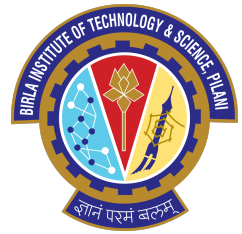
IS-ZC464: MACHINE LEARNING

Lecture-09: D-Tree (contd..), Random-Forest, K-NN, K-means



Dr. Kamlesh Tiwari
Assistant Professor
Department of Computer Science and Information Systems,
BITS Pilani, Pilani, Jhunjhunu-333031, Rajasthan, INDIA

September 15, 2018 (WILP @ BITS-Pilani Jul-Nov 2018)

Recap: Decision Tree

- **Decision Tree** represents disjunction of conjunctions. Many consistent decision trees are possible for same dataset
- **ID3** search strategy partition according to the attribute with highest

$$Gain(S, A) = Entropy(S) - \sum_{v \in Value(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

  Thereby selecting in the favor of shorter trees, and ones that place the attributes with highest information gain closest to the root
- **Issues** in decision trees include
  1. Overfitting (how deep to grow?)
  2. Handling continuous attributes (information gain and heuristics?)
  3. Choosing an appropriate attribute selection measure (Gain Ratio?)
  4. Missing attribute values (most common or probability?)
  5. Attributes with differing costs, and
  6. Improving computational efficiency

Recap: Decision Tree

Given following data

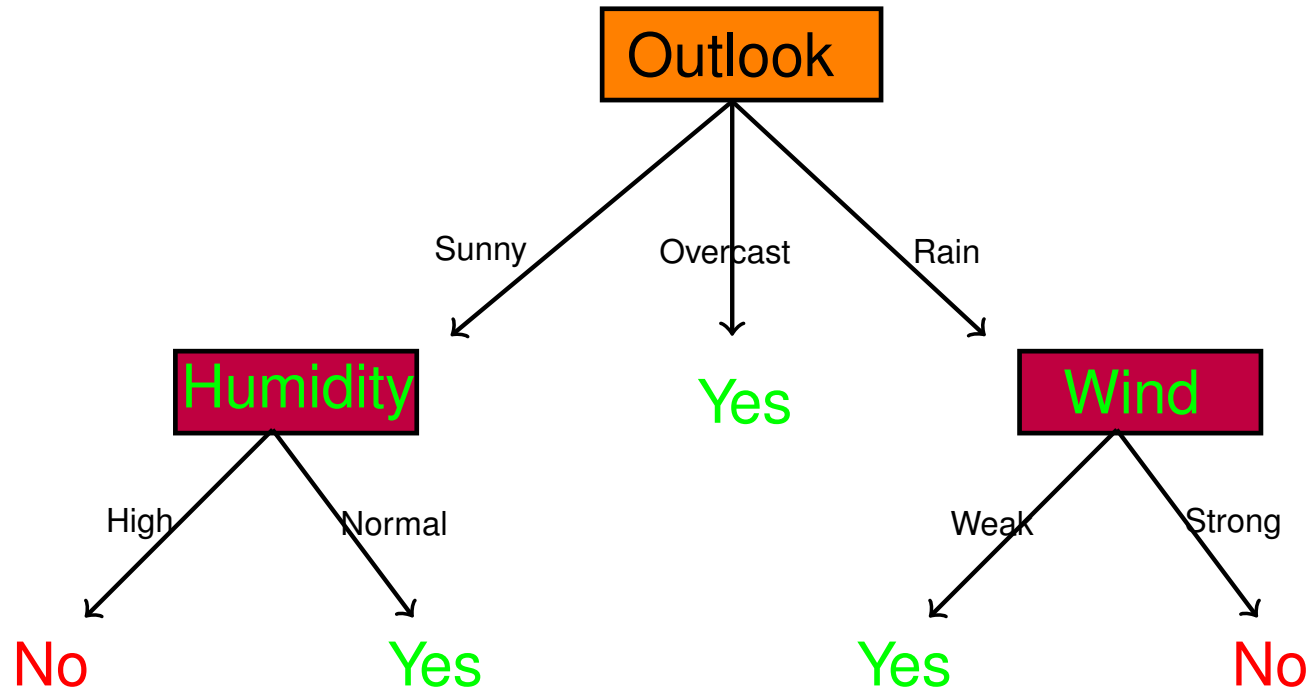
Day	Outlook	Temperature	Humidity	Wind	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rainy	Mild	High	Weak	Yes
D5	Rainy	Cool	Normal	Weak	Yes
D6	Rainy	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rainy	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rainy	Mild	High	Strong	No

Find classification for

$(Outlook = Rain, Humidity = High, Wind = Weak)$

ALERT: (missing value) what is Temperature?

Recap: Iterative-Dichotomiser-3 (ID3) Algorithm



Classification for $(Outlook = Rain, Humidity = High, Wind = Weak)$ is

YES

C4.5 Algorithm

It is a statistical classifier that extends ID3 algorithm by dealing with both **continuous** and discrete attributes, **missing values** and **pruning** trees after construction.

- Determines pessimistic estimate by calculating the rule accuracy over the training example, then calculating the standard deviation in this estimated accuracy assuming a binomial distribution.
- For large data sets, the pessimistic estimate is very close to the observed accuracy (the standard deviation is very small), whereas it grows further from the observed accuracy as the size of the data set decreases.
- Although this heuristic method is not statistically valid, it has nevertheless been found useful in practice.

Incorporating Continuous-Valued Attributes

- Dynamically define new discrete-valued attributes that partition the continuous attribute value into a discrete intervals
- Dynamically create a new boolean attribute A_c that is true if $A < c$ and false otherwise
- The question is, how to select the best value for the threshold c
- Pick a threshold, c, that produces the greatest information gain
- By sorting the examples according to the continuous attribute A, then identifying adjacent examples that differ in their target classification, we can generate a set of candidate thresholds midway between the corresponding values of A.
- Continuous attribute can also be splits into multiple intervals

Example

$S =$

Temperature	40	48	60	72	80	90
playTennis	N	N	Y	Y	Y	N

$$Gain(S, A) = Entropy(S) - \sum_{v \in Value(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

- Entropy(S)=1
- Partitioned at 44 produces [0+,1-] and [3+,2-] having entropies 0 and $-\frac{3}{5}\log\frac{3}{5} - \frac{2}{5}\log\frac{2}{5} = 0.29$
  $Gain(S, A_{44}) =$
  $1 - \frac{1}{6}0 - \frac{5}{6}0.29 = 0.243$

Similarly

- Determine $Gain(S, A_{54})$?
- Determine $Gain(S, A_{66})$?
- Determine $Gain(S, A_{76})$?
- Determine $Gain(S, A_{85})$?

Alternative Measures for Selecting Attributes

- There is a natural bias in the information gain measure that favors attributes with many values over those with few values
- Consider attribute *Date*, it would have the highest information gain of any of the attributes. Because *Date* alone perfectly predicts the target attribute over the training data.
- Thus, it would be selected as the decision attribute for the root node of the tree
- **Gain Ratio** is a measure that penalizes attributes such as *Date* by incorporating a term, called split information, that is sensitive to how broadly and uniformly the attribute splits the data:

$$splitInformation(S, A) = -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log \frac{|S_i|}{|S|}$$

- **Gain Ratio:** measure is defined as

$$GainRatio(S, A) = \frac{Gain(S, A)}{splitInformation(S, A)}$$

Alternative Measures for Selecting Attributes

- One practical issue arises using **GainRatio** when denominator is zero or very small ($|S_i| \lll |S|$)
- This makes **GainRatio** undefined or very large
- We can adopt some heuristic such as first calculating the Gain of each attribute, then applying the **GainRatio** test only considering those attributes with above average Gain
- An alternative to the **GainRatio**, is a distance-based measure introduced by Lopez de Mantaras (1991)[1]. Each attribute is evaluated based on the distance between the data partition it creates and the perfect partition (i.e., the partition that perfectly classifies the training data). The attribute whose partition is closest to the perfect partition is chosen.

[1] De Mántaras, R López, "A distance-based attribute selection measure for decision tree induction", Machine learning, 6(1), pp 81–92, Springer-1991

Missing Attribute Values

- In a medical domain, in which we wish to predict patient outcome based on various laboratory tests, it may be that the lab test Blood-Test-Result is available only for a subset of the patients.
- let for $< x, c(x) >$ we do not have value of $A(x)$
- One strategy for dealing with the missing attribute value is to assign it the value that is most common among training examples at node n
- A second, more complex procedure is to assign a probability to each of the possible values of A

Attributes with Differing Costs

- Instance attributes may have associated costs
- Consider attributes Temperature, BiopsyResult, Pulse, BloodTestResult
- We would prefer decision trees that use low-cost attributes where possible, relying on high-cost attributes only when needed to produce reliable classifications.
- We might divide the Gain by the cost of the attribute, so that lower-cost attributes would be preferred (without guarantee it puts a bias)
- Tan and Schlimmer for robots, demonstrated more efficient recognition strategies using $Gain^2(S, A)/Cost(A)$
- Nunez for medical diagnosis rules proposed (for $w \in [0, 1]$)

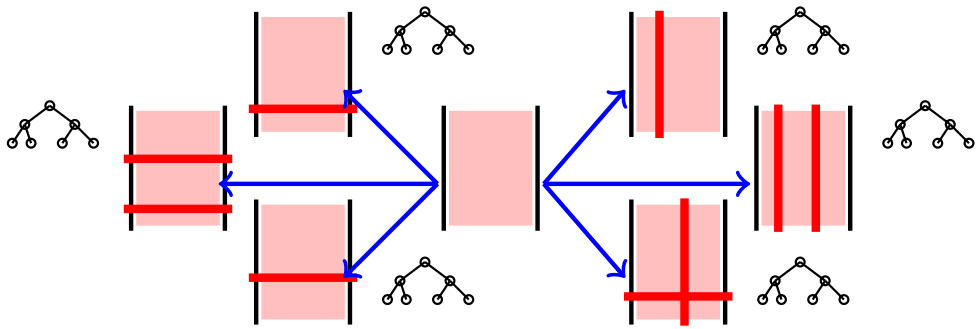
$$\frac{2^{Gain(S,A)} - 1}{(Cost(A) + 1)^w}$$

Random Forest

Combination of learning models (ensemble of classifiers) increases classification accuracy. Averaging compensates noise. Resulting model has low variance

Random Forest[2] is a large collection of decorrelated decision trees

- Training data is divided into a number of sub-sets (delete row or columns) that may have overlap (or subset of attributes)



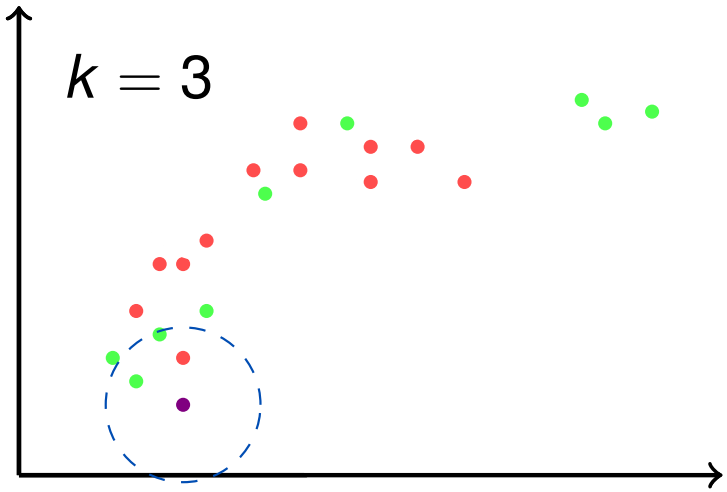
- For every subset, built a decision tree
- To classify new element: **apply voting**

[2] Leo Breiman, "Random Forests", ML 45, pp 5-32, 2001

K Nearest Neighbor (KNN)

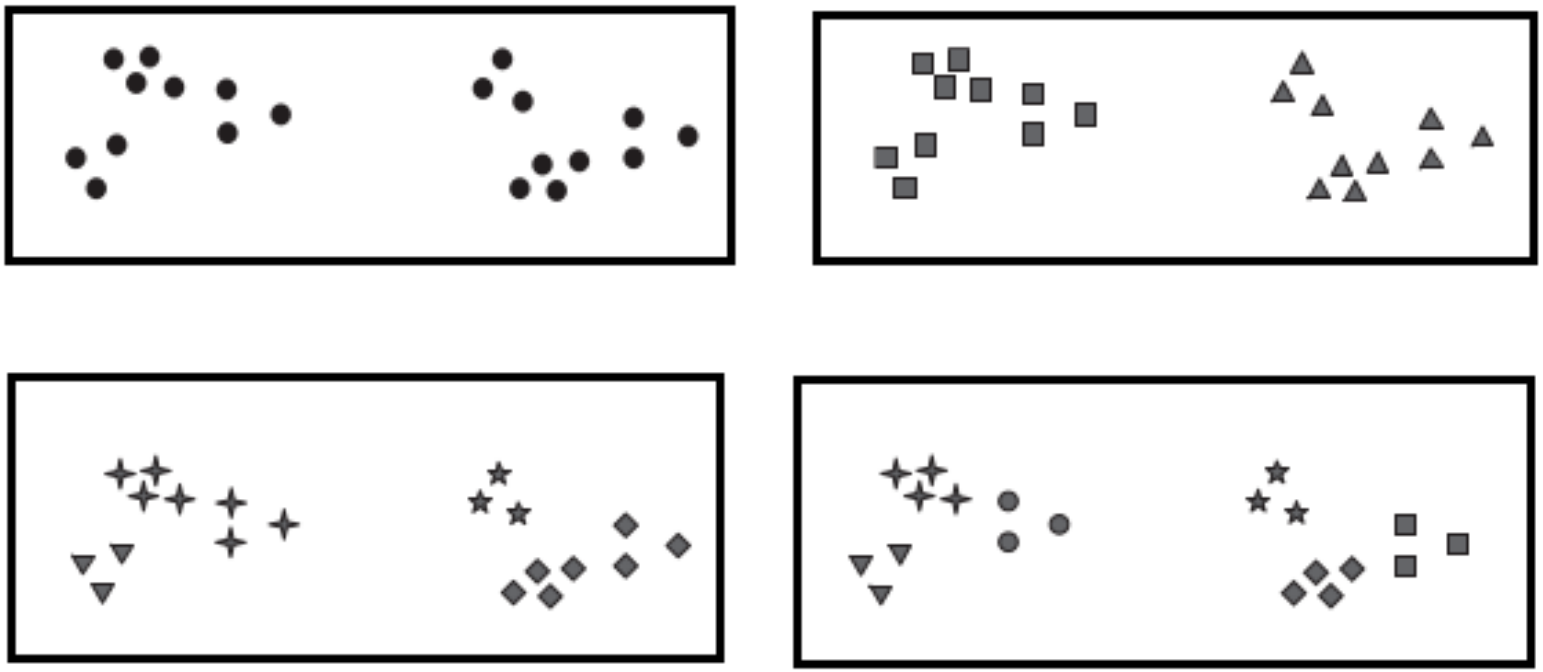
You are most likely as your friends (Bias)

- Two step algorithm
  1. Search k other datum points (most difficult part)
  2. Apply majority voting
- A lazy learner
- To avoid ties, k should NOT be a multiple of number of classes
- Small k is sensitive to noise and large one has high bias



Clustering

Grouping data based on their homogeneity (similarity or closeness).



Objects within a group are similar (or related) and are different from the objects in other groups. When it is better?

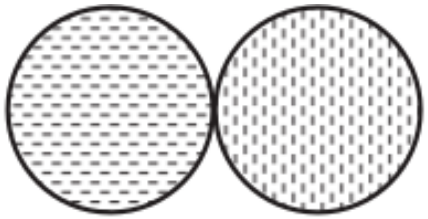
Clustering

- **Unsupervised** in nature (i.e. right answers are not known)
- Clustering is useful to 1) Summarization, 2) Compression, and 3) Efficiently Finding Nearest Neighbors
- **Type:**
  - Hierarchical (nested) versus Partitional
  - Exclusive versus Overlapping versus Fuzzy
  - Complete versus Partial
- **K-means:** This is a prototype-based[3], partitional clustering technique that attempts to find a user-specified number of clusters (K), which are represented by their centroids.

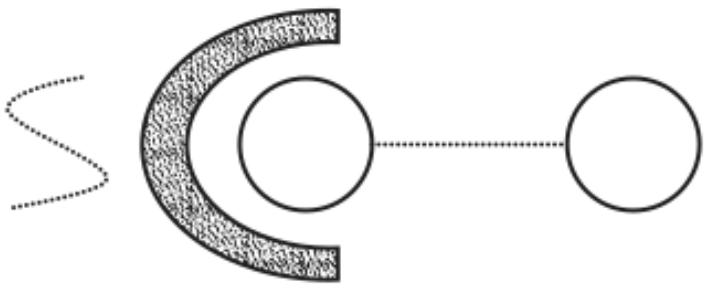
[3] object is closer (more similar) to a prototype

Clustering Approaches

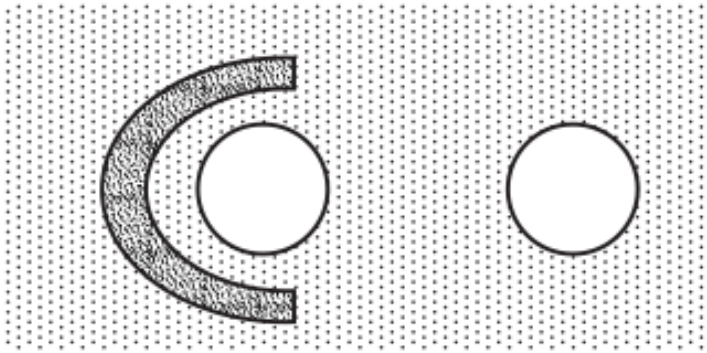
Well-separated clusters.



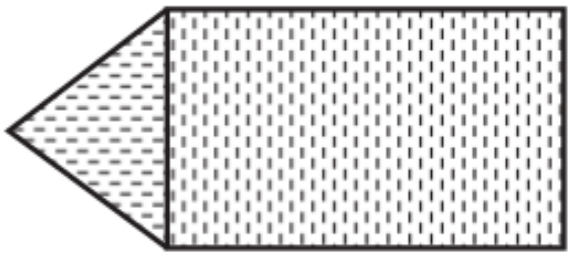
Center-based clusters.

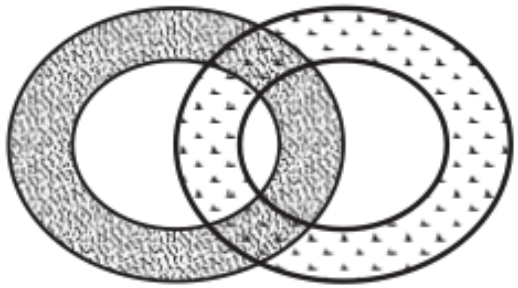


Contiguity-based clusters.



Density-based clusters.





Conceptual clusters.

K-means Algorithm

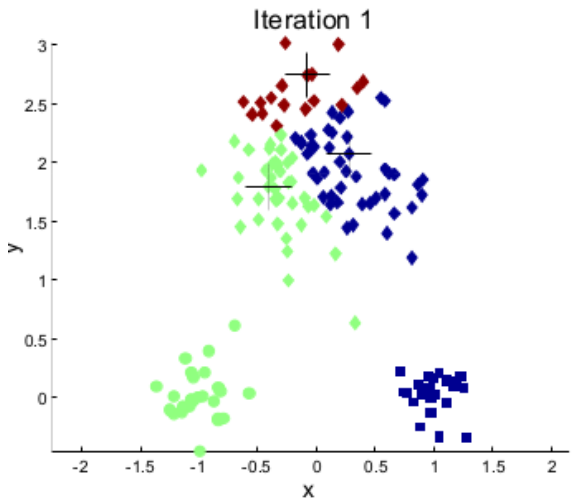
Number of clusters *i.e.* the value of K is provided by the user

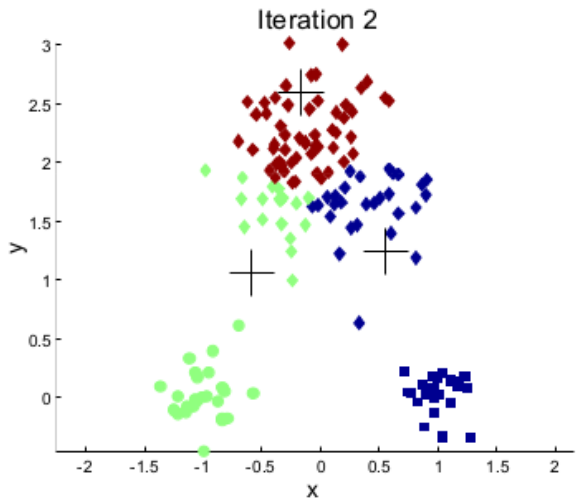
Algorithm 3: K-means

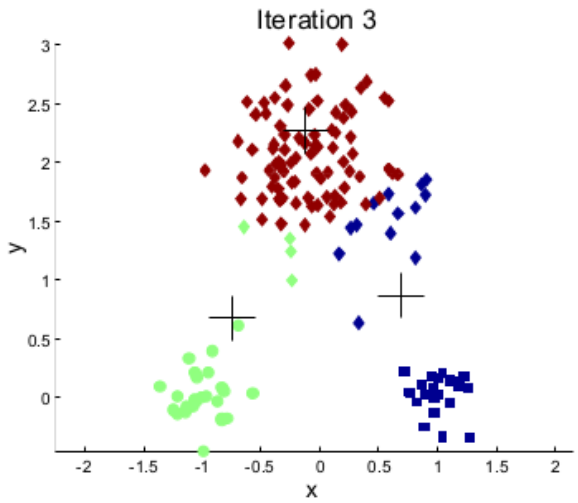
```
1 Randomly select K points as centroids
2 repeat
3     foreach datum point d_i do
4         Assign d_i to one of the closest centroids
          (thereby forming K clusters)
5     Recompute centroid (mean) for each cluster
6 until The centroids converge;
```

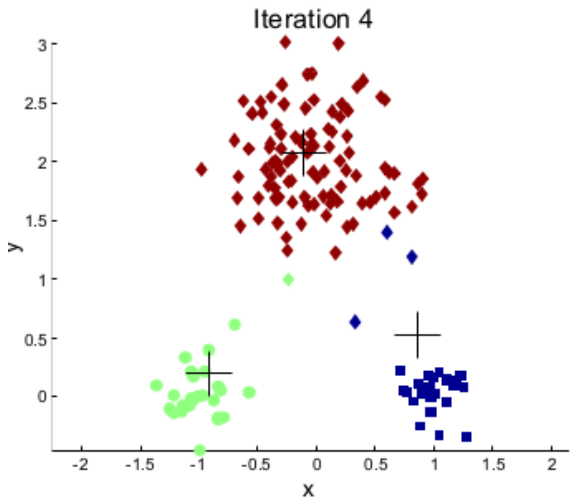
Closeness is measured by **Euclidean distance**, cosine similarity, correlation, Bregman divergence *etc*

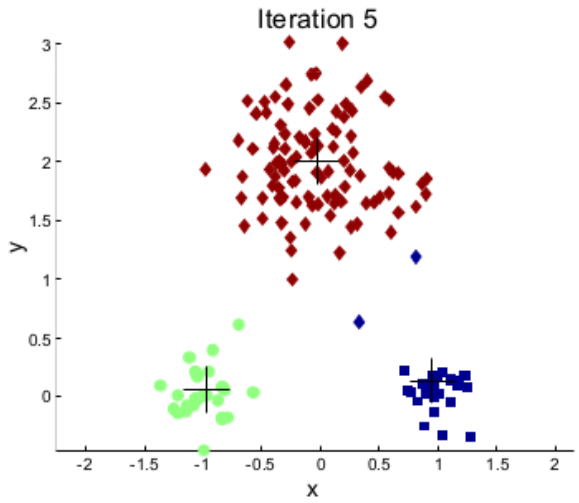
K-means in Action

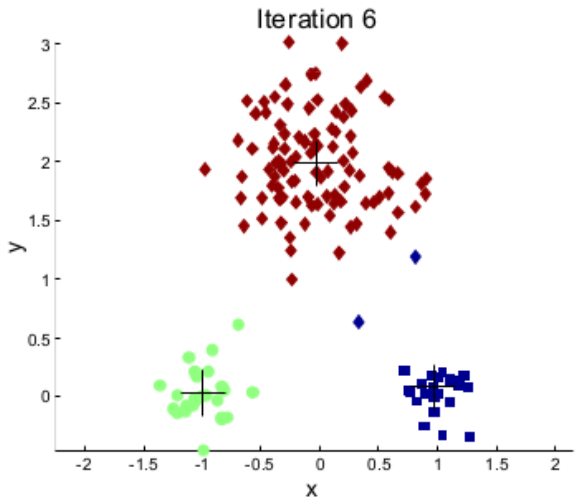












Evaluation of K-means[4]

For a given data set $\{x_1, x_2, ..., x_n\}$, let K-means partitions it in $\{S_1, S_2, ..., S_K\}$ then the objective is

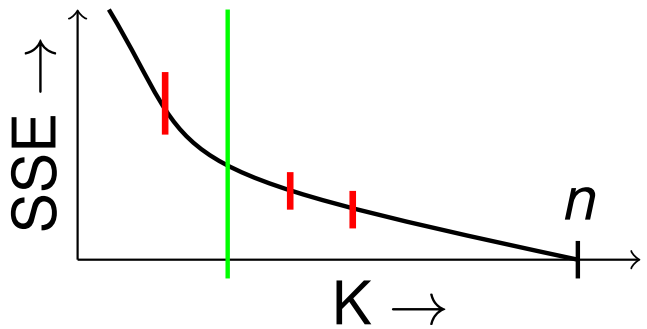
$$\underset{S}{\operatorname{argmin}} \sum_{i=1}^{K} \sum_{x \in S_i} dist^2(x, \mu_i)$$

where μ_i corresponds to i^{th} centroid. $\mu_i = \frac{1}{|S_i|} \sum_{x \in S_i} x$

- Typical choice for *dist* function is Euclidean Distance

How to proceed?

- Choose a K (How?)
  - Run K-means algorithm multiple times
  - Choose clusters corresponding to the one that minimized sum of squared error (SSE)
- If $K == n$, no error.
- Good clustering has smaller K

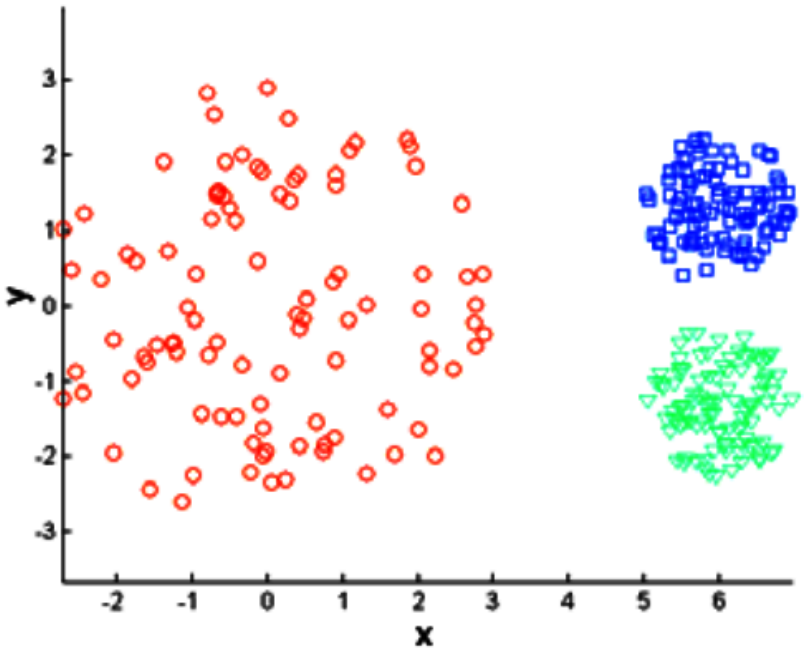


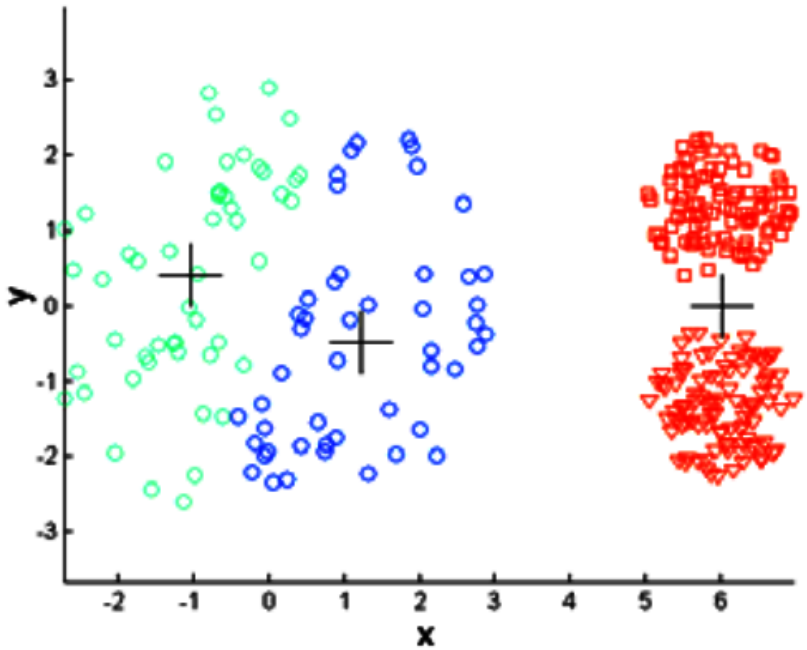
[4] Hamerly, Greg and Elkan, Charles, "Learning the k in k-means", pp 281–288, NIPS-2003

Evaluation of K-means

- **Choosing K:** 1) Domain Knowledge, 2) Preprocessing with another algorithm, 3) Iteration on K
- **Initialization of Centers:** 1) Random point in space, 2) Random point of data, 3) look for dense region, 4) Space uniformly in feature space
- **Cluster Quality:** 1) Diameter of cluster verses Inter-cluster distance, 2) Distance between members of a cluster and the cluster center, 3) Diameter of smallest sphere, 4) Ability to discover hidden patterns

Limitations of K-means





- Has problem when data has
  - Different size clusters
  - Different densities
  - Non-globular shape
- Handling Empty Clusters
- When there are outliers
- Updating Centroids Incrementally

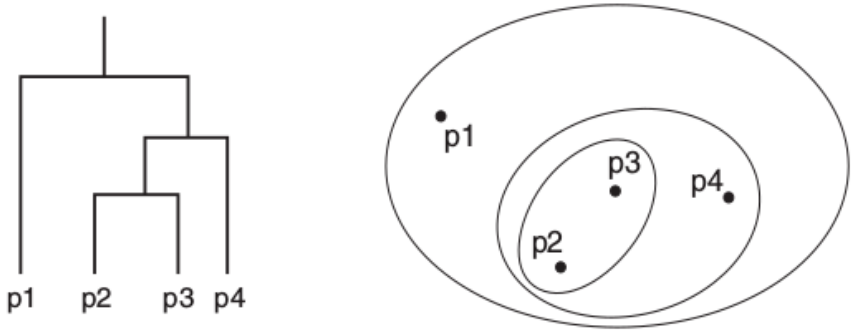
Important Note:

- K-Means and K-NN are different (K nearest neighbors)

K-NN is a **supervised** approach for **classification**

Other Clustering Approaches

- **K-Medoids:** chooses data point as center and minimizes a sum of pairwise dissimilarities. Resistance to noise and/or outliers
- **Agglomerative Hierarchical Clustering:** repeatedly merging the two closest clusters until a single (Single Link)

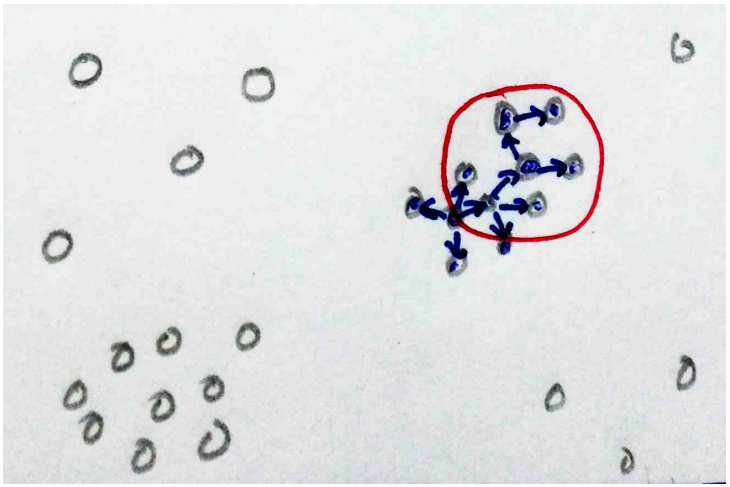


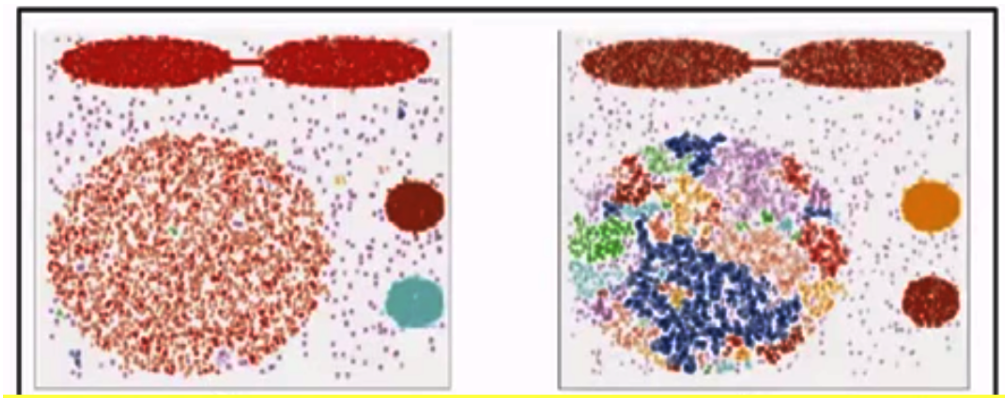
- **DBSCAN:** density-based clustering algorithm that produces a partitional clustering, in which the number of clusters is automatically determined by the algorithm.

DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a spatial clustering algorithm of KDD96

- Parameters (Eps/MinPts) and points (core/border/noise)
- Uses DFS

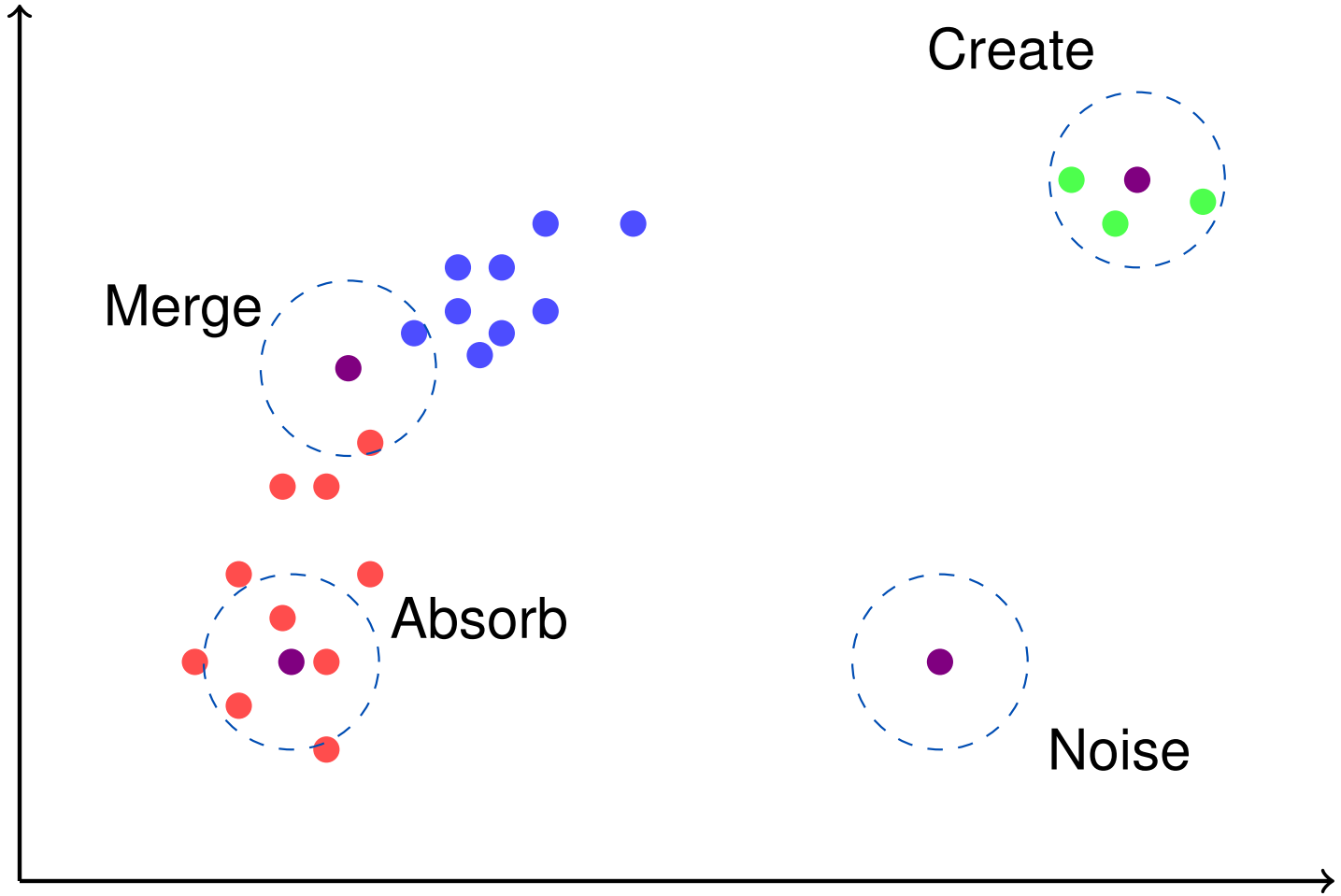




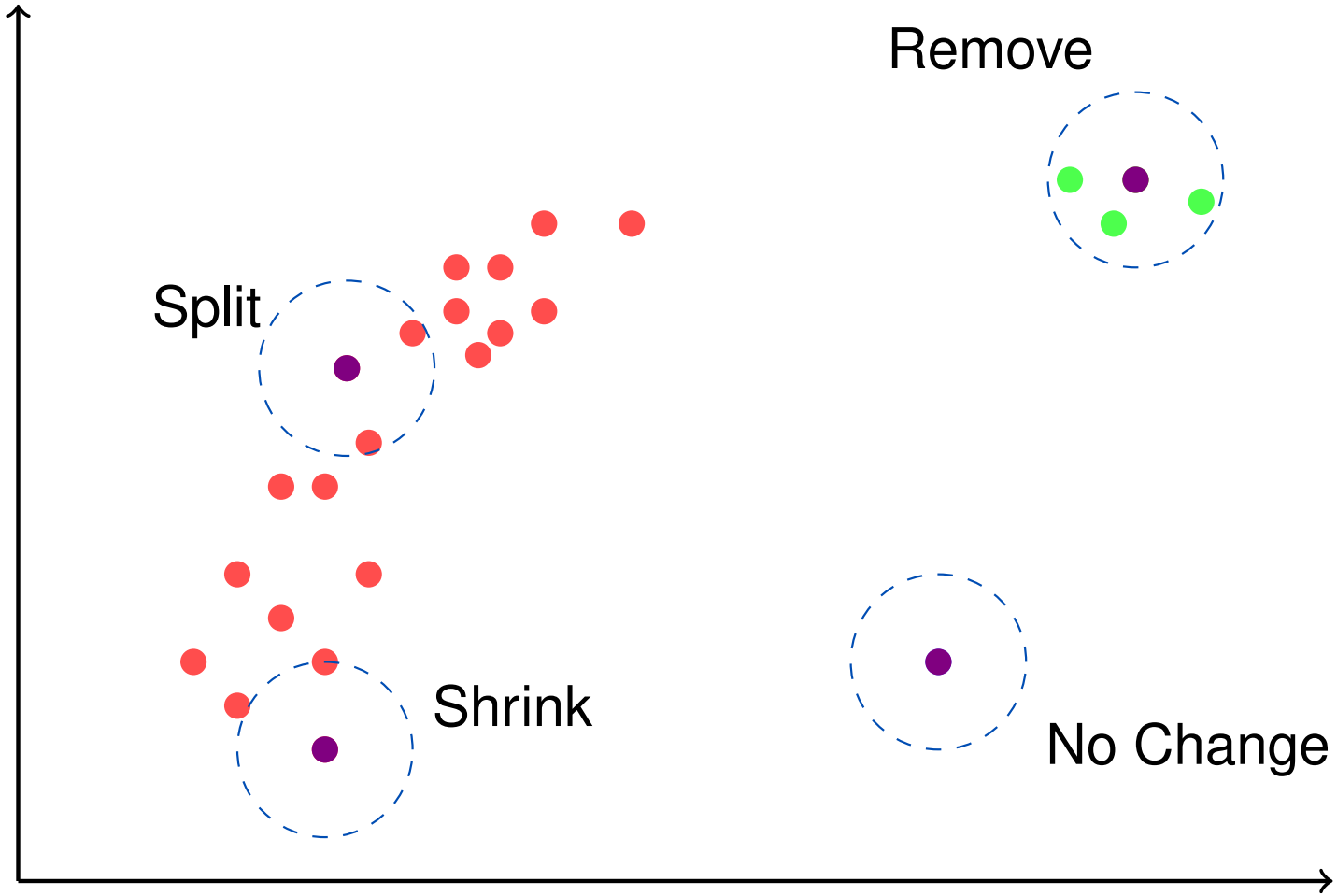
Figures from G. Karypis, E.-H. Han, and V. Kumar, *COMPUTER*, 32(8), 1999

- Disadvantage: Sensitive to parameters
- Advantage: 1) clusters of arbitrary shape, 2) Can handle dynamic databases

Incremental DBSCAN (Addition)

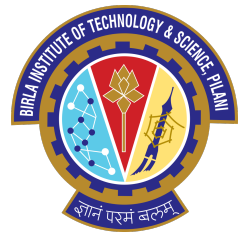


Incremental DBSCAN (Deletion)



IS-ZC464: MACHINE LEARNING

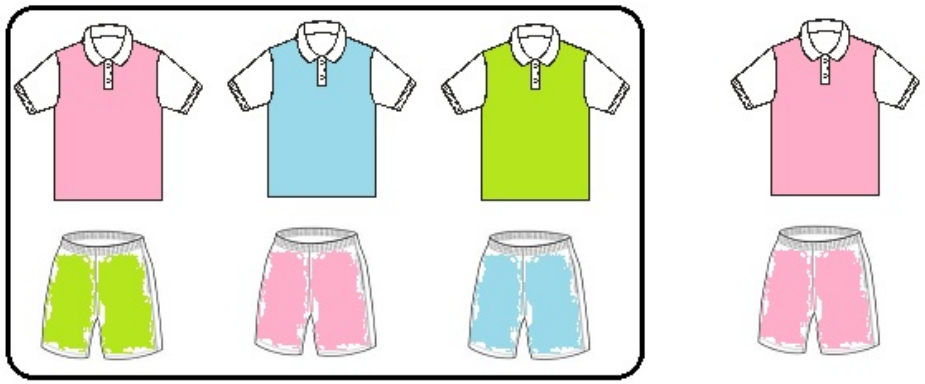
Lecture-10: Naive Bayes Classifier, Linear Model for Regression



Dr. Kamlesh Tiwari
Assistant Professor
Department of Computer Science and Information Systems,
BITS Pilani, Pilani, Jhunjhunu-333031, Rajasthan, INDIA

September 22, 2018 (WILP @ BITS-Pilani Jul-Nov 2018)

Probability: Independence



There are
three T-shirts T_r, T_g, T_b
and
three shorts S_r, S_g, S_b
in a shop

What is the probability that someone would purchase (T_r, S_r)

Independence

Selling any T-shirt or shorts

$$\begin{aligned} Pr(T_r, S_r) &= Pr(T_r) \times Pr(S_r) \\ &= 1/3 \times 1/3 = 1/9 \end{aligned}$$

Dependence

Selling T-shirt and shorts in set of same color

$$Pr(T_r, S_r) = Pr(T_r \ \& \ S_r) = 1/3$$

There something called conditional independence

Bayes Optimal Classifier

Switching the question, from "which is most probable hypothesis?" to "what is the most probable classification of the new instance?"

Is it possible to do better then MAP?

Example: Let posterior probabilities of three hypotheses h_1, h_2, h_3 given the training data are 0.4, 0.3, and 0.3 (obviously h_1 is MAP)

- Let classification of a new instance x is positive by h_1 and negative by h_2 and h_3
- By taking all hypotheses into account, the probability that x is positive is 0.4, and negative is 0.6
  - *Most probable classification is negative and it differs from MAP*

Bayes optimal classification:

$$\underset{v_j \in V}{\operatorname{argmax}} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

where classification v_j is from V and $P(v_j|D)$ is the correct classification

Bayes Optimal Classifier

$$\boxed{\text{argmax}_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)}$$

$V = \{\oplus, \ominus\}$

$P(h_1\|D)$=0.4	$P(\ominus\|h_1)$=0	$P(\oplus\|h_1)$=1
$P(h_2\|D)$=0.3	$P(\ominus\|h_2)$=1	$P(\oplus\|h_2)$=0
$P(h_3\|D)$=0.3	$P(\ominus\|h_3)$=1	$P(\oplus\|h_3)$=0

Therefore,

$$\sum_{h_i \in H} P(\oplus|h_i)P(h_i|D) = 0.4 \qquad \sum_{h_i \in H} P(\ominus|h_i)P(h_i|D) = 0.6$$

and

$$\text{argmax}_{v_j \in \{\oplus,\ominus\}} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) = \ominus$$

This type of classifier is called a Bayes optimal classifier, or Bayes optimal learner.

Bayes Optimal Classifier

- No other classification method using the same hypothesis space and same prior knowledge can outperform this method on an average.
- This method maximizes the probability that the new instance is classified correctly, given the available data, hypothesis space, and prior probabilities over the hypotheses
- Note that the predictions made by Bayes optimal classifier may not be contained in H

Issue: Although the Bayes optimal classifier obtains the best performance that can be achieved from the given training data, it can be quite costly to apply[1]

[1] we could have infinitely many hypothesis

GIBBS Algorithm [2]

A less optimal method is the Gibbs algorithm

- For a new instance x
  1. Choose a hypothesis $h \in H$ at random, according to the posterior probability distribution over H
  2. Use h to predict the classification of the instance x

Importance

Under certain conditions the expected misclassification error for the Gibbs algorithm is at most twice the expected error of the Bayes optimal classifier

[2] Opper, Manfred and Haussler, David, "Generalization performance of Bayes optimal classification algorithm for learning a perceptron", In Physical Review Letters, 66(20), pp-2677, APS-1991

Naive Bayes Classifier

Bayes classifier is a highly practical Bayesian learning method

- In some domains, its performance found to be comparable to neural network and decision tree
- The Bayesian approach to classify a new instance is to assign the most probable target value describing the instance
  $v_{MAP} = \text{argmax}_{v_j \in V} P(v_j | a_1, a_2, ..., a_n)$
- We can use Bayes theorem to rewrite this expression as

$$\begin{aligned} v_{MAP} &= \underset{v_j \in V}{\text{argmax}} \frac{P(a_1, a_2, ..., a_n | v_j) P(v_j)}{P(a_1, a_2, ..., a_n)} \\ &= \underset{v_j \in V}{\text{argmax}} P(a_1, a_2, ..., a_n | v_j) P(v_j) \end{aligned} \quad (1)$$

Naive Bayes has assumption is that the attribute values are conditionally independent given the target value

Naive Bayes Classifier

If attribute values are conditionally independent given the target value

- Under this assumption,
- Given a target value, the probability of observing the conjunction $< a_1, a_2, ..., a_n >$ is just the product of the probabilities.

$$P(a_1, a_2, ..., a_n | v_j) = \Pi_i P(a_i | v_j)$$

Naive Bayes classifier

is the one which

$$\underset{v_j \in V}{\text{argmax}} P(v_j) \Pi_i P(a_i | v_j)$$

Example: Naive Bayes Classification

Given the data

Day	Outlook	Temperature	Humidity	Wind	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rainy	Mild	High	Weak	Yes
D5	Rainy	Cool	Normal	Weak	Yes
D6	Rainy	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rainy	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rainy	Mild	High	Strong	No

Determine classification for $< Rain, Hot, High, Strong >$

Example: Naive Bayes Classification

Day	Outlook	Temperature	Humidity	Wind	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D6	Rainy	Cool	Normal	Strong	No
D8	Sunny	Mild	High	Weak	No
D14	Rainy	Mild	High	Strong	No

Day	Outlook	Temperature	Humidity	Wind	Play
D3	Overcast	Hot	High	Weak	Yes
D4	Rainy	Mild	High	Weak	Yes
D5	Rainy	Cool	Normal	Weak	Yes
D7	Overcast	Cool	Normal	Strong	Yes
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rainy	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes

P(Yes) = 9/14

P(No) = 5/14

Outlook

	Yes	No
Sunny	2/9	3/5
Overcast	4/9	0/5
Rain	3/9	2/5

Example: Naive Bayes Classification

- P(Yes) = 9/14 P(No) = 5/14

Outlook

	Yes	No
Sunny	2/9	3/5
Overcast	4/9	0/5
Rain	3/9	2/5

Humidity

	Yes	No
High	3/9	4/5
Low	6/9	1/5

Wind

	Yes	No
Strong	3/9	3/5
Weak	6/9	2/5

Temperature

	Yes	No
Hot	2/9	2/5
Mild	4/9	2/5
Cool	3/9	1/5

Example: Naive Bayes Classification

For $x =< Rain, Hot, High, Strong >$

P(Yes)

- $P(x|Yes) \times P(Yes)$
- $P(Rain|Yes) \times P(Hot|Yes) \times P(High|Yes) \times P(Strong|Yes) \times P(Yes)$
- $3/9 \times 2/9 \times 3/9 \times 3/9 \times 9/14$
- 0.005291...

P(No)

- $P(x|No) \times P(No)$
- $P(Rain|No) \times P(Hot|No) \times P(High|No) \times P(Strong|No) \times P(No)$
- $2/5 \times 2/5 \times 4/5 \times 3/5 \times 5/14$
- 0.027428...

So the classification is **No**

Regression

Regression predicts value of continuous a target variable

- A simplest model for regression can be a linear combination of the input variables

$$y(x, w) = w_0 + w_1 x_1 + ... + w_n x_n$$

where x is a n dimensional vector $(x_1, x_2, ..., x_n)$ representing some feature

- It can be extended by considering linear combination of fixed nonlinear functions ϕ (called basis functions)

$$y(x, w) = w_0 + \sum_{i=1}^{n} w_i \phi_i(x)$$

- In short $y(x, w) = w^T \phi(x)$
- Objective is to choose w such that it makes $y(x^{(i)}, w)$ as close to $y^{(i)}$ as possible

Regression

- Determining w, is similar to solving a minimization problem. Let us define a **squared error cost function** as

$$J(w) = \frac{1}{2m} \sum_{i=1}^{m} (y(x^{(i)}, w) - y^{(i)})^2$$

  where m is number of training examples
- Then one have to minimize the value of $J(w)$

$$\underset{w}{\operatorname{argmin}}\, J(w)$$

- Basic idea: Push w_i a bit against the direction of its gradient

Gradient Descent

Algorithm 5: Gradient Descent

1 Initialize w randomly
2 **repeat**
3 | Simultaneously update all w_j with $w_j - \alpha \frac{\partial}{\partial w_j} J(w)$
4 **until** *converge*;
5 **return** w

- Here α is a learning rate. If α is small enough then $J(w)$ would decrease in every iteration
  (large α can overshoot the minimum and may fail to converge)
- Susceptible to local minimum
- As it moves closer to local minimum, it automatically takes smaller steps as gradient decreases

Partial derivative term

$$\begin{aligned}
\frac{\partial}{\partial w_j} J(w) &= \frac{\partial}{\partial w_j} \frac{1}{2m} \sum_{i=1}^{m} (y(x^{(i)}, w) - y^{(i)})^2 \\
&= \frac{1}{2m} \sum_{i=1}^{m} \frac{\partial}{\partial w_j} (y(x^{(i)}, w) - y^{(i)})^2 \\
&= \frac{1}{2m} \sum_{i=1}^{m} 2(y(x^{(i)}, w) - y^{(i)}) \frac{\partial}{\partial w_j} (y(x^{(i)}, w) - y^{(i)}) \\
&= \frac{1}{m} \sum_{i=1}^{m} (y(x^{(i)}, w) - y^{(i)}) \frac{\partial}{\partial w_j} y(x^{(i)}, w)
\end{aligned}$$

For $y(x^{(i)}, w) = w_0 + w_1 x_1^{(i)} + ... + w_n x_n^{(i)}$ $\boxed{\frac{\partial}{\partial w_j} y(x^{(i)}, w) = x_j^{(i)}}$

$$\frac{\partial}{\partial w_j} J(w) = \frac{1}{m} \sum_{i=1}^{m} (y(x^{(i)}, w) - y^{(i)}) x_j^{(i)}$$

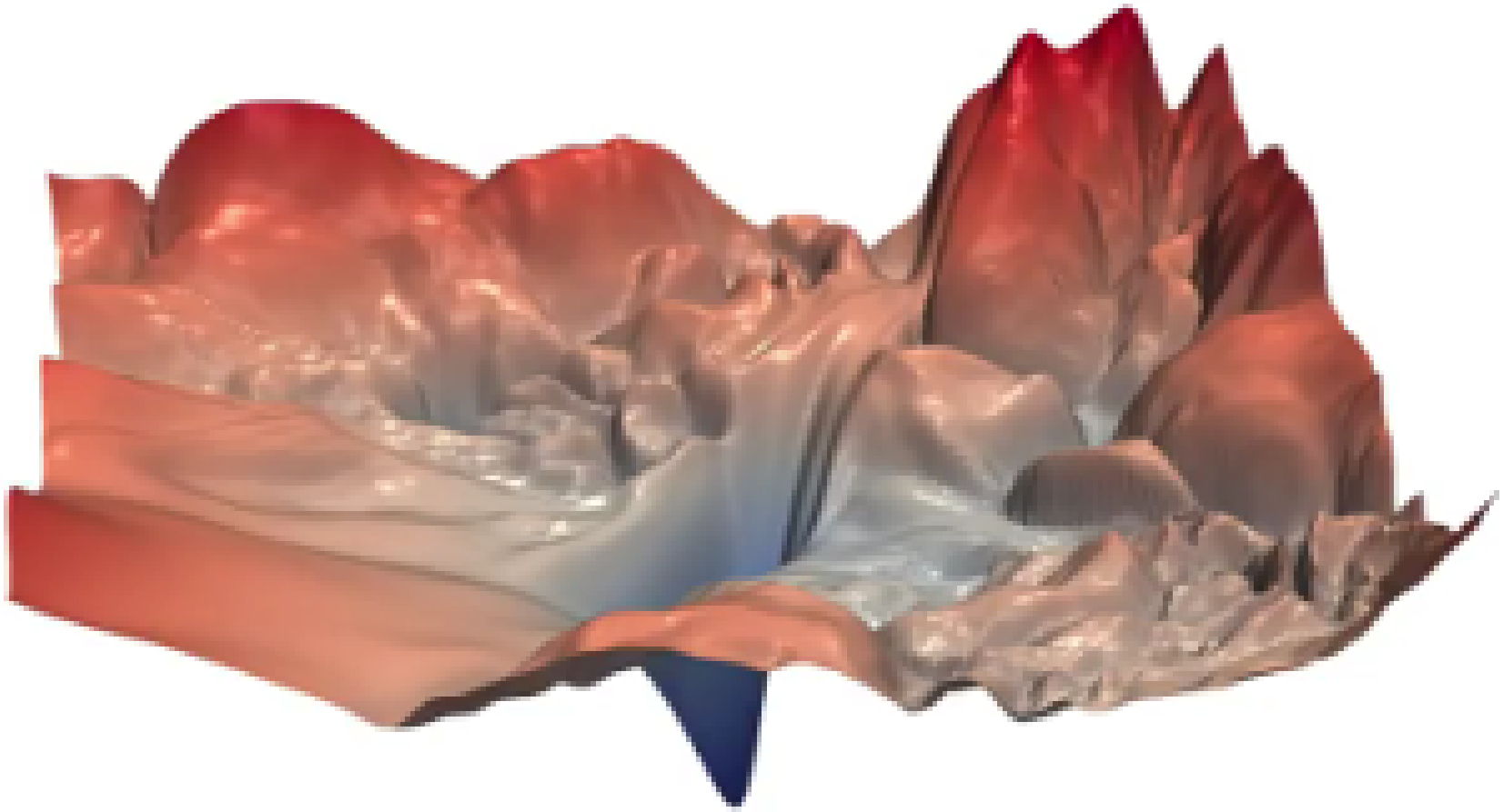
Batch-Gradient Descent

Algorithm 8: Batch-Gradient Descent

1 Initialize w randomly
2 **repeat**
3 | Simultaneously update all w_j with $w_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (y(x^{(i)}, w) - y^{(i)}) x_j^{(i)}$
4 **until** *converge*;
5 **return** w

- At every step it evaluate all training examples
- Some time it is also called multi-variate linear regression

Real Landscape



Example: Gradient Descent (learning rate α)

Consider following data

	x_1	x_2	x_3	y
1	10	50	20	10
2	11	31	22	12
3	11	12	15	4
4	20	55	20	22
5	23	41	27	1
6	31	12	35	9
7	13	18	12	23
8	21	55	16	16
9	32	56	27	22
10	8	22	35	11

Learning rate $\alpha = 0.1$

J=396.662506
w=(0.500 0.500 0.500 0.500)

J=19454472.000000
w=(-2.055 -51.070 -100.970 -62.640)

J=1036526813184.000000
w=(590.236 11518.771 23902.906 13778.349)

J=55230041021218816.000000
w=(-135891.922 -2653678.250 -5525792.000 -3170425.000)

J=2942865354556228763648.000000
w=(31365378.000 612476928.000 1275658624.000 731686912.000)

J=156806972273681738831495168.000000
w=(-7240111104.000 -141378551808.000 -294465732608.000
-168895037440.000)

J=8355266546526971027269827428352.000000
w=(1671254376448.000 32634791002112.000 67972370530304.000
38986479304704.000)

J=44520007922259187977070606888730624 0.000000

w=(-385780270759936.000 -7533178826784768.000

-15690251045437440.000 -8999357718200320.000)

Example: Gradient Descent (learning rate α)

Consider following data

	x_1	x_2	x_3	y
1	10	50	20	10
2	11	31	22	12
3	11	12	15	4
4	20	55	20	22
5	23	41	27	1
6	31	12	35	9
7	13	18	12	23
8	21	55	16	16
9	32	56	27	22
10	8	22	35	11

Learning rate $\alpha = 0.001$

J	w
396.663	(0.500 0.500 0.500 0.500)
664.137	(0.474 -0.016 -0.515 -0.131)
1131.021	(0.508 0.631 0.881 0.628)
1943.882	(0.464 -0.249 -0.910 -0.435)
3357.625	(0.523 0.888 1.492 0.914)
5815.401	(0.446 -0.630 -1.641 -0.908)
10087.491	(0.549 1.356 2.518 1.456)
17512.684	(0.415 -1.274 -2.941 -1.693)
30417.834	(0.592 2.183 4.276 2.432)
52847.020	(0.359 -2.383 -5.221 -3.028)
91828.805	(0.668 3.630 7.314 4.151)
159578.781	(0.263 -4.302 -9.200 -5.330)
277327.562	(0.799 6.152 12.580 7.155)
481973.594	(0.093 -7.633 -16.125 -9.316)
837646.250	(1.025 10.537 21.725 12.387)
1455801.375	(-0.201 -13.418 -28.168 -16.234)
2530147.500	(1.417 18.162 37.611 21.491)
4397349.000	(-0.715 -23.472 -49.103 -28.249)
7642525.500	(2.097 31.415 65.218 37.319)
13282603.000	(-1.608 -40.944 -85.492 -49.126)
23084998.000	(3.278 54.449 113.196 64.832)
40121436.000	(-3.162 -71.310 -148.738 -85.405)
69730584.000	(5.329 94.483 196.578 112.653)
121190936.000	(-5.863 -124.085 -258.660 -148.456)
210628448.000	(8.894 164.060 341.494 195.769)
366069856.000	(-10.559 -215.809 -449.705 -258.035)
636225152.000	(15.088 284.983 593.355 340.226)
1105751936.000	(-18.721 -375.224 -781.739 -448.479)
1921783808.000	(25.852 495.147 1031.086 591.291)
3340036608.000	(-32.908 -652.287 -1358.811 -779.468)

Example: Gradient Descent (learning rate α)

Consider following data

	x_1	x_2	x_3	y
1	10	50	20	10
2	11	31	22	12
3	11	12	15	4
4	20	55	20	22
5	23	41	27	1
6	31	12	35	9
7	13	18	12	23
8	21	55	16	16
9	32	56	27	22
10	8	22	35	11

Learning rate $\alpha = 0.0001$

J	w	
396.663	(0.500 0.500 0.500 0.500)	
246.798	(0.497 0.448 0.399 0.437)	
158.286	(0.495 0.408 0.321 0.388)	
105.980	(0.494 0.377 0.262 0.349)	
75.041	(0.493 0.353 0.218 0.319)	
56.711	(0.492 0.334 0.184 0.295)	
45.826	(0.491 0.320 0.159 0.276)	
39.335	(0.491 0.308 0.140 0.260)	
35.439	(0.490 0.299 0.126 0.248)	
33.077	(0.490 0.291 0.115 0.238)	
31.621	(0.490 0.285 0.108 0.229)	
30.703	(0.490 0.280 0.103 0.222)	
30.104	(0.490 0.276 0.099 0.216)	
29.694	(0.489 0.273 0.097 0.210)	
29.399	(0.489 0.270 0.096 0.206)	
29.172	(0.489 0.268 0.095 0.202)	
28.987	(0.489 0.266 0.096 0.198)	
28.830	(0.489 0.264 0.096 0.194)	
28.689	(0.489 0.262 0.097 0.191)	
28.560	(0.489 0.260 0.098 0.188)	
28.439	(0.489 0.259 0.099 0.185)	
28.325	(0.489 0.258 0.101 0.182)	
28.216	(0.489 0.256 0.102 0.179)	
28.111	(0.489 0.255 0.104 0.177)	
28.011	(0.489 0.254 0.105 0.174)	
27.913	(0.489 0.253 0.107 0.172)	
27.819	(0.489 0.252 0.109 0.170)	
27.728	(0.489 0.251 0.110 0.167)	
27.555	(0.490 0.249 0.114 0.163)	
24.926	(0.507 0.207 0.215 0.020)	**Iteration 300**
24.768	(0.710 0.219 0.213 0.005)	**Iteration 3000**

Some Tricks

- Apply feature scaling (like min-max normalization)
- Sometime mean normalization is also good idea
- Declare convergence when $J(w)$ decreases less then a small constant ϵ (say $\epsilon = 0.001$)
- Choose α as 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, ...
- Having with some insight of the problem, you may define new features as $f(x_i, x_j, ..., x_k)$
- In case one wish to use $y(x, w) = w_0 + w_1 x_1 + w_2 x_2^2 + w_3 x_3^4 + ...$ Replace the features x_2 with $x_2' = x_2^2$ and x_3 with $x_3' = x_3^4$ and so on. This would enable the use of gradient descent possible once again

Example: Gradient Descent (Feature scaling)

Feature scaling

	x_1	x_2	x_3	y
1	0.08	0.86	0.35	10
2	0.12	0.43	0.43	12
3	0.12	0.00	0.13	4
4	0.50	0.98	0.35	22
5	0.62	0.66	0.65	1
6	0.96	0.00	1.00	9
7	0.21	0.14	0.00	23
8	0.54	0.98	0.17	16
9	1.00	1.00	0.65	22
10	0.00	0.23	1.00	11

Learning rate $\alpha = 0.1$

J	w	
95.472	(0.500 0.500 0.500 0.500)	
73.399	(1.679 1.025 1.220 0.983)	
58.326	(2.658 1.455 1.822 1.364)	
48.020	(3.470 1.808 2.326 1.663)	
40.961	(4.147 2.096 2.749 1.893)	
36.116	(4.710 2.331 3.106 2.066)	
32.778	(5.180 2.522 3.407 2.193)	
30.468	(5.574 2.677 3.662 2.283)	
28.859	(5.903 2.803 3.880 2.341)	
27.729	(6.181 2.904 4.066 2.373)	
26.925	(6.415 2.985 4.226 2.385)	
26.344	(6.613 3.049 4.364 2.379)	
25.916	(6.782 3.100 4.485 2.360)	
25.593	(6.926 3.140 4.590 2.329)	
25.342	(7.050 3.170 4.683 2.289)	
25.141	(7.158 3.193 4.766 2.241)	
24.974	(7.252 3.210 4.839 2.188)	
24.833	(7.334 3.222 4.906 2.129)	
24.708	(7.407 3.230 4.966 2.067)	**Iteration 18**
24.596	(7.472 3.234 5.021 2.003)	
24.493	(7.530 3.236 5.071 1.935)	
24.397	(7.583 3.235 5.118 1.866)	
24.306	(7.632 3.233 5.161 1.796)	
24.219	(7.677 3.229 5.202 1.725)	
24.136	(7.718 3.225 5.241 1.653)	
24.056	(7.757 3.219 5.277 1.581)	
23.979	(7.794 3.213 5.311 1.509)	
23.903	(7.830 3.206 5.344 1.436)	
23.830	(7.863 3.198 5.375 1.364)	
23.759	(7.896 3.191 5.405 1.292)	**Iteration 30**
20.174	(12.021 4.618 4.794 -7.329)	**Iteration 3000**

Stochastic Gradient Descent

Gradient descent becomes computationally expensive with large training set.

