Machine Learning
Baruch College
Lecture 5

Miguel A. Castro

Today

- Announcements:
  - Project Proposals are due today, Wednesday April 29.
  - Final Exam is next Wednesday May 6th.
  - Project Presentation is Wednesday May 13th.
- Brief Review of Association Rules & Time Series.
- Survey of Various Useful Topics.
- Project and Exam questions…

Association Rules

- *Association Rules* (also known as Market Basket Analysis) are an example of a Nonparametric Technique, and of Unsupervised Learning.
- Recall the database of transactions $\mathcal{D}$ where 0/1 represents the presence/absence or occurrence/non-occurrence (respectively) of a Generalized Item:

$$\mathcal{D} \quad \begin{array}{c|ccccccc|}
 & i_1 & i_2 & i_3 & i_4 & \cdots & i_{n-1} & i_n \\
\hline
d_1 & 0 & 1 & 1 & 0 & \cdots & 0 & 1 \\
d_2 & 0 & 0 & 1 & 0 & \cdots & 1 & 0 \\
d_3 & 1 & 0 & 1 & 0 & \cdots & 0 & 0 \\
d_4 & 0 & 0 & 0 & 0 & \cdots & 1 & 0 \\
\vdots & & & & \vdots & & & \\
d_{m-1} & 1 & 1 & 0 & 1 & \cdots & 0 & 1 \\
d_m & 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\
\hline
\end{array}$$

- Association Rules are of the form "If X then Y" $(X \Rightarrow Y)$, where X and Y are disjoint subsets of the Generalized Items $\mathcal{I} = \{i_1, i_2, \dots, i_n\}$.
- Problem: *"Find Interesting Association Rules $X \Rightarrow Y$ from $\mathcal{D}$ in an automated way."*
- Problem: The Number of possible items (occurrences) can be very large, and the number of transactions can be huge; checking all combinations is intractable.

Association Rules

- Example.: X = "*INTC price went down and Energy Industry went up* at time t_1." $Y =$ "*AMAT price went down at time* $t_2 > t_1$."
- X is the *Antecedent* and Y is the *Consequent*.
- The *Support-Confidence Framework*, is commonly used, but it doesn't work, as it picks up spurious (random) rules.
- We prefer the *Dependency Framework* because it focuses on rules that deviate from random associations.
- Interesting Association Rules are *Actionable*.
- Interesting Association Rules are *Statistically Significant* and have a high *Strength of Dependence*.
- For Statistical Significance we use the χ^2 *Statistic*.
- For Strength of Dependence we use the Information-Theoretic *Dependency Coefficient*.

Association Rules

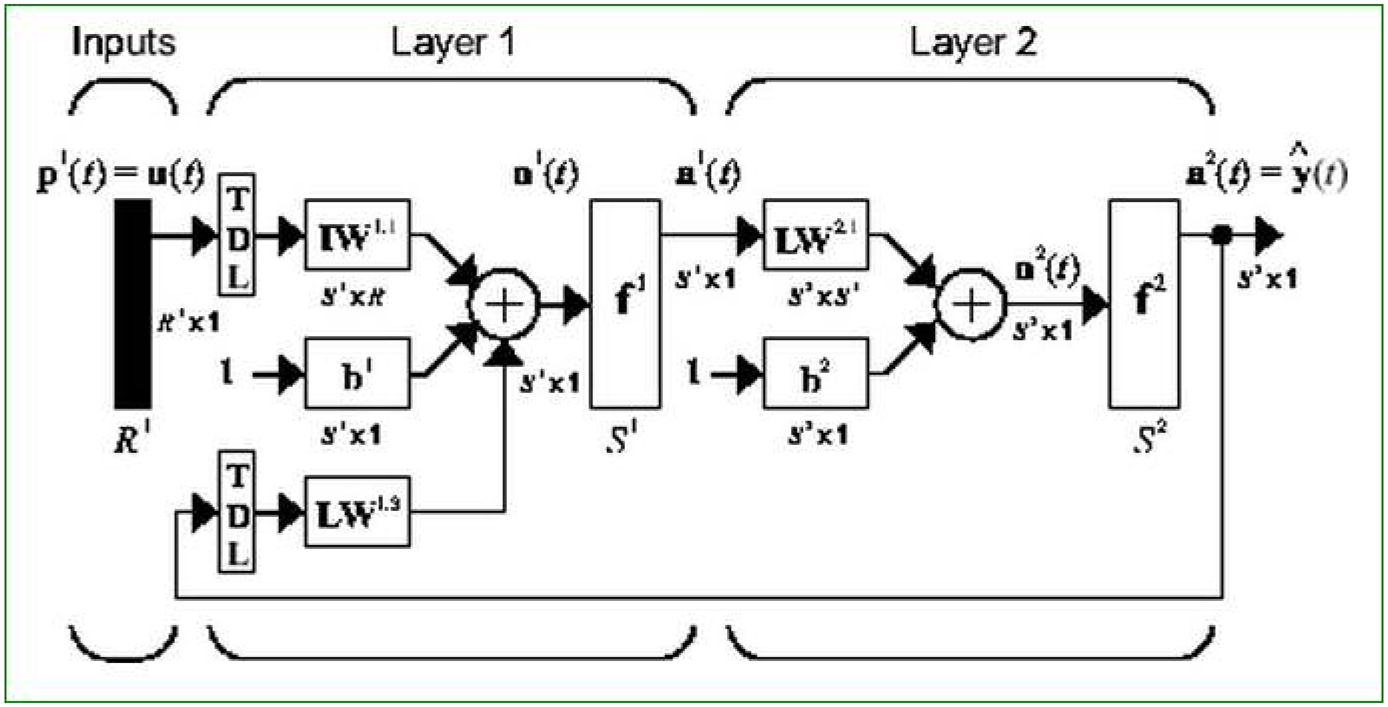
- The number of possibilities to check is intractable.
- However, an exhaustive search is unnecessary.
- The search can be limited to the <u>*Region of Interest*</u> which is far smaller and is constrained by:
  - The *External Closure* of the χ^2 statistic, and
  - The *Internal Closure* of Minterm Support.
- (Recall that Minterm Support was required to ensure the robustness of the χ^2 test.)
- Therefore, we only consider Itemsets to be interesting if they are Dependent at the $1 - \alpha$ level and if they are Minterm-Supported with $s = 5$ and $p = 80\%$.
- The Dependency Coefficient has *no closure relation*, and is used to rank the rules <u>*after*</u> they've been generated.

