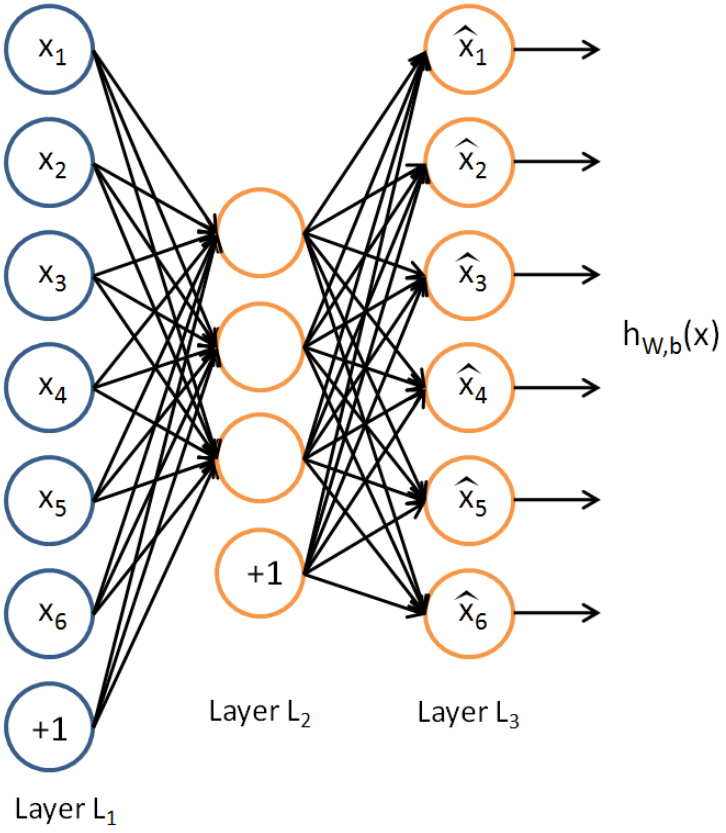
[2] *http://blog.sina.com.cn/s/blog_6a1b8c6b0101h9ho.html*

Unsupervised Learning

3.3 Auto-Encoder: Introduction



- The output layer has **the same number of nodes** as the input layer.
- Instead of being trained to predict the target value Y given inputs, autoencoders are trained to **reconstruct** their own inputs.
- Error Measure: $\|\text{Reconstruction} - \text{Input}\|^2$
- Training: **Back-propagation**



[1] NEURAL NETS FOR VISION, Marc'Aurelio Ranzato from Google, CVPR 2012 Tutorial on Deep Learning http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/tutorial_p2_nnets_ranzato_short.pdf

[2] http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/

[3] http://blog.sina.com.cn/s/blog_6a1b8c6b0101h9ho.html

Unsupervised Learning

3.3 Auto-Encoder: Introduction

Imposing constraints on the network, and discover interesting structure about the data

- limit the number of hidden units: If the input were completely random, this compression task would be very difficult. **If there is structure in the data**, for example, if some of the input features are correlated, then this algorithm will be able to discover some of those correlations. (**Auto-encoder with Bottleneck**)
- Impose a sparsity constraint on the hidden units: Sparse coding is a class of unsupervised methods for **learning sets of over-complete bases to represent data efficiently**. The advantage of having an over-complete basis is that our basis vectors are better able to capture structures and patterns inherent in the input data. (**Sparse Auto-encoders**)
- Add noise to the input: Train the autoencoder to reconstruct the input from a corrupted version of it, to force the hidden layer to **discover more robust features** and prevent it from simply learning the identity. (**Denoising Auto-encoders**)

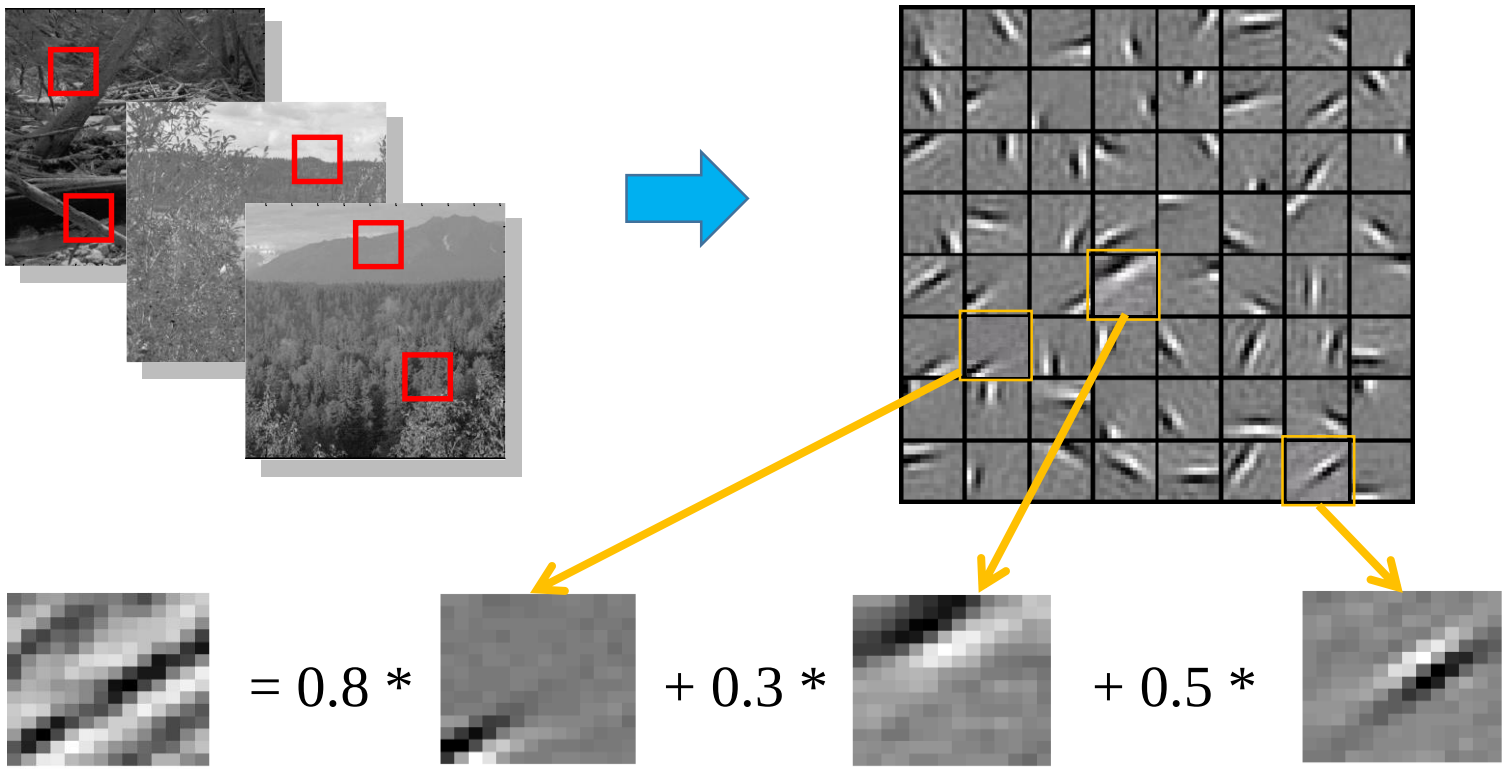
Unsupervised Learning

3.3 Auto-Encoder: Sparse Auto-encoders

$[a_1, \ldots, a_{64}] = [0, 0, \ldots, 0, \mathbf{0.8}, 0, \ldots, 0, \mathbf{0.3}, 0, \ldots, 0, \mathbf{0.5}, 0]$

(feature representation)

Sparse Coding



- Automatically learns to represent an image in terms of the edges that appear in it. Gives a more succinct, higher-level representation than the raw pixels.

[1] Unsupervised Feature Learning and Deep Learning Andrew Ng, https://www.csee.umbc.edu/courses/graduate/678/spring15/visionaudio.pdf

Unsupervised Learning

3.3 Auto-Encoder: Sparse Auto-encoders

Sparse Coding

- Sparse coding is a class of unsupervised methods for learning sets of over-complete bases to represent data efficiently.
- The aim of sparse coding is **to find a set of basis vectors $\boldsymbol{\phi_i}$** such that we can represent an input vector as a linear combination of these basis vectors:

$$x = \sum_{i=1}^{k} a_i \phi_i$$

- we wish to learn an over-complete set of basis vectors to represent input vectors $x \in \mathcal{R}^n$ (i.e. such that k>n).
- The advantage of having an over-complete basis is that our basis vectors are better **able to capture structures** and patterns inherent in the input data.

[1] http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/

Sparse Coding

- We define the sparse coding cost function on a set of m input vectors as

$$\min_{\alpha_i^{(j)},\phi_i} \sum_{j=1}^{m} \left\| x^{(j)} - \sum_{i=1}^{k} \alpha_i^{(j)} \phi_i \right\|^2 + \lambda \sum_{i=1}^{k} S\left(\alpha_i^{(j)}\right)$$

$$\text{Subject to } ||\phi_i||^2 \leq C, \forall i = 1, \ldots, k$$

- a reconstruction term which tries to force the algorithm to provide a good representation of input

- A sparsity penalty which forces our representation to be sparse.
- S(.) is a sparsity cost function which penalizes a_i for being far from zero.
- The constant λ is a scaling constant to determine the relative importance of these two contributions.

[1] http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/

Sparse Coding

$$\min_{\alpha_i^{(j)},\phi_i} \sum_{j=1}^{m} \left\| x^{(j)} - \sum_{i=1}^{k} \alpha_i^{(j)} \phi_i \right\|^2 + \lambda \sum_{i=1}^{k} S\left(\alpha_i^{(j)}\right)$$

Subject to $||\phi_i||^2 \leq C, \forall i = 1, \ldots, k$

Alternating optimization:

- Fix dictionary $\phi_1, \phi_2, \phi_3, \ldots, \phi_k$, optimize $\boldsymbol{a}$ (a standard LASSO problem)
- Fix activations $\boldsymbol{a}$, optimize $\phi_1, \phi_2, \phi_3, \ldots, \phi_k$ (a convex QP problem)

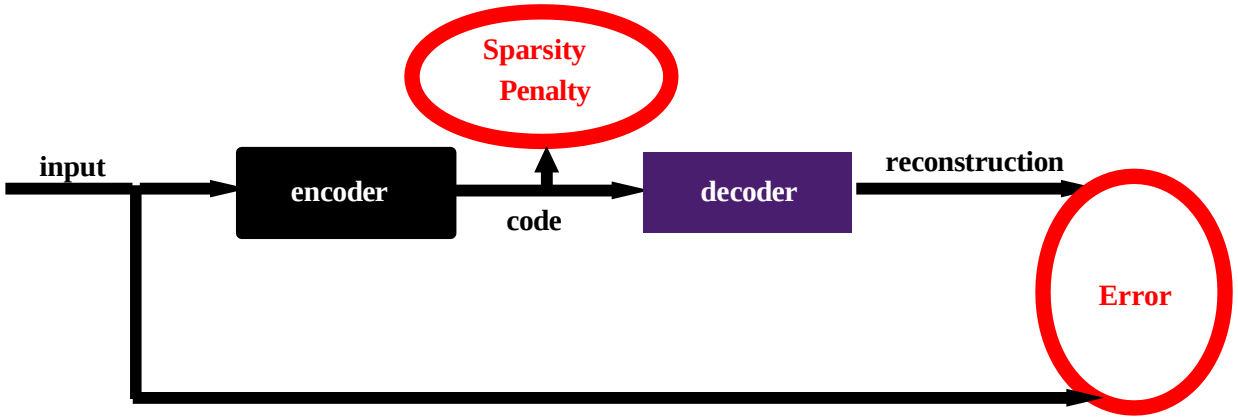
Limitation

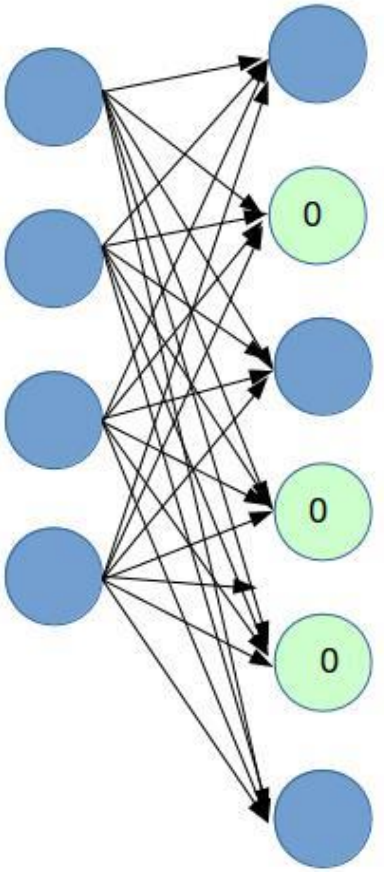
- Even after a set of basis vectors have been learnt, in order to "encode" a new data example, optimization must be performed to obtain the required coefficients. This **significant "runtime"** cost means that sparse coding is computationally expensive to implement even at test time.

[1] http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/

Sparse Auto-encoders

- We would like to constrain the neurons to be inactive most of the time.





[1] http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/

Sparse Auto-encoders

- suppose we have only a set of unlabeled training examples $\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots\ldots\}, where\ x^{(i)} \in \mathcal{R}^n$
- we will write $a_j^{(2)}(x)$ to denote the activation of this hidden unit when the network is given a specific input.
- Further, let

$$\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m}[a_j^{(2)}(x^{(i)})]$$

be the average activation of hidden unit j (averaged over the training set).

- We would like to (approximately) enforce the constraint

$$\hat{\rho}_j = \rho$$

Where ρ is a "**sparsity parameter**", typically a small value close to zero (say ρ=0.05).

[1] http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/

Unsupervised Learning

3.3 Auto-Encoder: Sparse Auto-encoders

Sparse Auto-encoders

Overall cost function:

$$J_{sparse}(W,b) = J(W,b) + \beta \sum_{j=1}^{s_2} KL(\rho \| \hat{\rho}_j)$$

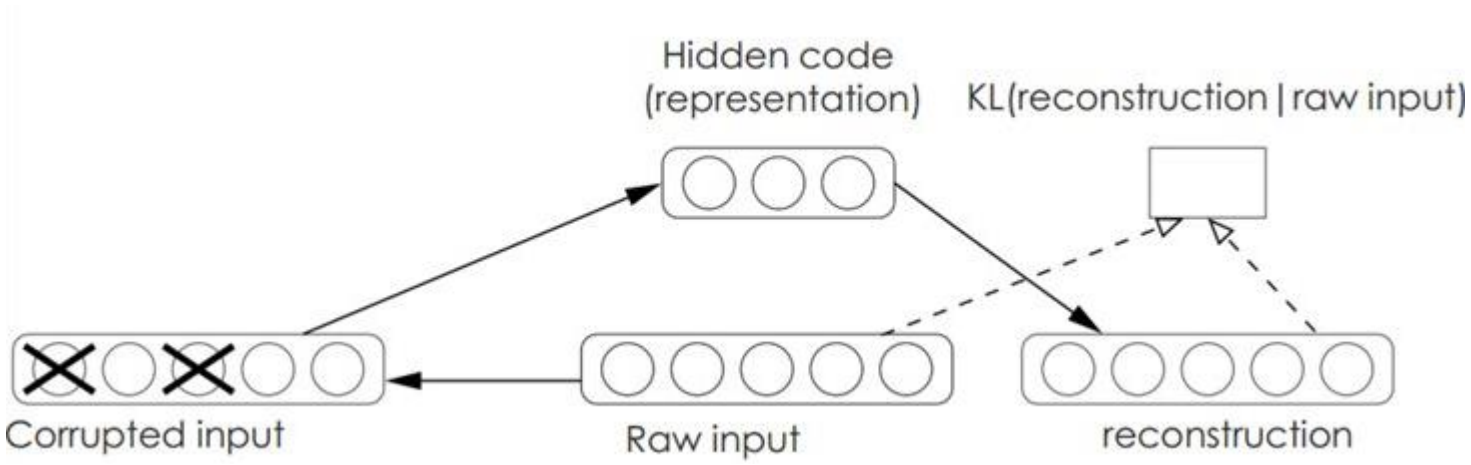
$J(W,b)$ is error function, $KL(\rho \| \hat{\rho}_j) = \rho log \frac{\rho}{\hat{\rho}_j} + (1-\rho) log \frac{1-\rho}{1-\hat{\rho}_j}$, s_2 is the number of neurons in the hidden layer.

Note: Error function and penalty term have many choices

[1] *http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/*

Unsupervised Learning

3.3 Auto-Encoder: Denoising Auto-encoders

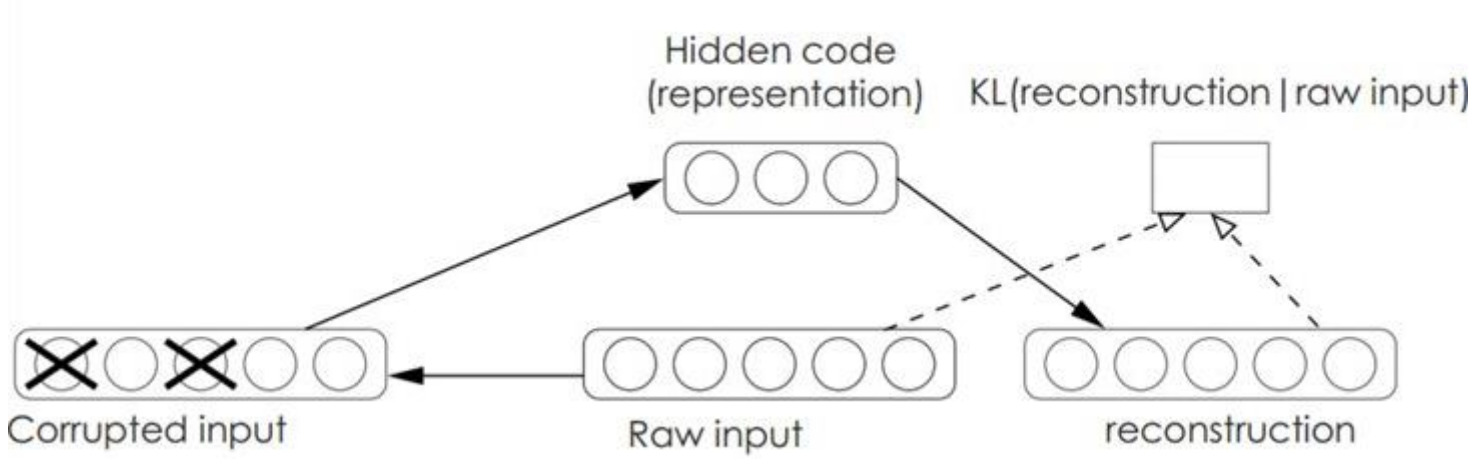


- Corrupt the input(e.g. set 25% of input to 0)
- Reconstruct the uncorrupted input
- Use uncorrupted encoding as input to next level

[1] *http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/*

Unsupervised Learning

3.3 Auto-Encoder: Denoising Auto-encoders



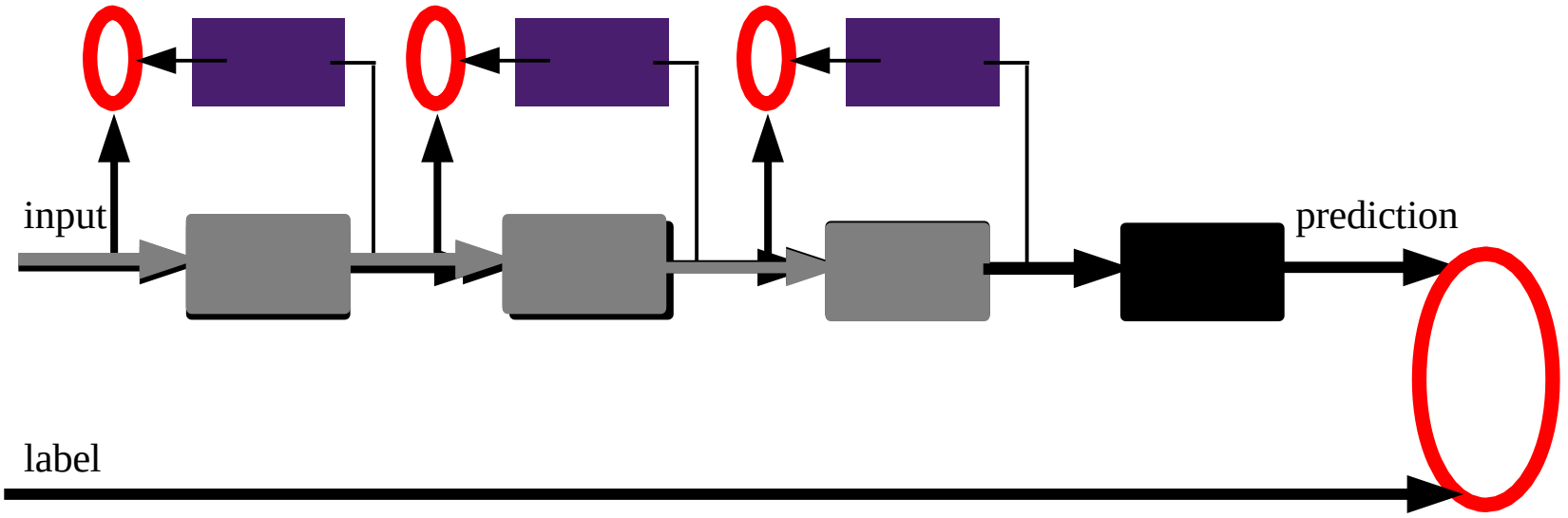
- In order to force the hidden layer **to discover more robust features** and prevent it from simply learning the identity, we train the autoencoder to reconstruct the input from a corrupted version of it.
- The denoising auto-encoder is a stochastic version of the auto-encoder. A denoising auto-encoder does two things:
  - try to **encode the input** (preserve the information about the input), and
  - try to **undo the effect of a corruption process** stochastically applied to the input of the auto-encoder. This can only be done by capturing the statistical dependencies between the inputs.

[1] http://deeplearning.net/tutorial/dA.html

Unsupervised Learning

3.3 Auto-Encoder: How to Use Feature Representation

- Given unlabeled data, learn features
- Use encoder to produce features and train another layer on the top
- feed features to classifier and jointly train the whole system



[1] NEURAL NETS FOR VISION, Marc'Aurelio Ranzato from Google, CVPR 2012 Tutorial on Deep Learning http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/tutorial_p2_nnets_ranzato_short.pdf

Traditional Signal Processing

- The Shannon/Nyquist sampling theorem specifies that to avoid losing information when capturing a signal, one must sample at least **two times faster than the signal bandwidth**.

- In many applications, including digital image and video cameras, the Nyquist rate is **so high that too many samples result**, making compression a necessity prior to storage or transmission.

- In other applications, including imaging systems (medical scanners and radars) and high-speed analog-to-digital converters, **increasing the sampling rate is very expensive**.

[1] http://dsp.rice.edu/sites/dsp.rice.edu/files/cs/baraniukCSlecture07.pdf

Signal processing using Compressive Sensing

- a signal processing technique for **efficiently acquiring and reconstructing a signal**, by finding solutions to underdetermined linear systems.

- This is based on the principle that, through optimization, the sparsity of a signal can be exploited to recover it from **far fewer samples than required by the Shannon-Nyquist** sampling theorem.

- There are two conditions under which recovery is possible.
  - ✓ The first one is **sparsity** which requires the signal to be sparse in some domain.
  - ✓ The second one is **incoherence** which is applied through the isometric property which is sufficient for sparse signals.

[1] *http://people.eecs.berkeley.edu/~oholtz/Talks/CS.pdf*

[2] *https://en.wikipedia.org/wiki/Compressed_sensing*

Mathematical Representation

- Original Data $x \epsilon \mathbb{R}^{N}$, Sampling Data y $\epsilon \mathbb{R}^{M}$, Φ is a $M \times N$ measurement matrix. The compressive sensing problem is that, given y and Φ, solving x.

$$y = \Phi x$$

- A general signal is not sparse, which can be represented by sparse linear combinations of basis: $x = \Psi s$. Ψ is a orthogonal basis matrix. s is a K-sparse signal.

[1] *http://blog.csdn.net/zouxy09/article/details/8118329*

Unsupervised Learning

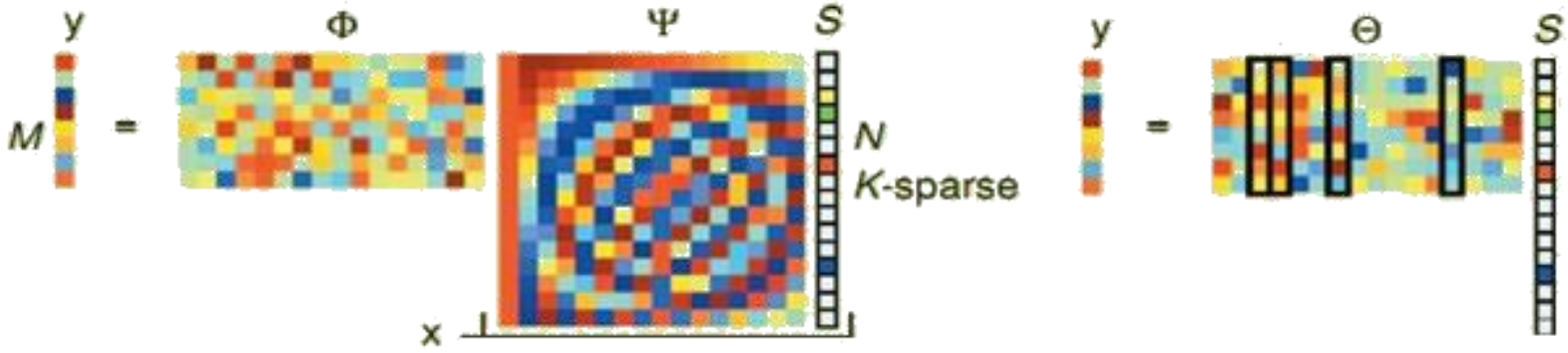
3.4 Compressive Sensing: Algorithm

Mathematical Representation

- Original Data $x \in \mathbb{R}^N$, Sampling Data y $\in \mathbb{R}^M$, s is a K-sparse signal.
- Φ is a $M \times N$ measurement matrix. Ψ is a orthogonal basis matrix.

$$y = \Phi x = \Phi \Psi s = \Theta s$$

- $\Theta = \Phi\Psi$ is called sensing matrix.
- $K < M \ll N$

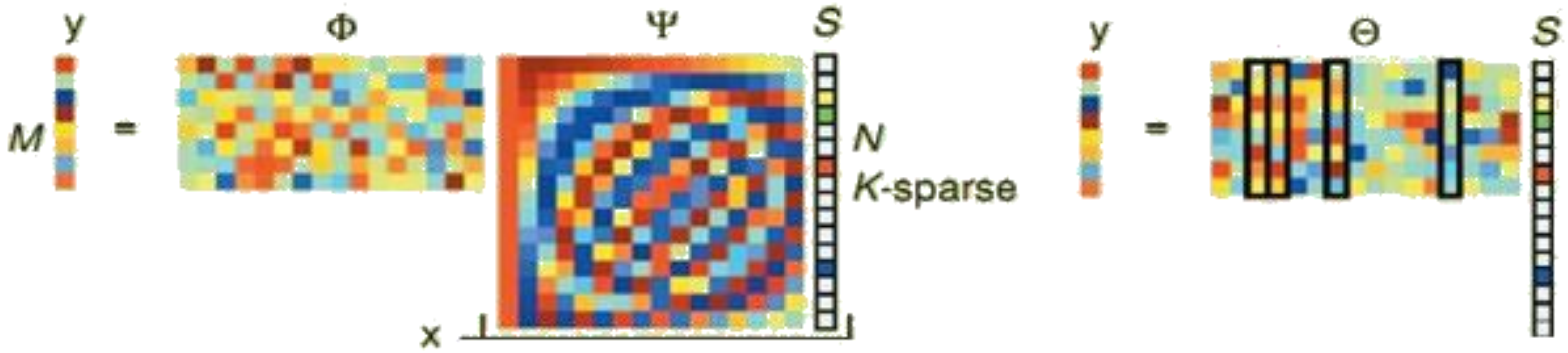


[1] http://blog.csdn.net/zouxy09/article/details/8118329

Unsupervised Learning

3.4 Compressive Sensing: Algorithm

$$y = \Phi x = \Phi \Psi s = \Theta s$$



Key problem is to design

- **a stable measurement matrix Φ**

  the salient information in any K-sparse or compressible signal is not damaged by the dimensionality reduction from $x \in \mathbb{R}^N$ to y $\in \mathbb{R}^M$

- **a reconstruction algorithm**

  recover x from only M ≈ K measurements y.