- Do don't wait; **look at only a single training example in each iteration.** Takes a random looking path and wonder around minimum (may not converge)

Algorithm 12: Stochastic-Gradient Descent

```
Initialize w randomly
Randomly shuffle the dataset
repeat
    for i = 1 to m do
        Simultaneously update all w_j with
        w_j − α(y(x^(i), w) − y^(i)) x_j^(i)
until converge;
return w
```

- The idea can be applied to other learning method as well

Example: Stochastic Gradient Descent

Feature scaling

	x_1	x_2	x_3	y
1	10	50	20	10
2	11	31	22	12
3	11	12	15	4
4	20	55	20	22
5	23	41	27	1
6	31	12	35	9
7	13	18	12	23
8	21	55	16	16
9	32	56	27	22
10	8	22	35	11

Learning rate $\alpha = 0.0008$

J	w
396.663	(0.500 0.500 0.500 0.500)
38.074	(0.484 0.320 -0.008 0.139)
108.403	(0.479 0.167 -0.068 -0.033)
42.720	(0.483 0.299 -0.016 0.116)
95.886	(0.479 0.185 -0.061 -0.013)
47.188	(0.482 0.283 -0.022 0.098)
87.268	(0.480 0.198 -0.055 0.002)
51.086	(0.482 0.272 -0.027 0.085)
81.238	(0.480 0.208 -0.052 0.013)
54.312	(0.482 0.263 -0.030 0.076)
76.962	(0.480 0.215 -0.049 0.022)
56.898	(0.482 0.257 -0.033 0.068)
73.893	(0.480 0.221 -0.047 0.028)
58.930	(0.481 0.252 -0.035 0.063)
71.671	(0.481 0.225 -0.045 0.033)
60.503	(0.481 0.248 -0.036 0.059)
70.050	(0.481 0.228 -0.044 0.036)
61.711	(0.481 0.246 -0.037 0.056)
68.860	(0.481 0.231 -0.043 0.039)
62.631	(0.481 0.244 -0.038 0.053)
67.983	(0.481 0.232 -0.042 0.041)
63.328	(0.481 0.242 -0.038 0.052)
67.334	(0.481 0.234 -0.042 0.042)
63.855	(0.481 0.241 -0.039 0.050)
66.853	(0.481 0.235 -0.041 0.043)
64.252	(0.481 0.240 -0.039 0.050)
66.495	(0.481 0.235 -0.041 0.044)
64.551	(0.481 0.240 -0.039 0.049)
66.229	(0.481 0.236 -0.041 0.045)
64.775	(0.481 0.239 -0.040 0.048)

Mini-Batch Gradient Descent

- Doing somewhat in between the batch-Gradient Descent (all m training sample) and Stochastic-Gradient Descent (only one training sample)
- Let b be size of mini batch

Algorithm 15: Mini-Batch Gradient Descent

Initialize w randomly
Randomly shuffle the dataset
repeat
    for $i = 1$ *to* $m - b + 1$ *in steps of* b **do**
        Simultaneously update all w_j with
        $w_j - \alpha \frac{1}{b} \sum_{i=1}^{b} (y(x^{(i)}, w) - y^{(i)}) x_j^{(i)}$
until *converge*;
return w

Normal Equation

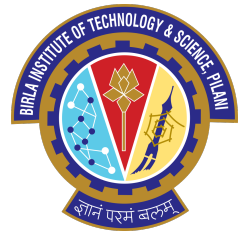
- Consider X as *design matrix*, constructed using training data of size $m \times (n+1)$
- And Y be the target values (matrix of size $m \times 1$)
- Weights could be computed directly using calculus

$$w = (X^T X)^{-1} X^T Y$$

- Computation of matrix inverse could need pseudo-inverse computation if $X^T X$ is non invertible (singular)
- This happens if features are linear combination of others, or the matrix is not square. Delete some features or apply regularization.
- No need to chose α
- Matrix computation is hard takes roughly $O(n^3)$ time. One can avoid normal equation if $n = 1000$

IS-ZC464: MACHINE LEARNING

Lecture-11: Logistic Regression, Bayesian Belief Networks



Dr. Kamlesh Tiwari
Assistant Professor
Department of Computer Science and Information Systems,
BITS Pilani, Pilani, Jhunjhunu-333031, Rajasthan, INDIA

October 06, 2018 (WILP @ BITS-Pilani Jul-Nov 2018)

Linear Regression

- Linear model for regression uses a linear combination of the input variables

$$y(x, w) = w_0 + w_1 x_1 + ... + w_n x_n$$

where x is a n dimensional vector $(x_1, x_2, ..., x_n)$. Objective is to choose w such that it makes $y(x^{(i)}, w)$ as close to $y^{(i)}$ as possible

- Finding w is similar to solving a minimization problem on a **squared error cost function** such as

$$J(w) = \frac{1}{2m} \sum_{i=1}^{m} (y(x^{(i)}, w) - y^{(i)})^2$$

where m is number of training examples. Then one have to minimize the value of $J(w)$

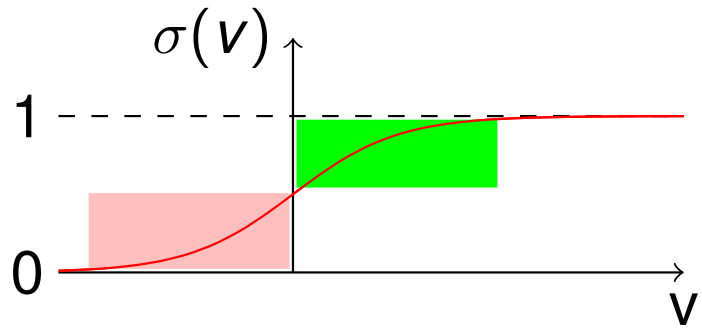
$$\underset{w}{\operatorname{argmin}}\, J(w)$$

Logistic Regression

Moving from linear regression $y(x, w) = w_0 + w_1 x_1 + ... + w_n x_n$ to **logistic regression**

$$y(x, w) = \sigma(w_0 + w_1 x_1 + ... + w_n x_n)$$

- Enables "classification" apart from the regression. Where σ is called as sigmoid function that produces values in range $[0, 1]$ and is defined as $\sigma(v) = \frac{1}{1+e^{-v}}$



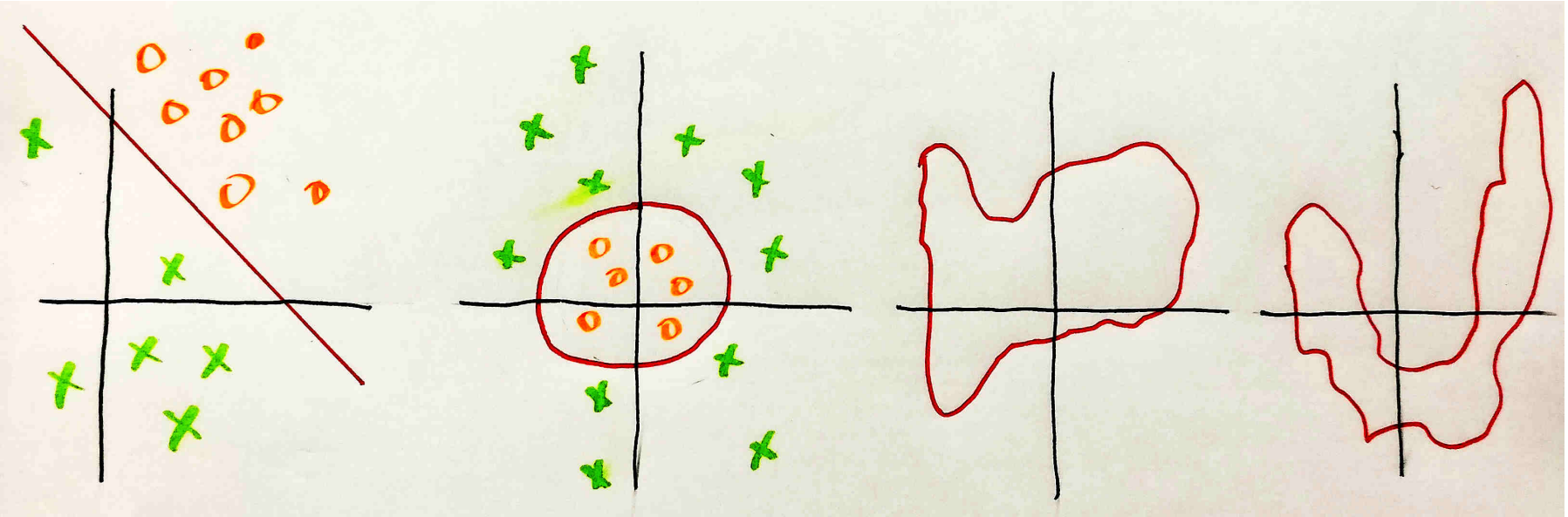
Decision on classification

$$classification = \begin{cases} 1 & \text{if } y(x, w) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Logistic Regression

$$classification = \begin{cases} 1 & \text{if } y(x, w) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

- This choice of w partitions the space into two sections and the hyper-plane separating them is called **decision boundary**
- By adding more complex or polynomial terms one can get more complex decision boundary



Cost Function

- Cost Function for liner regression

$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(y(x^{(i)}, w) - y^{(i)})^2$$

becomes a non convex function in case of logistic regression

- Therefore, a different cost function is chosen

$$J(w) = \frac{1}{m}\sum_{i=1}^{m} Cost(y(x^{(i)}, w), y^{(i)})$$

where

$$Cost(y(x^{(i)}, w), y^{(i)}) = \begin{cases} -\log(y(x^{(i)}, w)) & \text{if } y^{(i)} = 1 \\ -\log(1 - y(x^{(i)}, w)) & \text{otherwise} \end{cases}$$

- A simplified version is

$$Cost(y(x^{(i)}, w), y^{(i)}) = -y^{(i)} \log(y(x^{(i)}, w)) - (1 - y^{(i)}) \log(1 - y(x^{(i)}, w))$$

Cost Function

- Learning corresponds to the minimization of

$$\underset{w}{\operatorname{argmin}}\, J(w) = \frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}\log(y(x^{(i)}, w)) - (1 - y^{(i)})\log(1 - y(x^{(i)}, w))]$$

- Now Gradient Descent can be used

Algorithm 2: Logistic Regression

1 Initialize w randomly
2 **repeat**
3 | Simultaneously update all w_j with $w_j - \alpha \frac{\partial}{\partial w_j} J(w)$
4 **until** *converge*;
5 **return** w

The Partial Derivative

Let $v = w_0 x_0 + w_1 x_1 + \ldots + w_n x_n$
Then

$$\begin{aligned} \frac{\partial}{\partial w_j} v &= \frac{\partial}{\partial w_j}(w_0 x_0 + w_1 x_1 + \ldots + w_n x_n) \\ &= x_j \end{aligned} \tag{1}$$

$$\begin{aligned} J(w) &= \frac{1}{m}\sum_{i=1}^{m}[-y^{(i)} \log(y(x^{(i)}, w)) - (1 - y^{(i)}) \log(1 - y(x^{(i)}, w))] \\ \frac{\partial}{\partial w_j} J(w) &= \frac{1}{m}\sum_{i=1}^{m}[-\frac{\partial}{\partial w_j} y^{(i)} \log(y(x^{(i)}, w)) - \frac{\partial}{\partial w_j}(1 - y^{(i)}) \log(1 - y(x^{(i)}, w))] \\ &= \frac{1}{m}\sum_{i=1}^{m}[-A - B] \end{aligned} \tag{2}$$

The Partial Derivative

$$
\begin{aligned}
A &= \frac{\partial}{\partial w_j} y^{(i)} \log(y(x^{(i)}, w)) \\
&= y^{(i)} \times \frac{\partial}{\partial w_j} \log(y(x^{(i)}, w)) \\
&= y^{(i)} \times \frac{1}{y(x^{(i)}, w)} \times \frac{\partial}{\partial w_j} y(x^{(i)}, w) \\
&= y^{(i)} \times \frac{1}{\frac{1}{1+e^{-v}}} \times \frac{\partial}{\partial w_j} \frac{1}{1+e^{-v}} \\
&= y^{(i)} \times (1+e^{-v}) \times \frac{-1}{(1+e^{-v})^2} \times \frac{\partial}{\partial w_j} (1+e^{-v}) \\
&= \frac{-y^{(i)}}{1+e^{-v}} \times (0 + e^{-v} \times (-1) \frac{\partial}{\partial w_j} v) \\
&= y^{(i)} \times \frac{e^{-v}}{1+e^{-v}} \times x_j
\end{aligned} \tag{3}
$$

The Partial Derivative

$$
\begin{aligned}
B &= \frac{\partial}{\partial w_j}(1 - y^{(i)}) \log(1 - y(x^{(i)}, w)) \\
&= (1 - y^{(i)}) \times \frac{1}{1 - y(x^{(i)}, w)} \times \frac{\partial}{\partial w_j}(1 - y(x^{(i)}, w)) \\
&= (1 - y^{(i)}) \times \frac{-1}{1 - \frac{1}{1+e^{-v}}} \times \frac{\partial}{\partial w_j} y(x^{(i)}, w) \\
&= (1 - y^{(i)}) \times \frac{(-1)(1 + e^{-v})}{e^{-v}} \times \frac{\partial}{\partial w_j} \frac{1}{1 + e^{-v}} \\
&= (1 - y^{(i)}) \times \frac{(-1)(1 + e^{-v})}{e^{-v}} \times \frac{-1}{(1 + e^{-v})^2} \times \frac{\partial}{\partial w_j}(1 + e^{-v}) \\
&= (1 - y^{(i)}) \times \frac{(-1)(1 + e^{-v})}{e^{-v}} \times \frac{-1}{(1 + e^{-v})^2} \times (0 + e^{-v} \frac{\partial}{\partial w_j}(-v)) \\
&= (1 - y^{(i)}) \times \frac{(-1)(1 + e^{-v})}{e^{-v}} \times \frac{e^{-v}}{(1 + e^{-v})^2} \times \frac{\partial}{\partial w_j} v \\
&= (1 - y^{(i)}) \times \frac{-1}{1 + e^{-v}} \times x_j
\end{aligned}
\tag{4}
$$

The Partial Derivative

$$
\begin{aligned}
\frac{\partial}{\partial w_j} J(w) &= \frac{1}{m}\sum_{i=1}^{m}[-A - B] \\
&= \frac{1}{m}\sum_{i=1}^{m}[-y^{(i)} \times \frac{e^{-v}}{1+e^{-v}} \times x_j - (1-y^{(i)}) \times \frac{-1}{1+e^{-v}} \times x_j] \\
&= \frac{1}{m}\sum_{i=1}^{m}[(1-y^{(i)}) - y^{(i)} \times e^{-v}] \times \frac{x_j}{1+e^{-v}} \\
&= \frac{1}{m}\sum_{i=1}^{m}[1 - y^{(i)} \times (1+e^{-v})] \times \frac{x_j}{1+e^{-v}} \\
&= \frac{1}{m}\sum_{i=1}^{m}[\frac{1}{1+e^{-v}} - y^{(i)}] \times x_j \\
&= \frac{1}{m}\sum_{i=1}^{m}[y(x^{(i)}, w) - y^{(i)}] \times x_j \qquad (5)
\end{aligned}
$$

The Partial Derivative

Partial derivative term comes out to be

$$\frac{\partial}{\partial w_j} J(w) = \frac{\partial}{\partial w_j} \frac{1}{m} \sum_{i=1}^{m} [-y^{(i)} \log(y(x^{(i)}, w)) - (1 - y^{(i)}) \log(1 - y(x^{(i)}, w))]$$

where $y(x^{(i)}, w) = \frac{1}{1 + e^{-(w_0 + w_1 x_1^{(i)} + \ldots + w_n x_n^{(i)})}}$

$$\frac{\partial}{\partial w_j} J(w) = \frac{1}{m} \sum_{i=1}^{m} (y(x^{(i)}, w) - y^{(i)}) x_j^{(i)}$$

Algorithm 7: Logistic Regression

1. Initialize w randomly
2. **repeat**
3. Simultaneously update all w_j with $w_j - \alpha \times \frac{1}{m} \sum_{i=1}^{m} (y(x^{(i)}, w) - y^{(i)}) x_j^{(i)}$
4. **until** *converge*;
5. **return** w

It looks identical to liner regression but $y(x^{(i)}, w)$ is different

Example: Logistic Regression

Consider following data

	x_1	x_2	x_3	*Class*
1	2	2	2	1
2	3	2	2	1
3	2	3	2	1
4	2	2	3	1
5	7	6	9	0
6	9	7	6	0
7	9	6	7	0
8	6	8	9	0
9	8	9	6	0
10	8	9	9	0

Learning rate $\alpha = 0.01$

J	w
6.912	(0.500 0.500 0.500 0.500)
6.496	(0.494 0.453 0.455 0.454)
5.944	(0.488 0.406 0.410 0.408)
5.316	(0.482 0.360 0.366 0.363)
4.692	(0.477 0.313 0.321 0.317)
4.072	(0.471 0.267 0.277 0.272)
3.460	(0.465 0.221 0.233 0.227)
2.860	(0.460 0.175 0.189 0.182)
2.279	(0.454 0.130 0.146 0.138)
1.735	(0.449 0.086 0.104 0.095)
1.262	(0.445 0.044 0.064 0.054)
0.906	(0.441 0.008 0.029 0.018)
0.685	(0.438 -0.022 0.000 -0.011)
0.566	(0.437 -0.044 -0.020 -0.032)
0.504	(0.436 -0.060 -0.035 -0.048)
0.470	(0.436 -0.072 -0.046 -0.059)
0.451	(0.436 -0.081 -0.055 -0.068)
0.438	(0.436 -0.088 -0.061 -0.074)
0.431	(0.437 -0.093 -0.066 -0.080)
0.425	(0.438 -0.098 -0.070 -0.084)
0.422	(0.439 -0.101 -0.074 -0.088)
0.419	(0.440 -0.105 -0.077 -0.091)
0.417	(0.441 -0.107 -0.079 -0.093)
0.416	(0.443 -0.110 -0.081 -0.095)
0.415	(0.444 -0.112 -0.082 -0.097) **Iteration 25**
0.412	(0.451 -0.119 -0.088 -0.103) **Iteration 30**
0.348	(0.857 -0.179 -0.084 -0.132) **Iteration 300**
0.116	(3.256 -0.409 -0.135 -0.291) **Iteration 3000**
0.012	(7.596 -0.748 -0.361 -0.588) **Iteration 30000**
0.001	(11.975 -1.091 -0.599 -0.896) **Iteration 300000**

Bayesian Learning

Address “most probable classification of new instance” instead of “best hypothesis”. Searching possibility to do better then MAP

- **Bayes optimal classification:** $\text{argmax}_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$ Outperforms on an average but, quite costly to apply
- **GIBBS Algorithm:** Choose a hypothesis $h \in H$ at random, according to the posterior probability distribution over H. (Expected misclassification error is bounded to twice of the Bayes optimal classifier)
- **Naive Bayes Classifier:** $\text{argmax}_{v_j \in V} P(a_1, a_2, ..., a_n|v_j)P(v_j)$

$$\underset{v_j \in V}{\text{argmax}}\, P(v_j)\Pi_i P(a_i|v_j)$$

Highly practical method

Probability

- $P(x, y) = P(x) \times P(y|x)$
- **Independence** of x and y implies $P(y|x) = P(y)$
- Then $P(x, y) = P(x) \times P(y)$
- Bayes Rule

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

- **Marginal:** distribution of a single variable x can be obtained from a given joint distribution $p(x, y)$ by

$$p(x) = \sum_{y} p(x, y)$$

- The process of computing a marginal from a joint distribution is called marginalisation.

$$p(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) = \sum_{x_i} p(x_1, x_2, ..., x_n)$$

Marginalisation

- **Conditional Independence** when two variable are independent of each other provided we know state of some other variable

$$P(x, y|z) = P(x|z) \times P(y|z)$$

Consider: Soft XOR

A	B	$p(C{=}1|A, B)$
0	0	0.10
0	1	0.99
1	0	0.80
1	1	0.25

If $p(A{=}1) = 0.65$, $p(B{=}1) = 0.77$
Determine $p(A{=}1|C{=}0)$

- $p(A{=}1, C{=}0)$

$$\sum_B p(A = 1, B, C = 0) = 0.405$$

- $p(A{=}0, C{=}0) = 0.075$

- $p(A{=}1|C{=}0) =$

$$\frac{p(A = 1, C = 0)}{p(A = 1, C = 0) + p(A = 0, C = 0)}$$

$$= 0.843$$

Let's see this

$$\sum_j \big(p(J|R) \times f(R)\big) = f(R)$$

Proof:

$$\begin{aligned}
\sum_j \big(p(J|R) \times f(R)\big) &= \sum_j \big(\frac{p(J,R)}{p(R)} \times f(R)\big) \\
&= \frac{\sum_j \big(p(J,R) \times f(R)\big)}{p(R)} \\
&= \frac{f(R) \times \sum_j p(J,R)}{p(R)} \\
&= \frac{f(R) \times p(R)}{p(R)} \\
&= f(R)
\end{aligned}$$

Belief network (a graphical model)

- A **belief network** introduces structure into a probabilistic model by using graphs to represent independence assumptions among the variables
- Independently specifying all the attributed is overkill
- With distribution of n attributes, marginal for one takes $O(2^{n-1})$
- By constraining variable interaction (specifying independence) one can get the form like

$$p(x_1, x_2, ..., x_{100}) = \Pi_{i=1}^{99} \phi(x_i, x_{i+1})$$

- Belief networks are a convenient framework for representing such independence assumptions
- Belief networks are also called as *Bayes' Networks* or *Bayesian Belief Networks*

Modeling Independencies

One morning Tracey leaves her house and realities that her grass is wet. Is it due to overnight rain or did she forget to turn off the sprinkler last night? Next she notices that the grass of her neighbor, Jack, is also wet.

(R=1) $\rightarrow$ rain last night,
(S=1) $\rightarrow$ sprinkler on last night,
(J=1) $\rightarrow$ Jack's grass is wet,
(T=1) $\rightarrow$ Traceya's Grass is wet

- Model of Traceya's world involves probability distribution on T, J, R, S that has $2^4 = 16$ states

Modeling Independencies (contd..)

- However we know

$$\begin{aligned} p(T,J,R,S) &= p(T|J,R,S)p(J,R,S) \\ &= p(T|J,R,S)p(J|R,S)p(R,S) \\ &= p(T|J,R,S)p(J|R,S)p(R|S)p(S) \end{aligned}$$

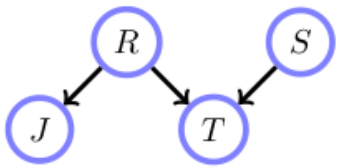
- Computation of $p(T|J,R,S)$ requires us to specify $2^3 = 8$ values
- With $p(T=1|J,R,S)$, one can use normalization to compute $p(T=0|J,R,S)$ as $1 - p(T=1|J,R,S)$
- Computation of other factors would also need 4+2+1 values

Conditional Independence

- We may assume that Traceya's grass is wet depends only directly on whether or not it has been raining and whether or not her sprinkler was on so $p(T|J,R,S) = p(T|R,S)$
- Assume that Jack's grass is wet is influenced only directly by whether or not it has been raining $p(J|R,S) = p(J|R)$
- Furthermore, we assume the rain is not directly influenced by the sprinkler $p(R|S) = p(R)$
- Therefore, our model becomes

$$\begin{aligned} p(T,J,R,S) &= p(T|J,R,S)p(J|R,S)p(R|S)p(S) \\ &= p(T|R,S)p(J|R)p(R)p(S) \end{aligned}$$

- Number of values we need to specify is 4+2+1+1=8
- We can represent these conditional independencies as



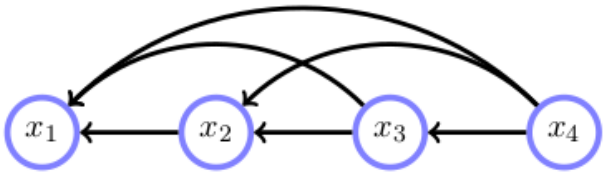
Belief network

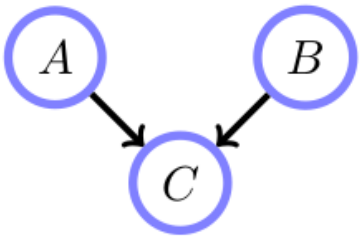
How to represent these conditional independencies?

- **Belief network** is a distribution of the form

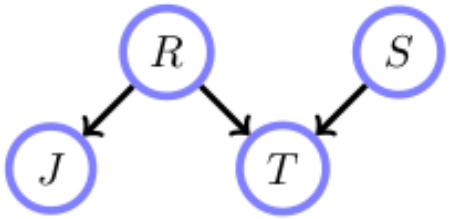
$$p(x_1, x_2, ..., x_n) = \Pi_{i=1}^{n} p(x_i|pa(x_i))$$

  where $pa(x_i)$ represent the parental variables of variable x_i
- Represented as a directed graph, with an arrow pointing from a parent variable to child variable, a belief network corresponds to a Directed Acyclic Graph (DAG)





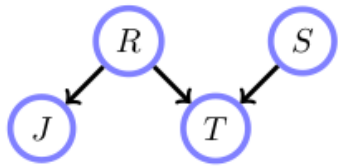
$$p(A, B, C) = p(C|A, B)p(A)p(B)$$



Example

One morning Tracey realises that her grass is wet and the grass of her neighbour, Jack, is also wet. Let the prior probabilities be $p(R{=}1) = 0.2$ and $p(S{=}1) = 0.1$. We set $p(J{=}1|R{=}1) = 1$, $p(J{=}1|R{=}0) = 0.2$, $p(T{=}1|R{=}1, S{=}0) = 1$, $p(T{=}1|R{=}1, S{=}1) = 1$, $p(T{=}1|R{=}0, S{=}1) = 0.9$, $p(T{=}1|R{=}0, S{=}0) = 0$

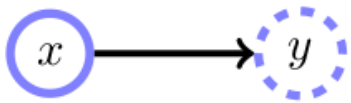
Using following Beleif Network; calculate



1. Probability that the sprinkler was *on* overnight, given that Traceya's grass is wet. $p(S{=}1|T{=}1) = 0.3382$
2. Probability that Traceya's sprinkler was *on* overnight, given that her grass is wet and that Jack's grass is also wet. $p(S{=}1|T{=}1, J{=}1) = 0.1604$

Uncertain evidence

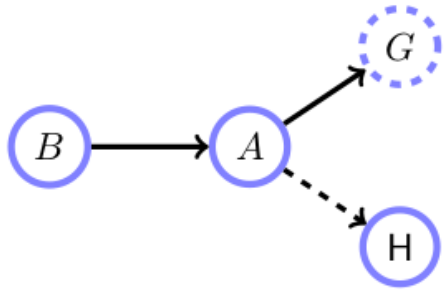
- **Soft or uncertain evidence**, let $dom(x) = \{red, blue, green\}$ and the vector $\tilde{y} = (0.6, 0.1, 0.3)$ represents the belief in the respective states. **Hard evidence** are like (0, 0, 1).
- Assumption is that $p(x|y, \tilde{y}) = p(x|y)$
- $p(x|\tilde{y}) = \sum_y p(x, y|\tilde{y}) = \sum_y p(x|y, \tilde{y})p(y|\tilde{y}) = \sum_y p(x|y)p(y|\tilde{y})$ where $p(y = i|\tilde{y})$ represents the probability that y is in state i
- Dashed circle is used to represent a variable in soft-evidence state



- **Example:** Let probability of Fire, when there is a fire alarm is 0.9. Monu said he is 70% sure that he had heard a fire alarm. What is probability of fire.
  $p(F = 1|\tilde{A}) = \sum_A p(F = 1|A)p(A|\tilde{A}) = p(F = 1|A = 0)p(A = 0|\tilde{A}) + p(F = 1|A = 1)p(A = 1|\tilde{A}) = 0.1 \times 0.3 + 0.9 \times 0.7 = 0.66$

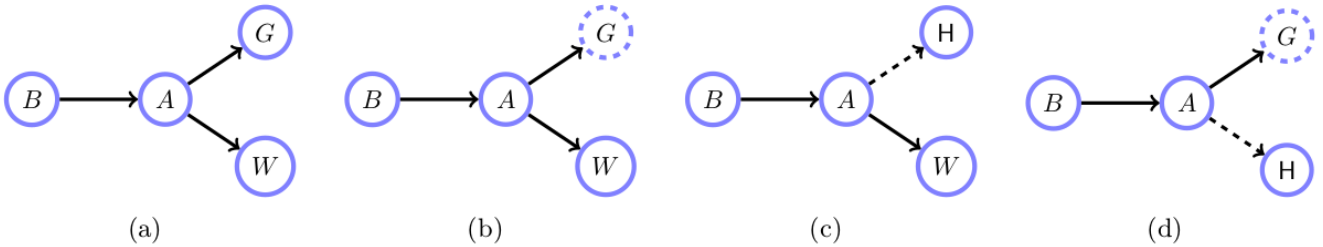
Unreliable Evidence

- **Unreliable:** (continued from previous example) I asked Raju about the alarm. It is believed that if alarm had sound, there is 80% chance that Raju would tell it sound. If alarm had NOT sounded, there is 70% chance that he would tell NOT sound.
- Unreliable evidences are modeled using dashed lines as below



Example

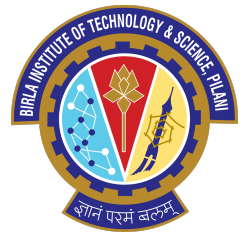
- Let B=1, means Holmes house has been burgled. A=1, means alarm went off. W=1, means Watson heard alarm. G=1, means Gibbon heard alarm.



(a) BN for the environment, (b) If Gibbon is a little def and is only 80% sure about the alarm sound being heard, (c) Replacement of evidence, (d) Holmes feels Watson's observation is unreliable

IS-ZC464: MACHINE LEARNING

Lecture-12: SVM, VC-Dimension, MC-Simulation



Dr. Kamlesh Tiwari
Assistant Professor
Department of Computer Science and Information Systems,
BITS Pilani, Pilani, Jhunjhunu-333031, Rajasthan, INDIA

October 18, 2018 (WILP @ BITS-Pilani Jul-Nov 2018)

Recap: Regression

Regression predicts value of continuous a target variable

x_1	x_2	x_3	y
10	50	20	10
11	31	22	12
11	12	15	4
20	55	20	22
23	41	27	1
31	12	35	9
13	18	12	23
21	55	16	16
32	56	27	22
8	22	35	??

What should come at the place of ??

Linear model uses a linear combination of the input variables for regression

$$y(x, w) = w_0 + w_1 x_1 + ... + w_n x_n$$

Objective is to choose w's that makes $y(x^{(i)}, w)$ as close to $y^{(i)}$ as possible. Similar to a minimizing

$$J(w) = \frac{1}{2m} \sum_{i=1}^{m} (y(x^{(i)}, w) - y^{(i)})^2$$

Recap: Linear Regression

x_1	x_2	x_3	y	$y(x^{(i)}, w)$	$(y(x^{(i)}, w) - y)^2$
10	50	20	10	8	4
11	31	22	12	9	9
11	12	15	4	3	1
20	55	20	22	26	16
23	41	27	1	1	0
31	12	35	9	4	25
13	18	12	23	30	49
21	55	16	16	13	9
32	56	27	22	21	1

Assume for some w we computed $y(x^{(i)}, w)$ then

$$J(w) = \frac{1}{2 \times 9} \times 114$$
$$= 6.33$$

Minimize the value of $J(w)$ using suitable w's $\text{argmin}_w J(w)$

Algorithm 1: Gradient Descent

1 Initialize w randomly
2 **repeat**
3 | Simultaneously update all w_j with $w_j - \alpha \frac{\partial}{\partial w_j} J(w)$
4 **until** *converge*;
5 **return** w

Recap: Similar Mechanism for Classification

Classification assigns a label (from a predefined fixed number set)

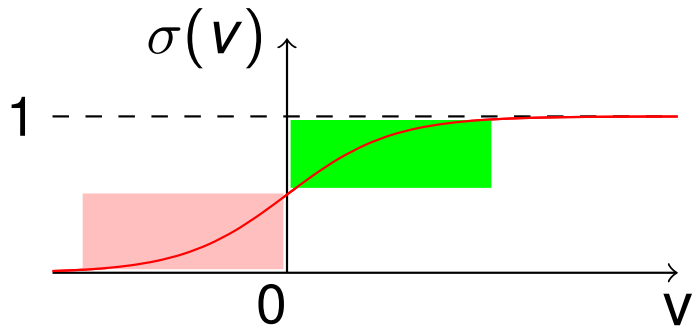
x_1	x_2	x_3	*Class*
10	50	20	1
11	31	22	1
11	12	15	0
20	55	20	0
23	41	27	0
31	12	35	1
13	18	12	0
21	55	16	1
32	56	27	0
8	22	35	??

What should come at the place of ??