Association Rules

- Other ways of managing computational complexity are *Sampling* and *Partitioning* of the database of transactions $\mathcal{D}$.
- Sampling is done by randomly selecting transactions *with replacement*.
- Partitioning is done by randomly assigning transactions to *disjoint* and *spanning* partitions of $\mathcal{D}$.
- Both of these methods pay a price in accuracy, which can be quantified by various useful *error bounds*, as we saw earlier.
- Another way to manage complexity is by *a priori* selecting interesting and actionable consequents, and finding corresponding antecedents.
- But this has the disadvantage of removing automated discovery, and possibly missing out on surprising rules.
- Itemset "*Roll-Ups*" or *Taxonomies* (*e.g.:* Industry Classifications, Market Cap, etc.) can be incorporated in the search for interesting rules.
- The search for rules with Taxonomies can be made tractable by exploiting *duality relations* when building the search algorithms.
- Association Rules concepts are straightforward (mostly vocabulary), but they can be quite powerful.
- Questions about Association Rules?

Dynamic ANNs and Time Series Prediction

- We can use ANNs to approximate a time series by treating the time series as a single input pattern, but introducing Delays ($\mathbf{D}$) via a *Tapped Delay Line* (TDL), and optionally by feeding the output back into the input layer via another TDL, as in *Recurrent or Closed-Loop Networks*:



- The inputs can include a "*Companion*" or "*Regressor*" series $\boldsymbol{x_t}$, the *Target* series itself, $\boldsymbol{y_t}$, and/or the *Output* series $\boldsymbol{\hat{y}_t}$.
- This can approximate a general time series of the form:

$$\boldsymbol{y_t} = f(\boldsymbol{x_{t-1}}, \dots, \boldsymbol{x_{t-r}}, \boldsymbol{y_{t-1}}, \dots, \boldsymbol{y_{t-m}}, \boldsymbol{\hat{y}_{t-1}}, \dots, \boldsymbol{\hat{y}_{t-n}}),$$

where f is the (possibly non-linear) function to be approximated.

- For example, the target can be "Exchange Rate" and regressors can be "Oil Price," "T-Bill Rates" etc.

Dynamic ANNs and Time Series Prediction

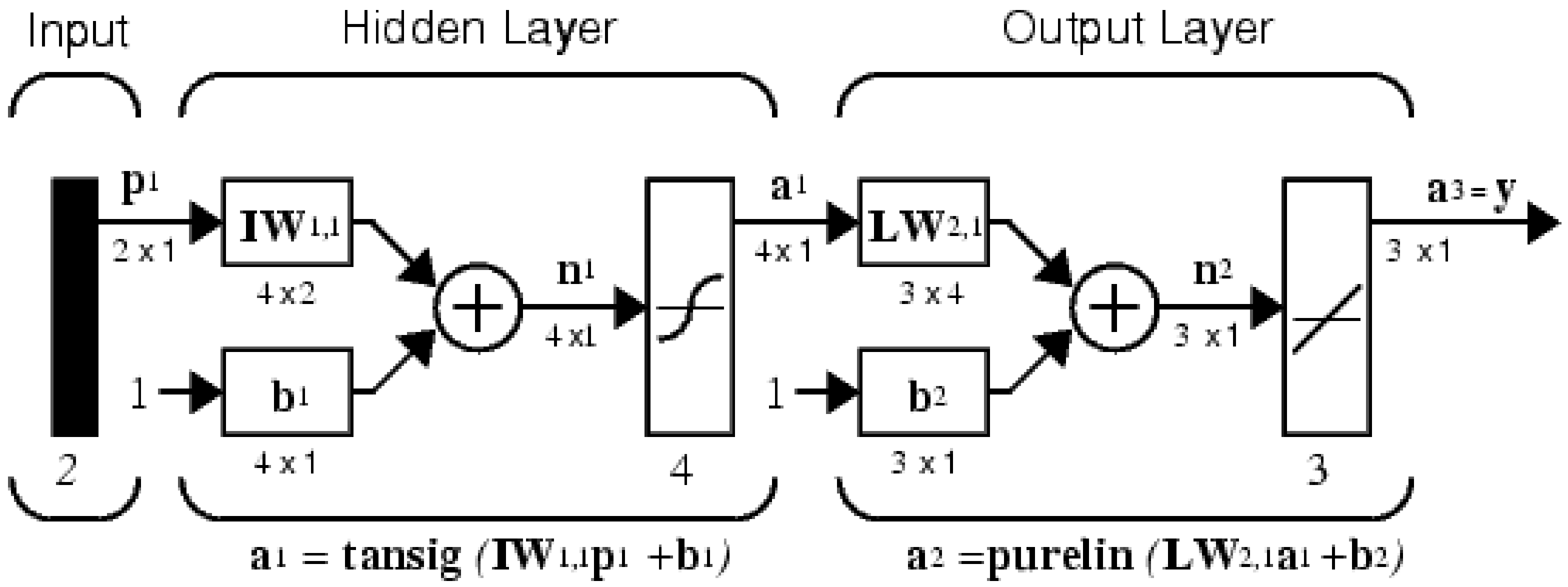
- This architecture can be easily modified to replicate an ARMA-plus-regressors model of the form:

$$y_t = \underset{\substack{\uparrow \\ \textbf{AR Term}}}{\sum_{j=1}^{p} \varphi_j y_{t-j}} - \underset{\substack{\uparrow \\ \textbf{MA Term}}}{\sum_{k=1}^{q} \theta_k \varepsilon_{t-k}} + \underset{\substack{\uparrow \\ \textbf{Regressor Term}}}{\sum_{l=1}^{r} \beta_l x_{t-l}}\,,$$

where the ε_t are innovations of the form $\varepsilon_t = y_t - \hat{y}_t$.

- This can be done by using ADALINEs in place of the activations f_1 and f_2, and with the appropriate set of TDLs.
- As an aside, note that linear models are very common in financial time series modeling.
- This is because any nonlinearities are usually difficult to capture due to the very low signal-to-noise.
- Questions?