[1] http://dsp.rice.edu/sites/dsp.rice.edu/files/cs/baraniukCSlecture07.pdf

Unsupervised Learning

3.4 Compressive Sensing: Algorithm

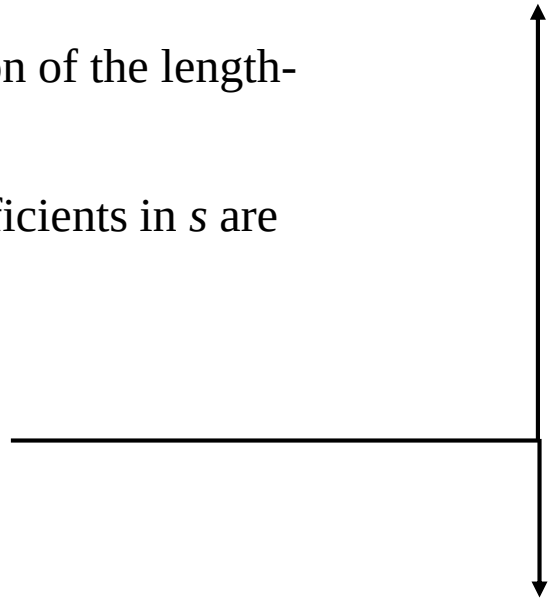
- The Restricted Isometry Property (RIP) introduced by Candes´ and Tao is a fundamental property in compressed sensing theory.
- It says that if a sampling matrix satisfies the RIP of certain order proportional to the sparsity of the signal, then the original signal can be reconstructed even if the sampling matrix provides a sample vector which is much smaller in size than the original signal.

Designing a Stable Measurement Matrix

- The measurement matrix Φ must allow the reconstruction of the length-N signal x from $M < N$ measurements (the vector y).
- If, x is *K-sparse* and the K locations of the nonzero coefficients in s are known, then the problem can be solved provided $M \geq K$.
- The requirement is

  (1) $\Theta = \Phi\Psi$ has **restricted isometry property (RIP)**

  (2) The matrix Φ is **incoherent** with the basis Ψ



RIP definition: Let A be an $m \times p$ matrix and let $1 \leq s \leq p$ be an integer. Suppose that there exists a constant $\delta_s \in (0,1)$ such that, for every $m \times s$ submatrix A_s of A and for every s-sparsity vector y,

$$(1 - \delta_s)||y||_2^2 \leq ||A_s y||_2^2 \leq (1 + \delta_s)||y||_2^2$$

Then, the matrix A is said to satisfy the s-restricted isometry property with restricted isometry constant δ_s

[1] http://dsp.rice.edu/sites/dsp.rice.edu/files/cs/baraniukCSlecture07.pdf

[2] Ying, Leslie, and Yi Ming Zou. "Linear transformations and restricted isometry property." 2009 IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE, 2009.

[3] https://en.wikipedia.org/wiki/Restricted_isometry_property

Designing a Signal Reconstruction Algorithm(1/3)

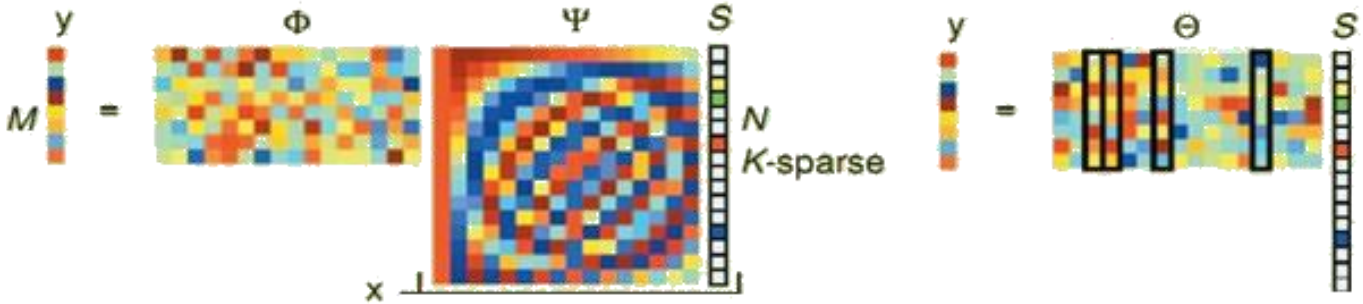
For K-sparse signals, since M < N in there are **infinitely** many s' that satisfy $\Theta s' = y$.

Therefore, the signal reconstruction algorithm aims to **find the sparse one**.

(1) Minimize l_2 norm reconstruction

$$\hat{s} = \underset{s'}{\operatorname{argmin}} ||s'||_2 \ \ s.t. \ \ \Theta s' = y$$

This optimization has the convenient closed-form solution $\hat{s} = \Theta^T(\Theta\Theta^T)^{-1}y$. Unfortunately, l_2 minimization will almost never find a K-sparse solution, **returning instead a non-sparse** $\hat{s}$ with many nonzero elements.



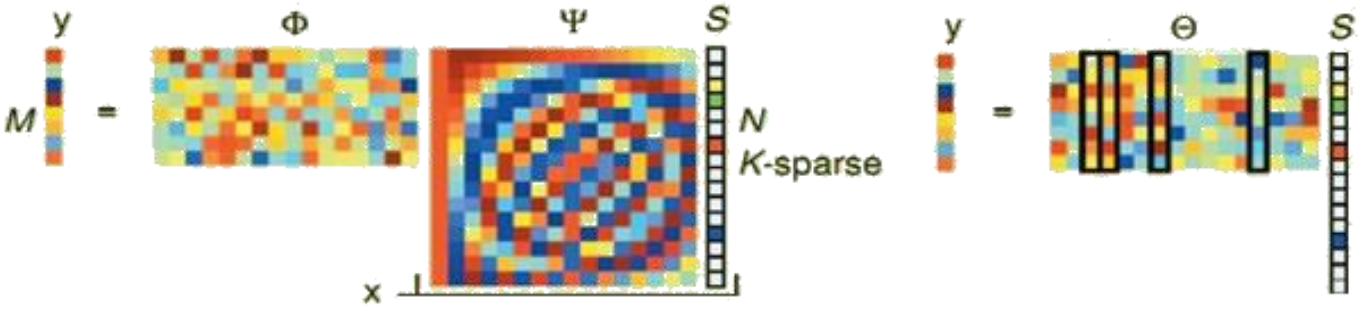
[1] http://dsp.rice.edu/sites/dsp.rice.edu/files/cs/baraniukCSlecture07.pdf

Designing a Signal Reconstruction Algorithm(2/3)

(2) Minimize l_0 norm reconstruction

$$\hat{s} = \underset{s'}{\operatorname{argmin}} ||s'||_0 \ \ s.t. \ \ \Theta s' = y$$

Consider the l_0 norm that counts the number of non-zero entries in s. Unfortunately, solving this problems is both **numerically unstable and NP-complete**.



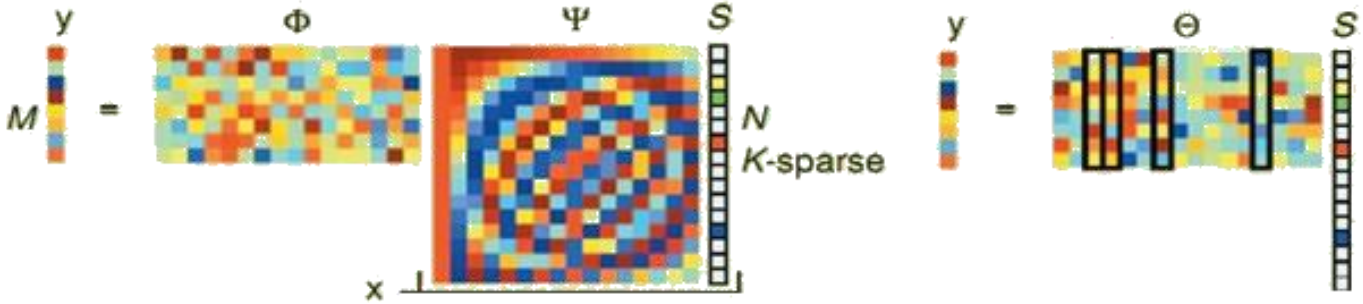
[1] http://dsp.rice.edu/sites/dsp.rice.edu/files/cs/baraniukCSlecture07.pdf

Designing a Signal Reconstruction Algorithm(3/3)

(3) Minimize l_1 norm reconstruction

$$\hat{s} = \underset{s'}{\operatorname{argmin}} ||s'||_1 \ \ s.t. \ \ \Theta s' = y$$

Surprisingly, optimization based on the l_1 norm **can exactly recover K-sparse signals** and **closely approximate compressible signals** with high probability. This is a convex optimization problem that conveniently reduces to a linear program known as basis pursuit whose computational complexity is about $O(N^3)$.



[1] http://dsp.rice.edu/sites/dsp.rice.edu/files/cs/baraniukCSlecture07.pdf

Part 4
Semi-supervised Learning

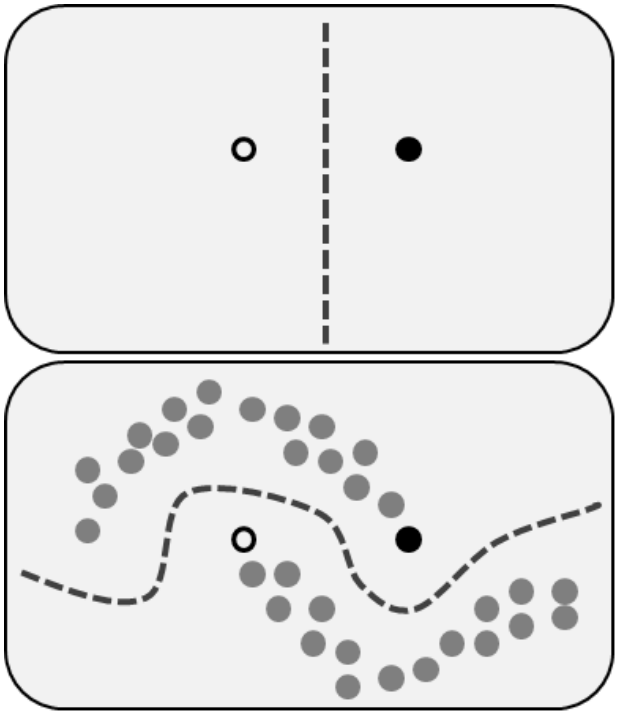
- Self-Training
- Generative Models
- Transductive SVMs
- Graph-based Algorithm
- Multi-view Algorithm

Semi-supervised Learning

4.0 Introduction: Basic Idea

- Supervised Learning models require labeled data

  Learning a reliable model usually requires plenty of labeled data
- Labeled Data: Expensive and Scarce
- Unlabeled Data: Abundant and Free/Cheap
- Semi-supervised Learning: Devising ways of utilizing unlabeled data with labeled data to learn better models



[1] https://www.cs.utah.edu/~piyush/teaching/8-11-slides.pdf

[2] https://en.wikipedia.org/wiki/Semi-supervised_learning

Semi-supervised Learning

4.0 Introduction: When Can Semi-Supervised Learning Work

In comparison with a supervised algorithm that uses only labeled data, can one hope to have a more accurate prediction by taking into account the unlabeled points?

Yes, with an important prerequisite:

- The distribution of examples, which **the unlabeled data will help elucidate**, be relevant for the classification problem. In a more mathematical formulation, one could say that the knowledge on p(x) that one gains through the unlabeled data has to carry information that is useful in the inference of p(y|x).
- If this is not the case, semi-supervised learning will not yield an improvement over supervised learning. It **might even happen** that using the unlabeled data **degrades the prediction accuracy** by misguiding the inference.

[1] Semi-Supervised Learning, Olivier Chapelle, Bernhard Schölkopf, Alexander Zien, MIT press, http://www.acad.bg/ebook/ml/MITPress-%20SemiSupervised%20Learning.pdf

Semi-supervised Learning

4.0 Introduction: Assumptions Used

- In order to make any use of unlabeled data, we must **assume some structure** to the underlying distribution of data. Semi-supervised learning algorithms make use of at least one of the following assumptions.
- Without such assumptions, it would never be possible to generalize from a finite training set to a set of possibly infinitely many unseen test cases.

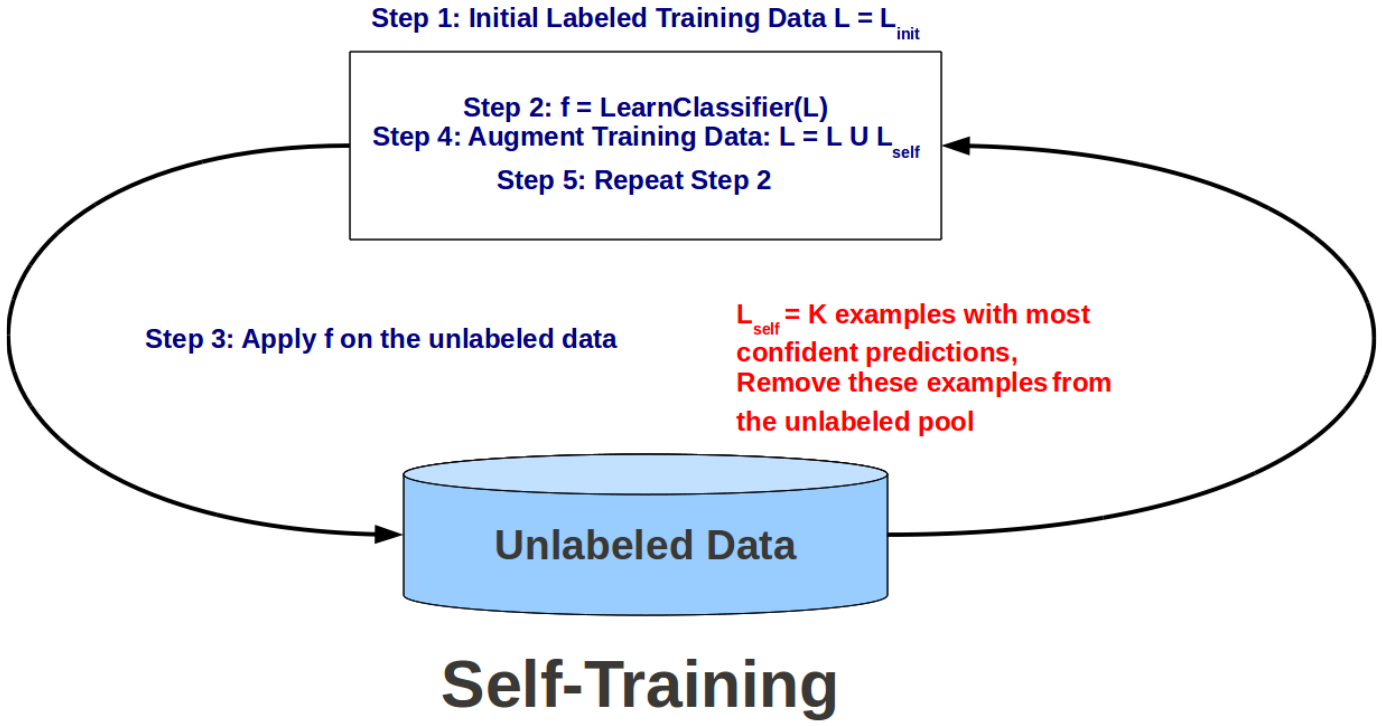
[1] Semi-Supervised Learning, Olivier Chapelle, Bernhard Schölkopf, Alexander Zien, MIT press, http://www.acad.bg/ebook/ml/MITPress-%20SemiSupervised%20Learning.pdf

Semi-supervised Learning

4.1 Self-Training

Given: Small amount of initial labeled training data

Idea: Train, predict, re-train using your own (best) predictions, repeat



- Can be used with any supervised learner. Often **works well in practice**
- **Prediction mistake can reinforce** itself

[1] https://www.cs.utah.edu/~piyush/teaching/8-11-slides.pdf

Semi-supervised Learning

4.1 Self-Training

Assumption

One's own high confidence predictions are correct.

Self-training algorithm

- Train f from (X_l, Y_l)
- Predict on $x \epsilon X_u$
- Add $(x, f(x))$ to labeled data
- Repeat

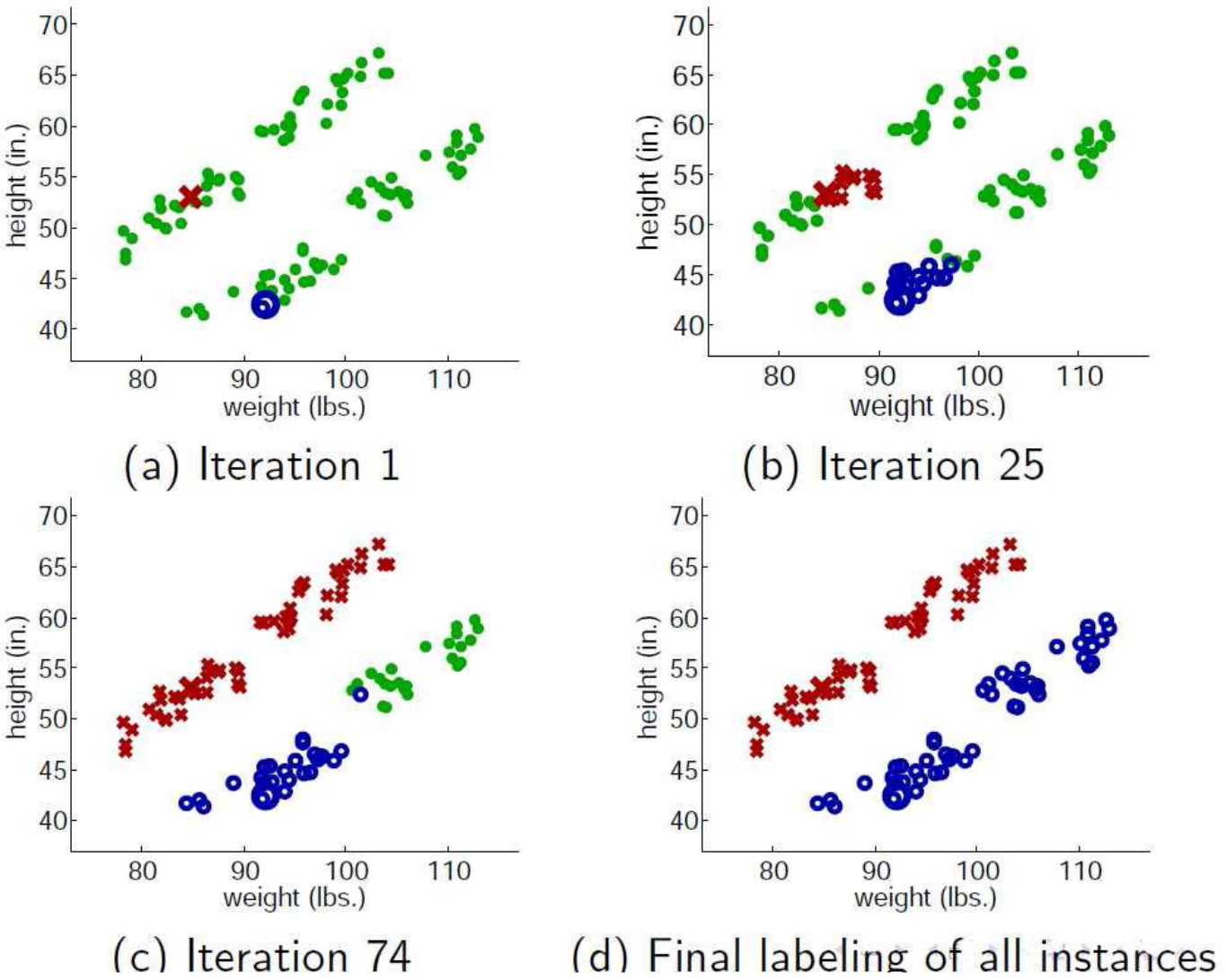
[1] http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

4.1 Self-Training

A good case

Base learner: KNN classifier



[1] https://www.cs.utah.edu/~piyush/teaching/8-11-slides.pdf

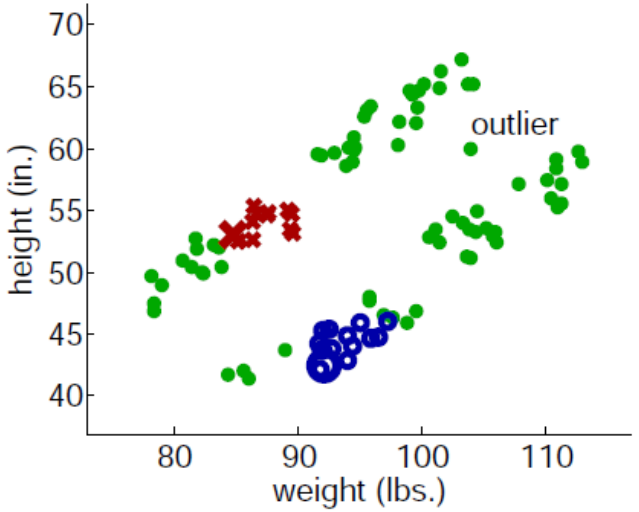
Semi-supervised Learning

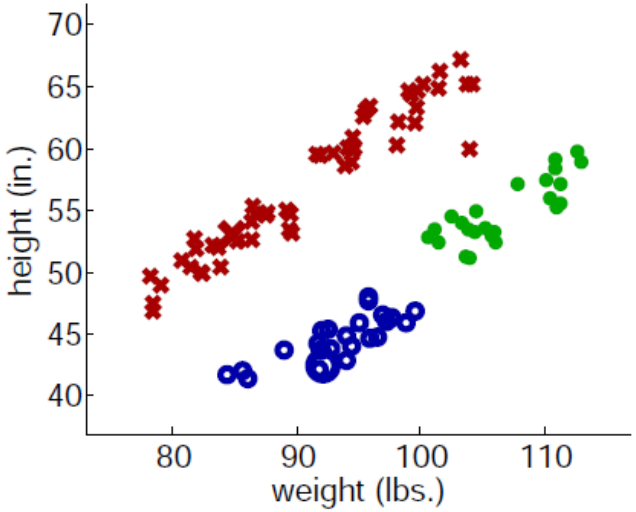
4.1 Self-Training

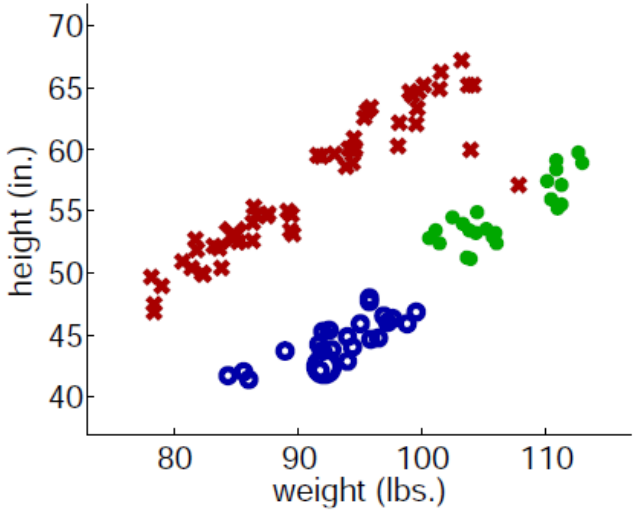
A bad case

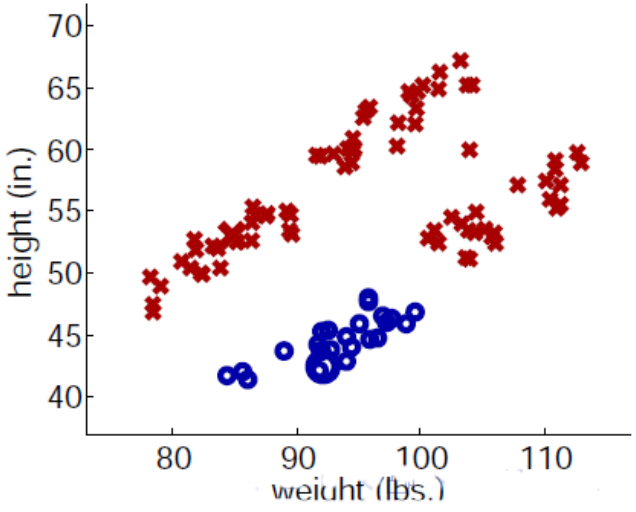
Base learner: KNN classifier

Things can go wrong if there are **outliers**. Mistakes get reinforced.









[1] https://www.cs.utah.edu/~piyush/teaching/8-11-slides.pdf

Semi-supervised Learning

4.1 Self-Training

Advantages

- The **simplest** semi-supervised learning method.
- A **wrapper** method, applies to existing (complex) classifiers.
- Often used in real tasks like natural language processing.

Disadvantage

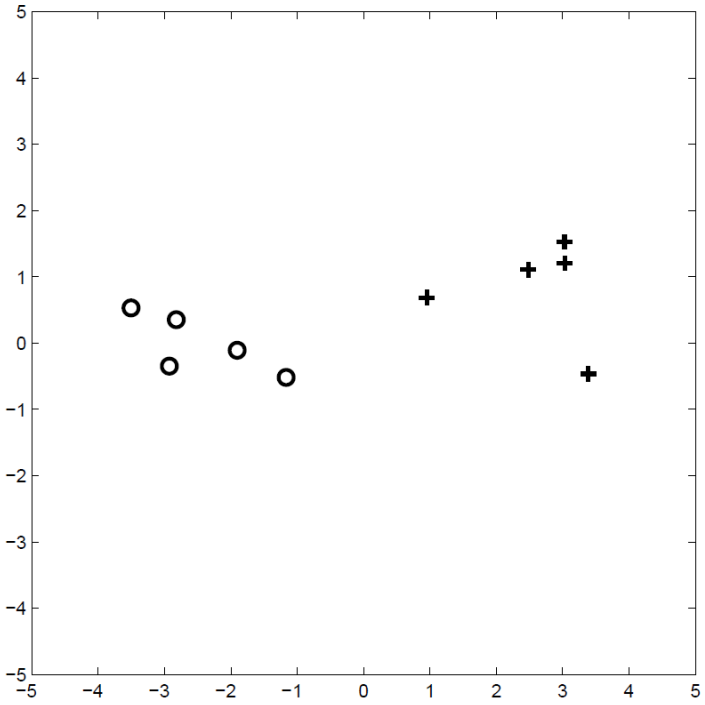
- **Early mistakes** could reinforce themselves
- Cannot say too much in terms of **convergence**.

http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

4.2 Generative Models: An Example of Gaussian Mixture Model(GMM)

Labeled data:



Assuming each class has a Gaussian distribution, what is the decision boundary?

http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

4.2 Generative Models: An Example of Gaussian Mixture Model(GMM)

Data: $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{l}$

Model parameters: $\theta = \{\mu_1, \mu_2, \Sigma_1, \Sigma_2\}$

Given training data $\mathcal{D}$, the **maximum likelihood estimate(MLE)** is

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, p(\mathcal{D}|\theta) = \underset{\theta}{\operatorname{argmax}}\, log\ p(\mathcal{D}|\theta)$$

$$\log p(D|\theta) = \sum_{i=1}^{l} \log p(y_i|\theta) p(\boldsymbol{x}_i|y_i, \theta)$$

Finding an MLE is an optimization problem to maximize the log likelihood. In supervised learning, the optimization problem is often straightforward and yields intuitive MLE solutions.