Regression would use

$$y(x, w) = \sigma(w_0 + w_1 x_1 + \ldots + w_n x_n)$$

where sigmoid function $\sigma(v) = \frac{1}{1+e^{-v}}$ produces values in range $[0, 1]$



choice of w gives **decision boundary**

Recap: Cost Function

Cost function is

$$J(w) = \frac{1}{m}\sum_{i=1}^{m} Cost(y(x^{(i)}, w), y^{(i)})$$

where

$$Cost(y(x^{(i)}, w), y^{(i)}) = -y^{(i)} \log(y(x^{(i)}, w)) - (1 - y^{(i)}) \log(1 - y(x^{(i)}, w))$$

Algorithm 3: Logistic Regression

1. Initialize w randomly
2. **repeat**
3. Simultaneously update all w_j with $w_j - \alpha \times \frac{1}{m}\sum_{i=1}^{m}(y(x^{(i)}, w) - y^{(i)})x_j^{(i)}$
4. **until** *converge*;
5. **return** w

It looks identical to liner regression but, $y(x^{(i)}, w)$ is different

Example: Logistic Regression

Consider following data

	x_1	x_2	x_3	*Class*
1	2	2	2	1
2	3	2	2	1
3	2	3	2	1
4	2	2	3	1
5	7	6	9	0
6	9	7	6	0
7	9	6	7	0
8	6	8	9	0
9	8	9	6	0
10	8	9	9	0

Learning rate $\alpha = 0.01$

J	w
6.912	(0.500 0.500 0.500 0.500)
6.496	(0.494 0.453 0.455 0.454)
5.944	(0.488 0.406 0.410 0.408)
5.316	(0.482 0.360 0.366 0.363)
4.692	(0.477 0.313 0.321 0.317)
4.072	(0.471 0.267 0.277 0.272)
3.460	(0.465 0.221 0.233 0.227)
2.860	(0.460 0.175 0.189 0.182)
2.279	(0.454 0.130 0.146 0.138)
1.735	(0.449 0.086 0.104 0.095)
1.262	(0.445 0.044 0.064 0.054)
0.906	(0.441 0.008 0.029 0.018)
0.685	(0.438 -0.022 0.000 -0.011)
0.566	(0.437 -0.044 -0.020 -0.032)
0.504	(0.436 -0.060 -0.035 -0.048)
0.470	(0.436 -0.072 -0.046 -0.059)
0.451	(0.436 -0.081 -0.055 -0.068)
0.438	(0.436 -0.088 -0.061 -0.074)
0.431	(0.437 -0.093 -0.066 -0.080)
0.425	(0.438 -0.098 -0.070 -0.084)
0.422	(0.439 -0.101 -0.074 -0.088)
0.419	(0.440 -0.105 -0.077 -0.091)
0.417	(0.441 -0.107 -0.079 -0.093)
0.416	(0.443 -0.110 -0.081 -0.095)
0.415	(0.444 -0.112 -0.082 -0.097) **Iteration 25**
0.412	(0.451 -0.119 -0.088 -0.103) **Iteration 30**
0.348	(0.857 -0.179 -0.084 -0.132) **Iteration 300**
0.116	(3.256 -0.409 -0.135 -0.291) **Iteration 3000**
0.012	(7.596 -0.748 -0.361 -0.588) **Iteration 30000**
0.001	(11.975 -1.091 -0.599 -0.896) **Iteration 300000**

Example: Find J

As (w_0, w_1, w_2, w_3)=$(0.5, 0.5, 0.5, 0.5)$, $v = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$
$y(x^{(i)}, w) = \sigma(v)$
And log term is $-y^{(i)} \log(y(x^{(i)}, w)) - (1 - y^{(i)}) \log(1 - y(x^{(i)}, w))$

i	x_1	x_2	x_3	$y^{(i)}$	v	$y(x^{(i)}, w)$	log term
1	2	2	2	1	3.5	0.970	0.029
2	3	2	2	1	4.0	0.982	0.018
3	2	3	2	1	4.0	0.982	0.018
4	2	2	3	1	4.0	0.982	0.018
5	7	6	9	0	11.5	0.999	11.49
6	9	7	6	0	11.5	0.999	11.49
7	9	6	7	0	11.5	0.999	11.49
8	6	8	9	0	12	0.999	11.51
9	8	9	6	0	12	0.999	11.51
10	8	9	9	0	13	0.999	11.51
						Total/10:	6.9118

Example: Find next W

Let (w_0, w_1, w_2, w_3)=(0.5, 0.5, 0.5, 0.5) and $t_i = (y(x^{(i)}, w) - y^{(i)})x_j^{(i)}$

Then $\frac{1}{m}\sum_{i=1}^{m}(y(x^{(i)}, w) - y^{(i)})x_j^{(i)} = \frac{1}{m}\sum_{i=1}^{m} t_i$ let $\hat{y}^{(i)} = y(x^{(i)}, w)$

Then update w_j with $w_j - \alpha \times \frac{1}{m}\sum_{i=1}^{m} t_i$ we have set $\alpha = 0.01$

i	x_0	x_1	x_2	x_3	$y^{(i)}$	$\hat{y}^{(i)}$	t_0	t_1	t_2	t_3
1	1	2	2	2	1	0.970	-0.029	-0.058	-0.058	-0.058
2	1	3	2	2	1	0.982	-0.017	-0.053	-0.035	-0.035
3	1	2	3	2	1	0.982	-0.017	-0.035	-0.053	-0.035
4	1	2	2	3	1	0.982	-0.017	-0.035	-0.035	-0.053
5	1	7	6	9	0	0.999	0.999	6.999	5.999	8.999
6	1	9	7	6	0	0.999	0.999	8.999	6.999	5.999
7	1	9	6	7	0	0.999	0.999	8.999	5.999	6.999
8	1	6	8	9	0	0.999	0.999	5.999	7.999	8.999
9	1	8	9	6	0	0.999	0.999	7.999	8.999	5.999
10	1	8	9	9	0	0.999	0.999	7.999	8.999	8.999
						Total	5.916	46.815	44.815	45.815
						$w_i - \alpha \times (total/m)$	0.494	0.453	0.455	0.454

Example: Classification across Iterations

Following table shows classification as the weights get modified along 1^{st}, 100^{th}, 300^{th} and 500^{th} iteration

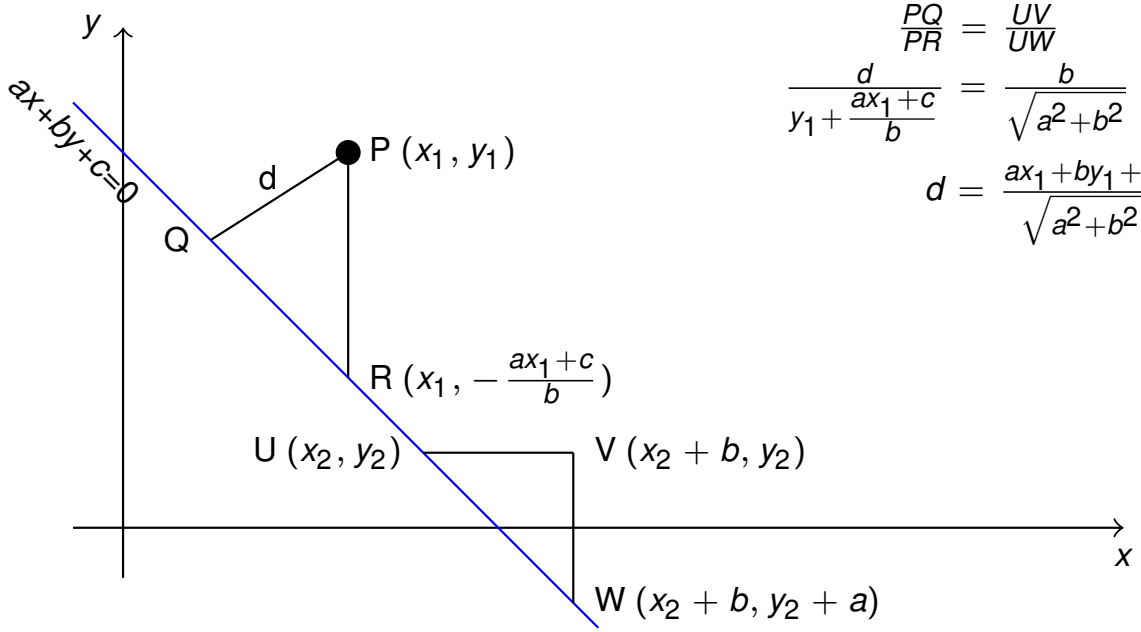
i	x_1	x_2	x_3	$y^{(i)}$	1	100	300	500
1	2	2	2	1	1	0	1	1
2	3	2	2	1	1	0	0	1
3	2	3	2	1	1	0	1	1
4	2	2	3	1	1	0	1	1
5	7	6	9	0	1	0	0	0
6	9	7	6	0	1	0	0	0
7	9	6	7	0	1	0	0	0
8	6	8	9	0	1	0	0	0
9	8	9	6	0	1	0	0	0
10	8	9	9	0	1	0	0	0

Geometry

- Consider a line $ax + by + c = 0$ and determine the length of perpendicular drawn on it from a point (x_1, y_1)

$$d = \frac{|ax_1 + by_1 + c|}{\sqrt{a^2 + b^2}}$$

- How? Proof?



$$\frac{PQ}{PR} = \frac{UV}{UW}$$

$$\frac{d}{y_1 + \frac{ax_1+c}{b}} = \frac{b}{\sqrt{a^2+b^2}}$$

$$d = \frac{ax_1+by_1+c}{\sqrt{a^2+b^2}}$$

Geometry

- So the length of a perpendicular on $ax + by + c = 0$ from (x_1, y_1) is

$$d = \frac{|ax_1 + by_1 + c|}{\sqrt{a^2 + b^2}}$$

- Similarly: distance of a point $X = (x_1, x_2, ..., x_n)$ from a hyperplane $(b, w_1, w_2, ..., w_n)$ is given by

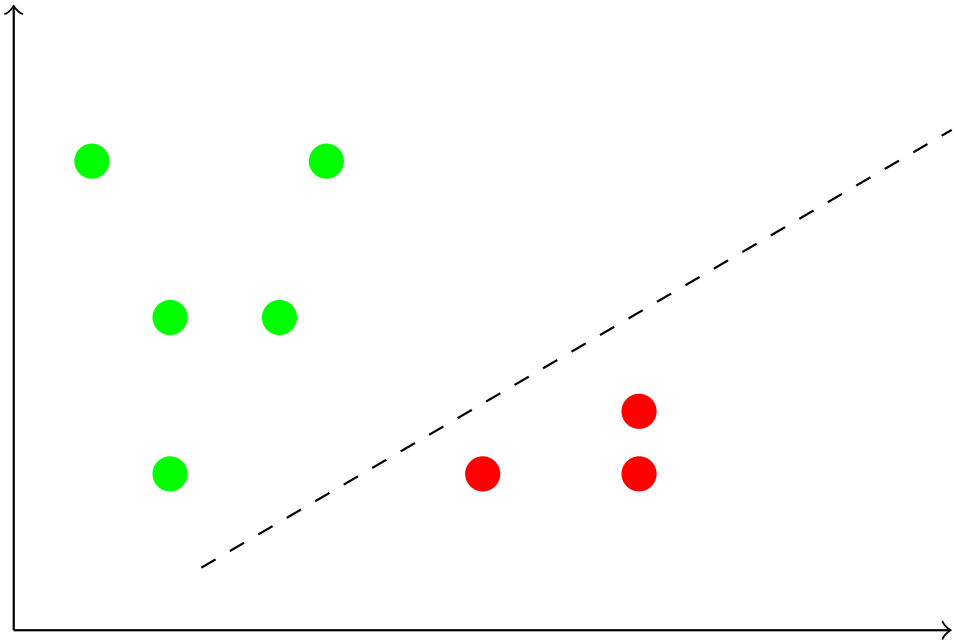
$$\frac{W^T X + b}{||W||}$$

where $||W||$ is norm [1]

[1] $||W|| = \sqrt{w_1^2 + w_2^2 + ... + w_n^2}$

Special Case of Classification

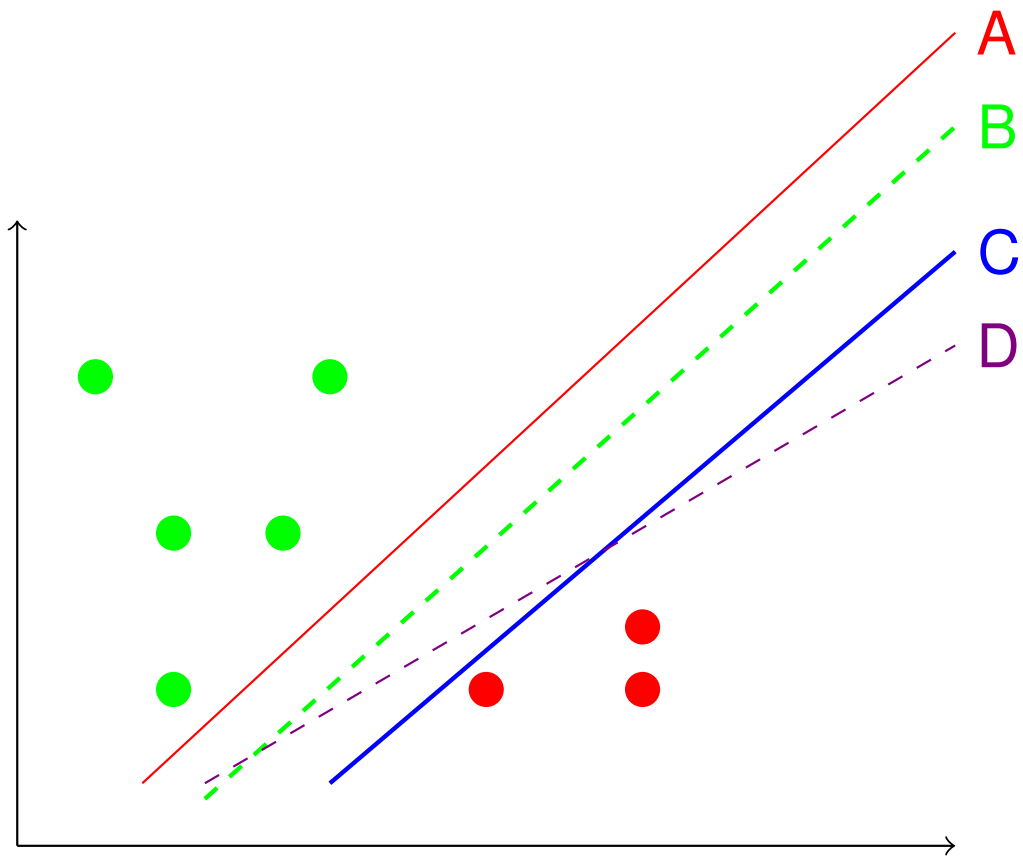
- Consider the case when dataset is linearly separable



- Most of the real-world problems are not linearly separable
- But, some time we can transform the data to a high-dimensional feature space where classes are linearly separable
- Caution: this could lead to over-fitting

Which decision boundary is better

Multiple decision boundaries are possible. Which one is better?

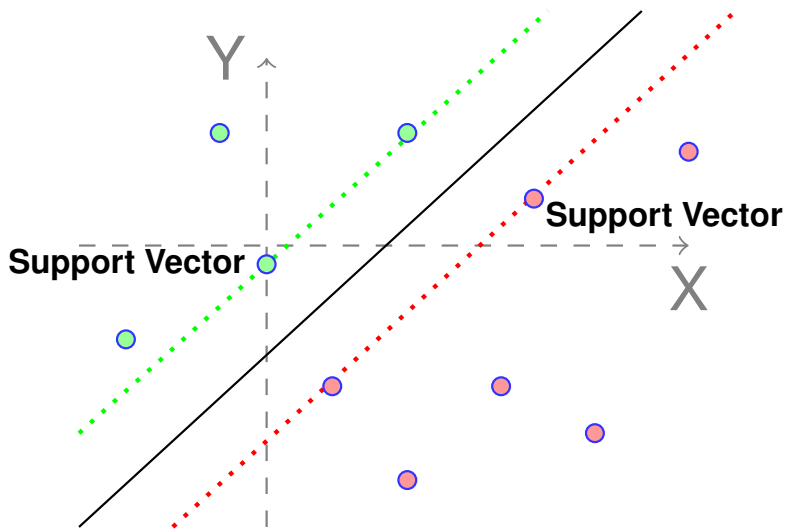


- Whether fat margin is better?

Support Vector Machine (SVM)

- SVM is a liner decision machine (uses $w^T x^{(i)} + b \geq 0$)
- We want $w^T x^{(i)} + b \geq \gamma$ for $+$ve (and $< -\gamma$ for $-$ve)
- Distance of a point (x, y) from hyper plane $w^T x + b = 0$ is $\frac{w^T x + b}{||w||}$
- Distance can be maximized, by either **maximizing** b or by **minimizing** $||w||$

- We needs $w^T x + b \geq \gamma ||w||$
  let $\gamma ||w|| = 1$
  - $w^T x + b \geq 1$ if x is $+$ve
  - $w^T x + b \leq -1$ if x is $-$ve
- It gives us $y^{(i)}(w^T x^{(i)} + b) \geq 1$;
- Points having $y^{(i)}(w^T x^{(i)} + b) = 1$ they are called **support vector**



Support Vector Machine (SVM)

- Minimization of w is same as minimization of $\Phi(w) = \frac{1}{2} w.w$
- Other constraints are $y^{(i)}(w^T x^{(i)} + b) \geq 1$
- Define a Lagrangian Multiplier
  $L(w, b) = \frac{1}{2} w.w - \sum \alpha_i (y^{(i)}(w^T x^{(i)} + b) - 1)$
- Derivative $\frac{\partial L}{\partial b} = -\sum \alpha_i y^{(i)}$ that should be equated to zero

$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

- Derivative $\frac{\partial L}{\partial w} = w - \sum \alpha_i y^{(i)} x^{(i)}$ equated to zero gives

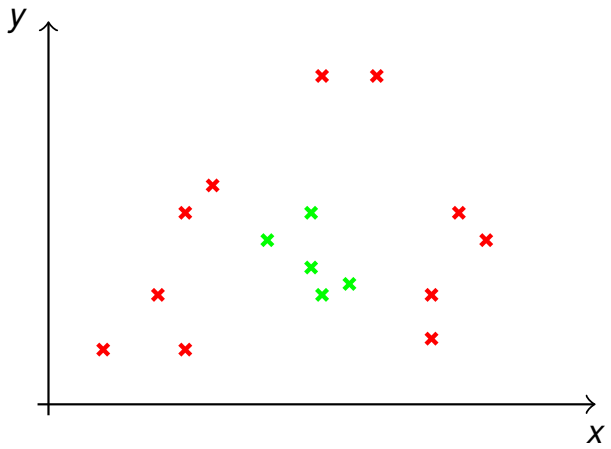
$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$$

Lagrangian with optimized values

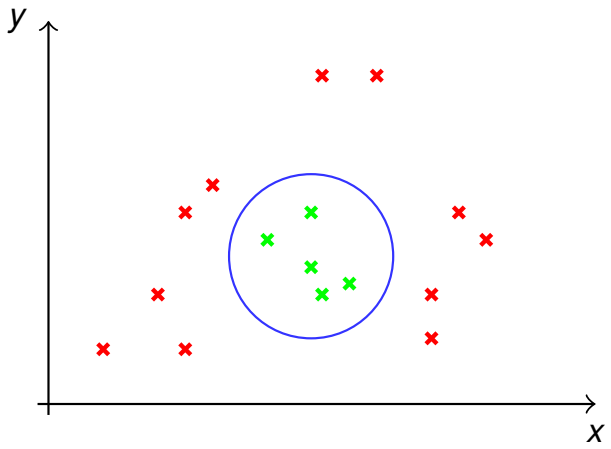
$$L(w,b) = \frac{1}{2} w.w - \sum \alpha_i (y^{(i)}(wx^{(i)} + b) - 1)$$
$$L(w,b) = \frac{1}{2} w.w - \sum \alpha_i y^{(i)} b - \sum \alpha_i y^{(i)} wx^{(i)} + \sum \alpha_i$$
$$L(w,b) = \frac{1}{2} w.w - \sum \alpha_i y^{(i)} wx^{(i)} + \sum \alpha_i$$
$$L(w,b) = \frac{1}{2} \sum \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)} x^{(j)} - \sum \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)} x^{(j)} + \sum \alpha_i$$
$$L(w,b) = \sum \alpha_i - \frac{1}{2} \sum \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)} x^{(j)}$$

- One have to maximize $L(w,b)$ subject to $\alpha_i \geq 0$ and $\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$
- If α_i is large the corresponding training sample is support vector. Otherwise, when $\alpha_i = 0$ it is not a support vector
- Optimized α_i provides the value of $w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$ to be used in linear decision boundary

Transformation: An Example



- Transform all 2D points $(x^{(i)}, y^{(i)})$ in 3D as $(x^{(i)}, y^{(i)}, z)$ where $z = (x^{(i)} - x_c)^2 + (y^{(i)} - y_c)^2$



Support Vector Machine (SVM)

Advantages

- It works well with clear margin of separation
- Do not stuck to local minima
- Effective in high dimensional spaces
- It is effective in cases where number of dimensions is greater than the number of samples
- Memory efficient as it uses a subset of training points in decision (called support vectors)

Disadvantages

- It doesn't perform well, when data set is large as the training time is higher
- Doesn't perform well when target classes are overlapping
- Does not directly provide probability estimates
- Higher classification confidence for points lying far from the decision boundary

Vapnik-Chervonenkis dimension

Measures the power of learner.

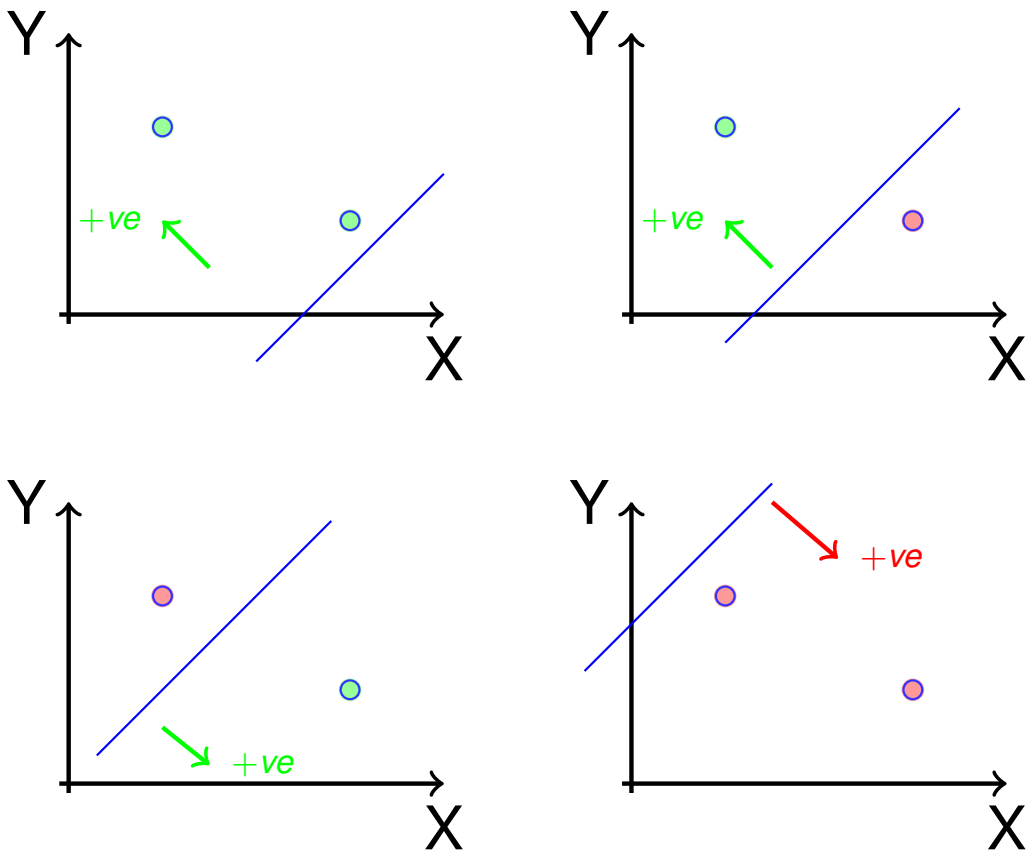
- The Vapnik-Chervonenkis dimension, $VC(H)$, of hypothesis space H defined over instance space X is the size of the largest finite subset of X shattered by H
- **Shattering:** a model can shatter m points $x^{(1)}, x^{(2)}, ..., x^{(m)}$ if and only if for all possible training set of $(x^{(1)}, y^{(1)})$, $(x^{(2)}, y^{(2)})$, ..., $(x^{(m)}, y^{(m)})$ there exists some parameter w such that model f_w achieves zero training error.
- Shattering can be considered as a two player game
  - Player-1 proposed d points (wish to select maximum value for d)
  - Player-2 assigns them labels
  - Player-1 provides parameters to the learner
- Consider the model
  $f_w(x) = sign(w^T x) = sign(w_0 x_0 + w_1 x_1 + w_2 x_2 + ...)$
- Consider the model $f_w(x) = sign(x^T x - w) = sign(\sum x_i^2 - w)$

Consider $f_w(x) = sign(w^T x)$

Consider the model

$f_w(x) = sign(w^T x) = sign(w_0 x_0 + w_1 x_1 + w_2 x_2 + ...)$

Shattering of two points

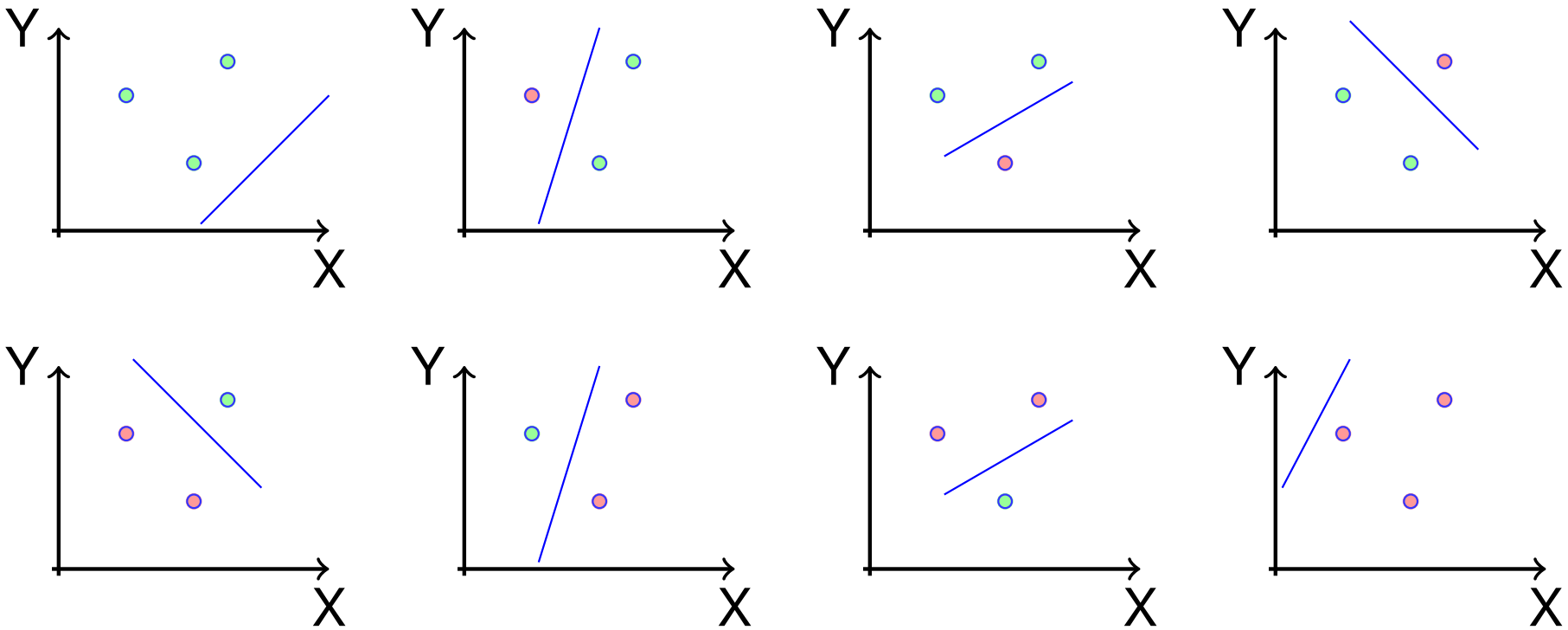


Consider $f_w(x) = sign(w^T x)$

Consider the model

$f_w(x) = sign(w^T x) = sign(w_0 x_0 + w_1 x_1 + w_2 x_2 + ...)$

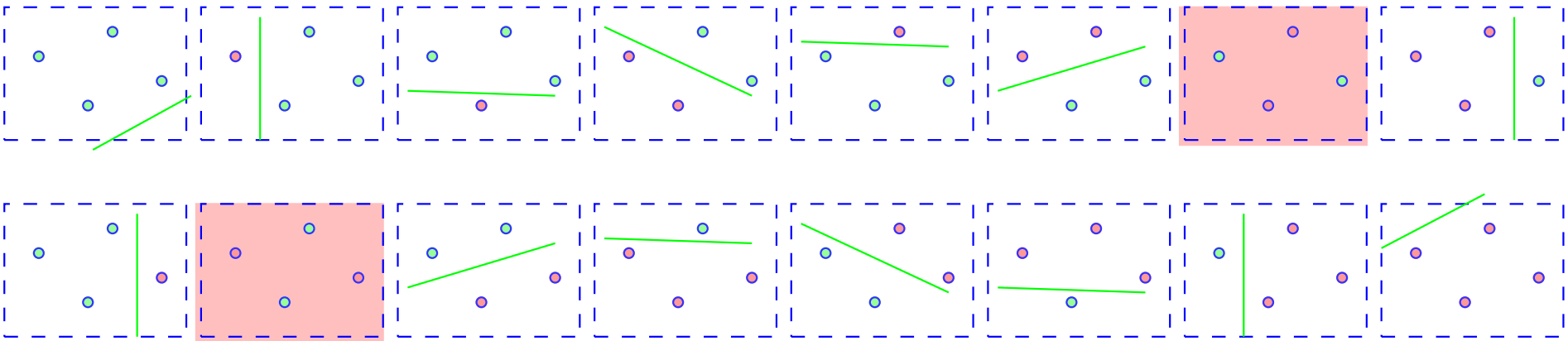
Shattering of three points



Consider $f_w(x) = sign(w^T x)$

Consider the model
$f_w(x) = sign(w^T x) = sign(w_0 x_0 + w_1 x_1 + w_2 x_2 + ...)$

Shattering of four points

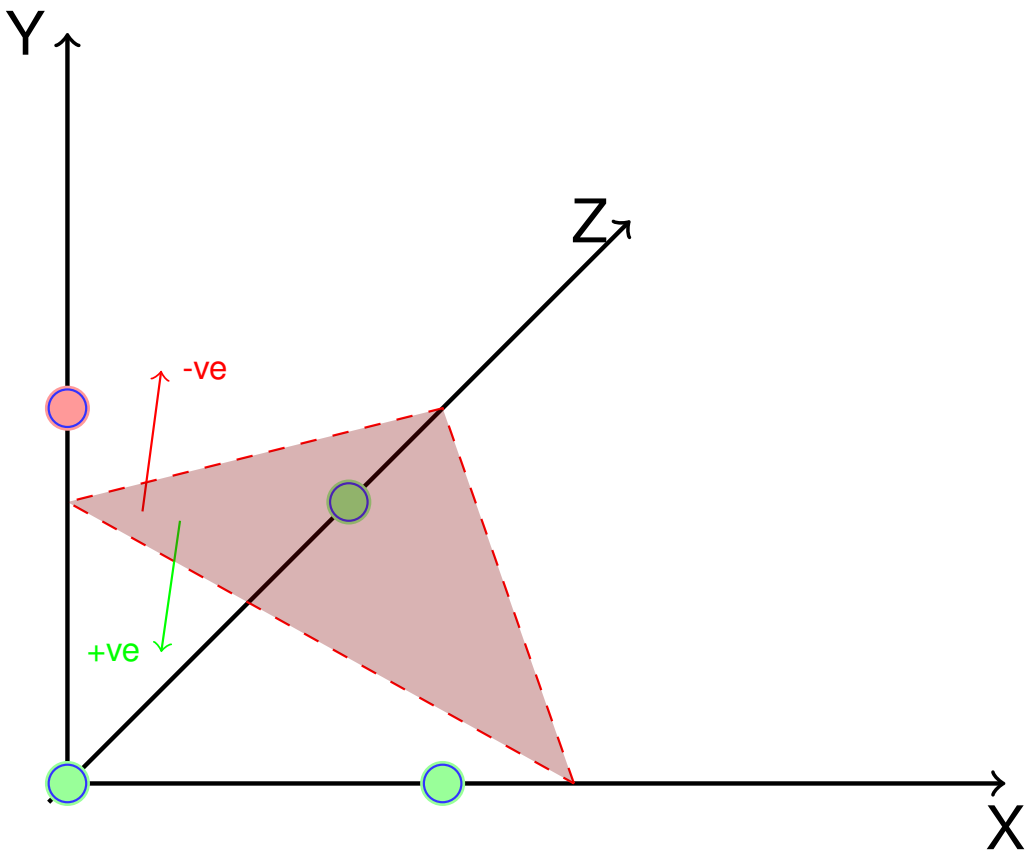


- There are few assignments for which it is not possible to have parameters so four points can not be shattered.

