ANKAM

# Introduction to Machine Learning

and its

*Applications to Biomedical Sciences*

Saurabh Singal

25 AUGUST, 2016

## The Very First Clinical Trial

King Gustav III of Sweden viewed coffee consumption as being dangerous.

- The king ordered the experiment to be conducted using two identical twins who had been condemned to death.
- One twin was made to drink three pots of coffee every day, and the other, tea.
- Two physicians  supervised this procedure.
- Both doctors died & Gustav III was assassinated  before seeing the final results.
- Of the twins, the tea drinker was the first to die, at age 83!

## “Pain Points” in  the Drug Design Process

- ❑Drug development process is long and costly with low probability of success.
- ❑Cost of developing a drug is between 2 and 3 billion dollars
- ❑ The entire process can take up to 10 years.
- ❑Many Clinical trials fail but this does not necessarily mean the drug was bad.
- ❑Analysis of the Clinical Trials data may lead us to golden nuggets. For example, FDA failed drug candidates could be re-analyzed to determine if there is a subpopulation of patients for which the drug-candidate is effective.
- ❑For every patient that responds to treatment by a drug, between 4 to 24 patients do not respond to the same drug- leading to “trial and error” approach in treatment.

## Addressing the “Pain Points” in Drug Design

- ❑Next Generation Sequencing (NGS) is becoming more mainstream and producing an avalanche of “Big Data”.
- ❑The data are not only big in size but also have very high dimension. Traditional Statistical techniques are ill suited to handle this type and volume of Data.
- ❑Predictive Algorithms and Techniques from Machine Learning which have proved successful on Image Recognition and Natural Language Processing can help in identifying important biomarkers, which  can lead to targeted therapies.
- ❑Intersection of Machine Learning with Molecular Biology to analyze Big Data is the key to Personalized Medicine.

## What is Artificial Intelligence?

- ❑ Field that deals with the design and application of algorithms for the analysis of, learning from and interpreting data.
- ❑ AI encompasses many branches of

➢ statistical learning,

➢ pattern recognition,

➢ clustering, and similarity-based methods,

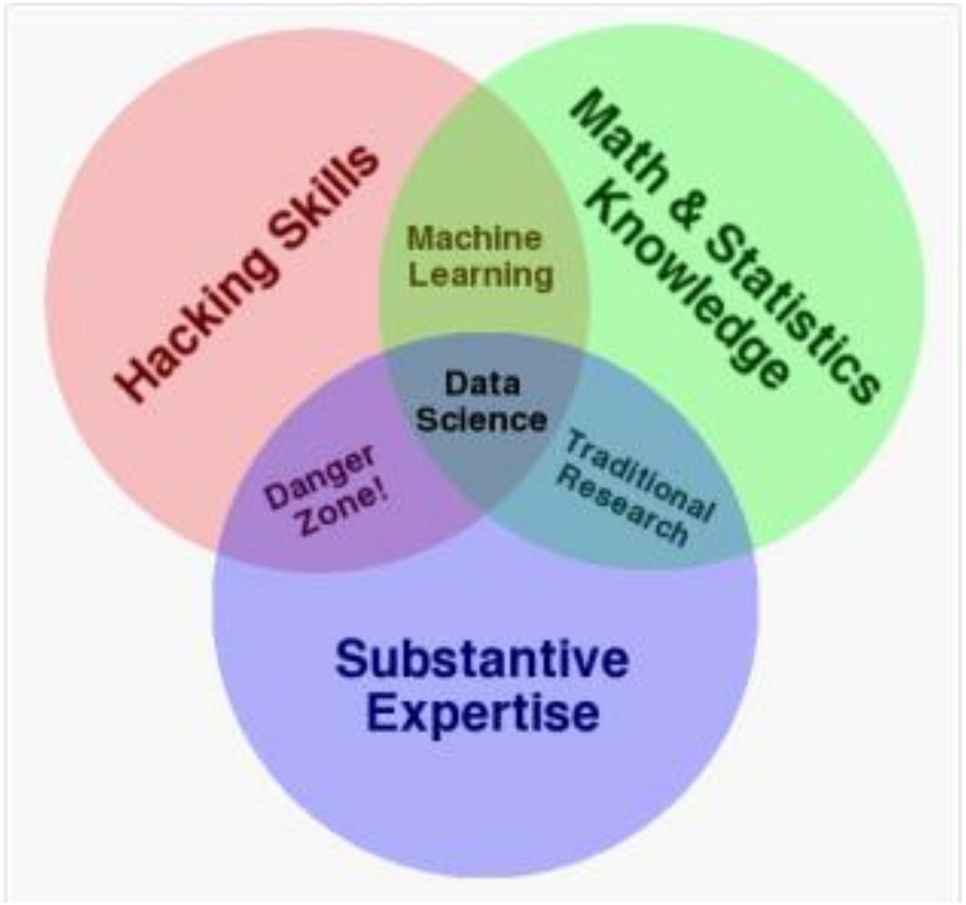
➢ biologically motivated approaches, such as neural networks,

➢ evolutionary computing or fuzzy modeling,

These are collectively described as ***Artificial Intelligence or Machine Learning***

*The field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience.*

## What is  Data Science ? ANKAM+Santen!



## Background : Applications of Machine Learning

Machine Learning can solve a wide array of real-life problems:

**Finance**

- Financial modeling
- News  Sentiment Analysis
- Trading signals
- Fraud detection

**Biology**

- Patient behaviors and disease
- New Drug Design
- Personalized Medicine
- computer-aided diagnosis

**Retail and Marketing**

- Item recommendation
- Market segmentation
- Targeted Ads
- Buyer Sentiment Analysis

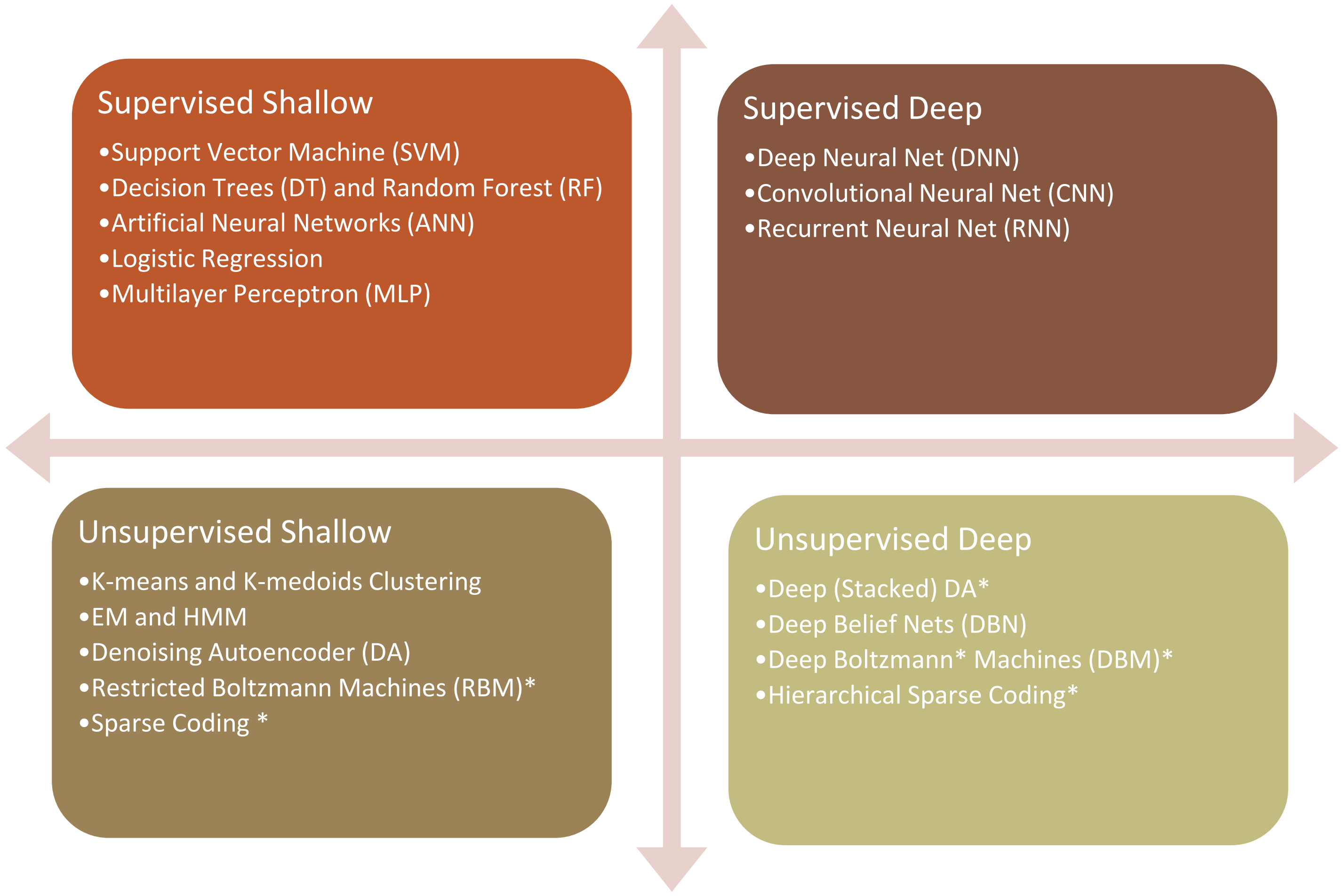
**Vision**

- Face recognition
- Handwriting Recognition
- Image Segmentation
- General Object Classification
- Personal Assistant

**Language**

- Language Translation
- Text Classification
- Sentiment Analysis
- Question Answering
- Summarization

# Taxonomy of Popular Machine Learning Algorithms



### Supervised Shallow

- Support Vector Machine (SVM)
- Decision Trees (DT) and Random Forest (RF)
- Artificial Neural Networks (ANN)
- Logistic Regression
- Multilayer Perceptron (MLP)

### Supervised Deep

- Deep Neural Net (DNN)
- Convolutional Neural Net (CNN)
- Recurrent Neural Net (RNN)

### Unsupervised Shallow

- K-means and K-medoids Clustering
- EM and HMM
- Denoising Autoencoder (DA)
- Restricted Boltzmann Machines (RBM)*
- Sparse Coding *

### Unsupervised Deep

- Deep (Stacked) DA*
- Deep Belief Nets (DBN)
- Deep Boltzmann* Machines (DBM)*
- Hierarchical Sparse Coding*

* Supervised version also exists

## How is AI used in Drug Discovery

❑Artificial intelligence is used to  streamline drug discovery and drug repurposing processes and significantly cutting time to market.

❑The speed of AI in these processes allows companies to develop drugs based on biological markers, with greater accuracy, rather than the scattergun approach of chemical screening. In this way, companies can zero in on particular indications which the drug is most likely to successfully treat.

❑It follows a logic-based, machine-learning approach that trains computers to learn from a set of biologically-active molecules, combining knowledge of traditional drug discovery methods and computer-based analyses.

❑ It focuses on the key stages in drug discovery of target identification, lead validation, and lead optimisation.

## Introducing the Iris Dataset

- ❑ The Iris dataset consists of lengths and widths of sepals and petals of three species of Iris: Iris Setosa, Iris Versicolour, and Iris Virginica.

- ❑ There are 50 examples of each species for a total of 150 examples in the entire dataset.

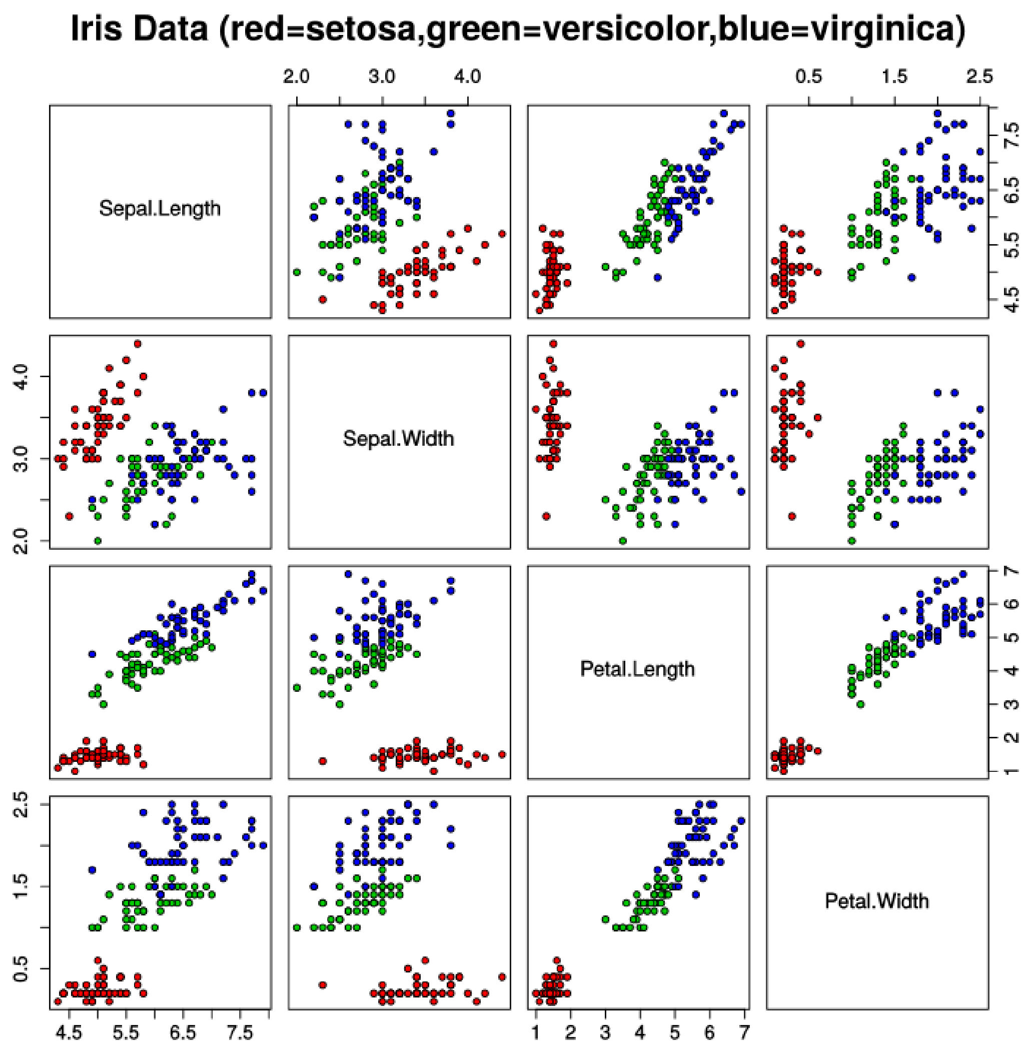
- ❑ The feature vectors are 4-dimensional, consisting of the sepal length, sepal width, petal length, and petal width.

- ❑ One class of Iris is linearly separable; however, two classes are not linearly separable.

## Uses of Iris Data.

- First used by R.A. Fisher for developing the linear discriminant model, then became a typical test case for many statistical classification techniques
- Now also used in testing machine learning algorithms such as support vector machines.
- This data set cannot be used in unsupervised learning such as cluster analysis as the data set only contains two clusters with rather obvious separation. One of the clusters contains Iris setosa, while the other cluster contains both Iris virginica and Iris versicolor and is not separable without the species information.
- Good example to explain the difference between supervised and unsupervised techniques in data mining: Fisher's linear discriminant model can only be obtained when the object species are known: class labels and clusters are not necessarily the same

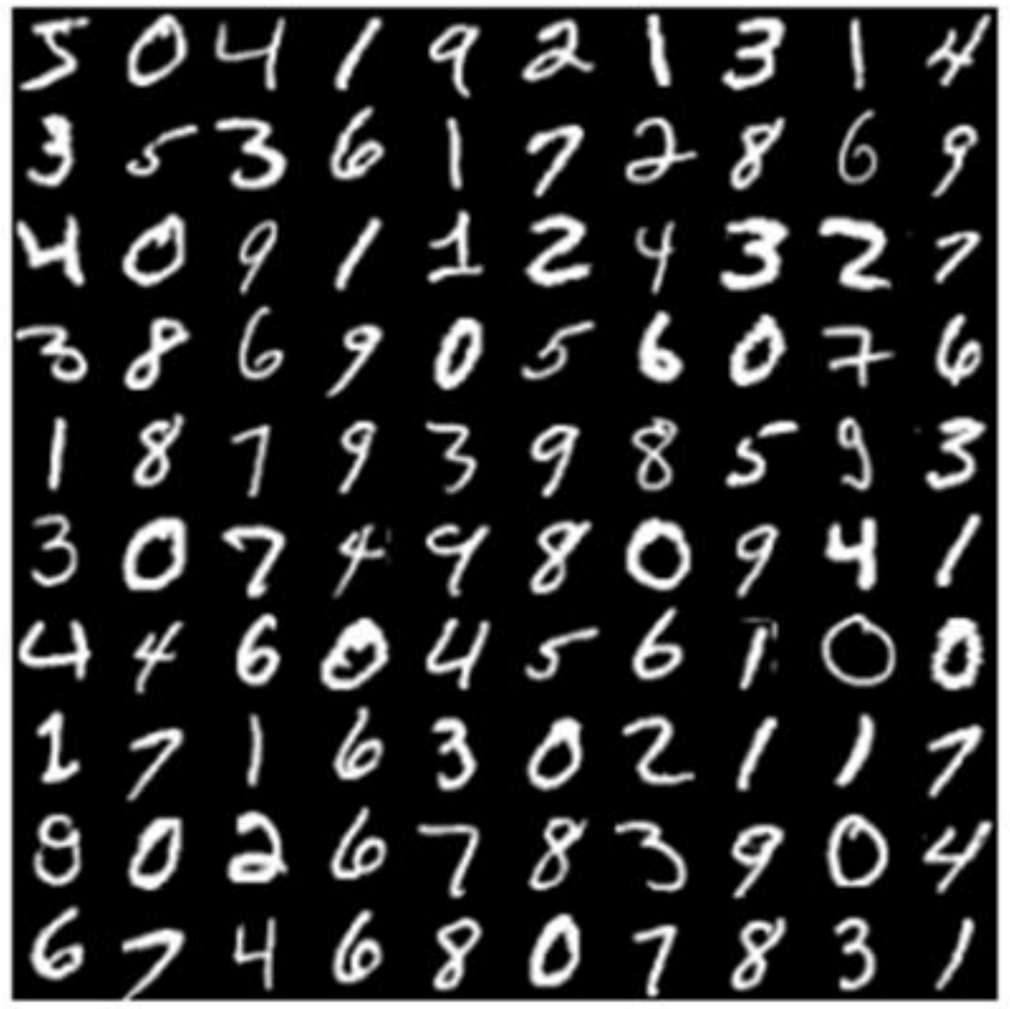
## Fisher's Iris Data Visualized



## The MNIST Handwritten Digits

- The MNIST database (Mixed National Institute of Standards and Technology database) is a large database of handwritten digits
- It is commonly used as a benchmark for  image processing algorithms
- It contains 60,000 training images and 10,000 testing images
- Each image is a *28 x 28* grayscale images that when flattened yields a feature vector of 784-dimensions.
- CIFAR 10 dataset a collection of 60,000 *32 x 32* RGB images in 10 classes, for a total of 6,000 images per class.