Classification: $p(y|x_{new}, \theta) = \frac{p(x_{new}, y|\theta)}{\Sigma_{y'}\, p(x_{new}, y'|\theta)}$

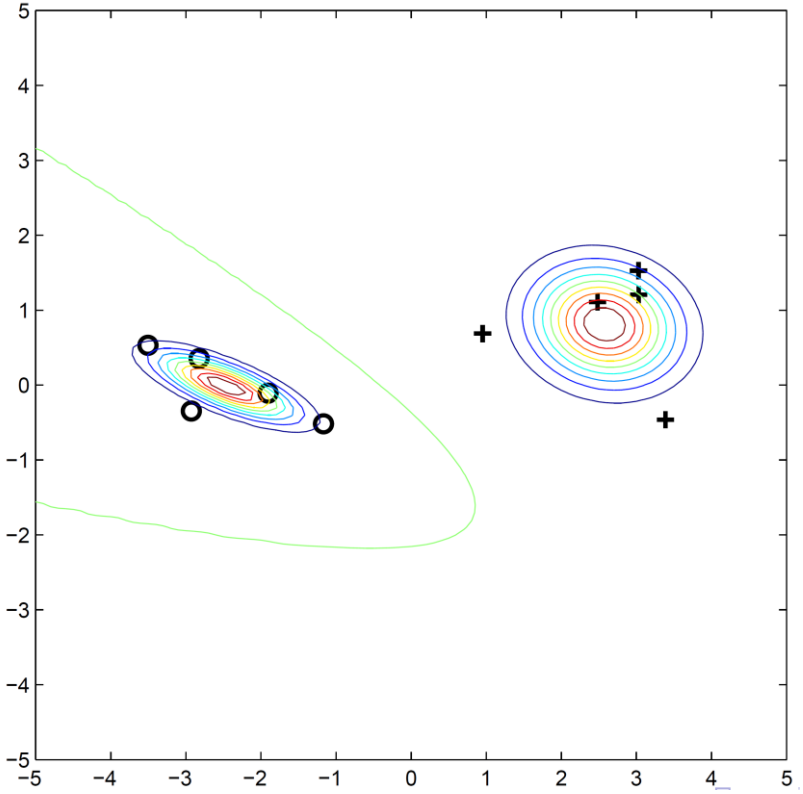
[1] http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

[2] Zhu X, Goldberg A B. Introduction to semi-supervised learning[J]. Synthesis lectures on artificial intelligence and machine learning, 2009, 3(1): 1-130.

Semi-supervised Learning

4.2 Generative Models: An Example of Gaussian Mixture Model(GMM)

The most likely model, and its decision boundary

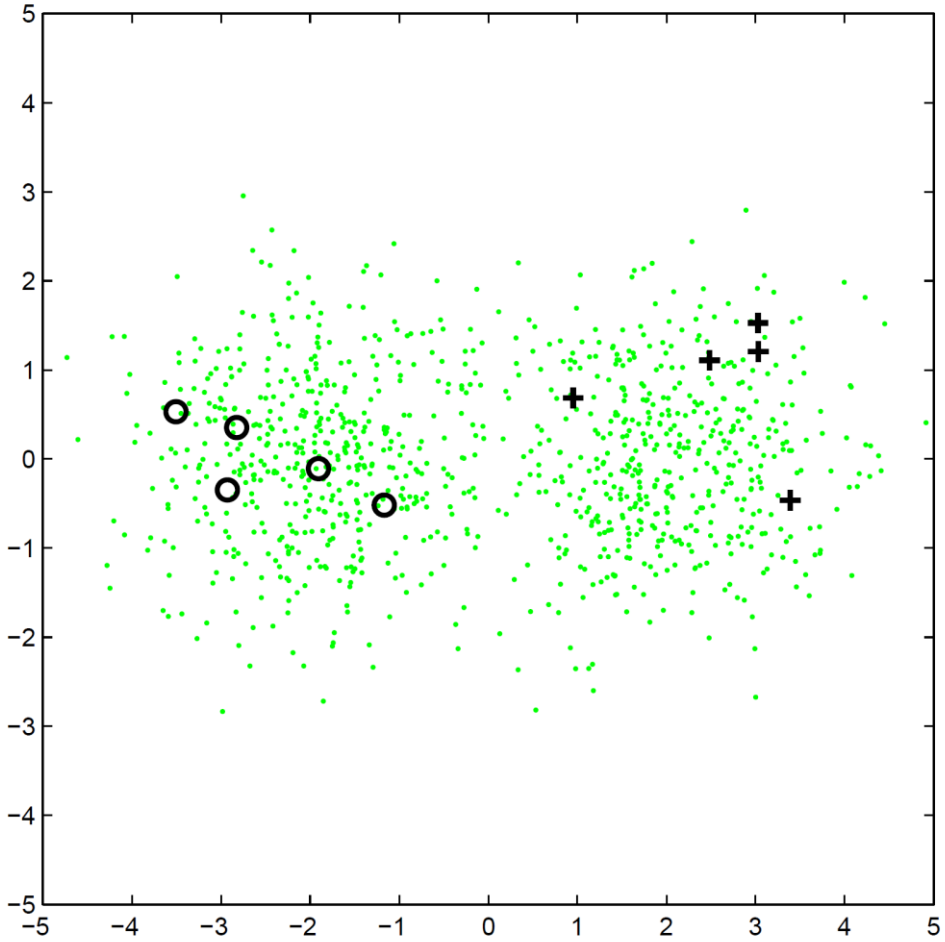


http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

4.2 Generative Models: An Example of Gaussian Mixture Model(GMM)

Adding unlabeled data:



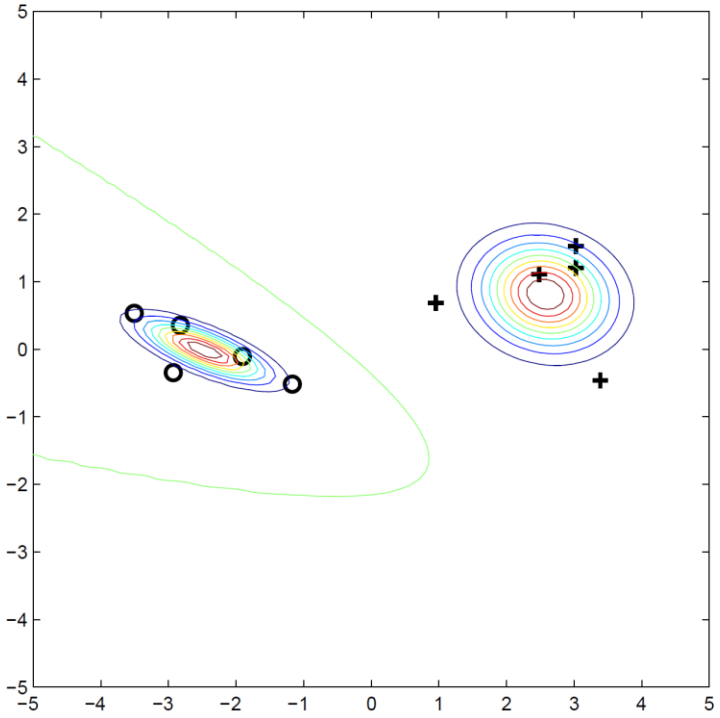
http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

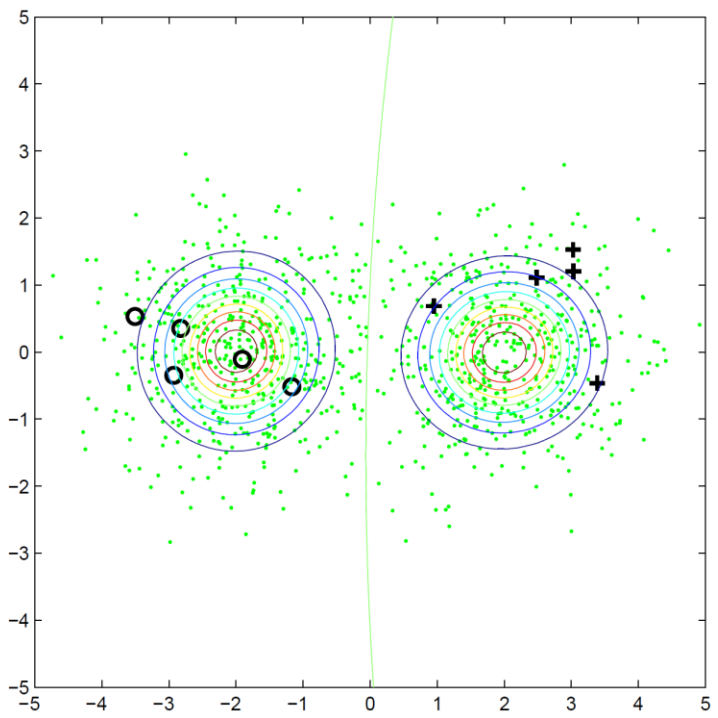
4.2 Generative Models: An Example of Gaussian Mixture Model(GMM)

They are different because they maximize different quantities.

$p(X_l, Y_l|\theta)$



$p(X_l, Y_l, X_u|\theta)$



http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

4.2 Generative Models: An Example of Gaussian Mixture Model(GMM)

Labeled data only

Data: $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{l}$

Model parameters: $\theta = \{\mu_1, \mu_2, \Sigma_1, \Sigma_2\}$

The log likelihood function: $\log p(D|\theta) = \sum_{i=1}^{l} \log p(y_i|\theta) p(\boldsymbol{x}_i|y_i, \theta)$

Labeled and unlabeled data

Data: $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_l, y_l), (\boldsymbol{x}_{l+1}, y_{l+1}), \dots, (\boldsymbol{x}_{l+u}, y_{l+u})\}$

Model parameters: $\theta = \{\mu_1, \mu_2, \Sigma_1, \Sigma_2\}$

The log likelihood function:

$$\log p(\mathcal{D}|\theta) = \sum_{i=1}^{l} \log p(y_i|\theta) p(\boldsymbol{x_i}|y_i, \theta) + \lambda \sum_{i=l+1}^{l+u} \log p(\boldsymbol{x}_i|\theta)$$

$$p(\boldsymbol{x}|\theta) = \sum_{y=1}^{C} p(\boldsymbol{x}, y|\theta) = \sum_{y=1}^{C} p(y|\theta) p(\boldsymbol{x}|y, \theta)$$

- The unlabeled data can make the log likelihood **non-concave** and hard to optimize.
- The **Expectation-Maximization (EM)** algorithm is one method to find a local optimum.

[1] http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

[2] Zhu X, Goldberg A B. Introduction to semi-supervised learning[J]. Synthesis lectures on artificial intelligence and machine learning, 2009, 3(1): 1-130.

Semi-supervised Learning

4.2 Generative Models: An Example of Gaussian Mixture Model(GMM)

Expectation-Maximization (EM) algorithm for GMM

Start from MLE on labeled data and get initial $\{\mu, \Sigma\}$, repeat:

- Expectation Step: compute the expected label $p(y|x,\theta) = \frac{p(x,y|\theta)}{\sum_{y'} p(x,y'|\theta)}$ for all unlabeled data x. Label them with the highest probability.
- Maximization Step: Update $\{\mu, \Sigma\}$

http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

4.2 Generative Models: Popular Models

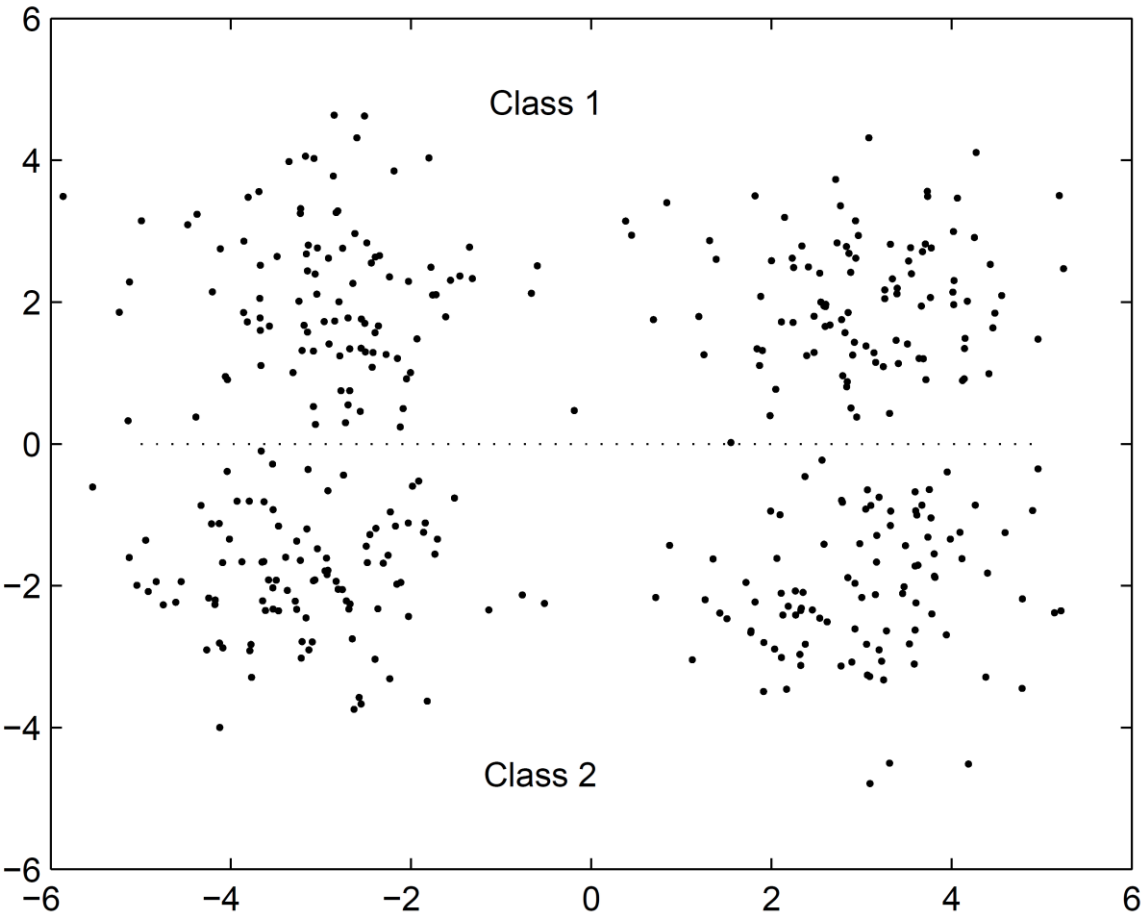
- Mixture of Gaussian distributions (GMM)
  - image classification
  - the EM algorithm
- Mixture of multinomial distributions (Naive Bayes)
  - text categorization
  - the EM algorithm
- Hidden Markov Models (HMM)
  - speech recognition
  - Baum-Welch algorithm

http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

4.2 Generative Models: Advantages and Disadvantages

Unlabeled data may hurt if generative model is wrong: example

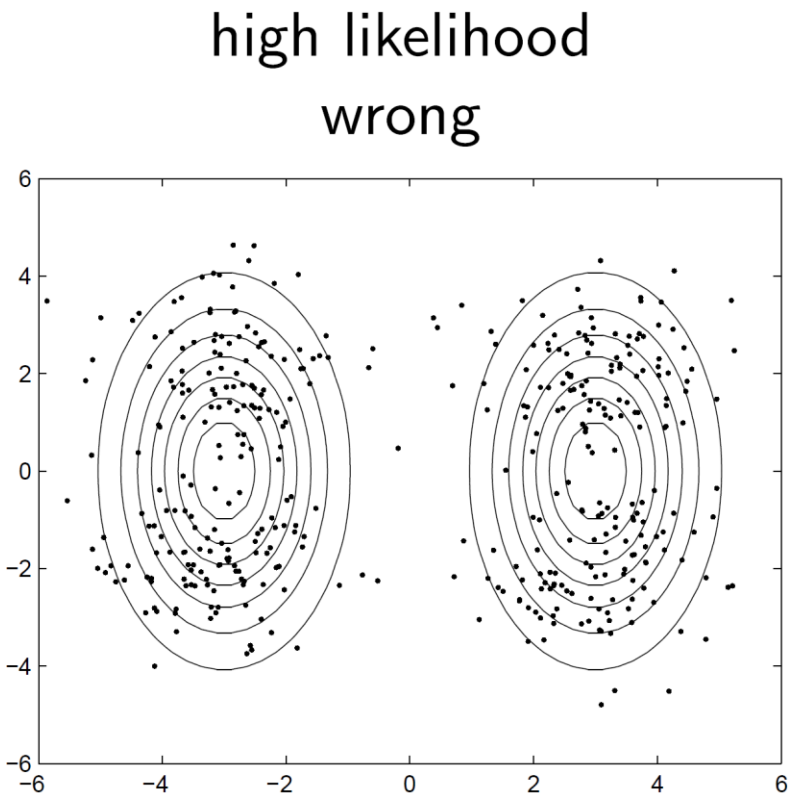


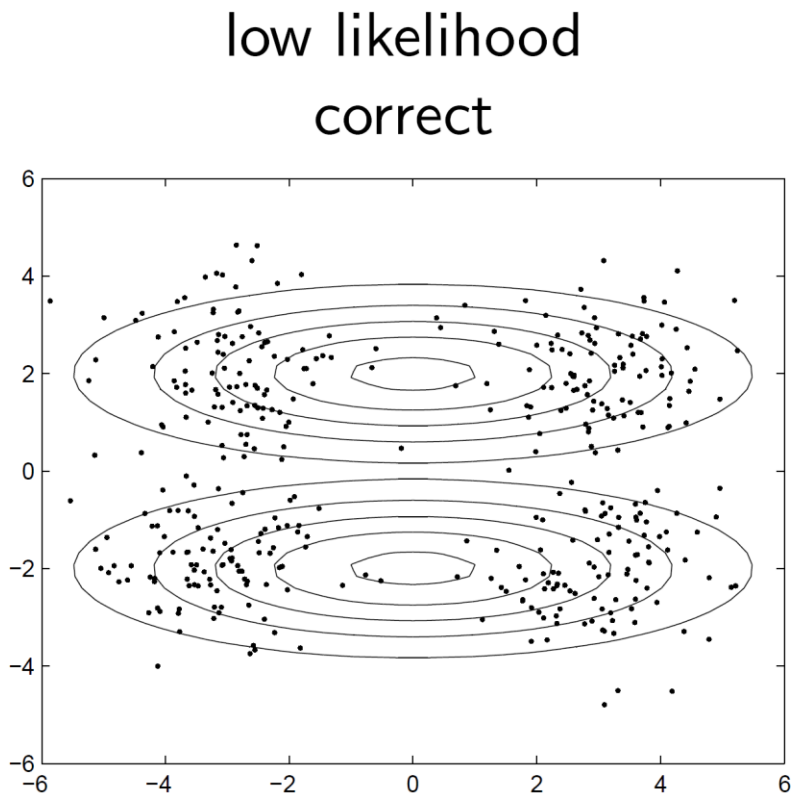
http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

4.2 Generative Models: Advantages and Disadvantages

Unlabeled data may hurt if generative model is wrong: example





http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

4.2 Generative Models: Advantages and Disadvantages

Advantages

- Clear, well-studied probabilistic framework
- Can be extremely effective, if the model is close to correct

Disadvantages

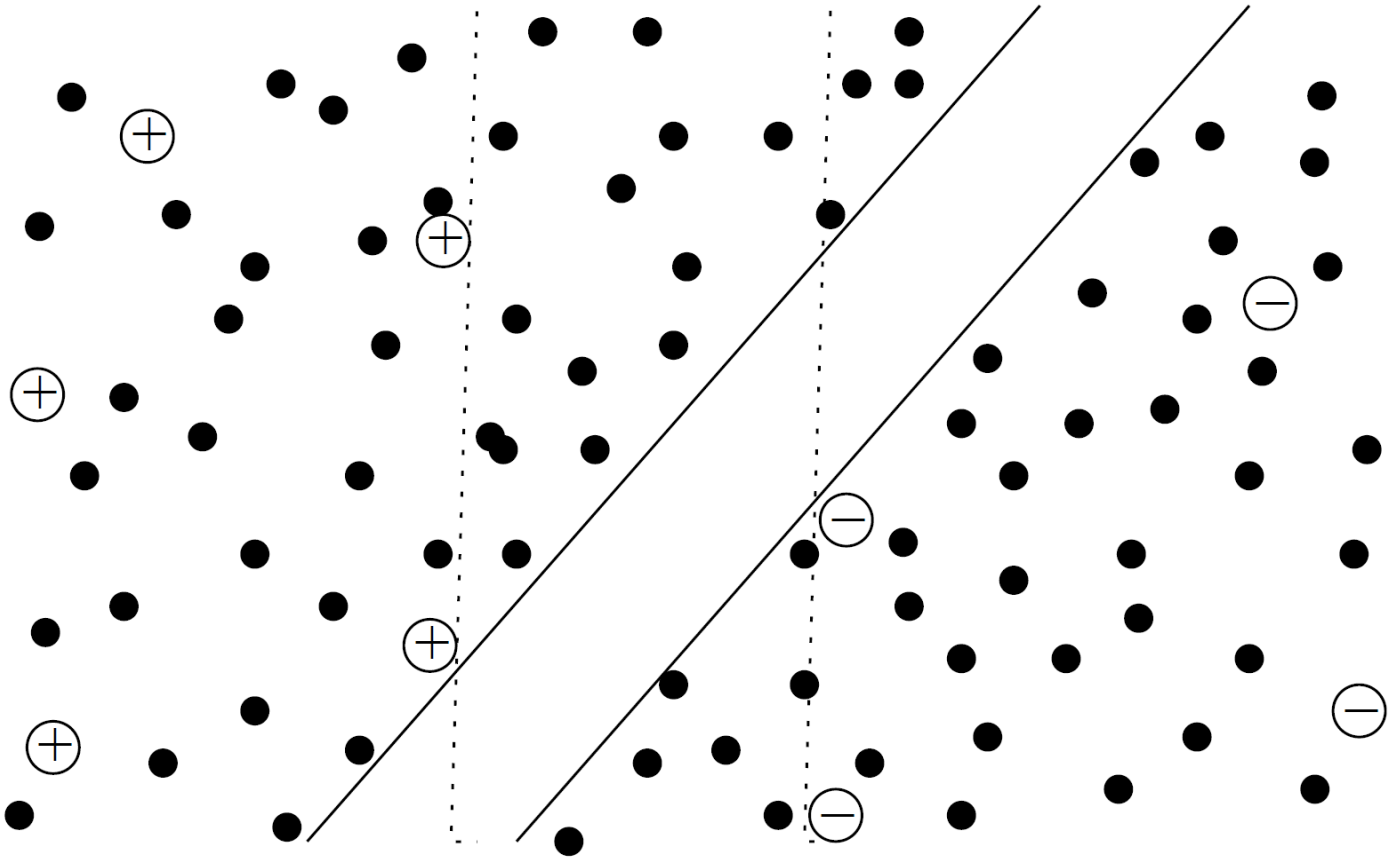
- Often difficult to verify the correctness of the model
- Model identifiability
- EM local optima
- Unlabeled data may hurt if generative model is wrong

http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

4.3 Transductive SVMs: Introduction

Basic idea: Maximizes “unlabeled data margin”



http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

4.3 Transductive SVMs: Introduction

Assumption

Unlabeled data from different classes are separated with large margin.

Transductive SVMs idea

- Enumerate all 2^u possible labeling of X_u
- Build one standard SVM for each labeling (and X_l)
- Pick the SVM with the largest margin

http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

4.3 Transductive SVMs: Standard SVM review

- Given a training set of instance-label pairs $(\boldsymbol{x}^{(i)}, y^{(i)})$, $i = 1, \ldots, \mathrm{m}$, where $\boldsymbol{x}^{(i)} \in R^n$ and $y^{(i)} \in \{1, -1\}$
- For $\boldsymbol{x}^{(i)}$, Calculate the m-dimensional $\boldsymbol{f}^{(i)}$**:**

$$f_1{}^{(i)} = K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(1)}), f_2{}^{(i)} = K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(2)}), \ldots, f_m{}^{(i)} = K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(m)})$$

- the SVM require the solution of the following optimization problem:

$$\min_{\boldsymbol{\alpha},\mathrm{b},\xi} \frac{1}{2}||\boldsymbol{\alpha}||^2 + \mathrm{C}\sum_{\mathrm{i}=1}^{\mathrm{m}} \xi_i$$

$$\text{subject to } y^{(i)}\left(\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{f}^{(i)} + \mathrm{b}\right) \geq 1\text{-}\xi_i, \text{ where } \mathrm{i} = 1,2, \ldots, \mathrm{m}$$

$$\text{and } \xi_i \geq 0$$

- Need to specify choice of penalty parameter C and choice of kernel before optimization.
- SVM finds a linear separating hyperplane with the maximal margin in this higher dimensional space.

Semi-supervised Learning

4.3 Transductive SVMs: Problem Formulation

Add the objective statement

$$\min_{\boldsymbol{\alpha},b,\xi} \frac{1}{2}||\boldsymbol{\alpha}||^2 + C\sum_{i=1}^{l}\xi_i + C'\sum_{i=l+1}^{l+u}\xi_i$$

Add the constraint

Directly optimizing the S3VM objective often produces unbalanced classification: most points fall in one class.

