Inspire…Educate…Transform.

# Artificial Neural Networks

**Lt. Suryaprakash Kompalli (PhD)**

Senior Mentor, International School of Engineering

# Outline

- Limitation of Algorithmic Solutions
- Networks in the Brain
- Artificial Neural Networks (NNs)
  - Perceptron
  - 3 Layer NNs
- Training of NNs
  - Back-propagation
  - Additional Parameters
  - Topology
- Lab Session

# Which images belong to same person?


CSE 7405c

# What type of movie?











CSE 7405c

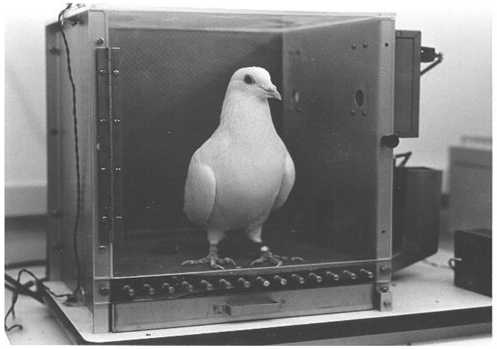
# Other Examples

- Handwriting recognition

- Sentiment Extraction
  - This application is very consistent at breaking down
  - Awesome !!! It takes ages to do simple tasks
  - Much less buggy than the old version, stable on slow hardware as well

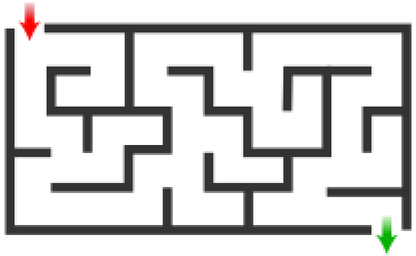
CSE 7405c

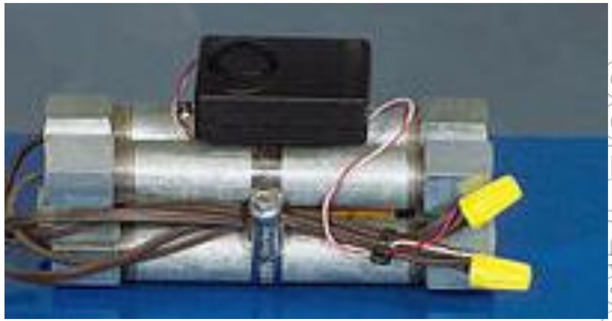
# Thinking is possible even with a "small" brain



Pigeons as art experts – 85% (Watanabe *et al.* 1995)

Source: https://en.wikipedia.org/wiki/Marc_Chagall
https://en.wikipedia.org/wiki/Vincent_van_Gogh





Mice trained to run mazes[1], detect drugs

[1] http://animals.pawnation.com/training-mice-run-mazes-11136.html ,
http://www.ratbehavior.org/RatsAndMazes.htm
[2] http://newsfeed.time.com/2012/11/16/israeli-company-trains-mice-t
detect-drugs-bombs/

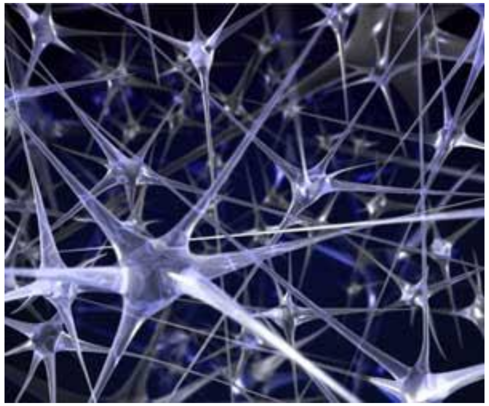
# The results

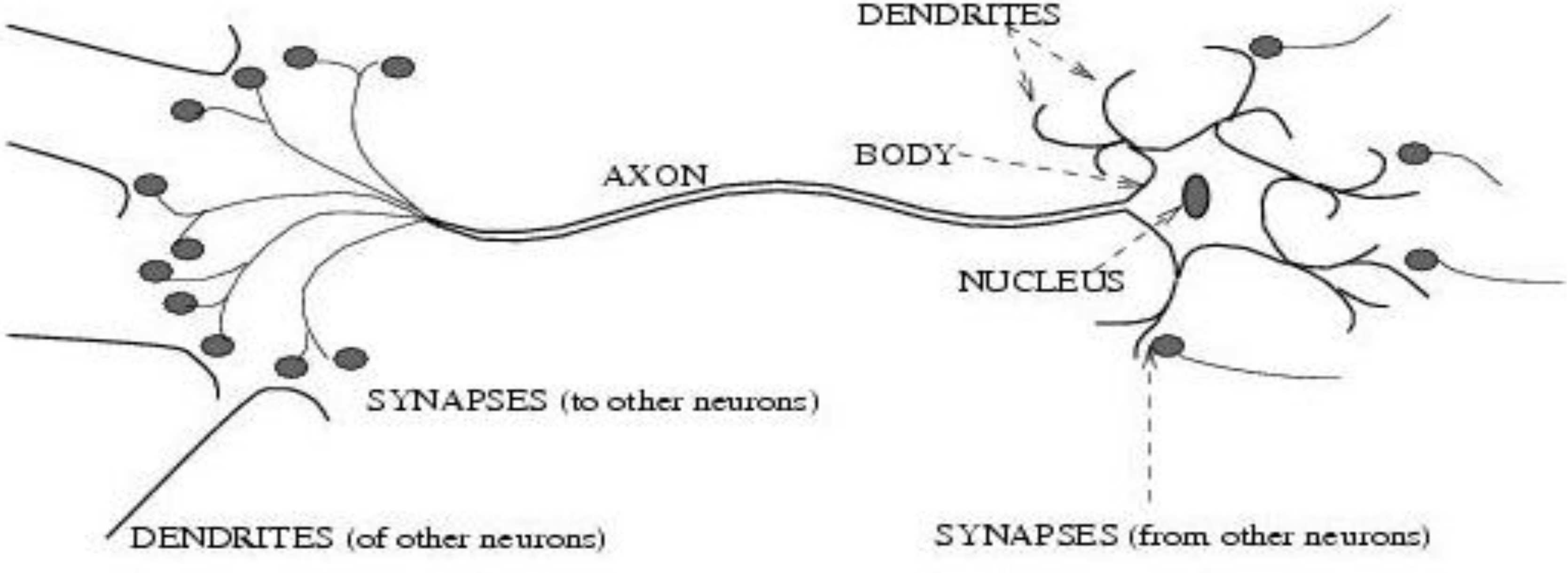
- Pigeons were able to discriminate between Van Gogh and Chagall with 95% accuracy (when presented with pictures they had been trained on)

- Discrimination still 85% successful for previously unseen paintings of the artists

- Mice can memorize mazes, odours of contraband (drugs / chemicals / explosives)

# So, how does the brain work?






Direction of signal is along the Axon from Nucleus to synapse

# Learning of a biological NN

- In 1949 Donald Hebb postulated one way for the network to learn. If a synapse is used more, it gets strengthened – releases more Neurotransmitter. This causes that particular path through the network to get stronger, while others, not used, get weaker.

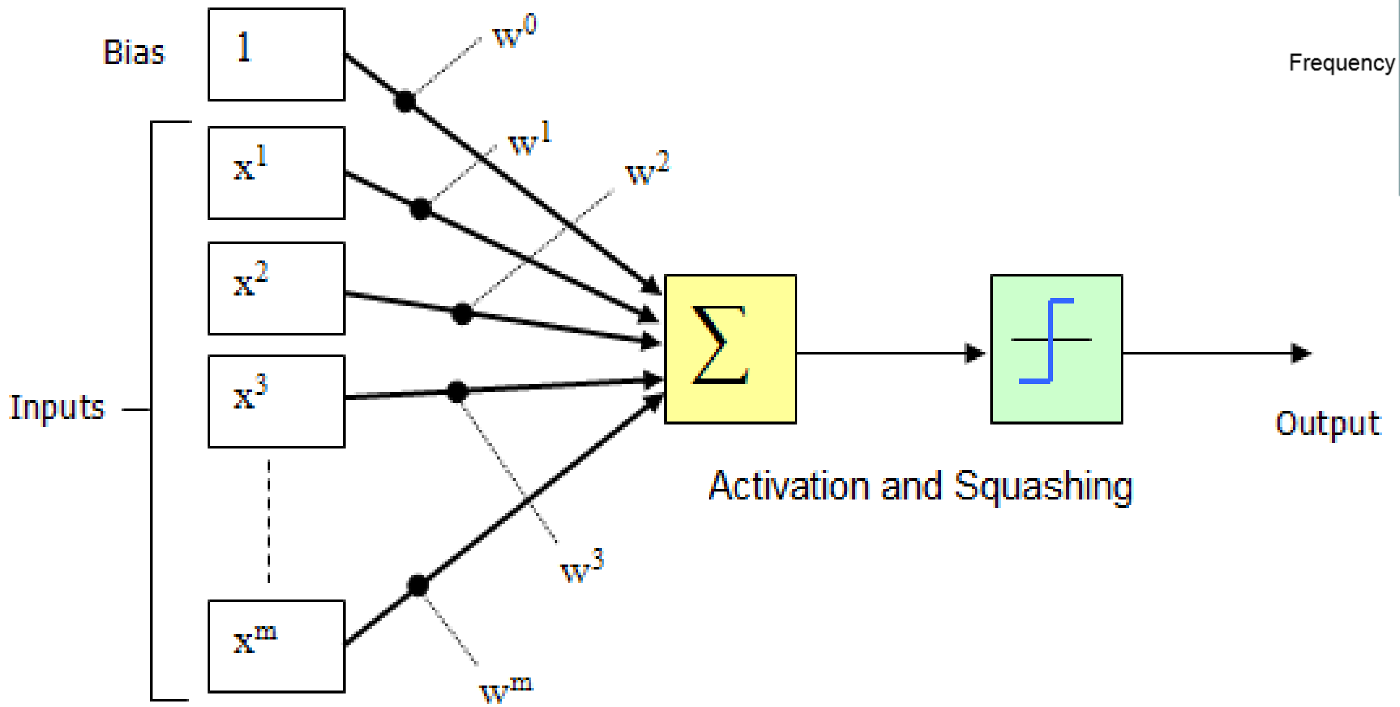
- You might say that each connection has a *weight* associated with it – larger weights produce more stimulation and smaller weights produce less.

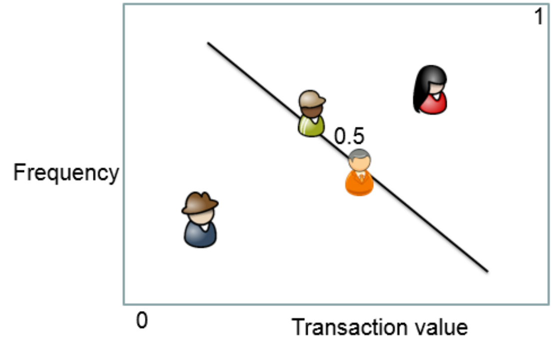
# Outline

- Limitation of Algorithmic Solutions
- Networks in the Brain
- Artificial Neural Networks (NNs)
  - Perceptron
  - 3 Layer NNs
- Training of NNs
  - Back-propagation
  - Additional Parameters
  - Topology
- Lab Session

CSE 7405c

# Perceptron

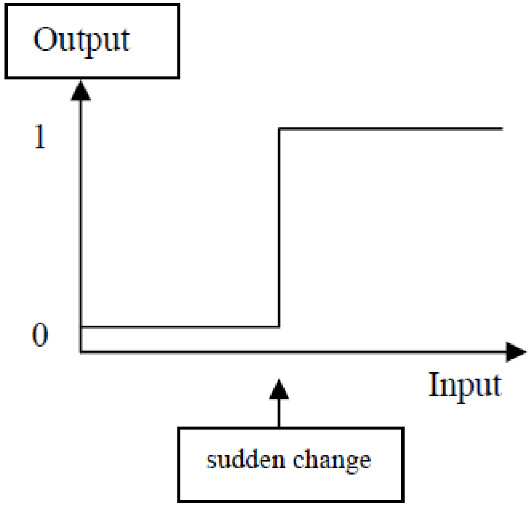


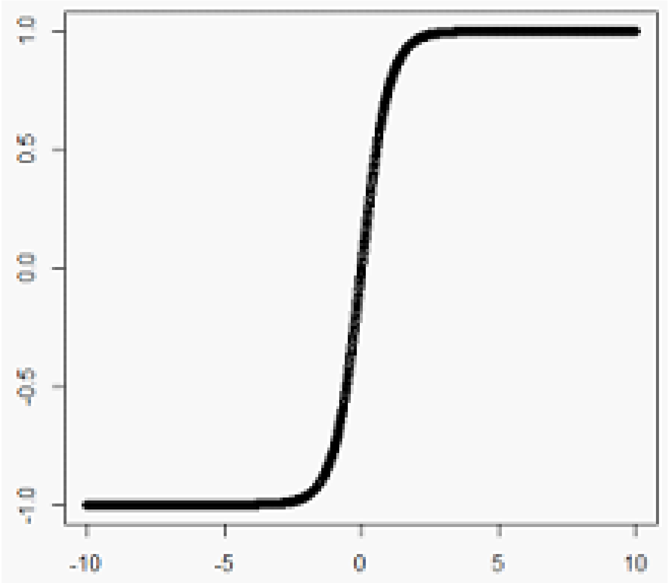




$$Y = mX + c$$

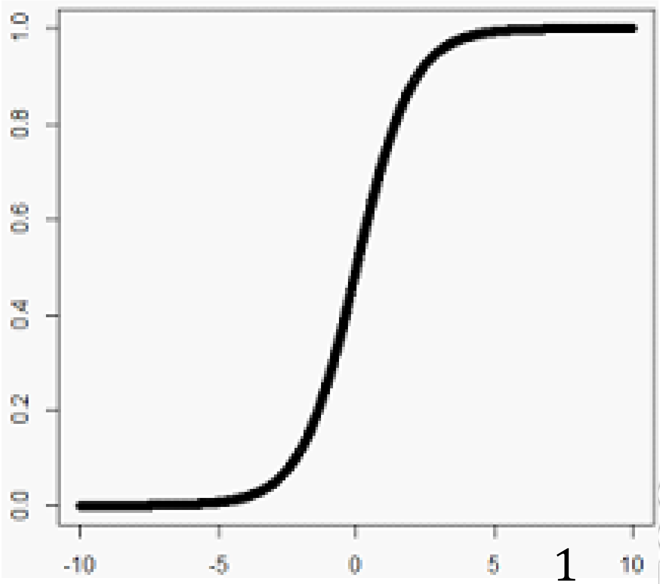
$$X_1W_1 + X_2W_2 + X_3W_3 + \ldots > T$$

# Squashing functions




Tanh(x) $\dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

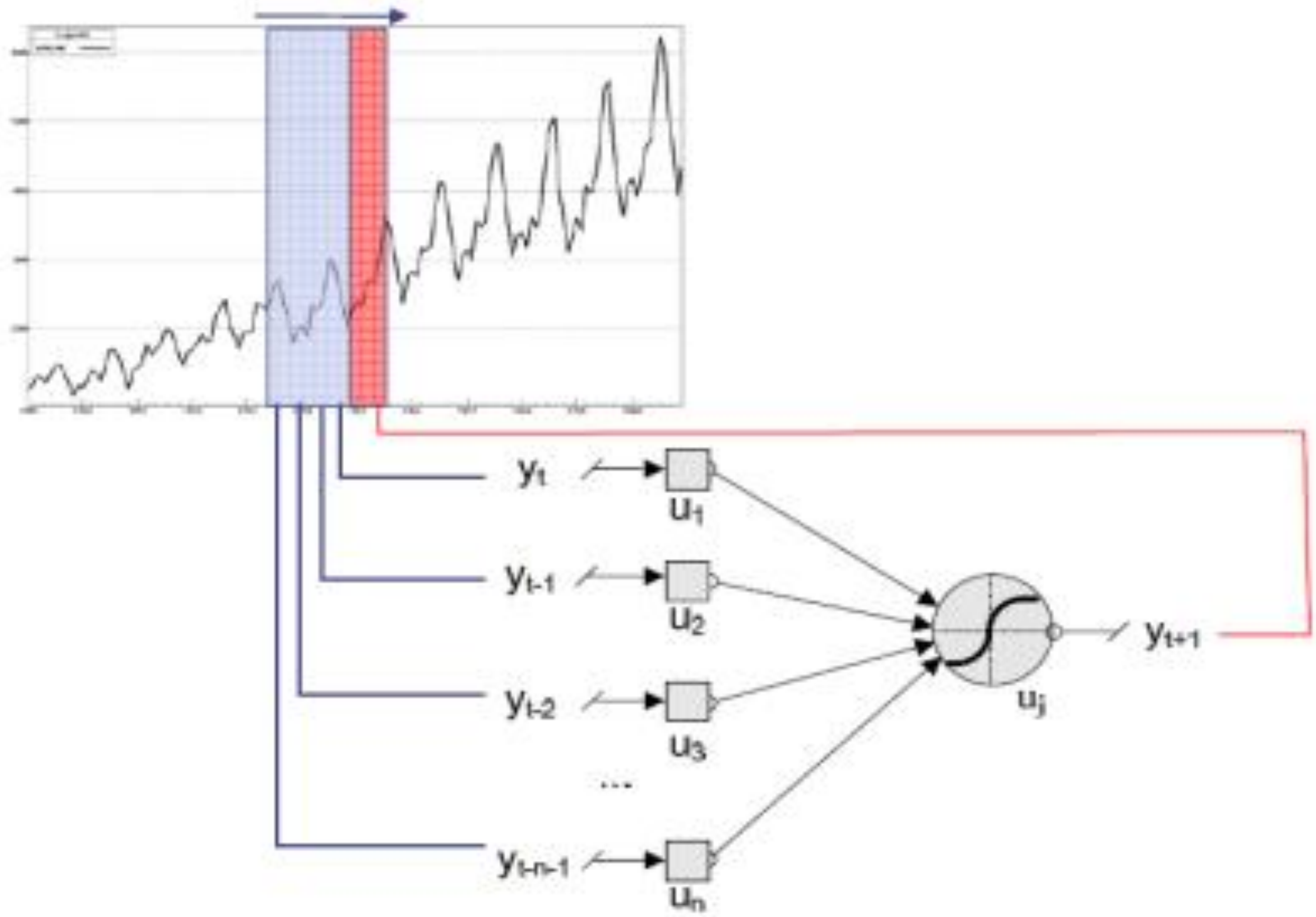



Logistic(x) $\dfrac{1}{1+e^{-x}}$

Good for deviations (time series)

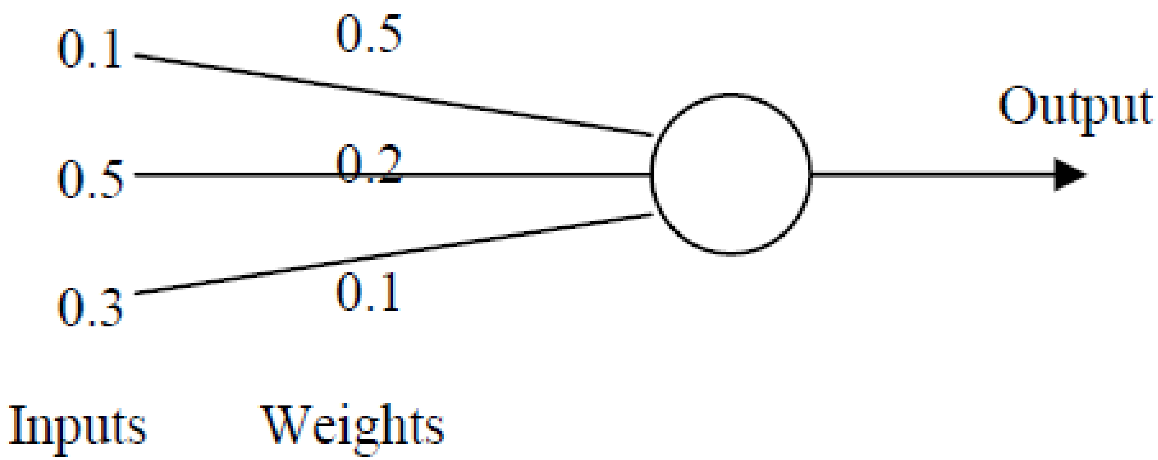
Good for averages (classification)

# Perceptron in a time series



# Q

Calculate the output from the neuron below assuming a threshold of 0.5:




What is the output for a sigmoid function? $\dfrac{1}{1+e^{-x}}$

http://www.dkriesel.com/_media/science/neuronalenetze-en-zeta2-2col-dkrieselcom.pdf

# Question

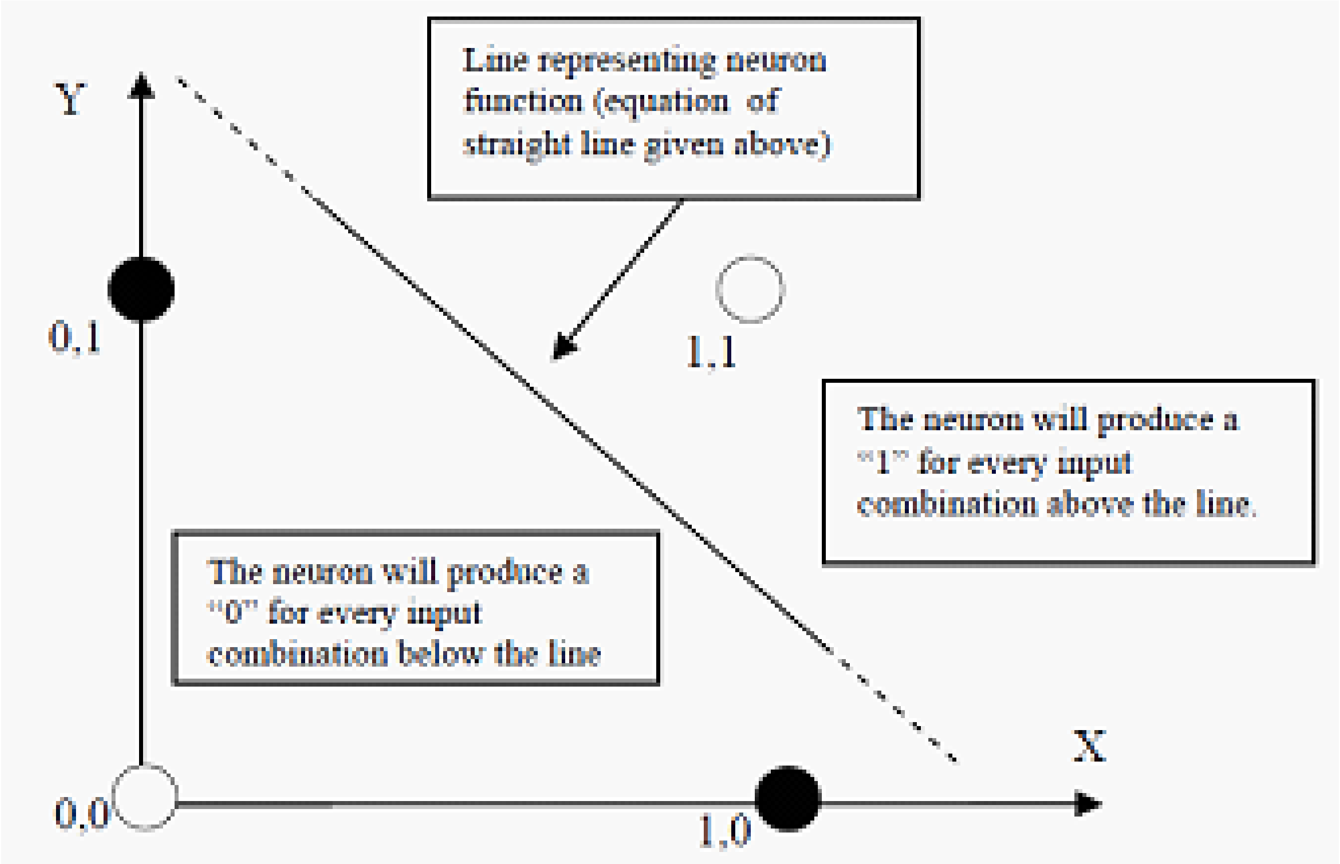
- A time series has a auto correlation with past day, last week same day, last quarter same day and last year same day.