## Sample of MNIST Handwritten Digits



## Topics in Machine Learning: Cross-Validation Explained

- ❑ Cross Validation is a **technique for model validation or assessing how our model will perform on new, unseen data instances.**
- ❑ The goal of cross validation is to limit overfitting.
  - ❖ A model is usually fitted using *known data* (*training dataset*)
  - ❖ Then the model is tested against unknown data (*testing dataset*).
  - ❖ But we can reserve a part of training data and not use it for training; instead we use this for validation , which means “test during training”.

.

## Topics in Machine Learning: Leave-one-out Cross Validation & Jackknife

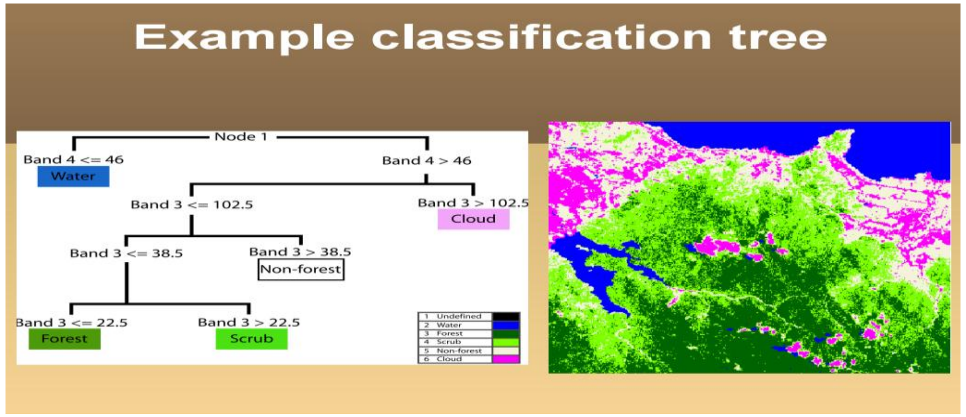
- **Exhaustive cross-validation** methods are cross-validation methods which learn and test on all possible ways to divide the original sample into a training and a validation set.
- The **leave one out cross validation** is similar to the **Jackknife**. The jackknife estimator of a parameter is found by systematically leaving out each observation from a dataset and calculating the estimate and then finding the average of these calculations. Given a sample of size *N*, the jackknife estimate is found by aggregating the estimates of each *N-1* estimate in the sample.
- NOTE: with cross-validation we compute a statistic on the left-out samples, while with jackknifing we compute a statistic from the kept samples only.

## Topics in Machine Learning: *k*-fold Cross-Validation

- Non-exhaustive cross validation methods do not compute all ways of splitting the original sample. Those methods are approximations of *leave-p-out cross-validation*.
- For example, one scheme is ***k*-fold Cross-Validation**. In *k*-fold cross-validation, the original sample is randomly partitioned into *k* equal sized subsamples. Of the *k* subsamples, a single subsample is retained as the validation data for testing the model, and the remaining *k* − 1 subsamples are used as training data. The cross-validation process is then repeated *k* times (the *folds*), with each of the *k* subsamples used exactly once as the validation data. The *k* results from the folds can then be averaged to produce a single estimation.
  - ❑All observations are used for both training and validation, and each observation is used for validation exactly once. The most common value f k is 10, resulting in 10-fold.
  - ❑When *k*=*n* (the number of observations), the *k*-fold cross-validation is exactly the leave-one-out cross-validation.

## What is a Decision Tree?

- Decision tree is a predictive model that uses a set of binary rules applied to calculate a target value. It can be used for classification or regression.
  - In Classification, categorical variables are used. E.g., whether a tumor is benign or malignant.
  - In Regression applications, continuous variables are used. E.g., the molecular activity for a compound on a target.
- A recursive tree generating algorithm is used to learn a Decision Tree, and the "best" branching policy is the one that results in the **largest information gain**. To avoid overfitting, pruning is used. Techniques like cross validation can be used for model validation and testing.
- Advantage of decision tree
  - Easy to interpret
  - Robust as regard to outliers in the training data
  - Prediction is fast once rule is developed

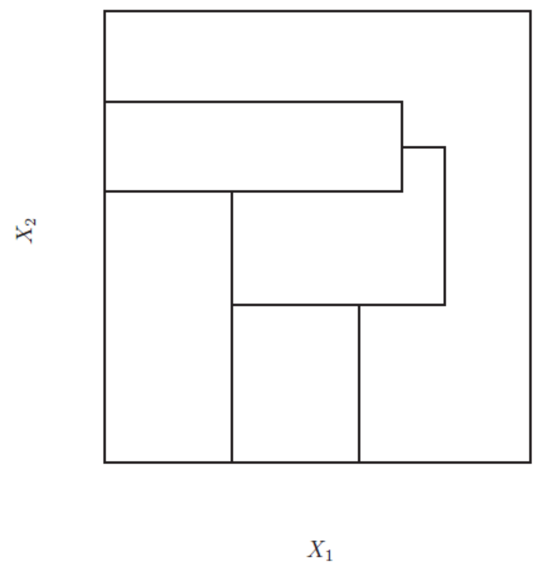


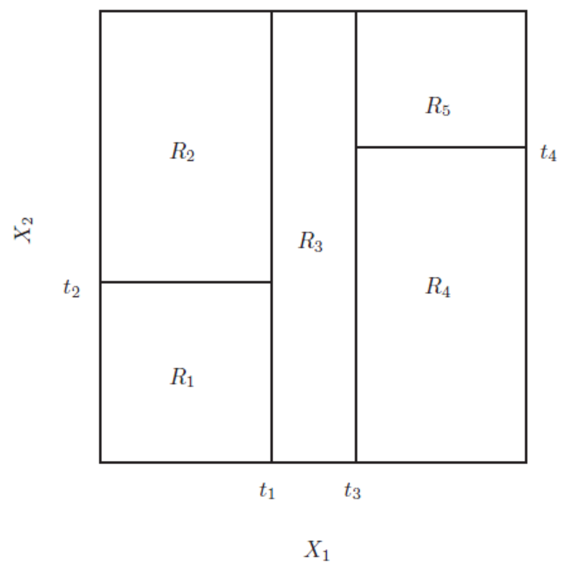
## What is a Decision Tree ? Contd.

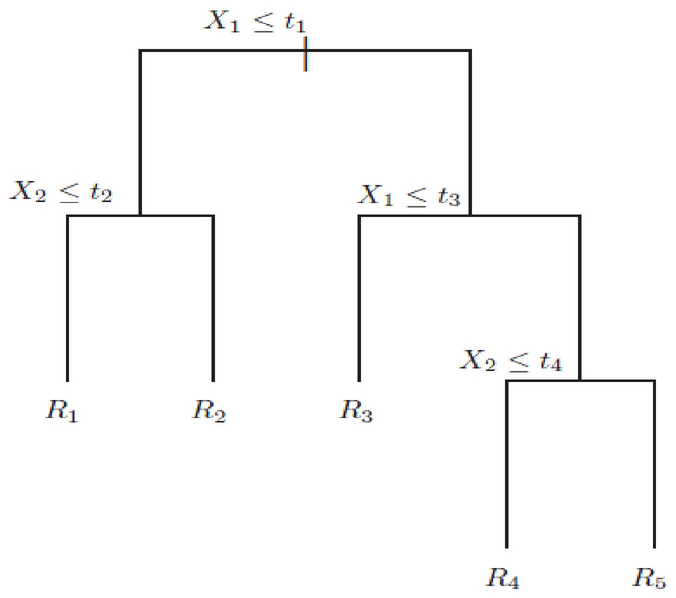
- The goal is minimize the RSS which is $$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$
- A *top-down*, *greedy* approach that is known as *recursive binary splitting.*
- The Recursive binary splitting approach is *top-down* because it begins at the top of the tree (at which point all observations belong to a single region) and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is *greedy* -at each step, the *best* split is made at that particular step, rather than looking ahead. It chooses the **myopically optimal** split: if the learner were only allowed one split, which single split would result in the best classification at the current step?
- The optimal split is the one that gives the maximum **information gain**. Sometimes information gain is used even when the optimality criterion is the sum-of-squares error. An alternative is to use the **Gini index** (a measure of statistical dispersion commonly used measure of inequality, especially income inequality.)

## What is a Decision Tree ? Contd.-2

The partitioning in the left figure cannot be achieved by Recursive Binary partitioning, where as the middle one can. The associated tree for the middle partitioning is shown in the figure on right







## Algorithm to Create a Decision Tree

1. Use recursive binary splitting to grow a large tree on the training data
2. Stop when each terminal node has fewer than some minimum number of observations.

3. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.

4. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each k = 1, . . .,K:

 (a) Repeat Steps 1 to 3 on all but the *k-th* fold of the training data.

 (b) Evaluate the mean squared prediction error on the data in the left-out *kth* fold, as a function of α.

Average the results for each value of α, and pick α to minimize the average error.

5. Return the subtree from Step 3 that corresponds to the chosen value of α.

## Topics in Machine Learning: Bagging Explained

- Bagging is short form for **Bootstrap Aggregating (bootstrapping** can refer to any test or metric that relies on resampling/random sampling with replacement.)
- This is a general technique used to reduce variance in ensemble learning and to control model overfitting.
- Bagging relies on the fact that averaging a set of observations reduces variance. If we had many training sets, we could build a separate prediction model using each training set, and average the resulting predictions. But we have only one training set. Therefore we create multiple training sets by **sampling uniformly with replacement** from the original training data.
- Decision trees suffer from *high variance*- if we split the training data into two parts at random, and fit a decision tree to both halves, the results that we get could be quite different. By bagging, this can be reduced..Random Forest is an example of Bagging applied to Decision Trees.
- Bagging is most often used in Decision Trees but also in Neural Networks and can be used even in linear regression.

## Topics in Machine Learning: Boosting Explained

- Boosting is  a paradigm in ensemble  learning to convert weak learners to strong learners.
- Boosting  is the answer (YES!) to the question: Can a set of weak learners create a single strong learner?
  - A weak learner is defined to be a classifier which is only slightly correlated with the true classification; a strong learner is arbitrarily well-correlated with the true classification.
  - Weak learners are combined but they are not all given the same weight- misclassified examples are given higher weights and re-training is performed.
  - Specific  schemes for model averaging using Boosting are ADABoost and BrownBoost
  - Boosting also reduces model variance.

## Topics in Machine Learning: Regularization Explained

- We want to avoid overfitting a model.
- Usually, model fitting proceeds by minimizing a “loss function”, which depends upon the difference between the target $\boldsymbol{y_j}$ & its predicted value, $\boldsymbol{\hat{y}_j}$

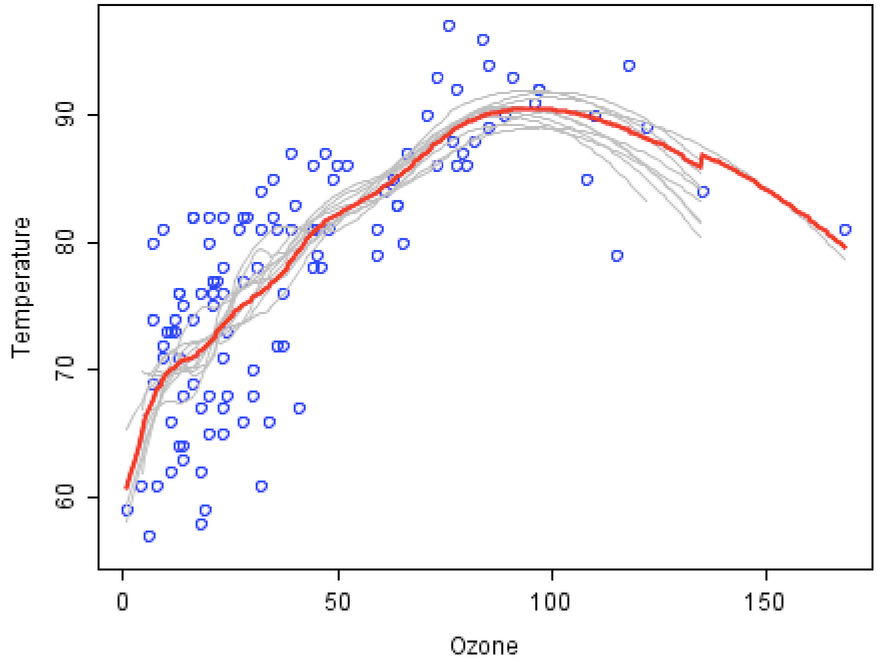
  Loss $= \sum_{j=1}^{N}\left(y_j - \hat{y}_j)\right)$
- When we minimize the loss function, we aim to find good values of model parameters (or weights).
- To prevent overfitting, we will add a regularization term λ*R(f(x))* to the loss function. Regularization term is a penalty on the complexity of our fitting function. The coefficient λ controls how much weight we want to give to the regularization term.
- The two most standard Regularization schemes are:

i. $L_1$ or the absolute value (“lasso”) penalty

ii. $L_2$ or quadratic (“ridge”) penalty

## What is a Random Forest (RF)?

- The **Random Forest** (Breiman, 2001) [1] is an Ensemble approach that uses many Decision Trees (weak learners) to form a refined final classification or regression (strong learner) that is more stable and accurate than all the individual decision trees.
- To train a RF model, a different subset of the training samples is selected ( approx. 2/3 of the original data) with replacement to train each tree. Final classification is made by majority voting and regression is made by tree averaging.
- In contrast to a single Decision Tree, RF is robust to data overfitting and thus no tree pruning is required. It is also robust as regard to outliers in the training data.



The ensemble strong learner (red curve) trained using Random Forest.

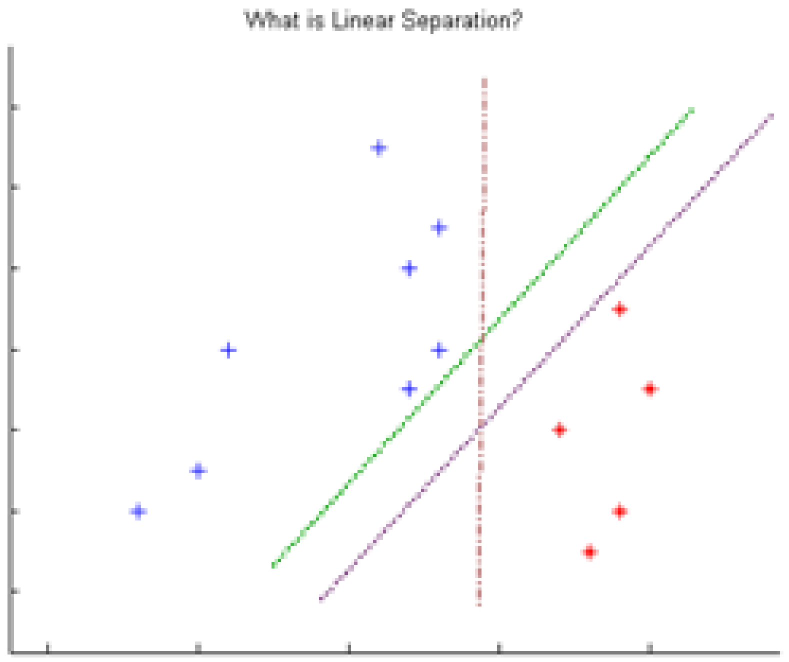
## What are Support Vector Machines?