Recall Feedforward NN Architecture



- The FNN is a Universal Mapping Approximator.
- We can solve many problems with it.
- Note: Hidden layer activity is ***distributed***.
- *I.e.*: For any input pattern, many neurons (nodes) could "fire," (meaning that they could have significant activation values).
- Next, we consider a *localized* neuron approach…

Radial Basis Functions

- On the other hand, consider a ***localized*** approach.
- Instead of distributed nodes, use ***Local nodes***, where each node activates in response to inputs in a particular region of the input space.
- One or very few adjacent nodes fire for any given input pattern.
- One or very few nodes are assigned to respond to a given input pattern.
- "In-between" patterns cause neighboring nodes to partially fire.

Radial Basis Functions

- Suppose we think of the node weights as points in weight space.
- Assume inputs and outputs are normalized, and ignore the bias weights, for the moment.
- We can plot these “node points” alongside “input points” in the same space, because they have the same dimensions.
- We are now free to define a “distance” between a node’s weights and an input pattern:

$$\boldsymbol{d} = \|\mathbf{x} - \mathbf{w}\|_{\mathbf{2}}.$$

where $\|\cdot\|_2$ is a 2-norm such as Euclidean Distance.

Radial Basis Functions

- For localized node behavior, we want the node’s activation to be high when the distance between input and node is small, and vice-versa.

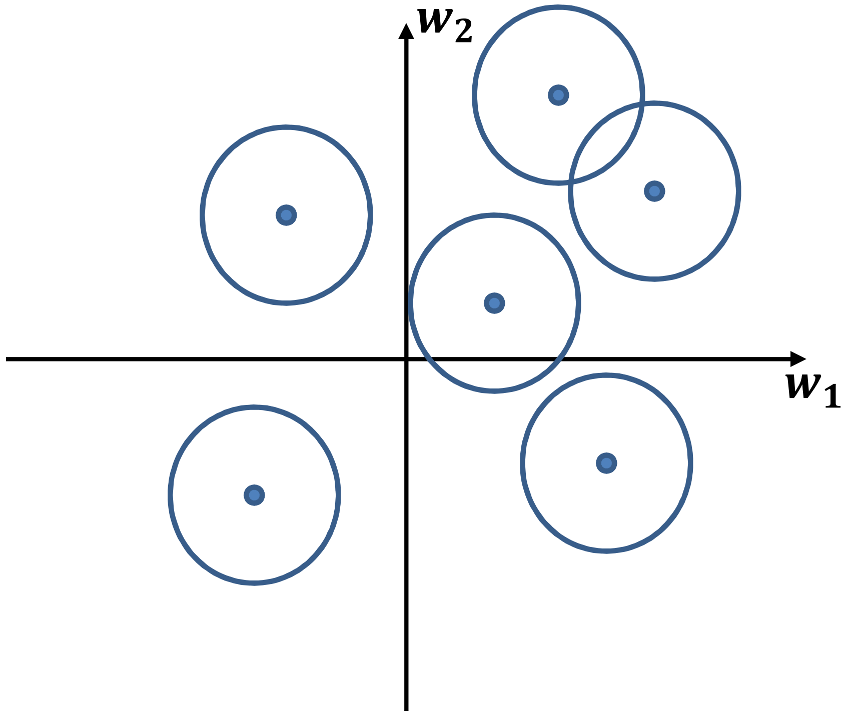
- There are many activations that would work, but a common one is the Gaussian:

$$\mathcal{G}(\mathbf{x}, \mathbf{w}, \sigma) = \exp\left(\frac{-\|\mathbf{x} - \mathbf{w}\|^2}{2\sigma^2}\right).$$

- Highest when $d = 0$, tapers off quickly depending on σ.

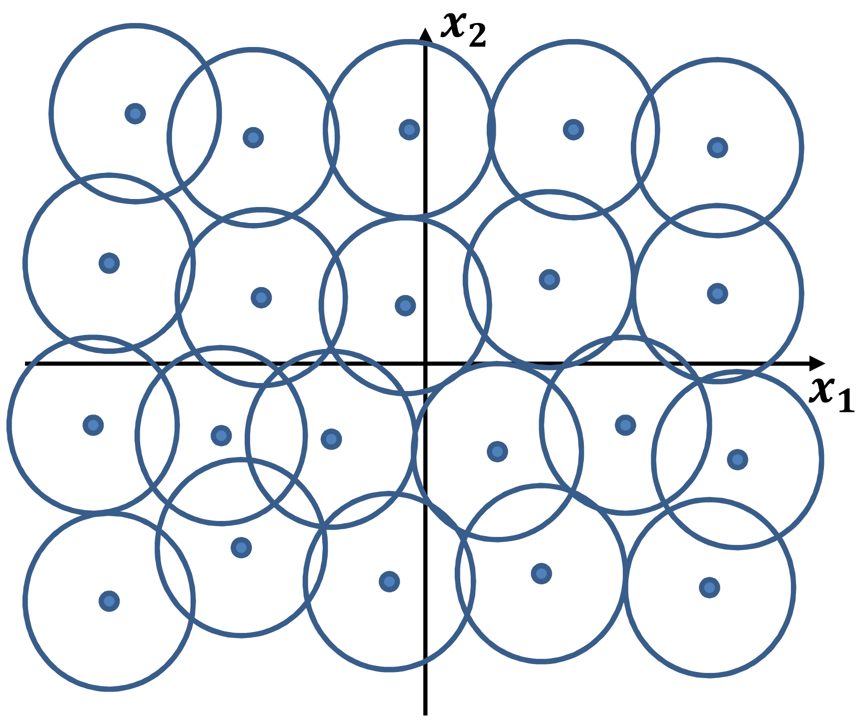
Radial Basis Functions

- Activation is non-negligible within an *effective radius*, and negligible outside:



Radial Basis Functions

- For universal approximation feature, we want to span the input space:



Radial Basis Functions

- Nodes form, effectively, a *basis* for the input space (*i.e.*, they span the input space).
- Activations are non-negligible within an effective radius.
- Hence the name "*Radial Basis Functions*."
- To span the input space, it's common to choose the Gaussian width as follows:

$$\sigma = \frac{d}{\sqrt{2N}},$$

where d is a typical spacing, and N is the number of nodes.