VC dimension of Liner classifier in d dimension

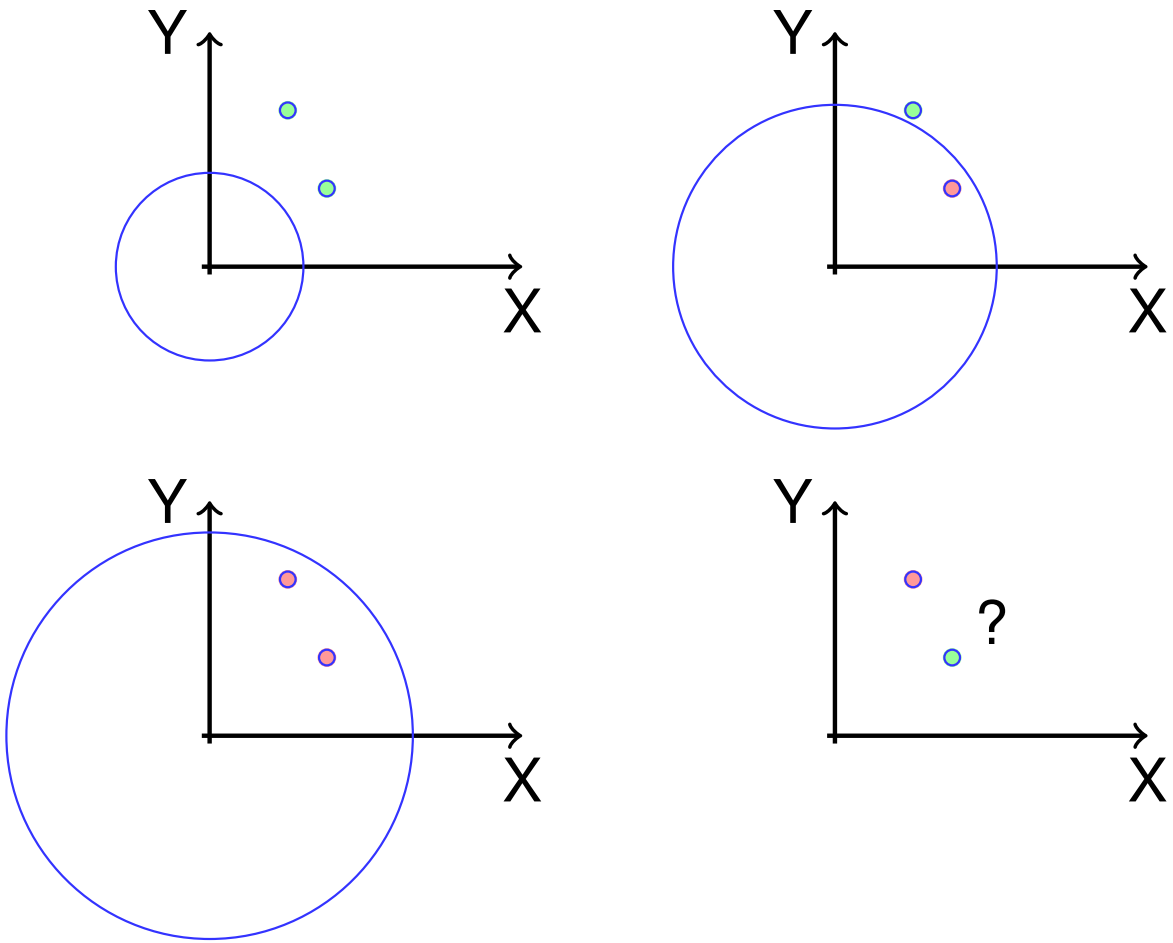
Liner classifier in d dimension have VC dimension at least $d + 1$

- d points are placed at axes and one at origin
- If label of point at origin and axes differ pass the plane through middle
- Otherwise through away from the axis point
- Shattering of d+1 point is always possible



Consider $f_w(x) = sign(x^T x - w)$

Consider the model $f_w(x) = sign(x^T x - w) = sign(\sum x_i^2 - w)$



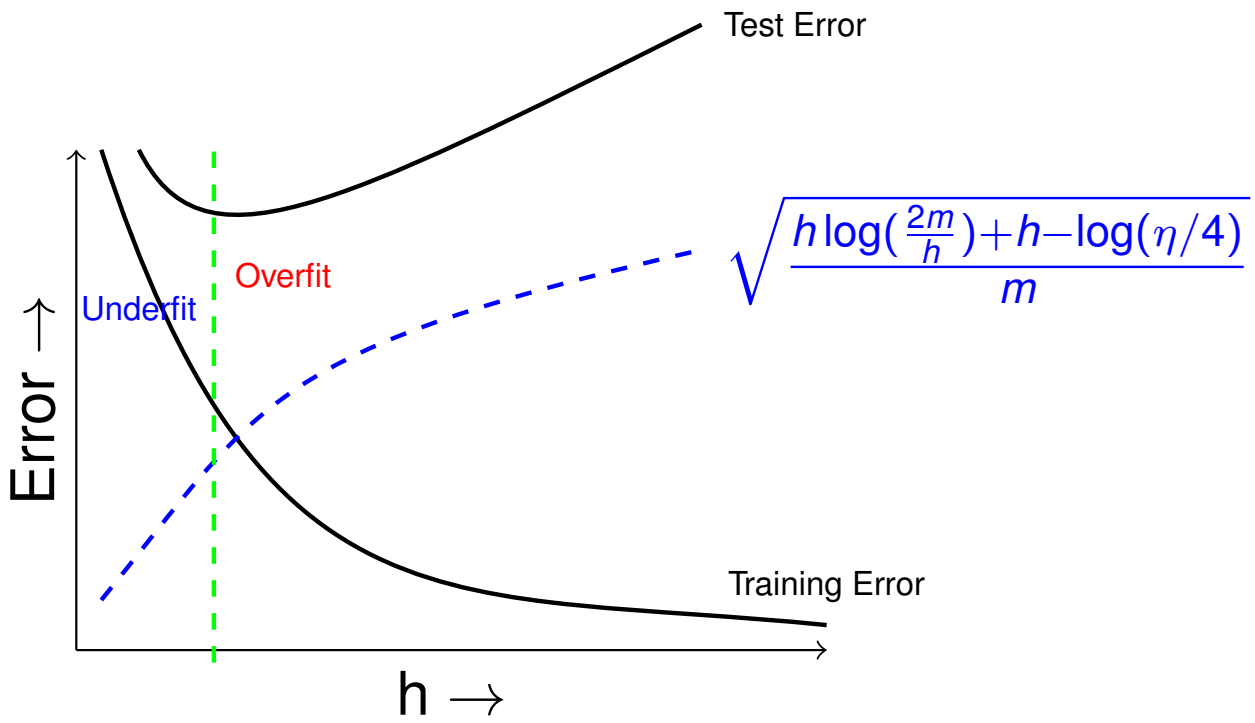
Vapnik-Chervonenkis dimension

- If VC dimension is h then $\exists\{x^{(1)}, x^{(2)}, ..., x^{(h)}\}$ such that $\forall\{y^{(1)}, y^{(2)}, ..., y^{(h)}\}\ \exists w$ such that $\forall i \quad f_w(x^{(i)}) = y^{(i)}$
- VC dimension of $sign(x^T x - w)$ is 1
- VC dimension of $sign(w^T x)$ is 3
  Liner classifier in d dimension can have VC dimension $d + 1$
- 2^d distinct hypothesis are required to shatter d instances. Hence, $2^d \leq |H|$ therefore, $d = \boxed{VC(H) \leq \log_2 |H|}$
- Training/Empirical error $R^{train}(w) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2}(|y^{(i)} - f_w(x^{(i)})|)$
- Test error $R^{test}(w) = E[\frac{1}{2}|y - f_w(x)|]$
- Vapnik showed, with probability $(1 - \eta)$

$$R^{test}(w) \leq R^{train}(w) + \sqrt{\frac{h \log(2m/h) + h - \log(\eta/4)}{m}}$$

Vapnik-Chervonenkis dimension

- Consider a neural network with w number of free parameters. Then VC dimension of the neural network is
  - $O(w \log w)$ when neurons have heavy side activation $sign(v)$
  - $O(w^2)$ when neurons have sigmoid activation $sign(1/(1+e^{-v}))$



- Number of training examples required is polynomial in VC dimension

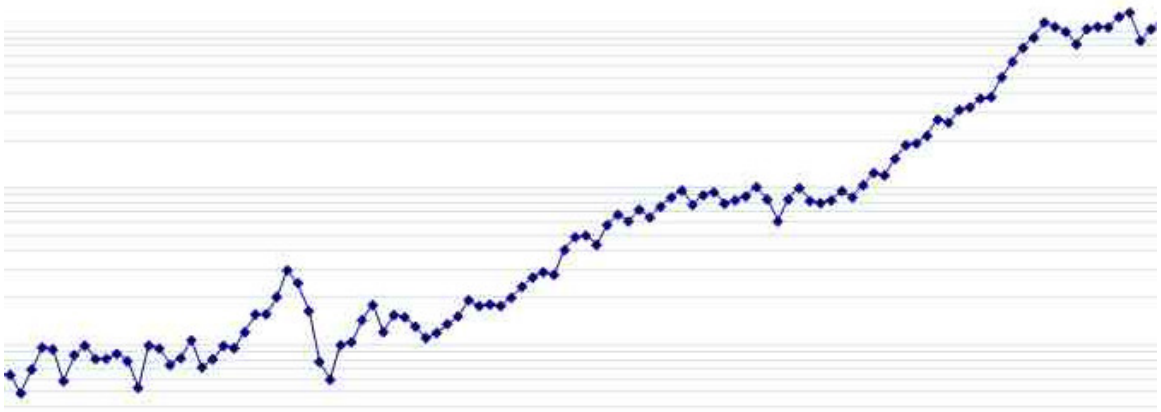
Monte Carlo Simulation

Monte Carlo simulation

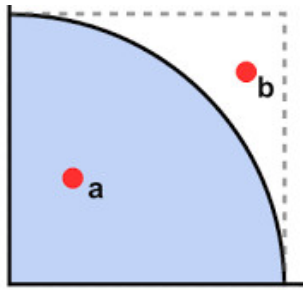
uses random numbers to simulate the environment

- Stock market:

$$v_{t+1} = v_t \times a^{randomNo}$$

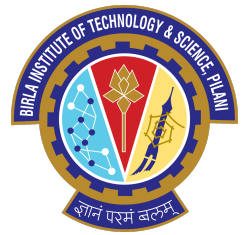


- Find value of π using rectangle and circle



IS-ZC464: MACHINE LEARNING

Lecture-13: Genetic Algorithms + Reinforcement Learning



Dr. Kamlesh Tiwari
Assistant Professor
Department of Computer Science and Information Systems,
BITS Pilani, Pilani, Jhunjhunu-333031, Rajasthan, INDIA

October 21, 2018 (WILP @ BITS-Pilani Jul-Nov 2018)

Genetic Algorithms

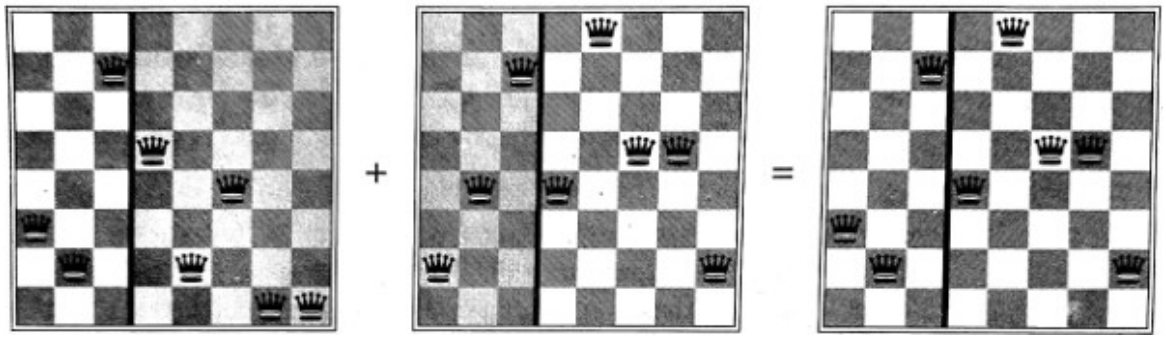
Evolution

- Recall Darwin's theory of evolution: "*Survival of the fittest*" [1]





Is it true? Let's formulate and try..



[1] Images taken from various sources on Internet

Genetic Algorithms

Learning approach of Genetic algorithms is based on simulated evolution (appeared in 1975)

- Search for a goal state begins with a population of initial hypotheses
  - Members of the current population give rise to the next generation population using random **mutation** and **crossover**
  - Hypotheses are evaluated using some `Fitness` measure
  - Most fit hypotheses act as a seeds for producing next generation
- Applied a variety of learning tasks and optimization problems (like robot control and learning parameters for ANN)
- GA is an generate-and-test approach. Search can be very complex, (may involve hill climbing, simulated annealing ...)
- Search can move abruptly. Crowding can happen

Without **guarantee**, GA often finds an object of high fitness

Genetic Algorithms

Algorithm 2: GA (`Fitness`, F_{th}, p, r, m)

1 P $\leftarrow$ generate p states at random
2 **while** $max(\texttt{Fitness}(h_1), \texttt{Fitness}(h_2), ..., \texttt{Fitness}(h_p)) < F_{th}$ **do**
3 **Select:** $(1 - r)p$ members of P
4 **Crossover:** on $(r \times p)/2$ pairs to produce two offspring
5 **Mutation:** randomly invert a bit of m percentage of population
6 **return** state h_i having maximum `Fitness`(h_i)

- `Fitness` function is typically a heuristic
- The `Fitness` function is a criterion for ranking hypotheses to select hypotheses probabilistically for inclusion in the next generation population

Genetic Algorithms

- Probability of selecting a hypothesis in next generation is

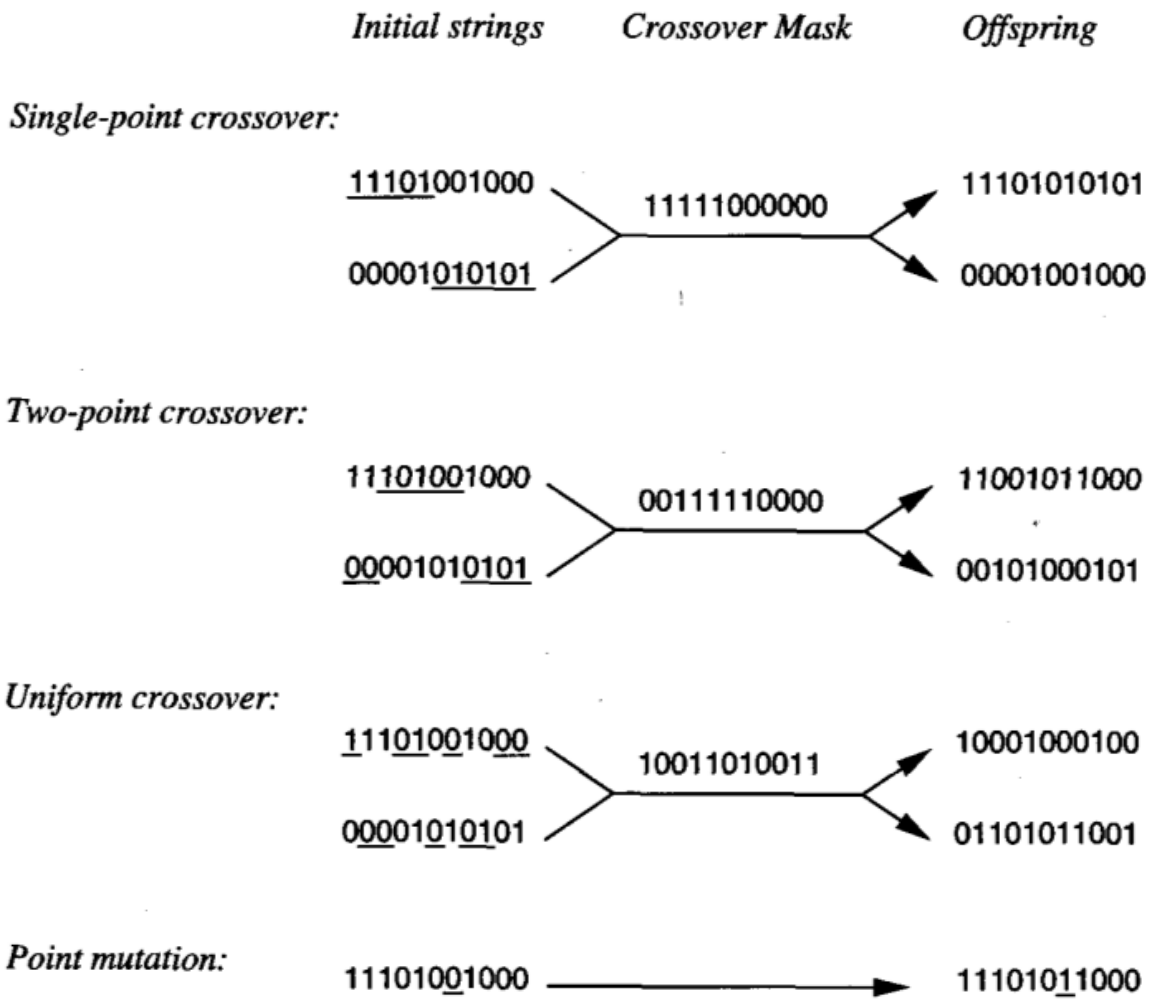
$$Pr(h_i) = \frac{\texttt{Fitness}(h_i)}{\sum_{j=1}^{p} \texttt{Fitness}(h_j)}$$

- Hypotheses are represented using fixed length bit strings (chromosome)
- Example: consider the attribute *Outlook*, which can take values from {Sunny, Overcast, Rain}
- Let string 010 means *Outlook*=Overcast
- Let string 011 means *Outlook*=Overcast $\vee$ Rain
- Consider second attribute *Wind*, that can take values from {Strong, Weak}
- The a rule such as (*Outlook*=Overcast $\vee$ Rain) $\wedge$ (*Wind*=Strong) is written as 01110

Post Condition

- Post condition such as *playTennis*=Yes can incorporated in similar way as 0111010
- Here we assume that the attribute *playTennis* can take any value from {Yes, No}
- It can also be represented as 011101 where single bit is used for boolean attributes
- Note that this representation contains a substring for each attribute in the hypothesis space, **even if that attribute is not constrained by the rule preconditions**
- This yields a fixed length bit-string representation for rules

Genetic Operators (crossover and mutation)



Fitness Function

- The fitness function defines the criterion for ranking potential hypotheses
- This is used in probabilistically selecting them for inclusion in the next generation population
- To learn classification, the fitness function typically is accuracy
- Although, one can use complexity or generality of the rule
- Or overall performance of the resulting procedure rather than performance of individual rules

Selection

- **Fitness proportionate**. Probability of selecting a hypothesis in next generation is

$$Pr(h_i) = \frac{\texttt{Fitness}(h_i)}{\sum_{j=1}^{p} \texttt{Fitness}(h_j)}$$

- **Tournament selection**. randomly pick two states and then with some predefined probability p the more fit of these two is then selected, and with probability $(1 - p)$ the less fit hypothesis is selected
- **Rank selection**. states are sorted by `Fitness` and the probability of selection of a hypothesis is proportional to its rank in this sorted list, rather than its `Fitness`
- **Elitist Model**. select a small proportion of the fittest candidates in current population intact into the next generation

Does GA works?

- Can we mathematically characterize the evolution
- **Schema:** string of 0, 1 or $*$ like $0*1$ denoting set $\{001, 011\}$
- String 1011 is **representative** of 2^4 schema
- Let $m(s,t)$ be number of instances of schema s in generation t
- Consider *selection*, let fitness of individual h be $f(h)$ and average fitness of whole n size population at time t be $\bar{f}(t)$
- $h \in s \cup p_t$ means 1) h is representative of s and 2) it is present in the population at time t
- Let $\hat{u}(s,t)$ be average fitness of instances of s at time t

$$\hat{u}(s,t) = \frac{\sum_{h \in s \cup p_t} f(h)}{m(s,t)}$$

- We know $Pr(h) = f(h)/(\sum f(h)) = f(h)/(n\bar{f}(t))$

Does GA works? (contd..)

- Probability that we will select a representative of schema s is

$$\begin{aligned} Pr(h \in s) &= \sum_{h \in s \cup p_t} Pr(h) = \sum_{h \in s \cup p_t} f(h)/(n\bar{f}(t)) \\ &= \frac{\hat{u}(s,t)}{n\bar{f}(t)} m(s,t) \end{aligned}$$

- Expected number of instances of s resulting from the n independent selection steps that create the entire new generation is just n times this probability. Therefore,

$$E[m(s,t+1)] = \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t)$$

Does GA works? (contd..)

$$E[m(s, t+1)] = \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t)$$

Expected number of representative instances of a schema s in the generation at time $t+1$ is

1. Proportional to the average fitness $\hat{u}(s,t)$ of instances of this schema at time t , and
2. Inversely proportional to the average fitness $\bar{f}(t)$ of all members of the population at time t

Thus, we can expect schema with above average fitness to be represented with increasing frequency on successive generations

Does GA works? (contd..)

Also consider negative effects of single point crossover and mutation

- Let p_c represents the probability of **crossover** on an individual. $d(s)$ be the distance between left most and right most defined bit of s and l be the length of individual bit string
- Let p_m represents the probability of **mutation** on an individual and $o(s)$ be number of defined bits in s

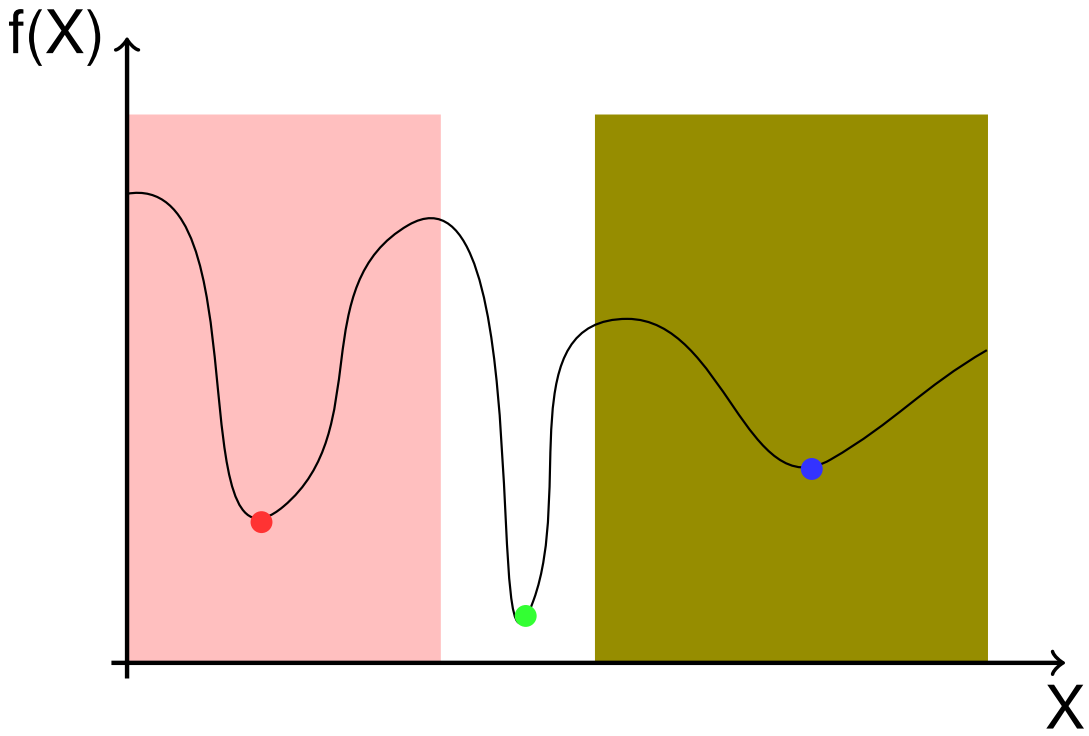
Full schema theorem thus provides a lower bound on the expected frequency of schema s, as follows

$$E[m(s,t+1)] \geq \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t)\left(1 - p_c \frac{d(s)}{l-1}\right)(1-p_m)^{o(s)}$$

Similar expression. More fit schemas will tend to grow in influence.

Optimization in GA

GA can be viewed as a general optimization method.



- Any traditional algorithms (such as gradient descent) with initial guess in pink or violet region leads to local minima
- GA would lead to global one
- Hybrid approach start with GS and then switch to traditional ones

Example

- Consider GABIL system[2] that uses a GA to learn boolean concepts represented by a disjunctive set of propositional rules
- The learning tasks included artificial learning tasks designed to explore the systems' generalization accuracy and the real world problem of breast cancer diagnosis
- Algorithm uses $r = 0.6$ fraction of the parent population replaced by crossover
- Mutation rate $m = 0.001$
- The population size p was varied from 100 to 1000, depending on the specific learning task
- Fitness function was $\texttt{Fitness}(h) = (correct(h))^2$
- GABIL achieved average generalization accuracy of 92.1% over 12 synthetic problems; whereas the performance of the other systems (C4.5, ID5R, and AQ14) ranged from 91.2% to 96.6%

[2] Using genetic algorithms for concept learning, *De Jong, Kenneth A and Spears, William M and Gordon, Diana F*, In Genetic Algorithms for Machine Learning, pp 5–32, Springer(1993)

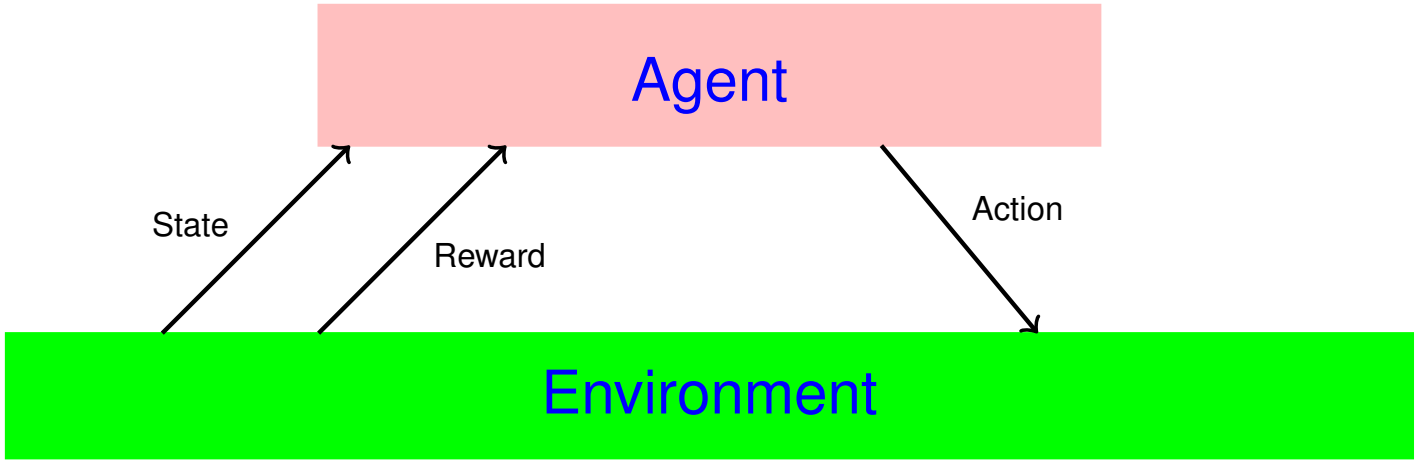
Reinforcement Learning

- Reinforcement learning works on generate and test methodology
- An agent automatically enumerate solution and compute reward/penalty

- Task of agent is to learn from indirect, delayed reward, to choose sequences of actions that produce the greatest cumulative reward
- Agent may have no prior knowledge of the effects of its actions on the environment
- Neither supervised nor unsupervised, it is trial and error
- Reinforcement learning algorithms are related to dynamic programming
- Example includes learning to control a mobile robot, learning to optimize operations in factories, and learning to play board games

State, Action, and Reward

Problem of learning a control policy to maximize cumulative reward is very general

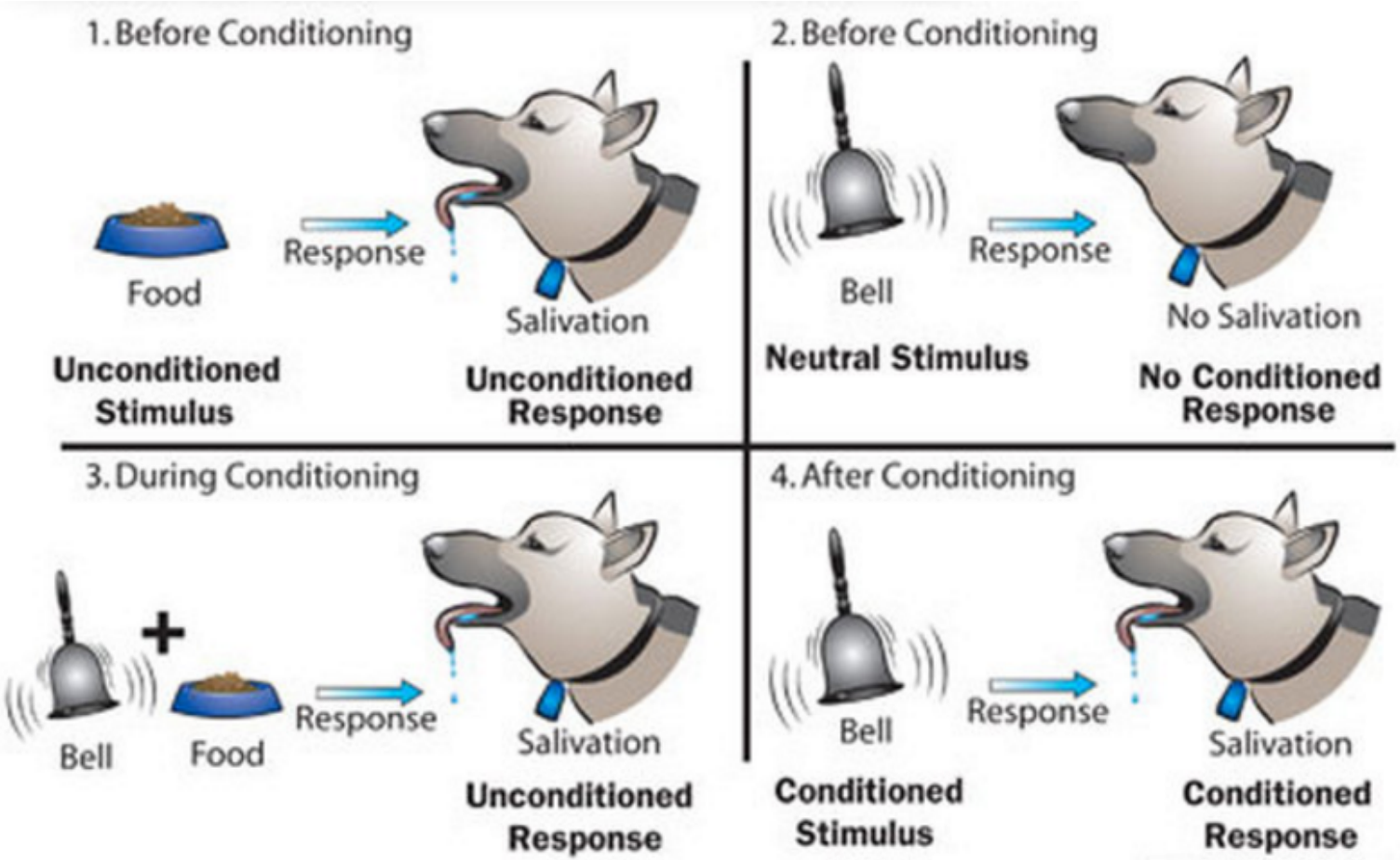


$$S_0 \xrightarrow[r_0]{a_0} S_1 \xrightarrow[r_1]{a_1} S_2 \xrightarrow[r_2]{a_2} S_3 \xrightarrow[r_3]{a_3} \cdots$$

Goal is to choose a policy that maximizes

$$r_o + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \ldots \qquad \textit{where } 0 \leq \gamma < 1$$

Reinforcement Learning



Rewards can lead to learning[3]

[3] Image from: http://www.krigolsonteaching.com/reinforcement-learning.html

Reinforcement Learning

- Target function to be learned is a policy $\pi : S \rightarrow A$ where S and A are set of states and actions. Important points are
  - Delayed reward: $r(s, \pi(s))$ is unavailable
  - Exploration: agent influences the distribution of training examples
  - Partially observable states: forward camera cannot see behind
  - Life-long learning: using previous experience to learn new tasks
- In Markov decision process (MDP), reward $r_t = r(s_t, a_t)$ and next state $s_{t+1} = \delta(s_t, a_t)$
- **Discounted cumulative reward** of a policy π from initial state s_t is

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

here $0 \leq \gamma < 1$ is a constant determining relative value of delayed versus immediate rewards

Other Reward Function

- Finite horizon reward

$$\sum_{i=0}^{h} r_{t+i}$$

- Average reward

$$\lim_{h \to \infty} \frac{1}{h} \sum_{i=0}^{h} r_{t+i}$$

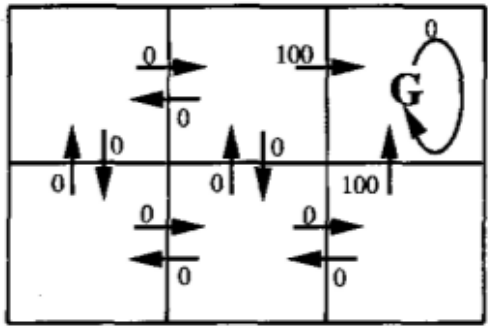
- Agent's learning task is to determine optimal policy

$$\pi^* = argmax_\pi V^\pi(s), \forall s$$

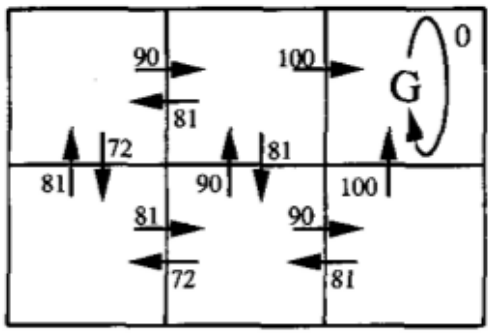
$V^{\pi^*}(s)$ is also represented as $V^*(s)$ for simplicity