- The Support Vector Machine (SVM) is a technique for classification and regression. Originally the SVM was devised for binary classification, or classifying data into two types. Generalization when there are more than two classes is relatively straightforward.
- For linearly separable data, SVM finds optimal decision boundary using a linear decision surface. When working with non-linearly separable data in the original space, SVM maps the patterns to a higher dimensional feature space in which the transformed data becomes linearly separable. This conversion can be done using kernel function, and the commonly used kernels functions are listed below:

| Name of Kernel Function | Definition |
|---|---|
| Linear | $K(\mathbf{u},\mathbf{v}) = \mathbf{u}^T\mathbf{v}$ |
| Polynomial of degree *d* | $K(\mathbf{u},\mathbf{v}) = (\mathbf{u}^T\mathbf{v}+1)^d$ |
| Gaussian Radial Basis Function (RBF) | $K(\mathbf{u},\mathbf{v}) = e^{-\frac{1}{2}[(\mathbf{u}\text{-}\mathbf{v})^T\Sigma^{-1}(\mathbf{u}\text{-}\mathbf{v})]}$ |
| Sigmoid | $K(\mathbf{u},\mathbf{v}) = \tanh[\mathbf{u}^T\mathbf{v}+b]$ |

- Solution to SVM can be formulated as a Quadratic Programming Problem. It can be easily implemented by most of the popular statistical languages (MATLAB, R, etc.) or packages (LibSVM).

## SVM and Linear Separation



*Red Asterisk markers and Blue Plus markers are patterns belonging to Class 1 and 2 respectively. Each of the three Straight Lines can separate the test patterns. This is an example of Linear Separation.*

## SVM and Linear Discrimination

Consider the function $g(\mathbf{x})=\mathbf{w}^T\mathbf{x} + b$.

If $\mathbf{w}^T\mathbf{x} + b>=0$, classify **x** as belonging to class 1

If $\mathbf{w}^T\mathbf{x} + b<0$, classify **x** as belonging to class 2

In the case of two-dimensional **x** and **w,** $\mathbf{w}^T\mathbf{x} + b=0$ defines a straight line. Points on one side of this straight line will be classified as belonging to class 1; points on the other side of this line will be classified as belonging to class 2.

But there are an infinite number of straight lines that can linearly separate the data points; we can simply vary b to get parallel lines that will do the job. So we need to determine the "best" or optimal **w** and *b*

## SVM and Maximum Margin Separation

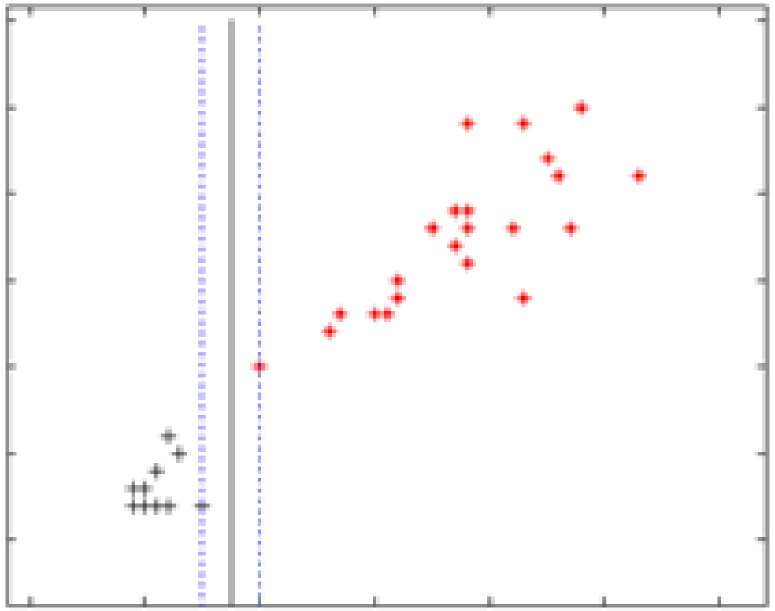
To choose "good" w and b, we measure the distance r(x) of x from the decision surface g(x)=0. The distance r, of a point x from the plane P specified by (w, b) is

$r(\mathbf{x};\mathbf{w},b) = |g(\mathbf{x})|/||\mathbf{w}|| = |\mathbf{w}^T\mathbf{x} + b|/||\mathbf{w}||$

When we talk of the distance from a point to a plane we mean the distance from x to the nearest point xp that lies on the plane P. The margin of separation, M, measures the distance between the two classes; $M=2/||\mathbf{w}||$

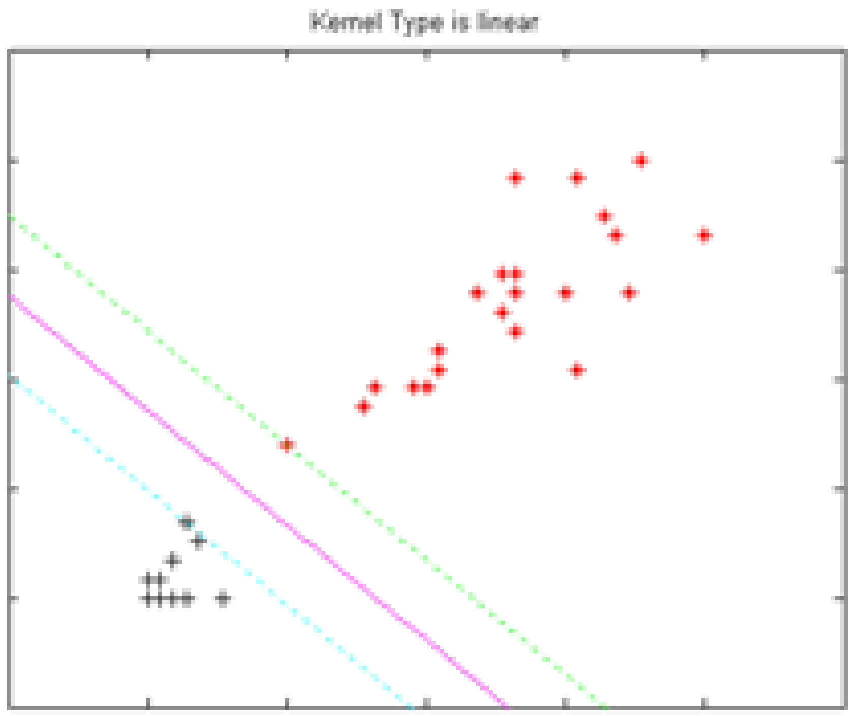
The ***optimal separating hyperplane*** separates the two classes and maximizes the distance to the closest point from either class. This provides a unique solution to the separating hyperplane problem. By maximizing the margin between the classes, it leads to better classification.

## SVM and Maximal Margin illustrated-1



*Separating Hyperplane (solid line) with Narrower Margin.*
*Margin is the distance between the dotted lines*

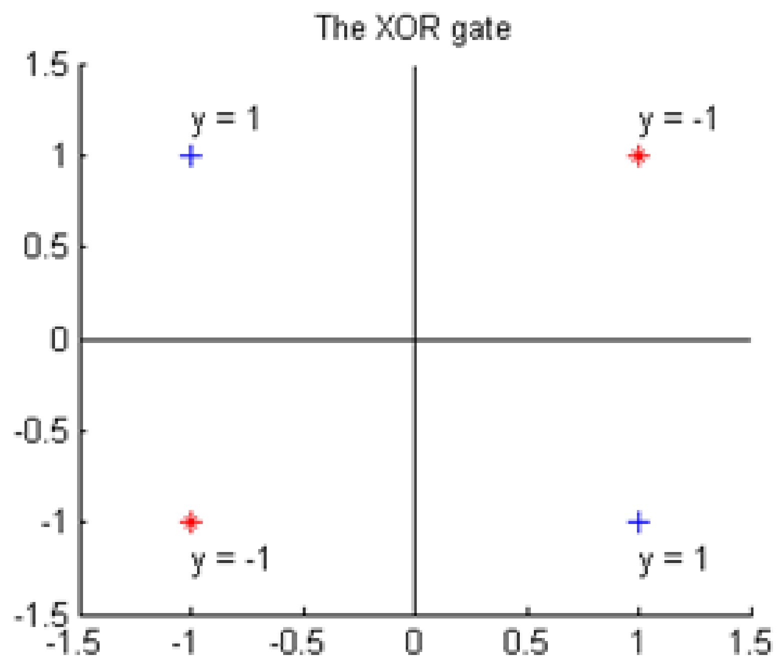
## SVM and Maximal Margin Illustrated-2



*Separating Hyperplane (solid magenta line) with Wider Margin.*
*Margin is the distance between the dotted lines*

## SVM and Data which are not Linearly Separable

The true power of SVMs comes into play when we have data points that are not separable by a linear decision surface. A canonical example is the XOR function (exclusive OR), which takes a value of 0 when both its inputs are the same and takes a value of 1 otherwise.



The XOR Gate.
There is no straight line that can separate the two classes.

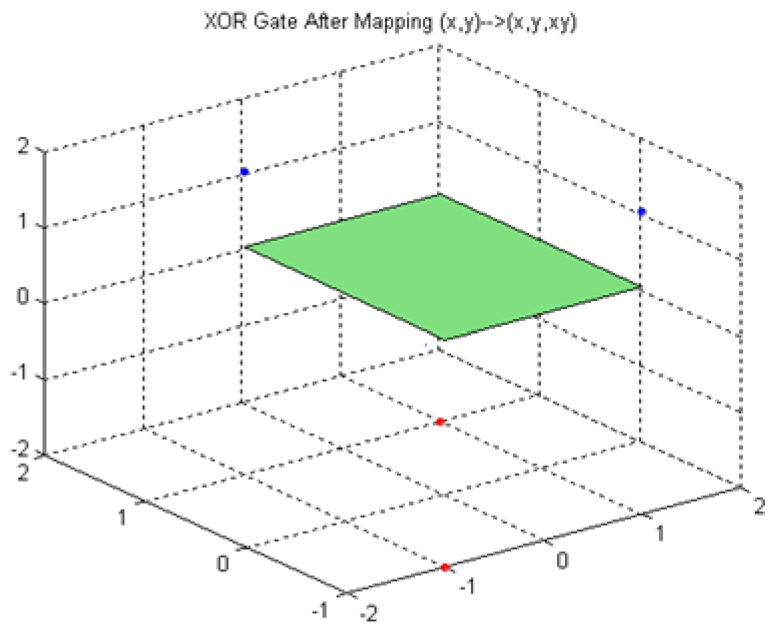
## SVM and Mapping to a Higher Dimension

These points cannot be linearly separated, that is, we cannot draw any straight line that will separate the classes. But there is a trick we can use; we define x3= x1.x2 and then augment the 2 dimensional input vectors by this third number, x3. In our XOR example, we get

| x=(x1,x2,x1.x2) | y |
|---|---|
| (1,1,1) | -1 |
| (-1,-1,1) | -1 |
| (1,-1,-1) | 1 |
| (-1,1,-1) | 1 |

*Table 2: The modified XOR Gate*

## SVM and Mapping to Higher Dimension-2



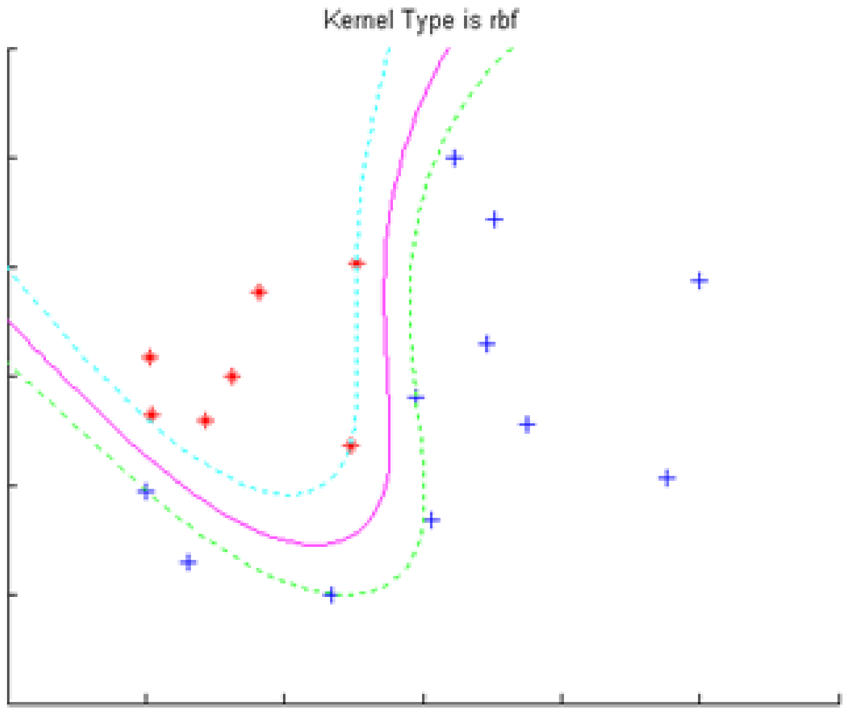
*Separation of the XOR Gate After Mapping to Higher Dimension.*

## SVM and the Kernel Trick

- SVM’s will use a non-linear function to map the training vectors or data points into a higher dimensional space.
- This higher dimensional space is called the Feature Space.
- We can then find and use a linear decision surface in the feature space, and this allows for non-linear separation in the original space.
- In our example, we used the mapping $\boldsymbol{\Psi}$ to map the two dimensional pattern space to a three dimensional feature space.

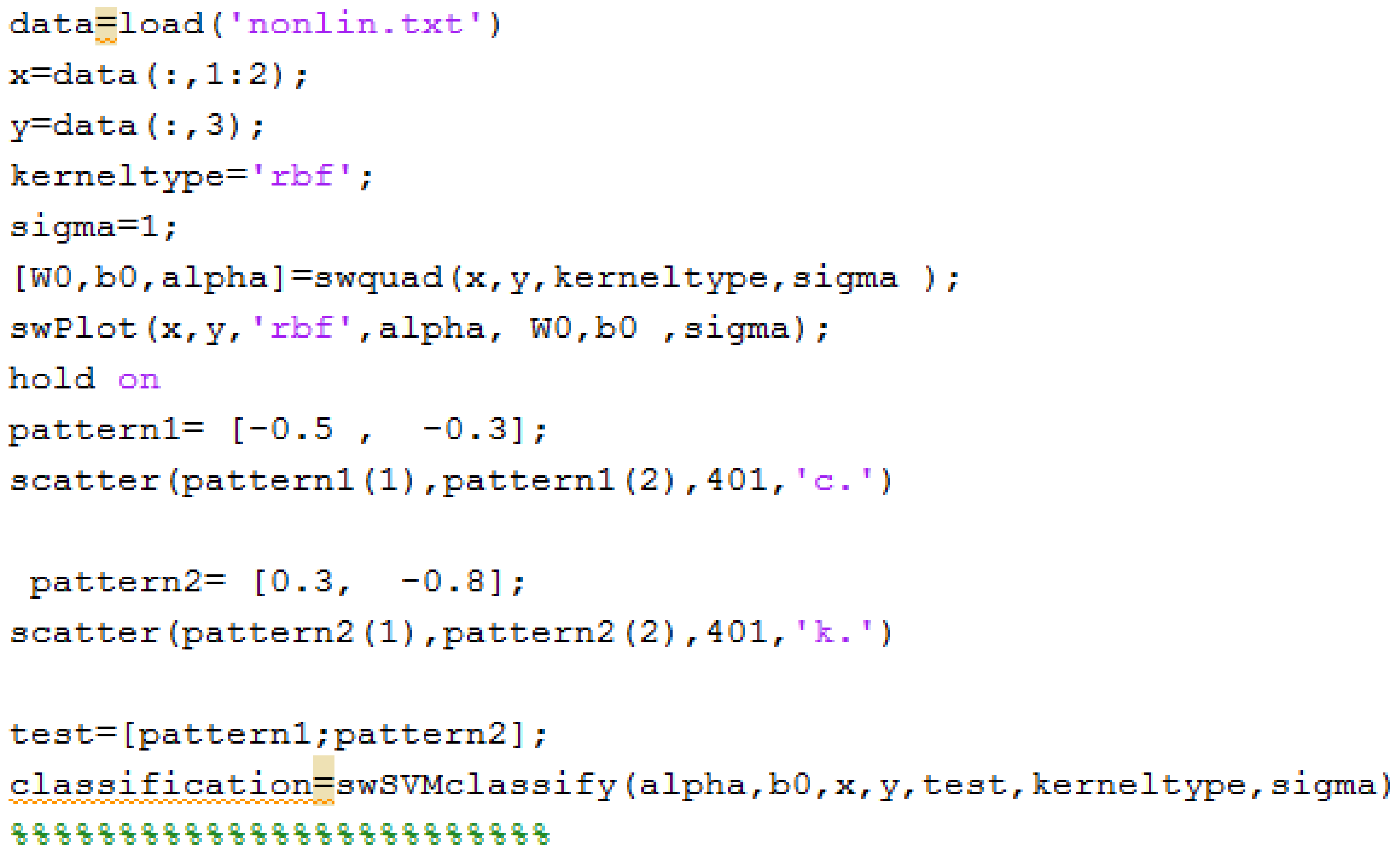
$$\Psi(u,v) \longmapsto (u,v,uv)$$

## SVM and non-linear separation using kernel to map to higher dimension



*Mapping to Higher Dimensional Feature Space Using RBFs Permits Non-Linear Separation.*

## MATLAB code for SVM (previous example)



```
data=load('nonlin.txt')
x=data(:,1:2);
y=data(:,3);
kerneltype='rbf';
sigma=1;
[W0,b0,alpha]=swquad(x,y,kerneltype,sigma );
swPlot(x,y,'rbf',alpha, W0,b0 ,sigma);
hold on
pattern1= [-0.5 ,  -0.3];
scatter(pattern1(1),pattern1(2),401,'c.')

 pattern2= [0.3,  -0.8];
scatter(pattern2(1),pattern2(2),401,'k.')

test=[pattern1;pattern2];
classification=swSVMclassify(alpha,b0,x,y,test,kerneltype,sigma)
%%%%%%%%%%%%%%%%%%%%%%%%%
```

## Muti-class Classification by SVM’s

- The two simple approaches are
  - One vs All (OVA) : Build K “one vs all”  classifiers and choose the class which classifies the test datum with greatest margin. One vs Rest would have been a more appropriate name.
  - All vs All  (AVA) : Build K(K-1)/2 pairwise binary classifiers  and choose the class that is selected by the most classifiers. One vs One would have been a more fitting name.
  - AVA is often faster even though it has $O(\mathbf{N}^2)$ classifier
  - Another approach is a structural  SVM which will not be described today.