- How many nodes do you have in the input

# LIMITATIONS OF PERCEPTRON AND HOW TO OVERCOME

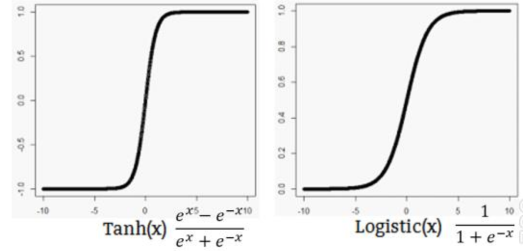
# Will it work here?

| X | Y | OUT |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

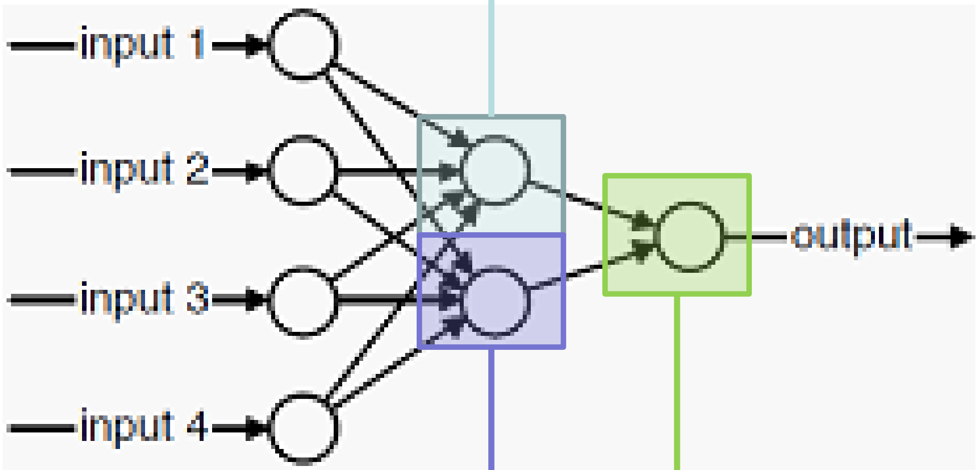


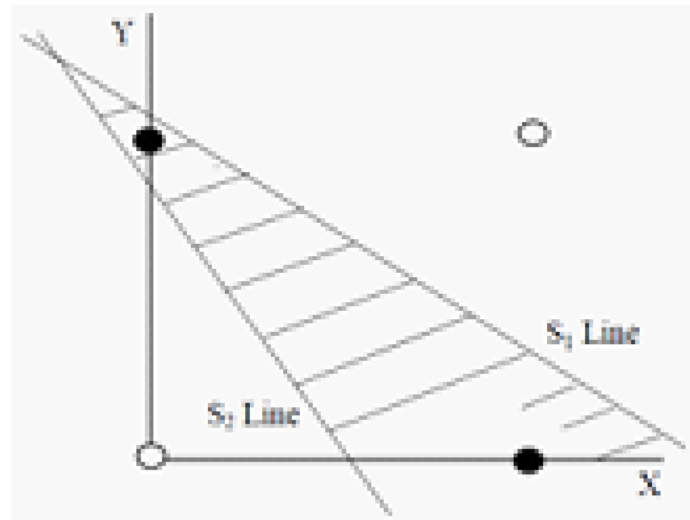

CSE 7405c

# A 3 layer network can learn any function




$$\emptyset_1 = f\left(\sum_{i=1}^{4} x_i w_{i1}\right)$$




$$\emptyset_2 = f\left(\sum_{i=1}^{4} x_i w_{i2}\right)$$

$$Output = f\left(\sum_{j=1}^{2} \emptyset_j w_j\right)$$

$$Output = f\left(\sum_{j=1}^{2} f\left(\sum_{i=1}^{4} x_i w_{ij}\right) w_j\right)$$

CSE 7405c

# A 3 layer network can learn any function

- Kolmogorov's theorem (1957) states:
  *"Any continuous real-valued function f can be represented by continuous functions of one variable"*

$$f(x_1, x_2, x_3, \ldots . x_n) = \sum_{j=1}^{2n+1} g_j\left(\sum_{i=1}^{n} \emptyset_{ij}(x_i)\right)$$

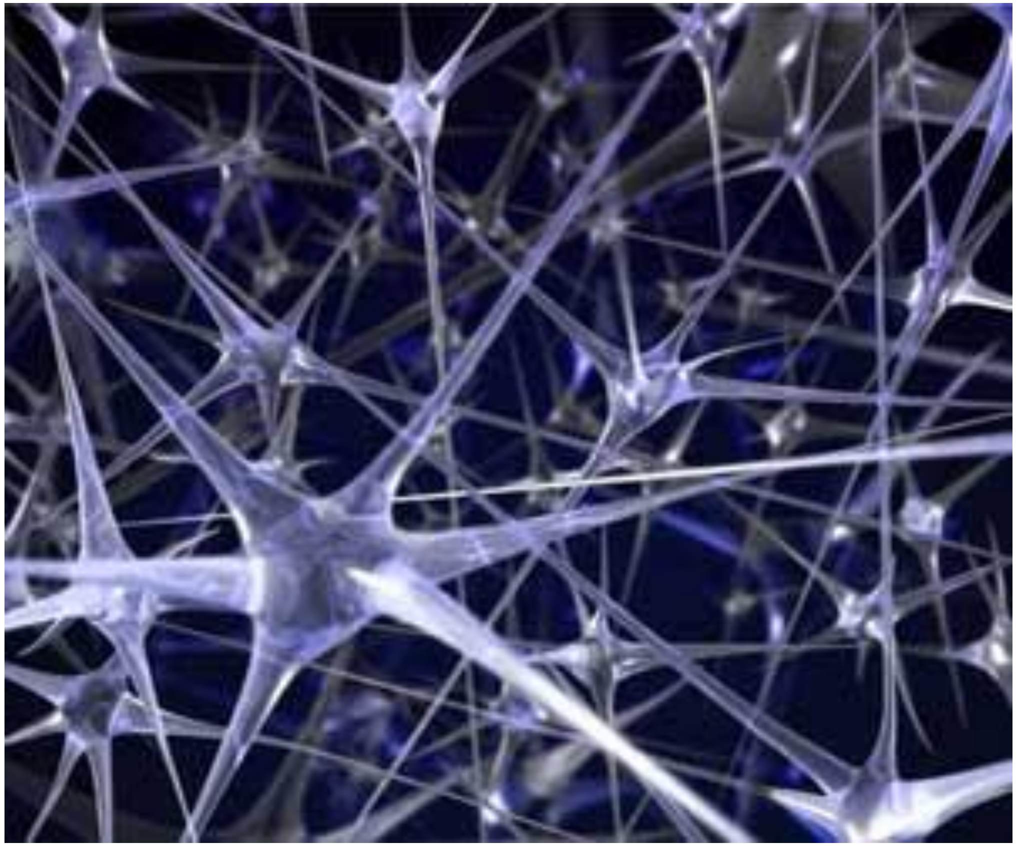
*$g_j$'s are properly chosen continuous functions of one variable, and the $\emptyset_{ij}$'s are continuous monotonically increasing functions independent of f.*

$$Output = \mathrm{f}\left(\sum_{j=1}^{2} \mathrm{f}\left(\sum_{i=1}^{4} x_i w_{ij}\right) w_j\right)$$

$$Output = \mathrm{f}\left(\sum_{j=1}^{2} g_j\left(\sum_{i=1}^{4} \emptyset_{ij}(x_i)\right)\right)$$

CSE 7405c

# How does brain do non linearity

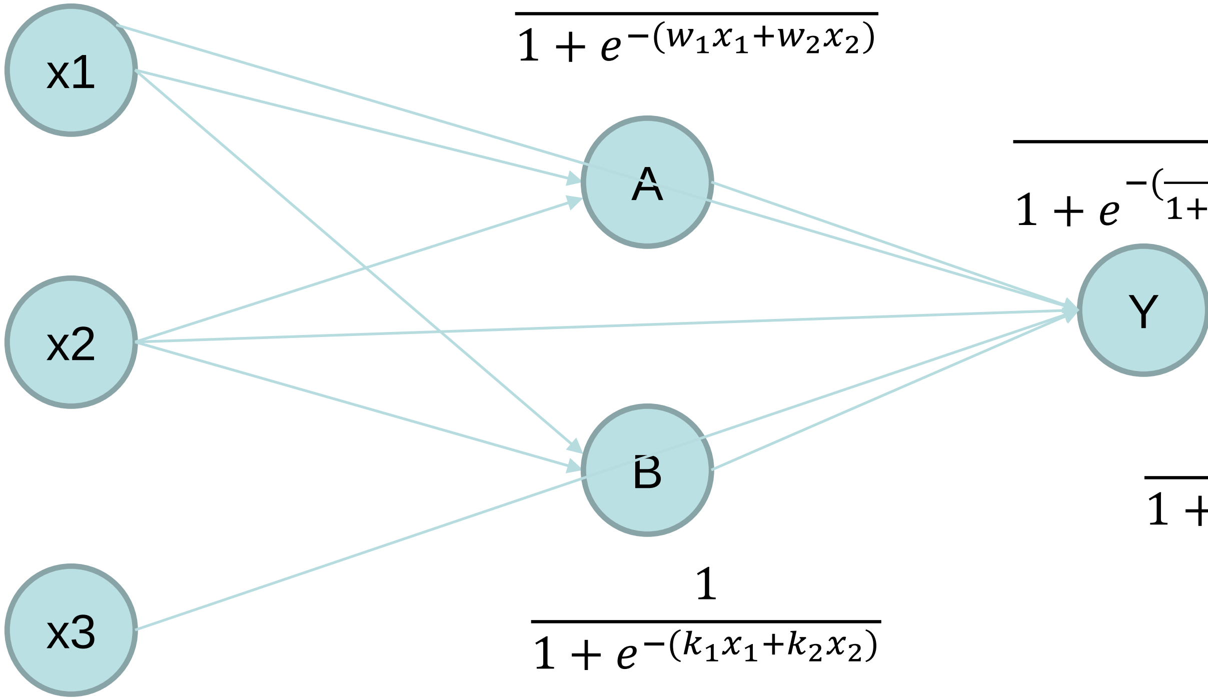


Many neurons connected together

# How to describe non-linearity

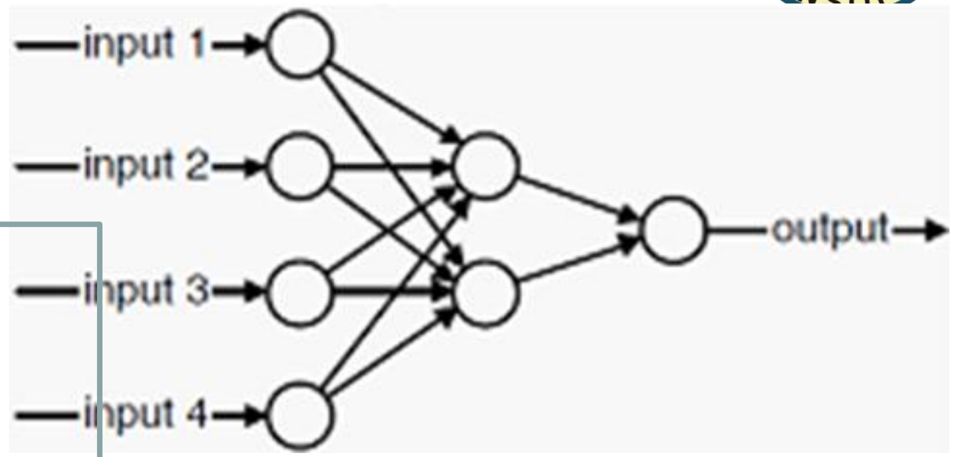
$$\frac{1}{1+e^{-(w_1x_1+w_2x_2)}}$$

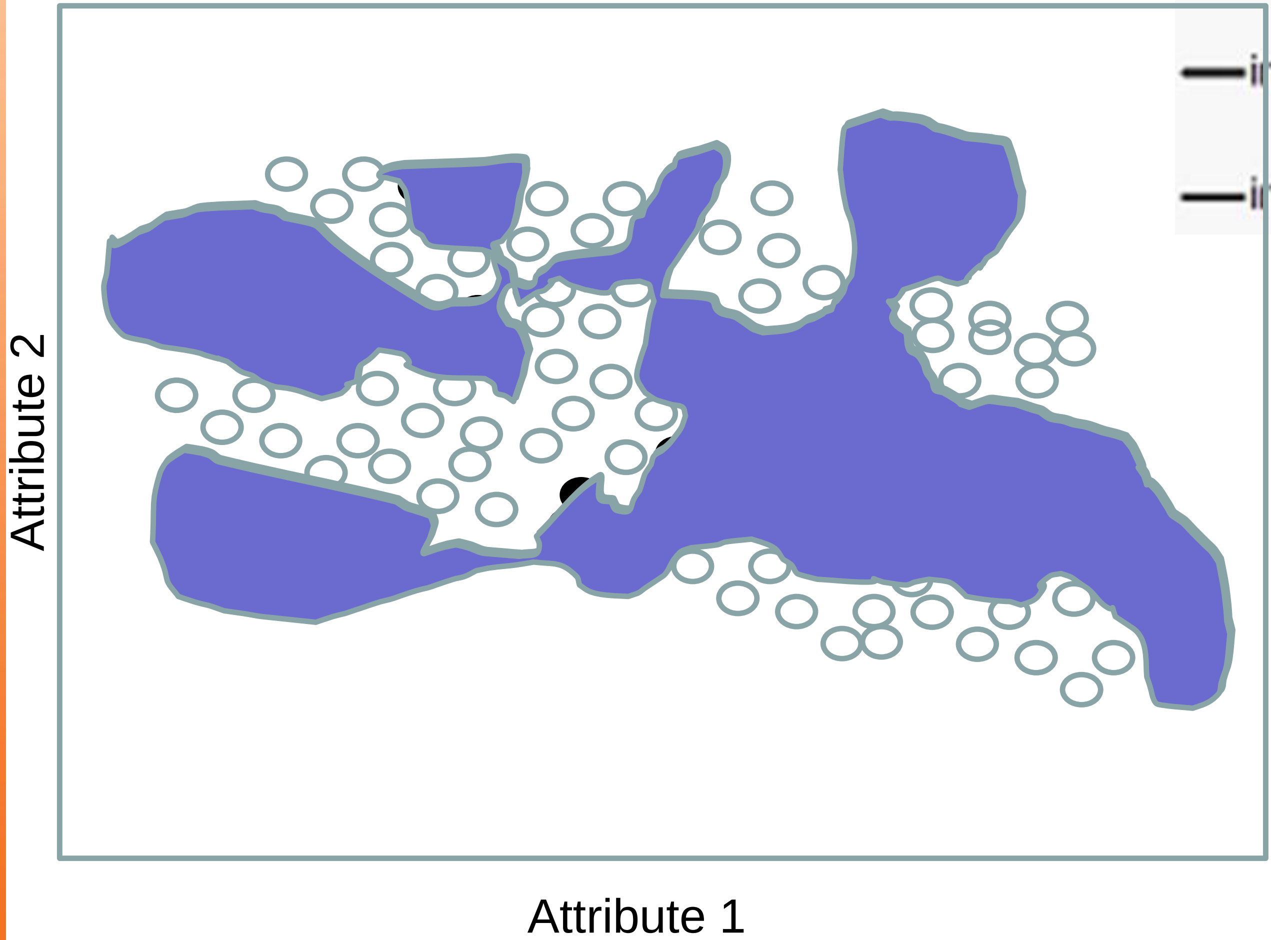
$$\frac{1}{1+e^{-\left(\frac{l_1}{1+e^{-(w_1x_1+w_2x_2)}}+\frac{l_2}{1+e^{-(k_1x_1+k_2x_2)}}\right)}}$$




$$\frac{1}{1+e^{-(w_1x_1+w_2x_2+w_3x_3)}}$$

$$\frac{1}{1+e^{-(k_1x_1+k_2x_2)}}$$


input 1
input 2
input 3
input 4
output


Attribute 2
Attribute 1

CSE 7405c

# Outline

- Limitation of Algorithmic Solutions
- Networks in the Brain
- Artificial Neural Networks (NNs)
  - Perceptron
  - 3 Layer NNs
- Training of NNs
  - Back-propagation
  - Additional Parameters
  - Topology
- Lab Session

CSE 7405c

# Adding non-linearity

There were no training algorithms for multi layered networks until 80s.  As we saw before, single layers are highly limited.  Hence, neural nets were not popular area of research up to 1980s.

Rosenblatt's ideas reëmerged however in the mid-nineteen-eighties, when Geoff Hinton, then a young professor at Carnegie-Mellon University, solved it with back-propagation algorithms

# How can you train a NN

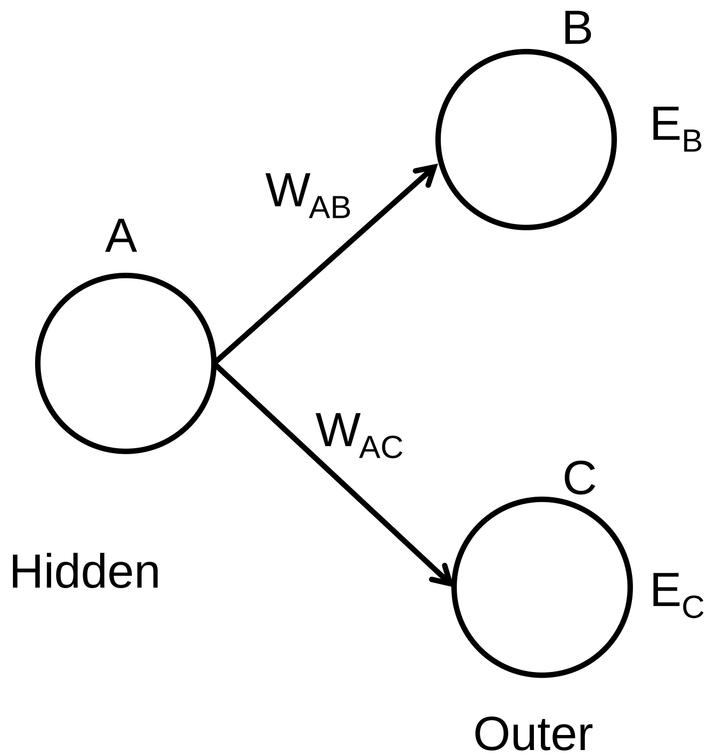
- Learning is changing weights
- In the very simple cases
  - **Start random**
  - **If** the output is correct **then** do nothing.
  - **If** the output is too high, decrease the weights attached to high inputs
  - **If** the output is too low, increase the weights attached to high inputs

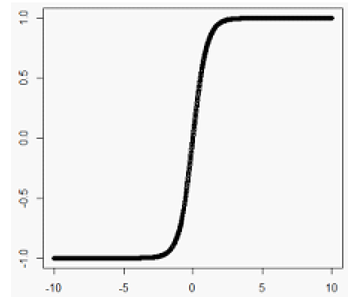
# Methods

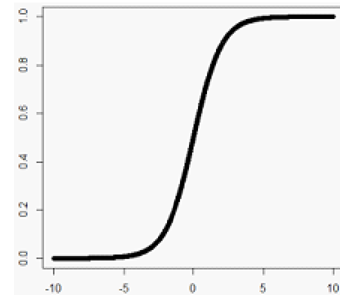
- Gradient descent (Back propagation, conjugate gradient)

- Evolutionary techniques (Genetic algorithms and simulated annealing)

# Back propagation:  Send the error back





Tanh(x)


Logistic(x)

$$\text{Error}_A = \text{Output}_A\,(1 - \text{Output}_A)(\text{Error}_B\, W_{AB} + \text{Error}_C\, W_{AC})$$

The "*Output(1-Output)*" term is necessary because of the Sigmoid (This is derivative of sigmoid).  If we were only using a threshold neuron it would just be *(Target – Output)*.

# NN Training

- Initialize all weights
  - +ve and −ve random values
- Forward phase:

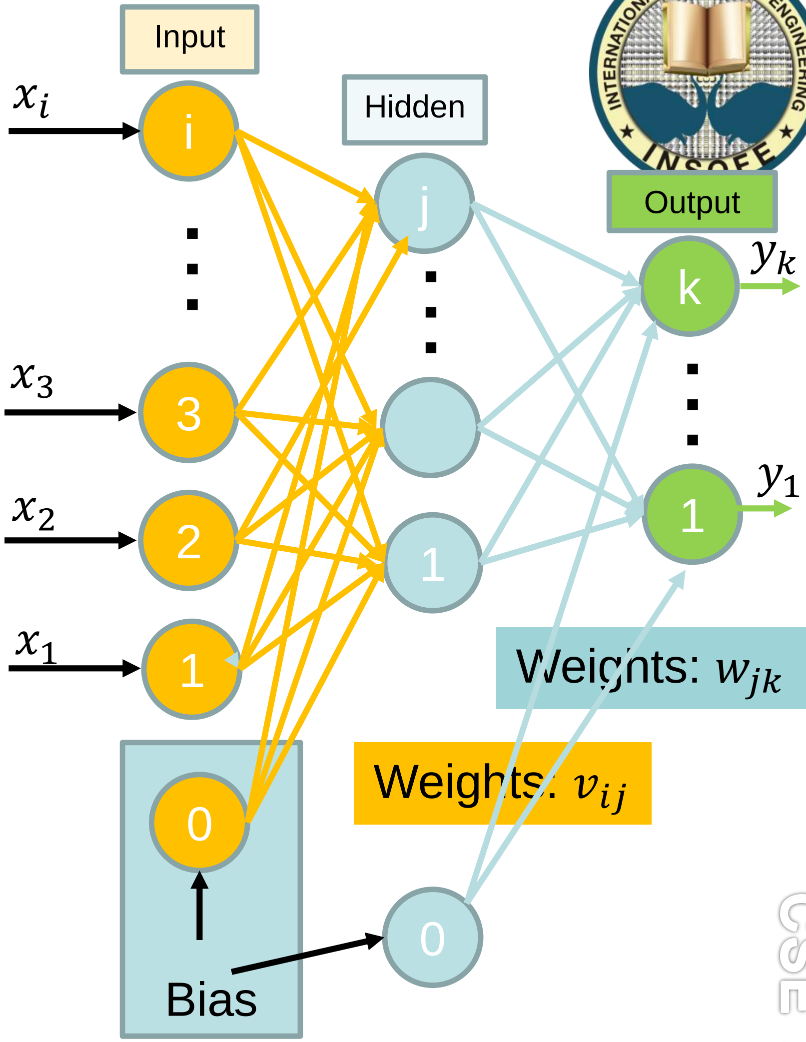
$$h_j = f\left(\sum_{i=0}^{I} x_i v_{ij}\right)$$

$$y_k = f\left(\sum_{j=0}^{J} h_j w_{jk}\right)$$

- Backward phase:

$$\delta\_output_k = (t_k - y_k) \ * \ y_k * (1 - y_k)$$

$$\delta\_hidden_j = h_j \ * (1 - h_j) \ \sum_{k=1}^{K} w_{jk} * \delta\_output_k$$




CSE 7405c

# NN Training

- Forward Phase: $h_j = f\left(\sum_{i=0}^{I} x_i v_{ij}\right)$ $y_k = f\left(\sum_{j=0}^{J} h_j w_{jk}\right)$
- Backward phase:

$$\delta\_output_k = (t_k - y_k) * y_k * (1 - y_k)$$

$$\delta\_hidden_j = h_j * (1 - h_j) \sum_{k=1}^{K} w_{jk} * \delta\_output_k$$

- Update weights:

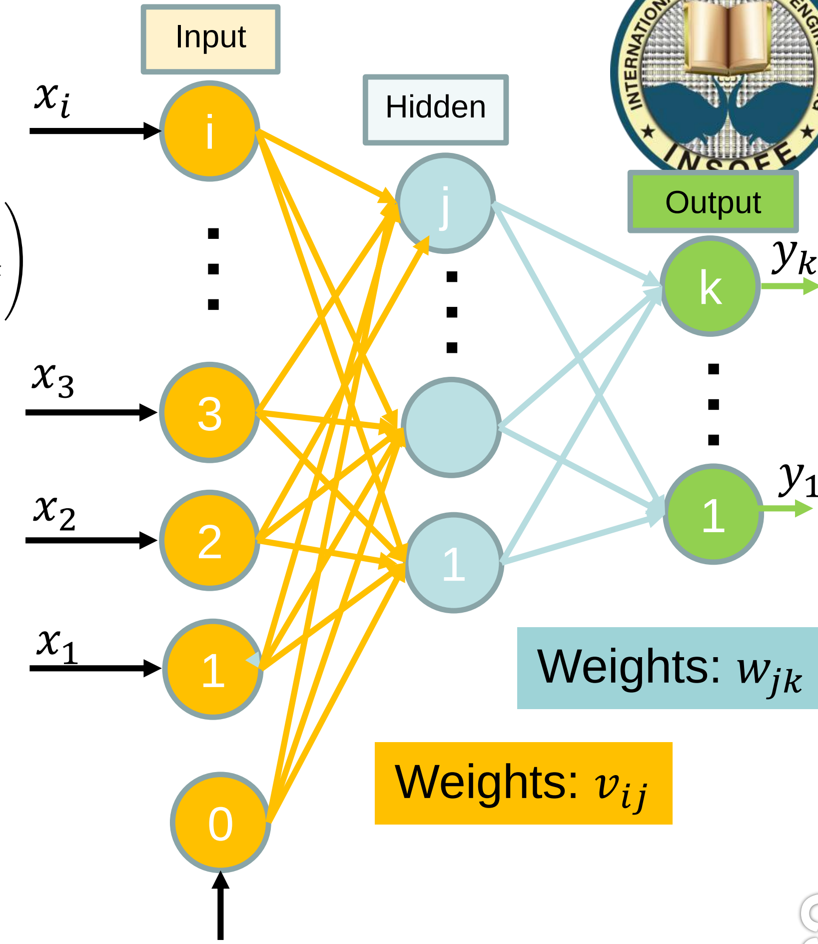
$$w_{jk} \leftarrow w_{jk} + \eta * \delta\_output_k * h_j$$

$$v_{ij} \leftarrow v_{ij} + \eta * \delta\_hidden_i * x_i$$

, where η is the learning rate

$0 < \eta < 1$, typical value would be 0.1 to 0.4

- When do you stop?
- Is training data always in same sequence?