An Example

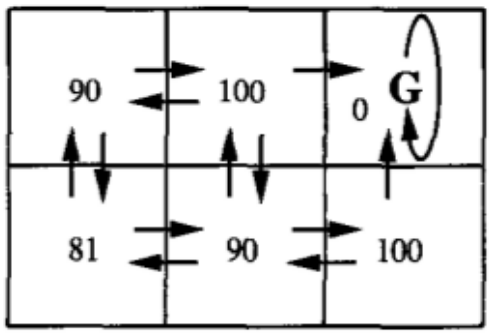
Let $\gamma = 0.9$



$r(s, a)$ (immediate reward)

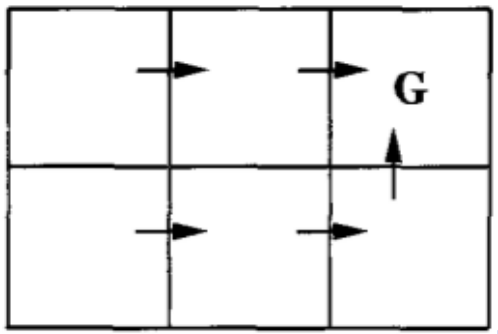


$Q(s, a)$ values



$V^*(s)$ values

One optimal policy



Q Learning

- Specifies a method for an agent to learn an optimal policy
- Agent should prefer state s_1 over state s_2 when $V^*(s_1) > V^*(s_2)$
- Optimal action in state s is

$$\pi^*(s) = \operatorname{argmax}_a \left(r(s,a) + \gamma V^*(\delta(s,a)) \right)$$

problem is that r and δ are not known

- Define a function $Q(s,a)$ representing maximum discounted cumulative reward that can be achieved starting from state s and applying action a as the first action

$$Q(s,a) = r(s,a) + \gamma V^*(\delta(s,a))$$

- Then

$$\pi^*(s) = \operatorname{argmax}_a Q(s,a)$$

Q Learning

- One can choose globally optimal action sequences by reacting repeatedly to the local values of Q for the current state
- Value of Q can be found through iterative approximation
- As $V^*(s) = max_{a'} Q(s, a')$ it is easy to see

$$Q(s, a) = r(s, a) + \gamma max_{a'} Q(\delta(s, a), a')$$

- In terms of iterative approximation

$$\hat{Q}(s, a) = r + \gamma max_{a'} \hat{Q}(s', a')$$

where immediate reward $r = r(s, a)$ and $s' = \delta(s, a)$

Q Learning Algorithm

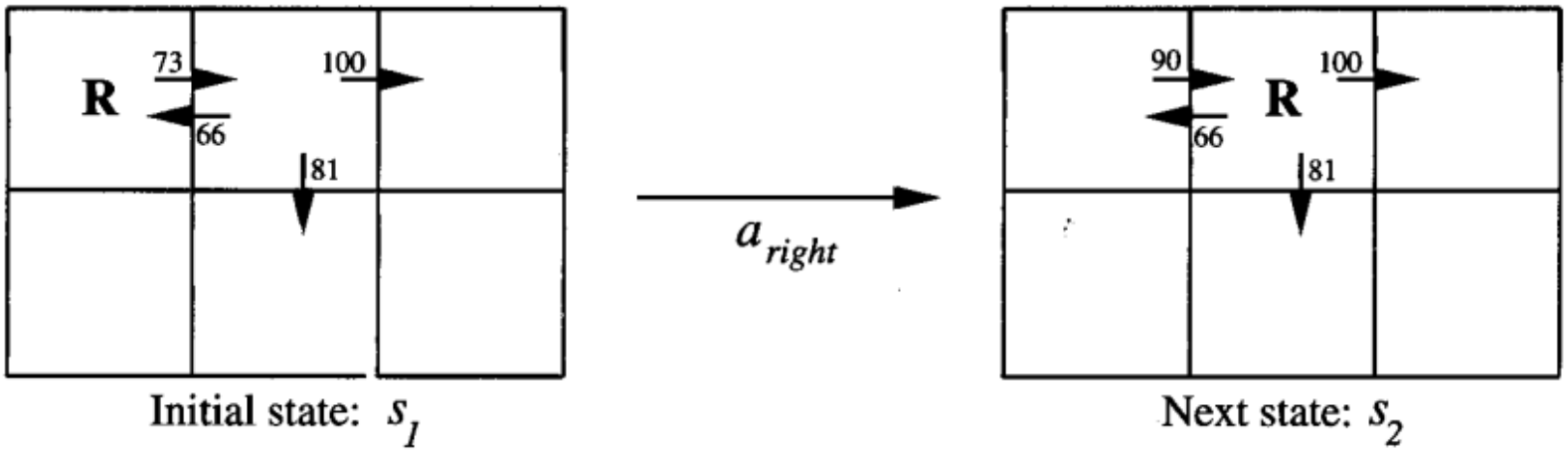
Algorithm for Q Learning can be specified as below

Algorithm 10: Q Learning

For each s, a initialize table entry $\hat{Q}(s, a) = 0$
Observe the current state s
while *True* **do**
    Select and action a and execute it
    Receive immediate response r
    Observe new stare s'
    Update the $\hat{Q}(s, a) = r + \gamma max_{a'} \hat{Q}(s', a')$
    $s = s'$

Q Learning Example

Consider a single action for the example below (let $\gamma = 0.9$)



$$\begin{aligned}
\hat{Q}(s_1, a_{right}) &= r + \gamma \max_{a'} \hat{Q}(s_2, a') \\
&= 0 + 0.9 \max_{a'} \{66, 81, 100\} \\
&= 90
\end{aligned}$$

Does it converge

- Under certain assumptions 1) Deterministic MDP, 2) rewards are bounded $|r(s,a)| < c$, 3) agent visits every possible state-action pair infinitely often
- Note that $\hat{Q}(s,a)$ does never decreases in any iteration

$$\hat{Q}_n(s,a) \leq \hat{Q}_{n+1}(s,a)$$

- And is upper bounded by $Q(s,a)$ *i.e.*

$$0 \leq \hat{Q}_n(s,a) \leq Q(s,a)$$

- Consider consecutive intervals during which each state-action transition occurs at least once.

Does it converge (contd)

- Let $\triangle_n$ be the maximum error in $\hat{Q}$ i.e.

$$\triangle_n = max_{s,a}|\hat{Q}(s,a) - Q(s,a)|$$

- For any $\hat{Q}_n(s,a)$ that is updated on iteration $n+1$, the magnitude of the error in the revised estimate $\hat{Q}_{n+1}(s,a)$ is

$$\begin{aligned}
|\hat{Q}_{n+1}(s,a) - Q(s,a)| &= |(r + \gamma max_{a'} \hat{Q}_n(s',a')) - (r + \gamma max_{a'} Q(s',a'))| \\
&= \gamma |max_{a'} \hat{Q}_n(s',a') - max_{a'} Q(s',a')| \\
&\leq \gamma max_{a'} |\hat{Q}_n(s',a') - Q(s',a')| \\
&\leq \gamma max_{s'',a'} |\hat{Q}_n(s'',a') - Q(s'',a')| \\
&= \gamma \ \triangle_n
\end{aligned}$$

- Starting from $\triangle_0$ error reduces to $\gamma \ \triangle_0 \ , \gamma^2 \ \triangle_0 \ , ..., \gamma^n \ \triangle_0 \ , ..., 0$

Therefore, $\hat{Q}$ converges to Q after infinitely many steps ...

Experimentation Strategies

- Can we select the action that maximizes $\hat{Q}(s, a)$, No as one have to visit each state infinitely often for convergence
- For some $k > 0$

$$Pr(a_i|s) = \frac{k^{\hat{Q}(s,a_i)}}{\sum_j k^{\hat{Q}(s,a_j)}}$$

Nondeterministic Rewards and Actions

- Reward and transition function may have probabilistic nature
- Policy π is then based on expected value. Discounted cumulative reward is taken as $V^{\pi}(s_t) = E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right]$
- Definition of Q becomes

$$\begin{aligned} Q(s,a) &= E[r(s,a) + \gamma V^*(\delta(s,a))] = E[r(s,a)] + \gamma E[V^*(\delta(s,a))] \\ &= E[r(s,a)] + \gamma \sum_{s'} P(s'|s,a) V^*(s') \\ &= E[r(s,a)] + \gamma \sum_{s'} P(s'|s,a) max_{a'} Q(s',a') \end{aligned}$$

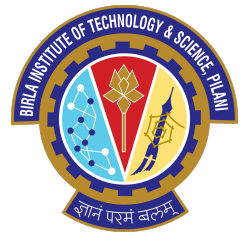
- Earlier training rule (for the deterministic) fails to converge here. Therefore, decaying weight average is taken as

$$\hat{Q}_n(s,a) = (1-\alpha_n)\hat{Q}_{n-1}(s,a) + \alpha_n[r + \gamma max_{a'} \hat{Q}_{n-1}(s',a')]$$

where $\alpha_n = \frac{1}{1+visits_n(s,a)}$ where $visits_n(s,a)$ is number of times (s,a) is visited till nth iteration.

IS-ZC464: MACHINE LEARNING

Lecture-14: Neural Network

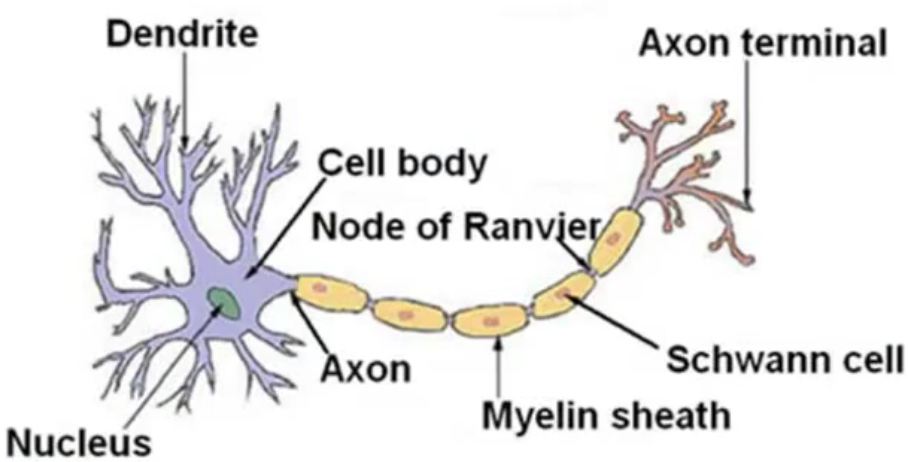


Dr. Kamlesh Tiwari
Assistant Professor
Department of Computer Science and Information Systems,
BITS Pilani, Pilani, Jhunjhunu-333031, Rajasthan, INDIA

October 27, 2018 (WILP @ BITS-Pilani Jul-Nov 2018)

Neural Network

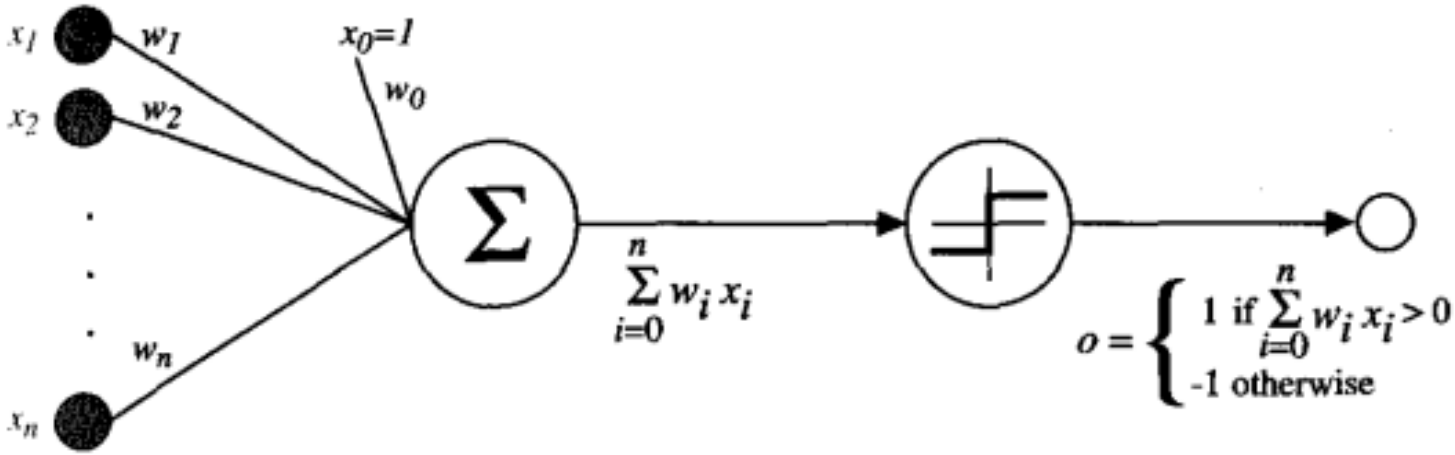
- **Neural Networks** are biologically motivated and mimic the brain
- Started by *W. McCulloch* study on working of neurons in 1943
- MADALINE (1959), an adaptive filter that eliminates echoes on phone lines was the first neural network
- Popularity of Neural Network diminished in 90's but, due to advances in **processing power** and availability of **large data** it again became state-of-the-art



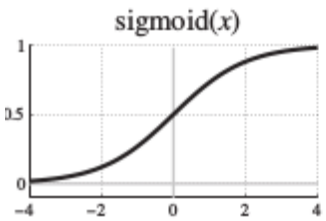
- Cell, Axon, Synopses, Molicules, and Dendrites
- Humans have 10^{11} neurons, each connected to 10^4 others, switches in 10^{-3} sec

Single Perceptron and Neuron

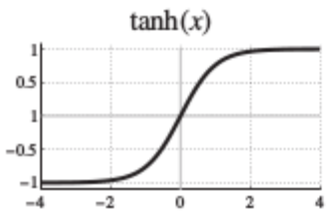
- Perceptron



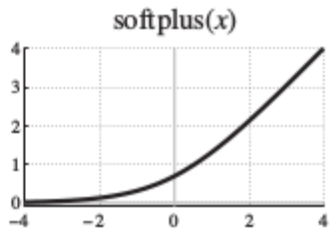
- Neuron: different activation functions could be used (*sigmoid*, *tanh*, *ReLU*, *softplus*)



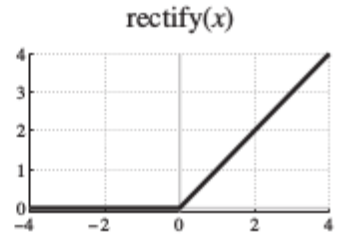
$$h(x) = \frac{1}{1+\exp(-x)}$$



$$h(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$$



$$h(x) = \log(1+\exp(x))$$

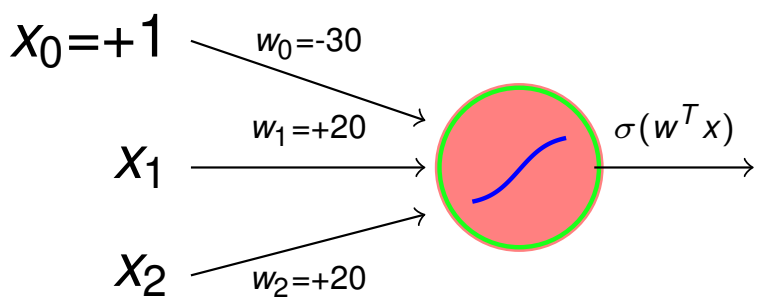


$$h(x) = \max(0, x)$$

Representation Power

- A single perceptron can represent many boolean functions
- Any *m*-of-*n* function can be represented by setting all input weights to the same value and setting threshold w_o accordingly. AND, OR, NOT is possible (where *m*-of-*n* functions is the functions where at least *m* of the *n* inputs to the perceptron must be true)

Example: consider the following neuron

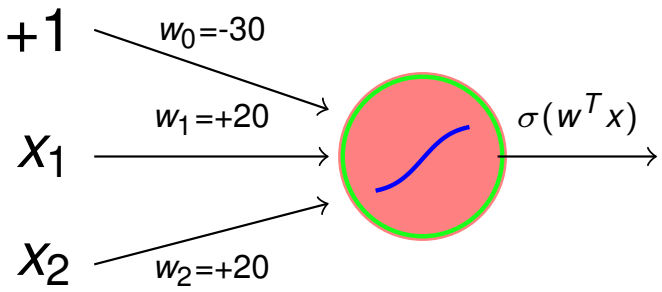


x_1	x_2	Output
0	0	0
0	1	0
1	0	0
1	1	1

It computes logical AND.

w_0=-10 gives OR. And w_0=10, w_1=-20 with single input gives NOT

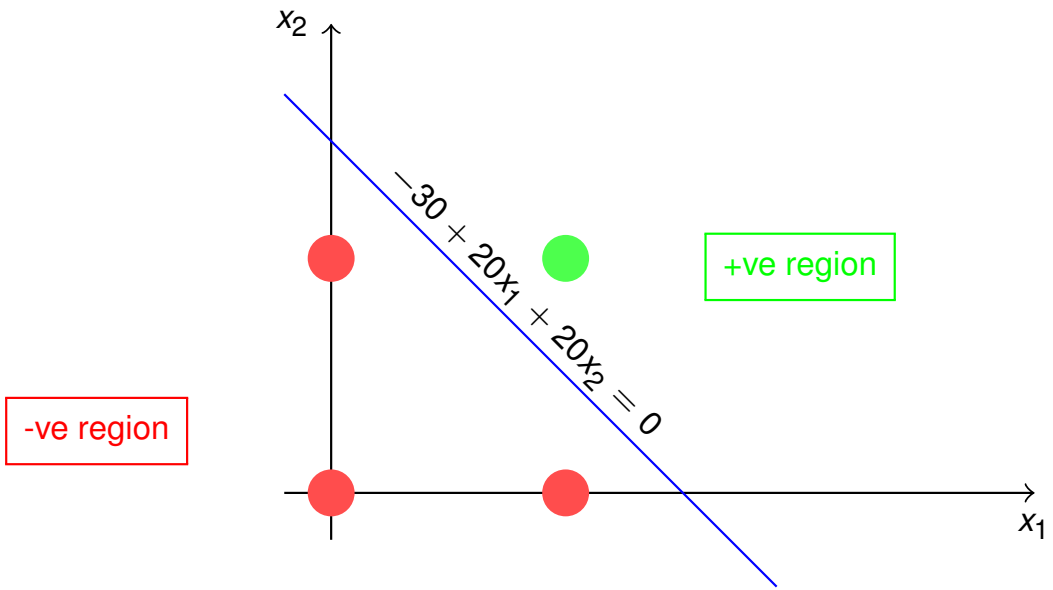
It Represents A Decision Boundary



Provides positive classification if

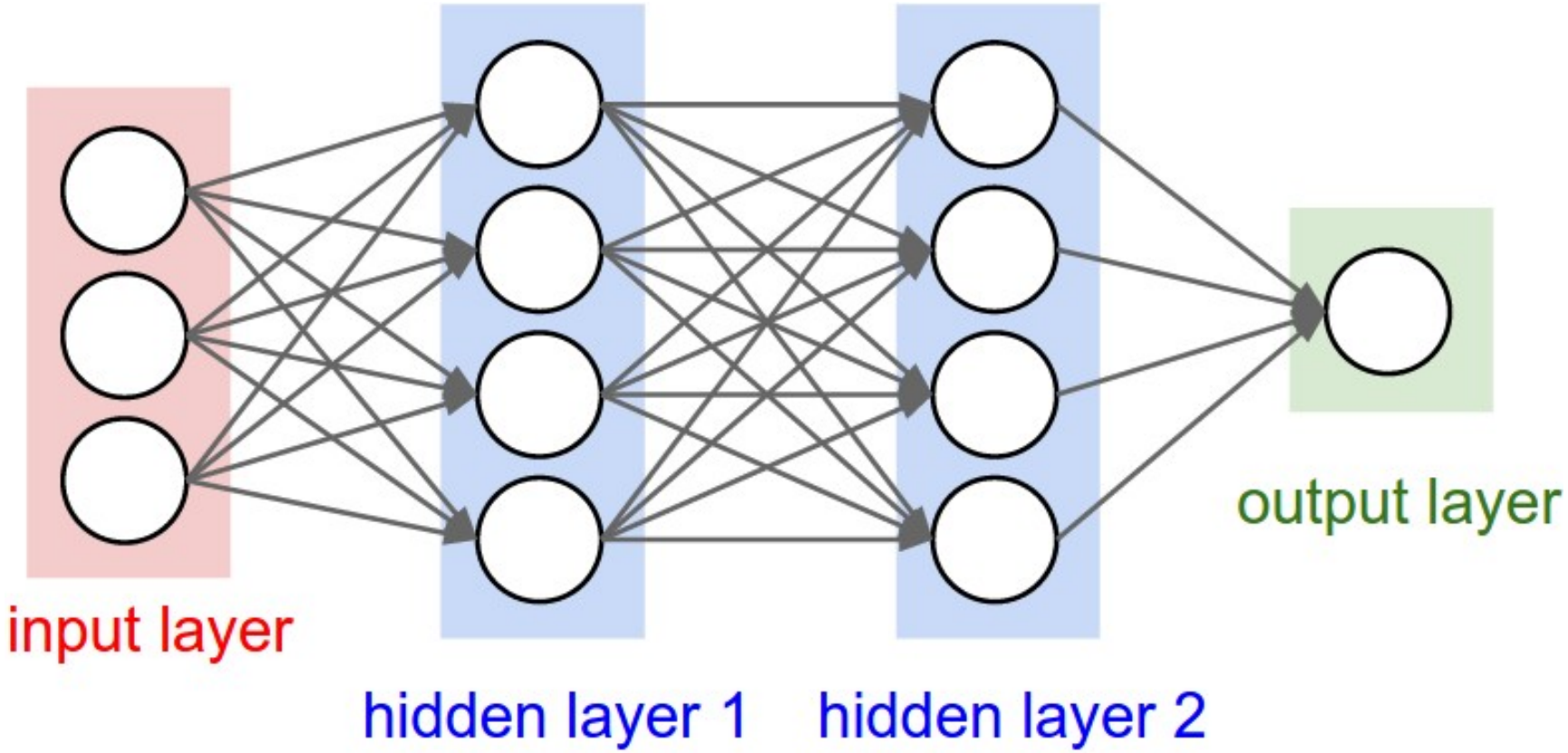
$$-30 + 20x_1 + 20x_2 \geq 0$$

Decision boundary represents a line



Neural Network

It is interconnection of neurons in layers



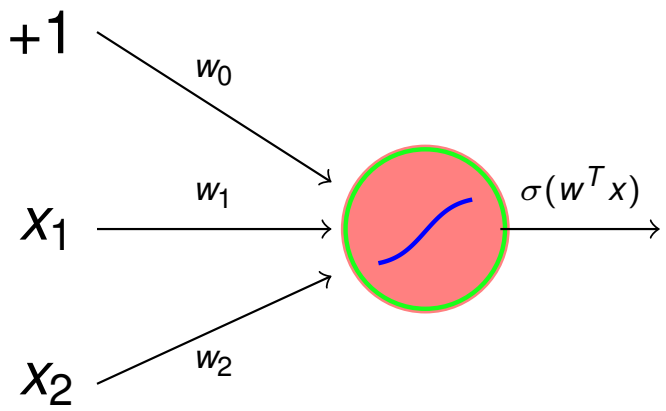
Any boolean function can be made with NN of only **two** levels deep

An Example

Design a **neuron** for

x_1	x_2	Classification
0	0	0
0	1	0
1	0	1
1	1	0

Let us assume following



We have following four equations

$$w_0 + w_1 \times (0) + w_2 \times (0) < 0 \quad (1)$$
$$w_0 + w_1 \times (0) + w_2 \times (1) < 0 \quad (2)$$
$$w_0 + w_1 \times (1) + w_2 \times (0) \geq 0 \quad (3)$$
$$w_0 + w_1 \times (1) + w_2 \times (1) < 0 \quad (4)$$

By (1) $w_0 < 0$ so let $w_0 = -1$
By (2) $w_0 + w_2 < 0$ so let $w_2 = -1$
By (3) $w_0 + w_1 \geq 0$ so let $w_1 = 1.5$
By (4) $w_0 + w_1 + w_2 < 0$ that is valid

$$\boxed{\text{So } (w_0, w_1, w_2) = (-1, -1, 1.5)}$$

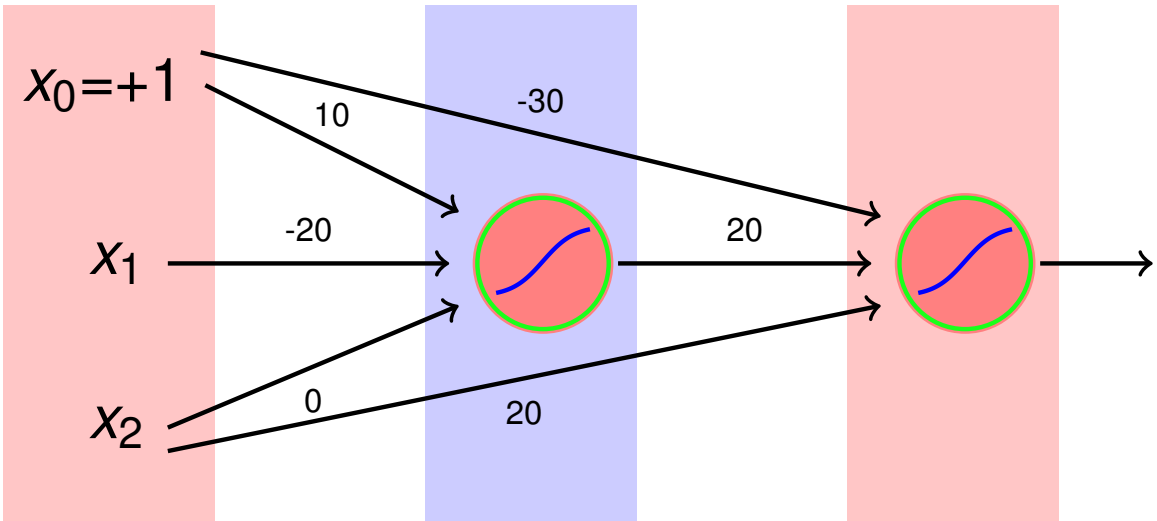
Other possibilities are also there

An Example

Design a **neural network** for

x_1	x_2	Classification
0	0	0
0	1	0
1	0	1
1	1	0

x_1	x_2	$\bar{x_2}$	$(x_1)AND(\bar{x_2})$
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0



Neural Network Applications

It is appropriate for problems with the following characteristics:

- Instances are represented by many attribute-value pairs
- The target function output may be discrete-valued, real-valued, or a vector of several real or discrete valued attributes
- The training examples may contain errors
- Long training times are acceptable
- Fast evaluation of the learned target function may be required
- The ability of humans to understand the learned target function is not important

Perceptron training rule

- There are several algorithms converging to somewhat different acceptable hypotheses
- One way is to begin with random weights
  - iteratively apply the perceptron to each training example,
  - modifying the perceptron weights whenever it misclassifies
  - until the perceptron classifies all training examples correctly
  - Weights are modified according to **perceptron training rule** as

$$w_i = w_i + \eta(t - o)x_i$$

- **Why would this strategy converge?**
  1. Weight does not change when classification is correct
  2. Perceptron outputs -1 when target is +1: weight increases
  3. Perceptron outputs +1 when target is -1: weight decreases
- Conversion with perceptron training rule is subject to linear separability of training example and appropriate η

Perceptron Training (delta rule)

- If the training examples are not linearly separable, the delta rule converges toward a best-fit approximation to the target concept.
- Uses gradient descent
- Consider unthresholded perceptron, $o(\vec{x}) = \vec{w}.\vec{x}$
- Training error is defined as $E(\vec{w}) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$
- Gradient would specify direction of steepest increase $\nabla E(\vec{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_n}\right]$
- Weights can be learned as $w_i = w_i - \eta \frac{\partial E}{\partial w_i}$
- It can be seen that $\frac{\partial E}{\partial w_i} = \sum_{d \in D}(t_d - o_d)(-x_{id})$

Perceptron Training (delta rule)

GRADIENT-DESCENT($training\_examples, \eta$)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where $\vec{x}$ is the vector of input values, and t is the target output value. η is the learning rate (e.g., .05).