## Hidden Markov Model (HMM): Informal Description

- Suppose we have a set of N urns, each with balls of M colours in different proportions.
- I randomly choose an urn and then select a ball from this urn with replacement.
- You get to see the colour of the ball but not which urn the ball comes from.
- Then, I choose another urn and select another ball.
- The process is repeated.
- This is a **Hidden Markov Model**.
- The urns are the "**hidden states**" and the colours of the balls are the "observed signals".
- A Markov chain governs the successive choices of the urns, i.e., the transition matrix of the Markov chain dictates the choice of the urns.

## Markov & Pushkin (why scientists should study poems)

- Suppose you are given a body of text and asked to guess whether the letter at a randomly selected position is a vowel or a constant. Since consonants occur more frequently than vowels, your best bet is to always guess consonant. Suppose we decide to be a little more helpful and tell you whether the letter preceding the one you chose is a vowel or consonant. Is there now a better strategy you can follow?

- In 1913, A.A. Markov was trying to answer the above problem analysed twenty thousand letters from Pushkin's poem Eugene Origin. He found that 43% letters were vowels and 57%, consonants. So in the first problem, one should always guess "consonant" and can hope to be correct 57% of the time.

## Markov & Pushkin - 2

- However, a vowel was followed by consonant 87% of the time. A consonant was followed by a vowel 66% of the time. Hence, guessing the opposite of the preceding letter would be a better strategy in the second case. Clearly, knowledge of the preceding letter is helpful.

- The real insight came when Markov took the analysis a step further. Markov investigated whether knowledge about the preceding two letters confers any additional advantage. He found that there was no significant advantage to knowing the additional preceding letter. This leads to the central idea of a Markov chain - while the successive outcomes are not independent, only the most recent outcome is of use in making a prediction about thenext outcome.

## What is a Gaussian Mixture Model (GMM)?

- Example on how to fit a mixture of two Gaussian distributions to an observed set of points.
- The algorithm used is called EM Algorithm (Expectation Maximization) and it is essentially an application of Bayes' Theorem.
- Suppose you have the following 10 points and we want to fit a mixture of two Gaussian distributions, $N_1 (\mu_1 , \sigma_1)$ and $N_2 (\mu_2 , \sigma_2)$ along with the probability $p_1$ that a point comes from $N_1$ (and probability $p_2 = 1 - p_1$ that the point is drawn from $N_2$ ).

Data = 8.4, 7.6, 4.2, 2.6, 5.1, 4.0, 7.8, 3.0, 4.8, 5.8

## Recognition of Digits: Mixture Models

Example based on ***Christopher M. Bishop, Pattern Recognition and Machine Learning.***

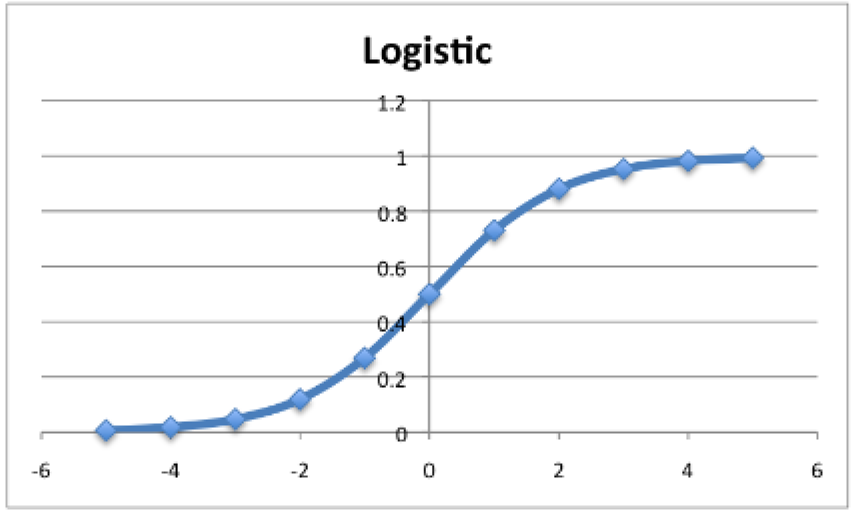
- Imagine that we are given an $N \times N$ black-and-white image that is known to be a scan of a hand-written digit between 0 and 9, but we don't know which digit is written.
- We can create a mixture model  $K=10$ different components, where each component is a vector of  N^{2} of Bernoulli distributions (one per pixel).
- Such a model can be trained with the expectation-maximization algorithm on an unlabeled set of hand-written digits, and will effectively cluster the images according to the digit being written.
- The same model could then be used to recognize the digit of another image simply by holding the parameters constant, computing the probability of the new image for each possible digit (a trivial calculation), and returning the digit that generated the highest probability.

## Many Ways to Recognize a Face

- ❑ Logistic Regression
- ❑ PCA
- ❑ K-nearest neighbours
- ❑ MLP NN
- ❑ CNN

## What is Logistic Regression?

- Logistic Regression is one of the most basic and important statistical techniques used in Machine Learning.
- It is closely related to neural networks, because the **logistic** function, also called the **Sigmoid** function, is heavily used as the ***activation function*** in Neural Networks. The logistic function is S-shaped and is defined as



$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

## Logistic Regression-contd.

- Logistic function is very useful in modelling probabilities since its range is [0,1]
- Hence it is very useful in Binary Classification as it predicts *p*, the probability of a data point being in class 1. (And therefore also predicts 1, which is *1-p*)

If we define $t = \boldsymbol{\beta}_0 + \boldsymbol{\beta}_1 X$

Then the definition of logisitic function becomes $$\frac{1}{1+e^{-(\beta_0+\beta_1 x)}}$$

## Softmax Function

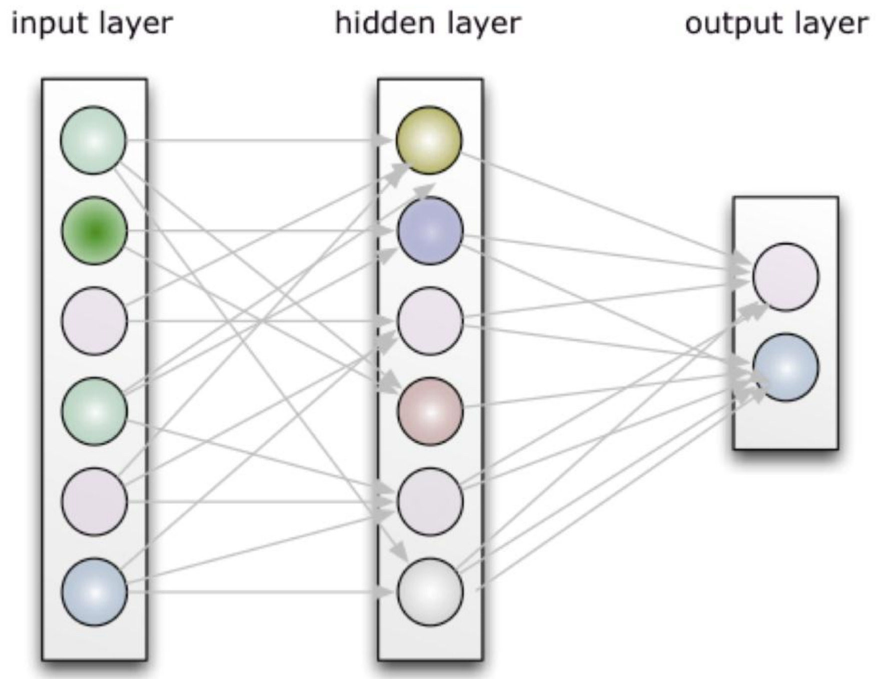
- The **Softmax** function is a multinomial generalization of the Logistic function. It is useful when dealing with multi-category classification instead of binary classification. For example, classifying handwritten digits into one of ten classes.

"squashes" a *K*-dimensional vector ***z*** of arbitrary real values to a *K*-dimensional vector ***σ(z)*** of real values in the range (0, 1) that add up to 1.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

## What is a Neural Network (NN)?

- Traditional NN uses a feedforward network structure and usually has only one layer. Compared with Deep Neural Network, its structure is simpler and the training is less computationally intensive.
- NN is useful when we have abundance of labeled data but without the knowledge of the underlying mapping function that generates the output. It also shines when data sets are noisy or containing missing variables.
- To train a NN, we first acquire labeled inputs (as high-dimensional vector) and outputs. We then design the structure of the network, such as number of layers and number of neurons in each layer. The formal training process starts with random initialization and feedforward and backpropgation.

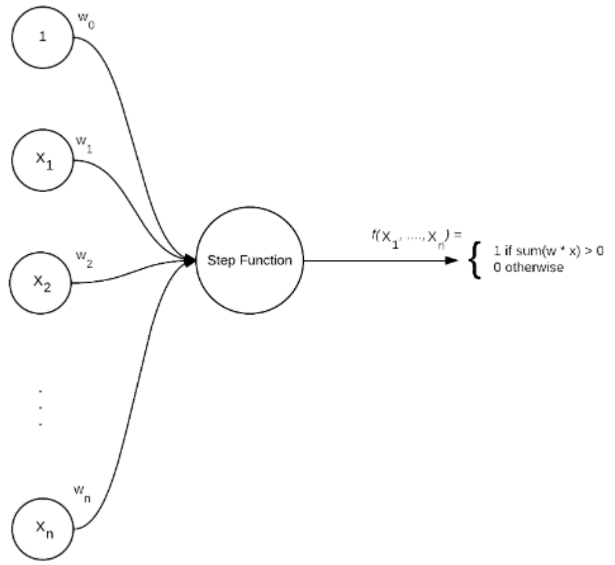


## What is a Neural Network *REALLY* ?

- There has been lot of hype surrounding Neural Networks, including exaggerated comparisons to human brain. This led to very high expectations which were not met, leading to disappointment with Neural Networks and AI for decades.
- Think of a Neural Network simply as a two stage, non-linear statistical model used for classification or regression.

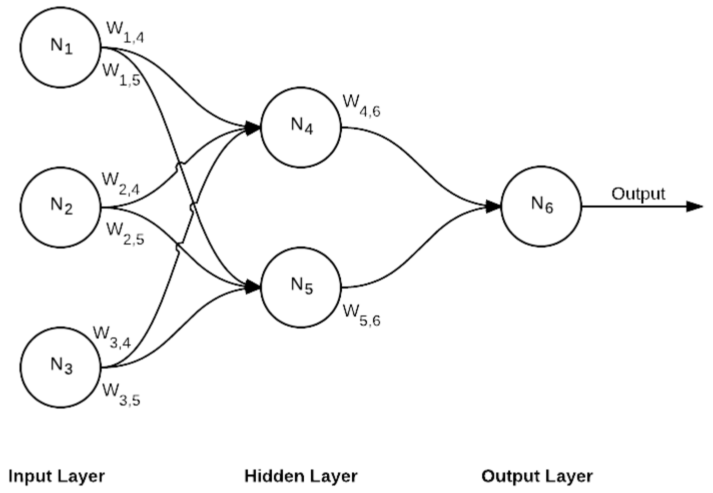
## The Simplest Neural Network: Perceptron

- The simplest neural network is described as a single hidden layer back-propagation network. There are *N* input nodes, one for each entry in the input feature vector, followed by only ***one layer*** in the network with just a ***single node*** in that layer. There exist connections and their corresponding weights, from the input 's to the single output node in the network. This node then takes the weighted sum of inputs and applies a *step function* to determine the output class label. The Perceptron outputs either a *0* or a *1* — *0* for class #1 and *1* for class #2; thus, in its original form, the Perceptron is simply a binary, two-class classifier. Perceptron is a *linear classifier*; it cannot solve non-linear problems such as XOR.



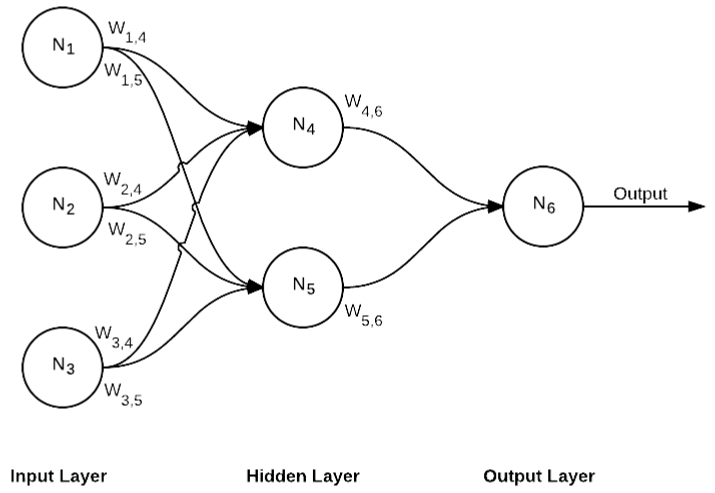
## Multilayer Feed Forward Neural Network

- In order to obtain non-linear separability, we can use ***multi-layer*** feedforward networks with ***non-linear activation functions***.
- A multi-layer feedforward network consists of multiple layers: 1 input layer, *N* hidden layers, and 1 output layer; in our figure we have one input layer, a hidden layer and an output



## Neural Network and Softmax Regression

- In order to obtain non-linear separability, we can use ***multi-layer*** feedforward networks with ***non-linear activation functions***.
- The output of the hidden layer is fed to a Softmax function, as in a multinomial logistic regression function.



## Gradient Descent and Neural Network Training

- The bottom of the bowl is the minimum loss, or  the best set of model parameters or in case of a neural network, the “weights
- The objective is to reach the bottom, taking  as few steps as possible…

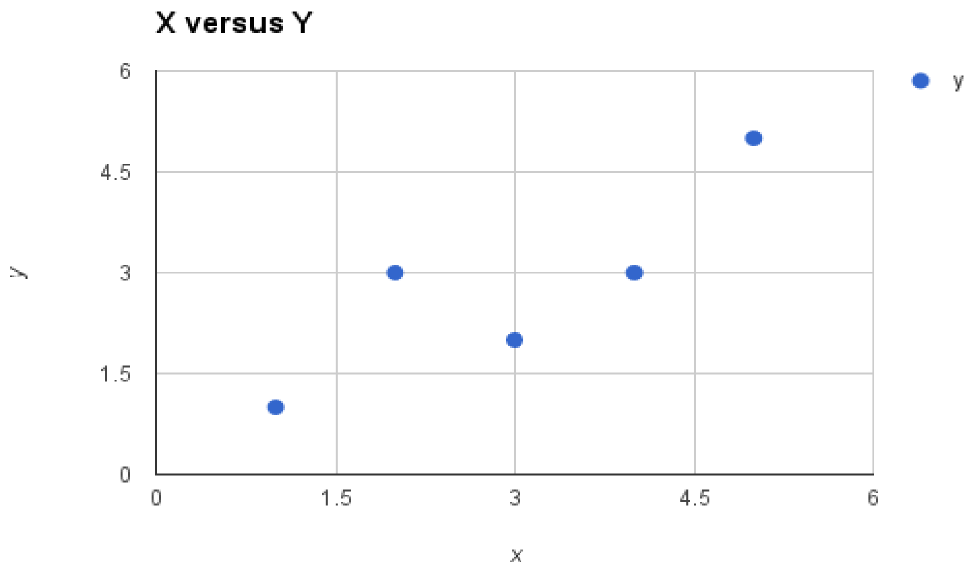
## Stochastic Gradient Descent

- In the original Gradient Descent scheme, all training examples are shown, the error calculated, and then we find gradients with respect to weights.
- This is called “batch” gradient descent but is very slow.
- Another approach is to show one example, compute error and gradients, update weights… this faster but noisy
- Stochastic Gradient – randomly choose a subset of the training examples (called “mini-batch”) for each epoch. This is better than choosing one example each epoch

## Illustrating Gradient Descent for Training a Simple Model

- We can work through a simple example of how gradient descent can be used to train a linear regression model.
- Although the use Gradient Descent is not necessary here, it serves as an illustration
- Here are the data

| x | y |
|---|---|
| 1 | 1 |
| 2 | 3 |
| 4 | 3 |
| 3 | 2 |
| 5 | 5 |



## Illustrating Gradient Descent -2

- In a simple linear regression, the model is $\boldsymbol{y= b_0+ b_1x}$
- Initialise both model parameters to 0.
- The model becomes  y = 0.0 + 0.0 * x
- Calculate the predicted value for y using our starting point coefficients for the first training instance: i=1, data(i) is  x=1, y=1

Prediction, p(i) = $\boldsymbol{b_0+ b_1x(}1)=0$

Error(i) = p(i)-  y(i) = $\boldsymbol{b_0+ b_1x(}1) – 1 = 0-1 = -1$

The loss function is L = $\frac{1}{2} *$(prediction - target) $^2$

Partial derivative of Error w.r.t. $b_0$ is 1 and w.r.t. $b_1$, it is x

## Illustrating Gradient Descent -3

- Keeping in mind what we just derived, and deciding to use a parameter α to control how big a step we want to take in the direction of the gradient, we see that