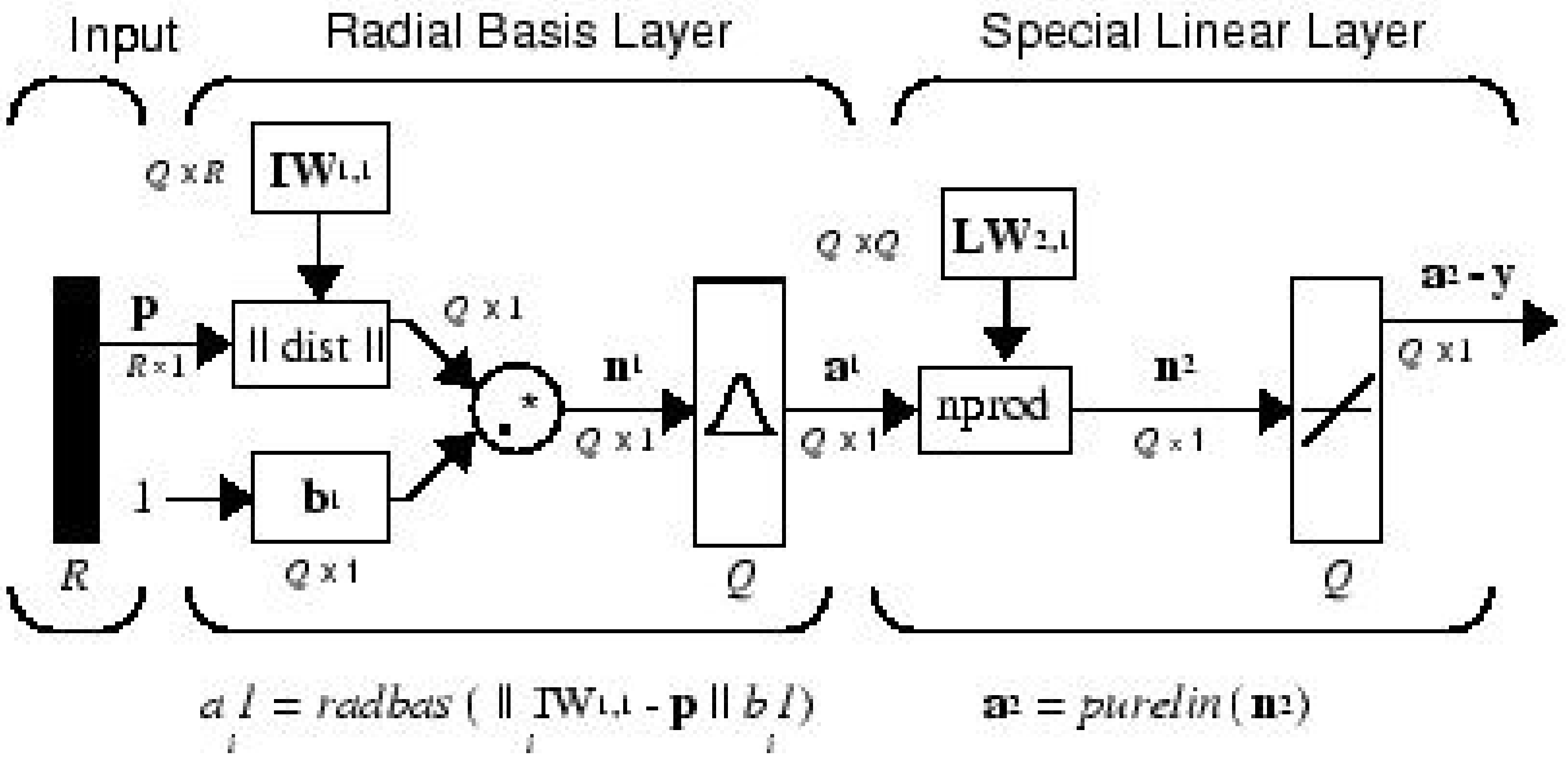
Radial Basis Functions

- Despite our best efforts to span the input space, a new input point could fall outside the effective node space.
- To avoid this, use Normalized Gaussians of the form:

$$\mathcal{G}(\mathbf{x}, \mathbf{w}, \sigma) = \frac{exp(-\|\mathbf{x} - \mathbf{w}\|^2/2\sigma^2)}{\sum_i exp(-\|\mathbf{x} - \mathbf{w}_i\|^2/2\sigma^2)}.$$

- Even for a far-away input, the node closest to it will fire.

Radial Basis Functions: NN Architecture



- Similar to FNN: It is a Universal Approximator.
- However, only one Hidden Layer is permitted.
- Activation based on “distance” instead of “dot-product” as in FNNs.

Radial Basis Functions: Training

- Training can be achieved by BackPropagation, as with regular FNNs.
- However, there are better (faster) alternatives to BackProp.
- Note that hidden-layer localized nodes and output nodes perform different functions and don't have to be trained simultaneously.
- Training can be done piecemeal by:
  - Positioning RBFs to be "representative" of the input space.
  - Using RBF activations to train the linear output layer.

Radial Basis Functions: Training

- Position RBFs by:
  - Randomly assigning centers within input space; or
  - Random sampling of the input space itself; or
  - A more refined method such as clustering (*e.g.* k-means clustering).
- Compute RBF activations (after selecting RBF widths, normalization, etc.)
- Train output layer by:
  - Using our old delta learning rule; or
  - “Inverting” the RBF activations…

Radial Basis Functions: Training

- If we let $\boldsymbol{\mathcal{G}}$ be the matrix of RBF activations such that $\boldsymbol{\mathcal{G}}_{ij}$ is the j^{th} node activation given the i^{th} input pattern:
- We can write the network output $\mathbf{y}$ as:

$$\mathbf{y} = \boldsymbol{\mathcal{G}}\mathbf{W},$$

where $\mathbf{W}$ is the matrix of output node weights, that we are trying to find.

