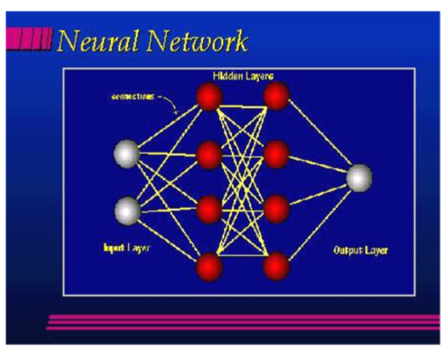
Data Mining
(EECS 6412)

## Neural Networks
(Introduction and Backpropagation)

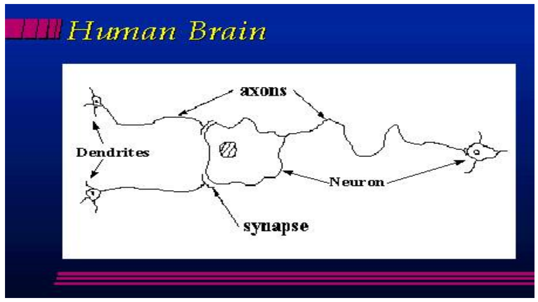
Aijun An
Department of Electrical Engineering and Computer Science
York University

## What is an Artificial Neural Network?

- It is a formalism for representing functions inspired from biological systems
- It consists of a few layers of interconnected computing units
  - Each unit computes a simple function.
  - The inputs of the units in one layer are the outputs of the units in the previous layer.
  - Each connection is associated with a weight.
  - Parallel computing can be performed among the units in each layer



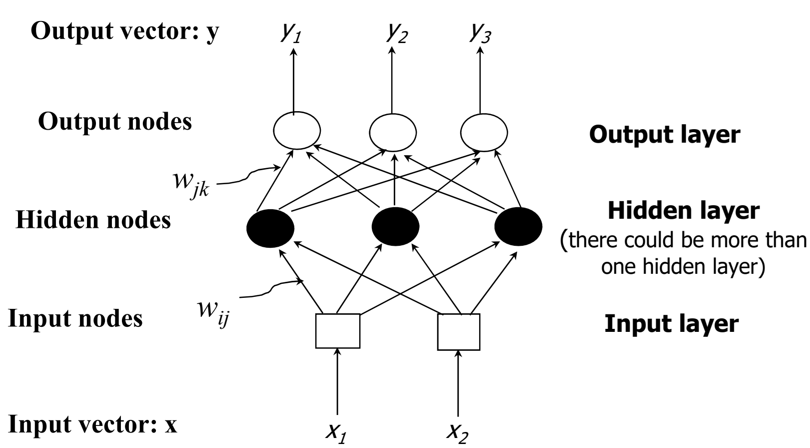
## Biological Motivation




- Biological Learning Systems are built of very complex webs of interconnected neurons.
  - Number of neurons: $10^{10}$
  - Connections per neuron: $10^4$ -- $10^5$
- Information-Processing abilities of biological neural systems must follow from highly *parallel processes* operating on representations that are distributed over many neurons
  - Scene recognition time: 0.1 second
- ANNs attempt to capture this mode of computation

3

## An Example of Neural Networks

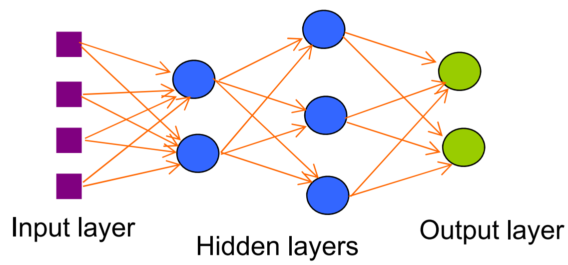



- Each hidden or output node or unit is called a neuron
- Each connection is associated with a weight
- The network is usually *fully connected*, unless otherwise specified.
  - Each unit provides an input to each unit in the next layer

4

## Feed-Forward Neural Network

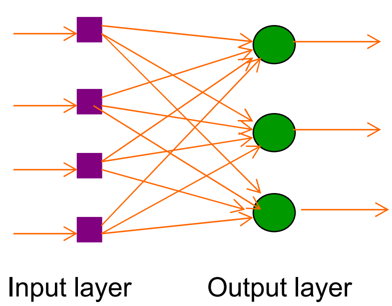
- The network is *feed-forward* if none of the connections cycles back to an input unit or a unit of a previous layer.
- An example:




5

## Single-layer Feed-Forward Neural Network

- A single-layer network does not have hidden layers




- The input layer is not counted as a "layer" because no computation is performed there.