  $b_0(t+1) = b_0(t) - \alpha * error$

  and

  $b_1(t+1) = b_1(t) - \alpha * error * x$

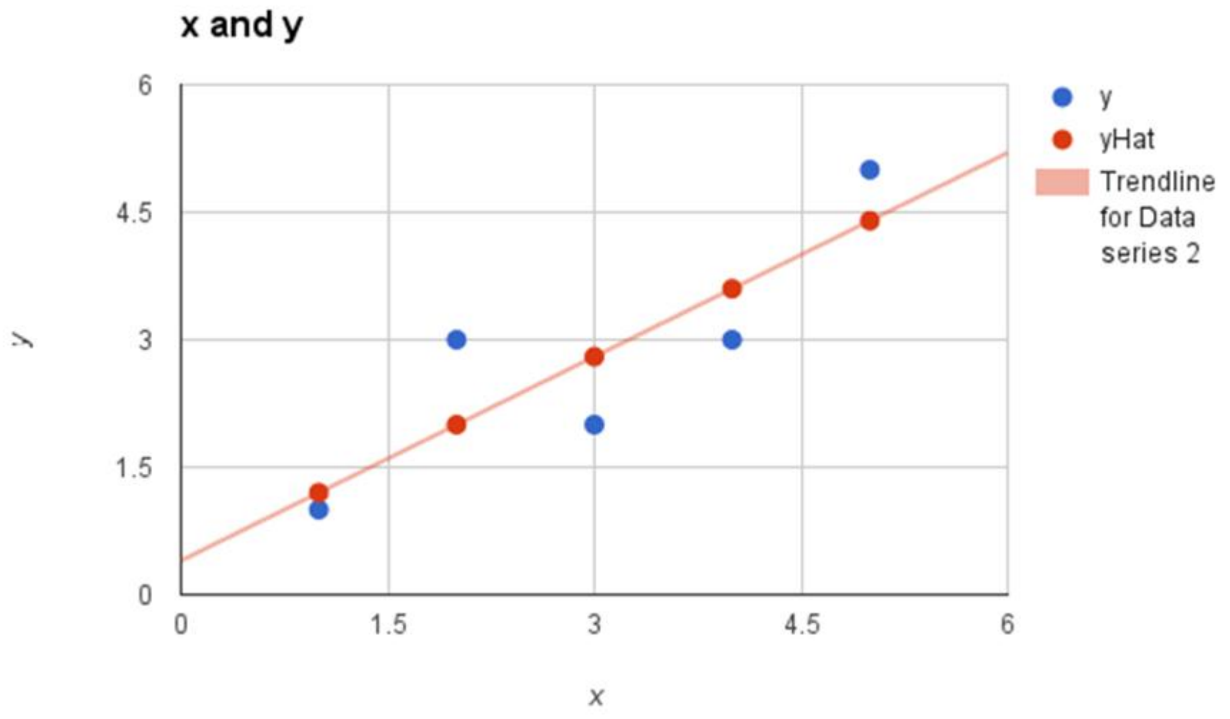
  $b_0$(t+1) = 0.0 – 0.01 * -1.0 = 0.01

  $b_1$ (t+1) = 0.0 – 0.01 * -1 * 1=0.01

- We finished one epoch. After 20 iterations, we have $b_0$ as 0.23 and $b_1$ as 0.79 (actual values are 0.4 and 0.8)

## Example of Gradient Descent in Action -4

When we use these  parameters to make pass the values of x through the model and output the predicted values, here is what we get.

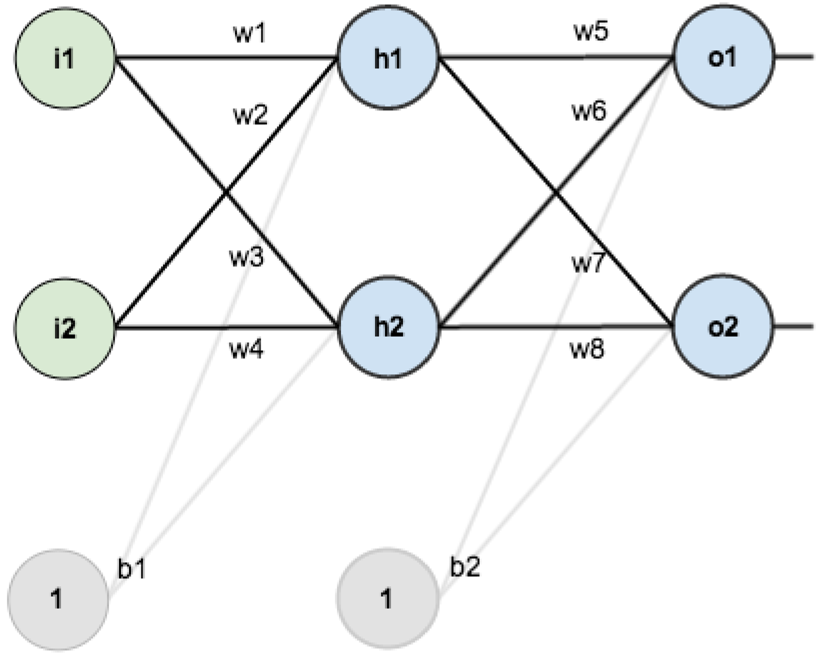


## The Backpropagation Algorithm Illustrated

It is a common method for training a neural network

**Overview:** Let  use a neural network with two inputs, two hidden neurons, two output neurons. Additionally, the hidden and output neurons will include a bias.

Here’s the basic structure:



## The Backpropagation Algorithm Illustrated-2

➢In order to have some numbers to work with, here are the initial weights, the biases, and training inputs/outputs:

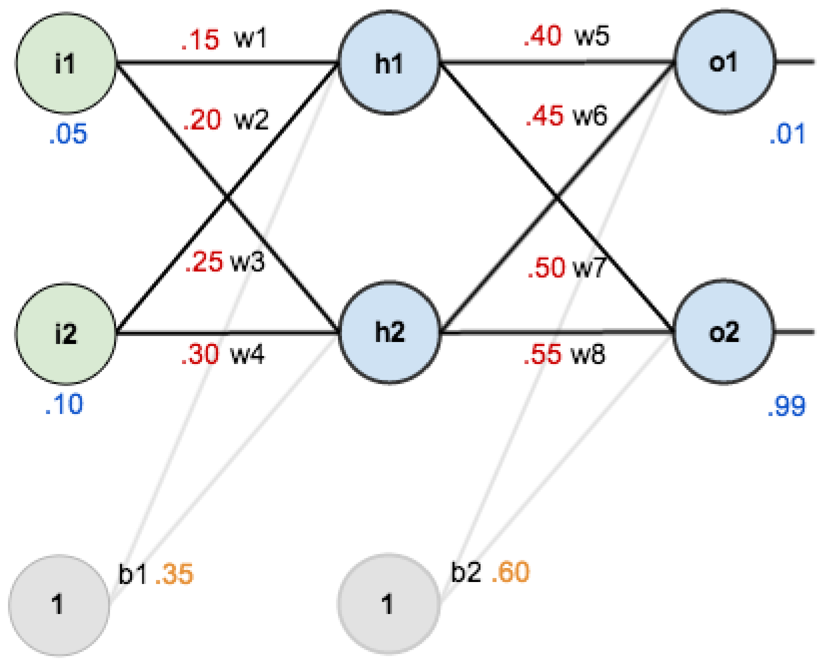
The goal of backpropagation is to optimize the weights so that the neural network can learn how to correctly map arbitrary inputs to outputs.

For the rest of this presentation we're going to work with a single training set:

- Given inputs 0.05 and 0.10
- We want the neural network to output 0.01 and 0.99.

<u>The Forward Pass</u>

- To do this we'll feed those inputs forward though the network.
- We figure out the *total net input* to each hidden layer neuron, *squash* the total net input using an *activation function* (here we use the *logistic function*),
- Repeat the process with the output layer neurons.



## The Backpropagation Algorithm-The Forward Pass (Equation)

- Calculating the total net input for $h_1$

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$
$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

- We then squash it using the logistic function to get the output of $h_1$

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

- Carrying out the same process for $h_2$

$$out_{h2} = 0.596884378$$

- We repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs.

- Here's the output for $o_1$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$
$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$
$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

- And carrying out the same process for $o_2$ we get

$$out_{o2} = 0.772928465$$

## Backpropagation: Calculating the Total Error

- We can now calculate the error for each output neuron using the squared error function and sum them to get the total error:
  $E_{total} = \sum \frac{1}{2}(target - output)^2$
- The $\frac{1}{2}$ is included so that exponent is cancelled when we differentiate later on. The result is eventually multiplied by a learning rate anyway so it doesn't matter that we introduce a constant here.
- For example, the target output for $o_1$ is 0.01 but the neural network output 0.75136507, therefore its error is:
  $E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$
- Repeating this process for $o_2$ (remembering that the target is 0.99) we get:
  $E_{o2} = 0.023560026$
- The total error for the neural network is the sum of these errors:
  $E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$

## The Backpropagation Algorithm :The Backwards Pass

- Goal with backpropagation is to update each of the weights in the network so that they cause the actual output to be closer the target output,
- Thereby minimizing the error for each output neuron and the network as a whole.

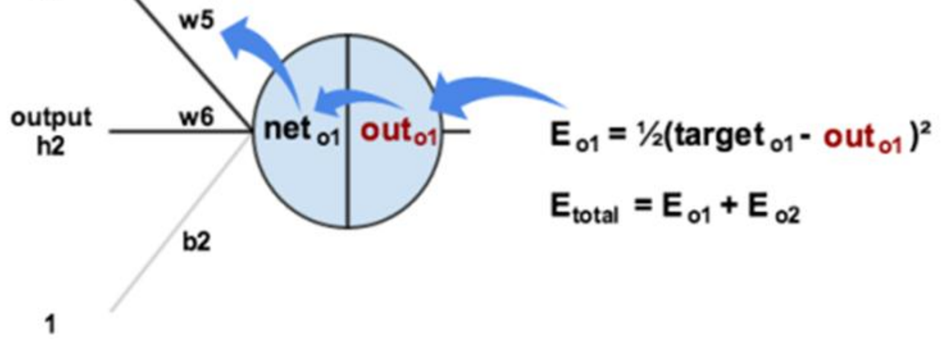
### Output Layer

- Consider $w_5$. How much a change in $w_5$ affects the total error, aka $\frac{\partial E_{total}}{\partial w_5}$.
- $\frac{\partial E_{total}}{\partial w_5}$ is read as "the partial derivative of $E_{total}$ with respect to $w_5$". You can also say "the gradient with respect to $w_5$".

By applying the chain rule we know that:

➢ Visually, here's what we're doing:

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



## The Backwards Pass-2

Let us examine how the total error change with respect to the output…

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

$-(target - out)$ is sometimes expressed as $out - target$

When we take the partial derivative of the total error with respect to $out_{o1}$, the quantity $\frac{1}{2}(target_{o2} - out_{o2})^2$ becomes zero because $out_{o1}$ does not affect it which means we’re taking the derivative of a constant which is zero.

Next, how much does the output of $o_1$ change with respect to its total net input?

## The Backwards Pass -3

The partial derivative of the logistic function is the output multiplied by 1 minus the output:

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

Finally, how much does the total net input of $o1$ change with respect to $w_5$?

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

# The Backwards Pass -4

Putting it all together:

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

You'll often see this calculation combined in the form of the delta rule:

$$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) * out_{h1}$$

Alternatively, we have $\frac{\partial E_{total}}{\partial out_{o1}}$ and $\frac{\partial out_{o1}}{\partial net_{o1}}$ which can be written as $\frac{\partial E_{total}}{\partial net_{o1}}$, aka $\delta_{o1}$ (the Greek letter delta) aka the *node delta*. We can use this to rewrite the calculation above:

$$\delta_{o1} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1})$$

Therefore:

$$\frac{\partial E_{total}}{\partial w_5} = \delta_{o1} out_{h1}$$

Some sources extract the negative sign from $\delta$ so it would be written as:

$$\frac{\partial E_{total}}{\partial w_5} = -\delta_{o1} out_{h1}$$

## Backward Pass -5

To decrease the error, we then subtract this value from the current weight (optionally multiplied by some learning rate, eta, which we'll set to 0.5):

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

Some sources use $\alpha$ (alpha) to represent the learning rate, others use $\eta$ (eta), and others even use $\epsilon$ (epsilon).

We can repeat this process to get the new weights $w_6$, $w_7$, and $w_8$:

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$

We perform the actual updates in the neural network *after* we have the new weights leading into the hidden layer neurons (ie, we use the original weights, not the updated weights, when we continue the backpropagation algorithm below).

## Backward Pass - 6

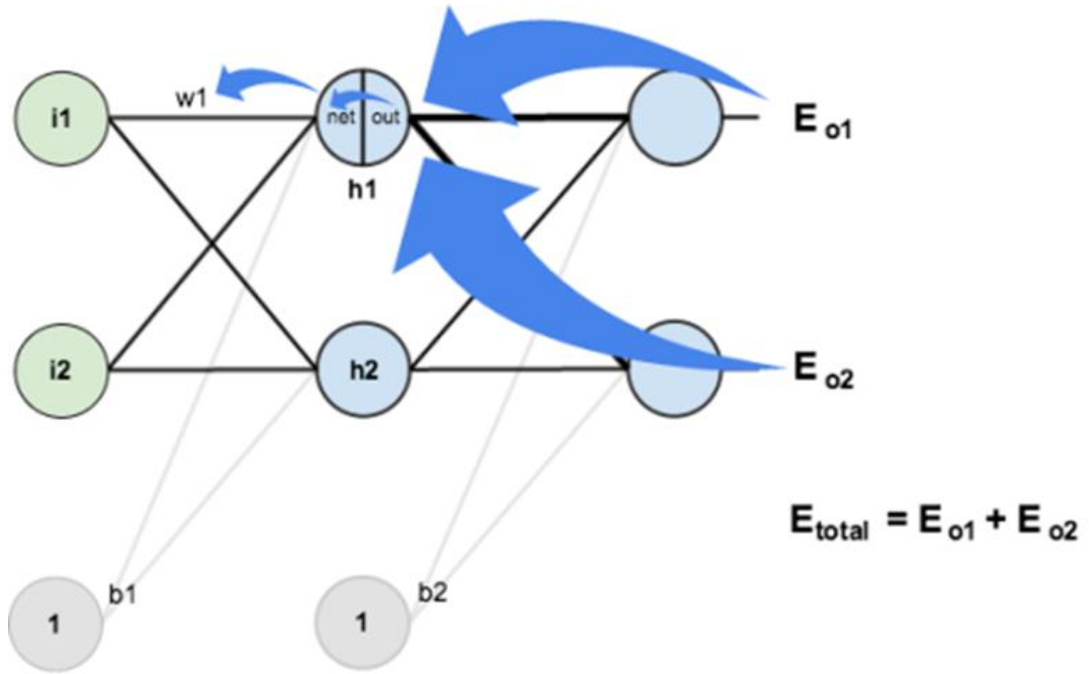
**Hidden Layer**

Next, we'll continue the backwards pass by calculating new values for $w_1$, $w_2$, $w_3$, and $w_4$.

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\downarrow$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



## Backward Pass -7

We're going to use a similar process as we did for the output layer, but slightly different to account for the fact that the output of each hidden layer neuron contributes to the output (and therefore error) of multiple output neurons. We know that $out_{h1}$ affects both $out_{o1}$ and $out_{o2}$ therefore the $\frac{\partial E_{total}}{\partial out_{h1}}$ needs to take into consideration its effect on the both output neurons:

Starting with $\frac{\partial E_{o1}}{\partial out_{h1}}$:

We can calculate $\frac{\partial E_{o1}}{\partial net_{o1}}$ using values we calculated earlier:

And $\frac{\partial net_{o1}}{\partial out_{h1}}$ is equal to $w_5$:

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$
$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$

Plugging them in:

Following the same process for $\frac{\partial E_{o2}}{\partial out_{o1}}$, we get:

$$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$$

## Backward Pass -8

Therefore:

Now that we have $\frac{\partial E_{total}}{\partial out_{h1}}$ , we need to figure out $\frac{\partial out_{h1}}{\partial net_{h1}}$ and then $\frac{\partial net_{h1}}{\partial w}$ for each weight:

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}}$$

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$$

We calculate the partial derivative of the total net input to $h_1$ with respect to $w_1$ the same as we did for the output neuron:

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

Putting it all together:

$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

## Backward Pass - Conclusion

We can now update $w_1$:

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

Repeating this for $w_2$, $w_3$, and $w_4$

$$w_2^+ = 0.19956143$$

$$w_3^+ = 0.24975114$$

$$w_4^+ = 0.29950229$$

Finally, we've updated all of our weights!

When we fed forward the 0.05 and 0.1 inputs originally, the error on the network was 0.298371109. After this first round of backpropagation, the total error is now down to 0.291027924. It might not seem like much, but after repeating this process 10,000 times, for example, the error plummets to 0.000035085. At this point, when we feed forward 0.05 and 0.1, the two outputs neurons generate 0.015912196 (vs 0.01 target) and 0.984065734 (vs 0.99 target).