- We want $\mathbf{W}$ to approximate the target outputs, $\mathbf{t}$, so we set:

$$\mathbf{t} = \boldsymbol{\mathcal{G}}\mathbf{W}.$$

Radial Basis Functions: Training

- This leaves us with the problem of inverting the activation matrix $\boldsymbol{\mathcal{G}}$ to solve for $\mathbf{W}$:

$$\mathbf{W} = \boldsymbol{\mathcal{G}}^{+}\mathbf{t}$$

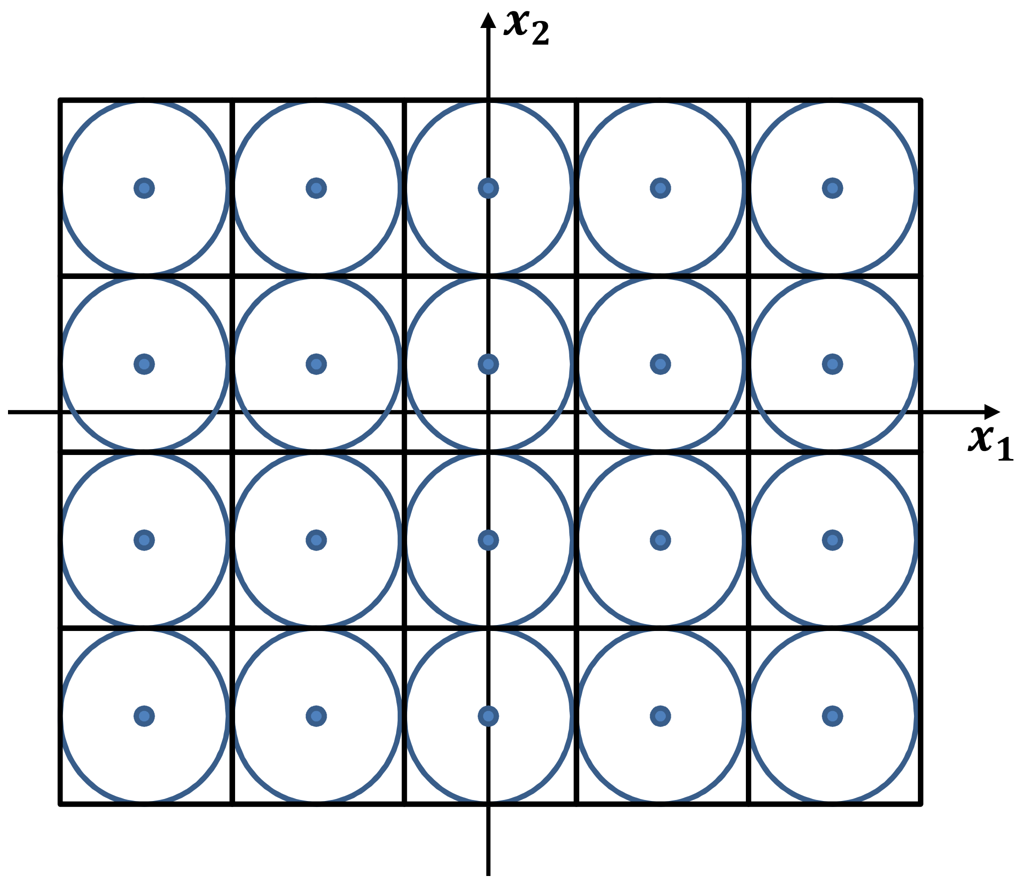
where $\boldsymbol{\mathcal{G}}^{+}$ is the pseudo-inverse of $\boldsymbol{\mathcal{G}}$:

$$\boldsymbol{\mathcal{G}}^{+} = (\boldsymbol{\mathcal{G}}^{T}\boldsymbol{\mathcal{G}})^{-1}\boldsymbol{\mathcal{G}}^{T}.$$

- We need the pseudo-inverse (instead of the inverse) because, in general, the number of hidden nodes is not equal to the number of input patterns.
  - In fact, we hope the number of hidden nodes is much less than the number of inputs; otherwise we may face severe overfitting.
  - This requires $\boldsymbol{\mathcal{G}}^{T}\boldsymbol{\mathcal{G}}$ to be non-singular.
- Note: this is faster than BackPropagation.
- Note: this is equivalent to a least-squares fit (*why?*).

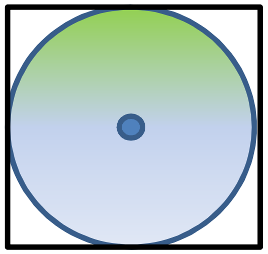
Radial Basis Functions: Training

- This is simpler and much faster learning.
- Problem: ensuring that we span the input space:



Radial Basis Functions: Coverage

- Must fill-in the gaps that hyper-spheres leave within hyper-cubes by overlapping the hyper-spheres.
- Note that coverage is perfect in 1-Dimension (100%), and it's not bad in 2-D:
  - Ratio of circle to enveloping square area is $\pi/4$.

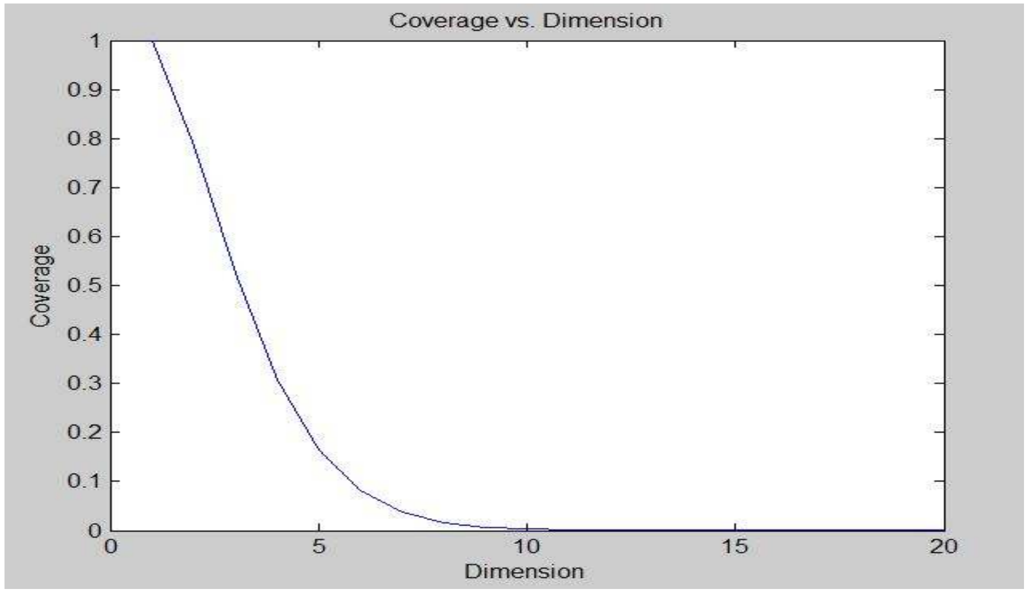


- But what happens to coverage in higher dimensions?