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i \quad \text{(T4.1)}$$

  - For each linear unit weight w_i, Do

$$w_i \leftarrow w_i + \Delta w_i \quad \text{(T4.2)}$$

- The perceptron training rule converges when training examples that are linearly separable. The delta rule converges toward the minimum error hypothesis.
- Linear programming can also be an another approach

Multilayer Networks and Backpropagation

- Single perceptron can only express linear decision surface
- We need units whose output is a nonlinear function of its inputs AND is also differentiable (Use Neuron not ~~Perceptron~~)

$$o(\vec{x}) = \sigma(\vec{w}.\vec{x})$$

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

- Backpropagation algorithm learns the weights for a fixed set of units and interconnections
- It employs gradient descent to attempt to minimize the squared error between the network output values and the target values for these outputs
- Let Error function is redefined as

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

Backpropagation (for 2 layers)

Algorithm 1: Backpropagation(D,η,n_{in},n_{out},n_{hidden})

Create the feedforward network with n_{in}, n_{out}, n_{hidden} layers
Randomly initialize weights to small values $\in [-0.05, +0.05]$
repeat
    for *each* $<\vec{x},\vec{t}>\in D$ **do**
        $o_u =$ get output from network $\forall$ unit u
        $\delta_k = o_k(1-o_k)(t_k - o_k)$ for all output unit k
        $\delta_h = o_h(1-o_h)\sum_{k\in outputs}(w_{kh}\delta_k)$ for all hidden unit h
        $w_{ji} = w_{ji} + \triangle w_{ji}$ where $\triangle w_{ji} = \eta\delta_j x_{ji}$
until *converge*;

Derivation of the Backpropagation Rule

- Recall error function is $E(\vec{w}) = \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd} - o_{kd})^2$
- For a single training example $E_d(\vec{w}) = \frac{1}{2}\sum_{k\in outputs}(t_k - o_k)^2$
- Weight w_{ji} is updated by adding $\triangle\ w_{ji} = -\eta\frac{\partial E_d}{\partial w_{ji}}$
- Let
  - x_{ji} be the ith input to unit j
  - w_{ji} be the weight associated with ith input to unit j
  - net_j be $\sum_i w_{ji}x_{ji}$ the weighted sum of input for unit j
  - o_j output computed by unit j. it is $\sigma(net_j)$
  - t_j target output for unit j
  - *outputs* be the set of units in final layer
  - *Downstream*(j) be the set of units whose immediate input include the output of unit j

We are interested in $\frac{\partial E_d}{\partial w_{ji}}$ it is $\frac{\partial E_d}{\partial net_j} \times \frac{\partial net_j}{\partial w_{ji}}$ and therefore, $\frac{\partial E_d}{\partial net_j} \times x_{ji}$

Value of $\frac{\partial E_d}{\partial net_j}$ for output units

- $\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial net_j}$
- $\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$
- Note that $o_j = \sigma(net_j)$ therefore $\frac{\partial o_j}{\partial net_j}$ is derivative of sigmoid

$$\begin{aligned} \frac{d}{dx}\sigma(x) &= \frac{d}{dx}\frac{1}{1+e^{-x}} = (-1)(1+e^{-x})^{-2}\frac{d}{dx}(1+e^{-x}) \\ &= (-1)(1+e^{-x})^{-2}(0-e^{-x}) \\ &= \frac{1}{1+e^{-x}} \times \frac{e^{-x}+1-1}{1+e^{-x}} = \sigma(x)(1-\sigma(x)) \end{aligned}$$

- As a result $\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j} = \sigma(net_j)(1 - \sigma(net_j)) = o_j(1 - o_j)$
- $\frac{\partial E_d}{\partial net_j} = -(t_j - o_j) o_j (1 - o_j)$
- Term $(t_j - o_j) o_j (1 - o_j)$ is treated as δ_j

Therefore, $\triangle w_{ji} = -\eta \frac{\partial E_d}{\partial net_j} \times x_{ji} = \boxed{\eta (t_j - o_j) o_j (1 - o_j) x_{ji}}$

Value of $\frac{\partial E_d}{\partial net_j}$ for hidden units

$$\begin{aligned}
\frac{\partial E_d}{\partial net_j} &= \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \times \frac{\partial net_k}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \times \frac{\partial net_k}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \times \frac{\partial net_k}{\partial o_j} \times \frac{\partial o_j}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \times w_{kj} \times o_j(1 - o_j)
\end{aligned}$$

- δ_j being $-\frac{\partial E_d}{\partial net_j} = o_j(1 - o_j) \sum_{k \in Downstream(j)} \delta_k \times w_{kj}$

Therefore, $\triangle\ w_{ji} = \eta \delta_j x_{ji} = \boxed{\eta(o_j(1 - o_j) \sum_{k \in Downstream(j)} \delta_k \times w_{kj}) x_{ji}}$

Backpropagation

- **Adding Momentum:** weight update during nth iteration depend partially on the update that occurred during the $(n-1)^{th}$ iteration

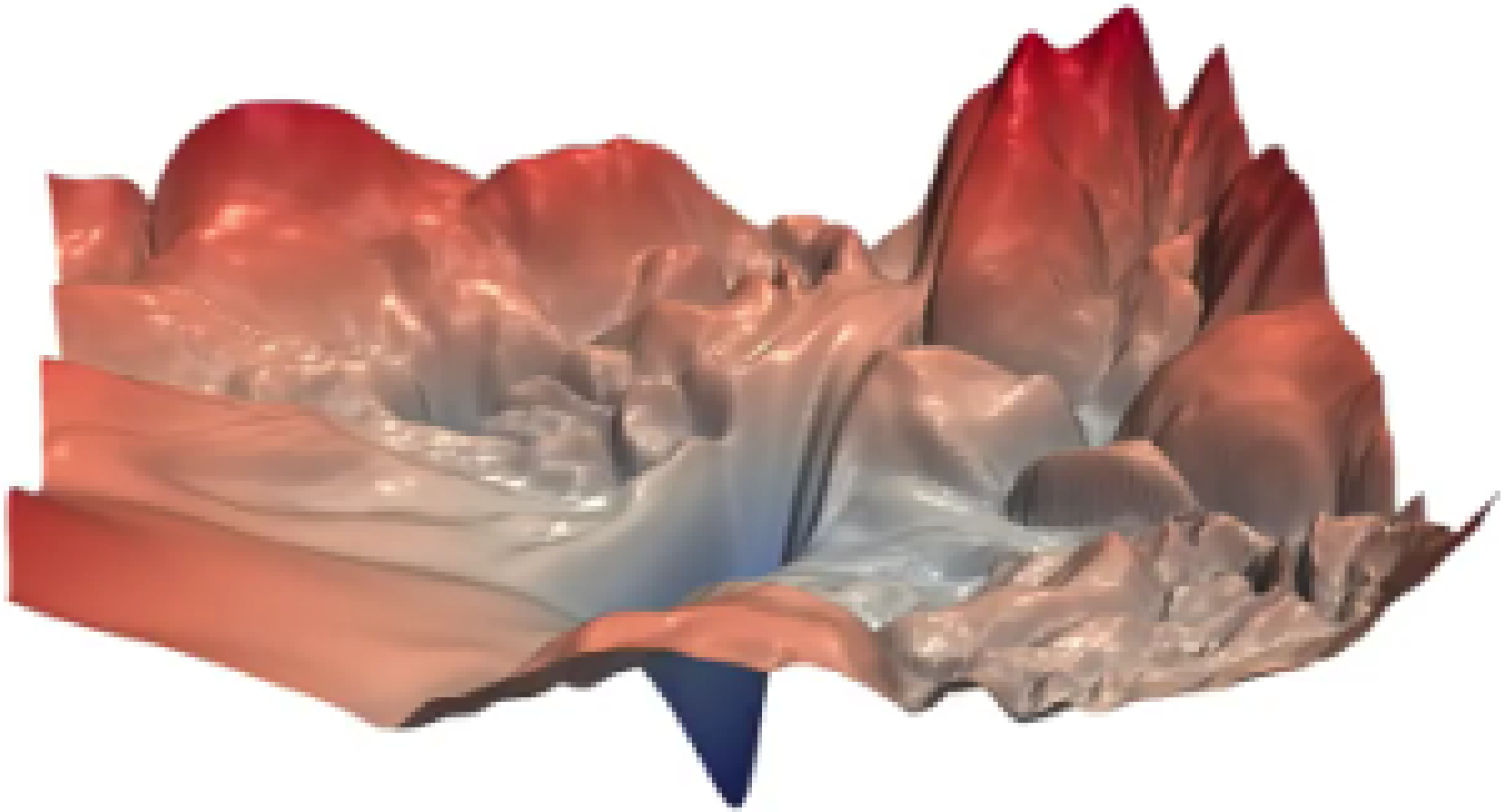
$$\triangle w_{ji}(n) = \eta \delta_j x_{ji} + \alpha \triangle w_{ji}(n-1)$$

- **Learning in arbitrary acyclic network:** for feed-forward networks of arbitrary depth, δ_r value for a unit in hidden layer is determined as

$$\delta_r = o_r(1-o_r) \sum_{s \in Downstream(r)} w_{sr} \times \delta_s$$

Backpropagation

- Result of Backpropagation over multilayer networks is only guaranteed to converge toward some local minimum and not necessarily to the global minimum error

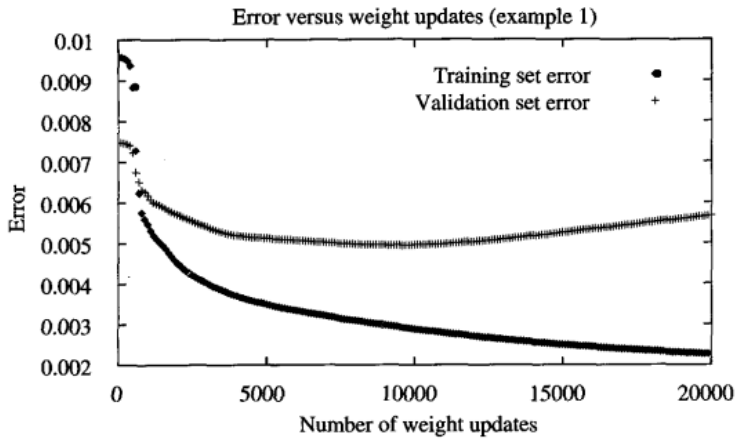


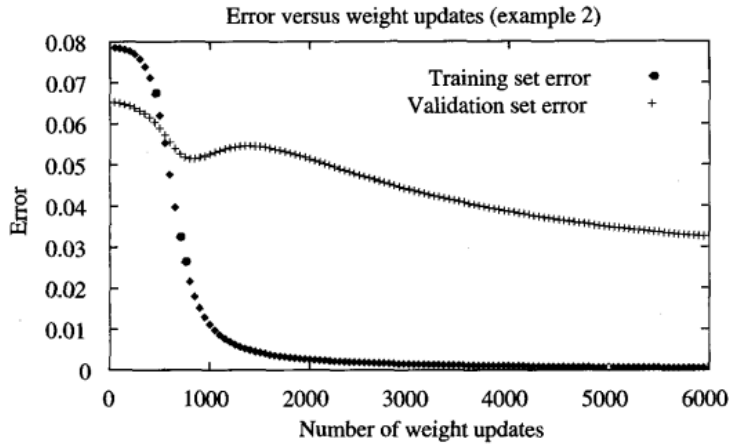
Backpropagation

- Result of Backpropagation over multilayer networks is only guaranteed to converge toward some local minimum and not necessarily to the global minimum error
- No methods are known to predict with certainty when local minima will cause difficulties
- Suggested to use momentum, true gradient descent or multiple networks (initialized with different random weights)
- Any boolean function can precisely be represented by some network having only **two** layers of units (Cybenko 1989; Hornik et al. 1989)
- Every bounded continuous function can be approximated with arbitrarily small error (under a finite norm) by a network with two layers of units
- Any function can be approximated to arbitrary accuracy by a network with three layers of units (Cybenko 1988)

Generalization, Overfitting, and Stopping Criterion

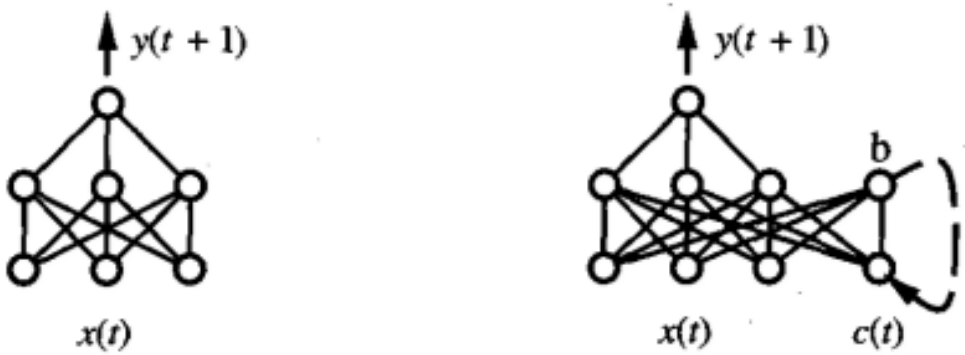
- Continue training until the error E on the training examples falls below some predetermined threshold could be a poor strategy
- Weight decay or use of validation set (k-fold ?) is suggested
- Input or output encoding can be used





Recurrent Networks (RNN)

- Recurrent networks are artificial neural networks that apply to time series data and that use outputs of network units at time t as the input to other units at time $t + 1$
- Feed forward network such as one shown in (a) can not capture the dependence of $y(t + 1)$ on earlier values of x such as $x(t - 1)$

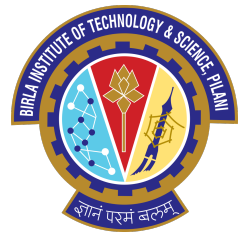


(a) Feedforward network (b) Recurrent network

- Value of $c(t)$ is defined as the value of unit b at time $t - 1$ by this it incorporates history of the network
- Training may involve unfolding and averaging.

IS-ZC464: MACHINE LEARNING

Lecture-15: SOM + Boosting + Bagging



Dr. Kamlesh Tiwari
Assistant Professor
Department of Computer Science and Information Systems,
BITS Pilani, Pilani, Jhunjhunu-333031, Rajasthan, INDIA

November 03, 2018 (WILP @ BITS-Pilani Jul-Nov 2018)

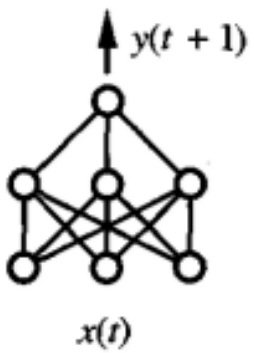
Recap: Neural Network

Neural networks are biologically motivated learning systems

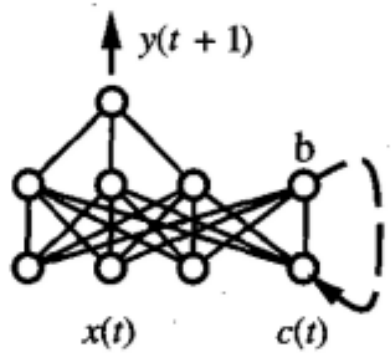
- **Training** for Multilayer Networks uses *Backpropagation* using error function $E(\vec{w}) = \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd} - o_{kd})^2$

$$\frac{\partial E_d}{\partial w_{ji}} = \begin{cases} \eta(t_j - o_j)o_j(1-o_j)x_{ji} & \text{Output units} \\ \eta(o_j(1-o_j)\sum_{k\in Downstream(j)}\delta_k \times w_{kj})x_{ji} & \text{Hidden units} \end{cases}$$

- **RNN:** Recurrent networks are artificial neural networks that apply to time series data



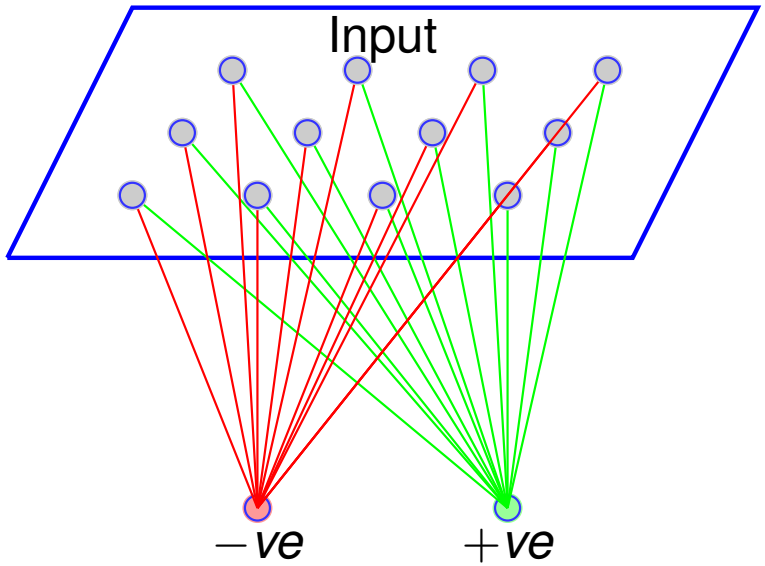
(a) Feedforward network



(b) Recurrent network

Self-Organizing Map (SOM)

- SOM neural network models is based on unsupervised, competitive learning
- It provides a topology preserving mapping
- Neurons are represented using weight vector called the codebook $(w_1, w_2, .., w_n)$. Consider input as $x = (x_1, x_2, ..., x_n)$
- SOM can be used of clustering and feature detection (SOFM)
- We would see Kohonen model



Self-Organizing Map (SOM)

- Three Process
  1. **Competition:** discriminant function is computed

$$argmin_j ||\vec{x} - \vec{w}_j||$$

  2. **Cooperation:** neuron in spacial neighborhood h_{ij} are excited

$$h_{ij} = e^{-d_{ij}^2/(2\sigma^2)}$$

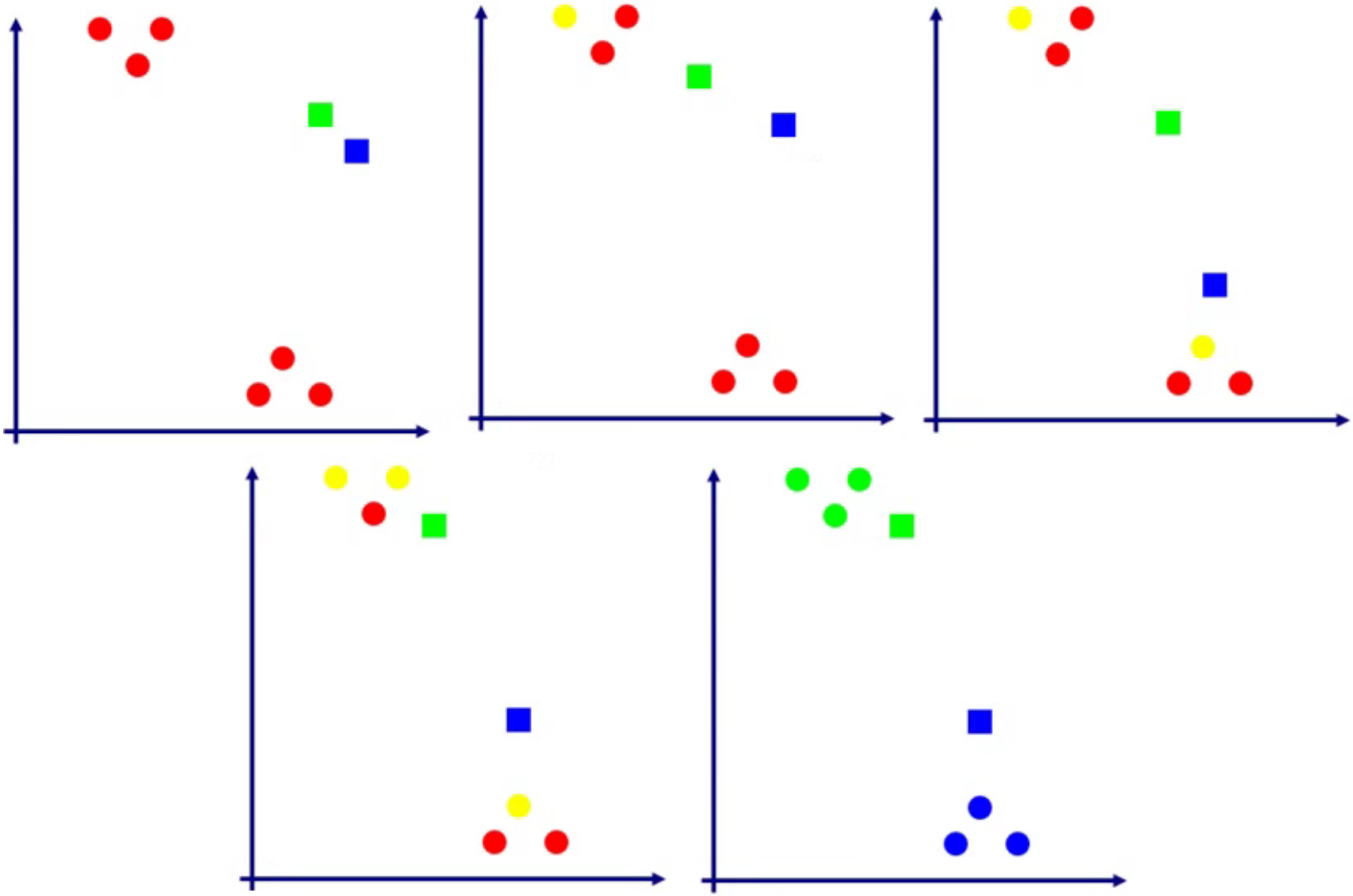
     d_{ij} is lateral distance between neuron i and j. σ decreases with every iteration $\sigma(n) = \sigma_0 e^{-n/\tau_1}$
  3. **Synaptic Adaptation:** weight adjustment (Hebbian learning)

$$\triangle \vec{w}_j = \eta h_{ij}(\vec{x} - \vec{w}_j)$$

$$\vec{w}_j(n+1) = \vec{w}_j(n) + \eta(n) h_{ij}(\vec{x} - \vec{w}_j)$$

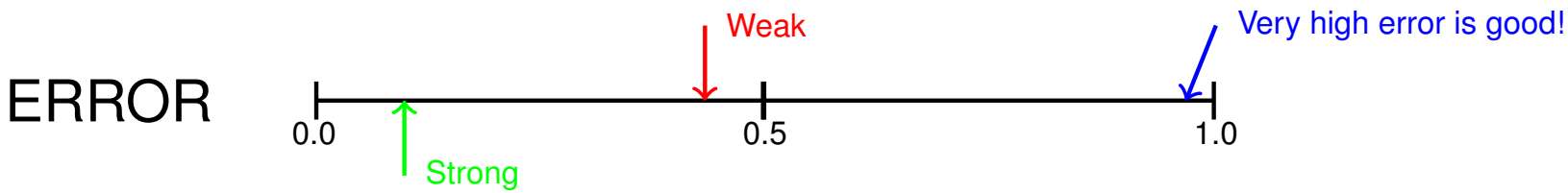
where $\eta(n) = \eta_0 e^{-n/\tau_2}$

Self-Organizing Map (SOM)



Classification

- Approaches of binary classification $[-1, +1]$
  - Linear classifiers
  - Quadratic classifiers
  - Bayesian classification
  - Support vector machines
  - Decision trees
  - Neural networks
  - k-nearest neighbor
- Which one to use?
- Weak and strong classifier?



Boosting

- Can we use ensemble of weak learners to get a strong one

$$H(x) = sign(h^1(x) + h^2(x) + h^3(x) + ...)$$

- What is error rate?

$$\epsilon = \frac{1}{m}\sum_{i=1}^{m}\left(H(x^{(i)}) \neq y^{(i)}\right) = \sum_{wrong}\frac{1}{m}$$

Three weak learners having error on different data points we are done

- But, most likely it is not going to be the case
- We can do the following
  1. Get weak learner h^1 (that is best)
  2. Exaggerate the data where h^1 errs and get weak learner h^2
  3. Exaggerate again the data for $h^1 \neq h^2$ and get weak learner h^3
- Will this work?

Boosting

- Boosting uses ensemble of learners (weak ones)
- AdaBoost[1] algorithm is one of the widely used boosting algorithm
- Given with m labeled training examples $(x^{(1)}, y^{(1)}), ..., (x^{(m)}, y^{(m)})$ where the $x^{(i)} \in \mathcal{X}$, and the labels $y^{(i)} \in \{-1, +1\}$
- Round t computes a distribution D_t is over training examples, and a weak learning algorithm is applied to find a weak hypothesis $h_t : \mathcal{X} \rightarrow \{-1, +1\}$. Aim is to find a weak hypothesis with low weighted error ϵ_t relative to D_t
- The final or combined hypothesis H computes the sign of a weighted combination of weak hypotheses

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$$

[1] A decision-theoretic generalization of on-line learning and an application to boosting, Freund, Yoav and Schapire, Robert E, European conference on computational learning theory, pp 23–37 Springer(1995)

Boosting

- Assume all data points have a weight w_i associated with them. Initially $w_i = \frac{1}{m}$ so that error rate becomes

$$\epsilon = \sum_{wrong} \frac{1}{m} = \sum_{i \in wrong} w_i$$

- One thing we want to make sure that sum of weights be 1

$$\sum w_i = 1$$

- Weights are modified in every round

AdaBoost Algorithm

Algorithm 4: AdaBoost

1 Initialize $w_1, ..., w_m$ to $\frac{1}{m}$
2 **for** *round* $t = 1$ *to* T **do**
3 Choose a hypothesis h^t that minimizes error ϵ^t over the current distribution
4 Compute $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon^t}{\epsilon^t} \right)$
5 Update values of all $w_1, ..., w_m$
6 **return** h and α

- Update of weights w_i at time t is done using

$$w_i^{t+1} = \frac{w_i^t}{z_t} e^{-(\alpha_t h^t(x^{(i)}) y^{(i)})}$$

  for all data point *i.e.* value of i varies from 1 to m
- Here z_t is a normalization factor

Normalization factor

- Here z_t is a normalization factor so

$$
\begin{aligned}
z_t &= \sum_{i=1}^{m} w_i^t e^{-(\alpha_t h^t(x^{(i)}) y^{(i)})} \\
&= \sum_{i \in right} w_i^t e^{-(\alpha_t h^t(x^{(i)}) y^{(i)})} + \sum_{i \in wrong} w_i^t e^{-(\alpha_t h^t(x^{(i)}) y^{(i)})} \\
&= \sum_{i \in right} w_i^t e^{-(\alpha_t)} + \sum_{i \in wrong} w_i^t e^{-(\alpha_t(-1))} \\
&= \sum_{i \in right} w_i^t \sqrt{\frac{\epsilon^t}{1-\epsilon^t}} + \sum_{i \in wrong} w_i^t \sqrt{\frac{1-\epsilon^t}{\epsilon^t}} \\
&= \sqrt{\frac{\epsilon^t}{1-\epsilon^t}} \sum_{i \in right} w_i^t + \sqrt{\frac{1-\epsilon^t}{\epsilon^t}} \sum_{i \in wrong} w_i^t \\
&= \sqrt{\frac{\epsilon^t}{1-\epsilon^t}} (1-\epsilon^t) + \sqrt{\frac{1-\epsilon^t}{\epsilon^t}} \epsilon^t \\
&= 2\sqrt{\epsilon^t (1-\epsilon^t)}
\end{aligned}
$$

Boosting

- Weight update for correctly classified data point

$$w_i^{t+1} = w_i^t \frac{1}{2(1-\epsilon^t)}$$

- Weight update is therefore for wrongly classified data point

$$w_i^{t+1} = w_i^t \frac{1}{2\epsilon^t}$$

Final hypothesis

$$H(x) = sign(\alpha_1 h^1(x) + \alpha_2 h^2(x) + \alpha_3 h^3(x) + ...)$$

$$\epsilon^t = \sum_{i \in wrong} w_i^t$$

AdaBoost at work

Consider the following data set

	(x	y)
s_1	(1,	-1)
s_2	(2,	-1)
s_3	(3,	+1)
s_4	(4,	+1)
s_5	(5,	+1)
s_6	(6,	+1)
s_7	(7,	+1)
s_8	(8,	-1)
s_9	(9,	-1)
s_{10}	(10,	-1)

- **Round-00:** initialize weights

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

- **Round-01:** Let sampling according to w produces $(3, 10, 8, 2, 7, 5, 1, 3, 8, 3)$ so following sub-set of data is considered

s_1	(1,	-1)
s_2	(2,	-1)
s_3	(3,	+1)
s_5	(5,	+1)
s_7	(7,	+1)
s_8	(8,	-1)
s_{10}	(10,	-1)

Consider various hypothesis

AdaBoost at work (Round-01)

			h_a	h_b	h_c	h_d	h_e	h_f	h_g	h_h
s_1	(1,	-1)	+1	-1	-1	-1	-1	-1	-1	-1
s_2	(2,	-1)	+1	+1	-1	-1	-1	-1	-1	-1
s_3	(3,	+1)	+1	+1	+1	-1	-1	-1	-1	-1
s_5	(5,	+1)	+1	+1	+1	+1	-1	-1	-1	-1
s_7	(7,	+1)	+1	+1	+1	+1	+1	-1	-1	-1
s_8	(8,	-1)	+1	+1	+1	+1	+1	+1	-1	-1
s_{10}	(10,	-1)	+1	+1	+1	+1	+1	+1	+1	-1

- Select h_c as $h1$
- What is decision threshold? 2.5
- Compute error on whole dataset

AdaBoost at work (Round-01)

Threshold is 2.5

	(x	y)	h_1
s_1	(1,	-1)	-1
s_2	(2,	-1)	-1
s_3	(3,	+1)	+1
s_4	(4,	+1)	+1
s_5	(5,	+1)	+1
s_6	(6,	+1)	+1
s_7	(7,	+1)	+1
s_8	(8,	-1)	+1
s_9	(9,	-1)	+1
s_{10}	(10,	-1)	+1

- Error rate $\epsilon = \sum_{i \in wrong} w_i$
  $= w_8 + w_9 + w_{10} = 0.1 + 0.1 + 0.1 = 0.3$
- $\alpha = \frac{1}{2} \ln\left(\frac{1-\epsilon}{\epsilon}\right) = \frac{1}{2} \ln\left(\frac{1-0.3}{0.3}\right) = 0.4236$
- Weights are modified according to

$$w_i = w_i \times \begin{cases} \frac{1}{2(1-\epsilon)} = 0.7142 & \text{correct} \\ \frac{1}{2\epsilon} = 1.6666 & \text{wrong} \end{cases}$$

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.17	0.17	0.17

AdaBoost at work (Round-02)

Consider the data set

	(x	y)
s_1	(1,	-1)
s_2	(2,	-1)
s_3	(3,	+1)
s_4	(4,	+1)
s_5	(5,	+1)
s_6	(6,	+1)
s_7	(7,	+1)
s_8	(8,	-1)
s_9	(9,	-1)
s_{10}	(10,	-1)

- **Round-02:** Let sampling according to new w produces $(8, 10, 7, 10, 3, 1, 10, 8, 4, 9)$ so following sub-set of data is considered

s_1	(1,	-1)
s_3	(3,	+1)
s_4	(4,	+1)
s_7	(7,	+1)
s_8	(8,	-1)
s_9	(9,	-1)
s_{10}	(10,	-1)

Consider various hypothesis

AdaBoost at work (Round-02)

			h_a	h_b	h_c	h_d	h_e	h_f	h_g	h_h
s_1	(1,	-1)	+1	-1	-1	-1	-1	-1	-1	-1
s_3	(3,	+1)	+1	+1	-1	-1	-1	-1	-1	-1
s_4	(4,	+1)	+1	+1	+1	-1	-1	-1	-1	-1
s_7	(7,	+1)	+1	+1	+1	+1	-1	-1	-1	-1
s_8	(8,	-1)	+1	+1	+1	+1	+1	-1	-1	-1
s_9	(9,	-1)	+1	+1	+1	+1	+1	+1	-1	-1
s_{10}	(10,	-1)	+1	+1	+1	+1	+1	+1	+1	-1

- Select h_h as $h2$
- What is decision threshold? 10.5
- Compute error on whole dataset

AdaBoost at work (Round-02)

Threshold is 10.5

	(x	y)	h_2
s_1	(1,	-1)	-1
s_2	(2,	-1)	-1
s_3	(3,	+1)	-1
s_4	(4,	+1)	-1
s_5	(5,	+1)	-1
s_6	(6,	+1)	-1
s_7	(7,	+1)	-1
s_8	(8,	-1)	-1
s_9	(9,	-1)	-1
s_{10}	(10,	-1)	-1

- Error rate $\epsilon = \sum_{i \in wrong} w_i$
  $= w_3 + w_4 + w_5 + w_7 = 4 \times 0.07 = 0.28$
- $\alpha = \frac{1}{2} \ln\left(\frac{1-\epsilon}{\epsilon}\right) = \frac{1}{2} \ln\left(\frac{1-0.28}{0.28}\right) = 0.4722$
- Weights are modified according to

$$w_i = w_i \times \begin{cases} \frac{1}{2(1-\epsilon)} = 0.6944 & \text{correct} \\ \frac{1}{2\epsilon} = 1.7857 & \text{wrong} \end{cases}$$

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.17	0.17	0.17
0.048	0.048	0.124	0.124	0.124	0.124	0.124	0.118	0.118	0.118

AdaBoost at work (Round-03)

Consider the data set

	(x	y)
s_1	(1,	-1)
s_2	(2,	-1)
s_3	(3,	+1)
s_4	(4,	+1)
s_5	(5,	+1)
s_6	(6,	+1)
s_7	(7,	+1)
s_8	(8,	-1)
s_9	(9,	-1)
s_{10}	(10,	-1)

- **Round-03:** Let sampling according to new w produces $(8, 7, 4, 8, 6, 9, 5, 8, 3, 4)$ so following sub-set of data is considered

	(x	y)
s_3	(3,	+1)
s_4	(4,	+1)
s_5	(5,	+1)
s_6	(6,	+1)
s_7	(7,	+1)
s_8	(8,	-1)
s_9	(9,	-1)

AdaBoost at work (Round-03)

			h_a	h_b	h_c	h_d	h_e	h_f	h_g	h_h
s_3	(3,	+1)	+1	-1	-1	-1	-1	-1	-1	-1
s_4	(4,	+1)	+1	+1	-1	-1	-1	-1	-1	-1
s_5	(5,	+1)	+1	+1	+1	-1	-1	-1	-1	-1
s_6	(6,	+1)	+1	+1	+1	+1	-1	-1	-1	-1
s_7	(7,	+1)	+1	+1	+1	+1	+1	-1	-1	-1
s_8	(8,	-1)	+1	+1	+1	+1	+1	+1	-1	-1
s_9	(9,	-1)	+1	+1	+1	+1	+1	+1	+1	-1

- Select h_a as $h3$
- What is decision threshold? 0.5
- Compute error on whole dataset

AdaBoost at work (Round-03)

Threshold is 0.5

	(x	y)	h_3
s_1	(1,	-1)	+1
s_2	(2,	-1)	+1
s_3	(3,	+1)	+1
s_4	(4,	+1)	+1
s_5	(5,	+1)	+1
s_6	(6,	+1)	+1
s_7	(7,	+1)	+1
s_8	(8,	-1)	+1
s_9	(9,	-1)	+1
s_{10}	(10,	-1)	+1

- Error rate $\epsilon = \sum_{i \in wrong} w_i$
  $= 2 \times 0.048 + 3 \times 0.118 = 0.45$
- $\alpha = \frac{1}{2} \ln\left(\frac{1-\epsilon}{\epsilon}\right) = \frac{1}{2} \ln\left(\frac{1-0.45}{0.45}\right) = 0.10$
- We may continue for next round like that
- Let use see our accuracy now

AdaBoost at work (Round-03)

$(\alpha_1, \alpha_2, \alpha_3) = (0.4236, 0.4722, 0.45)$

	(x	y)	h_1	h_2	h_3	$sign(\sum_i(\alpha_i \times h_i))$
s_1	(1,	-1)	-1	-1	+1	-1
s_2	(2,	-1)	-1	-1	+1	-1
s_3	(3,	+1)	+1	-1	+1	+1
s_4	(4,	+1)	+1	-1	+1	+1
s_5	(5,	+1)	+1	-1	+1	+1
s_6	(6,	+1)	+1	-1	+1	+1
s_7	(7,	+1)	+1	-1	+1	+1
s_8	(8,	-1)	+1	-1	+1	+1
s_9	(9,	-1)	+1	-1	+1	+1
s_{10}	(10,	-1)	+1	-1	+1	+1

Bagging

- A technique that repeatedly samples (with replacement) m training data points according the uniform distribution
- Some data points may be repeated or omitted
- Sample set is expected to have 63% of items. Because each item has probability $1 - (1 - \frac{1}{m})^m$ of being selected that converges to $1 - \frac{1}{e}$ as m increases

Algorithm 10: Bagging

```
1 Let k be the number of bootstrap samples
2 for i = 1 to k do
3     Create D_i; a bootstrap sample of size m
4     Train classifier C_i on D_i
5 return C_i's
```

$$C(x) = \text{argmax}_y \sum_i \delta(C_i(x) = y) \qquad \delta(true) = 1; othewise\ 0$$

Bagging Example

Consider the following data set

	(x	y)
s_1	(0.1,	+1)
s_2	(0.2,	+1)
s_3	(0.3,	+1)
s_4	(0.4,	-1)
s_5	(0.5,	-1)
s_6	(0.6,	-1)
s_7	(0.7,	-1)
s_8	(0.8,	+1)
s_9	(0.9,	+1)
s_{10}	(1.0,	+1)

Let sampling gets data points as s_1, s_2, s_3, s_4, s_5, s_6, s_9, s_{10} Then

(x y)	0	.15	.25	.35	.45	.55	.65	.75	1.05
s_1 (0.1,+1)	+1	-1	-1	-1	-1	-1	-1	-1	-1
s_2 (0.2,+1)	+1	+1	-1	-1	-1	-1	-1	-1	-1
s_3 (0.3,+1)	+1	+1	+1	-1	-1	-1	-1	-1	-1
s_4 (0.4,-1)	+1	+1	+1	+1	-1	-1	-1	-1	-1
s_5 (0.5,-1)	+1	+1	+1	+1	+1	-1	-1	-1	-1
s_6 (0.6,-1)	+1	+1	+1	+1	+1	+1	-1	-1	-1
s_9 (0.9,+1)	+1	+1	+1	+1	+1	+1	+1	-1	-1
s_{10} (1.0,+1)	+1	+1	+1	+1	+1	+1	+1	+1	-1

Best classifier can be obtained by taking threshold at 0.35 and reversing the sign.

$$Classification = \begin{cases} +1 & \text{if } x \leq 0.35 \\ -1 & \text{otherwise} \end{cases}$$

Bagging Example

Similarly

	(x	y)
s_1	(0.1,	+1)
s_2	(0.2,	+1)
s_3	(0.3,	+1)
s_4	(0.4,	-1)
s_5	(0.5,	-1)
s_6	(0.6,	-1)
s_7	(0.7,	-1)
s_8	(0.8,	+1)
s_9	(0.9,	+1)
s_{10}	(1.0,	+1)

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.8	0.9	1	1	1
y	1	1	1	-1	-1	1	1	1	1	1

x <= 0.65 ==> y = 1
x > 0.65 ==> y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ==> y = 1
x > 0.3 ==> y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ==> y = -1
x > 0.05 ==> y = 1

Bagging Example

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

x <= 0.65 ==> y = 1
x > 0.65 ==> y = 1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

x <= 0.3 ==> y = 1
x > 0.3 ==> y = -1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

x <= 0.05 ==> y = -1
x > 0.05 ==> y = 1

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1
True Class	1	1	1	-1	-1	-1	-1	1	1	1

IS-ZC464: MACHINE LEARNING

Lecture-16: Deep Learning

Dr. Kamlesh Tiwari
Assistant Professor
Department of Computer Science and Information Systems,
BITS Pilani, Pilani, Jhunjhunu-333031, Rajasthan, INDIA

November 10, 2018 (WILP @ BITS-Pilani Jul-Nov 2018)

Deep Learning (DL)

An approach to learn data using neural network

Capable of solving very complex problems

- ImageNet Challenge (2011): identify objects in images (millions)
- Generate audio: temporal dependence

DL learns underlying features/representations directly from data.