Heuristic class balance:

$$s.t.\ \frac{1}{l}\sum_{i=1}^{l} y^{(i)} = \frac{1}{u-l}\sum_{i=l+1}^{l+u} y^{(i)}$$

Relaxed class balancing constraint:

$$s.t.\ \frac{1}{l}\sum_{i=1}^{l} y^{(i)} = \frac{1}{u-l}\sum_{i=l+1}^{l+u}(\boldsymbol{\alpha}^T \boldsymbol{f}^{(i)} + b)$$

http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

4.3 Transductive SVMs: Problem Formulation

- Given a training set of instance-label pairs $(\boldsymbol{x}^{(i)}, y^{(i)})$, $i = 1, \ldots, l$, where $\boldsymbol{x}^{(i)} \in R^n$ and $y^{(i)} \in \{1, -1\}$, and unlabeled points $\boldsymbol{x}^{(i)}$, $i = l+1, \ldots, l+u$
- For $\boldsymbol{x}^{(i)}$, Calculate the m-dimensional $\boldsymbol{f}^{(i)}$**:**

$$f_1{}^{(i)} = K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(1)}), f_2{}^{(i)} = K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(2)}), \ldots, f_u{}^{(i)} = K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(l+u)})$$

- the SVM require the solution of the following optimization problem:

$$\min_{\boldsymbol{\alpha}, \mathrm{b}, \xi} \frac{1}{2}||\boldsymbol{\alpha}||^2 + \mathrm{C}\sum_{\mathrm{i}=1}^{l} \xi_i + \mathrm{C}' \sum_{\mathrm{i}=l+1}^{u} \xi_i$$

subject to $y^{(i)}\left(\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{f}^{(i)} + \mathrm{b}\right) \geq 1\text{-}\xi_i$, where $\mathrm{i} = 1,2, \ldots, l$

And $|\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{f}^{(i)} + \mathrm{b}| \geq 1\text{-}\xi_i$, where $\mathrm{i} = l+1, \ldots, l+u$

And $\frac{1}{l}\sum_{i=1}^{l} y^{(i)} = \frac{1}{u-l}\sum_{i=l+1}^{u}(\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{f}^{(i)} + \mathrm{b})$

Semi-supervised Learning

4.3 Transductive SVMs: Optimization challenge

Finding a solution for semi-supervised SVM is difficult. The current optimization method:

- $\text{SVM}^{\text{light}}$
- ∇S3VM
- Continuation method
- Concave-Convex Procedure
- Branch and Bound

http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

4.3 Transductive SVMs: Algorithm

SVM$^{\text{light}}$

- Train SVM with labeled data
- Sort unlabeled data by calculating $\alpha^T f^{(i)} + b$. Label y=1, -1 for the appropriate portions
- For $l + 1 \leq i, j \leq u$
  - If exchange the label of $x^{(i)}$ and $x^{(j)}$, the objective function can be reduced then exchange.
  - Until no labels switchable

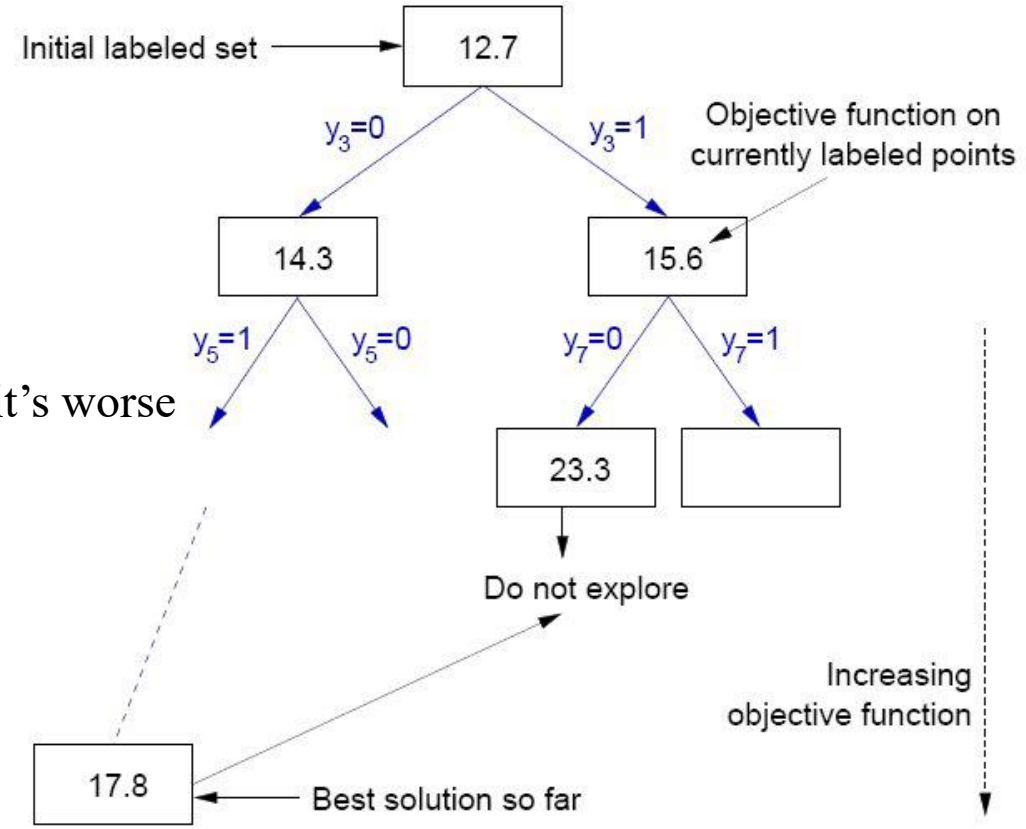
http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Branch and Bound

Branch and Bound finds the exact global solution

Unfortunately it can only handle a few hundred unlabeled points

- Depth-first search on the tree
- Keep the best complete objective so far
- Prune internal node (and its subtree) if it's worse than the best objective



http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

4.3 Transductive SVMs: Advantages and Disadvantages

Advantages

- Applicable wherever SVMs are applicable
- Clear mathematical framework

Disadvantages

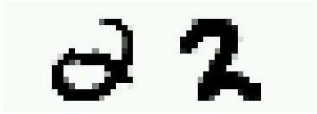
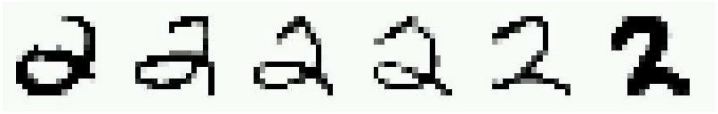
- Optimization difficult
- Can be trapped in bad local optima
- More modest assumption than generative model or graph-based methods, potentially lesser gain

http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

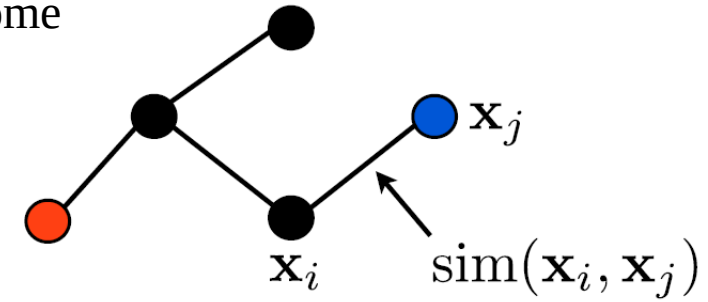
Semi-supervised Learning

4.4 Graph-based Algorithm: Introduction

Graph based approaches exploit the property of label smoothness

	
not similar	'indirectly' similar with stepping stones

- Represent each example (labeled/unlabeled) as vertices of some graph
- The labels should vary smoothly along the graph
- Nearby vertices should have similar labels



[1] https://www.cs.utah.edu/~piyush/teaching/8-11-slides.pdf

[2] http://graph-ssl.wdfiles.com/local--files/blog%3A_start/graph_ssl_acl12_tutorial_slides_final.pdf

Semi-supervised Learning

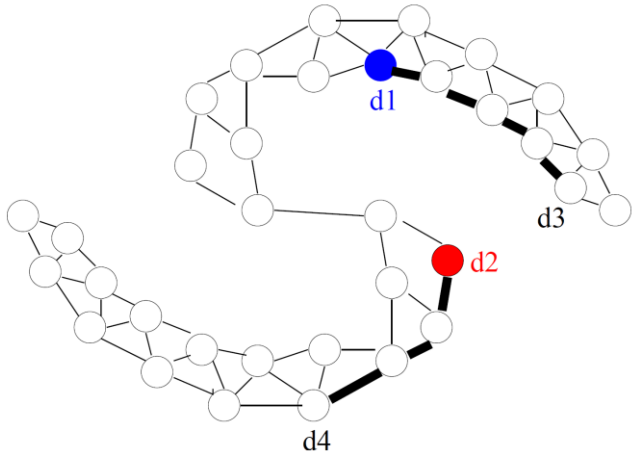
4.4 Graph-based Algorithm: Introduction

Assumption

A graph is given on the labeled and unlabeled data. Instances connected by heavy edge tend to have the same label.

Two stages

- Graph construction
- Label Inference



[1] https://www.cs.utah.edu/~piyush/teaching/8-11-slides.pdf

[2] http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

[3] http://graph-ssl.wdfiles.com/local--files/blog%3A_start/graph_ssl_acl12_tutorial_slides_final.pdf

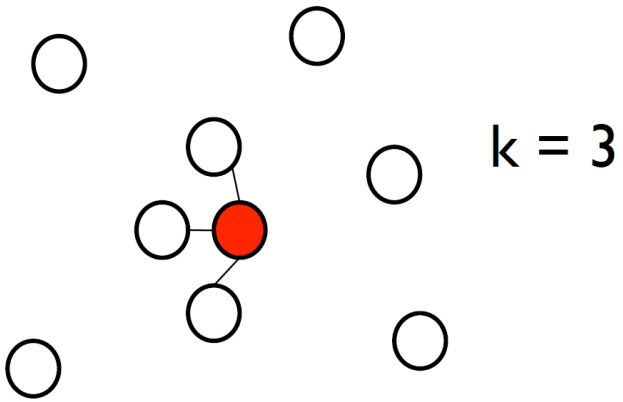
Semi-supervised Learning

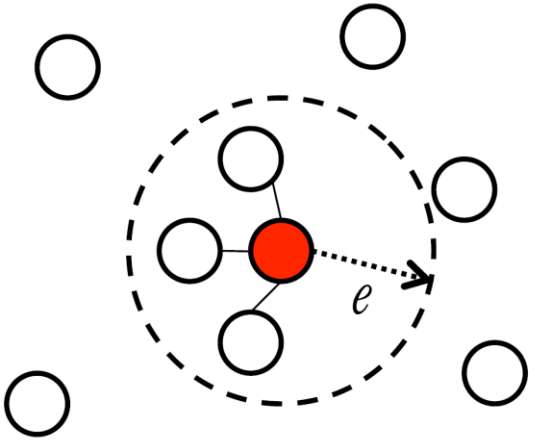
4.4 Graph-based Algorithm: Graph construction

Graph construction

- k-nearest-neighbor graph: add edges between an instance and its k-nearest neighbors (0, 1 weights)
- e-Neighborhood: add edges to all instances inside a ball of radius e
- fully connected graph, weight decays with distance

$$w = \exp\left(-\frac{\left\|x_i - x_j\right\|^2}{\sigma^2}\right)$$





[1] https://www.cs.utah.edu/~piyush/teaching/8-11-slides.pdf

[2] http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

[3] http://graph-ssl.wdfiles.com/local--files/blog%3A_start/graph_ssl_acl12_tutorial_slides_final.pdf

Semi-supervised Learning

4.4 Graph-based Algorithm: Algorithm

Some graph-based algorithms

- mincut
- harmonic
- local and global consistency
- manifold regularization

http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

4.4 Graph-based Algorithm: Algorithm

Harmonic

- Initially, set $f(x_i) = y_i$ for i=1,2,…,l, and $f(x_j)$ arbitrarily (e.g., 0) for $x_j \in X_u$
- Repeat until convergence: Set $f(x_i) = \frac{\sum_{j \sim i} w_{ij} f(x_j)}{\sum_{j \sim i} w_{ij}}$, $\forall x_i \in X_u$, i.e., the average of the neighbors. Note $f(X_l)$ is fixed.

http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Problems with Harmonic

- It fixes the given labels Y_l
  - What if some labels are wrong?
  - Want to be flexible and disagree with given labels occasionally
  - It cannot handle new test points directly

- f is only defined on X_u
  - We have to add new test points to the graph, and find a new harmonic solution

Manifold Regularization

- Input: Kernel K, parameters λ_1, λ_2, (X_l, Y_l), X_u
- Assume the predictions on the entire data $\mathcal{L} \cup \mathcal{U}$ to be defined by function f, $f(x) = \sum_{i=1}^{l} \alpha_i K(x_i, x)$
- Solve the optimization problem

$$\min_f \sum_{i \in \mathcal{L}} V(y_i, f(x_i)) + \lambda_1 \sum_{i,j \in \mathcal{L},\mathcal{U}} w_{ij}(f_i - f_j)^2 + \lambda_2 ||f||_K^2$$

minimize the loss on labeled data

ensure smoothness of labels of labeled and unlabeled data

minimize the norm of the function

- Classify a new test point x by sign of calculating $f(x)$

[1] https://www.cs.utah.edu/~piyush/teaching/8-11-slides.pdf

Manifold Regularization

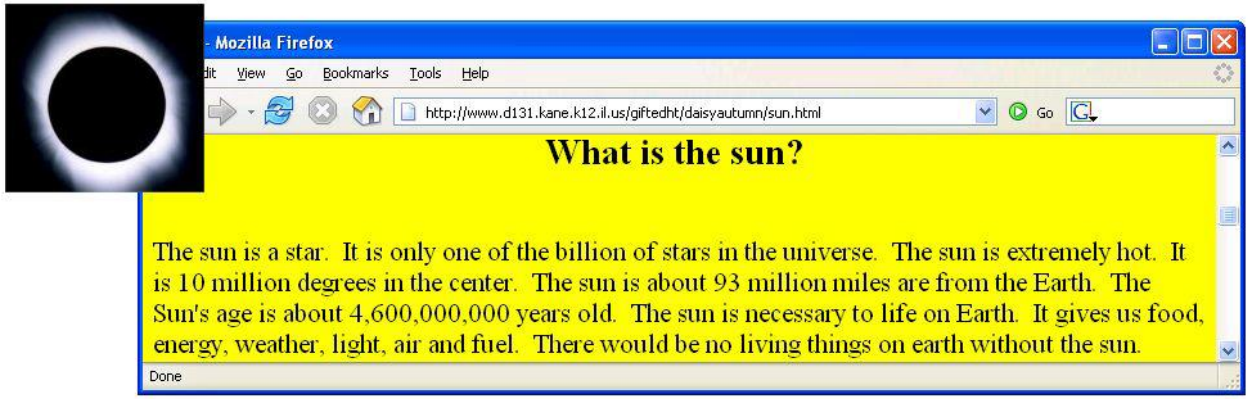
- Allows but penalizes $f(X_l) \neq Y_l$
- Automatically applies to new test data

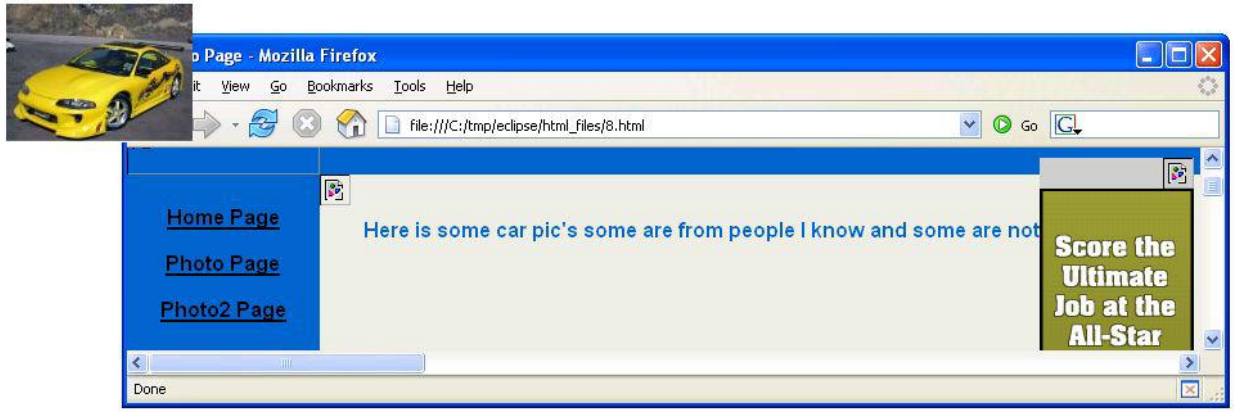
http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

4.5 Multi-view Algorithm: Co-training

Two views of an item: image and HTML text





http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

4.5 Multi-view Algorithm: Co-training

Feature Split

- Each instance is represented by two sets of features $x = [x^{(1)}; x^{(2)}]$

  $x^{(1)}$ = image features

  $x^{(2)}$ = web page text

- This is a natural feature split

Co-training idea

- Train an image classifier and a text classifier
- The two classifiers teach each other

http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

4.5 Multi-view Algorithm: Co-training

Assumption

feature split $x = [x^{(1)}; x^{(2)}]$ exists

$x^{(1)}$ or $x^{(2)}$ alone is sufficient to train a good classifier

$x^{(1)}$ and $x^{(2)}$ are conditionally independent given the class

Co-training Algorithm

- Train two classifiers: $f^{(1)}$ from $(X_l^{(1)}, Y_l)$, $f^{(2)}$ from $(X_l^{(2)}, Y_l)$
- Classify X_u with $f^{(1)}$ and $f^{(2)}$ separately.
- Add $f^{(1)}$'s k-most-confident $(x, f^{(1)}(x))$ to $f^{(2)}$'s labeled data.
- Add $f^{(2)}$'s k-most-confident $(x, f^{(2)}(x))$ to $f^{(1)}$'s labeled data.
- Repeat.
- Finally, use a voting or averaging to make predictions on the test data

http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

4.5 Multi-view Algorithm: Co-training

Pros

- Simple wrapper method. Applies to almost all existing classifiers
- Less sensitive to mistakes than self-training

Cons

- Natural feature splits may not exist
- Models using BOTH features should do better

http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Semi-supervised Learning

4.5 Multi-view Algorithm

- General Idea: Train multiple classifiers, each using a different view
- Classify unlabeled data with all classifiers
- Add majority vote label
- Co-training is a special type of multi-view learning algorithm

[1] https://www.cs.utah.edu/~piyush/teaching/8-11-slides.pdf

[2] http://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf

Part 5
Reinforcement Learning

- Background: Markov Decision Process
- Reinforcement Learning: Actor Critic
- Reinforcement Learning: Q-Learning

Reinforcement Learning

5.1 Background: Markov Decision Process

Definition

A Markov decision process is a 5-tuple $(S, A, P(\,;\cdot), R(\,;\cdot), \gamma)$, where

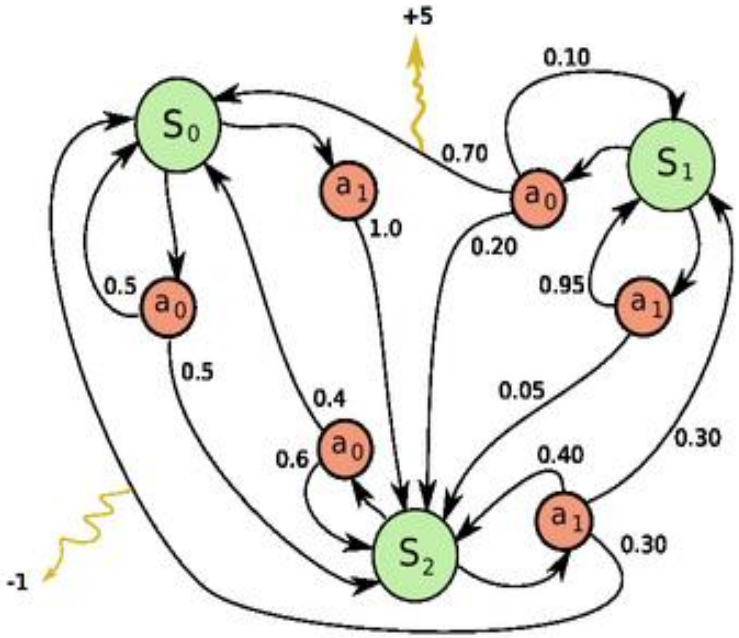
- S is a finite set of states,
- A is a finite set of actions (alternatively, A_s is the finite set of actions available from state s),
- $P_a(s, s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability that action a in state s at time t will lead to state s' at time $t + 1$,
- $R_a(s, s')$ is the immediate reward (or expected immediate reward) received after transition to state s' from state s,
- $\gamma \in [0,1]$ is the discount factor, which represents the difference in importance between future rewards and present rewards.

Note: The theory of Markov decision processes does not state that S or A are finite, but the basic algorithms below assume that they are finite.

[1] https://en.wikipedia.org/wiki/Markov_decision_process

Reinforcement Learning

5.1 Background: Markov Decision Process



Example of a simple MDP with three states and two actions.

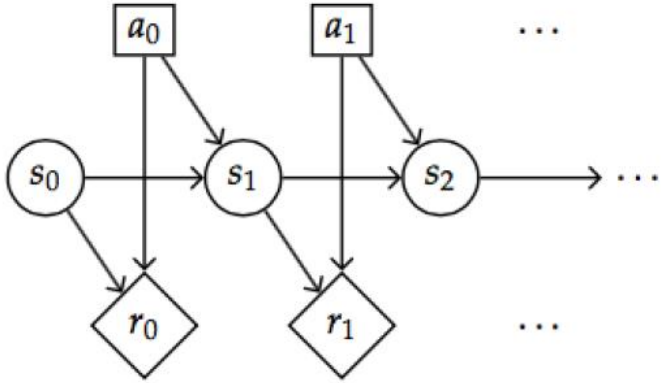


Figure: Two Step Markov Decision Process

[1] https://en.wikipedia.org/wiki/Markov_decision_process

Solutions

- MDPs can be solved by linear programming or dynamic programming.
- Reinforcement Learning is a technique of approximate dynamic programming.

[1] https://en.wikipedia.org/wiki/Markov_decision_process

[2] Valerio L, Bruno R, Passarella A. Cellular traffic offloading via opportunistic networking with reinforcement learning[J]. Computer Communications, 2015, 71: 129-141.

Reinforcement Learning

5.1 Background: Markov Decision Process

Dynamic Programming

- Suppose we know the state transition function P and the reward function R, and we wish to calculate the policy that maximizes the expected discounted reward.
- The standard family of algorithms to calculate this optimal policy requires storage for two arrays indexed by state: value V, which contains real values, and policy π which contains actions. At the end of the algorithm, π will contain the solution and $V(s)$ will contain the discounted sum of the rewards to be earned (on average) by following that solution from state s.

[1] https://en.wikipedia.org/wiki/Markov_decision_process

Dynamic Programming

- The algorithm has the following two kinds of steps, which are repeated in some order for all the states until no further changes take place. They are defined recursively as follows:

$$\pi(s) := \arg\max_a \{ \sum_{s'} P_a(s,s')(R_a(s,s') + \gamma V(s')) \}$$

$$V(s) := \sum_{s'} P_{\pi(s)}(s,s')(R_{\pi(s)}(s,s') + \gamma V(s'))$$

- Their order depends on the variant of the algorithm: value iteration and policy iteration.

[1] https://en.wikipedia.org/wiki/Markov_decision_process

Reinforcement Learning

5.1 Background: Markov Decision Process

Dynamic Programming: Value Iteration

- In value iteration (Bellman 1957), which is also called backward induction, the π function is not used; instead, the value of $\pi(s)$ is calculated within $V(s)$ whenever it is needed.

- Substituting the calculation of $\pi(s)$ into the calculation of $V(s)$ gives the combined step:

$$\mathrm{V_{i+1}}(s) := \max_a \left\{ \sum_{s'} P_a(s,s') \left(R_a(s,s') + \gamma V_i(s') \right) \right\},$$

where i is the iteration number.

- Value iteration starts at $i = 0$ and V_0 as a guess of the value function. It then iterates, repeatedly computing V_{i+1} for all states s, until V converges with the left-hand side equal to the right-hand side (which is the "Bellman equation" for this problem).

[1] https://en.wikipedia.org/wiki/Markov_decision_process

Dynamic Programming: Policy Iteration

- The policy iteration algorithm proceeds as follows:

  Step1: Initialize π randomly.

  Step2: Repeat Step 2.1 2.2 until convergence

  --Step 2.1 Let $V := V^{\pi}$.

  --Step 2.2 For each state s, let $\pi(s) := arg\max_{a} \sum_{s'} P_a(s,s')V(s')$.

- Thus, the inner-loop repeatedly computes the value function for the current policy, and then updates the policy using the current value function. (The policy π found in step (b) is also called the policy that is greedy with respect to V .) Note that step (a) can be done via solving Bellman's equations, which in the case of a fixed policy, is just a set of $|S|$ linear equations in $|S|$ variables.
- After at most a finite number of iterations of this algorithm, V will converge to V^*, and π will converge to π^*.

Source *Andrew Ng, Stanford CS229 Lecture notes, Reinforcement Learning and Control, https://see.stanford.edu/materials/aimlcs229/cs229-notes12.pdf*

Reinforcement Learning

5.1 Background: Markov Decision Process

Dynamic Programming: Value Iteration and Policy Iteration

- Both value iteration and policy iteration are standard algorithms for solving MDPs, and there isn't currently universal agreement over which algorithm is better.
- For small MDPs, policy iteration is often very fast and converges with very few iterations.
- However, for MDPs with large state spaces, solving for V^{π} explicitly would involve solving a large system of linear equations, and could be difficult. In these problems, value iteration maybe preferred. For this reason, in practice value iteration seems to be used more often than policy iteration.

Source *https://en.wikipedia.org/wiki/Markov_decision_process*

Reinforcement Learning

5.2 Reinforcement Learning

- MDPs can be solved by linear programming or dynamic programming.
- Reinforcement Learning is a technique of approximate dynamic programming.
- The main difference between the classical techniques and reinforcement learning algorithms is that the latter do not need knowledge about the MDP and they target large MDPs where exact methods become infeasible.
- RL methods are known to find very good approximations of the optimal policy. Two well known approaches of Reinforcement Learning: Actor–Critic Learning and Q-Learning.

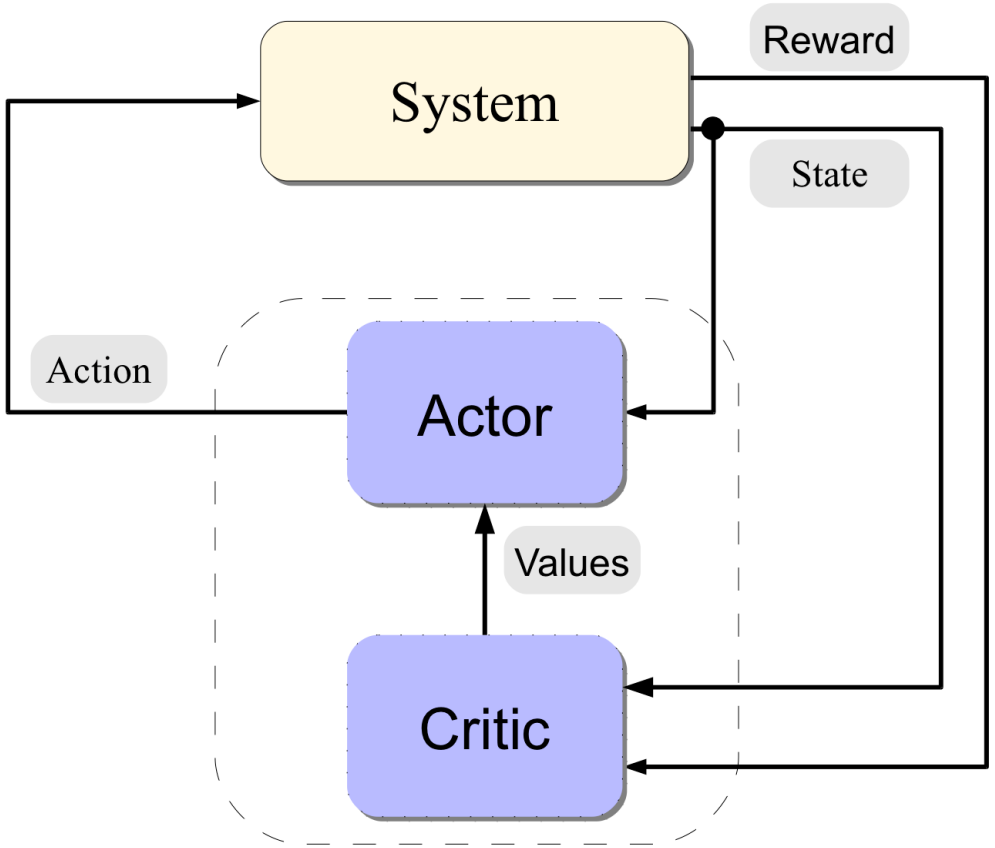
[1] https://en.wikipedia.org/wiki/Markov_decision_process

[2] Valerio L, Bruno R, Passarella A. Cellular traffic offloading via opportunistic networking with reinforcement learning[J]. Computer Communications, 2015, 71: 129-141.