Radial Basis Functions: Coverage

- Coverage c_n of a hypersphere of volume v_n in relation to a hypercube in n dimensions is:

$$c_n = \frac{v_n}{2^n}, \qquad v_n = \frac{2\pi}{n} v_{n-2}:$$



- Pretty bad after a few dimensions! → *Curse of Dimensionality*.

Radial Basis Functions

- Other activations, besides Gaussian, are possible.
- In general, hidden-layer outputs can be of the form:

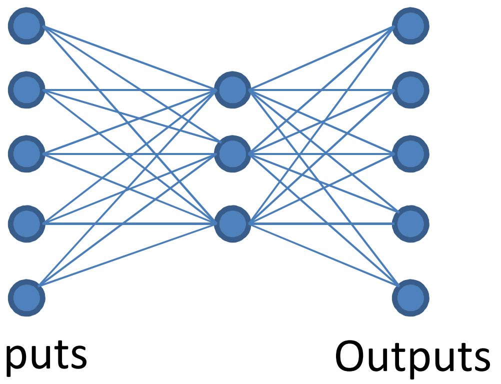
$$f(\mathbf{x}) = \sum_i w_i \psi_i(\mathbf{x}),$$

where the activations ψ_i are Basis Functions.

- Common basis functions are polynomials, particularly cubic polynomials, which are equivalent to *Cubic Splining*.

Aside: Auto-Associative Networks

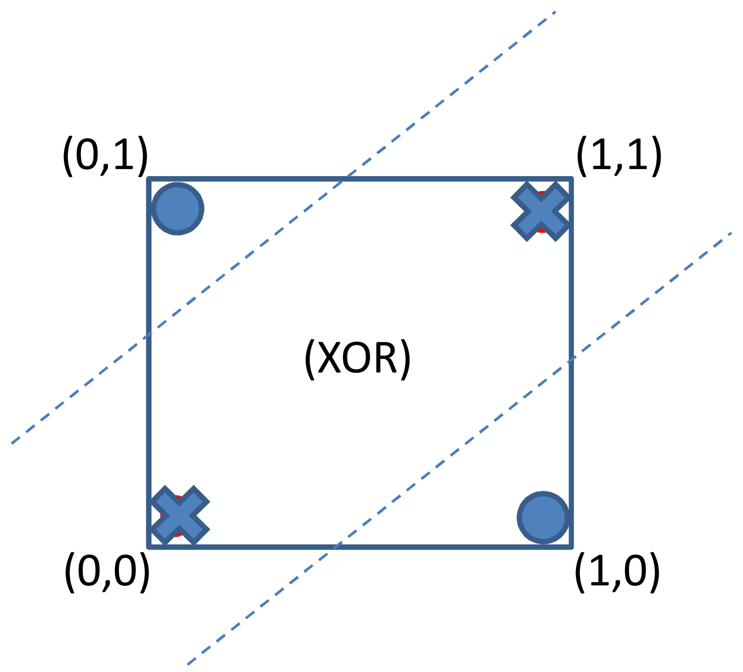
- A network that is trained to *reproduce the inputs*.



- Want outputs to match inputs as closely as possible.
- What’s the use of this?
- *Data compression* when the hidden layer is smaller than the input dimension (as in the illustration above).
- Used in image compression, sound compression, de-noising, etc.
- One can show this performs PCA-like feature reduction.

Remember Perceptron

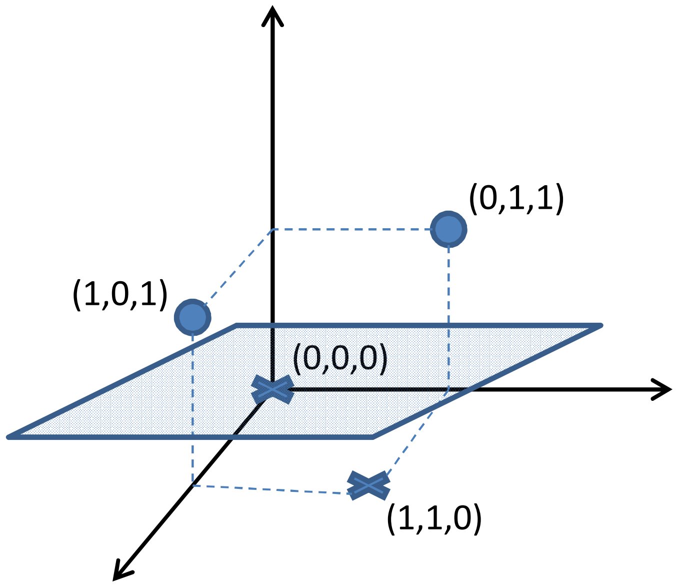
- The Perceptron could not solve the XOR problem (a Non-Linearly Separable problem).



- We built a Multi-Layer Perceptron to manually "solve" the problem by, effectively, *increasing the dimensionality* of the input space.

Increasing Dimensionality

- If we increased the Input Dimension in a certain way:



- We can now solve this problem using a Single Plane that separates the two classes; *i.e.*, the problem becomes ***Linearly Separable in the higher dimension***.