6

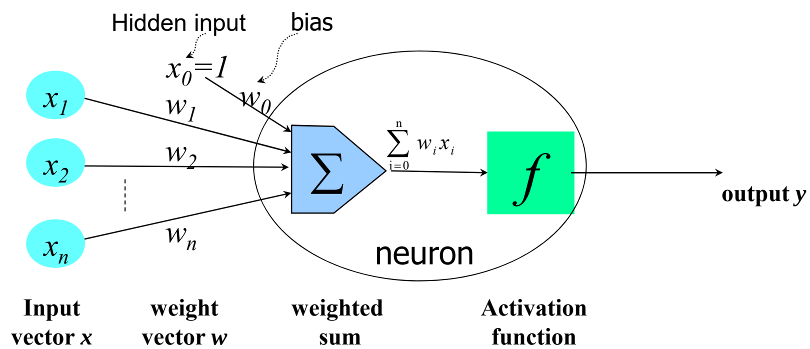
## Multi-layer Feed-Forward Neural Network

- A multilayer network contains one or more hidden layers
- The examples in slides 4 and 5 are *multi-layer feed-forward neural networks*.
- It is the popular neural network structure
- It overcomes the limitation of single-layer NN: single-layer NN cannot handle non-linearly separable learning tasks.

7

## What Happens inside a Neuron?

- A neuron is the basic information processing unit of a NN



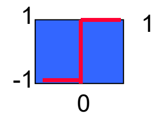

- A neuron first sums the weighted inputs and then apply an activation function
- Activation function maps a large input domain onto a smaller range (thus called *squashing function*)

8

## Examples of Activation Functions

- Threshold logic function, such as,

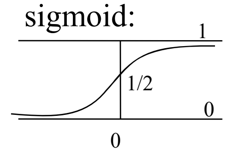
$$f(x) = sign(x) = \begin{cases} 1 & x > 0 \\ -1 & otherwise \end{cases}$$




- Sigmoid function:
  - commonly used

$$f(x) = \frac{1}{1+e^{-x}}$$

sigmoid:




  - differentiable:

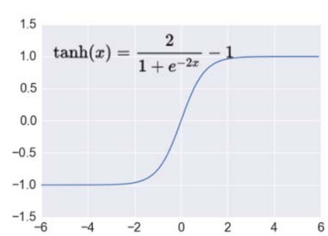
$$f'(x) = f(x)(1 - f(x))$$

a good feature for the purpose of training neural networks



## Examples of Activation Functions

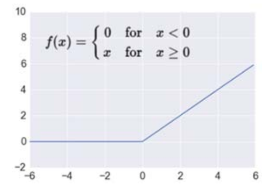
- Tanh function:




- Tanh is a scaled sigmoid: $\tanh(x)=2sigm(2x)-1$
- Range: (-1, 1)
- Differentiable.
- Advantage over sigmoid: larger derivative
  - For input range [-1,1], the derivative of Tanh ranges [0.42, 1], while the derivative of Sigmoid ranges [0.20, 0.25].

## Examples of Activation Functions

- Rectified Linear Units (ReLU),




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

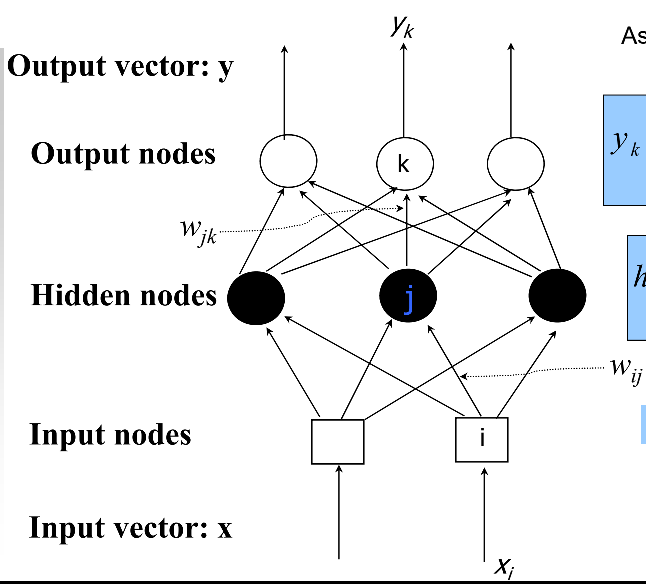
  - Commonly used in deep neural networks
  - Differentiable:

$$f'(x) = \begin{cases} 1, & \text{if x > 0} \\ 0, & \text{otherwise} \end{cases}$$