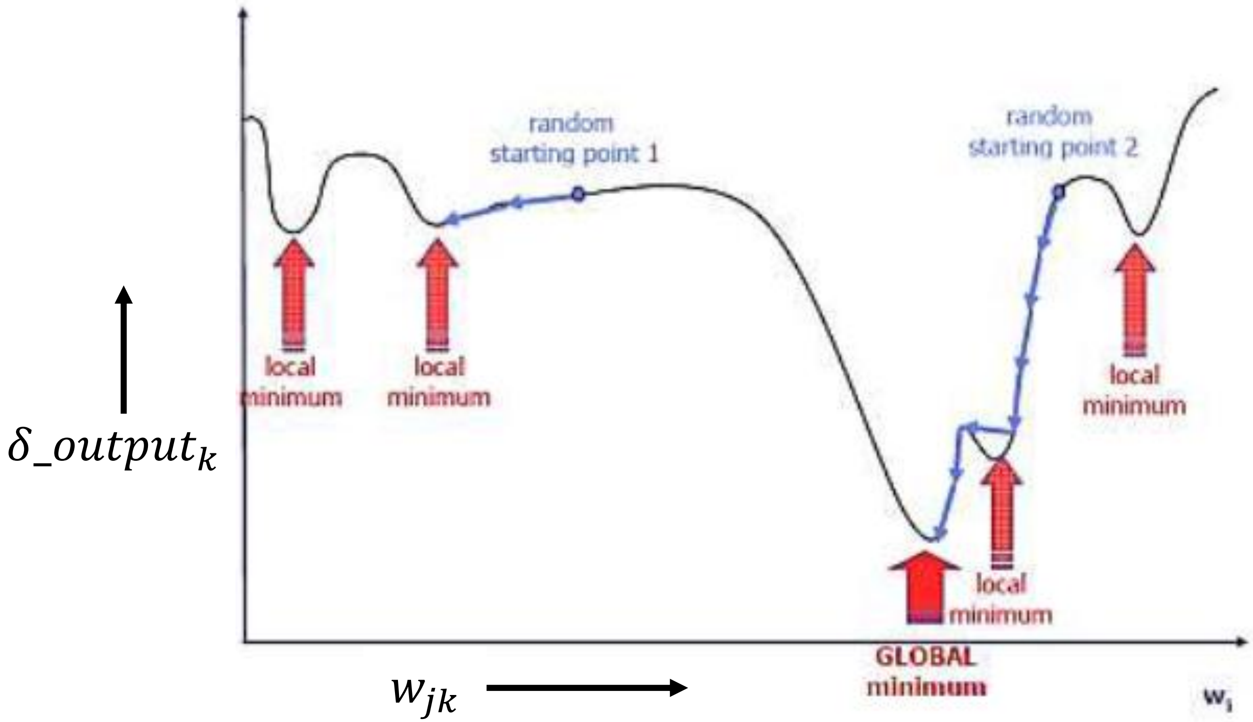
# Outline

- Limitation of Algorithmic Solutions
- Networks in the Brain
- Artificial Neural Networks (NNs)
  - Perceptron
  - 3 Layer NNs
- Training of NNs
  - Back-propagation
  - Additional Considerations during Training
  - Topology
- Lab Session

# NN Training: Additional Considerations

- Which is correct
  - Show an input…adjust weights, show another and adjust weights…,once the input is all over, start with row 1 if needed
  - Show an input train until convergence, show second one, train until convergence…

# NN Training: Other Methods




- Gradient descent (Back propagation, conjugate gradient)
- Evolutionary techniques (Genetic algorithms and simulated annealing)

# NN Training: Additional Considerations

- Momentum: Resistance to directional change

$$w_{jk} \leftarrow w_{jk} + \eta * \delta\_output_k * h_j + \alpha * \Delta w_{jk}^{t-1}$$

$$v_{ij} \leftarrow v_{ij} + \eta * \delta\_hidden_i * x_i + \alpha * \Delta v_{ij}^{t-1}$$

$\alpha$ is the momentum, where $0 < \alpha < 1$

- Learning rate: Resistance to fast learning
  - The learning rate ($\eta$), can be set to a high value

    $0 < \eta < 1$, typical value would be 0.1 to 0.4

  - Gradually reduced with subsequent epochs

CSE 7405c

# NN Training: Additional Considerations

- Jittering: Add small amount of noise to data
- Is training data always in same sequence?
- Weight decay: Weight is multiplied by 0< ε< 1
- Early stopping: Stop when $\delta\_output_k$ has small slope
- ~~Bayesian estimation~~

# NN Training: Additional Considerations

- Variable selection
  - Input layer will need to have as many nodes as there are inputs
  - Dependence, correlation, dimensionality reduction etc.
  - Use decision trees or random forests to first identify the relevant inputs
- Data pre-processing
- Data separation into training, validation and test models

# NN Training: Additional Considerations

- Variable selection example
  - Original time series data is X1, X2, X3,X4..Xn
  - Auto Correlation Function (ACF) showed three lags
  - What will be your input and output?

| Input | | | |
|---|---|---|---|
| Column 1 | Column 2 | Column 3 | Output |
| X1 | X2 | X3 | X4 |
| X2 | X3 | X4 | X5 |
| X4 | X5 | X6 | X7 |
| … | … | … | … |
| Xn-2 | Xn-1 | Xn | Xn+1 |

# NN Training: Additional Considerations

- Sensitivity: Which input is most influential?
- Find the average value for each input. We can think of this average value as the center of the test set.
- Measure the output of the network when all inputs are at their average value.
- Measure the output of the network when each input is modified, one at a time, to be at its minimum and maximum values (usually −1 and 1, respectively).

# Q

- There are two inputs A, B.  The output at average values of A and B is 5
- When A is at minimum, the output is 0 and when B is at minimum, the output is 15.
- Which node influences the network more

# Outline

- Limitation of Algorithmic Solutions
- Networks in the Brain
- Artificial Neural Networks (NNs)
  - Perceptron
  - 3 Layer NNs
- Training of NNs
  - Back-propagation
  - Additional Considerations during Training
  - Topology
- Lab Session

CSE 7405c

# Topology

- 90%: One hidden layer
- 10%: Two hidden layers

- Never more than that! It might lead to overfit

# Topology: Number of nodes

- One hidden node for each class
- 0.5 to 3 times the input neurons
- Geometric pyramid rule: Where input has m nodes and output has n nodes, the hidden layer should have sqrt(mXn)
- Remember Kolmogorov's theorem?

$$f(x_1, x_2, x_3, \dots . x_n) = \sum_{j=1}^{2n+1} g_j\left(\sum_{i=1}^{n} \emptyset_{ij}(x_i)\right)$$

$$Output = \mathrm{f}\left(\sum_{j=1}^{2} \mathrm{f}\left(\sum_{i=1}^{4} x_i w_{ij}\right) w_j\right)$$

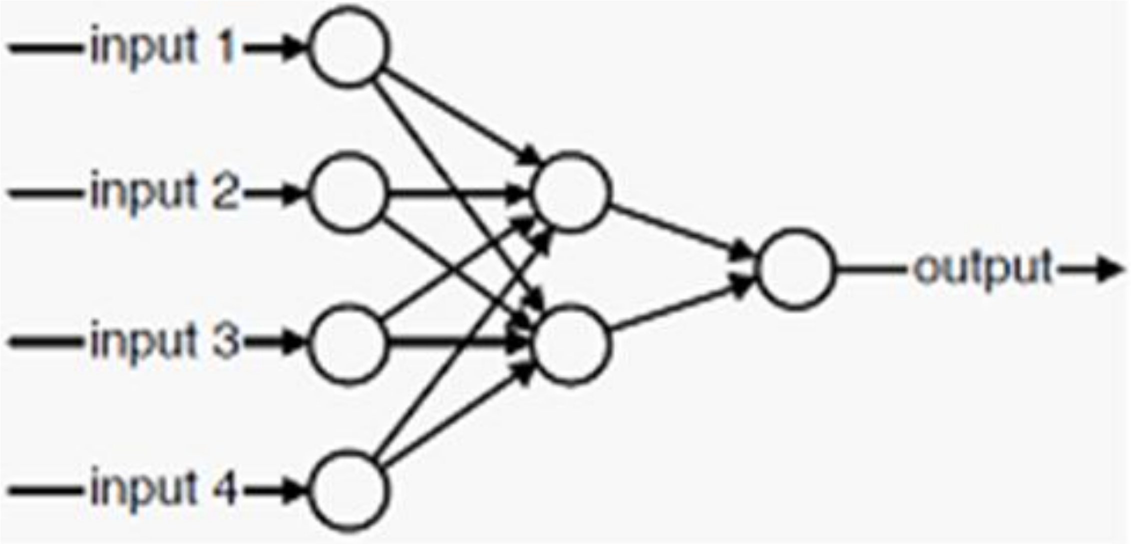
$$Output = \mathrm{f}\left(\sum_{j=1}^{2} g_j\left(\sum_{i=1}^{4} \emptyset_{ij}(x_i)\right)\right)$$

CSE 7405c

# Topology: Geometric Pyramid Rule

- If there are 9 input and 4 output, how many hidden nodes will you start with based on geometric pyramid rule
- Sqrt( 9 * 4) = 6

# Topology: Nodes and Data




$$H * (I + O) + H + O$$

5 input features and 10 units in the hidden network and 1 output, then there are 71 weights in the network.

# Topology: Other Considerations

- The weights should be 1/100th of the amount of training data set

- A NN with 4 input, 2 hidden and 2 outputs require how much of data?
  - H * (I + O) + H + O

  = 2*(4+2) + 2 + 2

  = 16

  Approx 1600 samples

# Topology: Baum-Haussler

- Baum-Haussler rule states that

$$N_{hidden} \leq \frac{N_{train} E_{tolerance}}{N_{input} N_{output}}$$

$N_{hidden}$ is the number of hidden nodes,
$N_{train}$ is the number of training patterns,
$E_{tolerance}$ is the error,
$N_{input}$ and $N_{output}$ are the number of input and output nodes respectively.

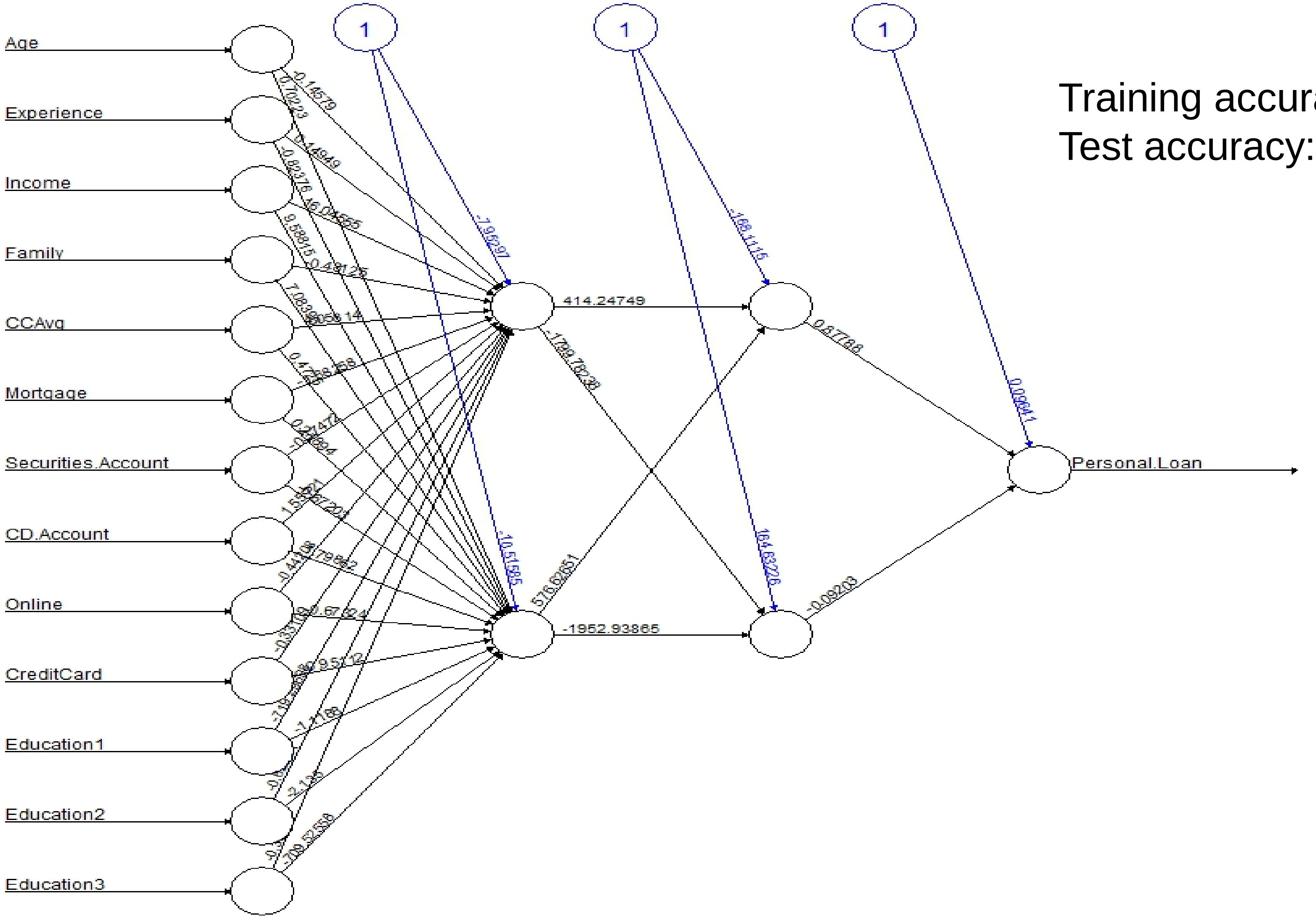
- For 9 inputs, 4 outputs, if I have 10000 test data and allow only 0.01 error, how many nodes will I need

$$N_{hidden} \le \frac{N_{train} E_{tolerance}}{N_{input} N_{output}}$$

$$\frac{10000 * 0.01}{9 * 4} = 2.78 \ge 2$$

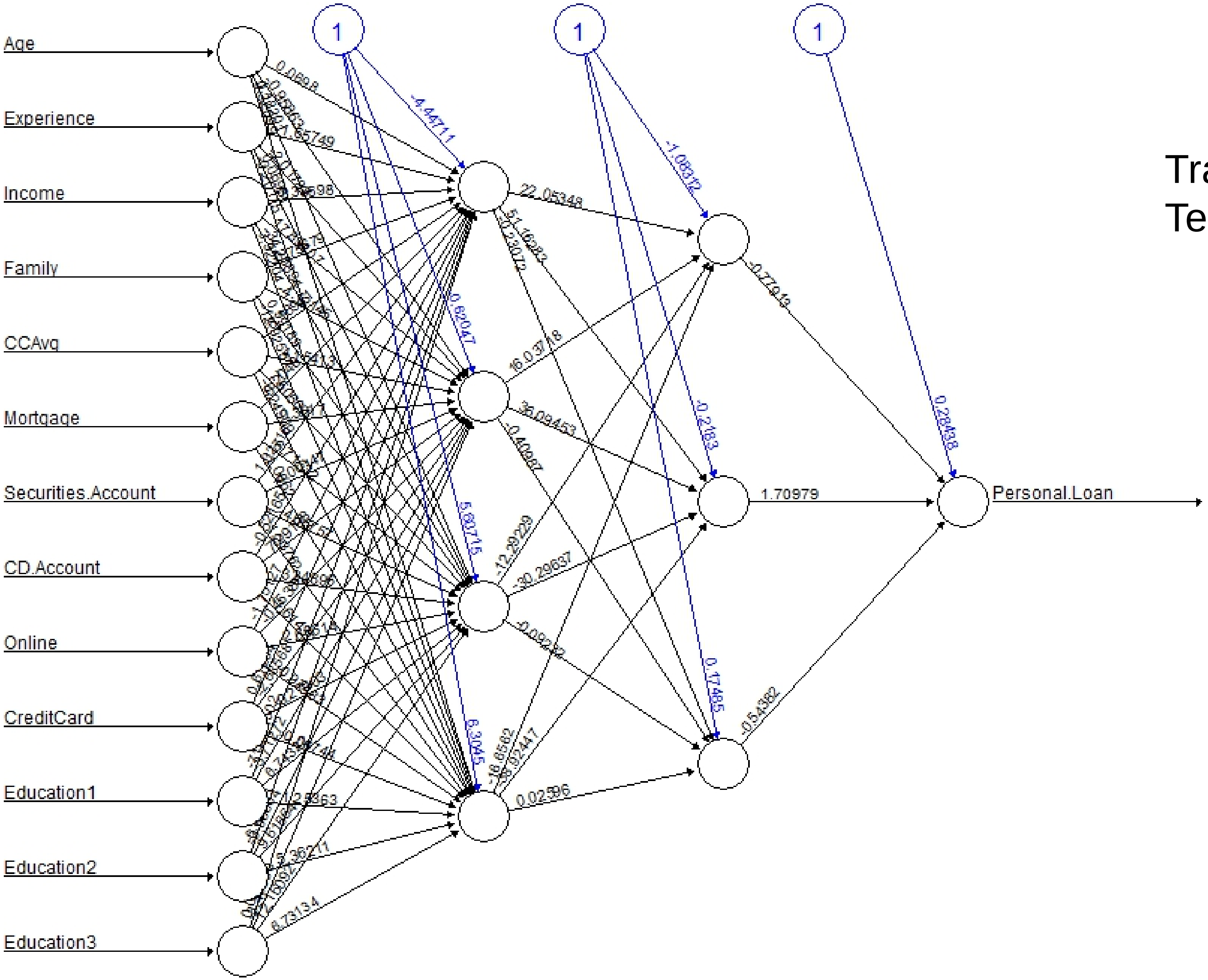
CSE 7405c

# Topology: Example 2x2 ANN




Training accuracy: 97.36%
Test accuracy: 89.38%

# Topology: Example 4x3 ANN

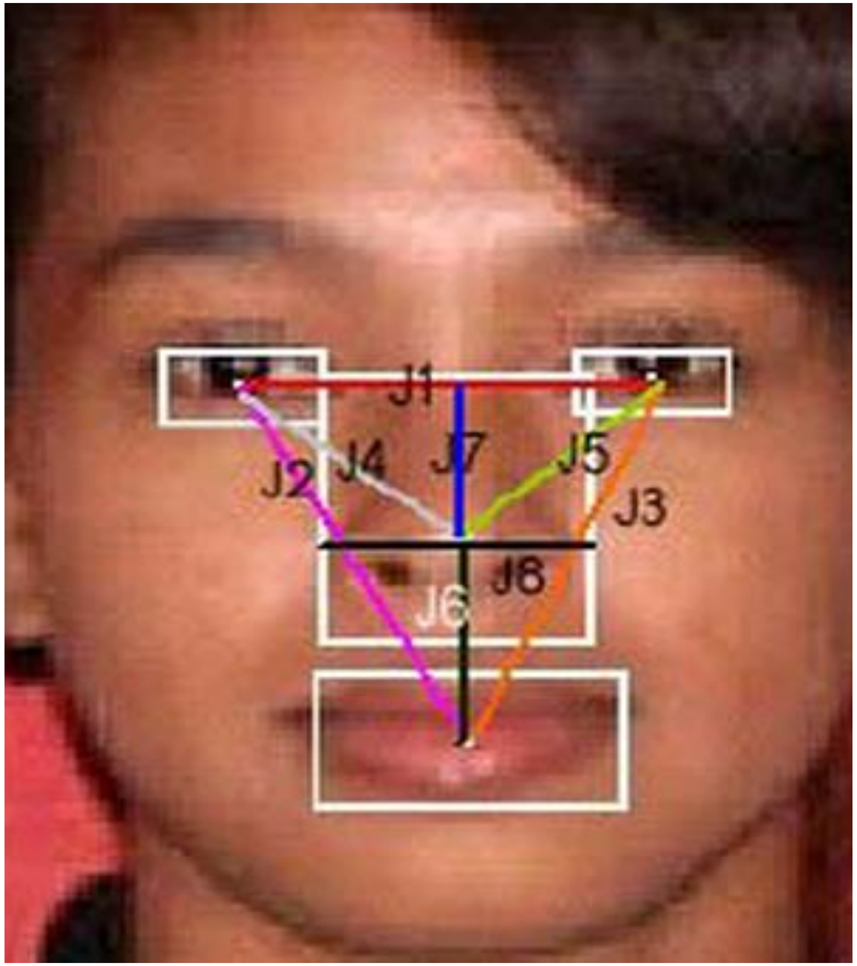



Training accuracy: 100%
Test accuracy: 86.24%

# Outline

- Limitation of Algorithmic Solutions
- Networks in the Brain
- Artificial Neural Networks (NNs)
  - Perceptron
  - 3 Layer NNs
- Training of NNs
  - Back-propagation
  - Additional Considerations during Training
  - Topology
- Lab Session