Increasing Dimensionality for Linear Separability

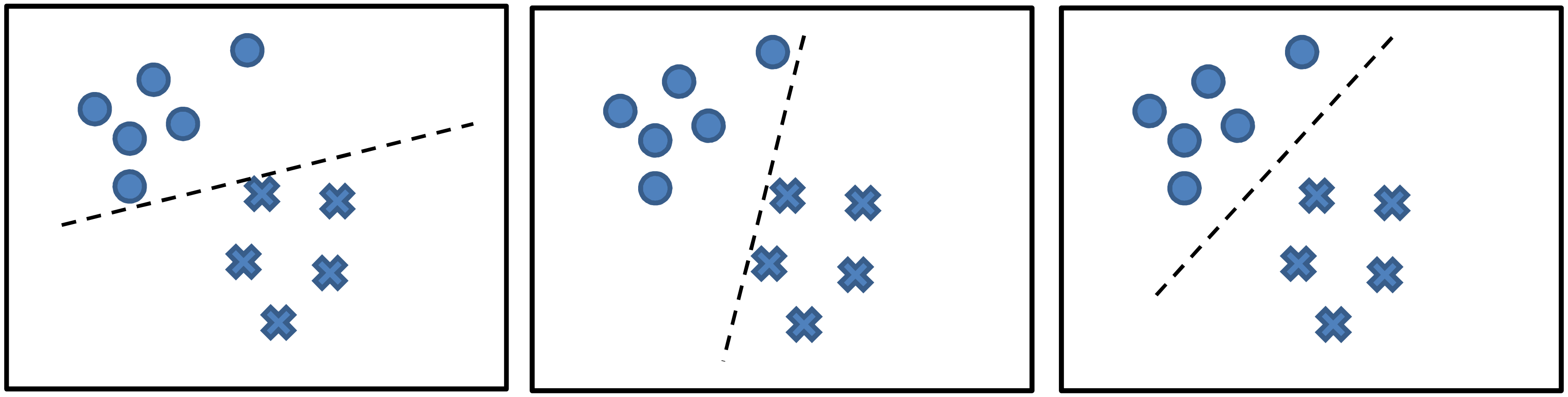
- We can show that every problem that’s not linearly separable in a given dimension, is guaranteed to be linearly separable *in a higher dimension*.
- At worst, we can introduce an extra dimension for every class to guarantee Linear Separability (though this is usually overkill).
- We can refer to the minimum number of dimensions required for linear separability as the <u>*Linear Separability Index*</u>*.*
- The Linear Separability Index is always less than or equal to one plus the number of classes.
- When they are equal, we can say that the input space has <u>*Maximal Linear Complexity*</u>*.*

Increasing Dimensionality for Linear Separability

- The idea would be to do this in an ***automated*** way.
- The problem is to work out which dimensions to use to separate the input data.
- Are there optimal ways of separating the classes?

Optimal Separability

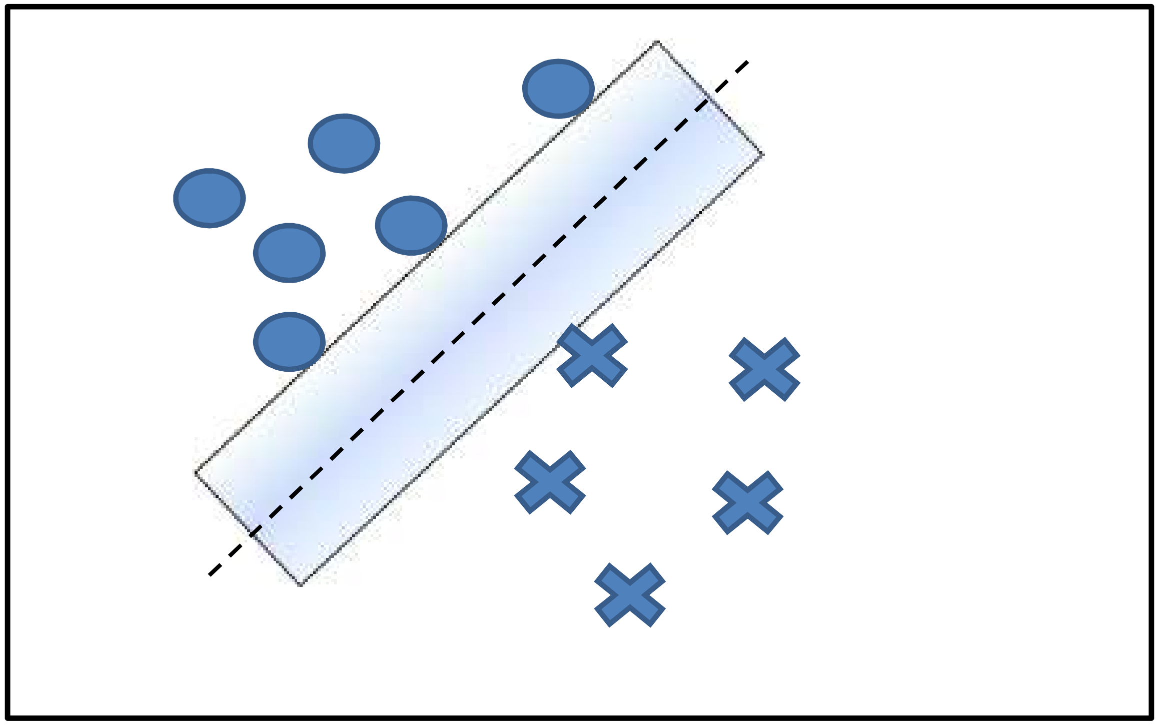
- Compare the following classifications:


(a) (b) (c)

- Is there one classification that is best?
- Clearly, (c) is the most stable (robust) to the addition of further data points.
- How can we capture this insight in an automated way?