  - Major advantages:
    - Efficient computation
    - Sparse activation
    - No vanishing gradient problem in network training



## How Output is Computed?




Assume sigmoid is used

$$y_k = \frac{1}{1 + e^{-\sum_j w_{jk} h_j}}$$

$$h_j = \frac{1}{1 + e^{-\sum_i w_{ij} x_i}}$$

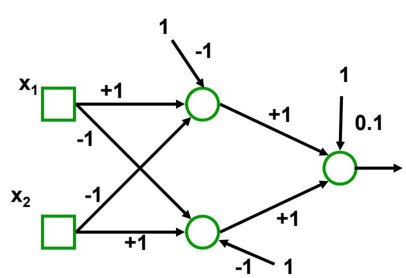
Pass the input value

## Function Modeled by a Neural Network

- A neural network can be used to model a function:
  - **y**=f(**x**) where **x** is the input vector and **y** is the output vector
- The function it models is determined by
  - Topology of the neural network:
    - the number of input units in the input layer
    - the number of hidden layers
    - the number of units in each hidden layer
    - the number of units in the output layer
  - Values of weights
  - Activation function chosen.

13

## What function does this NN model?




- $x_1$ and $x_2$ takes binary values of {-1, 1}
- Activation function is:

$$f(x) = sign(x) = \begin{cases} 1 & x > 0 \\ -1 & otherwise \end{cases}$$

14

## Neural Networks for Classification

- A neural net can be used for both regression and classification.
- To model a classification function
  - If there are two classes, we can use one output unit y.
    - If y>0.5, class 1
    - If y<0.5, class 0

    or

    - If y>0, class 1
    - If y<0, class 0
  - If there are more than two classes,
    - we can use one output unit per class.
    - Example is classified into the class corresponding to the output unit with the largest output value.
    - To output probability of each class, a *soft-max* layer is added:

$$z_i = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

15

## Inputs to Neural Networks

- Neural net can take continuous input variables naturally.
- A discrete attribute can be encoded by
  - using one input unit per value of the attribute
  - For example, if the domain of attribute A is {a1, a2, a3}
    - We assign 3 input units to represent A.
    - If A=a1, the input to the 3 units is 1, 0, 0.
    - If A =a2, the input to the 3 units is 0, 1, 0.
    - If A=a3, the input to the 3 units is 0, 0, 1.
- The input values to a neural net are usually normalized to speed up the learning phase.

16

## Training a Neural Network

- Before training can begin, the user must decide on the network topology
  - # of hidden layers and # of units in each hidden layer
  - Type of the activity function
  - no clear rules on the best topology
  - network design is a trial-and-error process
- The objective of training
  - search in the space of sets of weights to obtain a set of weights that makes almost all the tuples in the training data classified correctly.
- Also called *connectionist* learning
- Most popular neural network learning algorithm is *backpropagation*



## Training: Backpropagation algorithm

- Performs learning on a multi-layer feed-forward neural network, given a set of $N$ labeled training examples
- Objective: searches for weight values that minimize a loss function, e.g., the total error of the network over the set of training examples (training set):

$$E = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}(t_k^n - o_k^n)^2$$

$N$ is the number of training examples, $K$ is the number of output units, $t$ is the true output and $o$ is the computed output

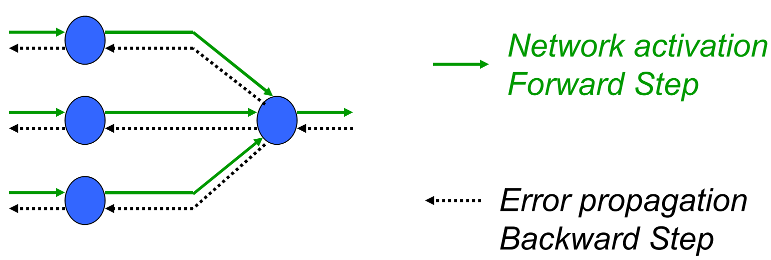
## Training: Backpropagation algorithm

- The algorithm first initializes the weights with small random values
- Then backpropagation consists of the repeated application of the following two passes:
  - **Forward pass**: in this step the network is activated on one example and the output of the network is computed.
  - **Backward pass**: in this step, starting at the output layer, the error of the output layer is computed and is propagated backwards through the network, layer by layer, and weights are updated.