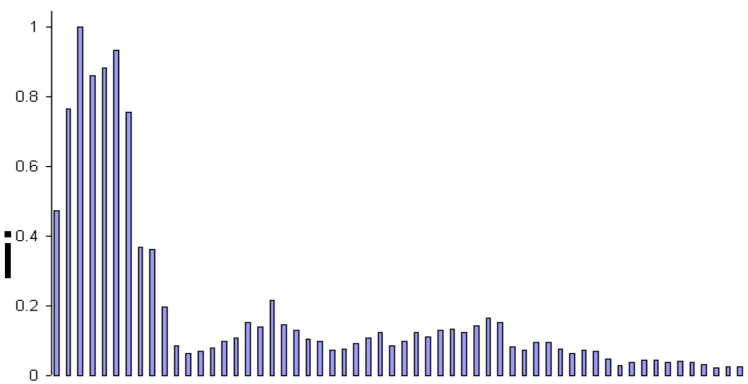
CSE 7405c


J1
J2
J4
J7
J5
J3
J6
J8

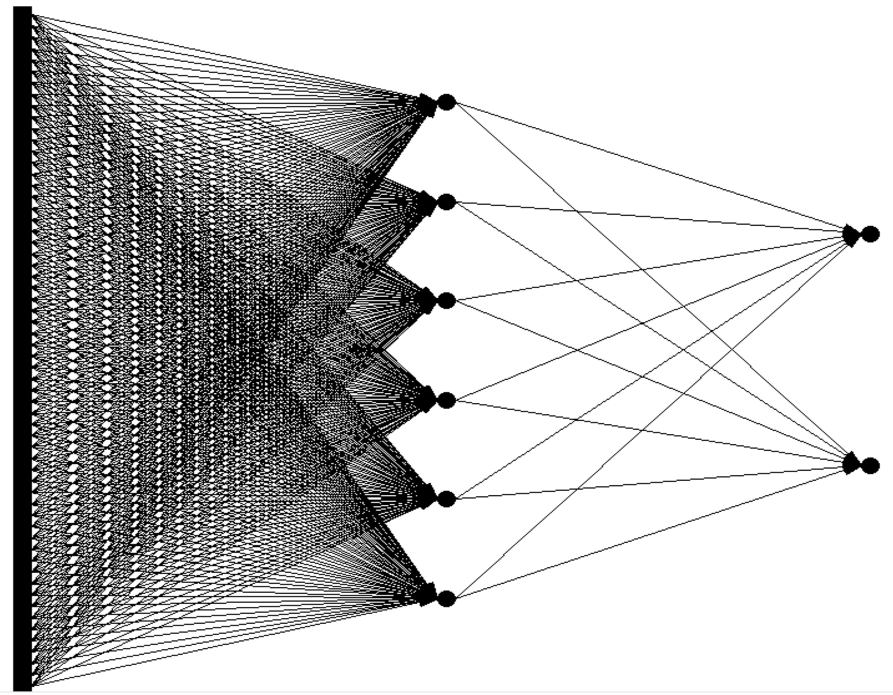
# Example: Voice Recognition

- Task: Learn to discriminate between two different voices saying "Hello"

- Data
  - Sources
    - Steve Simpson
    - David Raubenheimer
  - Format
    - Frequency distribution (60 bi
    - Analogy: cochlea



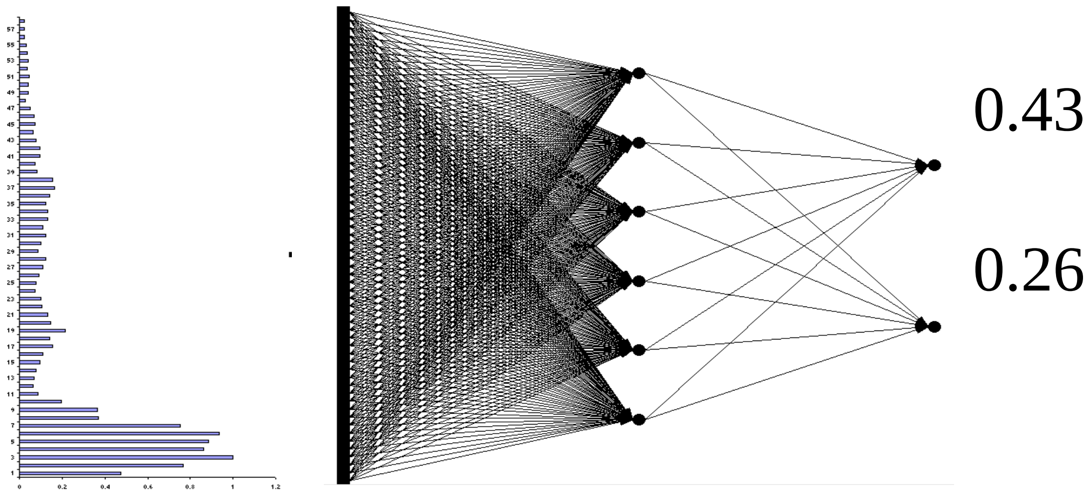
- Network architecture
  - Feed forward network
    - 60 input (one for each frequency bin)
    - 6 hidden
    - 2 output (0-1 for "Steve", 1-0 for "David")



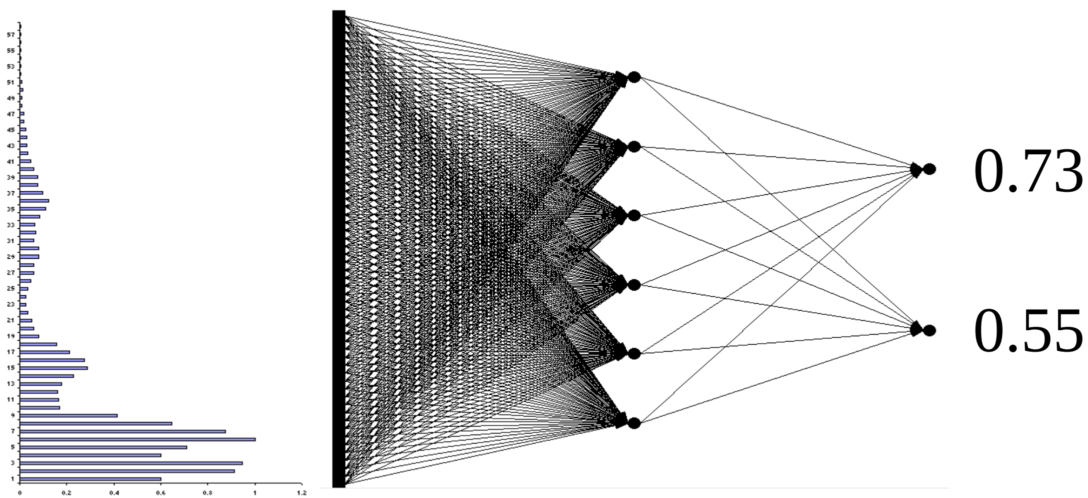
CSE 7405c

# Untrained network

Steve

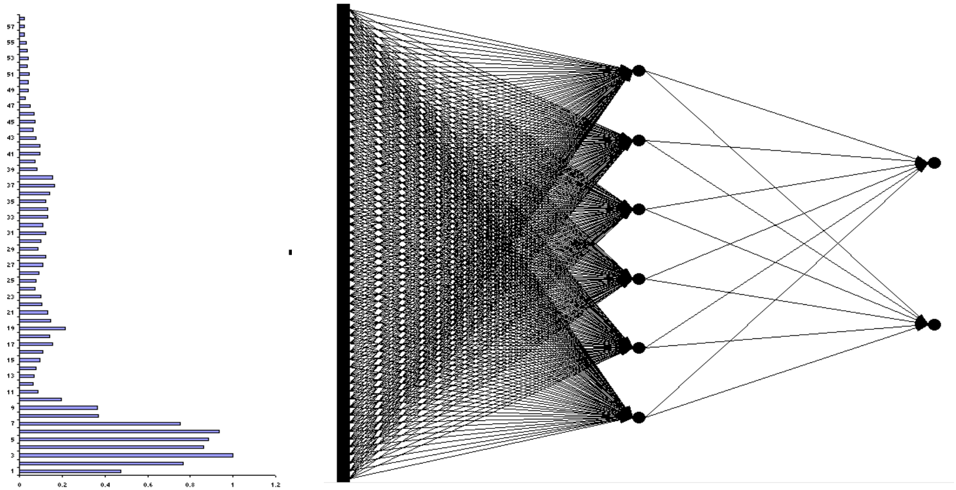



David



# Calculate error

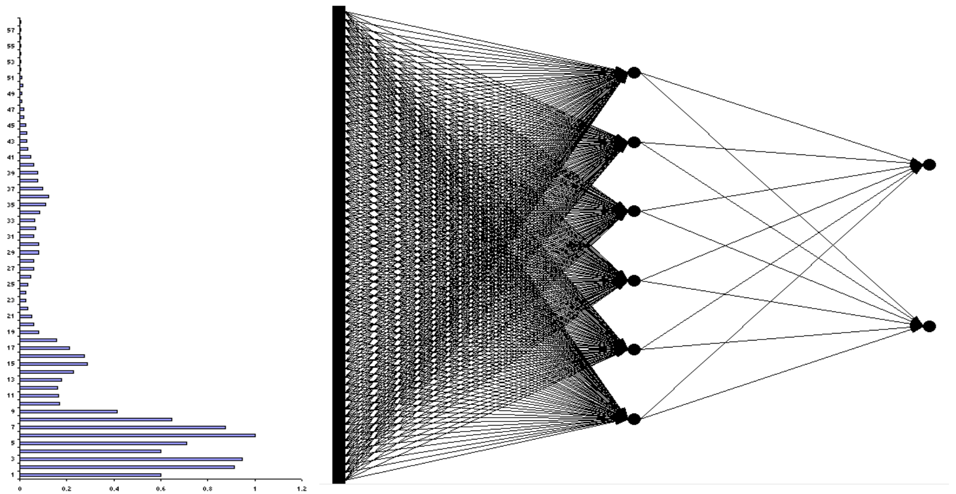
Steve



0.43 – 0 = 0.43

0.26 –1 = 0.74

David

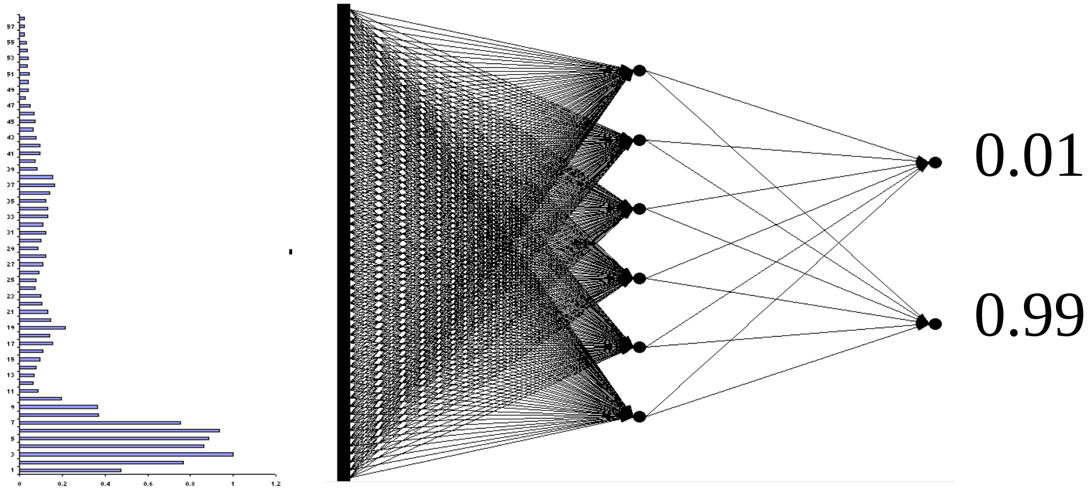


0.73 – 1 = 0.27

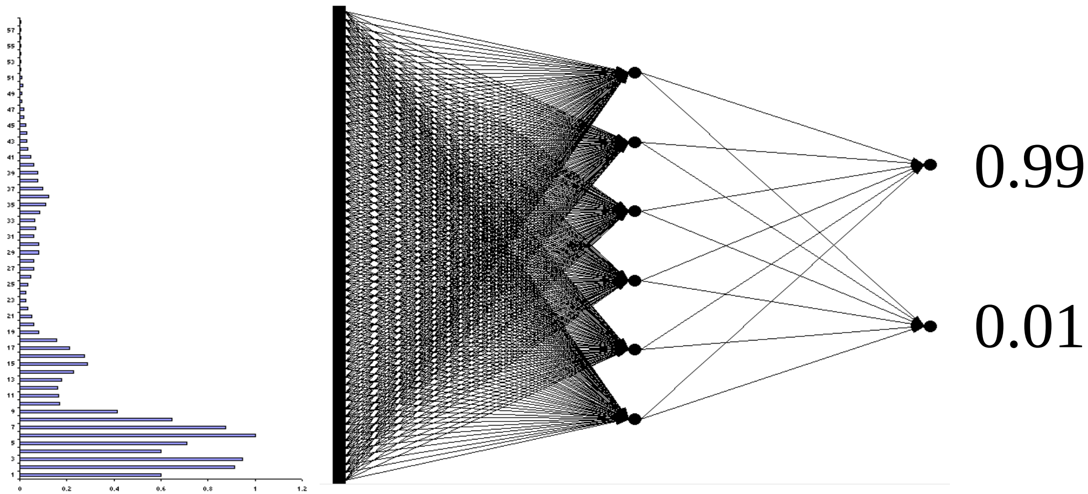
0.55 – 0 = 0.55

# Trained network

Steve




David



- Results – Voice Recognition
  - Performance of trained network
    - Discrimination accuracy between known "Hello"s
      - 100%
    - Discrimination accuracy between new "Hello"'s
      - 100%

# ALVINN

- Neural net driven self driving cars
- 30X4X30 ANN using gradient descent
- http://www.youtube.com/watch?v=0GXuqw3cgwU

# FEED FORWARD IS JUST ONE ARCHITECTURE

CSE 7405c

# Self organizing maps

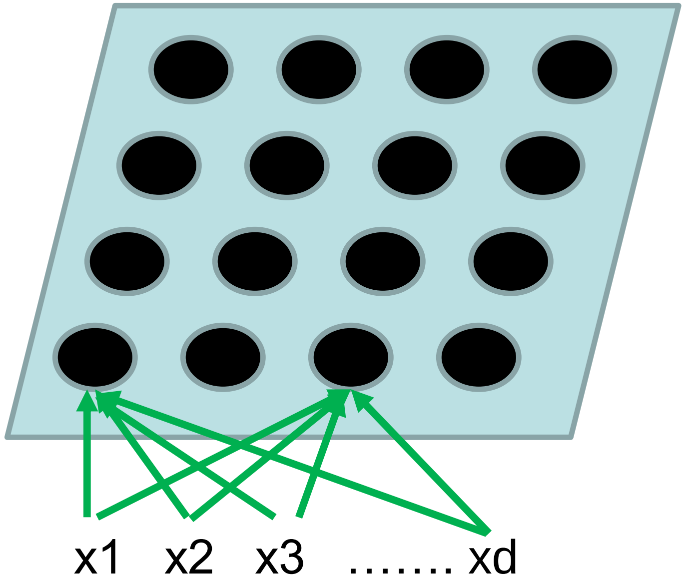
- Invented by Teuvo Kohonen, a professor of the Academy of Finland.

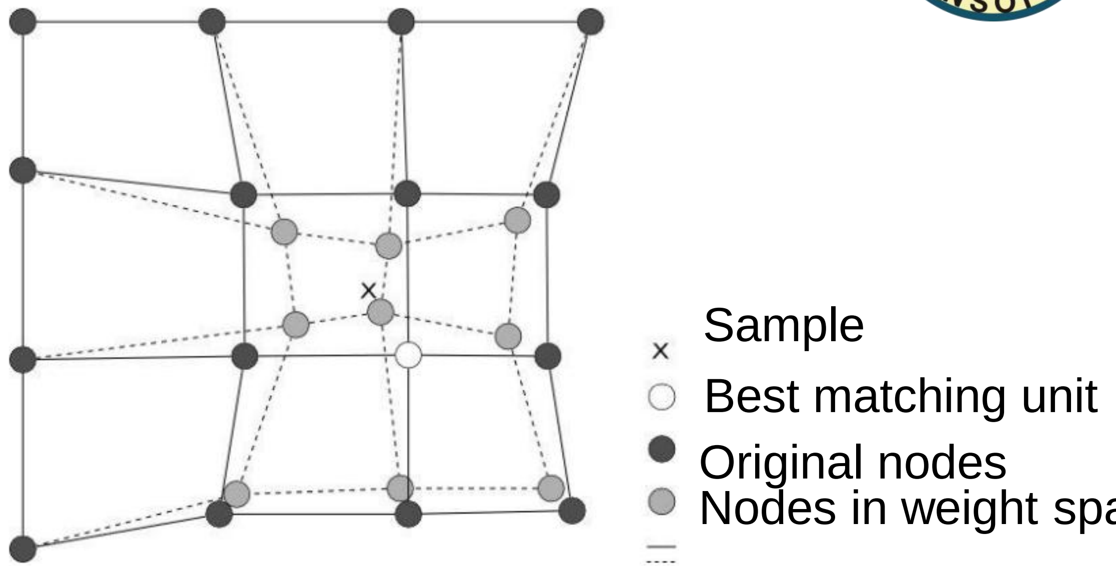
- Kohonen described SOM as a visualization and analysis tool kit for high dimensional data.

- SOMS are however used for clustering, dimensionality reduction and classification.

# The Basics

http://ssdi.di.fct.unl.pt/aadm/aadm1011/slides/LectureSOM.pdf




- Several nodes arranged in a grid at uniform distance.
- Weight from each input (1…d) to node (1…n)
- Training modifies weights:
  - Similar inputs activate neighboring nodes
  - After training, distance between weights indicates how close the nodes are

# BMU

- *Which of the two nodes 1:(0.1, 0.2, 0.3) and 2:(0.9, 0.1, 0.1) are closer to the input vector I=(1, 0, 0).*

- *Distance1 = sqrt( $(1 - 0.1)^2 + (0 - 0.2)^2 + (0 - 0.3)^2$ ) = 0.96*

  *Distance2= 0.17*

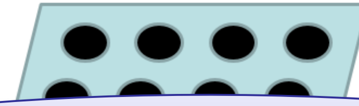
  *Hence, BMU is 2*

CSE 7405c

# Training?

Change learning rate over time: $\eta(t) = \eta_0 \exp(-\frac{t}{\tau_\eta})$

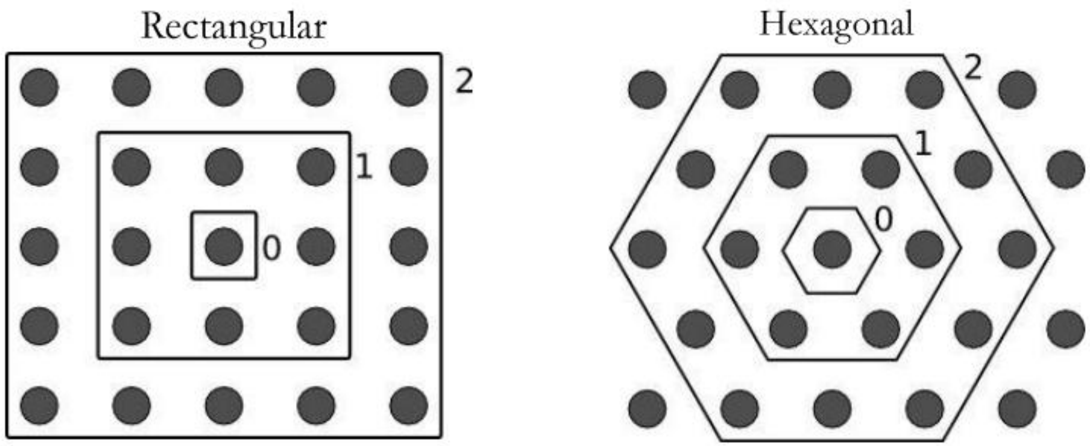
Change neighborhood over time: $\sigma(t) = \sigma_0 \exp(-\frac{t}{\tau_\sigma})$



- Start with random we
- For each input sample (X):
  - Find $d_j(X) = \sum_{i=1}^{D}(x_i - w_{ij})^2$ $j\ is\ from\ 1\ to\ N,$
  - B=Best Match Unit/Node which gives least $_j(X)$
  - Distance from B to other nodes: $T_{j,B} = \exp(-S_{j,B}^2/2\sigma^2)$
    - $S_{j,k} is\ dist\ from\ node\ j\ to\ k\ in\ the\ grid\ space$
  - Update each node weight: $\Delta w_{ij} = \eta * T_{j,B} * (x_i - w_{ij})$
- Shuffle the training set, repeat till weights stop changing

# SOM Parameters

**http://ssdi.di.fct.unl.pt/aadm/aadm1011/slides/LectureSOM.pdf**



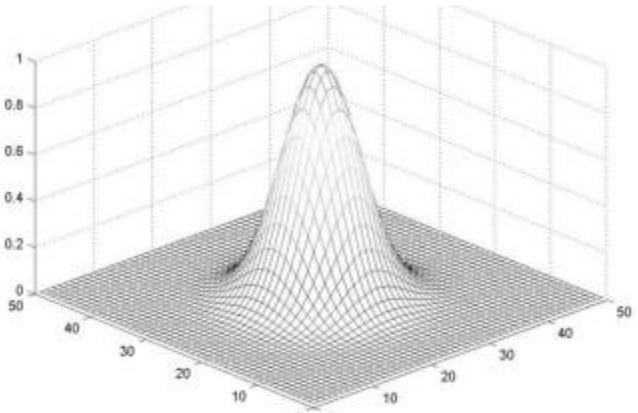

Classical vs Batch

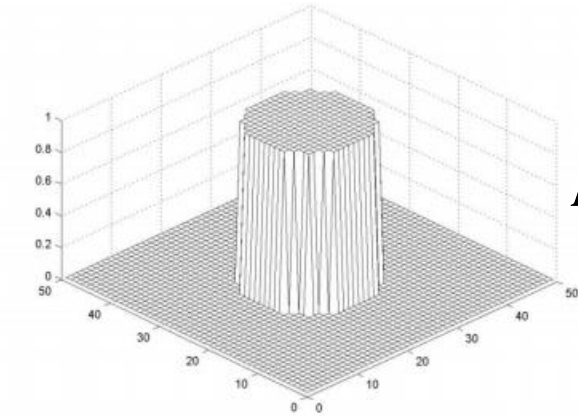
Weights updated at each sample

Weights updated at each epoch

Node distance: $S_{j,k}$ can be Manhattan or Euclidian

Rectangular/Hexagonal





*Radius set to entire SOM, reduced over time*

$$\sigma(t) = \sigma_0 \exp\left(-\frac{t}{\tau_\sigma}\right)$$

$$\tau_\sigma = \frac{N}{\log(\sigma_0)}$$

Reducing influence: $T_{j,B} = \exp(-S_{j,B}^2/2\sigma^2)$

$\sigma^2$ may be reduced with epochs

Constant influence:

$T_{j,B} = 1\ if\ \left\|S_{j,k}\right\|^2 < \ \delta(t)\ else\ 0$

# How to use SOM?

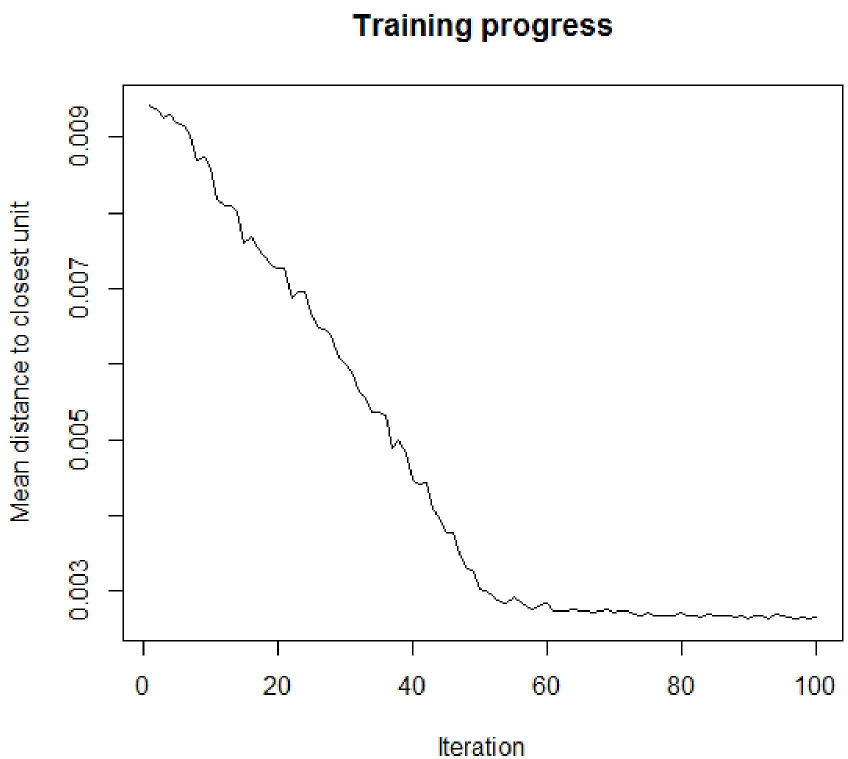
| | | |
|---|---|---|
| Pre-process data:<br>Standardization<br>Remove invalids<br>Remove outliers | Build Self Organizing Maps | Generate heatmaps and clusters to interpret data |

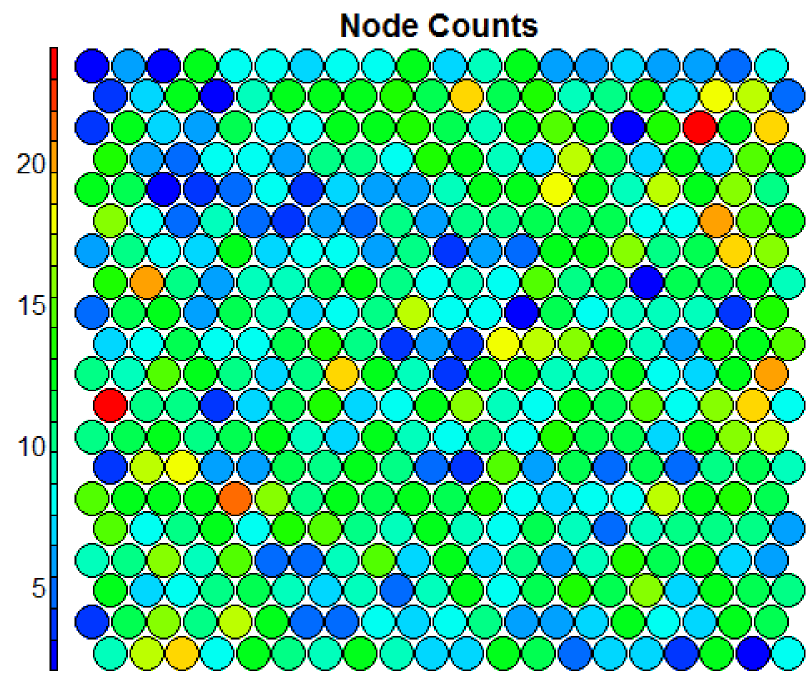
- In a random class, if students with similar features sit as close as possible to each other
  - Age, grades, hobbies etc
- If you examine their grades with respect to new positions, this will be a SOM

# SOMs in Practice

- Example from a tutorial by Shane Lynn
  - Other examples:
    - http://www.r-bloggers.com/self-organising-maps-for-customer-segmentation-using-r/
    - http://manuals.bioinformatics.ucr.edu/home/R_BioCondManual#clustering_primer
- Census data from Dublin:
  - ~4000 localities
  - Average values for: age, household size, education, car ownership,
  - Percentage values for: health, rent, employment, internet, marital status

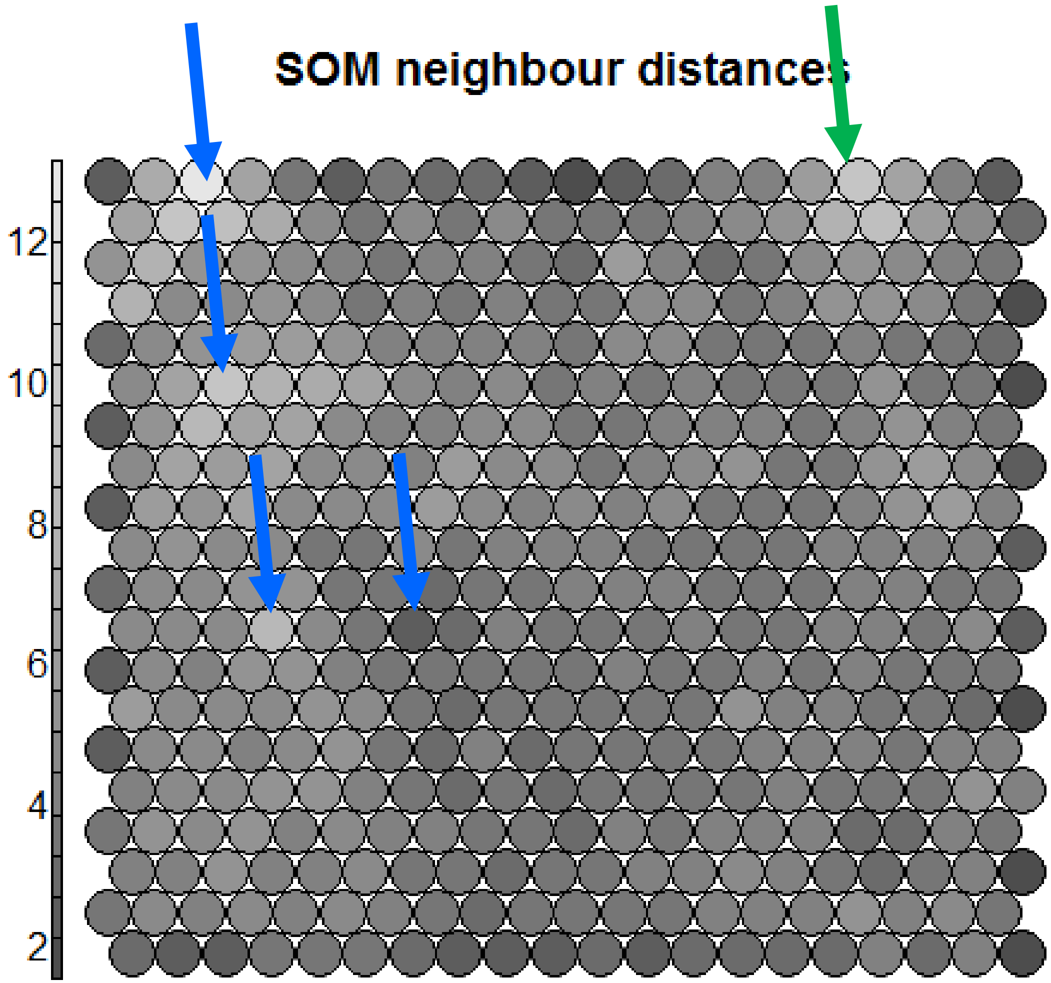
# SOM on Census Data

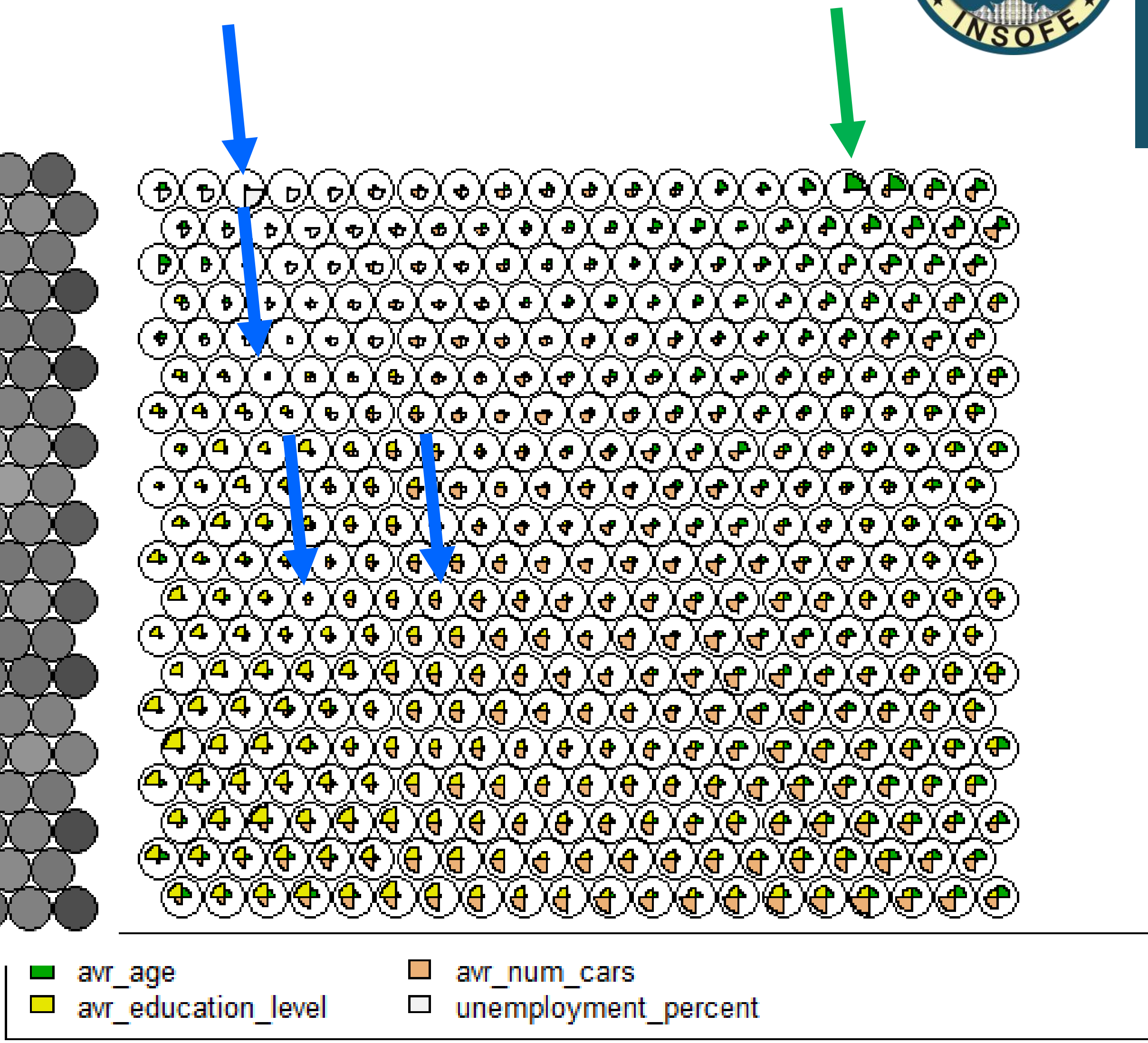



Number of data points mapping to each node.
Many blues/reds indicate too large/small model