Reinforcement Learning

5.2.1 Reinforcement Learning: Actor Critic

- The Actor corresponds to an action-selection policy that maps states to actions in a probabilistic manner
- The Critic is a function predictor that during time learns the value of a state in terms of the expected cumulative rewards



[0] Valerio L, Bruno R, Passarella A. Cellular traffic offloading via opportunistic networking with reinforcement learning[J]. Computer Communications, 2015, 71: 129-141.

Reinforcement Learning

5.2.1 Reinforcement Learning: Actor Critic

➢ Critic

The following example of Critic is designed by exploiting a temporal-difference (TD) learning method. TD learning is a well-known incremental method introduced by Sutton to estimate the expected return for a given state.

When the transition from the state s_t to the state s_{t+1} and the reward $R_{a_t}(s_t, s_{t+1})$ are obtained, the estimation of the value of the state s_t is updated in the following way:

$$\hat{V}^{\pi}_{new}(s_t) = \hat{V}^{\pi}_{old}(s_t) + \alpha(R_{a_t}(s_t, s_{t+1}) + \gamma \hat{V}^{\pi}_{old}(x_{t+1}) - \hat{V}^{\pi}_{old}(x_t))$$

where α is a discount parameter.

[0] Valerio L, Bruno R, Passarella A. Cellular traffic offloading via opportunistic networking with reinforcement learning[J]. Computer Communications, 2015, 71: 129-141.

Reinforcement Learning

5.2.1 Reinforcement Learning: Actor Critic

- Actor

The Actor chooses actions in a probabilistic (rather than deterministic) way.

The following shows the Gibbs Softmax method, a well known approach in RL for action selection.

For each state s_t, the policy function (i.e., the probability of selecting action a_t) is given by:

$$\pi(s_t, a_t) = \frac{e^{p(s_t,a_t)}}{\sum_{b_t \in A} e^{p(s_t,b_t)}}$$

$p(s_t, a_t)$ is preference function.

Update the preference function:

$$p_{new}(s_t, a_t) = p_{old}(s_t, a_t) + \beta(R_{a_t}(s_t, s_{t+1}) + \gamma \hat{V}^{\pi}(s_{t+1}) - \hat{V}^{\pi}_{old}(s_t))$$

where $0 < \beta \leq 1$.

[0] Valerio L, Bruno R, Passarella A. Cellular traffic offloading via opportunistic networking with reinforcement learning[J]. Computer Communications, 2015, 71: 129-141.

Reinforcement Learning

5.2.1 Reinforcement Learning: Actor Critic

Summary

- At time t the system is in the state s_t.
- After that, the Actor draws and executes an action a_t according to the policy function, and the new state s_{t+1} and the reward value $R_{a_t}(s_t, s_{t+1})$ are observed.
- The Critic updates the estimation of the value function $V^{\pi}(s_t)$.
- Finally, the policy for state s_t is updated.
- This procedure is repeated every time step.

[0] Valerio L, Bruno R, Passarella A. Cellular traffic offloading via opportunistic networking with reinforcement learning[J]. Computer Communications, 2015, 71: 129-141.

Reinforcement Learning

5.2.2 Reinforcement Learning: Q-Learning

- Q-learning is a model-free reinforcement learning technique based on value iteration
- Q-Learning evaluates the pair state action $Q(s, a)$
- Formally, after every state transition, the value for the state-action pair is updated as follows:

$$Q_{new}(s_t, a_t) \leftarrow \underbrace{Q_{old}(s_t, a_t)}_{old\ value} + \alpha \cdot (\overbrace{\underbrace{R_{a_t}(s_t, s_{t+1})}_{rward} + \underbrace{\gamma}_{discount\ factor} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{estimate\ of\ optimal\ future\ value}}^{learned\ value} - \underbrace{Q_{old}(s_t, a_t)}_{old\ value})$$

where $0 < \alpha, \gamma < 1$ are discount parameters.

[0] Valerio L, Bruno R, Passarella A. Cellular traffic offloading via opportunistic networking with reinforcement learning[J]. Computer Communications, 2015, 71: 129-141.

Reinforcement Learning

5.2.2 Reinforcement Learning: Q-Learning

Q-Learning algorithm can be used with two different action selection policies: ε-greedy and Softmax

ε-greedy

Let be U a uniform continuous random variable and u_t a sample drawn at time t.

$$\pi(s_{t-1}) = \begin{cases} draw\ random\ action\ \ if\ u_t < \varepsilon \\ \underset{a}{\operatorname{argmax}}\, Q(s_t, a)\ \ if\ u_t \geq \varepsilon \end{cases}$$

Softmax

$$\pi(s_t, a_t) = \frac{e^{Q(s_t, a_t)}}{\sum_{b_t \in A} e^{Q(s_t, b_t)}}$$

[0] Valerio L, Bruno R, Passarella A. Cellular traffic offloading via opportunistic networking with reinforcement learning[J]. Computer Communications, 2015, 71: 129-141.

Reinforcement Learning

5.2.2 Reinforcement Learning: Q-Learning

Summary

- At time t the system is in the state s_t.
- After having drawn and executed the action a_t according to policy, the new state s_{t+1} and the reward value $R_{a_t}(s_t, s_{t+1})$ are observed.
- Then the action-value for state s_t are updated.
- This procedure is repeated every time step.

[0] Valerio L, Bruno R, Passarella A. Cellular traffic offloading via opportunistic networking with reinforcement learning[J]. Computer Communications, 2015, 71: 129-141.

Convergence Theorem

Given a finite MDP(S, A, P, R), the Q-learning algorithm, given by the update rule

$$Q_{new}(s_t, a_t) \leftarrow \underbrace{Q_{old}(s_t, a_t)}_{old\ value} + \alpha_t(s_t, a_t) \cdot (\overbrace{\underbrace{R_{a_t}(s_t, s_{t+1})}_{rward} + \underbrace{\gamma}_{discount\ factor} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{estimate\ of\ optimal\ future\ value}}^{learned\ value} - \underbrace{Q_{old}(s_t, a_t)}_{old\ value})$$

converges w.p.1 to the optimal Q-function as long as

$$\sum_t \alpha_t(s, a) = \infty \quad \sum_t \alpha_t^2(s, a) < \infty$$

For all $(s, a) \epsilon\ S \times A$

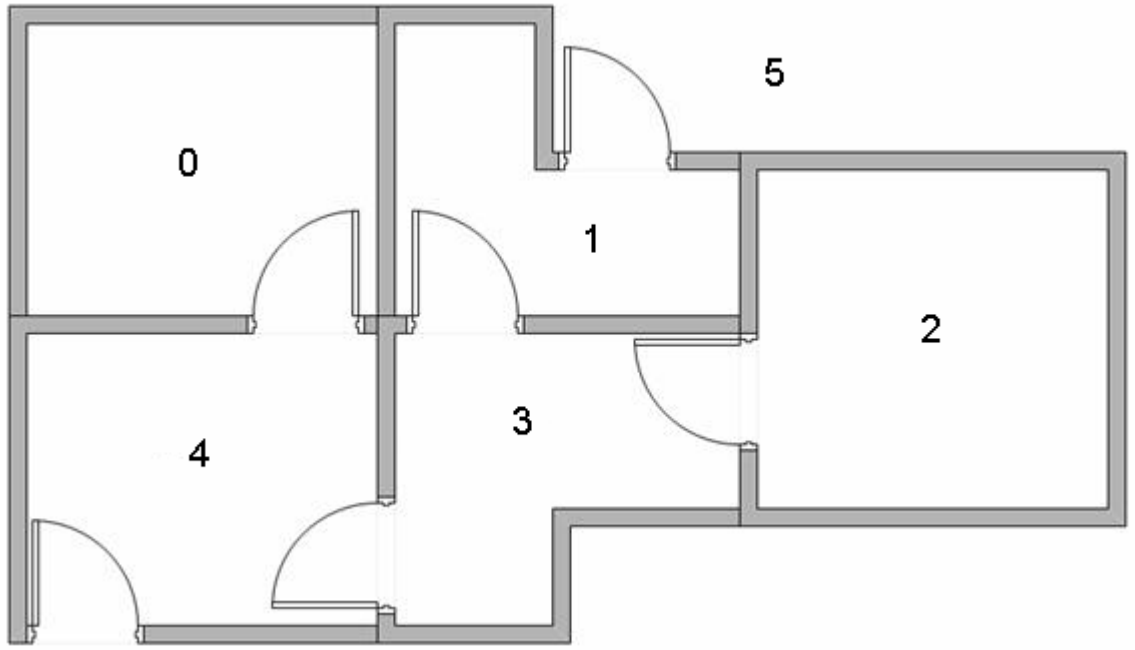
In fully deterministic environments, a learning rate of $\alpha_t(s_t, a_t)$=1 is optimal. When the problem is stochastic, the algorithm still converges under some technical conditions on the learning rate, that require it to decrease to zero.[2]

[1] Melo F S. Convergence of Q-learning: A simple proof[J]. Institute Of Systems and Robotics, Tech. Rep, 2001.
[2] https://en.wikipedia.org/wiki/Q-learning#Learning_rate

Reinforcement Learning

5.2.2 Reinforcement Learning: Q-Learning

Example

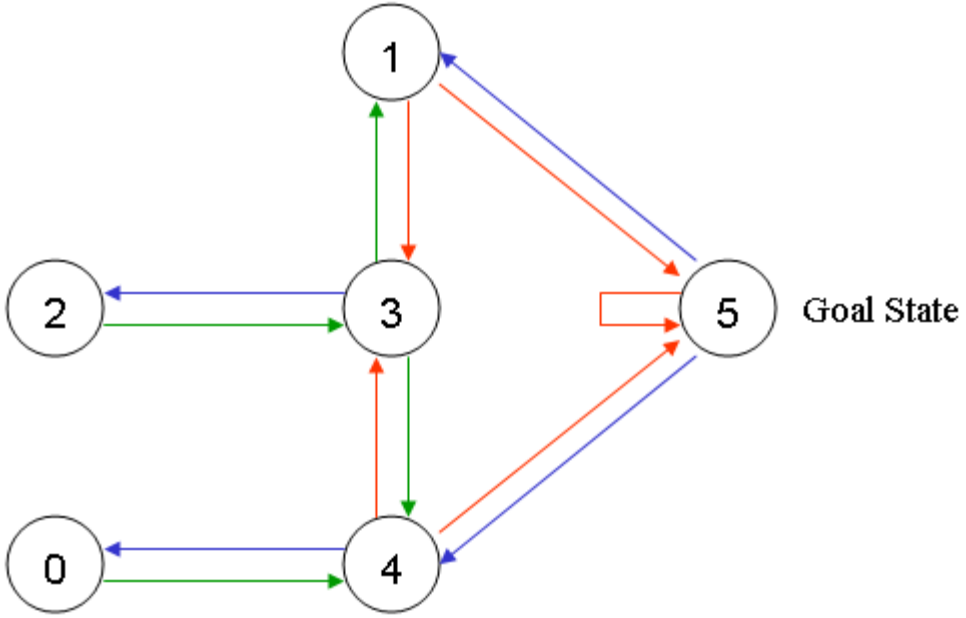


- Suppose we have 5 rooms in a building connected by doors as shown in the figure below.
- We'll number each room 0 through 4.
- The outside of the building can be thought of as one big room (5).
- Notice that doors 1 and 4 lead into the building from room 5 (outside).

Source *http://mnemstudio.org/path-finding-q-learning-tutorial.htm*

Example

We can represent the rooms on a graph, each room as a node, and each door as a link.



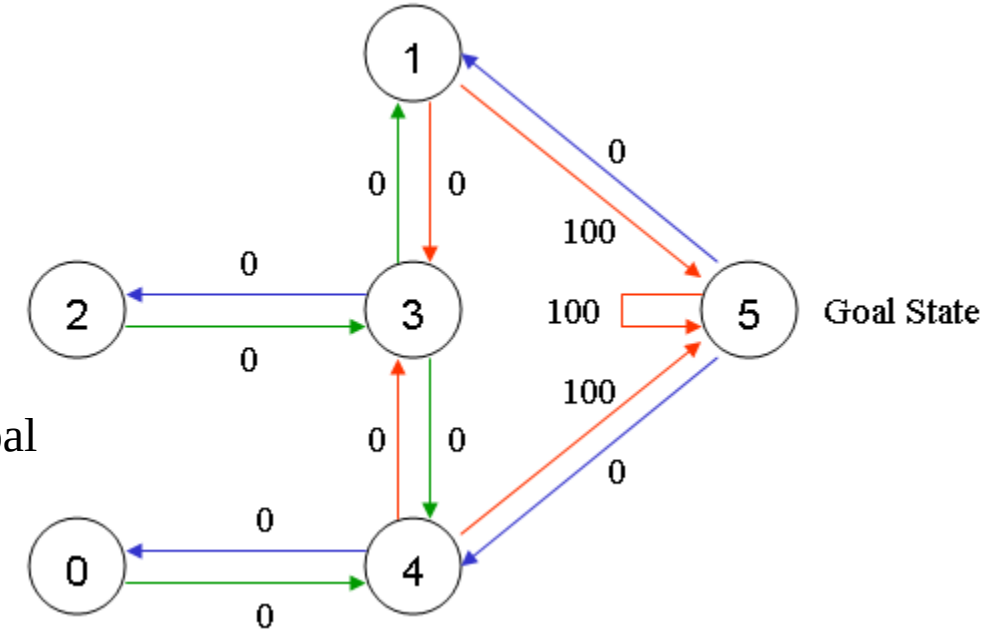
Source *http://mnemstudio.org/path-finding-q-learning-tutorial.htm*

Reinforcement Learning

5.2.2 Reinforcement Learning: Q-Learning

Example

- For this example, we'd like to put an agent in any room, and from that room, go outside the building (this will be our target room). In other words, the goal room is number 5.
- To set this room as a goal, we'll associate a reward value to each door (i.e. link between nodes). The doors that lead immediately to the goal have an instant reward of 100. Other doors not directly connected to the target room have zero reward.
- Because doors are two-way (0 leads to 4, and 4 leads back to 0), two arrows are assigned to each room. Each arrow contains an instant reward value.
- We can put the state diagram and the instant reward values into the following reward table, "matrix R".



$$R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} \multicolumn{6}{c}{\text{Action}} \\ 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

Note: The -1's in the table represent null values (i.e.; where there isn't a link between nodes). For example, State 0 cannot go to State 1.

Source *http://mnemstudio.org/path-finding-q-learning-tutorial.htm*

Reinforcement Learning

5.2.2 Reinforcement Learning: Q-Learning

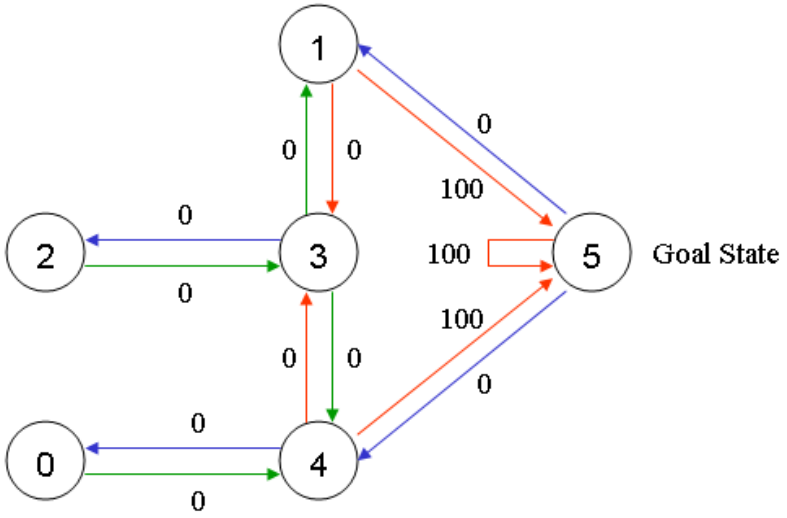
Example

To understand how the Q-learning algorithm works, we'll go through a few episodes step by step.

We'll start by setting the value of the learning parameter $\gamma = 0.8$, and the initial state as Room 1.

Initialize matrix Q as a zero matrix:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$



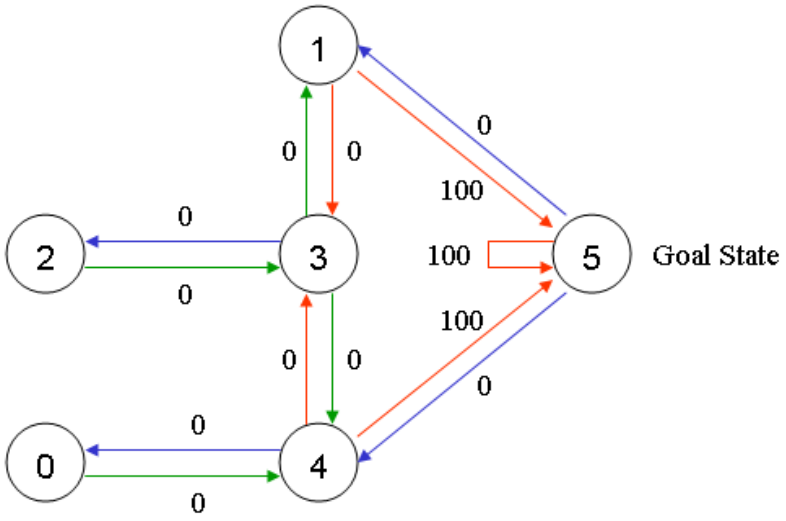
Look at the second row (state 1) of matrix R. There are two possible actions for the current state 1: go to state 3, or go to state 5. By random selection, we select to go to 5 as our action.

Source *http://mnemstudio.org/path-finding-q-learning-tutorial.htm*

Reinforcement Learning

5.2.2 Reinforcement Learning: Q-Learning

Example



Now let's imagine what would happen if our agent were in state 5. Look at the sixth row of the reward matrix R (i.e. state 5). It has 3 possible actions: go to state 1, 4 or 5.

$$Q(state, action) = R(state, action) + \gamma * Max[Q(next\ state, all\ actions)]$$

$$Q(1,5) = R(1,5) + 0.8 * Max[Q(5,1), Q(5,4), Q(5,5)] = 100 + 0.8 * 0 = 100$$

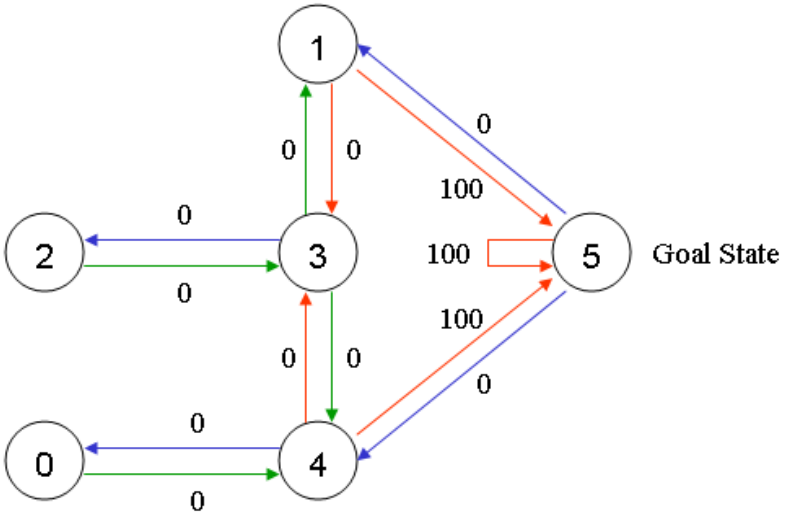
Since matrix Q is still initialized to zero, Q(5, 1), Q(5, 4), Q(5, 5), are all zero. The result of this computation for Q(1, 5) is 100 because of the instant reward from R(5, 1).

Source *http://mnemstudio.org/path-finding-q-learning-tutorial.htm*

Example

The next state, 5, now becomes the current state. Because 5 is the goal state, we've finished one episode. Our agent's brain now contains an updated matrix Q as:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

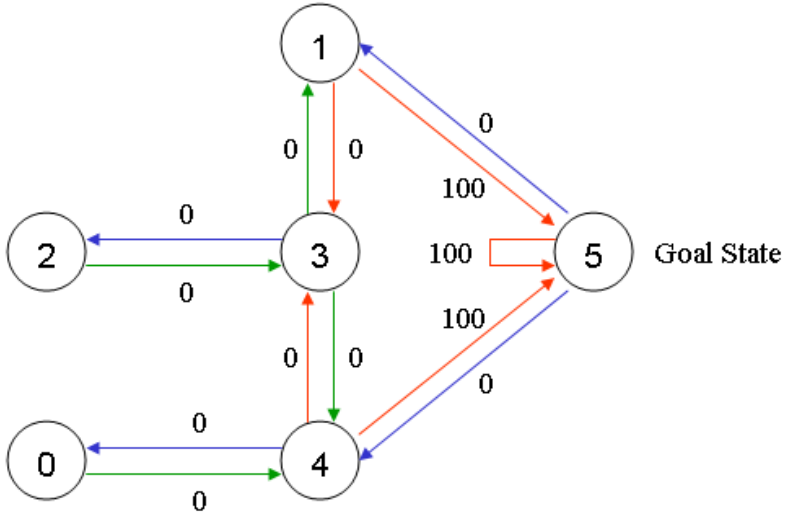


Source *http://mnemstudio.org/path-finding-q-learning-tutorial.htm*

Reinforcement Learning

5.2.2 Reinforcement Learning: Q-Learning

Example



For the next episode, we start with a randomly chosen initial state. This time, we have state 3 as our initial state.

Look at the fourth row of matrix R; it has 3 possible actions: go to state 1, 2 or 4. By random selection, we select to go to state 1 as our action.

Now we imagine that we are in state 1. Look at the second row of reward matrix R (i.e. state 1). It has 2 possible actions: go to state 3 or state 5. Then, we compute the Q value:

$$Q(state, action) = R(state, action) + \gamma * Max[Q(next\ state, all\ actions)]$$

$$Q(1,5) = R(1,5) + 0.8 * Max[Q(1,2), Q(1,5)] = 0 + 0.8 * Max(0,100) = 80$$

Source *http://mnemstudio.org/path-finding-q-learning-tutorial.htm*

Reinforcement Learning

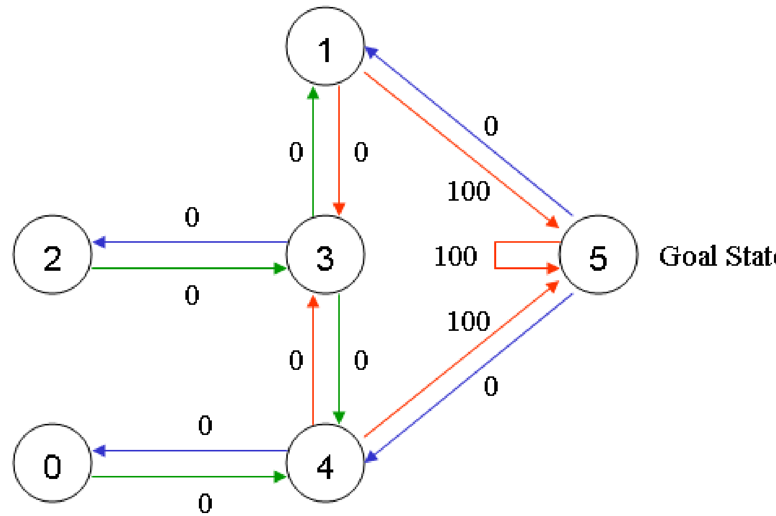
5.2.2 Reinforcement Learning: Q-Learning

Example

We use the updated matrix Q from the last episode. Q(1, 3) = 0 and Q(1, 5) = 100. The result of the computation is Q(3, 1) = 80 because the reward is zero. The matrix Q becomes:

$Q =$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	100
2	0	0	0	0	0	0
3	0	80	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

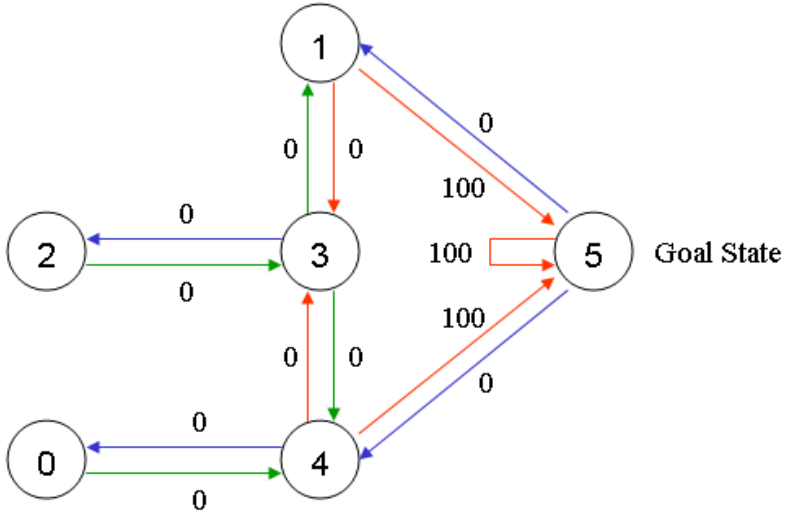


The next state, 1, now becomes the current state. We repeat the inner loop of the Q learning algorithm because state 1 is not the goal state.

Source *http://mnemstudio.org/path-finding-q-learning-tutorial.htm*

5.2.2 Reinforcement Learning: Q-Learning

Example



So, starting the new loop with the current state 1, there are two possible actions: go to state 3, or go to state 5. By lucky draw, our action selected is 5.

Now, imaging we're in state 5, there are three possible actions: go to state 1, 4 or 5. We compute the Q value using the maximum value of these possible actions.

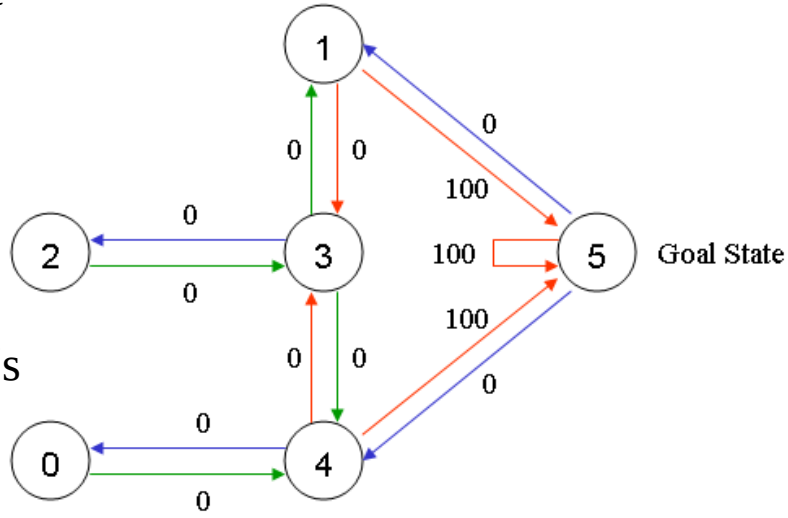
$$Q(state, action) = R(state, action) + \gamma * Max[Q(next\ state, all\ actions)]$$

$$Q(1,5) = R(1,5) + 0.8 * Max[Q(5,1), Q(5,4), Q(5,5)] = 100 + 0.8 * 0 = 100$$

Source *http://mnemstudio.org/path-finding-q-learning-tutorial.htm*

Example

The updated entries of matrix Q, Q(5, 1), Q(5, 4), Q(5, 5), are all zero. The result of this computation for Q(1, 5) is 100 because of the instant reward from R(5, 1). This result does not change the Q matrix.