## The Backpropagation Algorithm-summary

A brief  representation of the backpropagation algorithm:

```
1: Initialize all weights w with random values
2: Until convergence:
    2.1: For each feature vector x and expected output y:
        2.1.1: Pass x through the network and obtain the output.
        2.1.2: Calculate the error of the output nodes.
        2.1.3: Calculate the error at hidden nodes.
        2.1.4: Use the errors to compute  Delta w_{i,j}
        2.1.5: Accumulate the errors for each Delta w_{i,j}
    2.2: Perform learning by adding the accumulated Delta w_{i,j} to w_{i, j}
```

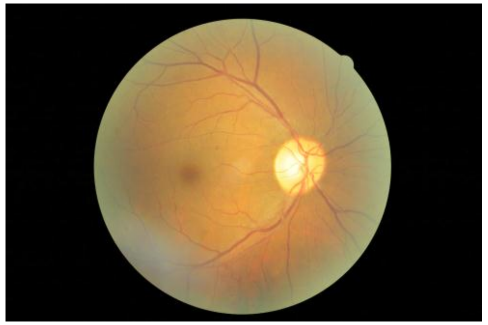
## What is Deep Learning?

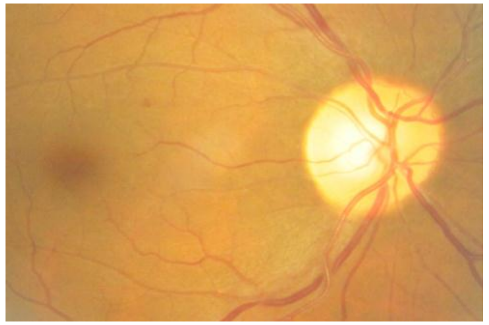
- What is Deep Learning
  - It is a paradigm in machine learning in which researchers train computer algorithms to spot meaningful patterns by showing them lots of data, rather than explicitly program the rules
  - Neural Nets that mimic the human brain
  - Deep architecture to enable predictions or classifications with unseen accuracy
  - It enables universal classification: the same model can classify languages, images, videos, audios
- Advantages of Deep Learning
  - Scalable: can scale to billions of parameters to learn complex concepts
  - Fast: Model training is fast and a trained model makes online prediction for unseen inputs
  - Unified: it brings a unified approach to data-driven knowledge discovery

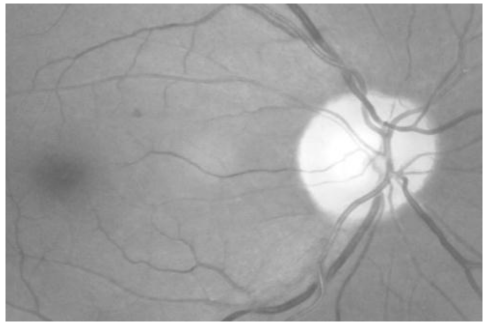
## What is a Convolutional Neural Network?(CNN)

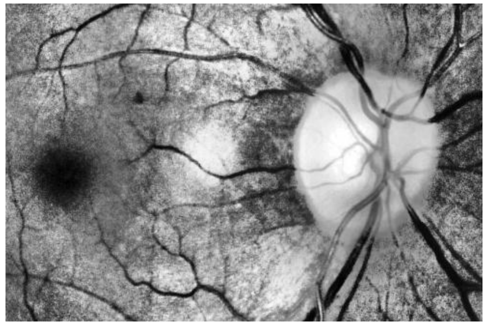
- ***Image classification*** is the task of applying computer vision and machine learning algorithms to extract meaning from an image.
- This could be as simple as ***assigning a label to an image*** from a ***pre-defined set of categories***
- or it could be as advanced as interpreting the contents of an image and returning a human-readable sentence.
- A class **of Deep Neural Networks called Convolutional Neural Networks** are state of the art in Image Recognition and Classification.
- They attracted attention by winning the 2012 ImageNet competition - the computer vision Olympics

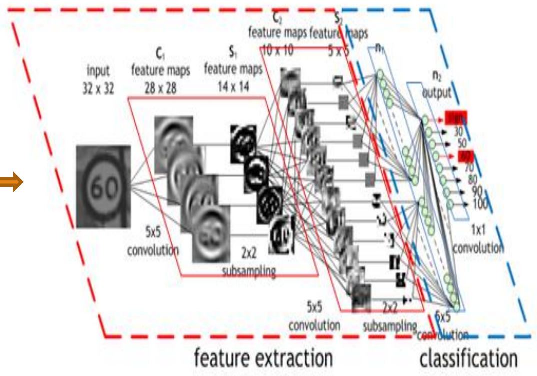
## How ANKAM use Neural Net for Diabetic Retinopathy diagnosis


Original Image


Cropped Image


Extracted Green Channel


After Adaptive Equalization



Run through Neural Net

Diagnosis

**Load Retinal Scan**
DICOM based storage and exchange of medical image-data

**Image Preprocessing**
Gaussian Filters, Adaptive Contrast Equalization, etc.

**Unsupervised feature extraction**
Using CNNs trained on large amounts of data

**Supervised learning**
Logistic regression, Decision Forest, etc.

**Output decision**
Give rating based on severity

## Feature Extraction and Dimensionality Reduction

- Dimensionality reduction is an important pre-processing step in various machine learning techniques, which transforms the data from high dimensional space into a lower dimensional feature space. This step aims at removing redundant and irrelevant features, which results in improved training efficiency, result comprehensibility and prediction accuracy.
- Commonly used Dimensionality Reduction techniques include but not limited to
  - Principal Component Analysis (PCA) and Kernel PCA
  - Linear Discriminant Analysis (LDA)
  - Manifold Learning (Isomap, locally linear embedding (LLE), Hessian LLE, etc.)
  - The **Autoencoder**, which forms the building blocks or simpler part of several Deep Learning systems

## Feature Engineering

Feature Engineering is manually designing what the input x's should be.

- In the case of visual object recognition problems, when we something other than feeding raw pixels into the classifier, we engineering features.
- Often we get much better performance when we use features relevant to the vision problem  This is called feature engineering.
- For example, for vessel detection or extraction, there might be some special set of steps to be carried which will be different from face detection…
- One important example is the Viola-Jones Face Detection framework

## Face Detection: Viola-Jones and Haar Cascades/ HOG

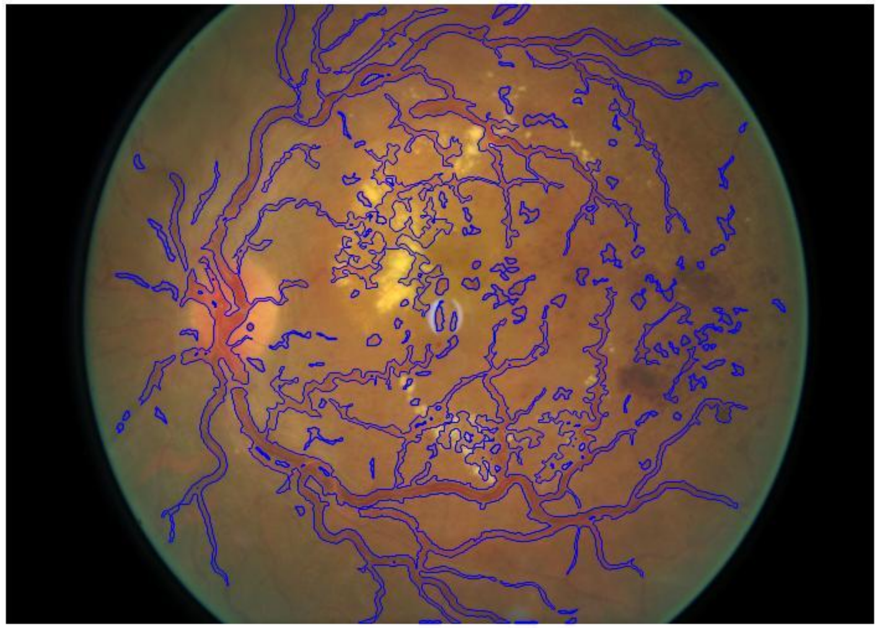
Haar Features – All human faces share some similar properties. These regularities may be matched using **Haar Features**.

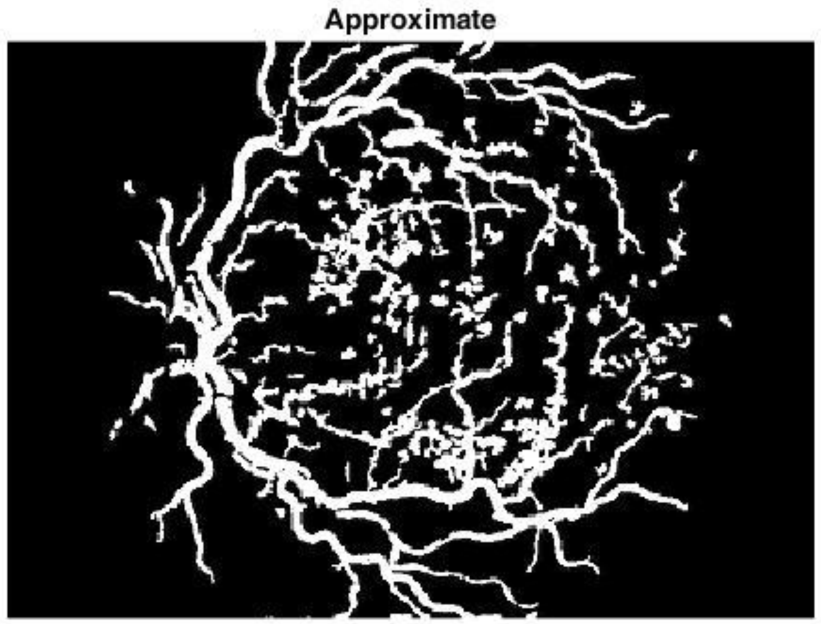
- ❑A few properties common to human faces:
- ❑The eye region is darker than the upper-cheeks.
- ❑The nose bridge region is brighter than the eyes.
- ❑Composition of properties forming matchable facial features:
- ❑Location and size: eyes, mouth, bridge of nose
- ❑Value: oriented gradients of pixel intensities

Framework has 4 steps -Haar Feature Selection,Creating an Integral Image,Adaboost Training,Cascading Classifiers

HOG – Histogram of Oriented Gradients

## Our System for Retinal Vessel Extraction





## Before Deep Learning, Every problem had different approach

- Bag of Words for Document Classification, borrowed by Computer vision as Visual BOW
- HMM for Automated Speech Recognition
- In image processing there were 100's of different schemes:

I. Corner Detectors (Shi & Tomasi, Harris),

II. Edge Detectors (Canny, Sobel) ,

III. Circular Shapes (Hough Transform)

IV. Various filters (e.g. COSFIRE for vessel detection)

V. Feature Detectors like SIFT/SURF

VI. BLOB Detectors(LoG- Laplacian of Gaussians; DoH- Determinant of Hessian;DoG – Difference of Gaussians

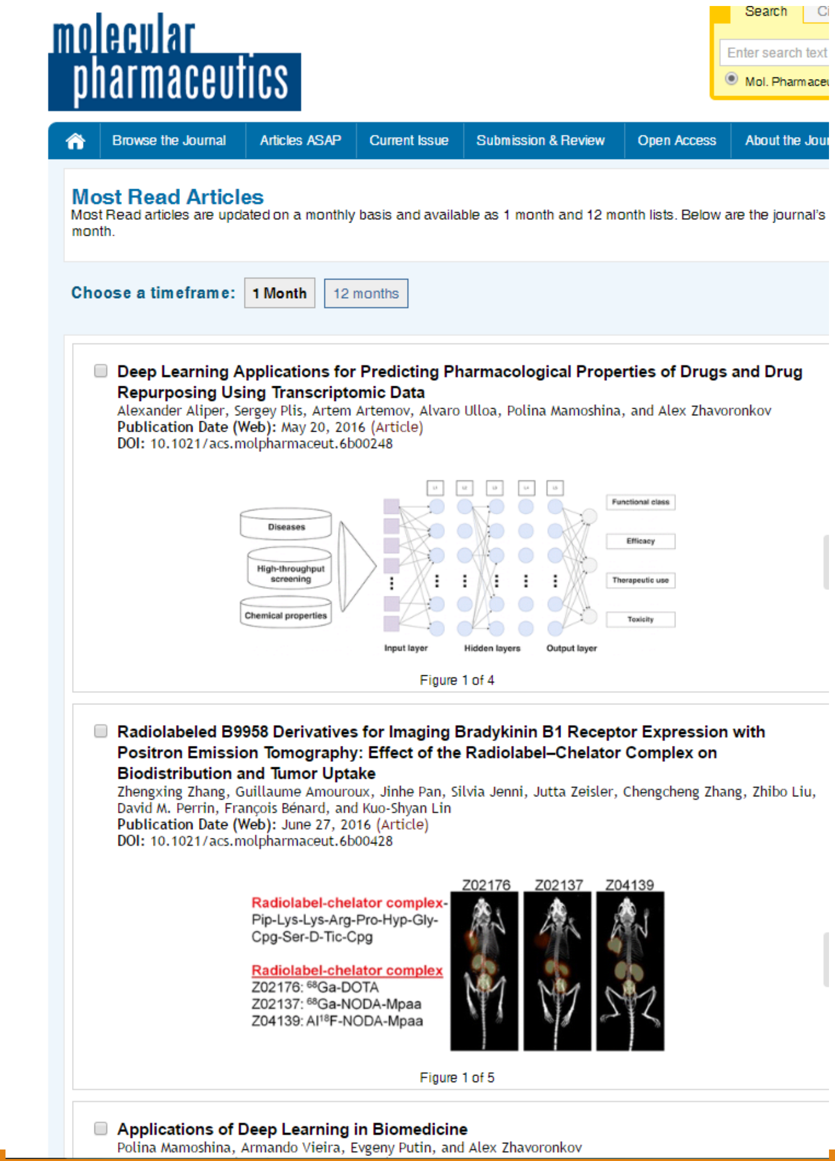
VII. Viola Jones for Face Detection, then HOG

## The Unification: Deep Learning

- DNN architectures generate compositional models, where extra layers enable composition of features from lower layers, giving a huge learning capacity and thus the potential of modeling complex patterns of speech data.
- One fundamental principle of deep learning is to do away with hand-crafted feature engineering and to use raw features.
- Many aspects of speech recognition have been taken over by a deep learning method called Long short term memory (LSTM), a recurrent neural network- LSTM RNN's can learn "Very Deep Learning" tasks that require memories of events that happened thousands of discrete time steps ago, which is very important for speech recognition

# “Most Read” articles in Molecular Pharmaceutics

1. **[Deep Learning Applications for Predicting Pharmacological Properties of Drugs and Drug Repurposing Using Transcriptomic Data]()**
2. Radiolabeled B9958 Derivatives for Imaging Bradykinin B1 Receptor Expression with Positron Emission Tomography: Effect of the Radiolabel–Chelator Complex on Biodistribution and Tumor Uptake
3. **[Applications of Deep Learning in Biomedicine]()**



**Most Read Articles**

Most Read articles are updated on a monthly basis and available as 1 month and 12 month lists. Below are the journal's month.

Choose a timeframe: **1 Month** 12 months

**Deep Learning Applications for Predicting Pharmacological Properties of Drugs and Drug Repurposing Using Transcriptomic Data**
Alexander Aliper, Sergey Plis, Artem Artemov, Alvaro Ulloa, Polina Mamoshina, and Alex Zhavoronkov
**Publication Date (Web):** May 20, 2016 (Article)
**DOI:** 10.1021/acs.molpharmaceut.6b00248

Figure 1 of 4

**Radiolabeled B9958 Derivatives for Imaging Bradykinin B1 Receptor Expression with Positron Emission Tomography: Effect of the Radiolabel–Chelator Complex on Biodistribution and Tumor Uptake**
Zhengxing Zhang, Guillaume Amouroux, Jinhe Pan, Silvia Jenni, Jutta Zeisler, Chengcheng Zhang, Zhibo Liu, David M. Perrin, François Bénard, and Kuo-Shyan Lin
**Publication Date (Web):** June 27, 2016 (Article)
**DOI:** 10.1021/acs.molpharmaceut.6b00428

Radiolabel-chelator complex-Pip-Lys-Lys-Arg-Pro-Hyp-Gly-Cpg-Ser-D-Tic-Cpg
Radiolabel-chelator complex
Z02176: ⁶⁸Ga-DOTA
Z02137: ⁶⁸Ga-NODA-Mpaa
Z04139: Al¹⁸F-NODA-Mpaa
Z02176
Z02137
Z04139

Figure 1 of 5

**Applications of Deep Learning in Biomedicine**
Polina Mamoshina, Armando Vieira, Evgeny Putin, and Alex Zhavoronkov

# Drug Repositioning for Ophthalmology Example

## LUMIGANTM (0.03%) AND LATISSE TM(0.03%), BOTH CONTAINS BIMATOPROST 0.3MG/ML

**LumiganTM (0.03%)**

- Indicated for reduction of elevated intraocular pressure in patients with open angle glaucoma or ocular hypertension
- The recommended dosage is one drop in the affected eye(s) once daily in the evening

**Latisse TM(0.03%)**

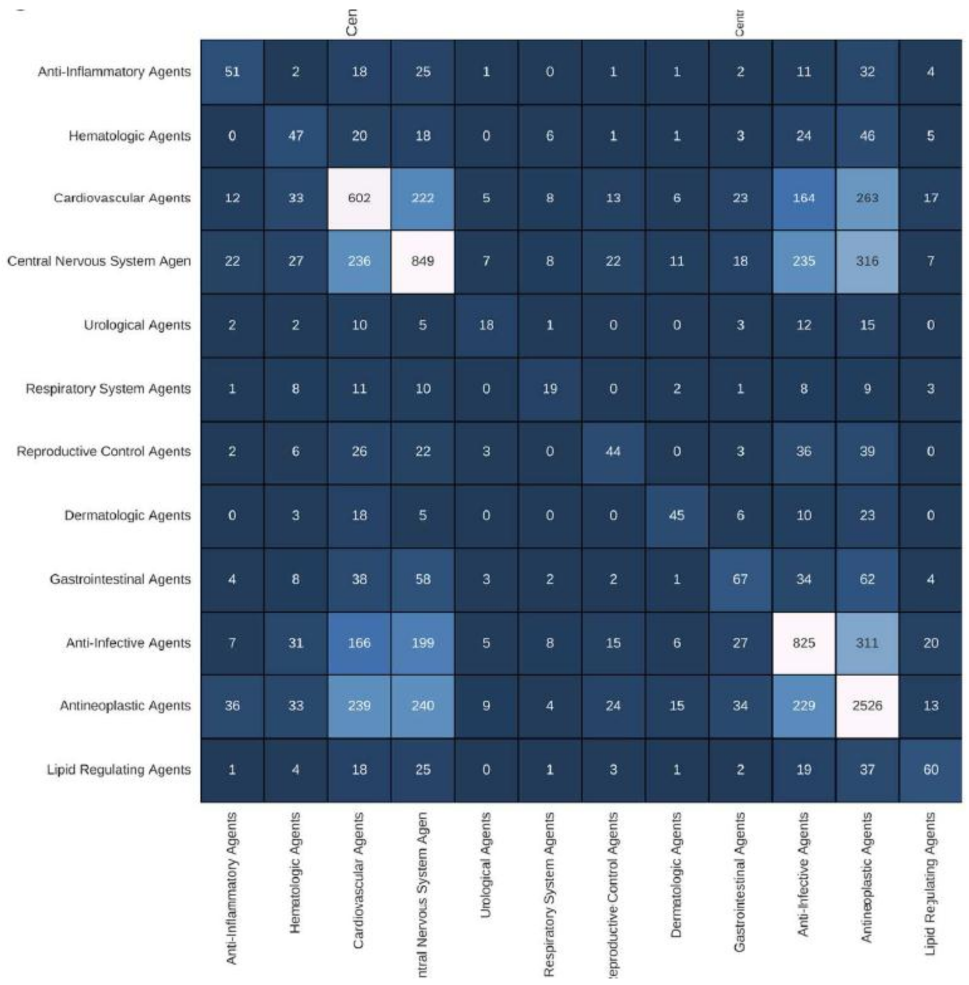
- Indicated to treat hypotrichosis of the eyelashes by increasing their growth including length , thickness and darkness
- Once nightly, place one drop on sterile applicator supplied with the package and apply evenly along the skin of upper eyelid margin at the base of eyelashes

## Drug Repositioning and Deep Learning

The paper by Aliper [8] classified various drugs to therapeutic categories solely based on their transcriptional profiles.