## Backpropagation

- Back-propagation training algorithm




- Backprop adjusts the weights of the NN in order to minimize the network total mean squared error.

## Steps of Backpropagation Algorithm

- Initialize the weights with small random values (e.g., ranging from -1.0 to 1.0, or -0.5 to 0.5)
- For each training example, (*assuming sigmoid function is used*)
  - Propagate the inputs forward to obtain the output
    - the activation values of the hidden and then output units are computed.
  - Backpropagate the error to update weights
    - For each output unit $k$, calculate the error term:

$$\delta_k = o_k(1-o_k)(t_k - o_k)$$

where $o_k$ is the computed output and $t_k$ is the true output.

- For each hidden unit $h$, calculate the error term:

$$\delta_h = o_h(1-o_h) \sum_{k \in next\_higher\_layer} w_{hk}\delta_k$$

- Update each network weight $w_{ij}$

$$w_{ij} = w_{ij} + \Delta w_{ij}$$

where $\Delta w_{ij} = \eta \delta_j o_i$ and $\eta$ is learning rate (0 - 1)

21

## How $\Delta w_{ij}$ is derived

- Objective: minimize the mean square error:

$$E = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}(t_k^n - o_k^n)^2$$

$N$ is the number of training examples, $K$ is the number of output units, $t$ is the true output and $o$ is the computed output.

- $\Delta w_{ij}$ is calculated by

$$\Delta w_{ij} = -\eta \frac{\partial E_d}{\partial w_{ij}}$$

where $E_d$ is the error on training example $d$, summed over all output units in the network:

$$E_d = \frac{1}{2}\sum_{k \in outputs}(t_k - o_k)^2$$

- This weight-updating rule is called the *delta rule* or *gradient descent rule*

22

## Backpropagation (*Cont'd*)

- Weight updates can be either case-based or epoch-based
  - *Case updating*: weights are updated after presentation of each training example (as in our description)
  - *Epoch updating*: weight increments are accumulated and are updated after all the training examples have been presented
  - *Mini-batch updating*: present a randomly-selected subset of examples; weight increments are accumulated, and weights are updated after all the selected examples have been presented – ***Stochastic Gradient Descent*** (SGD)
- Terminating condition: Training stops when
  - all $\Delta w_{ij}$ are smaller than a specified threshold, or
  - the percentage of examples misclassified in the previous epoch is below some threshold, or
  - a pre-specified number of epochs has expired.



## Some Other Loss Functions

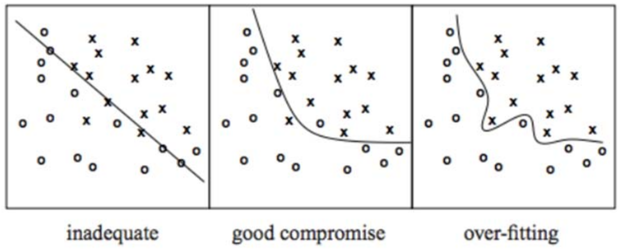
- Cross entropy:

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}\left[y_k^{(i)} \log \hat{y}_k^{(i)} + \left(1 - y_k^{(i)}\right) \log\left(1 - \hat{y}_k^{(i)}\right)\right]$$

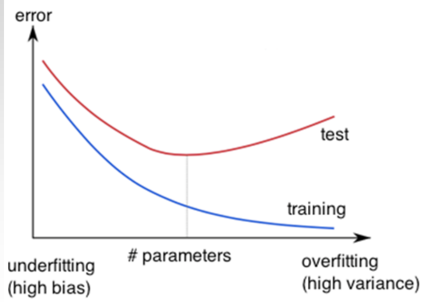
- Mean absolute error:

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left|y^{(i)} - \hat{y}^{(i)}\right|$$