Because 5 is the goal state, we finish this episode. Our agent's brain now contain updated matrix Q as:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Source *http://mnemstudio.org/path-finding-q-learning-tutorial.htm*

Example

If our agent learns more through further episodes, it will finally reach convergence values in matrix Q like:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\ 0 & 400 & 0 & 0 & 400 & 500 \end{bmatrix} \end{array}$$

This matrix Q, can then be normalized (i.e.; converted to percentage) by dividing all non-zero entries by the highest number (500 in this case):

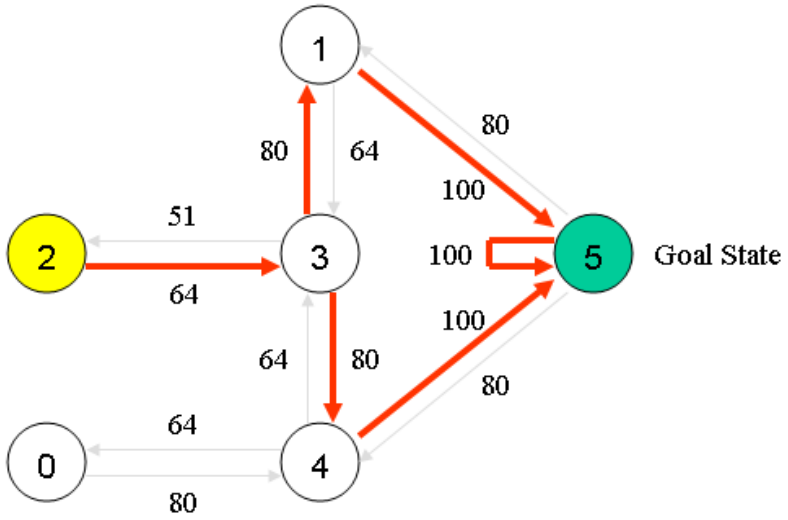
$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix} \end{array}$$

Once the matrix Q gets close enough to a state of convergence, we know our agent has learned the most optimal paths to the goal state. Tracing the best sequences of states is as simple as following the links with the highest values at each state.

Source *http://mnemstudio.org/path-finding-q-learning-tutorial.htm*

Example

For example, from initial State 2, the agent can use the matrix Q as a guide:
From State 2 the maximum Q values suggests the action to go to state 3.
From State 3 the maximum Q values suggest two alternatives: go to state 1 or 4. Suppose we arbitrarily choose to go to 1.
From State 1 the maximum Q values suggests the action to go to state 5.
Thus the sequence is 2 - 3 - 1 - 5.



Source *http://mnemstudio.org/path-finding-q-learning-tutorial.htm*

Reinforcement Learning

5.3 Reinforcement Learning: Features

Reinforcement learning differs from standard supervised learning in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected. Further, there is a focus on on-line performance.

Reinforcement learning can be used in large environments in any of the following situations:

- A model of the environment is known, but an analytic solution is not available;
- Only a simulation model of the environment is given (the subject of simulation-based optimization);
- The only way to collect information about the environment is by interacting with it.

[0] https://en.wikipedia.org/wiki/Reinforcement_learning

Part 6
Deep Learning

- Intuition
- Convolutional Neural Network
- Recurrent and Recursive Nets

Deep Learning

6.1 Introduction: History

- Inspired by the architectural depth of the brain, researchers wanted for decades to train deep multi-layer neural networks.
- **No successful attempts were reported before 2006:** Researchers reported positive experimental results with typically two or three levels (i.e. one or two hidden layers), but training deeper networks consistently yielded poorer results.
- **Exception:** convolutional neural networks, LeCun 1998
- SVM: Vapnik and his co-workers developed the Support Vector Machine (1993). It is a shallow architecture.
- **Digression:** In the 1990's, many researchers abandoned neural networks with multiple adaptive hidden layers because SVMs worked better, and there was no successful attempts to train deep networks.
- **Breakthrough in 2006**

[1] CMU slides, http://www.cs.cmu.edu/~aarti/Class/10701_Spring14/slides/DeepLearning.pdf

Breakthrough in 2006

- **Deep Belief Networks (DBN)**

Hinton, G. E, Osindero, S., and Teh, Y. W. (2006).

A fast learning algorithm for deep belief nets.

Neural Computation, 18:1527-1554.

- **Autoencoders**

Bengio, Y., Lamblin, P., Popovici, P., Larochelle, H. (2007).

Greedy Layer-Wise Training of Deep Networks,

Advances in Neural Information Processing Systems 19

[1] CMU slides, http://www.cs.cmu.edu/~aarti/Class/10701_Spring14/slides/DeepLearning.pdf

Deep Learning

6.1 Introduction: Intutition

Conventional machine-learning techniques were limited in their ability to process natural data in their raw form. For decades, constructing a pattern-recognition or machine-learning system required careful engineering and considerable domain expertise to design a feature extractor that transformed the raw data (such as the pixel values of an image) into a suitable internal representation or feature vector from which the learning subsystem, often a classifier, could detect or classify patterns in the input.

Representation learning is a set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection or classification. Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level.

[1] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." Nature 521.7553 (2015): 436-444.

Deep Learning

6.1 Introduction: Intutition

An image, for example, comes in the form of an array of pixel values, and the learned features in the first layer of representation typically represent the presence or absence of edges at particular orientations and locations in the image. The second layer typically detects motifs by spotting particular arrangements of edges, regardless of small variations in the edge positions. The third layer may assemble motifs into larger combinations that correspond to parts of familiar objects, and subsequent layers would detect objects as combinations of these parts. The key aspect of deep learning is that these layers of features are not designed by human engineers: they are learned from data using a general-purpose learning procedure.

[1] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." Nature 521.7553 (2015): 436-444.

Selective-invariance Dilemma

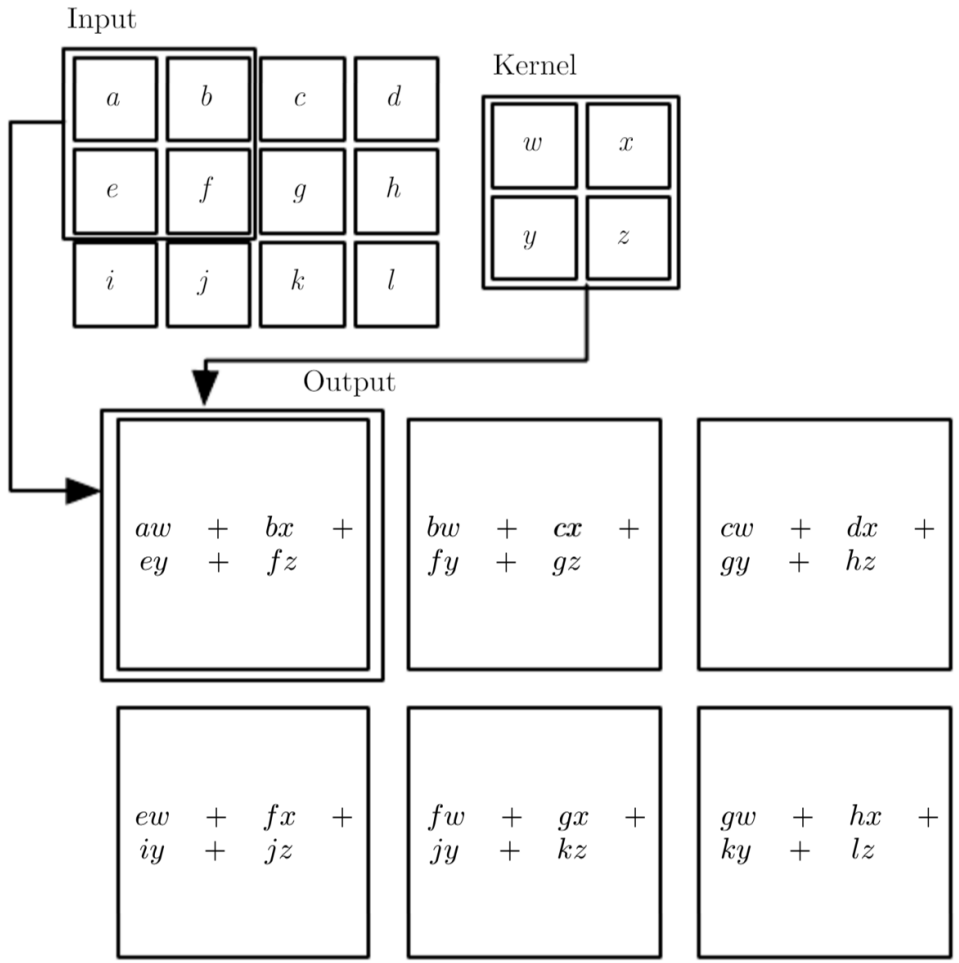
- Need to produce representations that are selective to the aspects of the image that are important for discrimination, but that are invariant to irrelevant aspects
- Example: At the pixel level, images of two Samoyeds in different poses and in different environments may be very different from each other, whereas two images of a Samoyed and a wolf in the same position and on similar backgrounds may be very similar to each other. The latter two should be distinguished, while the former two should be classified in the same category.
- Shallow classifier couldn’t fulfill above tasks.
- The conventional option is to hand design good feature extractors, which requires a considerable amount of engineering skill and domain expertise. But this can all be avoided if good features can be learned automatically using a general-purpose learning procedure. This is the key advantage of deep learning.

[1] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." Nature 521.7553 (2015): 436-444.

Deep Learning

6.2 Convolutional Neural Network: What is convolution

- Convolution operation: $s(t) = \int x(a)w(t -$

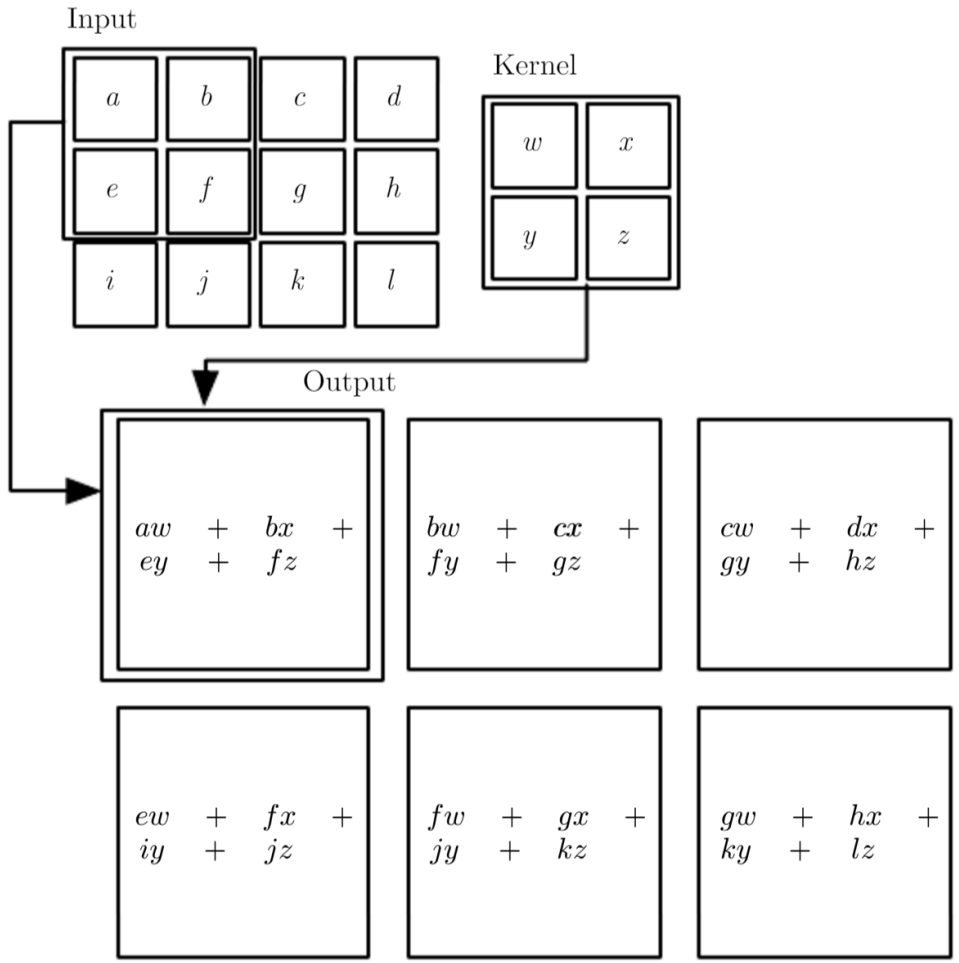


An example of 2-D convolution without kernel-flipping.

Deep Learning

6.2 Convolutional Neural Network: What is convolution

- Convolution operation: $s(t) = \int x(a)w(t -$



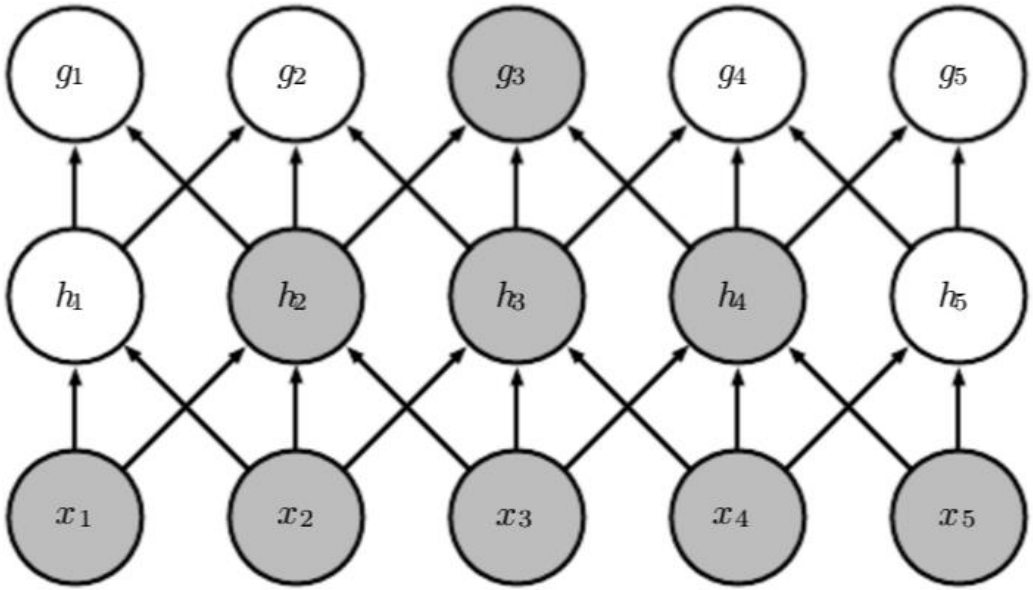
An example of 2-D convolution without kernel-flipping.

Deep Learning

6.2 Convolutional Neural Network: Motivation of convolution

1. Sparse Interactions

- Convolutional networks typically have sparse interactions(also referred to as sparse connectivity or sparse weights). This is accomplished by making the kernel smaller than the input.
- For example, when processing an image, the input image might have thousands or millions of pixels, but we can detect small, meaningful features such as edges with kernels that occupy only tens or hundreds of pixels. This means that we need to store fewer parameters, which both reduces the memory requirements of the model and improves its statistical efficiency. It also means that computing the output requires fewer operations.



Explanation of Sparse Interactions

The receptive field of the units in the deeper layers of a convolutional network is larger than the receptive field of the units in the shallow layers. This effect increases if the network includes architectural features like strided convolution or pooling. This means that even though direct connections in a convolutional net are very sparse, units in the deeper layers can be indirectly connected to all or most of the input image.

2. Parameter Sharing

- Parameter sharing refers to using the same parameter for more than one function in a model. The parameter sharing used by the convolution operation means that rather than learning a separate set of parameters for every location, we learn only one set. Convolution is thus dramatically more efficient than dense matrix multiplication in terms of the memory requirements and statistical efficiency.

3. Equivariant Representations

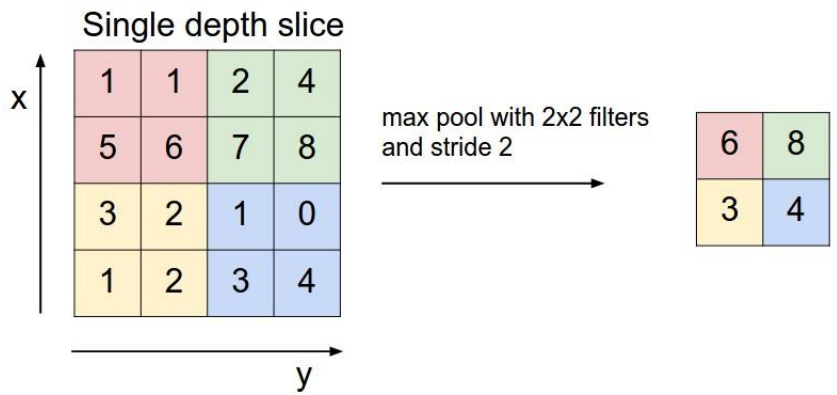
- In the case of convolution, the particular form of parameter sharing causes the layer to have a property called equivariance to translation. To say a function is equivariant means that if the input changes, the output changes in the same way. Specifically, a function $f(x)$ is equivariant to a function g if $f(g(x)) = g(f(x))$.
- When processing time series data, this means that convolution produces a sort of timeline that shows when different features appear in the input. If we move an event later in time in the input, the exact same representation of it will appear in the output, just later in time. Similarly with images, convolution creates a 2-D map of where certain features appear in the input. If we move the object in the input, its representation will move the same amount in the output.
- Convolution is not naturally equivariant to some other transformations, such as changes in the scale or rotation of an image. Other mechanisms are necessary for handling these kinds of transformations.

Deep Learning

6.2 Convolutional Neural Network: Pooling

Features:

- In all cases, pooling helps to make the representation become approximately invariant to small translations of the input. Invariance to local translation can be a very useful property if we care more about whether some feature is present than exactly where it is.
- Because pooling summarizes the responses over a whole neighborhood, it is possible to use fewer pooling units than detector units, by reporting summary statistics for pooling regions. This improves the computational efficiency of the network.
- For many tasks, pooling is essential for handling inputs of varying size.

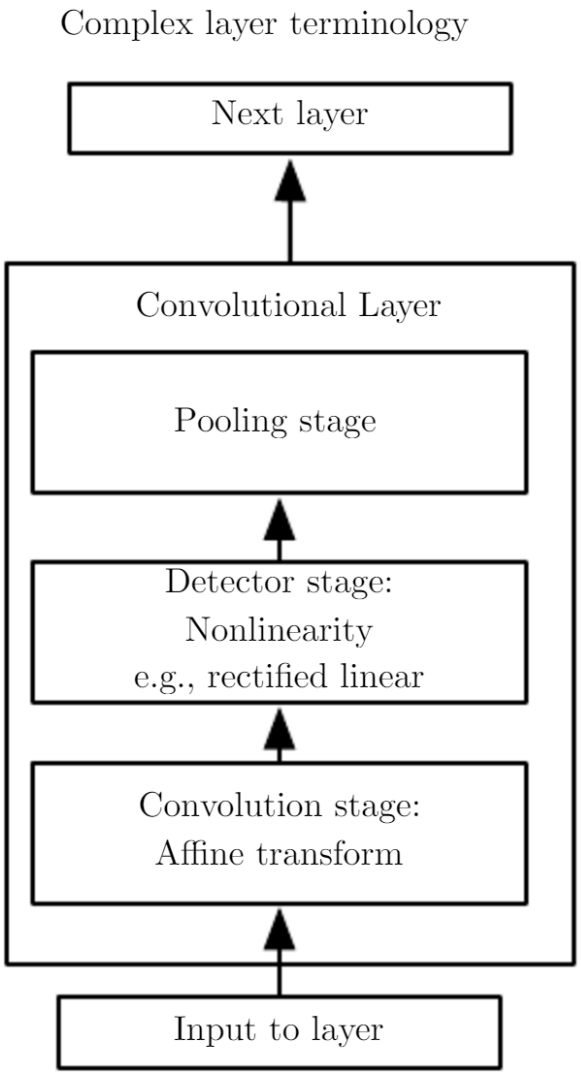


Usage:

- Some theoretical work gives guidance as to which kinds of pooling one should use in various situations (Boureau et al., 2010). It is also possible to dynamically pool features together, for example, by running a clustering algorithm on the locations of interesting features (Boureau et al., 2011). This approach yields a different set of pooling regions for each image. Another approach is to learn a single pooling structure that is then applied to all images (Jia et al., 2012).
- Pooling can complicate some kinds of neural network architectures that use top-down information, such as Boltzmann machines and auto encoders.

Deep Learning

6.2 Convolutional Neural Network: Architectures of convolutional networks

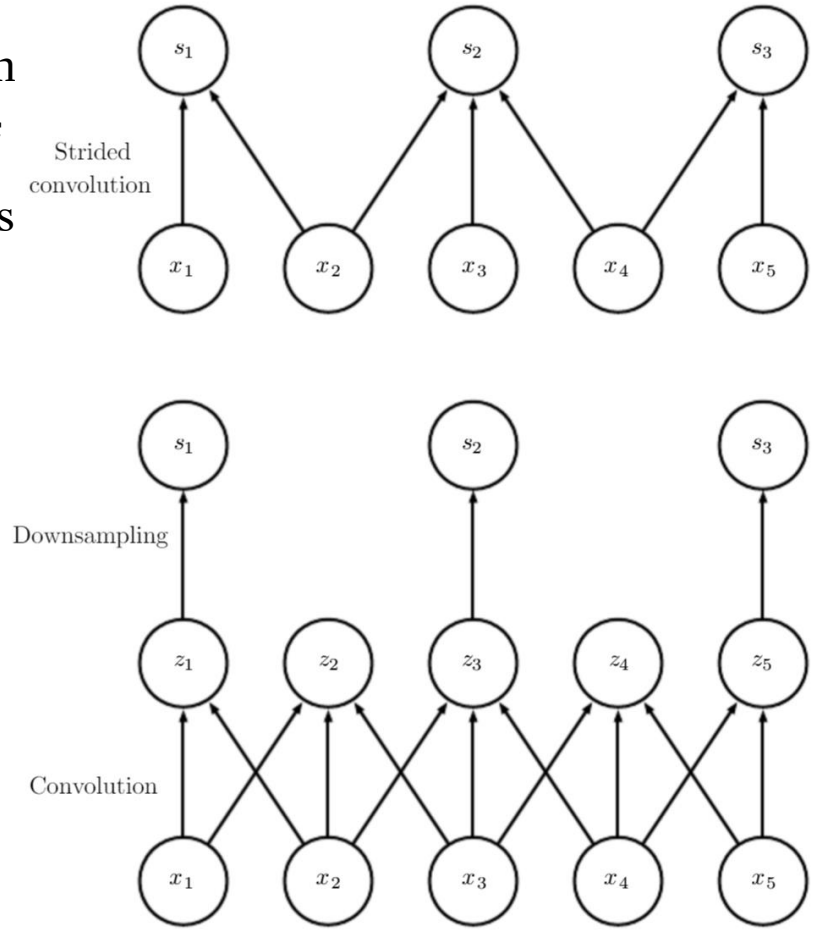


the convolutional net is viewed as a small number of relatively complex layers, with each layer having many “stages.” In this terminology, there is a one-to-one mapping between kernel tensors and network layers.

6.2 Convolutional Neural Network: Variants of the Basic Convolution Function

1. Convolution with a stride

- We may want to skip over some positions of the kernel in order to reduce the computational cost (at the expense of not extracting our features as finely). We can think of this as down sampling the output of the full convolution function. It is also possible to define a separate stride for each direction of motion.



Convolution with a stride

- *(Top)Convolution with a stride length of two implemented in a single operation.*
- *(Bottom)Convolution with a stride greater than one pixel is mathematically equivalent to convolution with unit stride followed by downsampling.*
- *Obviously, the two-step approach involving downsampling is computationally wasteful, because it computes many values that are then discarded.*

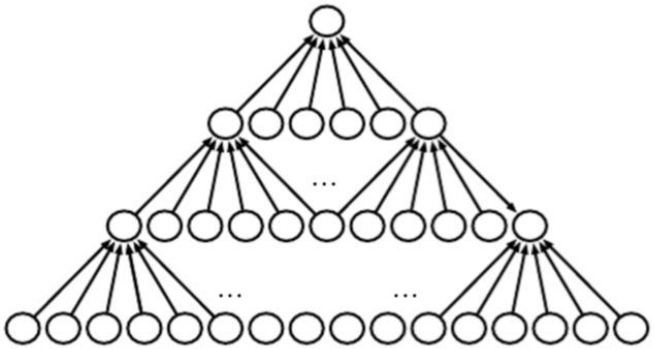
[1] Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville, http://www.deeplearningbook.org, Book in preparation for MIT Press,2016

Deep Learning

6.2 Convolutional Neural Network: Variants of the Basic Convolution Function

2. Zero padding

- One essential feature of any convolutional network implementation is the ability to implicitly zero-pad the input in order to make it wider.