- It is possible to interpret the classification results from different angles.
- For instance, as it was shown, in confusion matrices (Fig. 3) the “misclassified” samples for a certain drug might in fact be an indication of its potential for novel use, or repurposing, in these exact “incorrectly” assigned conditions.
- Misclassification, therefore, may lead to unexpected new discoveries. This approach opens a great avenue for application of DL in the drug repurposing field.

# Drug Repositioning-Confusion Matrix



| | Anti-Inflammatory Agents | Hematologic Agents | Cardiovascular Agents | ntral Nervous System Agen | Urological Agents | Respiratory System Agents | eproductive Control Agents | Dermatologic Agents | Gastrointestinal Agents | Anti-Infective Agents | Antineoplastic Agents | Lipid Regulating Agents |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Anti-Inflammatory Agents | 51 | 2 | 18 | 25 | 1 | 0 | 1 | 1 | 2 | 11 | 32 | 4 |
| Hematologic Agents | 0 | 47 | 20 | 18 | 0 | 6 | 1 | 1 | 3 | 24 | 46 | 5 |
| Cardiovascular Agents | 12 | 33 | 602 | 222 | 5 | 8 | 13 | 6 | 23 | 164 | 263 | 17 |
| Central Nervous System Agen | 22 | 27 | 236 | 849 | 7 | 8 | 22 | 11 | 18 | 235 | 316 | 7 |
| Urological Agents | 2 | 2 | 10 | 5 | 18 | 1 | 0 | 0 | 3 | 12 | 15 | 0 |
| Respiratory System Agents | 1 | 8 | 11 | 10 | 0 | 19 | 0 | 2 | 1 | 8 | 9 | 3 |
| Reproductive Control Agents | 2 | 6 | 26 | 22 | 3 | 0 | 44 | 0 | 3 | 36 | 39 | 0 |
| Dermatologic Agents | 0 | 3 | 18 | 5 | 0 | 0 | 0 | 45 | 6 | 10 | 23 | 0 |
| Gastrointestinal Agents | 4 | 8 | 38 | 58 | 3 | 2 | 2 | 1 | 67 | 34 | 62 | 4 |
| Anti-Infective Agents | 7 | 31 | 166 | 199 | 5 | 8 | 15 | 6 | 27 | 825 | 311 | 20 |
| Antineoplastic Agents | 36 | 33 | 239 | 240 | 9 | 4 | 24 | 15 | 34 | 229 | 2526 | 13 |
| Lipid Regulating Agents | 1 | 4 | 18 | 25 | 0 | 1 | 3 | 1 | 2 | 19 | 37 | 60 |

## Drug Repurposing: More from Aliper paper

***From the Abstract:***

"We demonstrate how deep neural networks (DNN) trained on large transcriptional response data sets can classify various drugs to therapeutic categories solely based on their transcriptional profiles. We used the perturbation samples of ***678 drugs across 3 cell lines*** (A549, MCF-7 and PC-3 from the LINCS project ) and linked those to ***12 therapeutic use categories*** derived from MeSH. To train the DNN, we utilized both gene level transcriptomic data and transcriptomic data processed using a **pathway activation scoring** algorithm, for a pooled dataset of samples perturbed with different concentrations of the drug for 6 and 24 hours. In both gene and pathway level classification, DNN achieved high classification accuracy…"

## The Merck Molecular Activity Challenge

- When developing new medicines it is important to identify molecules that are highly active toward their intended targets but not toward other targets that might cause side effects.
- The objective of this competition was to identify the best techniques for predicting biological activities of different molecules, given numerical descriptors generated from their chemical structures.
- The challenge was based on 15 molecular activity data sets, each for a biologically relevant target. Each row in the dataset corresponds to a molecule and contains descriptors derived from that molecule's chemical structure.

## Merck Challenge, Deep Learning and Virtual Screening

- The first prize in Merck Kaggle challenge went to a team of academics who used Deep Neural Networks trained by GPUs.
- After the Merck Kaggle challenge was won by Deep Learning, there has been a heightened interest in applying Deep learning  to drug design.
- Unterthiner et al [3] evaluated several techniques including SVM and Deep Learning, on a large database – the CHeMBl database, which has 13 million compound descriptors, 1.3 million compounds, and 5,000 drug targets.
- In terms of accuracy, as measured by AUC (area under the curve), Deep Learning and SVM rated very high compared to conventional techniques like logistic regression or commercial programs like "Pipeline Pilot".

## Virtual Screening, Binding Affinity and Scoring Functions

- Virtual Screening (VS) is a computational technique used in drug discovery to search libraries of small molecules in order to identify those structures which are most likely to bind to a drug target, typically a protein receptor or enzyme. The end result is the reduction in the number of subsequent in- vitro and in-vivo experiments required.
- The goal of VS is to identify a small number of compounds that are far more likely to bind to a given antibiotic drug target than compounds selected at random.
- Scoring functions are used to predict the strength of association or binding affinity between two molecules, such as, a small organic compound (drug) & the drug's biological target (a protein receptor)

## Virtual Screening, Neural Networks & Random Forests

- Unfortunately, these scoring functions have had drawbacks since they have many false positives and negatives. **It is here that Machine Learning is proving to be most powerful.**
- Neural Network based scoring functions are emerging as a powerful alternative. E.g., **NNScore[4**] and **NNScore2** [5]
- Random Forests are also used – e.g., **RFScore[6**] based on Binding Affinity data for Protein-Ligand complexes to implicitly capture binding effects that are hard to model explicitly(Ballester and Mitchell)

## Structure Activity Relationship (SAR) and SVM

- Structure–Activity Relationship (SAR) analysis is used to reduce the search for new drugs. The aim of SAR analysis is to discover rules that successfully predict the activity of a previously unseen compound based on its physico-chemical descriptors.
- Standard statistical approaches are less useful for SAR problems where
  - data sets contain few compounds and many descriptors
  - the relationship between structure and activity is highly non-linear
- Machines Learning Techniques have the power to tackle these issues. Burbidge [7] shows the power of Support Vector Machines in SAR analysis
  - **The Problem**: To predict the inhibition of an enzyme, dihydrofolate reductase by pyrimidines.
  - **Goal**: The task is to learn the relationship $d_n > d_m$, which states that drug *n* has a higher activity than drug *m*.

## References

[1]. Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5-32.

[2]. Socher, R., Lin, C. C., Manning, C., & Ng, A. Y. (2011). Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)* (pp. 129-136).

[3]. Unterthiner, T., Mayr, A., ünter Klambauer, G., Steijaert, M., Wegner, J. K., Ceulemans, H., & Hochreiter, S. (2014). Deep learning as an opportunity in virtual screening. In *Deep Learning and Representation Learning Workshop, NIPS*.

[4]. Durrant, J. D., & McCammon, J. A. (2010). NNScore: A neural-network-based scoring function for the characterization of protein– ligand complexes. *Journal of chemical information and modeling*, *50*(10), 1865-1871.

[5]. Durrant, J. D., & McCammon, J. A. (2011). NNScore 2.0: a neural-network receptor–ligand scoring function. *Journal of chemical information and modeling*,*51*(11), 2897-2903.

[6]. Ballester, P. J., & Mitchell, J. B. (2010). A machine learning approach to predicting protein–ligand binding affinity with applications to molecular docking. *Bioinformatics*, *26*(9), 1169-1175.

[7]. Burbidge, R., Trotter, M., Buxton, B., & Holden, S. (2001). Drug design by machine learning: support vector machines for pharmaceutical data analysis.*Computers & chemistry*, *26*(1), 5-14.

## References

[8]. Aliper, Alexander, Sergey Plis, Artem Artemov, Alvaro Ulloa, Polina Mamoshina, and Alex Zhavoronkov. (2016) Deep learning applications for predicting pharmacological properties of drugs and drug repurposing using transcriptomic data. In *Molecular Pharmaceutics*

## APPENDIX

- ❑Fitting GMM by Expectation-Minimization
- ❑More on HMM

## Using Expectation Maximization to fit GMM

- To find the probability observing a given point x, we need to sum over all possible values of hidden variable h, which can be 1 or 2 (indicating from which distribution the point x comes from).
- Start with initial estimates: $p_1 = p_2 = 0.5$; $\mu_1 = 4$, $\mu_2 = 7$, $\sigma_1{}^2 = \sigma_2{}^2 = 1$

$P(x) = \sum_{h=1}^{h=2} P(x,h) = \sum_{h=1}^{h=2} P(x|h)P(h)$

For each data point xi , assign single hidden value $h_i$.

$h_i = \arg max_h\, P(h)\, P(x_i|h)$

- Identify GMM component generating each point.

Update parameters in $P(h), P(x|h)$ by simply counting and normalizing to get MLE for $\mu j$ , $j$ , $\sigma j$ .

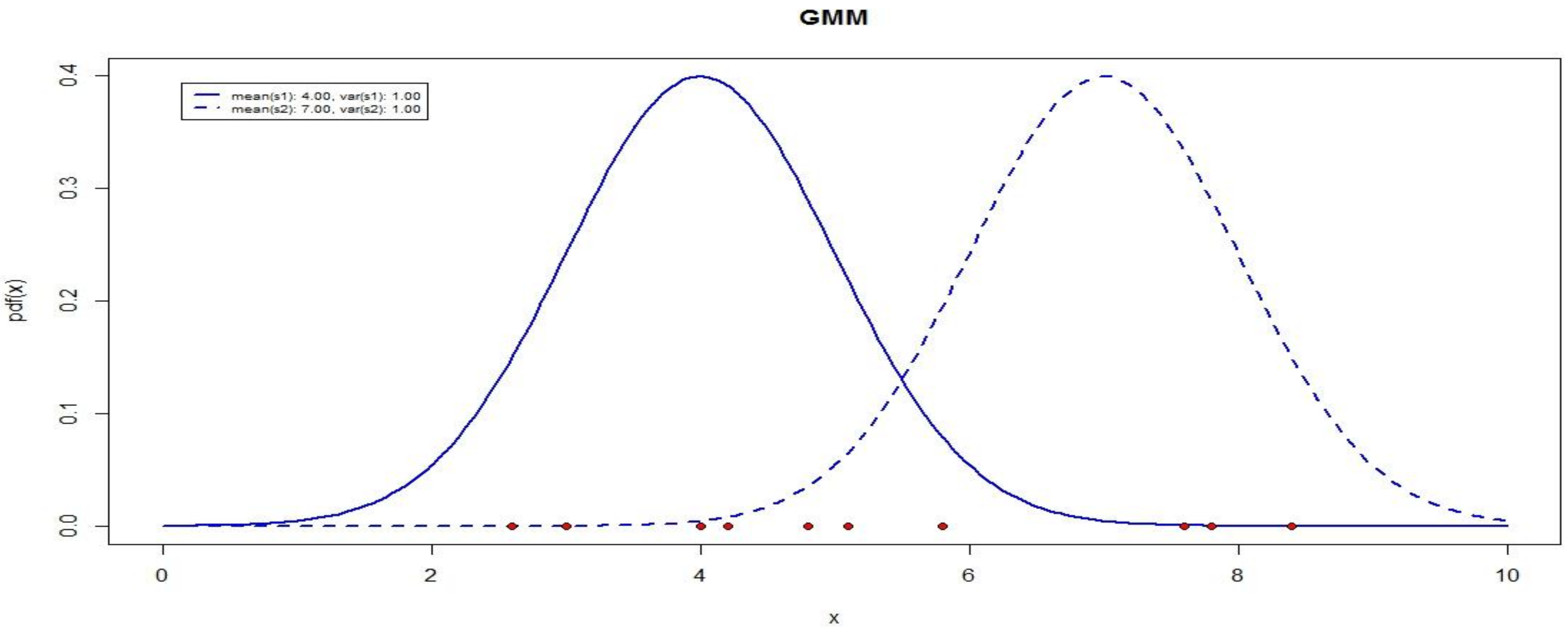
Posterior prob $P^{\sim}(h|xi) = \frac{P(h,xi)}{\sum_h P(h,xi)}$.

## GMM: Parameters after each Iteration

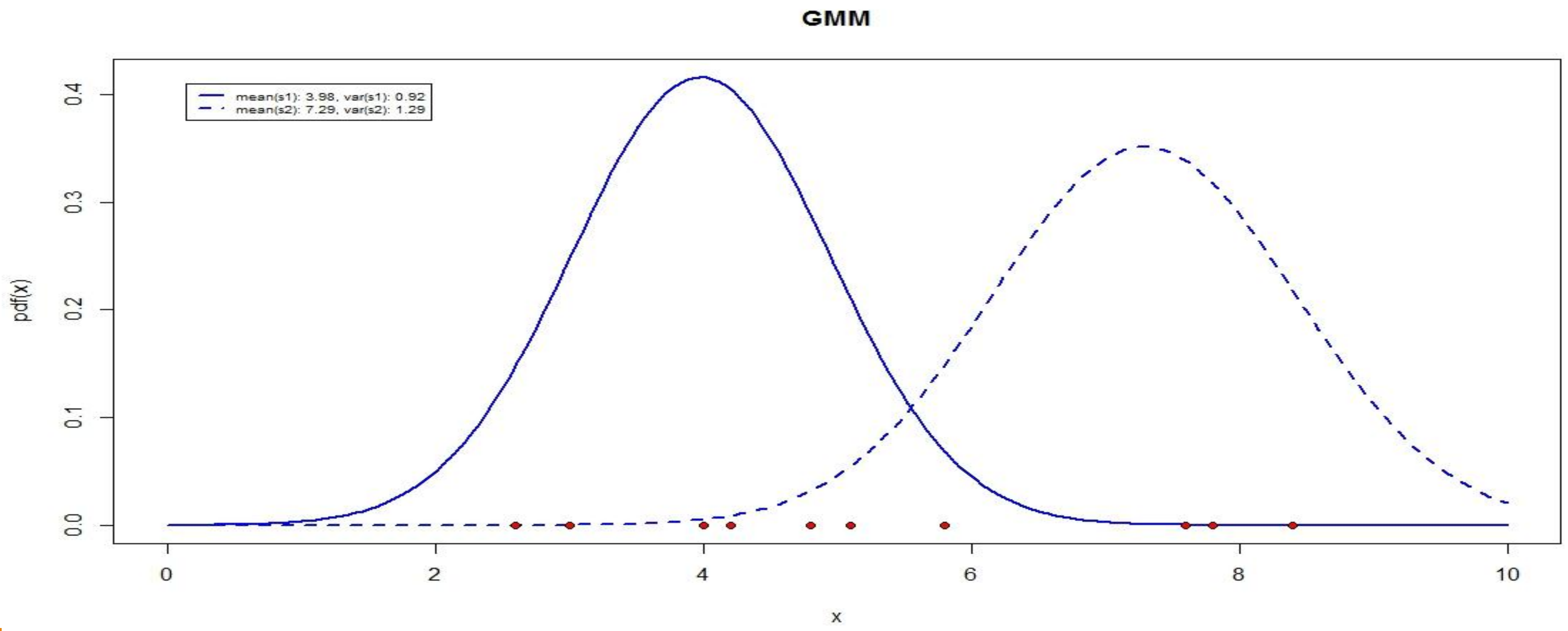
After only a few iterations, the algorithm converges to the following parameter estimates: $p_1$ =0.7, $\mu_1$ =4.22, $\sigma_1{}^2$ =1.13; and $p_2$ = 0.3, $\mu_2$ = 7.93 , $\sigma_2{}^2$ = 0.12 .The intermediate values of the parameters at each iteration are shown in the table below. The next few figures show the distributions plotted at some of the iterations.