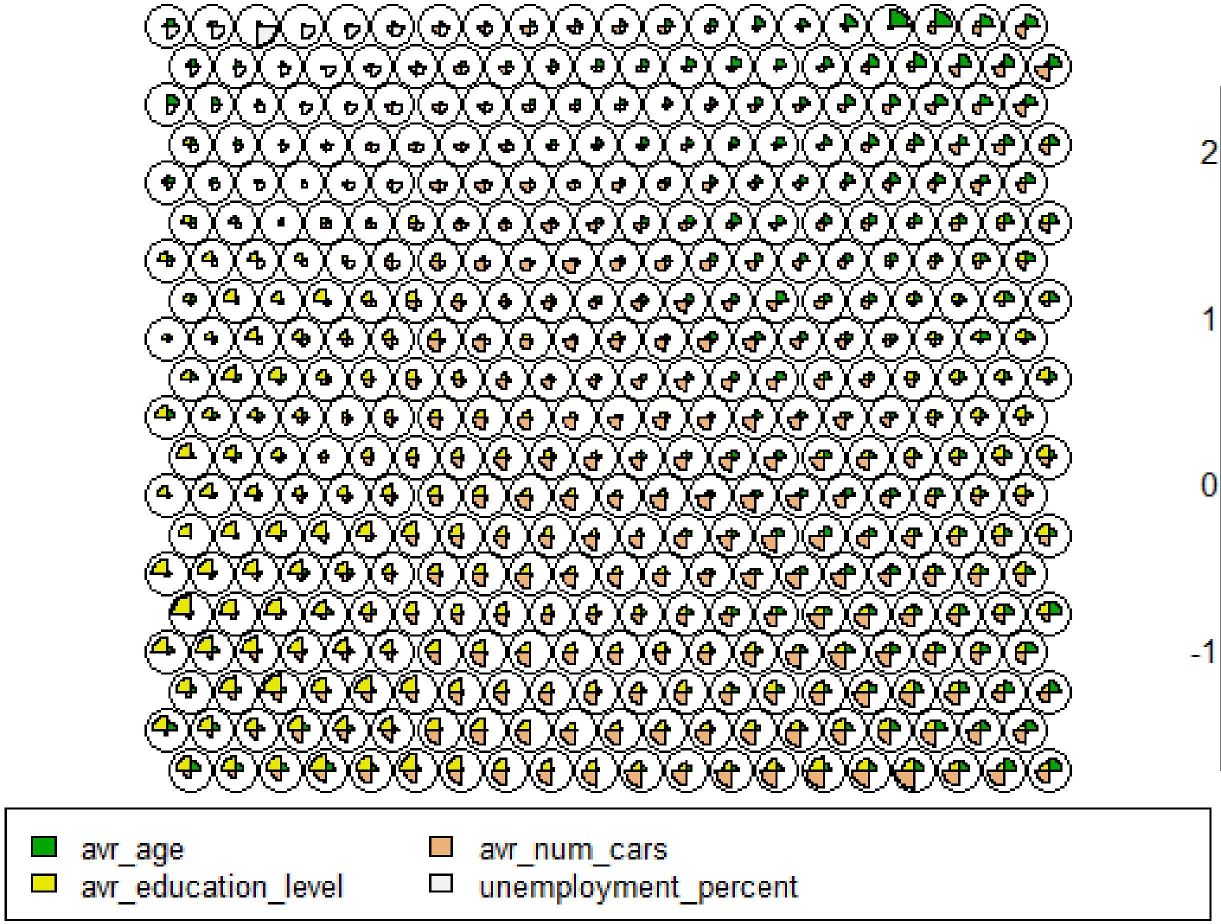
# SOM on Census Data

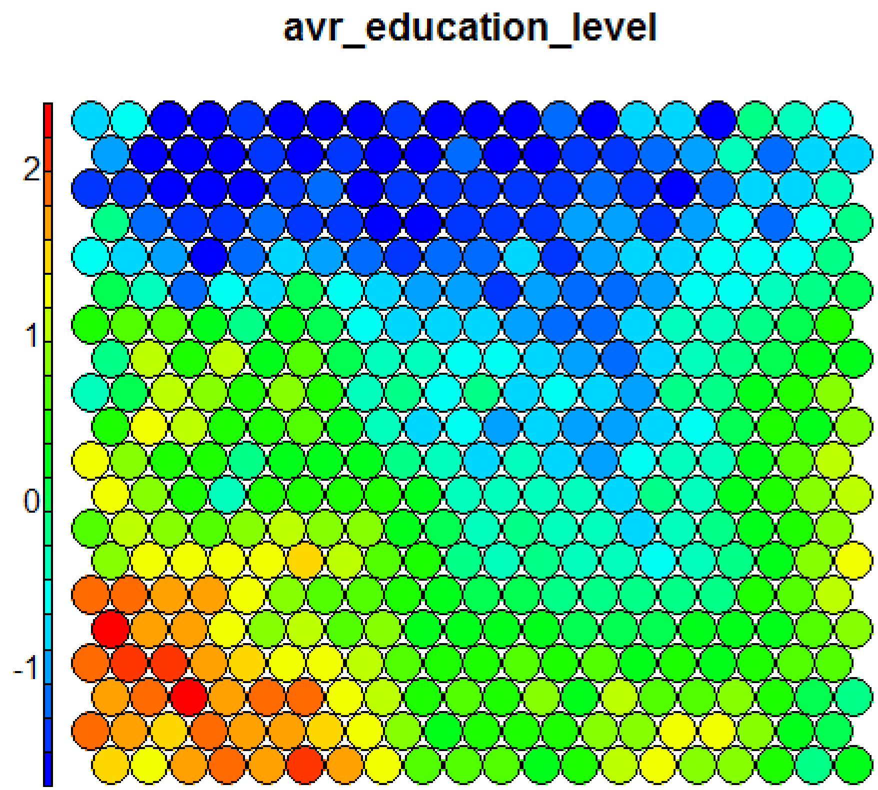



Interesting patterns can be seen

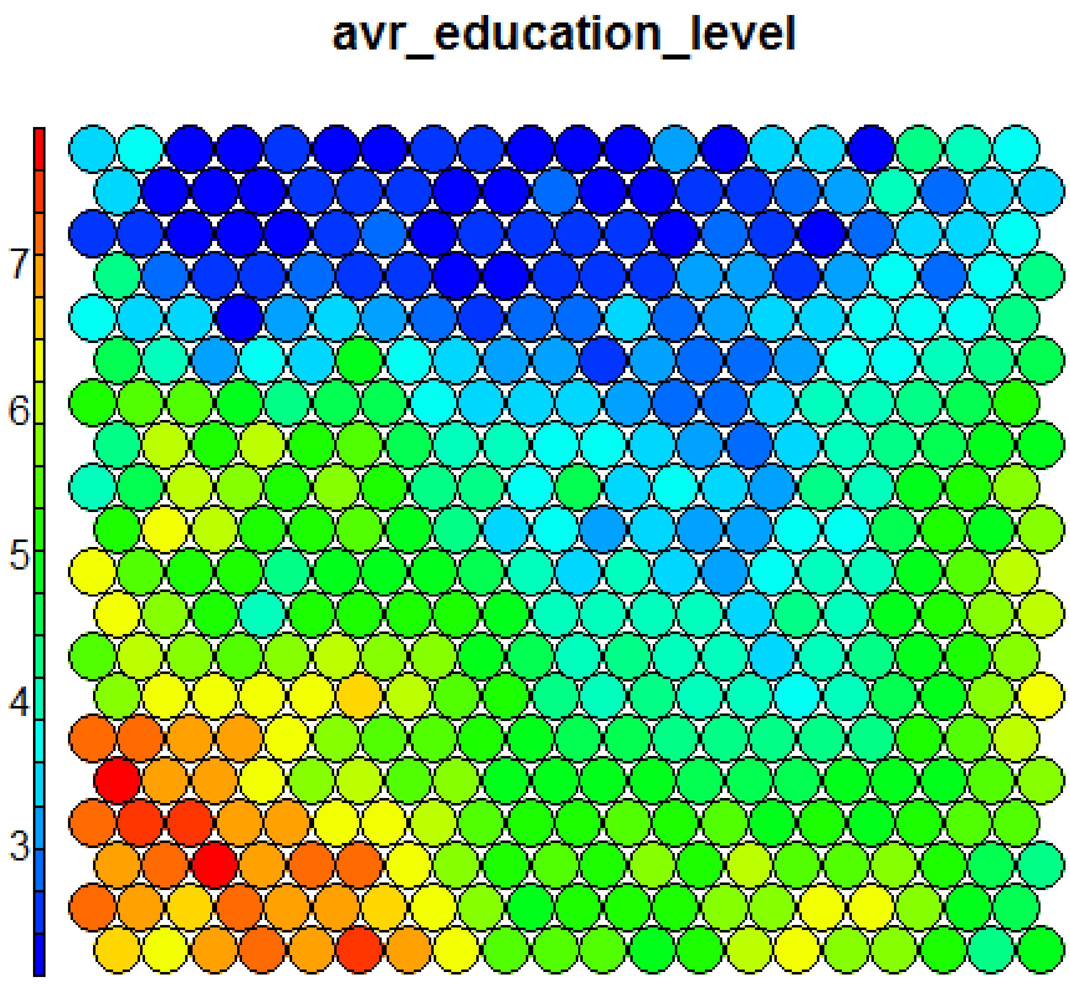
# SOM on Census Data

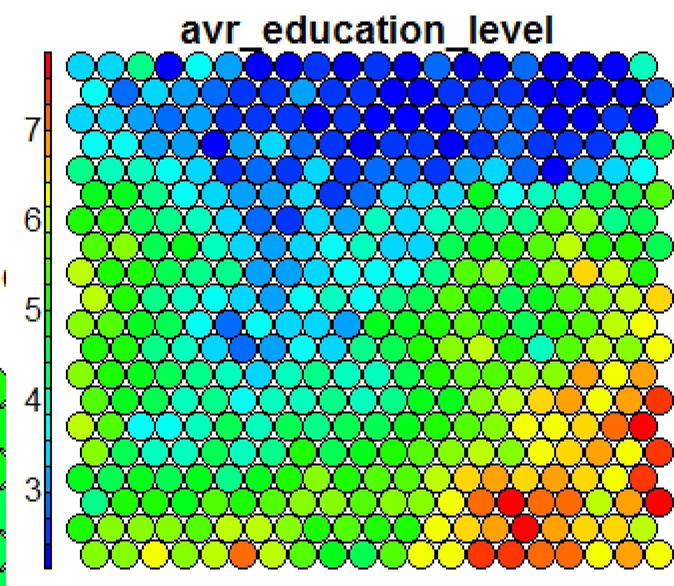


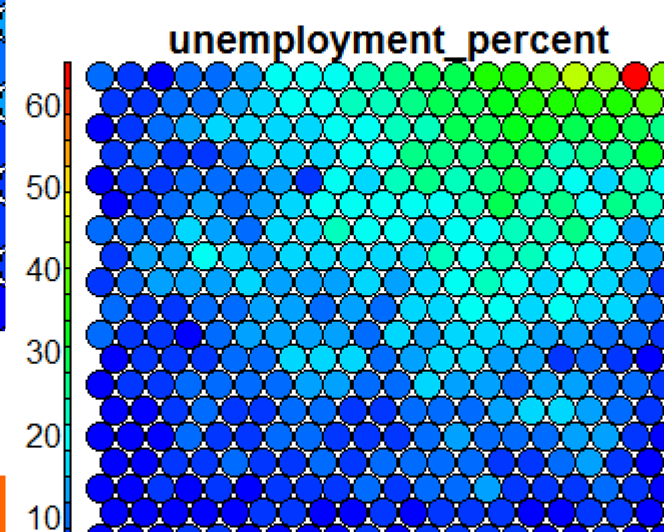




Heat map of average education level

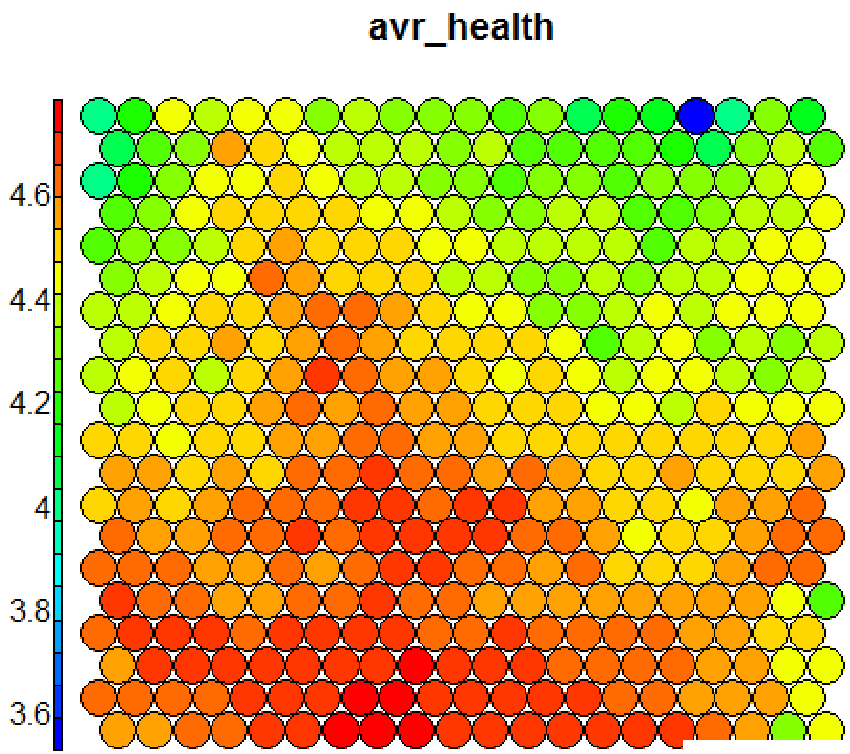
# SOM on Census Data

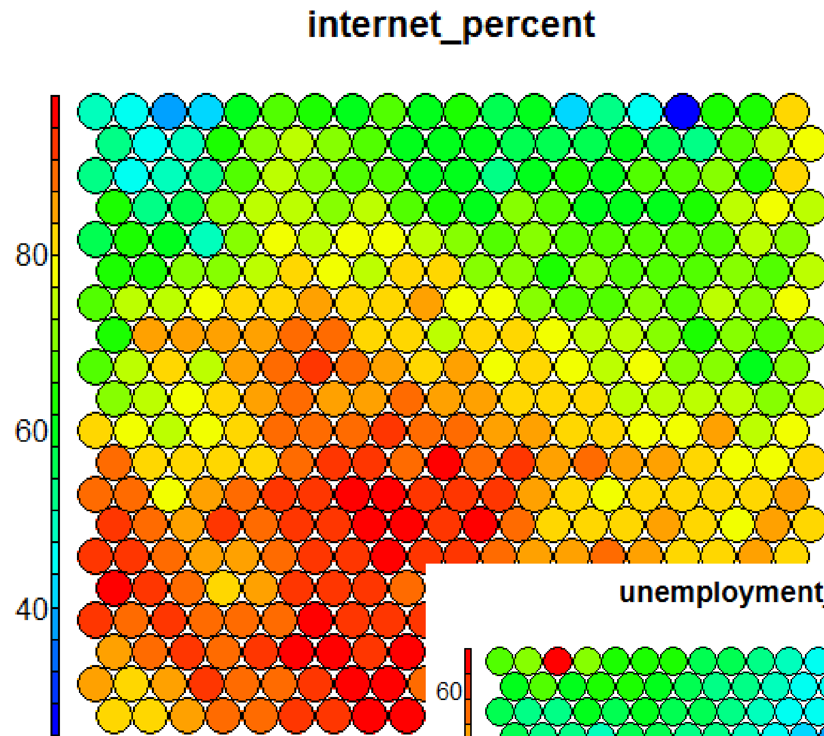


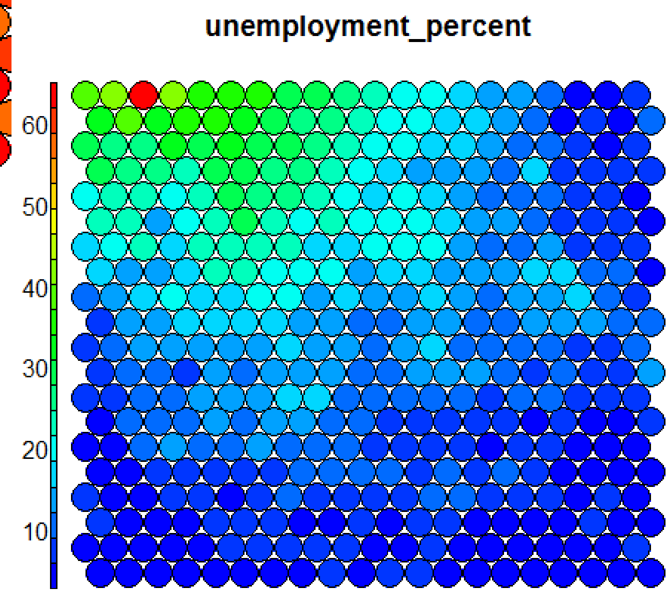



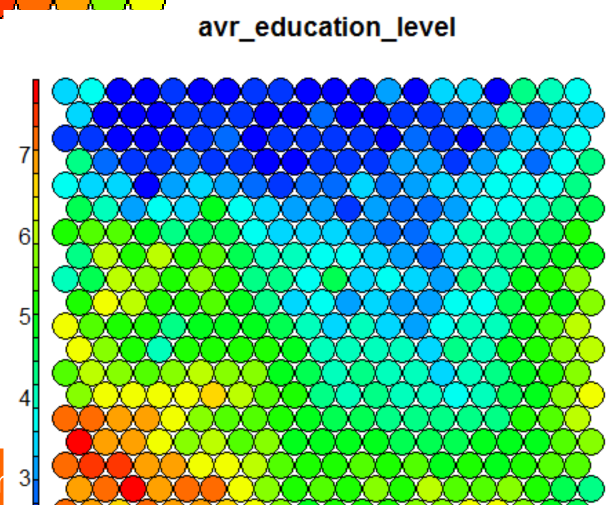



# SOM on Census Data



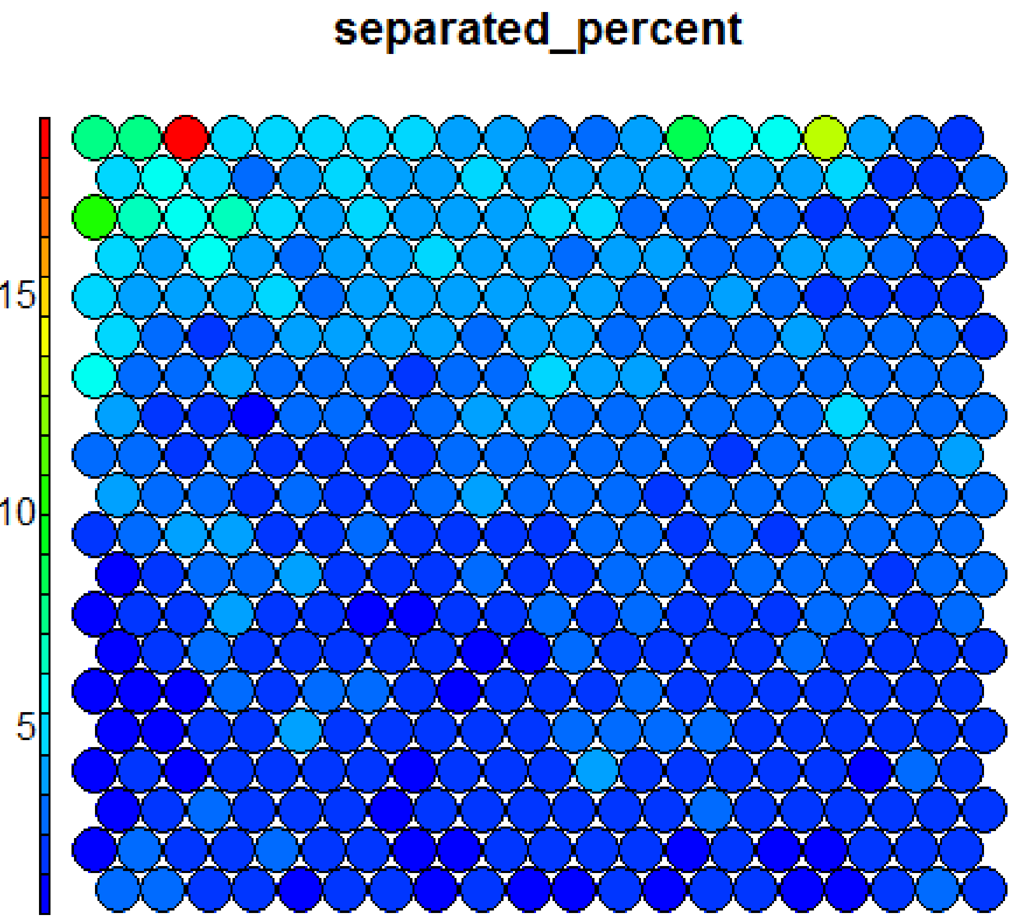



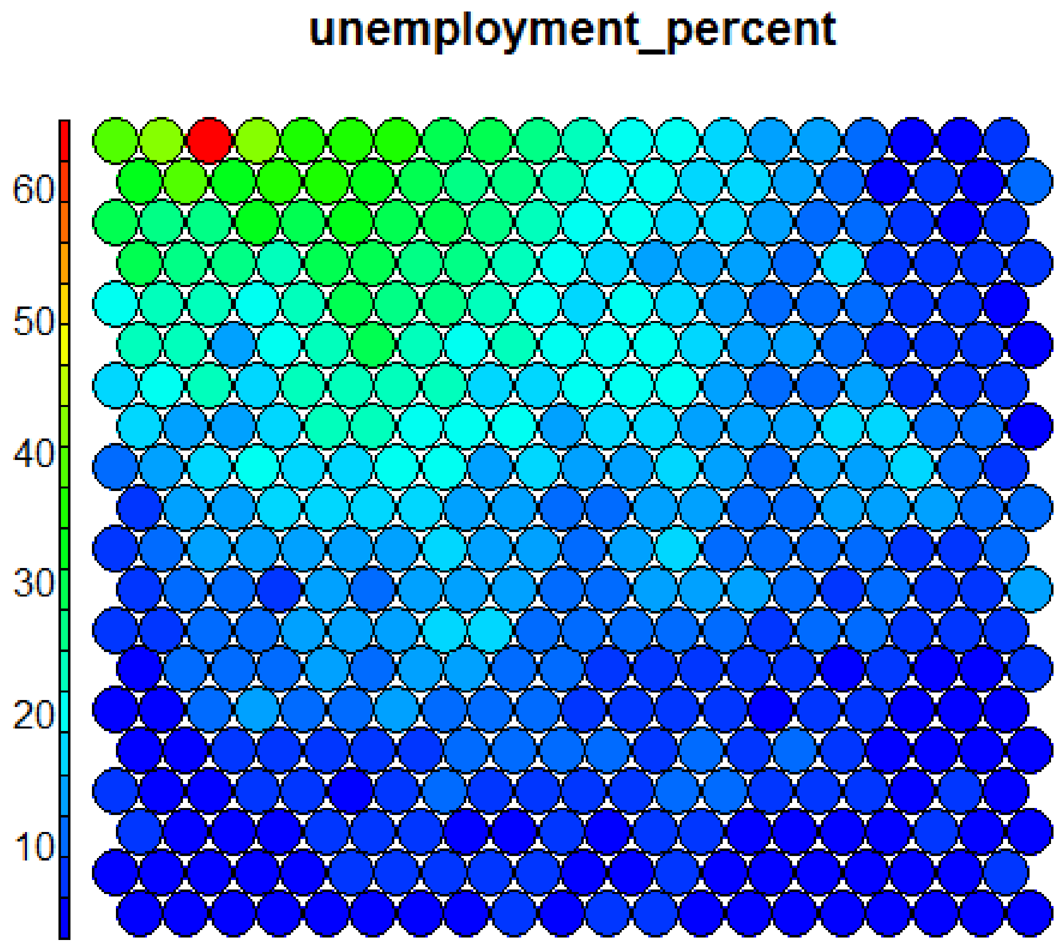



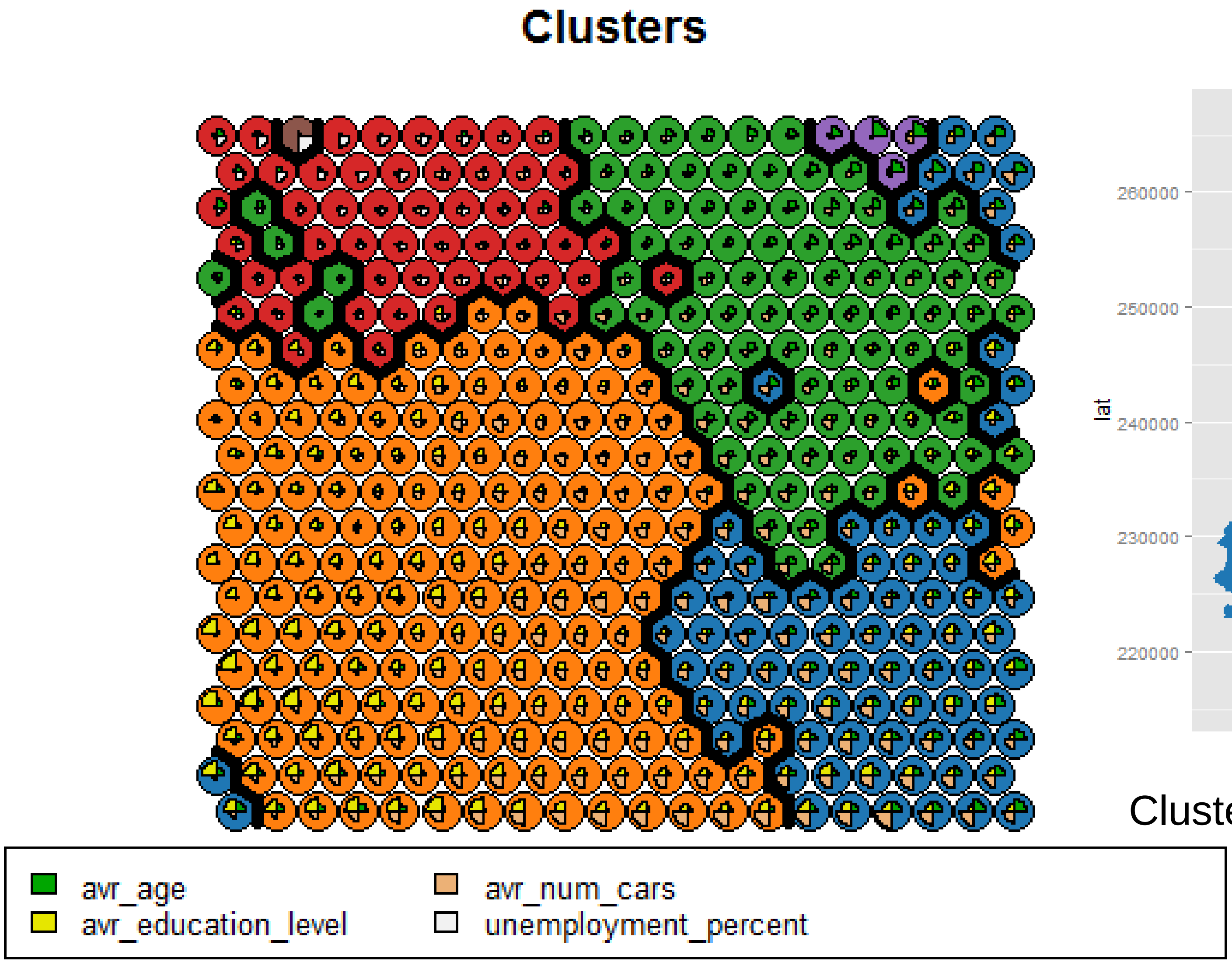

Class to take 15 minute break – run SOM code and suggest other comparisons

# SOM on Census Data

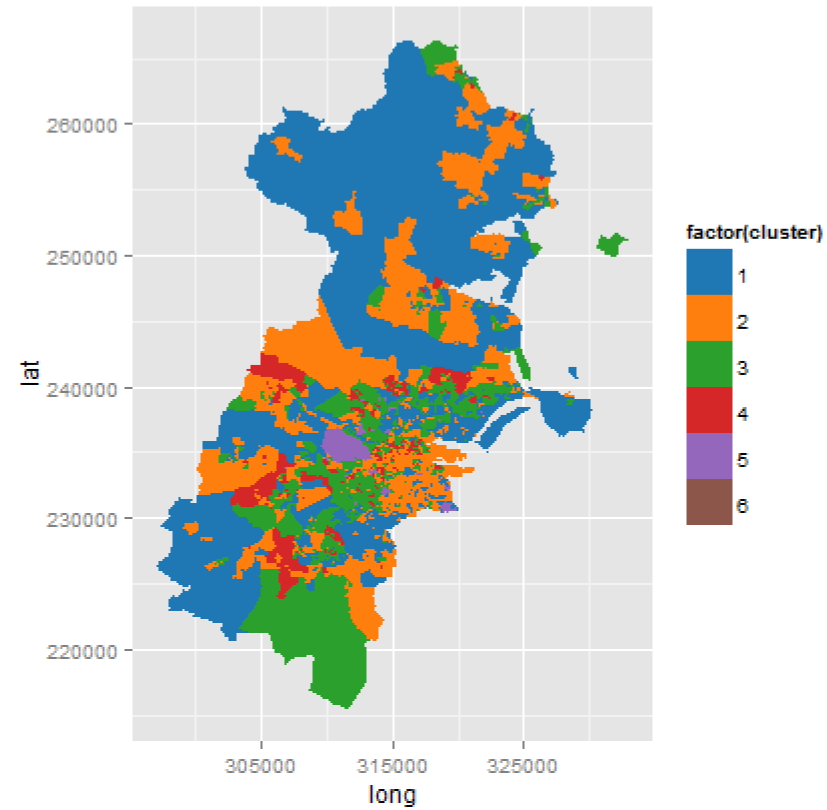


# SOM on Census Data




Cluster Nodes using Hierarchical clustering




Clusters on the map of Dublin

# References for Kohonen SOMs

- Online tutorials:
  - https://www.youtube.com/watch?v=LjJeT7rwvF4
  - http://www.cs.bham.ac.uk/~jxb/INC/l16.pdf
  - http://www.r-bloggers.com/self-organising-maps-for-customer-segmentation-using-r/
  - http://ssdi.di.fct.unl.pt/aadm/aadm1011/slides/LectureSOM.pdf
  - http://www.academia.edu/11322466/Using_R_to_Map_Crime_Density_and_Demographics_in_Boston_from_2012-2014

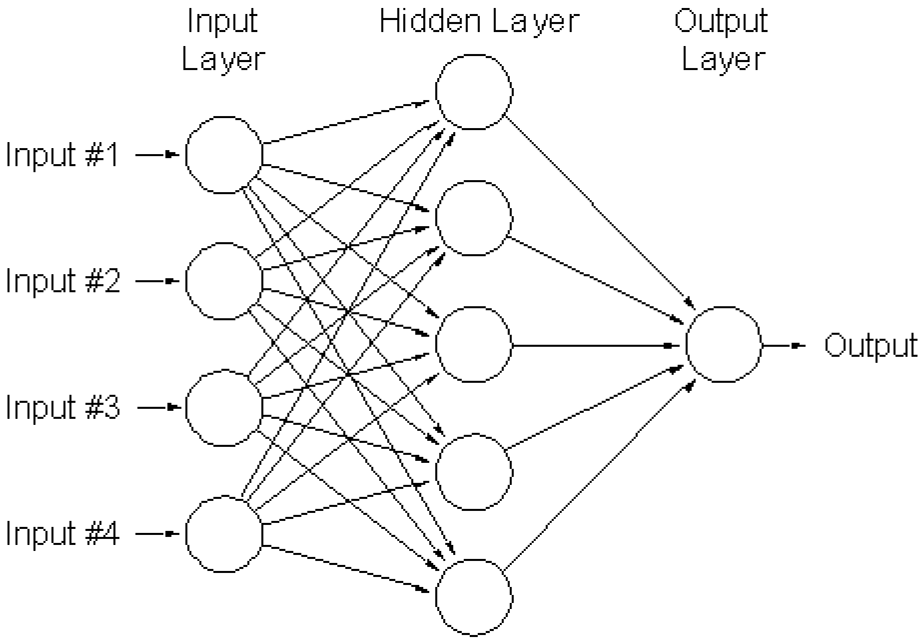
# ANN died too

- http://www.newyorker.com/news/news-desk/is-deep-learning-a-revolution-in-artificial-intelligence

- They learned slowly and inefficiently, and as Steven Pinker and I showed, couldn’t master even some of the basic things that children do, like learning the past tense of regular verbs. By the late nineteen-nineties, neural networks had again begun to fall out of favor.