Optimal Separability

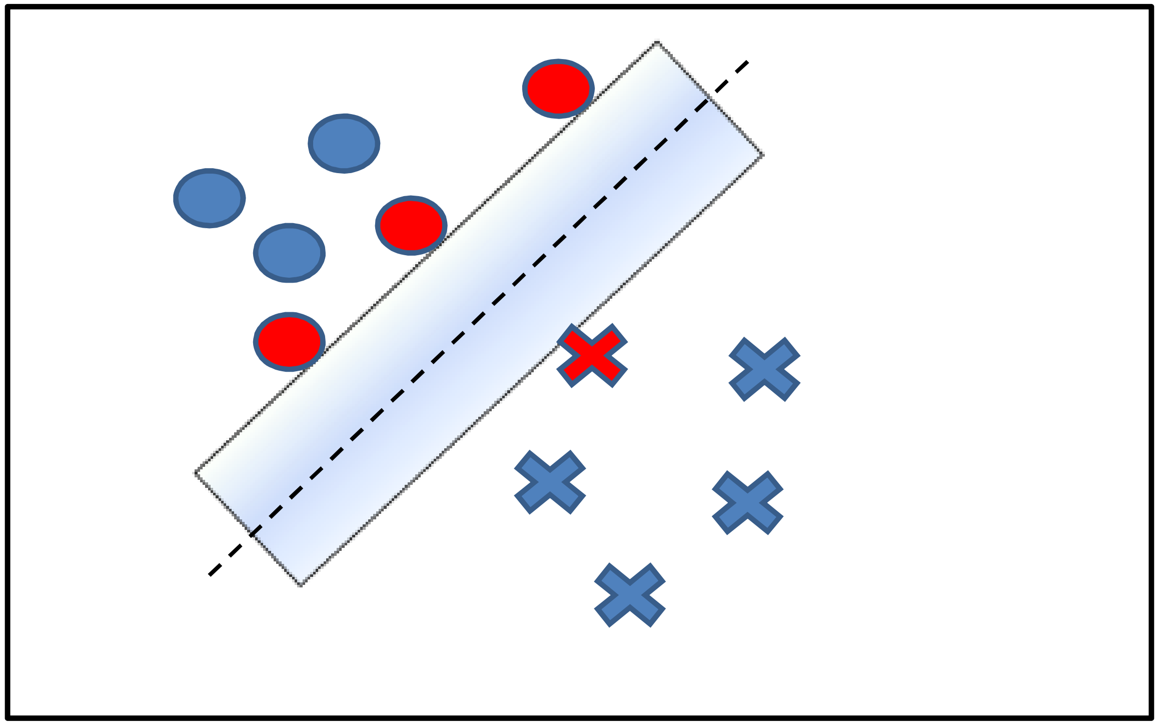
- Consider placing a symmetric *Exclusion or Margin Zone* of size (radius) *R* around the divisor:



- The idea is that *Maximizing R*, the size of this Margin Zone, would lead to the classifier with the *least* confusion or misclassification (especially upon addition of new data).
- This is called the *Maximal-Margin Linear Classifier*.

Optimal Separability

- The input data points that lie closest to the classifier line are called *Support Vectors*.



- Support Vectors are shown here in red.

Optimal Separability

- The Support Vectors are the most useful in classification, because they are the ones that could cause the most confusion or misclassification.
- As in the Perceptron case, the classification line (hyperplane) is defined by: $\boldsymbol{w} \cdot \boldsymbol{x} + b = \boldsymbol{0}$.
- The Exclusion Zone is defined likewise as: $|\boldsymbol{w} \cdot \boldsymbol{x} + b| \leq R$.
- The classification criterion is $\boldsymbol{w} \cdot \boldsymbol{x} + b \leq -R$ for one class, and $\boldsymbol{w} \cdot \boldsymbol{x} + b > R$ for the other.

Optimal Separability

- Now, R is a parameter, and we need an objective criterion to specify R.
- Knowing that the weight vector is perpendicular to the classifier line, we can find that (left as an exercise):

$$R = 1/(2\sqrt{\boldsymbol{w} \cdot \boldsymbol{w}}).$$

- Maximizing the Exclusion Zone, is thus equivalent to minimizing the "*energy*" of the weights (their norm squared). (Note: We saw this earlier in the context of regularization.)
- We want to do this, subject to a constraint that penalizes misclassification (otherwise we would get the trivial result $w = 0$).

Optimal Separability

- We can thus minimize a tradeoff between the weight norm and misclassification:

$$\boldsymbol{w} \cdot \boldsymbol{w} + \lambda \sum_i \varepsilon_i ,$$

  where the ε_i are misclassification errors (*i.e.*, the distance of misclassified points to the classifier boundary line), and λ is a tradeoff parameter.
- This can be solved using Lagrange Multipliers via the Karush-Kuhn-Tucker method.
- This is standard in quadratic programming, but it is outside our scope (left as further reading).

Kernels and Support Vectors (Outline)

- While the Exclusion Zone method will give better (more optimal) separation boundaries than the Perceptron, it won't work unless the problem is linearly separable to begin with.
- We are left with the problem of transforming the input space (possibly by adding dimensions) so as to make the problem Linearly Separable in order to use the Exclusion Zone scheme outlined above, which uses Support Vectors.

Kernels and Support Vectors (Outline)

- Suppose we have a Non-Linearly Separable problem where the data are separable by some non-linear boundary.
- The idea is to introduce a set of transformations to the input data that are general enough to capture such a decision boundary.
- We could start with polynomials of up to degree 2, where we keep linear terms, cross terms, and quadratic terms.
- This has augmented our complexity from N to $N^2/2$.

Kernels and Support Vectors (Outline)

- If we continue to increase the dimensionality of our problem, the complexity will explode.
- Fortunately, it is possible to keep the problem to linear order in the number of data points through the introduction of a *Gram Matrix* (also known as a *Kernel Matrix*).
- The functions thus introduced are known as *Kernels*, and they are essentially normalized basis functions that span the input space, while stretching the input space so as to make the classes Linearly Separable.
- While any positive definite symmetric function is suitable as a kernel, there are a few commonly used kernel functions with nice properties.
- This is the justification for *Support Vector Machines*.
- This is left as further reading…

A Quick Word About Bootstrapping

- Bootstrapping consists of training various models and averaging their responses.
- Bootstrapping is done by *Sampling with Replacement* from the database, in effect "augmenting" the data.
- This allows for *<u>better estimation</u>*, and just as importantly, for a *<u>measure of the dispersion</u>* of the estimation (without doing heavy duty statistical calculations).
- The idea is to train many models on re-sampled data and aggregate their outputs (by taking the average and std dev of the outputs).
- Tradeoff: Computationally Intensive.
- Bootstrapping is also known as "*Ensemble Learning*."
- Despite the complexity, it is a very useful and widely used technique!

A Quick Word About Bootstrapping

- For classifiers we have:
- *Boosting* consists of training new classifiers on the data that was misclassified by previous classifiers.
- *AdaBoost* resamples the data by placing more weight on misclassified points.
- *Bagging* consists of aggregating the outputs of many classifiers and setting the overall output by "*Majority Voting*."
- These are powerful techniques that are easy to implement, although they are also computationally intensive.

That’s it!

- Please fill out course evaluations online
- Projects, Exam Questions…