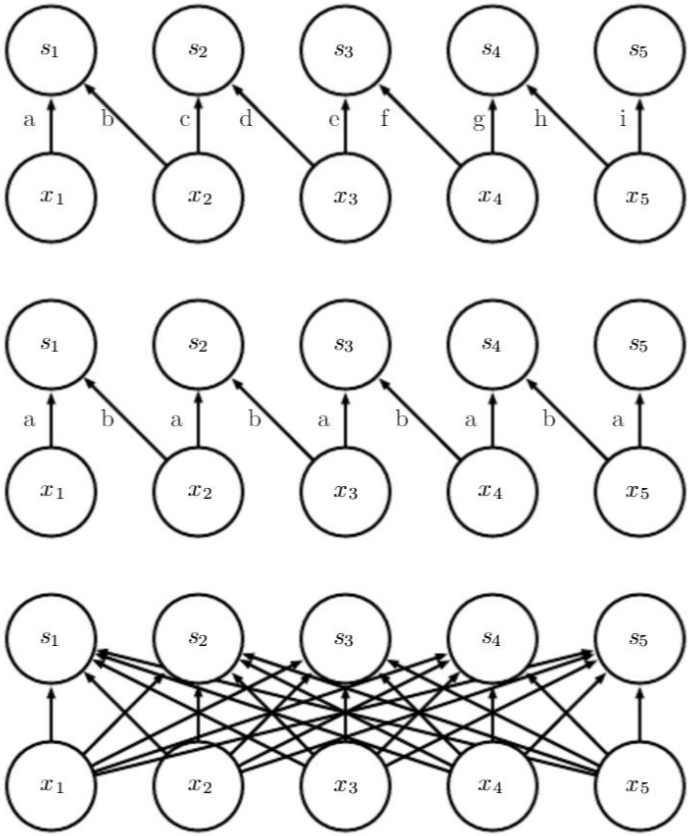
The effect of zero padding on network size

- *Consider a convolutional network with a kernel of width six at every layer. In this example, we do not use any pooling, soonly the convolution operation itself shrinks the network size.*
- *(Top) In this convolutional network, we do not use any implicit zero padding. This causes the representation to shrink by five pixels at each layer.*
- *(Bottom) By adding five implicit zeroes to each layer, we prevent the representation from shrinking with depth. This allows us to make an arbitrarily deep convolutional network.*

[1] Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville, http://www.deeplearningbook.org, Book in preparation for MIT Press,2016

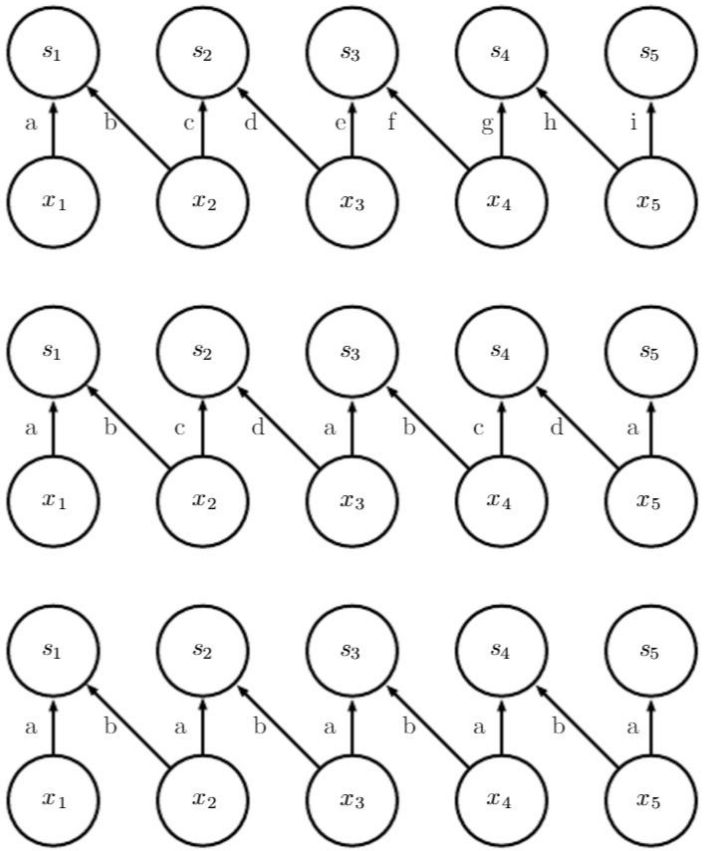
3. Locally connected layers, tiled convolution

- Locally connected layers are useful when we know that each feature should be a function of a small part of space, but there is no reason to think that the same feature should occur across all of space.
- Tiled convolution (Gregor and LeCun, 2010a; Le et al., 2010) offers a compromise between a convolutional layer and a locally connected layer. Rather than learning a separate set of weights at every spatial location, we learn a set of kernels that we rotate through as we move through space.



Comparison of local connections, convolution, and full connections

- (Top) A locally connected layer with a patch size of two pixels. Each edge is labeled with a unique letter to show that each edge is associated with its own weight parameter.
- (Center) A convolutional layer with a kernel width of two pixels. This model has exactly the same connectivity as the locally connected layer. The locally connected layer has no parameter sharing.
- (Bottom) A fully connected layer does not have the restricted connectivity of the locally connected layer.



A comparison of locally connected layers, tiled convolution, and standard convolution

- All three have the same sets of connections between units, when the same size of kernel is used. This diagram illustrates the use of a kernel that is two pixels wide. The differences between the methods lies in how they share parameters.
- (Top) A locally connected layer has no sharing at all.
- (Center) Tiled convolution has a set of t different kernels. Here we illustrate the case of t= 2.
- (Bottom) Traditional convolution is equivalent to tiled convolution with t= 1. There is only one kernel and it is applied everywhere.

Deep Learning

6.2 Convolutional Neural Network: Data Types

- The data used with a convolutional network usually consists of several channels, each channel being the observation of a different quantity at some point in space or time.
- For an example of convolutional networks applied to video, see *Chen et al.(2010)*.

Deep Learning

6.2 Convolutional Neural Network: Data Types

	Single Channel	Multi-Channel
1-D	Audio waveform: The axis we convolve over corresponds to time. We discretize time and measure the amplitude of the waveform once per time step.	Skeleton animation data: Animations of 3-D computer-rendered characters are generated by altering the pose of a "skeleton" overtime. At each point in time, the pose of the character is described by a specification of the angles of each of the joints in the character's skeleton. Each channel in the data we feed to the convolutional model represents the angle about one axis of one joint.
2-D	Audio data that has been preprocessed with a Fourier transform: We can transform the audio wave-form into a 2D tensor with different rows corresponding to different frequencies and different columns corresponding to different points in time. Using convolution in the time makes the model equivariant to shifts in time. Using convolution across the frequency axis makes the model equivariant to frequency, so that the same melody played in a different octave produces the same representation but at a different height in the network's output.	Color image data: One channel contains the red pixels, one the green pixels, and one the blue pixels. The convolution kernel moves over both the horizontal and vertical axes of the image, conferring translation equivariance in both directions.
3-D	Volumetric data: A common source of this kind of data is medical imaging technology, such as CT scans.	Color video data: One axis corresponds to time, one to the height of the video frame, and one to the width of the video frame.

Processing variable sized inputs

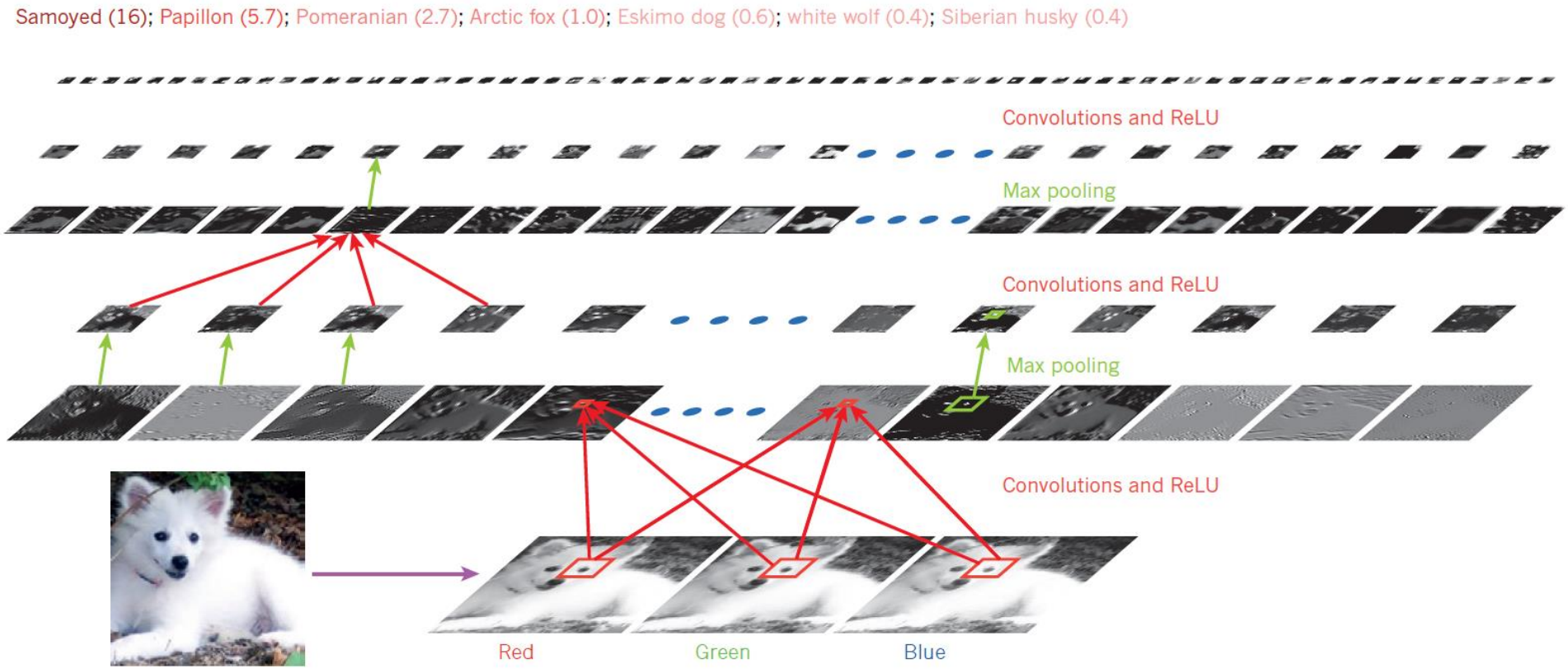
- One advantage to convolutional networks is that they can also process inputs with varying spatial extents. These kinds of input simply cannot be represented by traditional, matrix multiplication-based neural networks.
- For example, consider a collection of images, where each image has a different width and height. It is unclear how to model such inputs with a weight matrix of fixed size. Convolution is straightforward to apply; the kernel is simply applied a different number of times depending on the size of the input, and the output of the convolution operation scales accordingly. In some cases, the network must produce some fixed-size output. In this case we must make some additional design steps, like inserting a pooling layer whose pooling regions scale in size proportional to the size of the input, in order to maintain a fixed number of pooled outputs.

Processing variable sized inputs

- Note that the use of convolution for processing variable sized inputs only makes sense for inputs that have variable size because they contain varying amounts of observation of the same kind of thing—different lengths of recordings overtime, different widths of observations over space, etc. Convolution does not make sense if the input has variable size because it can optionally include different kinds of observations. For example, if we are processing college applications, and our features consist of both grades and standardized test scores, but not every applicant took the standardized test, then it does not make sense to convolve the same weights over both the features corresponding to the grades and the features corresponding to the test scores.

Deep Learning

6.2 Convolutional Neural Network: Example



The outputs of each layer (horizontally) of a typical convolutional network architecture applied to the image of a Samoyed dog (bottom left; and RGB (red, green, blue) inputs, bottom right). Each rectangular image is a feature map corresponding to the output for one of the learned features, detected at each of the image positions. Information flows bottom up, with lower-level features acting as oriented edge detectors, and a score is computed for each image class in output. ReLU, rectified linear unit.

[1] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." Nature 521.7553 (2015): 436-444.

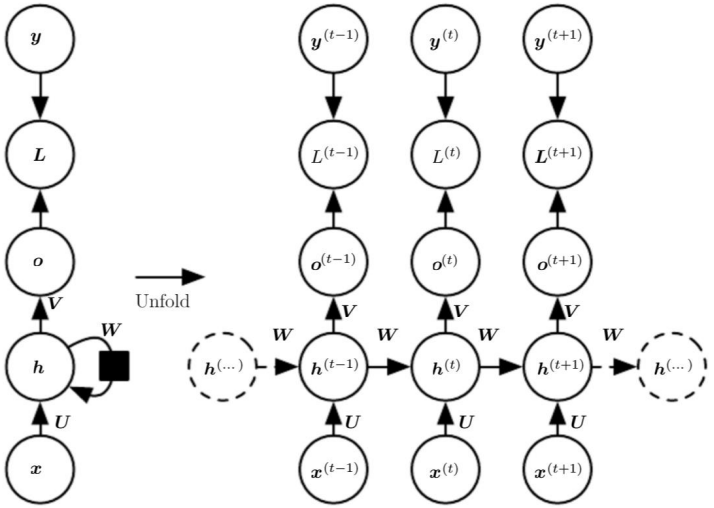
Deep Learning

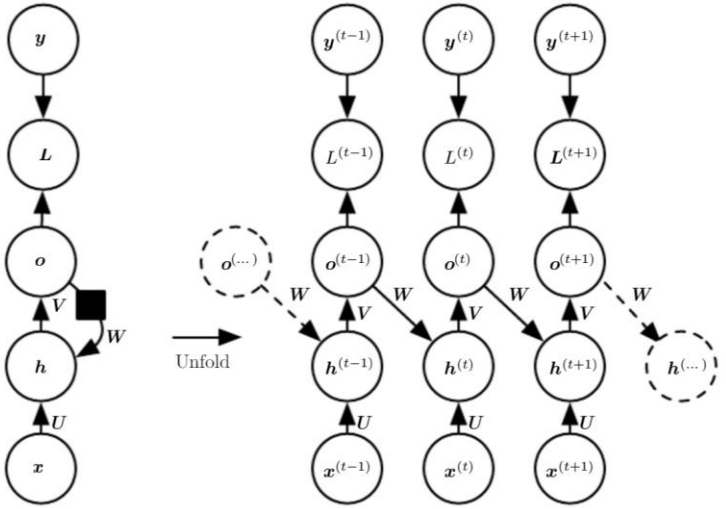
6.2 Recurrent and Recursive Nets: Overview

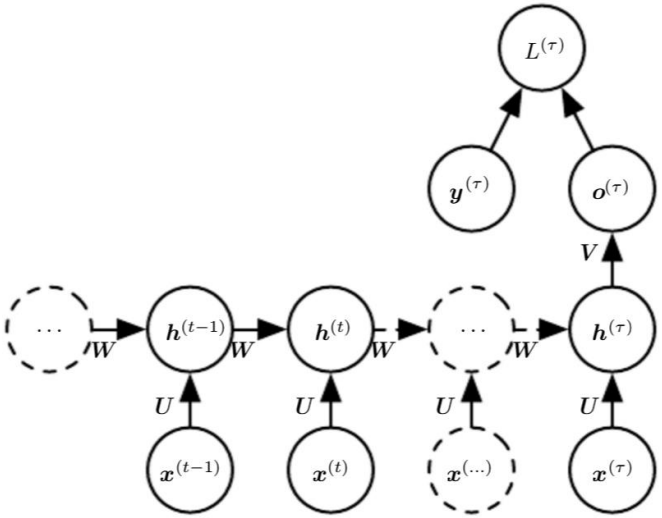
- Recurrent neural networks or RNNs (Rumelhart et al., 1986a) are a family of neural networks for processing **sequential data**. Most recurrent networks can also process sequences of variable length.
- A recurrent neural network shares the same weights across several time steps.
- Some examples of important design patterns for recurrent neural networks include the following:
- Recurrent networks that produce an output at each time step and have recurrent connections between hidden units, illustrated in left figure.
- Recurrent networks that produce an output at each time step and have recurrent connections only from the output at one time step to the hidden units at the next time step, illustrated in medium figure.
- Recurrent networks with recurrent connections between hidden units, that read an entire sequence and then produce a single output, illustrated in right figure.

Deep Learning

6.2 Recurrent and Recursive Nets: Overview



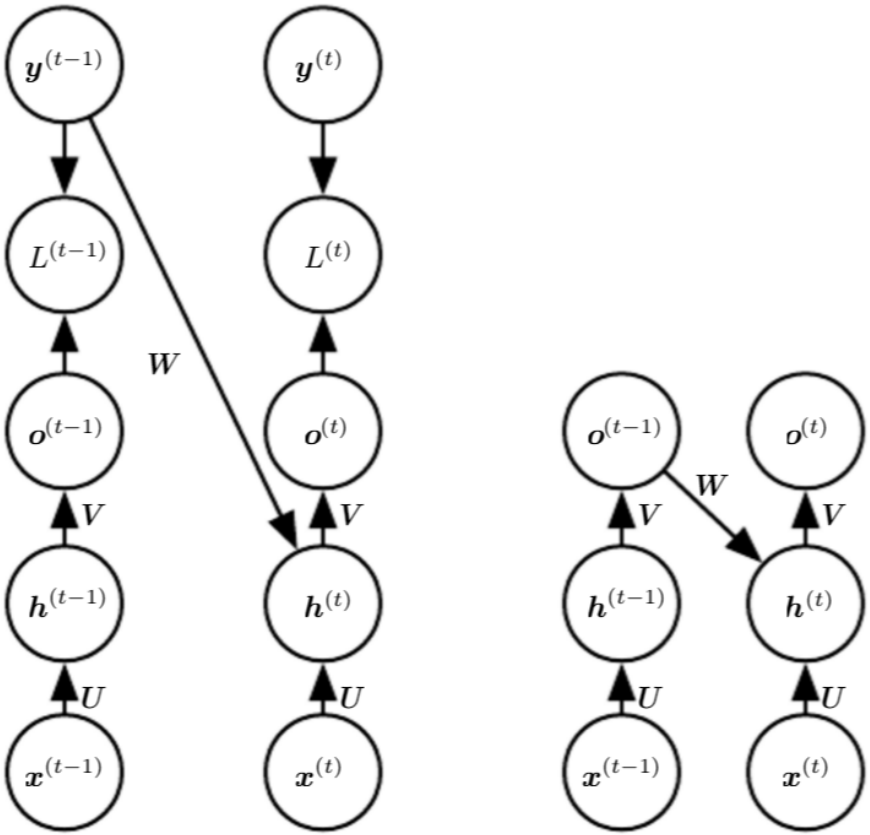




[1] Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville, http://www.deeplearningbook.org, Book in preparation for MIT Press,2016

Deep Learning

6.2 Recurrent and Recursive Nets: Teacher Forcing and Networks with Output Recurrence



- Illustration of teacher forcing. Teacher forcing is a training technique that is applicable to RNNs that have connections from their output to their hidden states at the next time step.
- (Left) At train time, we feed the correct output $y^{(t)}$ drawn from the trainset as input to $h^{(t+1)}$.
- (Right) When the model is deployed, we approximate the correct output $y^{(t)}$ with the model's output $o^{(t)}$, and feed the output back into the model.

[1] Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville, http://www.deeplearningbook.org, Book in preparation for MIT Press,2016

Deep Learning

6.2 Recurrent and Recursive Nets: Teacher Forcing and Networks with Output Recurrence

- Models that have recurrent connections from their outputs leading back into the model may be trained with teacher forcing.
- Teacher forcing is a procedure that emerges from the maximum likelihood criterion, in which during training the model receives the ground truth output $y^{(t)}$ as input at time t+ 1. We can see this by examining a sequence with two time steps. The conditional maximum likelihood criterion is

$$\log p(\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)} | \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}) = \log p(\boldsymbol{y}^{(2)} | \boldsymbol{y}^{(1)}, \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}) + \log p(\boldsymbol{y}^{(1)} | \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)})$$

- Maximum likelihood thus specifies that during training, rather than feeding the model's own output back into itself, these connections should be fed with the target values specifying what the correct output should be.
- The disadvantage of strict teacher forcing arises if the network is going to be later used in an open-loop mode, with the network outputs (or samples from the output distribution) fed back as input. In this case, the kind of inputs that the network sees during training could be quite different from the kind of inputs that it will see at test time.

Deep Learning

6.2 Recurrent and Recursive Nets: Computing the Gradient in a Recurrent Neural Network

The use of back-propagation on the unrolled graph is called the back-propagation through time (BPTT) algorithm.

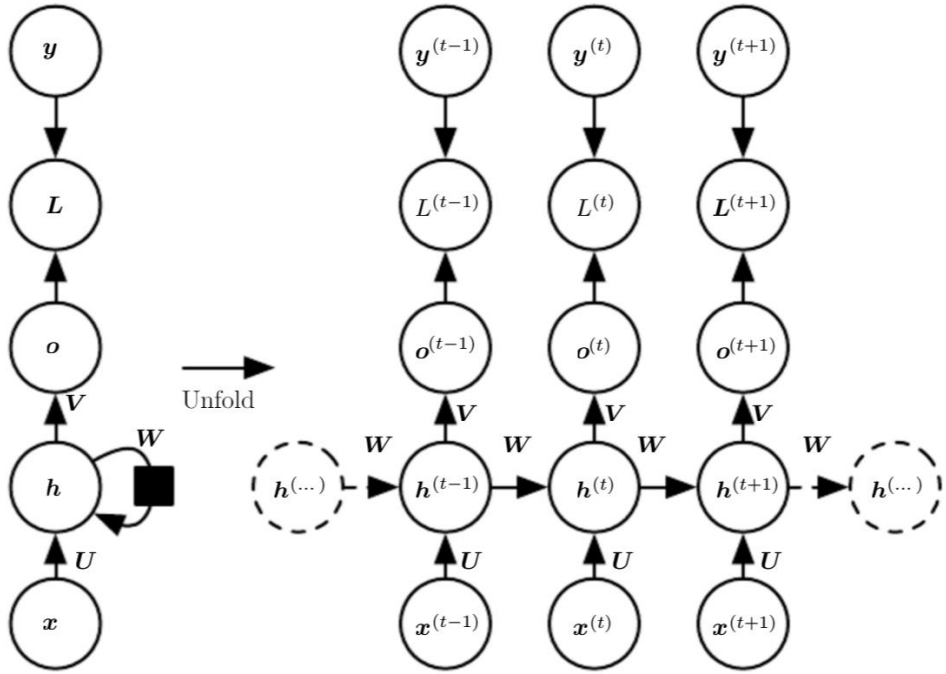
Take the right figure and following setting as an example to compute the gradient:

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}$$

$$\boldsymbol{h}^{(t)} = \tanh\left(\boldsymbol{a}^{(t)}\right)$$

$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}$$

$$\hat{\boldsymbol{y}}^{(t)} = softmax(\boldsymbol{o}^{(t)})$$



$$L\left(\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, \dots, \boldsymbol{y}^{(t)}\}\right) = \sum_t L^{(t)}$$

$$= -\sum_t log p_{model}(y^{(t)} | \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(t)}\})$$

Where $p_{model}(\boldsymbol{y}^{(t)} | \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(t)}\}$ is given by reading the entry for $y^{(t)}$ from the model's output vector $\hat{\boldsymbol{y}}^{(t)}$.

In this derivation we assume that the outputs $\boldsymbol{o}^{(t)}$ are used as the argument to the softmax function to obtain the vector $\hat{\boldsymbol{y}}$ of probabilities over the output. We also assume that the loss is the negative log-likelihood of the true target y(t) given the input so far.

[1] Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville, http://www.deeplearningbook.org, Book in preparation for MIT Press,2016

Start the recursion with the nodes immediately preceding the final loss

$$\frac{\partial L}{\partial L^{(t)}} = 1$$

The gradient on the outputs at time step t, for all i, t

$$(\nabla_{\boldsymbol{o}^{(t)}} L)_{\boldsymbol{i}} = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - 1_{i,y^{(t)}}$$

At the final time step τ

$$\nabla_{\boldsymbol{h}^{(\tau)}} L = \boldsymbol{V}^T \nabla_{\boldsymbol{o}^{(\tau)}} L$$

We can then iterate backwards in time to back-propagate gradients through time

$$\nabla_{\boldsymbol{h}^{(t)}} L = \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}}\right)^T \left(\nabla_{\boldsymbol{h}^{(t+1)}} L\right) + \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}}\right)^T \left(\nabla_{\boldsymbol{o}^{(t)}} L\right)$$

$$= \boldsymbol{W}^T \left(\nabla_{\boldsymbol{h}^{(t+1)}} L\right) diag\left(1 - \left(\boldsymbol{h}^{(t+1)}\right)^2\right) + \boldsymbol{V}^T \left(\nabla_{\boldsymbol{o}^{(t)}} L\right)$$

Where $diag\left(1 - \left(h^{(t+1)}\right)^2\right)$ indicates the diagonal matrix containing the elements $1 - (h_i^{(t+1)})^2$.

This is the Jacobian of the hyperbolic tangent associated with the hidden unit i at time t + 1.

6.2 Recurrent and Recursive Nets: Computing the Gradient in a Recurrent Neural Network

However, the $\nabla_{\boldsymbol{W}} f$ operator used in calculus takes into account the contribution of $\boldsymbol{W}$ to the value of f due to all edges in the computational graph. To resolve this ambiguity, we introduce dummy variables $\boldsymbol{W}^{(t)}$ that are defined to be copies of $\boldsymbol{W}$ but with each $\boldsymbol{W}^{(t)}$used only at time step t.

$$\nabla_{\boldsymbol{c}} L = \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}}\right)^T \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L$$

$$\nabla_{\boldsymbol{b}} L = \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}}\right)^T \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t diag\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \nabla_{\boldsymbol{h}^{(t)}} L$$

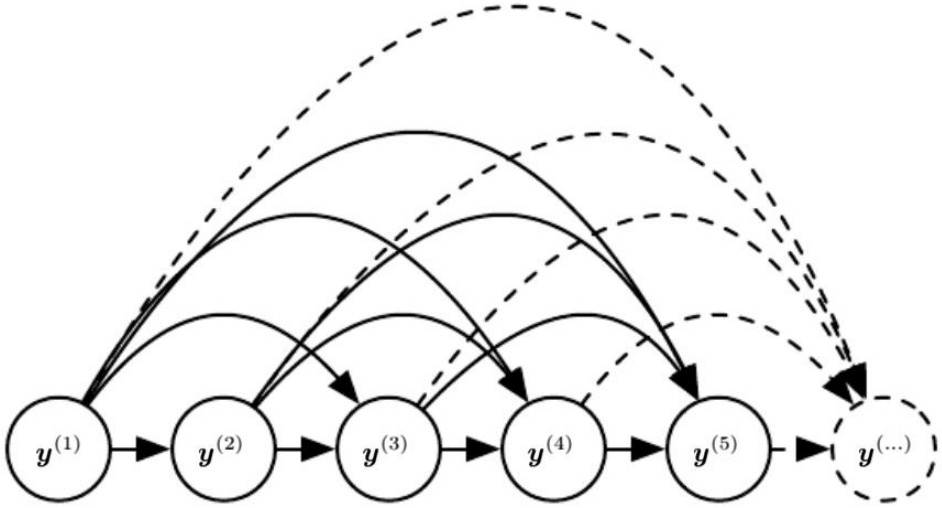
$$\nabla_{\boldsymbol{V}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}}\right) \nabla_{\boldsymbol{V}} o_i^{(t)} = \sum_t \left(\nabla_{\boldsymbol{o}^{(t)}} L\right) \boldsymbol{h}^{(t)^T}$$

$$\nabla_{\boldsymbol{W}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{W}} h_i^{(t)} = \sum_t diag\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right)\left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{h}^{(t-1)^T}$$

$$\nabla_{\boldsymbol{U}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{U}^{(t)}} h_i^{(t)} = \sum_t diag\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right)\left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{x}^{(t)^T}$$

Deep Learning

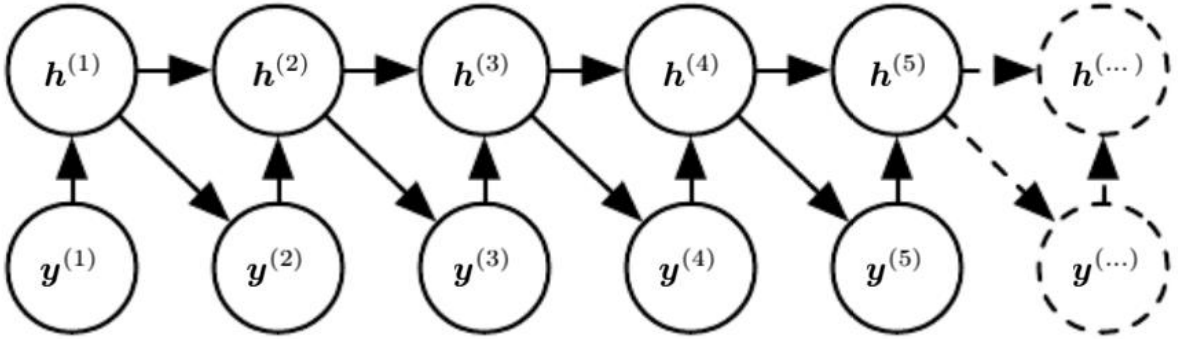
6.2 Recurrent and Recursive Nets: Recurrent Networks as Directed Graphical Models



Fully connected graphical model for a sequence $y^{(1)}, y^{(2)}, y^{(3)} \ldots y^{(t)}, \ldots$: every past observation $y^{(i)}$ may influence the conditional distribution of some $y^{(t)}$ (for t > i), given the previous values.