- They need trained samples and that is not how we learn

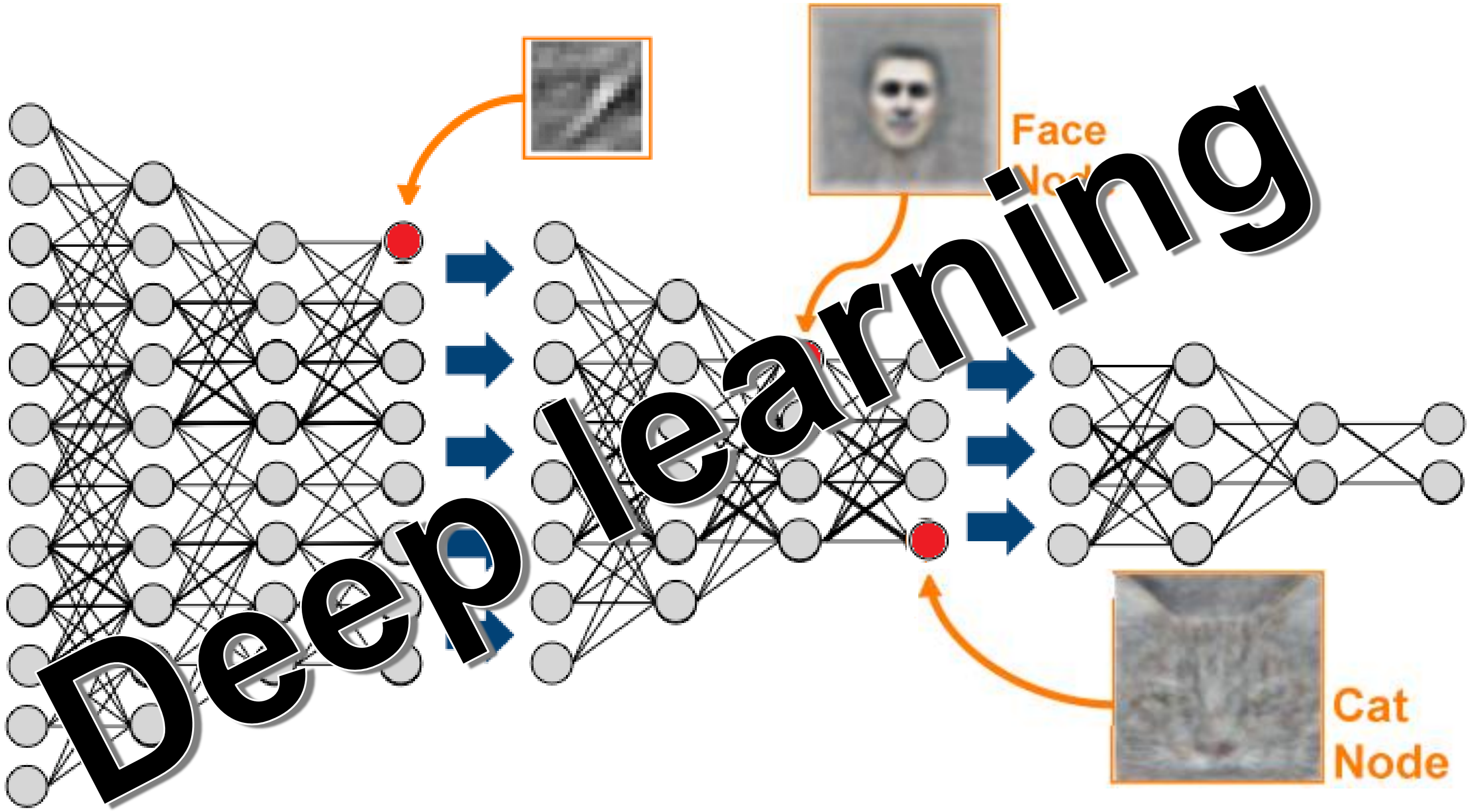
# Brain and Artificial Neuralnet



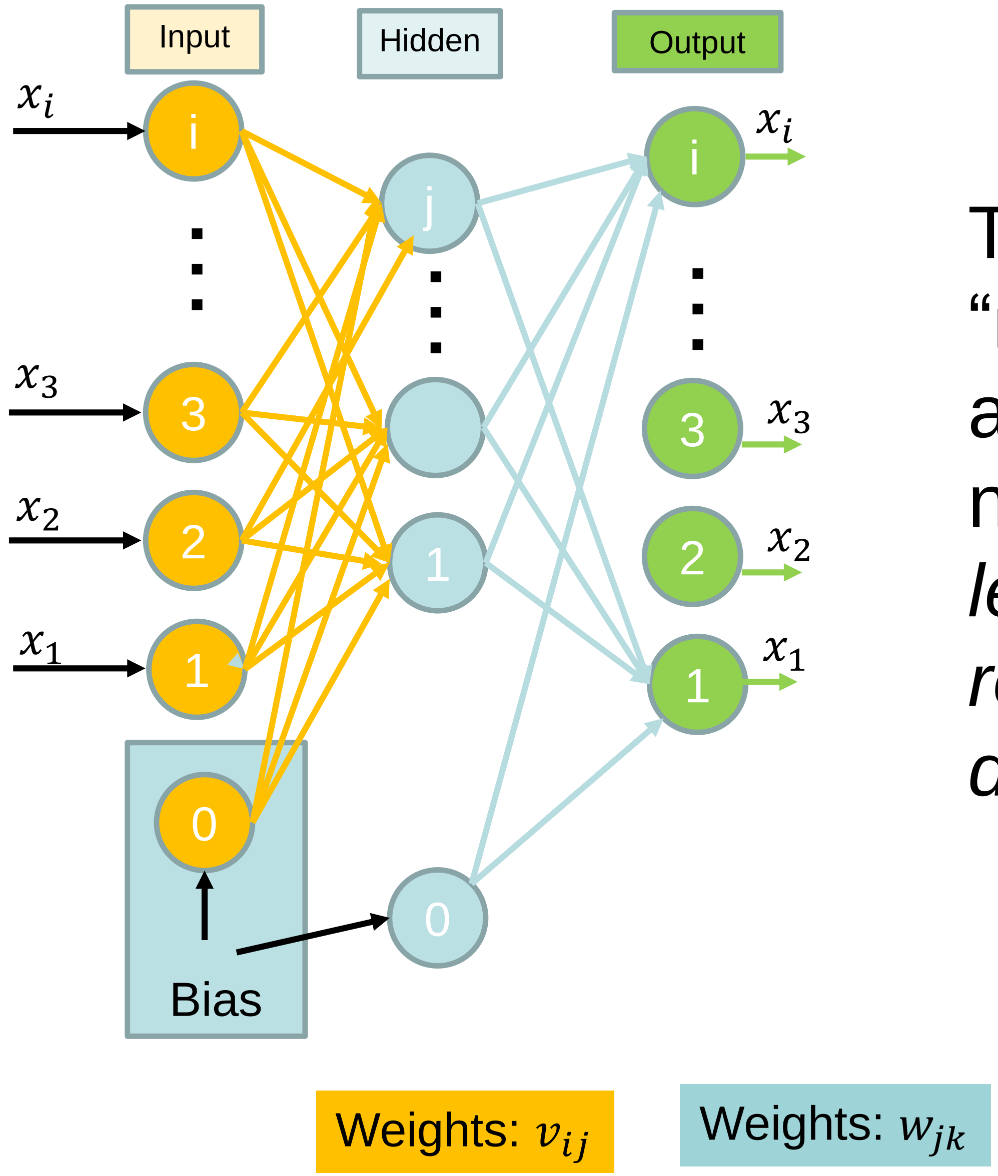
# Adding more layers

- Vanishing gradients: as we add more and more hidden layers, backpropagation becomes less and less useful in passing information to the lower layers. In effect, as information is passed back, the gradients begin to vanish and become small relative to the weights of the networks.

- Overfitting: perhaps the central problem in Machine Learning. Briefly, overfitting describes the phenomenon of fitting the training data *too* closely, maybe with hypotheses that are *too* complex. In such a case, your learner ends up fitting the training data really well, but will perform much, much more poorly on real examples.


Face Node
Cat Node
Deep learning

# Autoencoders



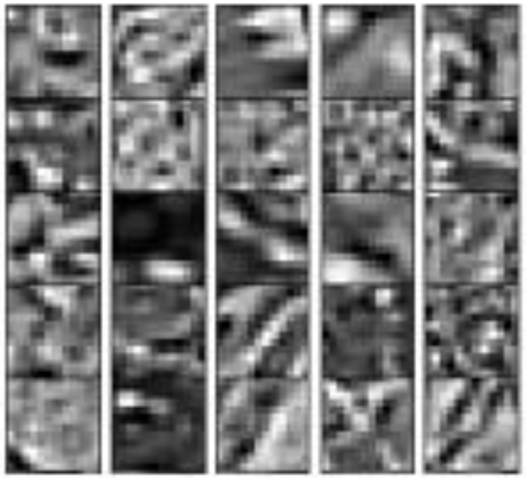

The network is trained to "recreate" the input. An autoencoder is a feed forward neural network which aims to *learn a compressed, distributed representation (encoding) of a dataset.*

# Autoencoders Additional Considerations

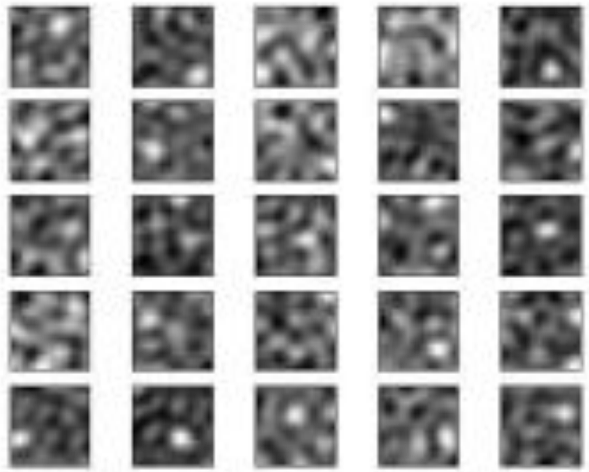
- Denoising:
  - For every sample X=($x_1, x_2, \ldots .x_n$), add noise to some $x_i$ to create $\bar{X}$.
  - During training, use $\bar{X}$ at input and X at output
  - Noise can be:
    - Gaussian
    - Set some % of input to zeros
    - Salt and Pepper (Similar to jitter)
  - http://deeplearning.net/tutorial/dA.html
  - http://jmlr.csail.mit.edu/papers/volume11/vincent10a/vincent10a.pdf

# Autoencoders Additional Considerations

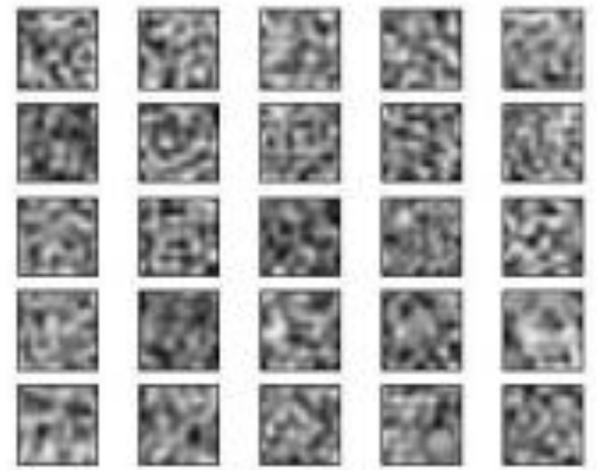
- Feature extraction:



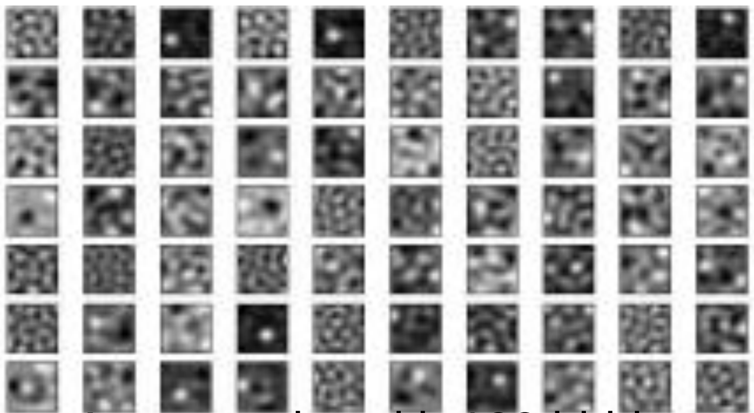
Input patches



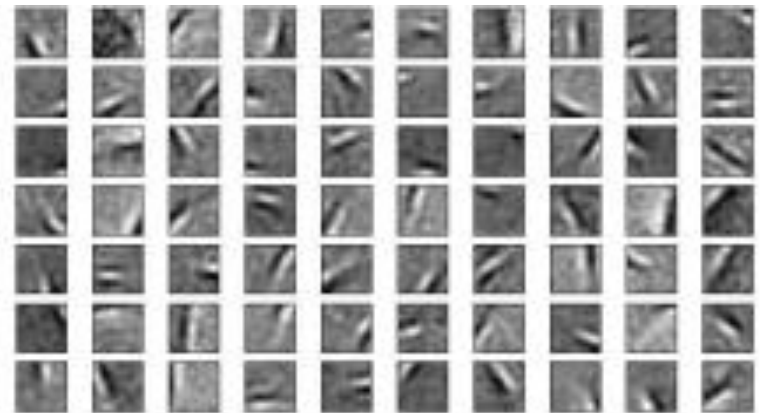
Autoencoder with very few hidden nodes (50)



Autoencoder with too many hidden nodes (200)



Autoencoder with 100 hidden nodes



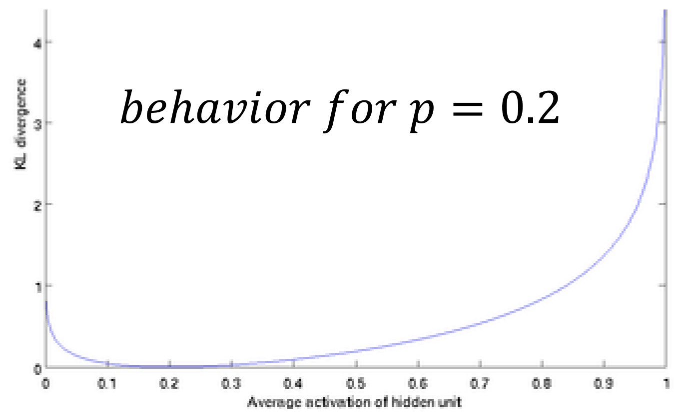
Autoencoder with 100 hidden nodes + Gaussian noise

# Autoencoders Additional Considerations

- Reducing number of connections: $h_j = f\left(\sum_{i=0}^{I} x_i v_{ij}\right)$
  - $Let\ h_j^m be\ activation\ for\ sample\ X_m$
  - Average activation of hidden node j: $\widehat{p_j} = \sum_{m=1}^{M} h_j^m$
  - Enforce that each hidden node be active only a small % of time (e.g. Sparsity Parameter $p$ =0.05, so 5%)

$$sparsity_j = \sum_{j=1}^{j} p\ \log\frac{p}{\widehat{p_j}} + (1-p)log\frac{1-p}{1-\widehat{p_j}}$$

$$w_{jk} \leftarrow w_{jk} + \eta * \ \delta\_input_k\ * h_j +\ \beta\ * sparsity_j$$

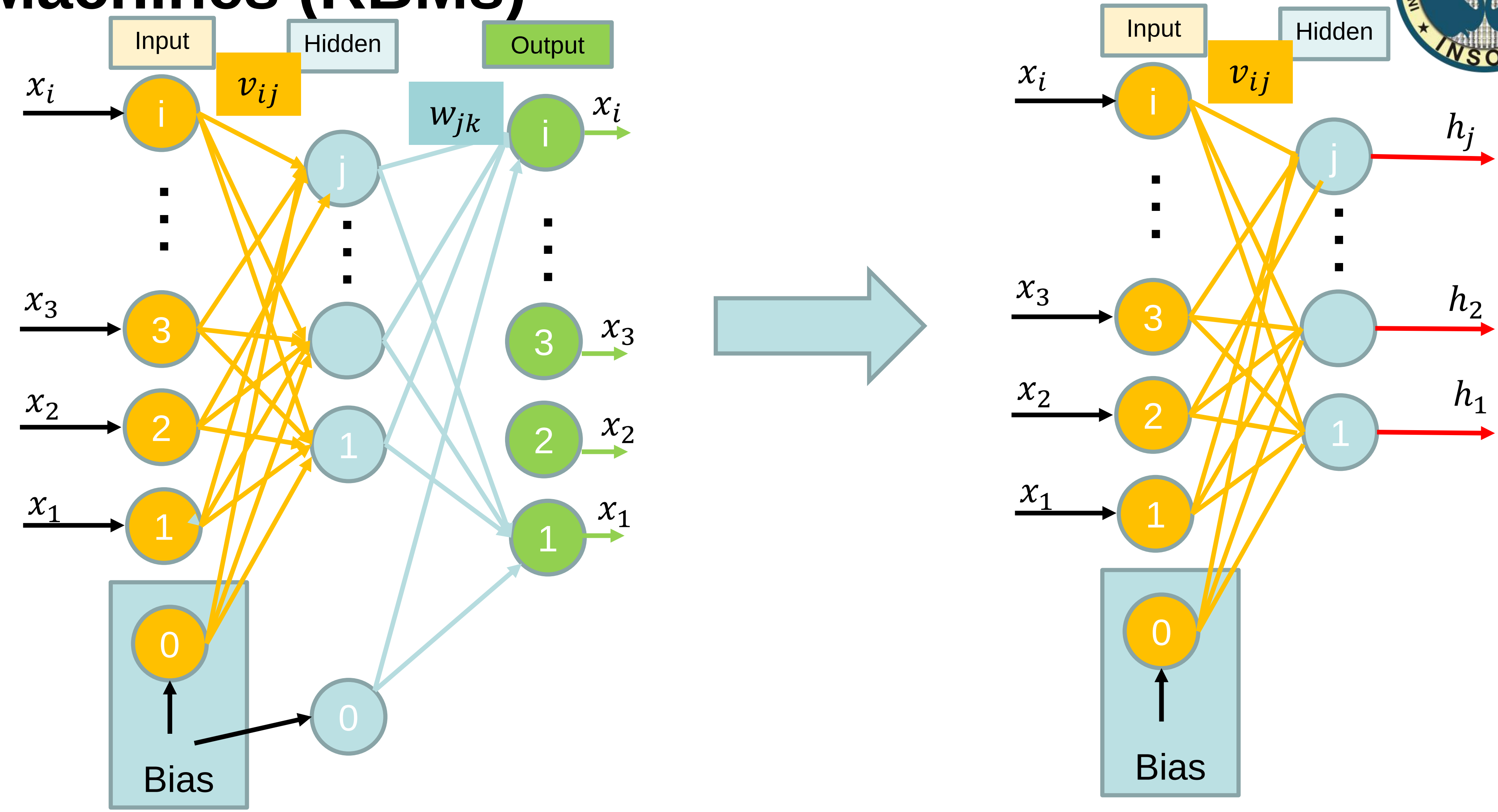


# Autoencoders

- The intuition behind this architecture is that the network will not learn a “mapping” between the training data and its labels, but will instead learn the *internal structure* and features of the data itself.

- Usually, the number of hidden units is smaller than the input/output layers, which forces the network to learn only the most important features and achieves a dimensionality reduction.

# Autoencoders to Restricted Boltzman Machines (RBMs)



# Autoencoders to RBMs

- Initialize all weights
  - +ve and –ve random values
- Forward phase:

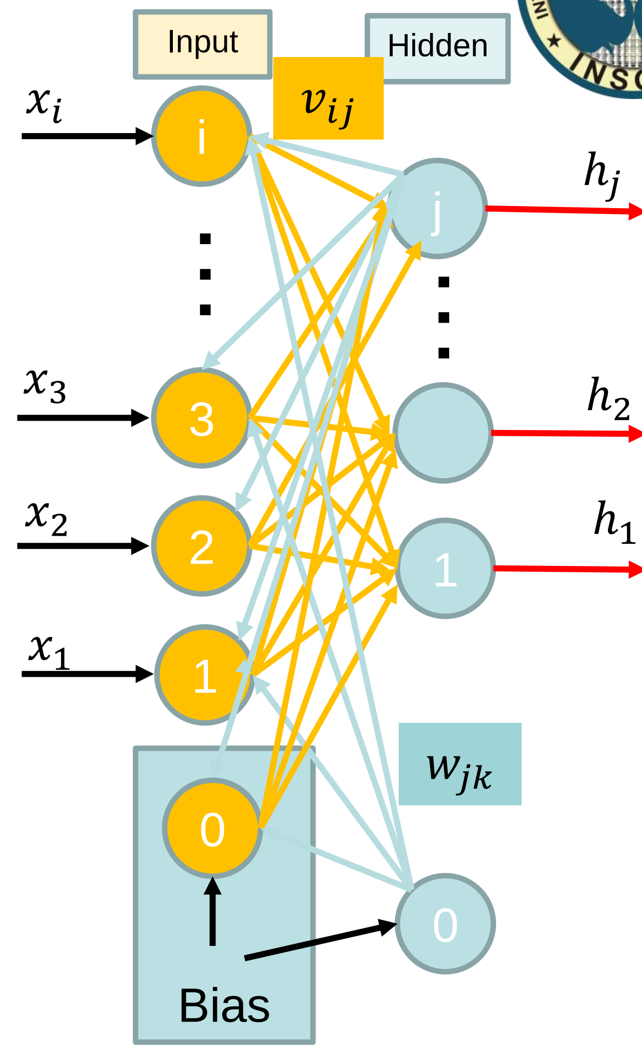
$$h_j = f\left(\sum_{i=0}^{I} x_i v_{ij}\right)$$

$$\tilde{x}_k = f\left(\sum_{j=0}^{J} h_j w_{jk}\right)$$

- Backward phase:

$$\delta\_input_k = (x_k - \tilde{x}_k) * \tilde{x}_k * (1 - \tilde{x}_k)$$

$$\delta\_hidden_j = h_j * (1 - h_j) \sum_{k=0}^{K} w_{jk} * \delta\_input_k$$



# Autoencoders to RBMs

- Forward Phase:

$$h_j = f\left(\sum_{i=0}^{I} x_i v_{ij}\right)$$

$$\tilde{x}_k = f\left(\sum_{j=0}^{J} h_j w_{jk}\right)$$

- Backward phase:

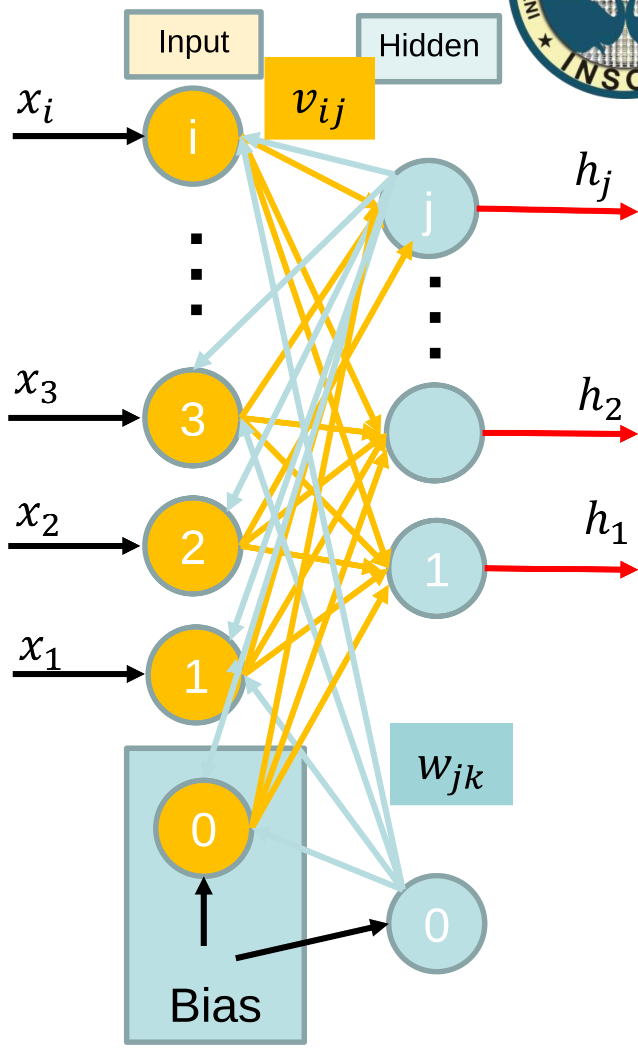
$$\delta\_input_k = (x_k - \tilde{x}_k) \ * \tilde{x}_k * (1 - \tilde{x}_k)$$

$$\delta\_hidden_j = h_j \ * (1 - h_j) \ \sum_{k=0}^{K} w_{jk} * \delta\_output_k$$

- Update weights:

$$w_{jk} \leftarrow w_{jk} + \eta * \ \delta\_input_k \ * h_j$$

$$v_{ij} \leftarrow v_{ij} + \eta * \ \delta\_hidden_j \ * x_i$$

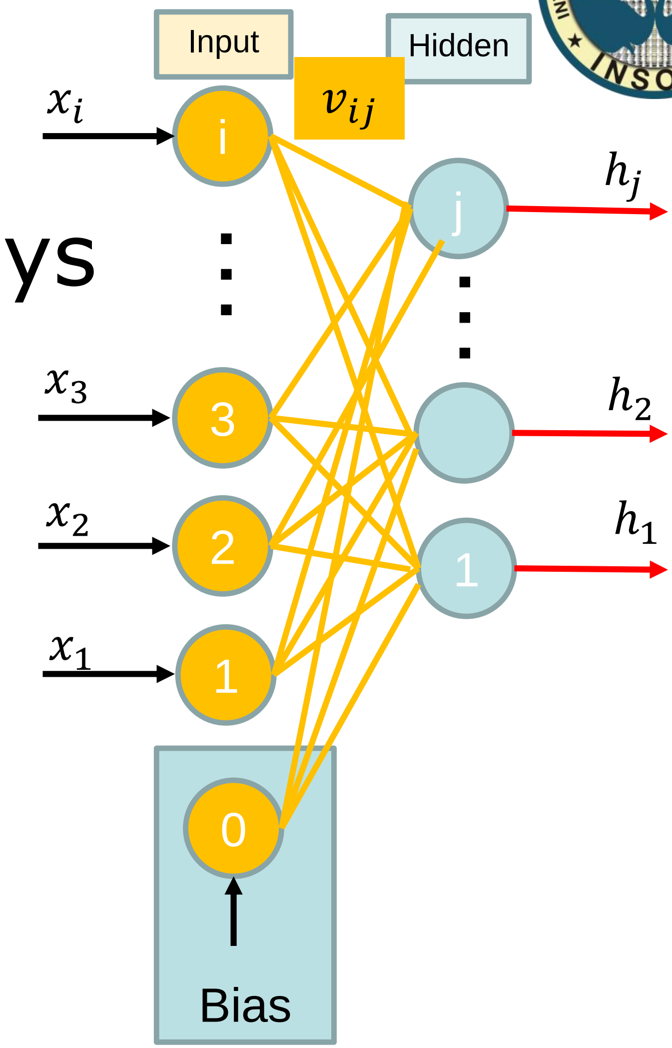


# Restricted Boltzman Machines

- Weights ($v_{ij}$) work both ways

$$Activation, a_j = \sum_{i=0}^{I} x_i v_{ij}$$

$$h_j = \sigma(a_j) \qquad \sigma(a_j) = \frac{1}{1 + e^{-a_j}}$$




- Hidden unit "j" is:
  - On with probability $h_j$
  - Off with probability $1 - h_j$
- i.e Units with similar activations are on or off