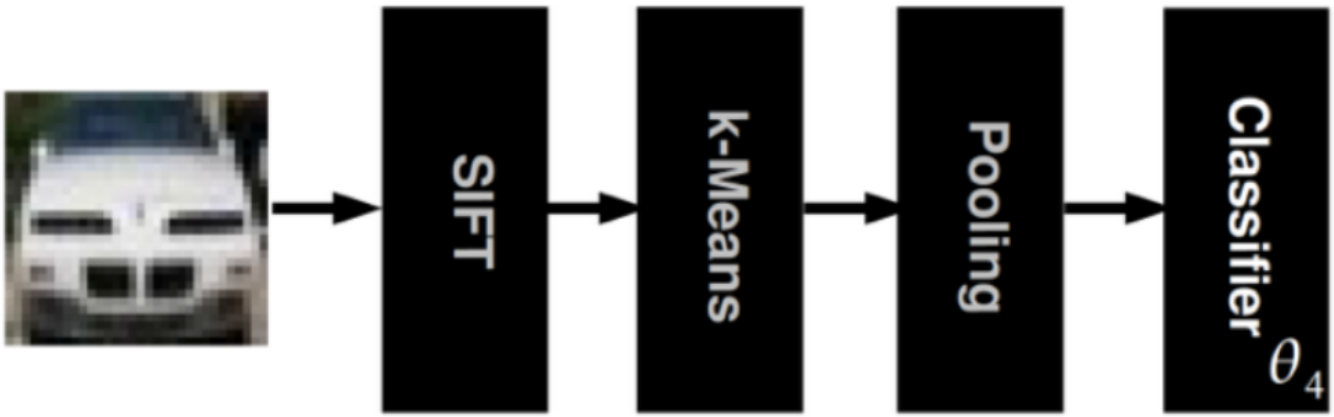
- Traditional way to hand engineer feature is brittle and not scalable
- DL involves **NO** low, mid and high level feature pipeline
- **Why now?** because we have data (imageNet), compute power (GPUs) and software (tensorFlow)
- Deep learning is NOT just adding more layers to a neural network.

Deep Learning

- **Shallow to Deep:** is linear combination to composition

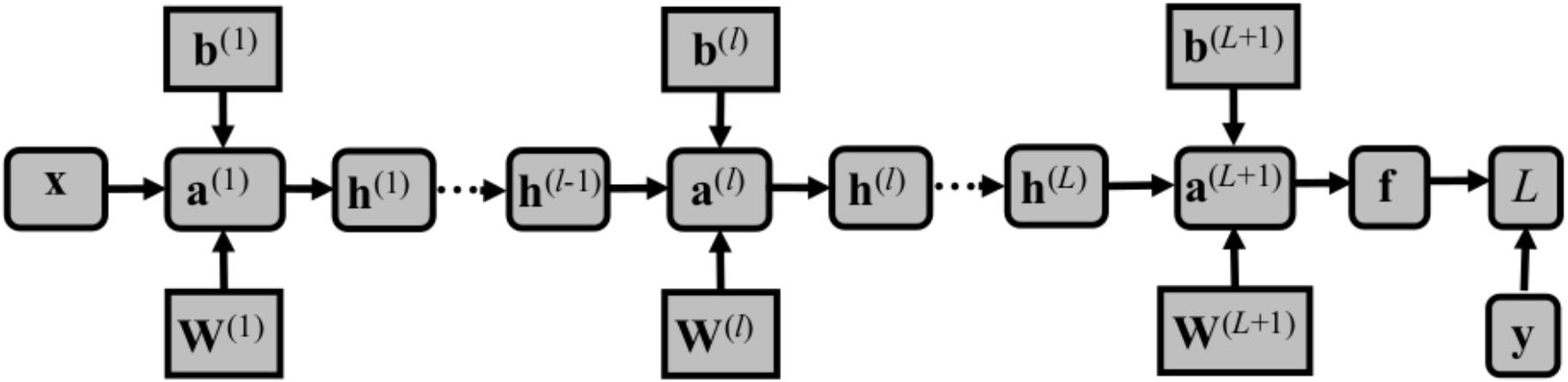
$$\sum_i g_i \quad \rightarrow \quad g_1(g_2(g_3(...)))$$

- Why?



- Optimization is difficult for a non-convex, non-linear systems
- Freezing some layers makes it shallow.

RNN

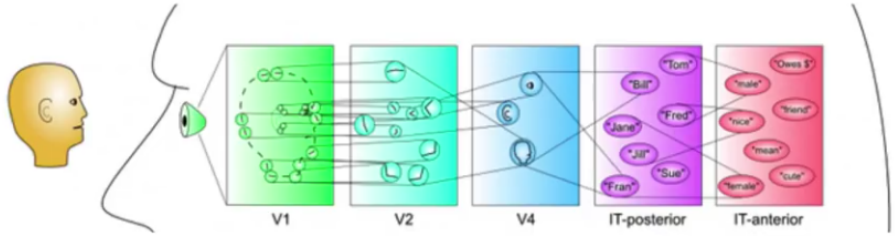


- Loops in networks allow information to persist[1]
- Applied to speech recognition, language modeling, translation, image captioning ...
- Long-Term Dependencies may have a problem

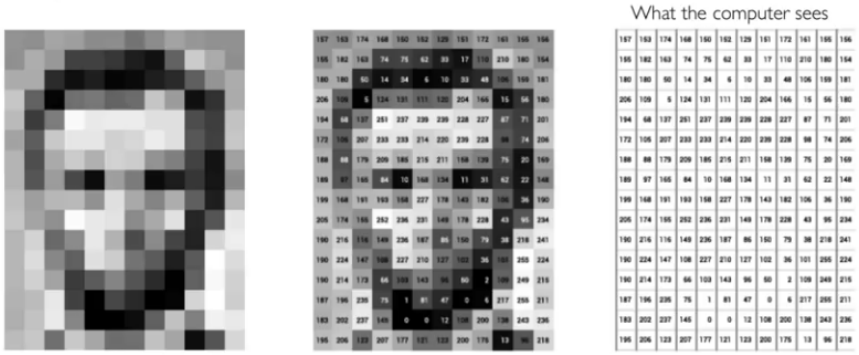
[1] See: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Visual Cortex is Hierarchical

- Visual Cortex is Hierarchical



- Images are numbers (so determine feature)

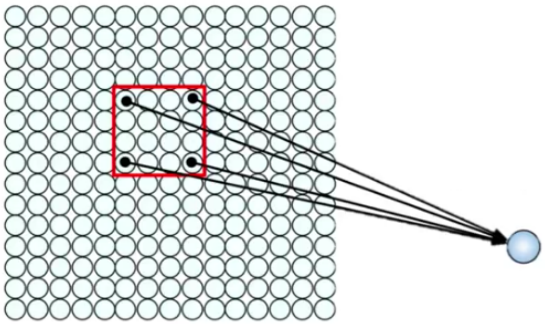


There could be occlusion, viewpoint variation, scale variation, illumination variation, deformation, background clutter, noise, intra-class variation and much more.

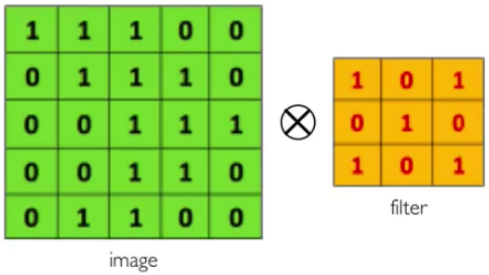
DL helps to get hierarchy of features

Vision

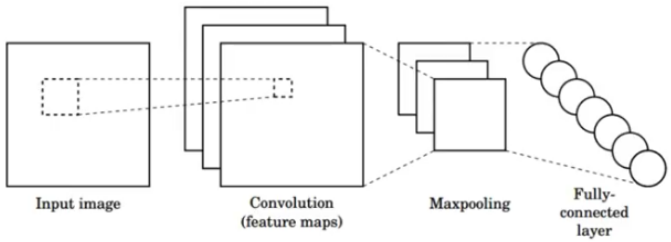
- Show the patches of image



- Convolution (apply stride, padding and pooling)

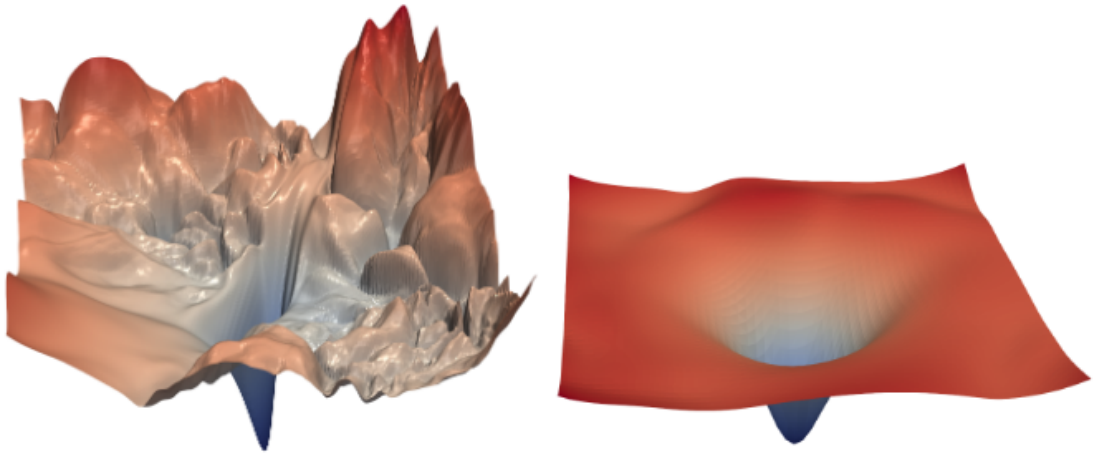


- CNN



Regularization

- Dropout[2]



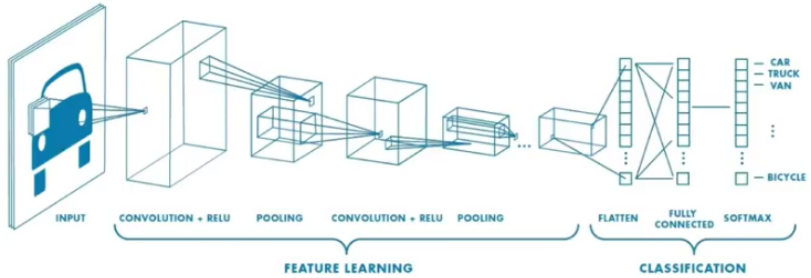
(a) without skip connections (b) with skip connections

- Early stopping

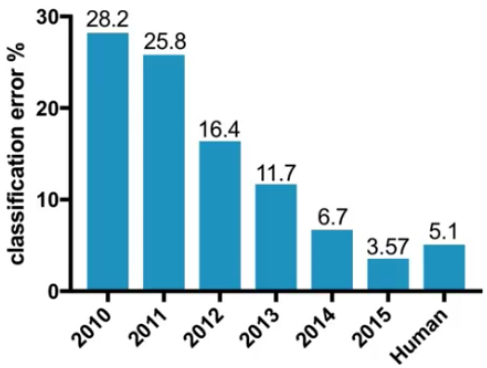
[2]Visualizing the Loss Landscape of Neural Nets, Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, Tom Goldstein, 2018

Vision

- CNN for classification



- ImageNet: 4 million images in 21841 categories (Challenge 1000 categories)



2012: AlexNet. First CNN to win.
- 8 layers, 61 million parameters

2013: ZFNet
- 8 layers, more filters

2014:VGG
- 19 layers

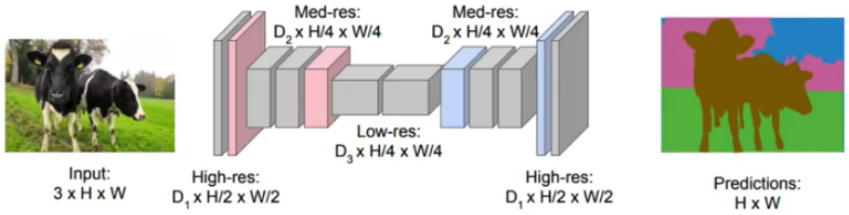
2014: GoogLeNet
- "Inception" modules
- 22 layers, 5million parameters

2015: ResNet
- 152 layers

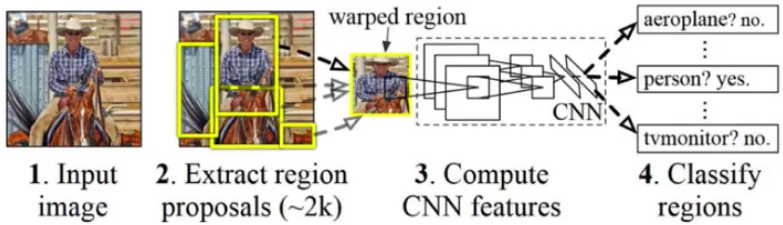
- More to do

Vision

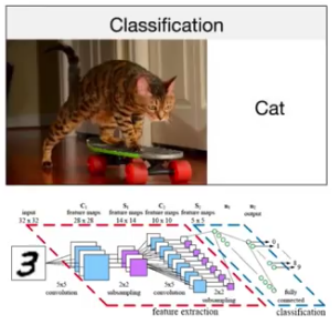
- Segmentation

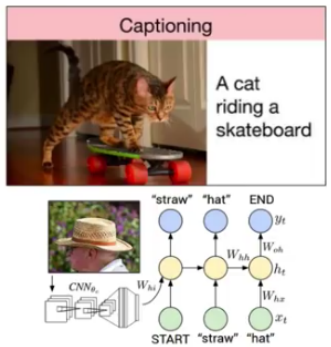


- Object Detection (RCNN)

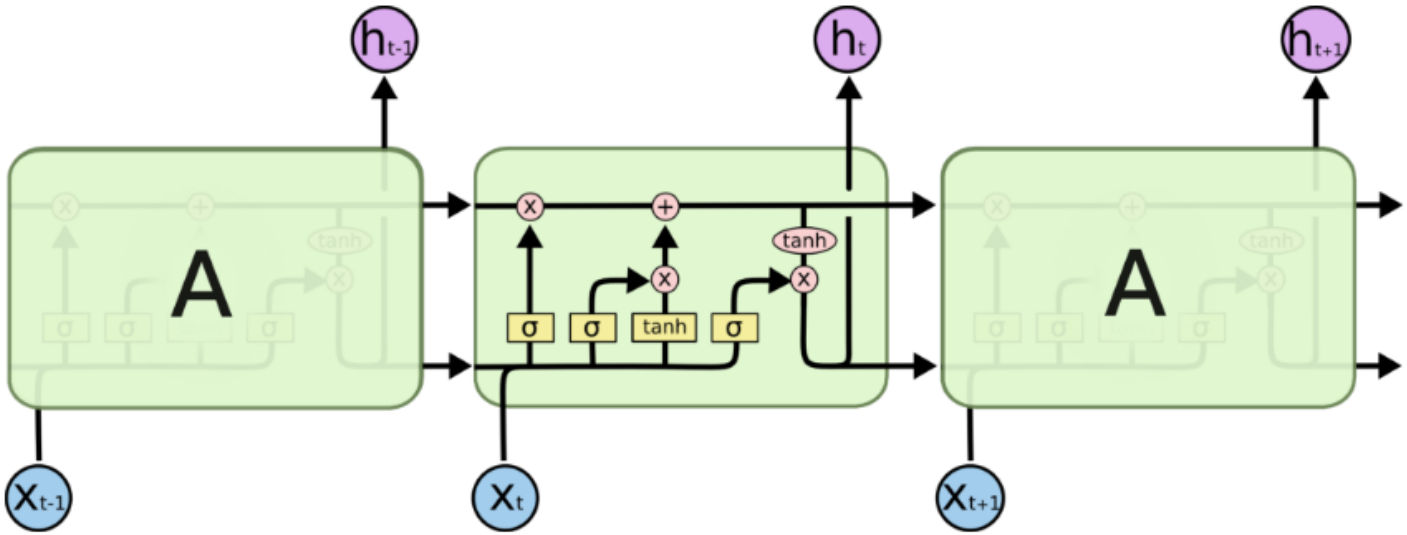


- Image Captioning



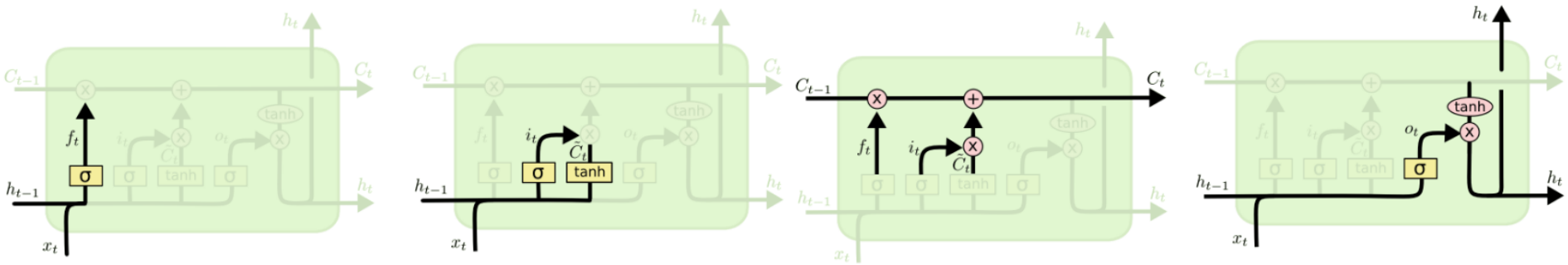


Long Short Term Memory (LSTM)



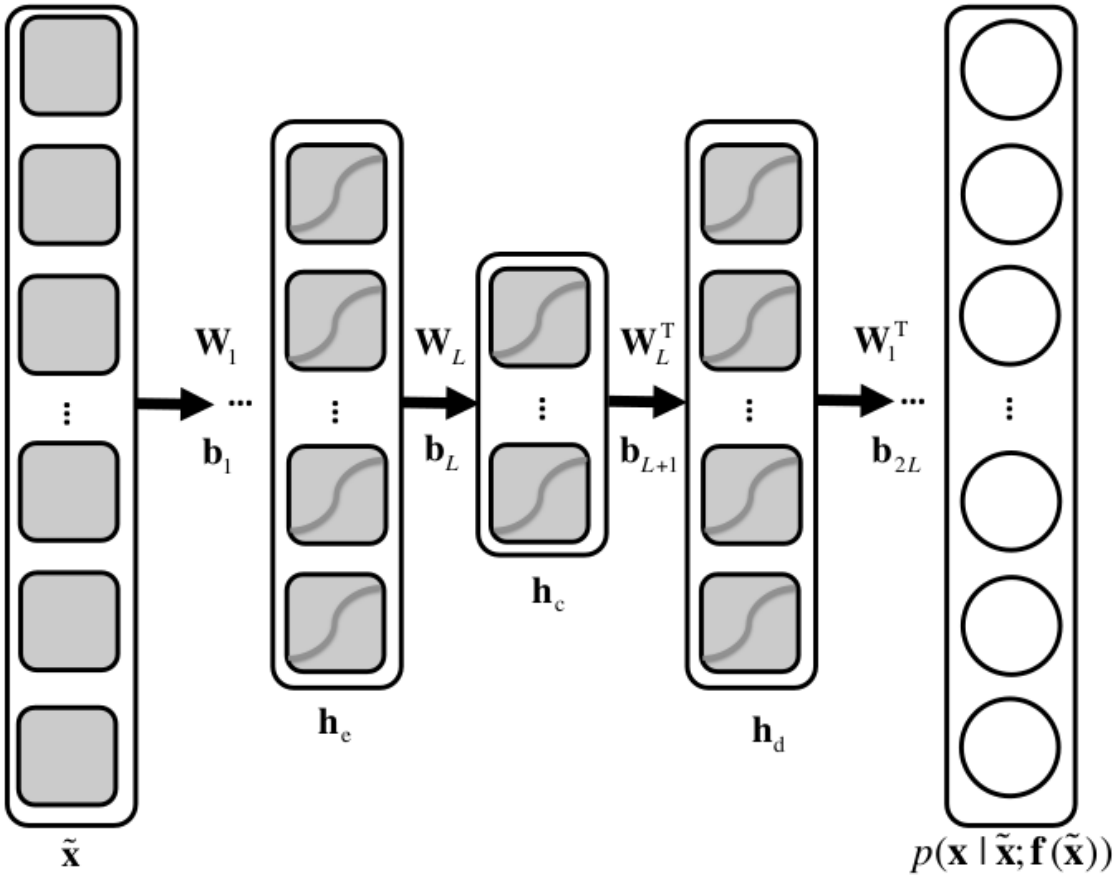
- RNNs capable of learning long-term dependencies
- Uses a state variable
- There are four neural network layer
- Three **gates**, can let the value through or block

Long Short Term Memory (LSTM)



- What to forget? $f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$
- $i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$ $\quad \hat{C}_t = tanh(W_c[h_{t-1}, x_t] + b_c)$
- What to store? $C_t = f_t \times C_{t-1} + i_t \times \hat{C}_t$
- What to output? $o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$ $\quad h_t = o_t \times tanh(C_t)$
- Output is a filtered version.
- LSTMs work a lot better for most tasks
- Next thing: Attention (can focus on relevant part or subset of data)

Autoencoders

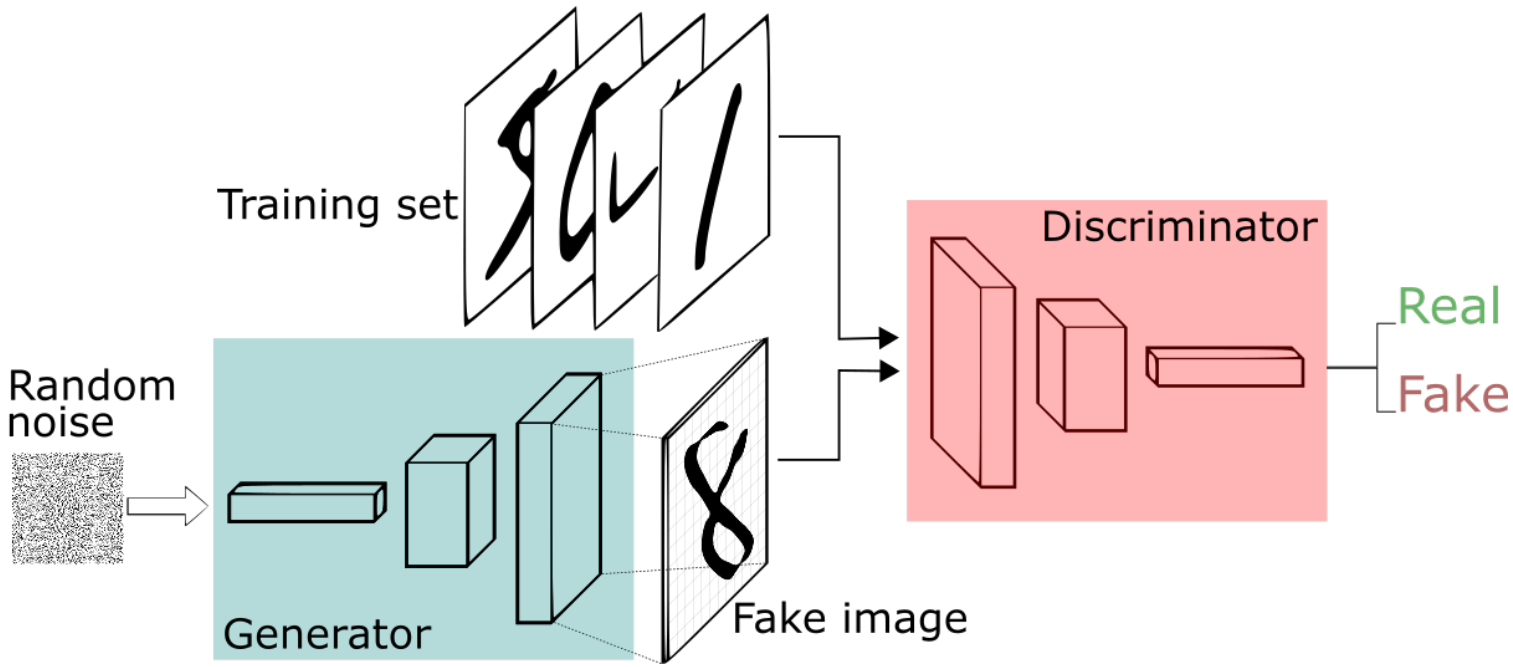


- An unsupervised learning algorithm, setting the target values to be equal to the inputs
- Learns an approximation

Generative Adversarial Networks (GAN)

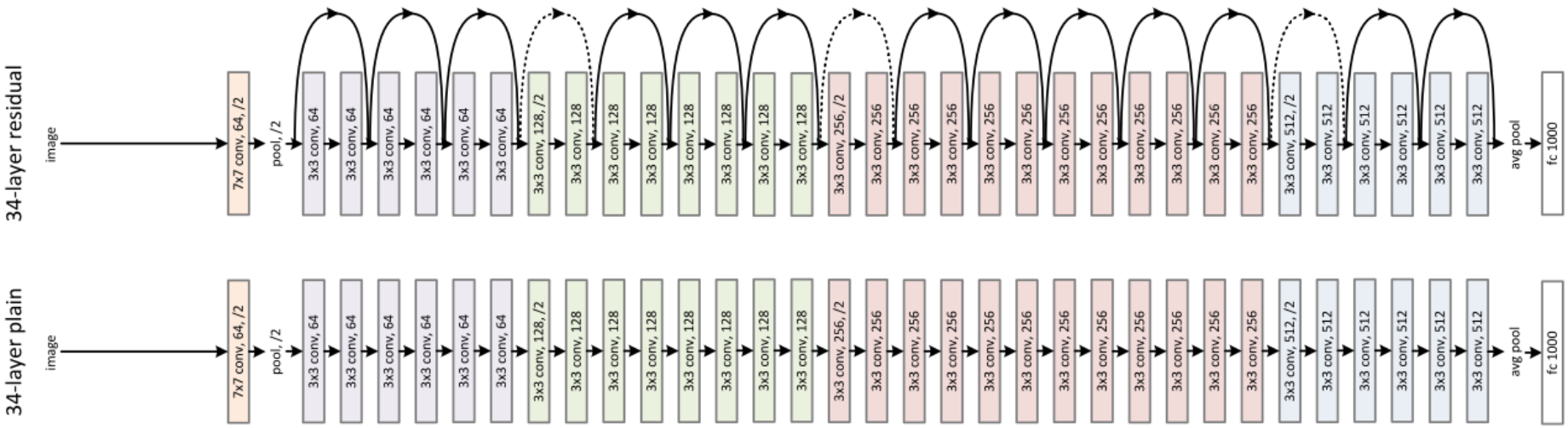
GAN has two NN generator and discriminator

- Worst possible training by min-max game
- generator wish to fool discriminator
- Underline distribution plays a role



Residual Networks (ResNets)

- It's always beneficial to have a deeper network.
- But, going deeper doesn't always help. The network becomes harder to train because of the vanishing gradient problem.
- Add skip connection. This would be at least as good as the shallower network.
- If the residual network learns something, it would be an add-on.

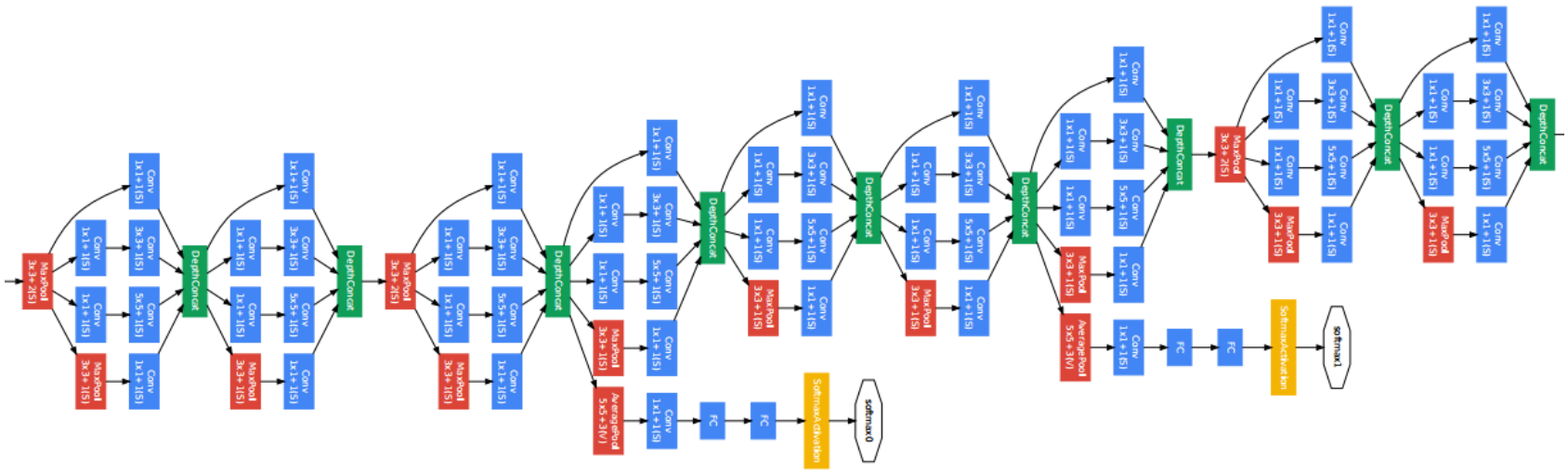


Inception Network

Have multiple filter n/w at the same level. Make the network wide

- GoogLeNet was the winner of the ImageNet Large Scale Visual Recognition Competition 2014. Also called Inception v1.
- 22 conv layers and average pooling at end of each last module.
- Two auxiliary classifiers to prevent the vanishing gradient problem
- Total loss is the weighted sum of the auxiliary losses and real loss.

$$Total\ \ Loss = Real\ \ Loss + 0.3 \times (Aux\_Loss1 + Aux\_Loss2)$$



Cognitive Computing

Brain is modeled using hierarchy of subsystems with different activation time and accuracy.

Brain Regions	Function	Operations	Type of Process
Amygdala	Detecting, Learning about Stimuli	Detects potentially threatening stimuli and associates them with appropriate actions	Automatic
Basal Ganglia	Registering Rewards, Acquiring Habits, Selective gating of behavior	Mediates selection and initiation of actions, Automatizes sequences of behavior and reinforced thoughts	Automatic or controlled
Lateral prefrontal/ association cortices	Retrieving and Storing Semantic Emotion Knowledge	Identifies stimuli, differentiates feeling states; attributes emotional qualities to stimuli; repository of regulatory strategies, lay emotion knowledge	Retrieval can be automatic or controlled
Anterior Cingulate Cortex	Conflict Monitoring	Monitors on-going behavior and determines whether a change is necessary or not	Conflicts detected automatically, but making changes takes control
Ventral/ Medial Orbital Frontal Cortex	Context- dependent action selection	Inhibits on-going emotional responses based on analyses of context	Controlled
Hippocampus	Long-term strategies	Understanding spatial relations within the environment	Controlled

Higher level does complex and controlled mechanisms, **lower level** does fast automatic process.