| ***Iteration*** | $p_1$ | $\mu_1$ | $\sigma_1{}^2$ | $p_2$ | $\mu_2$ | $\sigma_2{}^2$ |
|---:|---:|---:|---:|---:|---:|---:|
| 1 | 0.59 | 3.98 | 0.92 | 0.41 | 7.29 | 1.29 |
| 2 | 0.62 | 4.03 | 0.97 | 0.38 | 7.41 | 1.12 |
| 3 | 0.64 | 4.08 | 1.00 | 0.36 | 7.54 | 0.88 |
| 4 | 0.66 | 4.14 | 1.05 | 0.34 | 7.69 | 0.59 |
| 5 | 0.69 | 4.19 | 1.10 | 0.31 | 7.85 | 0.28 |
| 6 | 0.70 | 4.22 | 1.13 | 0.30 | 7.93 | 0.12 |
| 7 | 0.70 | 4.22 | 1.13 | 0.30 | 7.93 | 0.12 |
| 8 | 0.70 | 4.22 | 1.13 | 0.30 | 7.93 | 0.12 |
| 9 | 0.70 | 4.22 | 1.13 | 0.30 | 7.93 | 0.12 |
| 10 | 0.70 | 4.22 | 1.13 | 0.30 | 7.93 | 0.12 |

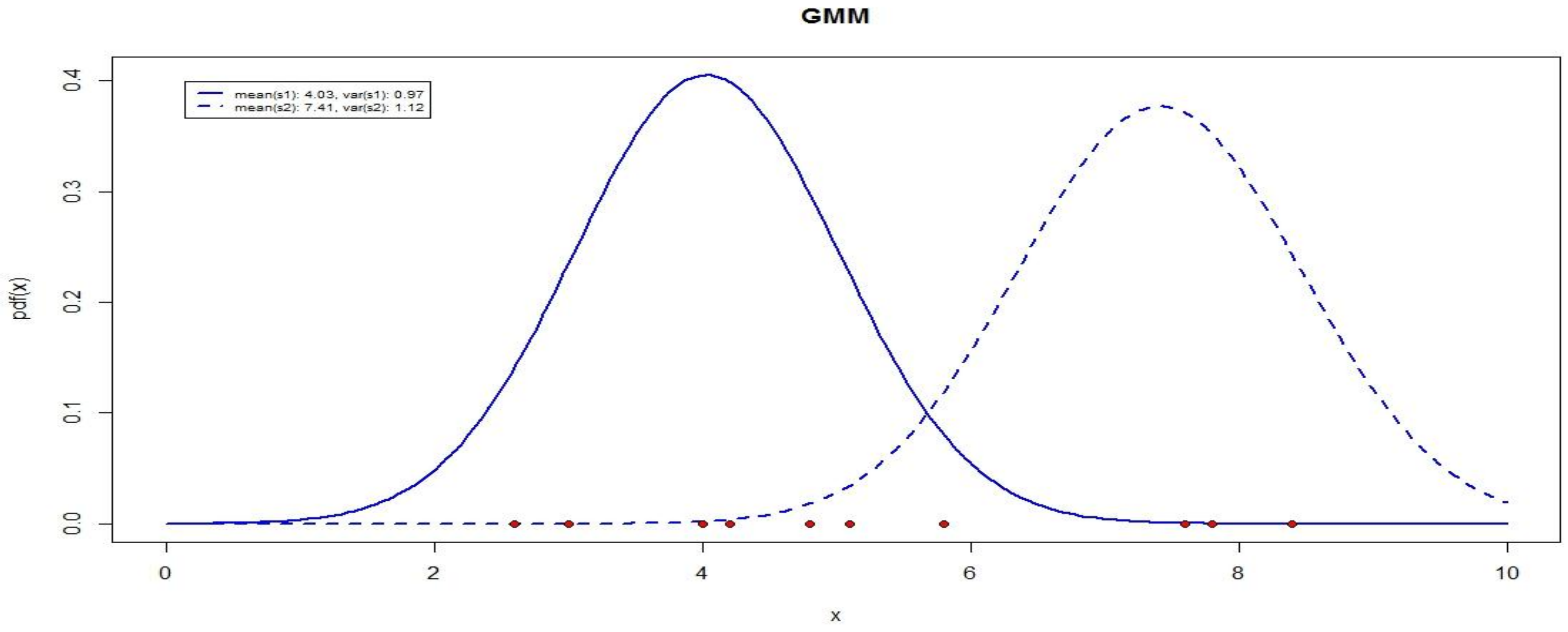
## GMM Figure using Initial Parameters



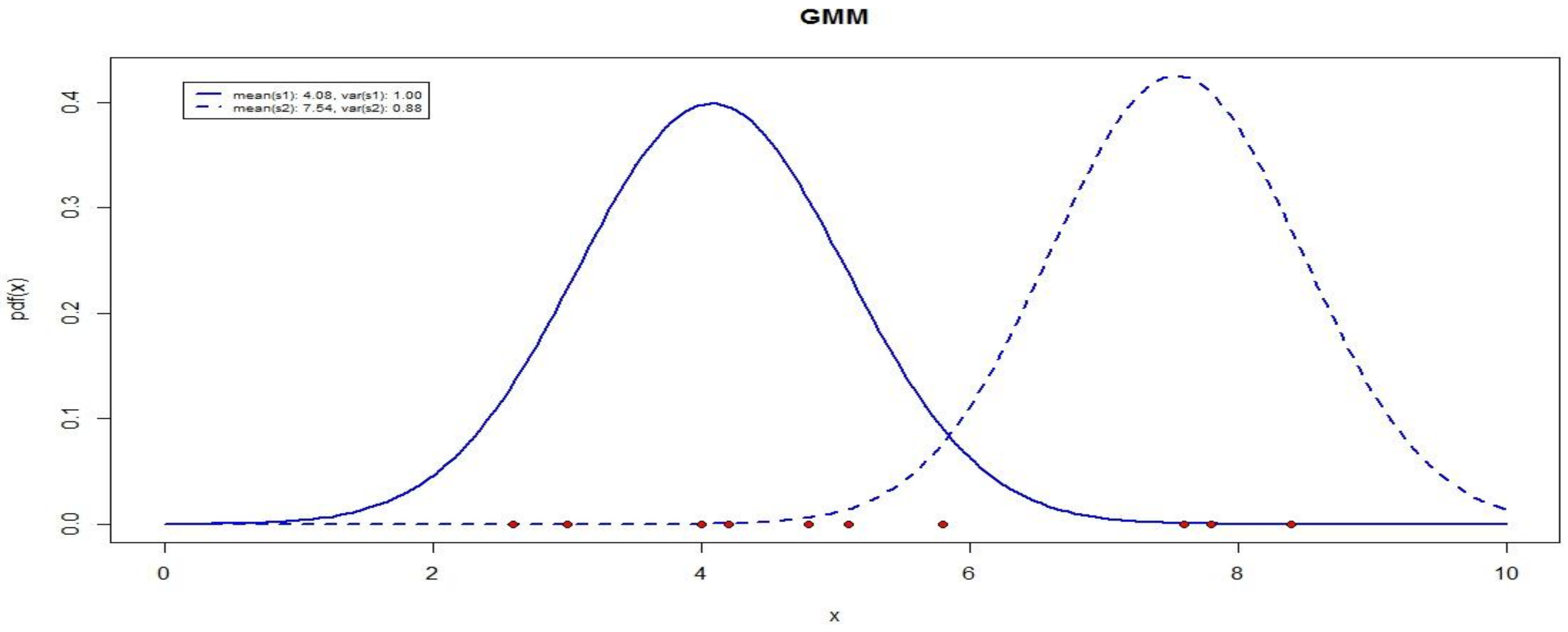
## GMM-Figure after Iteration 1



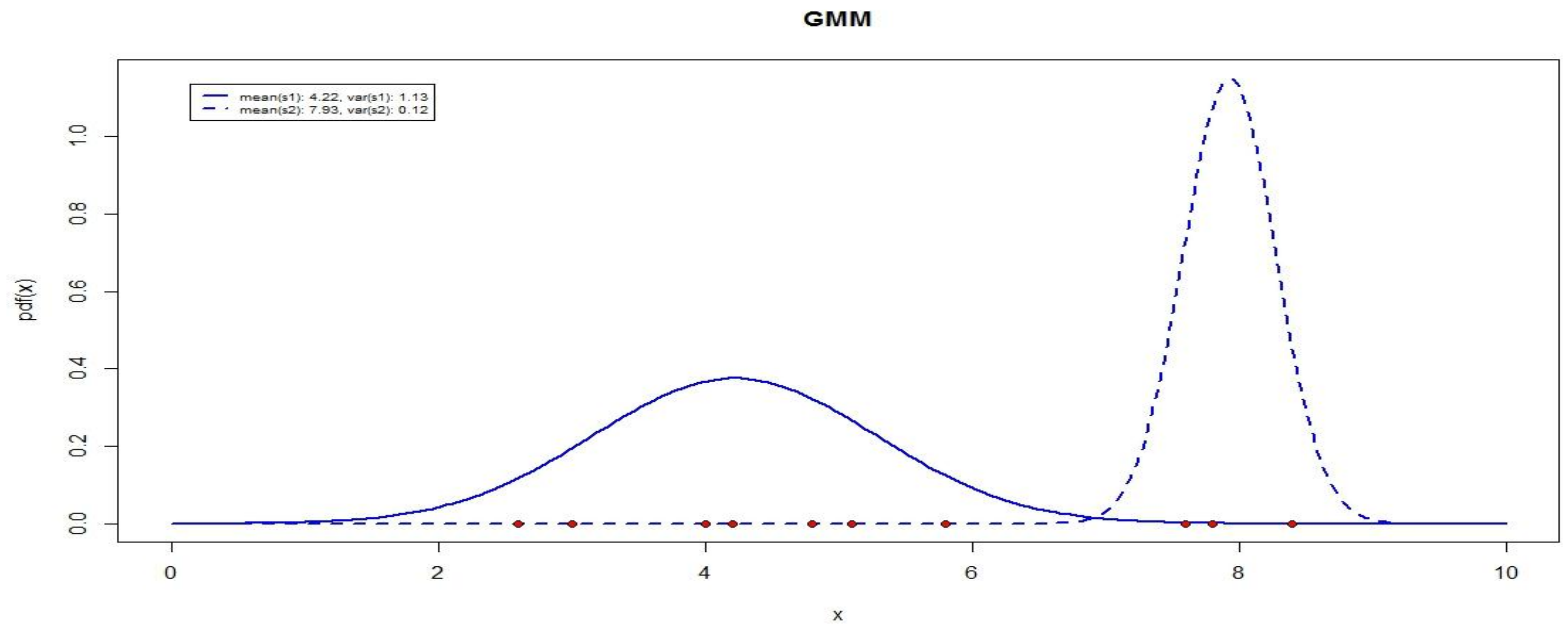
## GMM figure after Iteration 2



## GMM figure after iteration 3



## GMM- Final Figure after Iteration 10



## MATLAB code to fit the GMM

```
 p1 = 0.6;   p2 = 1-p1;
mu1= mean(x)*1.1 ; mu2 = mean(x) * 0.9;
 sigsqr1=var(x) *1.1;    sigsqr2=var(x)*0.9;
 numpoints=length(x); numiters=10;
 for i=1:numiters
   if i==1
       prob=[p1 p2];    mu= [mu1 mu2];     sigsqr=[sigsqr1 sigsqr2];  sigma_1_sqrd=sigsqr(1);   sigma_2_sqrd=sigsqr(2);
   else
       prob=[ p1_hat(i-1)  p2_hat(i-1)]; mu=[mu1_hat(i-1) mu2_hat(i-1)]; sigma_1_sqrd=sig1sqr_hat(i-1);sigma_2_sqrd=sig2sqr_hat(i-1);
       p1_hat(i-1);
   end
a1 = normpdf(x,mu(1),sqrt( sigma_1_sqrd)); b1 = normpdf(x,mu(2),sqrt( sigma_2_sqrd));
 P1N1_1 = prob(1)*a1  ;    P2N2_1  = prob(2)* b1 ;  Pxi= P1N1_1+P2N2_1  ;
```

## MATLAB code to fit the GMM-contd.

```
prob_1_given_xi = P1N1_1./Pxi ;    prob_2_given_xi = P2N2_1./Pxi    ;
   p1_hat(i)=sum(prob_1_given_xi)/(sum(prob_1_given_xi)+sum(prob_2_given_xi))       ;
  p2_hat(i)= sum(prob_2_given_xi)/(sum(prob_1_given_xi)+sum(prob_2_given_xi))        ;
  xi_prob_1_given_xi=x.*prob_1_given_xi   ;    xi_prob_2_given_xi = x.*prob_2_given_xi ;
            mu1_hat(i)=sum(xi_prob_1_given_xi)/sum(prob_1_given_xi);
    mu2_hat(i)=sum(xi_prob_2_given_xi)/sum(prob_2_given_xi);
    rep_mu1= repmat(mu1_hat(i),numpoints,1);  rep_mu2= repmat(mu2_hat(i),numpoints,1);
    centred_x_sqr_prob_1_given_xi = (x-rep_mu1).^2 .* prob_1_given_xi  ;
    centred_x_sqr_prob_2_given_xi = (x-rep_mu2).^2 .* prob_2_given_xi  ;
        sig1sqr_hat(i)=sum(centred_x_sqr_prob_1_given_xi)/sum(prob_1_given_xi) ;
    sig2sqr_hat(i)=sum(centred_x_sqr_prob_2_given_xi)/sum(prob_2_given_xi) ;
    end
```

## R code to plot the GMM figures

```
g1 = cbind(c(4,1),c(3.98,0.92),c(4.03,0.97),c(4.08,1.00),c(4.22,1.13));
g2 = cbind(c(7,1),c(7.29,1.29),c(7.41,1.12),c(7.54,0.88),c(7.93,0.12));
for (t in 1:ncol(g1))
{
  x<-seq(0,10,length=200)
  s1 = sprintf("mean(s1): %.2f, var(s1): %.2f", g1[1,t],g1[2,t])
  s2 = sprintf("mean(s2): %.2f, var(s2): %.2f", g2[1,t],g2[2,t])
  y<-dnorm(x,mean=g1[1,t], sd=sqrt(g1[2,t]))
  y2<-dnorm(x,mean=g2[1,t], sd=sqrt(g2[2,t]))
  matplot(x, cbind(y,y2),type="l",ylab= "pdf(x)",col=c("blue","blue"),lty=c(1,2), lwd = c(2,2))
  title("GMM")
```

## R code to plot the GMM figures-contd.

```
par(xpd=TRUE)
  legend("topleft", cex=0.6, inset=.05, legend=c(s1, s2), lwd=c(2.5,2.5),lty=c(1,2), col=c("blue","blue"))
  points(8.4,0, pch=21, bg="red")
  points(7.6,0,pch=21, bg="red")
  points(4.2,0,pch=21, bg="red")
  points(2.6,0,pch=21, bg="red")
  points(5.1,0,pch=21, bg="red")
  points(4.0,0,pch=21, bg="red")
  points(7.8,0,pch=21, bg="red")
  points(3.0,0,pch=21, bg="red")
  points(4.8,0,pch=21, bg="red");  points(5.8,0,pch=21, bg="red"); }
```

## Characterizing a Hidden Markov Model (HMM)

1. The N hidden states.
2. The M distinct types of observations, which are the colours of the balls in this example. We could have had continuous valued observations of course.
3. The state transition probability matrix $A = \{a_{jk}\}$ giving the conditional probability that we are in state *k* at time *(t+1)* given that we were in state *j* at time *t*, *i.e.*, $a_{jk} = P[Q_{t+1} = Sk \mid Q_t = S_j]$,where *S* and *Q* are both used for denoting the states; $S_1, S_2, \ldots, S_N$ are the *N* states and $Q_t$ is the state at time *t*.
4. The probability distribution of the observations, conditional on a state. In our example, this is the conditional probability of choosing a ball of a given colour, after the urn has been selected. $B_{jv}$ = probability of observing a ball of *v-th* colour from the *j-th* urn.
5. The initial state distribution $\Pi$, which is the probability distribution governing the initial choice of the states. The probability that the *j-th* urn is the first urn to be selected is $\Pi_j$.

## Three canonical problems of an HMM

1. **Evaluation Problem**. Given the parameters of an HMM, i.e. given ***λ**,* calculate the probability of realisation of a sequence of observations ***O***. (**Forward-Backward algorithms**). That is, compute $P[O|\lambda]$

2. **Decoding or Classification Problem**. Given an observation sequence ***O***, find the sequence of hidden states most likely to have occurred.
   That is, compute $\underset{Q}{argmax}\ P[\boldsymbol{Q}/\boldsymbol{O}]$

*w*here ***Q*** = $q_1, q_2, \ldots, q_T$ is the series of states indexed by time. (**Viterbi Algorithm**)

3. **Estimation or Training Problem.** Given a sequence of observations, fit the HMM. That is estimate $\boldsymbol{\lambda} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\Pi})$ the parameters of the HMM, that maximize $P[O|\lambda]$. (**Baum-Welch Algorithm,** a type of EM algorithm**)**