# Restricted Boltzman Machines – Training Contrastive Divergence

- Set input to one sample

$$Activation, a_j = \sum_{i=0}^{I} x_i v_{ij}$$

$$h_j = \sigma(a_j)$$

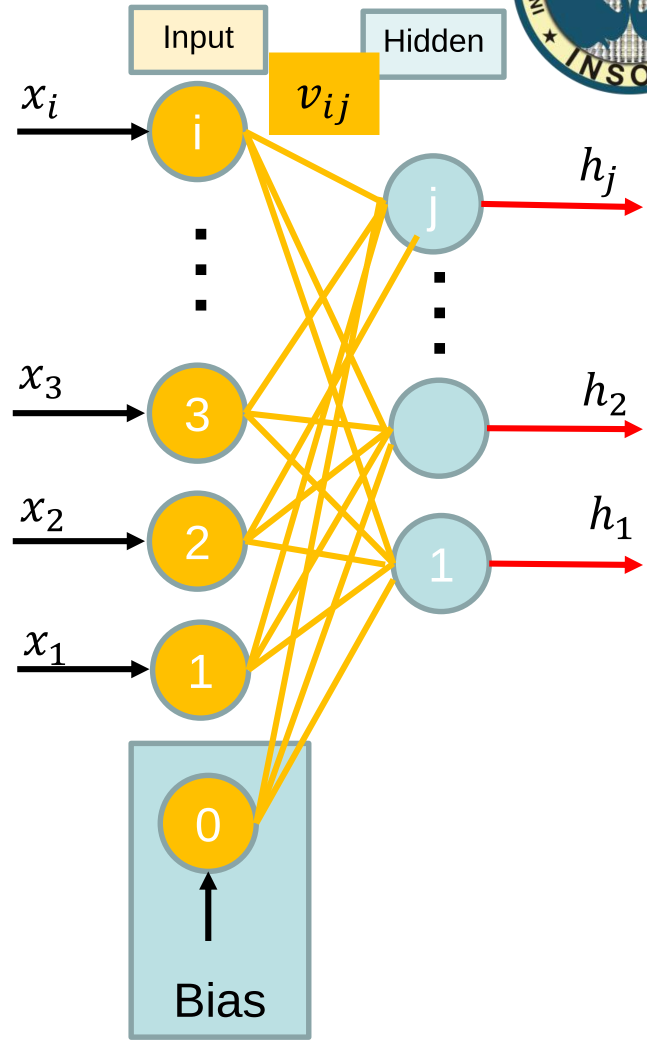
$$Activation, a_i = \sum_{j=1}^{J} h_i v_{ij} \qquad Activation, \widetilde{a}_j = \sum_{i=0}^{I} \widetilde{x}_i v_{ij}$$

$$\widetilde{x}_i = \sigma(a_i) \qquad \widetilde{h}_j = \sigma(\widetilde{a}_j)$$

- Compute:

$$positive_{ij} = x_i * h_j \quad negative_{ij} = \widetilde{x}_i * \widetilde{h}_j$$

- Update: $v_{ij}(t+1) = v_{ij}(t) + \eta * (positive_{ij} - negative_{ij})$



# Why does this make sense

$$positive_{ij} = x_i * h_j \qquad negative_{ij} = \widetilde{x}_i * \widetilde{h}_j$$

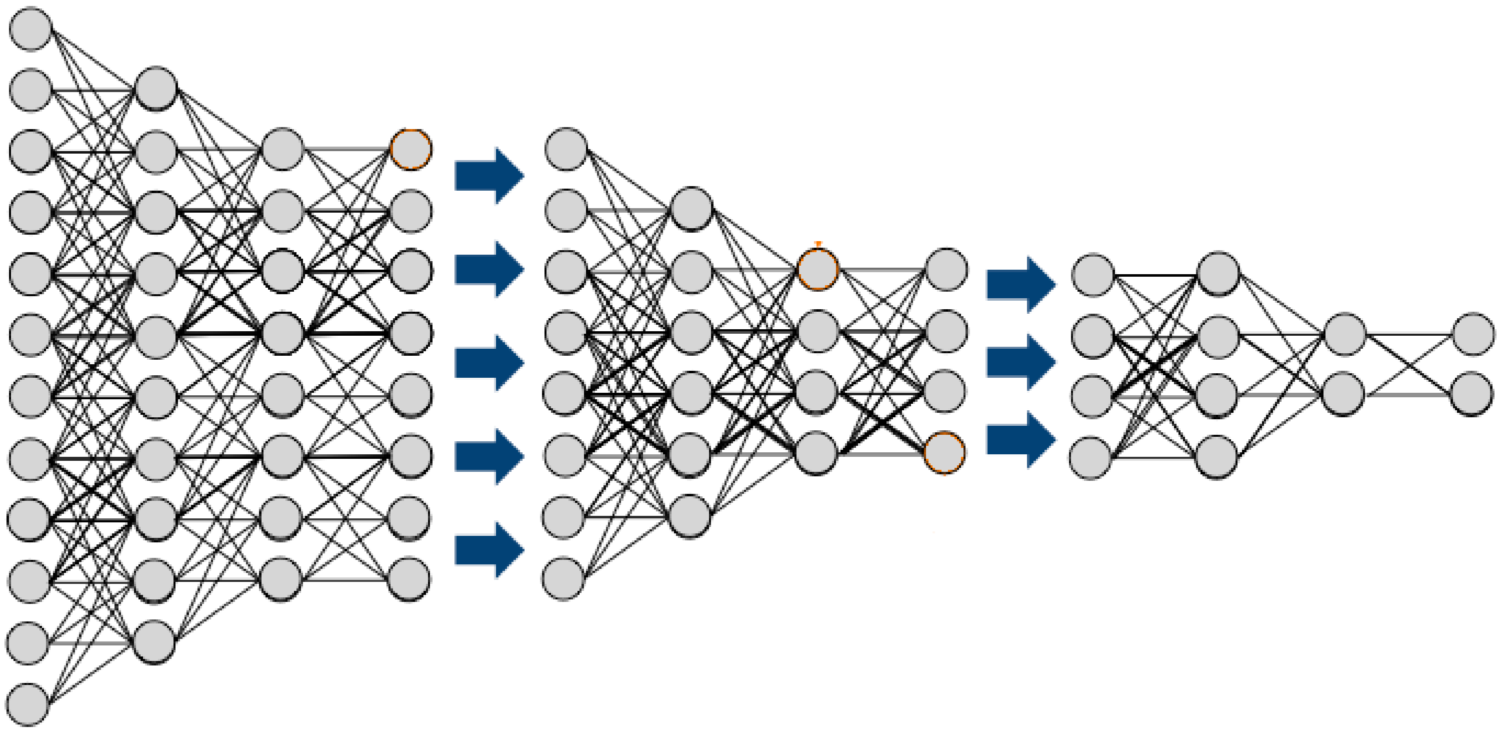
- $positive_{ij}$ measures the association between the ith and jth unit that we *want* the network to learn from our training examples;
- $negative_{ij}$ measures the association that the network *itself* generates (or "daydreams" about) when no units are fixed to training data.
- So by adding $positive_{ij} - negative_{ij}$ to each edge weight, we're helping the network's daydreams better match the reality of our training examples.
- This update rule is called contrastive divergence, which is basically a funky term for "approximate gradient descent".

# RBM – additional references

- http://blog.echen.me/2011/07/18/introduction-to-restricted-boltzmann-machines/
- http://imonad.com/rbm/restricted-boltzmann-machine/
- http://www.cs.toronto.edu/~hinton/absps/guideTR.pdf

# Stack them

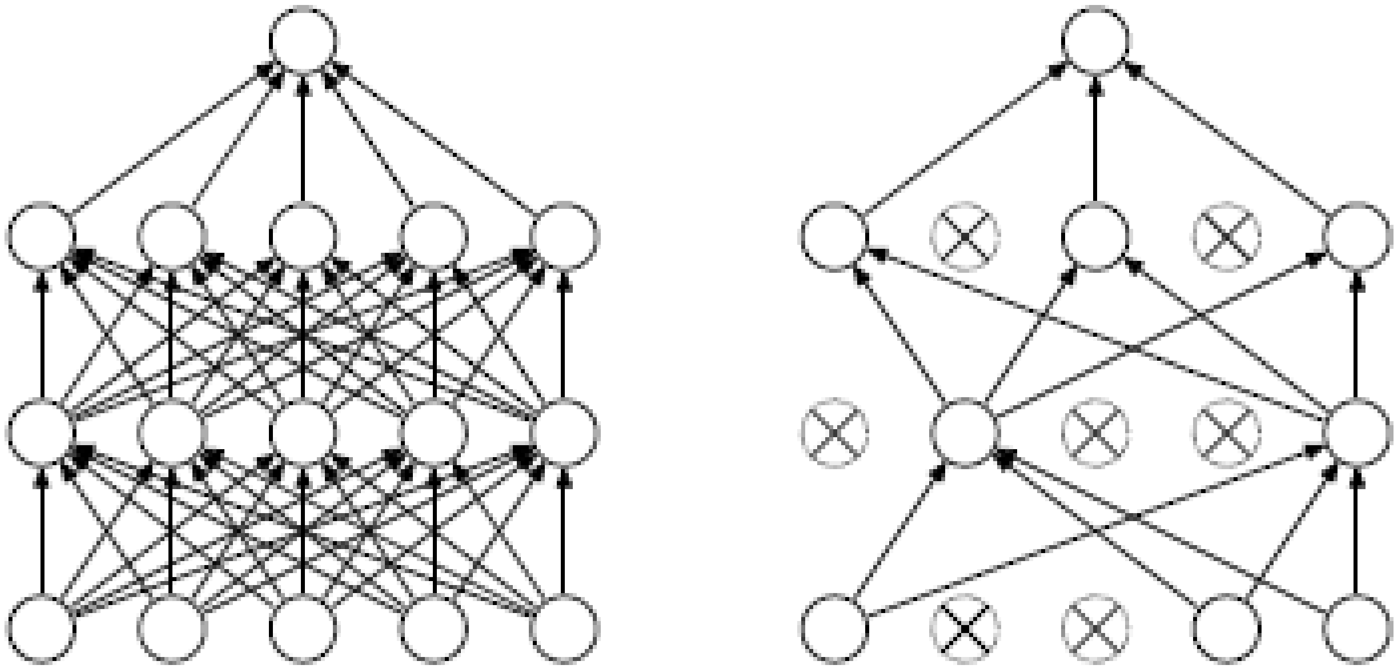
- Hidden layer 1 becomes visible layer for the hidden layer 2 and one can keep stacking



# PRACTICAL TIPS

# Drop outs

**http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf**



(a) Standard Neural Net (b) After applying dropout.

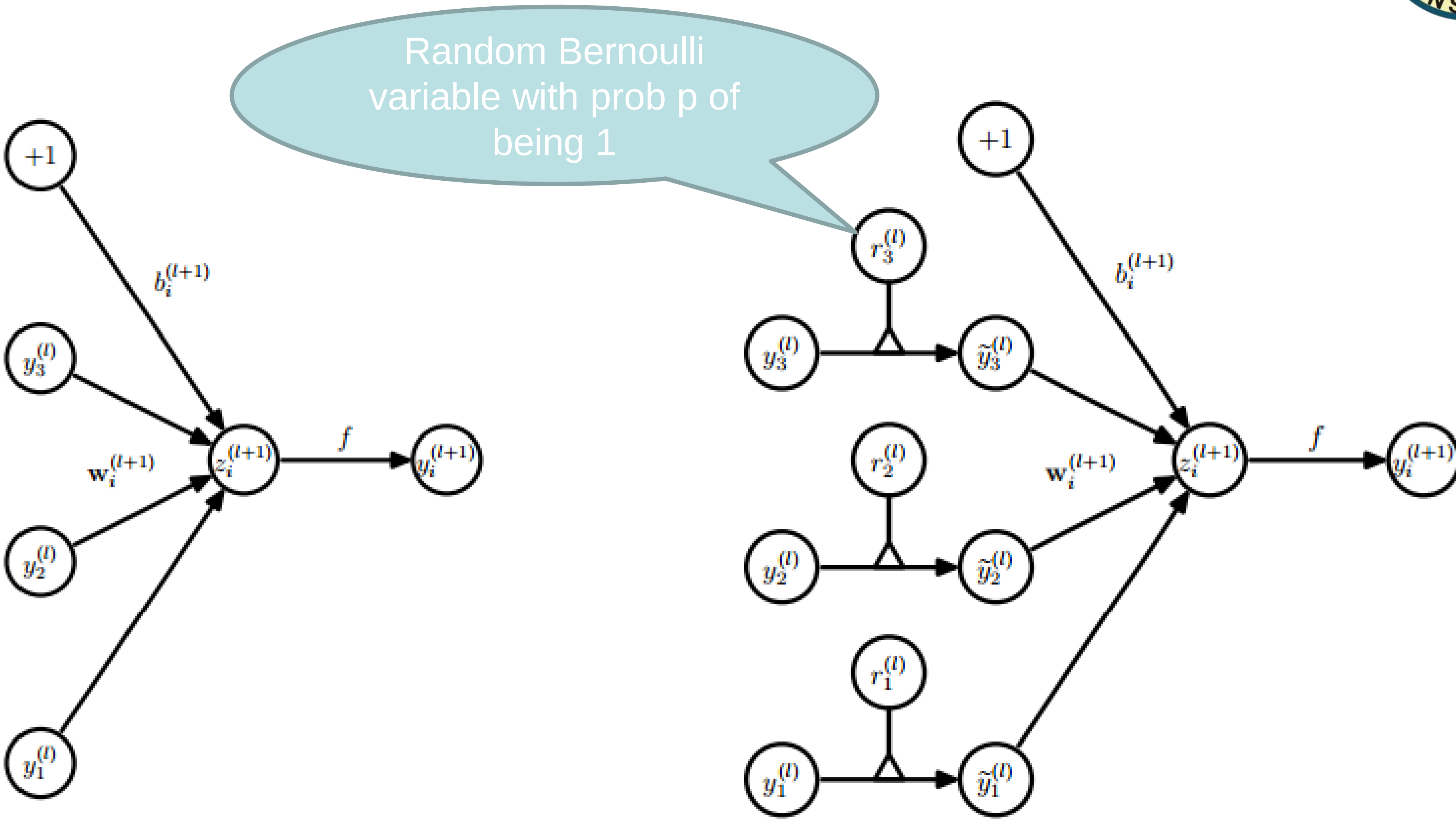
# Practical Tips

- Stop sooner:
  - Flat learning rate
  - Test/Holdout accuracies
- Dropout
  - Alter inputs using a random distribution
- Sparse Autoencoders
  - Hidden nodes should be less than inputs
  - Use sparsity during learning

$$sparsity_j = \sum_{j=1}^{j} p \ \log\frac{p}{\hat{p}_j} + (1-p)log\frac{1-p}{1-\hat{p}_j} \qquad w_{jk} \leftarrow w_{jk} + \eta * \ \delta\_input_k \ * h_j + \ \beta \ * sparsity_j$$

# Drop out

**http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf**




(a) Standard network

(b) Dropout network

CSE 7405c

# Drop out

http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

### 6.1.1 MNIST

| Method | Unit Type | Architecture | Error % |
|---|---|---|---|
| Standard Neural Net (Simard et al., 2003) | Logistic | 2 layers, 800 units | 1.60 |
| SVM Gaussian kernel | NA | NA | 1.40 |
| Dropout NN | Logistic | 3 layers, 1024 units | 1.35 |
| Dropout NN | ReLU | 3 layers, 1024 units | 1.25 |
| Dropout NN + max-norm constraint | ReLU | 3 layers, 1024 units | 1.06 |
| Dropout NN + max-norm constraint | ReLU | 3 layers, 2048 units | 1.04 |
| Dropout NN + max-norm constraint | ReLU | 2 layers, 4096 units | 1.01 |
| Dropout NN + max-norm constraint | ReLU | 2 layers, 8192 units | 0.95 |
| Dropout NN + max-norm constraint (Goodfellow et al., 2013) | Maxout | 2 layers, $(5 \times 240)$ units | 0.94 |
| DBN + finetuning (Hinton and Salakhutdinov, 2006) | Logistic | 500-500-2000 | 1.18 |
| DBM + finetuning (Salakhutdinov and Hinton, 2009) | Logistic | 500-500-2000 | 0.96 |
| DBN + dropout finetuning | Logistic | 500-500-2000 | 0.92 |
| DBM + dropout finetuning | Logistic | 500-500-2000 | **0.79** |

# Sparse autoencoders

- What are the ideal properties of machine generated features
  - An example must be explained primarily by a few features
  - A feature must belong to only a few features
  - All features should have similar activities

# Deep learners as predictive machines

- Build a stacked autoencoders or RBMs
- Make the last layer as desired output
- Learn weights through one network level back propagation using already computed weights as initial weights

# A more interesting approach

- Use autoencoders, RBMs as feature generators
- Create 100s of features
- Run a simple linear model or a random forest (wopal wabbit or sofia)

# Review

$$h_j = f\left(\sum_{i=0}^{I} x_i v_{ij}\right)$$

$$y_k = f\left(\sum_{j=0}^{J} h_j w_{jk}\right)$$

$$w_{jk} \leftarrow w_{jk} + \eta * \ \delta\_output_k \ * h_j \ + \ \alpha * \ \Delta w_{jk}^{t-1}$$

$$v_{ij} \leftarrow v_{ij} + \eta * \ \delta\_hidden_i \ * x_i + \ \alpha * \Delta v_{ij}^{t-1}$$

- Neural nets:
  - Training rate, momentum, jitter, Rules for hidden nodes, sensitivity
- SOM
  - Clustering unsupervised data
- Autoencoders $w_{jk} \leftarrow w_{jk} + \eta * \ \delta\_input_k \ * h_j + \ \beta \ * sparsity_j$
  - Noisy input, sparsity, stacking for deep NNs
- RBMs
  - Contrastive divergence, drop outs

## International School of Engineering


The Oval, Level 2, iLabs Centre, Plot No. 18, Madhapur, Hyderabad - 500 081

For Individuals: +91-9177585755 or 040-65743991

For Corporates: +91-7893866005

Web: http://www.insofe.edu.in

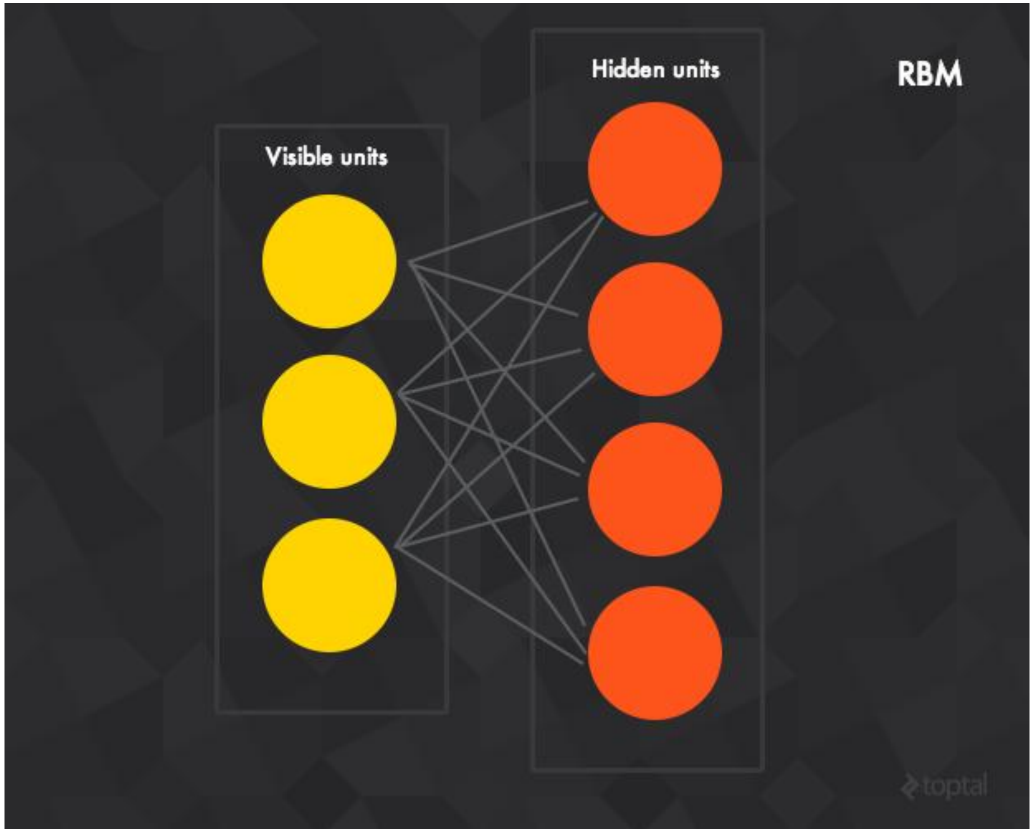
Facebook: https://www.facebook.com/intl.school.engineering

Twitter: https://twitter.com/IntlSchEngg

YouTube: http://www.youtube.com/InsofeVideos

SlideShare: http://www.slideshare.net/INSOFE

# Restricted boltzman machines




In their simple form, they are binary

Activation energy $a_i = \sum w_{ij} x_j$
$p_i = \sigma(a_i)$, where $\sigma$ is the is the logistic function.
Turn unit *i* on with probability of $p_i$

# Learning in RBMs:

**http://blog.echen.me/2011/07/18/introduction-to-restricted-boltzmann-machines/**
**http://www.toptal.com/machine-learning/an-introduction-to-deep-learning-from-perceptrons-to-deep-networks**

- Positive phase
  - Update the states of the hidden units using the logistic activation rule described above
  - Then for each edge $e_{ij}$, compute positive $(e_{ij})$=$x_i x_j$ (i.e., for each pair of units, measure whether they're both on).

# Negative phase

- Reconstruct the visible units in a similar manner: for each visible unit, compute its activation energy $a_i$, and update its state. (Note that this *reconstruction* may not match the original preferences.)
- Then update the hidden units again, and compute Negative($e_{ij}$)=$x_i x_j$ for each edge.

# Weight updates

- Update the weight of each edge $e_{ij}$ by setting $w_{ij} = w_{ij} + L(\text{Positive}(e_{ij}) - \text{Negative}(e_{ij}))$, where L is a learning rate.
- Repeat over all training examples.
- Continue until the network converges (i.e., the error between the training examples and their reconstructions falls below some threshold) or we reach some maximum number of epochs.