Deep Learning

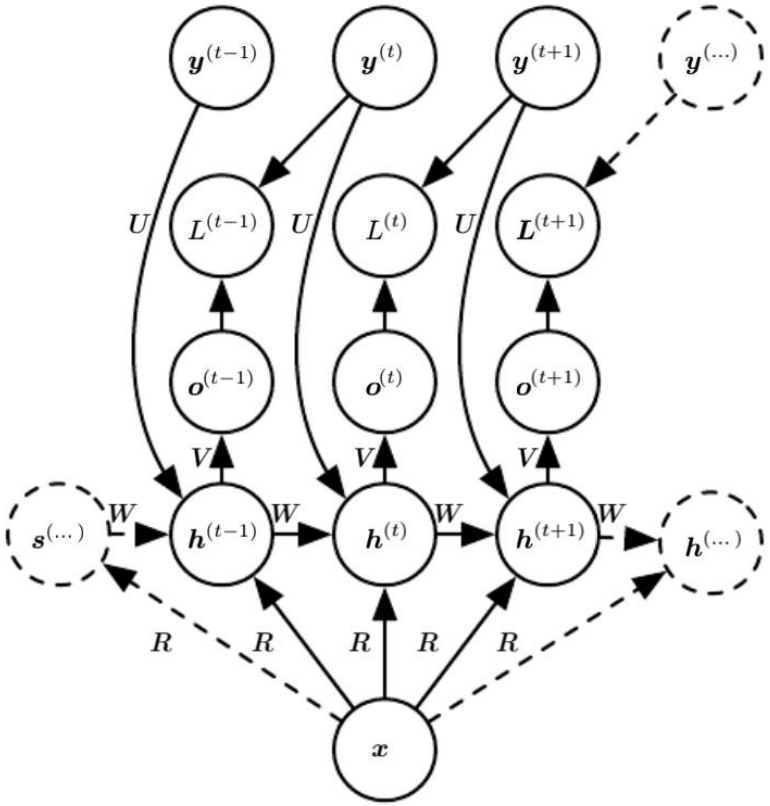
6.2 Recurrent and Recursive Nets: Recurrent Networks as Directed Graphical Models



Introducing the state variable in the graphical model of the RNN, even though it is a deterministic function of its inputs, helps to see how we can obtain a very efficient parametrization. Every stage in the sequence (for $h^{(t)}$ and $y^{(t)}$) involves the same structure (the same number of inputs for each node) and can share the same parameters with the other stages.

Deep Learning

6.2 Recurrent and Recursive Nets: Modeling Sequences Conditioned on Context with RNNs

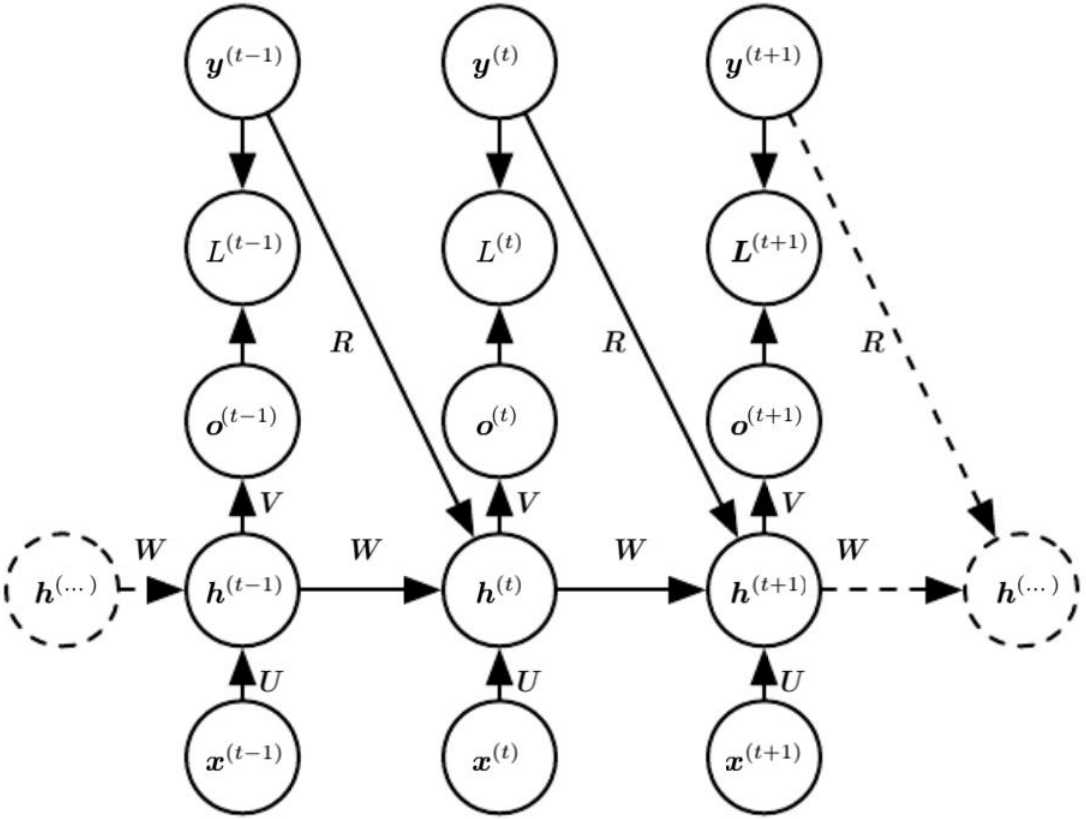


An RNN that maps a fixed-length vector $\boldsymbol{x}$ into a distribution over sequences $\boldsymbol{Y}$. This RNN is appropriate for tasks such as image captioning, where a single image is used as input to a model that then produces a sequence of words describing the image. Each element $y^{(t)}$ of the observed output sequence serves both as input (for the current time step) and, during training, as target (for the previous time step).

[1] Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville, http://www.deeplearningbook.org, Book in preparation for MIT Press,2016

Deep Learning

6.2 Recurrent and Recursive Nets: Modeling Sequences Conditioned on Context with RNNs



A conditional recurrent neural network mapping a variable-length sequence of $\boldsymbol{x}$ values into a distribution over sequences of $\boldsymbol{y}$ values of the same length. This RNN contains connections from the previous output to the current state. These connections allow this RNN to model an arbitrary distribution over sequences of $\boldsymbol{y}$ given sequences of $\boldsymbol{x}$ of the same length.

[1] Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville, http://www.deeplearningbook.org, Book in preparation for MIT Press,2016

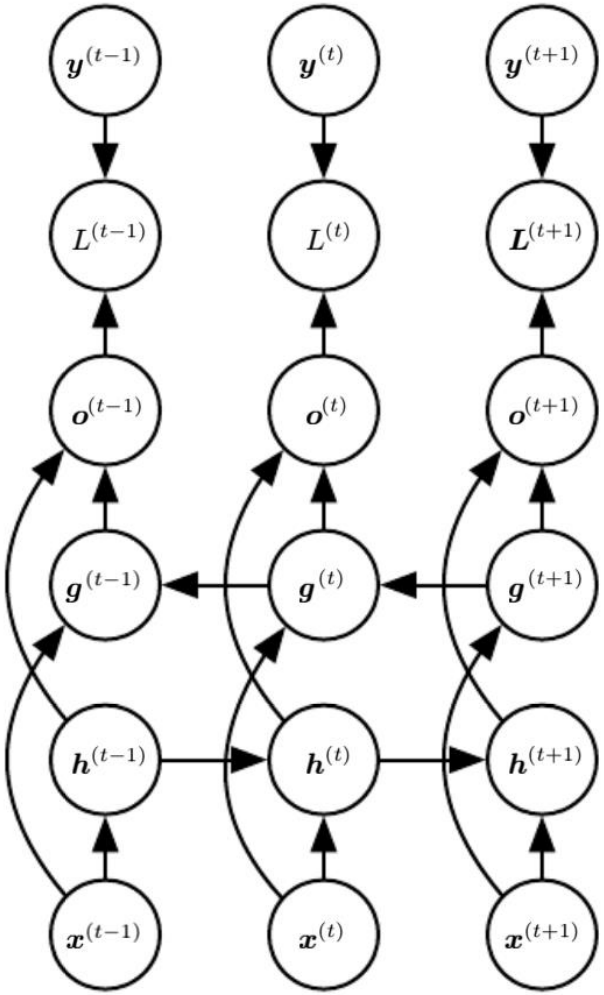
Deep Learning

6.2 Recurrent and Recursive Nets: Bidirectional RNNs

In many applications we want to output a prediction of y(t) which may depend on the whole input sequence. For example, in speech recognition, the correct interpretation of the current sound as a phoneme may depend on the next few phonemes because of co-articulation and potentially may even depend on the next few words because of the linguistic dependencies between nearby words: if there are two interpretations of the current word that are both acoustically plausible, we may have to look far into the future (and the past) to disambiguate them. This is also true of handwriting recognition and many other sequence-to-sequence learning tasks, described in the next section.

Deep Learning

6.2 Recurrent and Recursive Nets: Bidirectional RNNs



Computation of a typical bidirectional recurrent neural network, meant to learn to map input sequences $\boldsymbol{x}$ to target sequences $\boldsymbol{y}$, with loss $L^{(t)}$ at each step t. The $\boldsymbol{h}$ recurrence propagates information forward in time (towards the right) while the $\boldsymbol{g}$ recurrence propagates information backward in time (towards the left). Thus at each point t, the output units $\boldsymbol{o}^{(t)}$ can benefit from a relevant summary of the past in its $\boldsymbol{h}^{(t)}$ input and from a relevant summary of the future in its $\boldsymbol{g}^{(t)}$ input.

Part 7
More Topics

- How to Choose
- The history of RBF

Accuracy

Getting the most accurate answer possible isn't always necessary. Sometimes an approximation is adequate, depending on what you want to use it for. If that's the case, you may be able to cut your processing time dramatically by sticking with more approximate methods. Another advantage of more approximate methods is that they naturally tend to avoid overfitting.

Training time

The number of minutes or hours necessary to train a model varies a great deal between algorithms. Training time is often closely tied to accuracy—one typically accompanies the other. In addition, some algorithms are more sensitive to the number of data points than others. When time is limited it can drive the choice of algorithm, especially when the data set is large.

Source *https://azure.microsoft.com/en-us/documentation/articles/machine-learning-algorithm-choice/*

More Topics

7.1 How to Choose

Number of parameters

Parameters are the knobs a data scientist gets to turn when setting up an algorithm. They are numbers that affect the algorithm's behavior, such as error tolerance or number of iterations, or options between variants of how the algorithm behaves. The training time and accuracy of the algorithm can sometimes be quite sensitive to getting just the right settings. Typically, algorithms with large numbers parameters require the most trial and error to find a good combination.

The upside is that having many parameters typically indicates that an algorithm has greater flexibility. It can often achieve very good accuracy. Provided you can find the right combination of parameter settings.

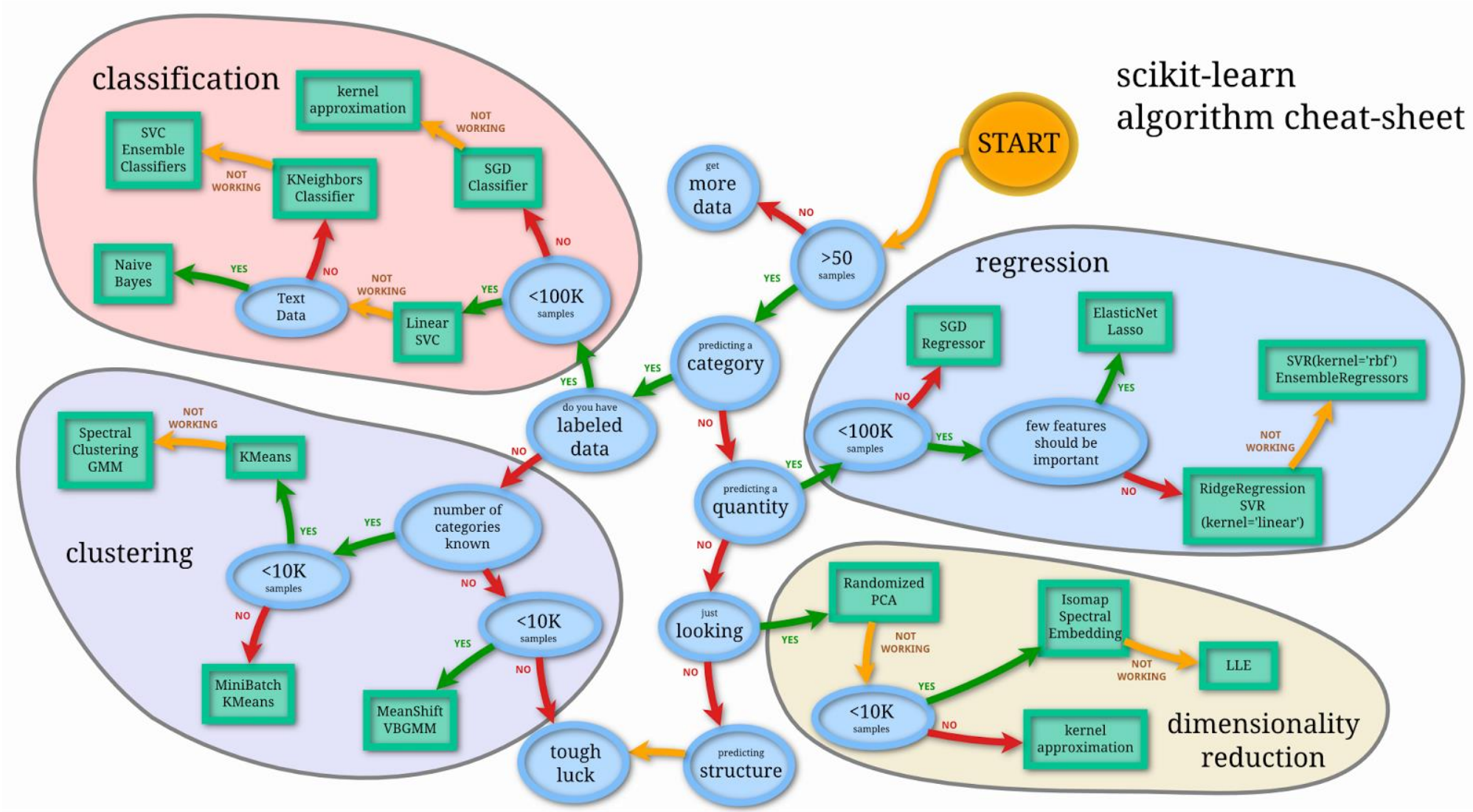
Number of features

For certain types of data, the number of features can be very large compared to the number of data points. This is often the case with genetics or textual data. The large number of features can bog down some learning algorithms, making training time unfeasibly long. Support Vector Machines are particularly well suited to this case.

Source *https://azure.microsoft.com/en-us/documentation/articles/machine-learning-algorithm-choice/*

More Topics

7.1 How to Choose



[1] http://peekaboo-vision.blogspot.jp/2013/01/machine-learning-cheat-sheet-for-scikit.html/

More Topics

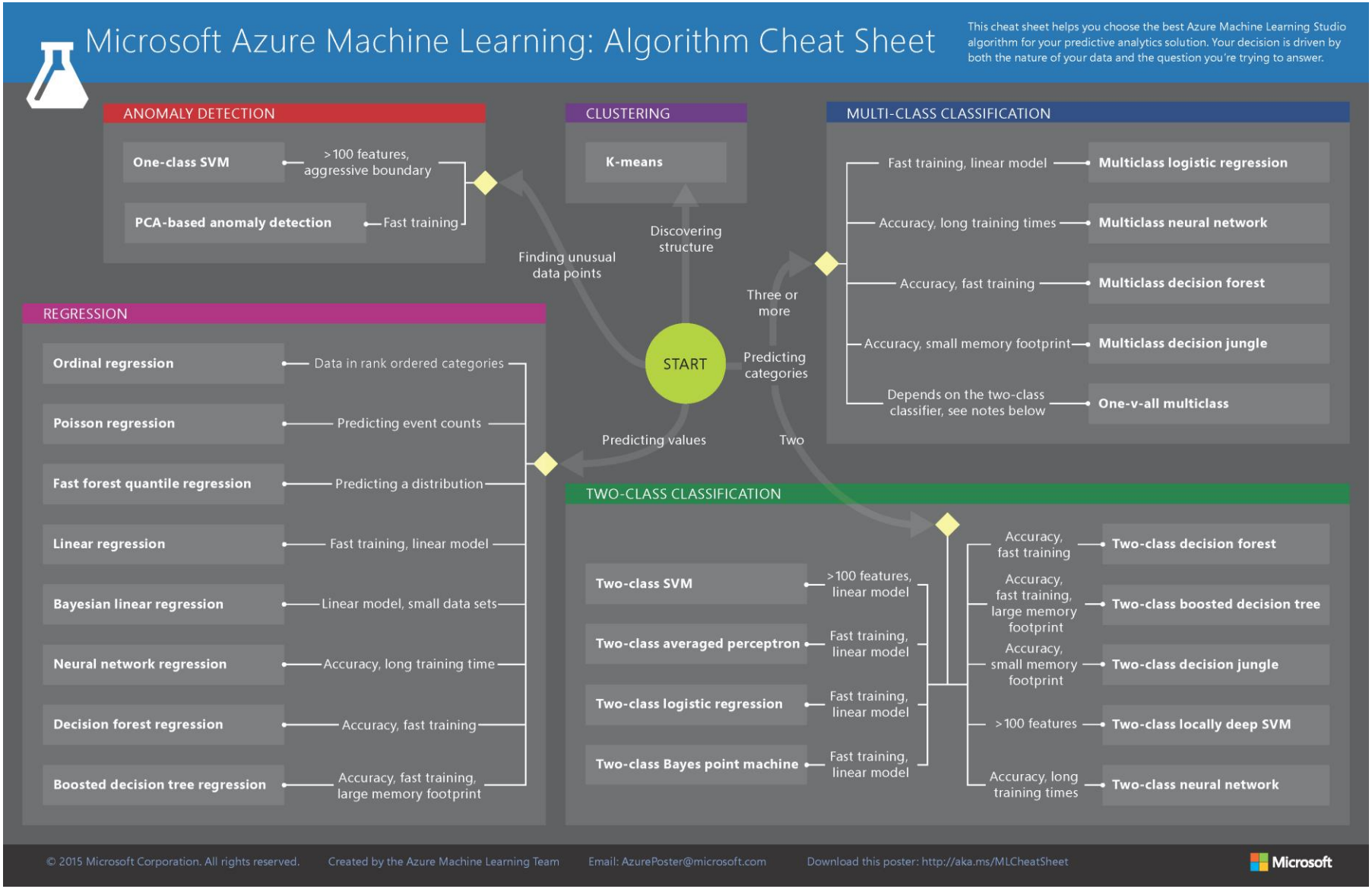
7.1 How to Choose

Algorithm	KNN	Linear regression	Logistic regression	Naive Bayes	Decision trees	Random Forests	AdaBoost	Neural networks
Problem Type	Either	Regression	Classification	Classification	Either	Either	Either	Either
Results interpretable by you?	Yes	Yes	Somewhat	Somewhat	Somewhat	A little	A little	No
Easy to explain algorithm to others?	Yes	Yes	Somewhat	Somewhat	Somewhat	No	No	No
Average predictive accuracy	Lower	Lower	Lower	Lower	Lower	Higher	Higher	Higher
Training speed	Fast	Fast	Fast	Fast (excluding feature extraction)	Fast	Slow	Slow	Slow
Prediction speed	Depends on n	Fast	Fast	Fast	Fast	Moderate	Fast	Fast
Amount of parameter tuning needed (excluding feature selection)	Minimal	None (excluding regularization)	None (excluding regularization)	Some for feature extraction	Some	Some	Some	Lots
Performs well with small number of observations?	No	Yes	Yes	Yes	No	No	No	No
Handles lots of irrelevant features well (separates signal from noise)?	No	No	No	Yes	No	Yes (unless noise ratio is very high)	Yes	Yes
Automatically learns feature interactions?	No	No	No	No	Yes	Yes	Yes	Yes
Gives calibrated probabilities of class membership?	Yes	N/A	Yes	No	Possibly	Possibly	Possibly	Possibly
Parametric?	No	Yes	Yes	Yes	No	No	No	No
Features might need scaling?	Yes	No (unless regularized)	No (unless regularized)	No	No	No	No	Yes

[1] *http://www.dataschool.io/comparing-supervised-learning-algorithms/*

More Topics

7.1 How to Choose



[1] *https://azure.microsoft.com/en-us/documentation/articles/machine-learning-algorithm-cheat-sheet/*

More Topics

7.2 The history of RBF

The history of RBFs goes back to 1971. Hardy innovated probably the most famous RBF, multi-quadratics (MQ) function, to deal with surface fitting on topography and irregular surfaces, and later considered the MQ as a consistent solution of biharmonic solution to give a physical interpretation of its prowess. In 1975, Duchon proposed another famous RBF, thin-plate splines (TPS), by employing the minimum bending energy theory of the surface of a thin plate. In 1982, Franke extensively tested 29 different algorithms on the typical benchmark function interpolation problems, and ranked the MQ-RBF and TPS-RBF as two of the best candidates based on the following criteria: timing, storage, accuracy, visual pleasantness of the surface, and ease of implementation. Since then, the RBFs have become popular in the scientific computing community, such as computer graphics, data processing, and economics.

In 1990, Kansa first developed an RBF collocation scheme for solving PDEs of elliptic, parabolic, and hyperbolic types, in particular, using the MQ. The methodology is now often called the Kansa method. This pioneering work kicks off a research boom in the RBFs and their applications to numerical PDEs. The Kansa method is meshless and has distinct advantages compared with the classical methods, such as superior convergence, integration-free, and easy implementation.

Chen, Wen, Zhuo-Jia Fu, and Ching-Shyang Chen. Recent advances in radial basis function collocation methods. Heidelberg: Springer, 2014.

More Topics

7.2 The history of RBF

It is interesting to point out that, prior to Kansa's pioneer work, Nardini and Brebbia in the early 1980s, without knowing the RBFs, applied the function 1+r, an ad hoc RBF, as the basis function in the dual reciprocity method (DRM) for effectively eliminating domain integral in the context of the boundary element methods (BEM). This original work gives rise to currently popular dual reciprocity BEM (DR-BEM).

On the other hand, the method of fundamental solutions (MFS) was first proposed by Kupradze and Aleksidze, also known as the regular BEM, the superposition method, desingularized method, the charge simulation method, etc. The MFS uses the fundamental solutions of the governing equation of interest as the basis functions. It is noted that fundamental solutions of radially invariant differential operators, like the Laplace or the Helmholtz operator, have radial form with respect to origin and appear like the RBFs. Thus, we can consider the fundamental solutions as a special type of the RBFs. Unlike the Kansa method, the MFS only requires the discretization at boundary nodes to solve the homogeneous problems. Hence the MFS is classified as a boundary-type RBF collocation method.

More Topics

7.2 The history of RBF

For nonhomogeneous problems, the MFS should be combined with additional techniques to evaluate the particular solution, i.e., Monte Carlo method, radial integration method, dual reciprocity method (DRM), and multiple reciprocity method (MRM), and so on. In the past decade, the DRM and the MRM have emerged as two promising techniques to handle nonhomogeneous term. For instance, the so-called DR-BEM and MR-BEM are very popular in the BEM community.

Being meshless and dimensionality-independent, quite a few RBF-based schemes for numerical solutions of PDEs have been developed in the last two decades. In contrast to the traditional meshed-based methods such as finite difference, finite element, and boundary element methods, the RBF collocation methods are mathematically very simple to implement and truly meshless, which avoid troublesome mesh generation for high-dimensional problems involving irregular or moving boundary. In general, the RBF-based methods can be classified into domain-type and boundary-type categories, such as the above-mentioned Kansa method and the MFS.

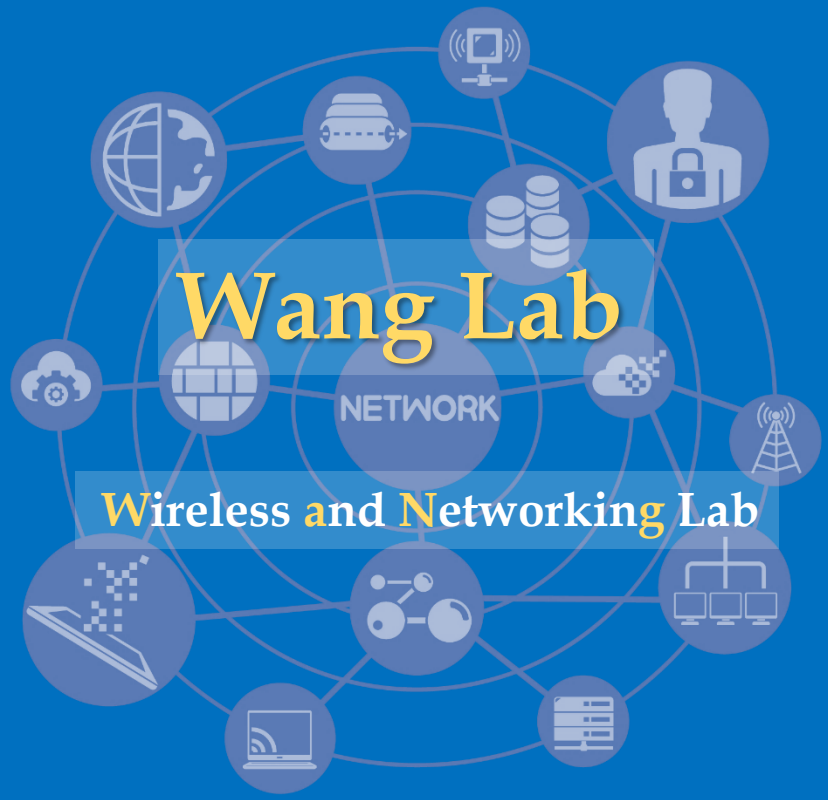
Chen, Wen, Zhuo-Jia Fu, and Ching-Shyang Chen. Recent advances in radial basis function collocation methods. Heidelberg: Springer, 2014.

More Topics

7.2 The history of RBF

Despite numerous successful applications in a wide range of fields, the traditional RBFs still encounter some disadvantages compared with the other numerical techniques. In implementation, the construction and the use of efficient and stable distance functions are often intuitive and largely based on experiences. For instance, although the MQ enjoys the reputation of spectral accuracy, the determination of its optimal shape parameter is often problem-dependent and remains an open issue, and no mature mathematical theory and methodology are so far available for its applications to various problems. The compactly supported RBFs (CS-RBFs) are a recent class of potentially important RBFs. Although in theory the CS-RBFs can result in a sparse interpolation matrix, their lower order of accuracy causes some concern to its practical use. Recently, the Green second identity was found to be a powerful alternative tool to create and analyze the kernel-based RBFs. In addition, one can find several books on the mathematical analysis of the RBFs.

Chen, Wen, Zhuo-Jia Fu, and Ching-Shyang Chen. Recent advances in radial basis function collocation methods. Heidelberg: Springer, 2014.



Machine Learning Algorithms

Peng Shi

2016.11.15

Thanks!