# Overfitting




http://wiki.bethanycrane.com/overfitting-of-data




**Learned hypothesis may fit the training data very well, even outliers (noise) but fail to generalize to new examples (test data)**

https://www.neuraldesigner.com/images/learning/selection_error.svg

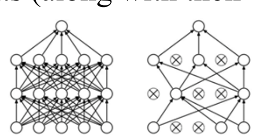
# Prevent Overfitting

- Regularization
  - Add regularization term to the objective function to penalize big weights

L2 norm of $\theta$

$$J_{reg}(\theta) = J(\theta) + \lambda \sum_k \theta_k^2$$

- Dropout
  - Randomly drop units (along with their connections) during training



- Early stopping
  - Use validation error to decide when to stop training
  - Stop when monitored quantity has not improved after n subsequent epochs

26

## Advantages of Neural Networks

- Solve problems that are too complicated for conventional techniques
  - can model both linear or non-linear functions
  - the output of target function may be discrete-valued, real-valued, or a vector of several real- or discrete-valued
- Able to learn from complicated or noisy data
- Problem solving (e.g., classification) with a neural network is fast

27

## Limitations

- Training is slow
  - could take hours or days
- In general, few rules for setting up the parameters regarding the topology of the network
- Difficult for human to understand the learned target function
  - Effective "Black Boxes" whose rules of operation are completely unknown
- Important to have many known examples for training (inputs and related outputs)

28

## Appropriate Problems for Neural Network Learning

- The training examples may contain errors.
- Long training times are acceptable.
- Fast evaluation of the learned target function may be required.
- The ability for humans to understand the learned target function is not important.
- Instances are represented by many attribute-value pairs (e.g., the pixels of a picture. ALVINN [Mitchell, p. 84]).
- The target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.

29

## Applications

- Speech Recognition
- Computer Vision
- Pattern Recognition
- Financial prediction
- Screening Mortgage & Credit Applications
- Oil & Gas Exploration
- Medical Diagnosis
- Targeted Marketing

30

## History of Neural Networks

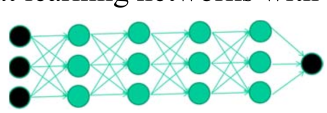
- 1943: McCulloch and Pitts proposed a model of a neuron --> Perceptron.
- 1960s: Widrow and Hoff explored Perceptron networks (which they called "Adelines") and the delta rule.
- 1962: Rosenblatt proved the convergence of the perceptron training rule.
- 1969: Minsky and Papert showed that the Perceptron cannot deal with nonlinearly-separable data sets---even those that represent simple function such as X-OR.
- 1970-1985: Very little research on Neural Nets
- 1986: Invention of Backpropagation [Rumelhart and McClelland, but also Parker and earlier on: Werbos] which can learn from nonlinearly-separable data sets.
- Since 1986: A lot of research in Neural Nets!

31

## Deep Neural Networks

- Deep neural network
  - Neural network with several layers of nodes between input and output
- The previous NN training algorithm is
  - good for training networks with 1 or 2 hidden layers

  - Not good at learning networks with more hidden layers.



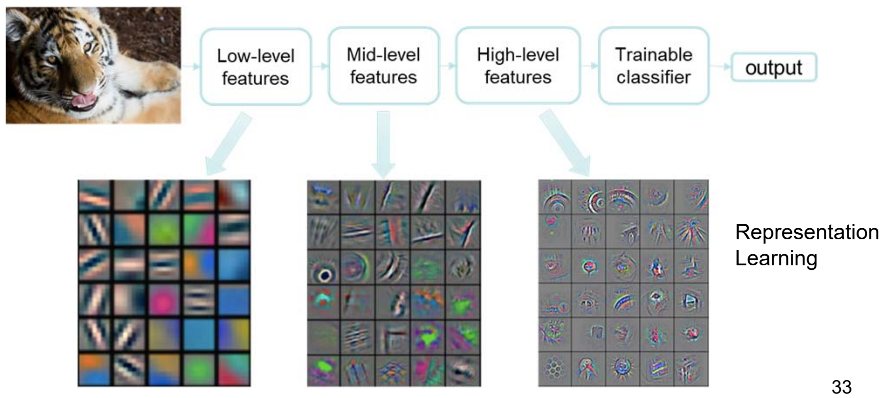
Perform worse than NN with 1 or 2 layers

- Reason: many basins in loss function exist, easily get stuck in poor local minima.

32

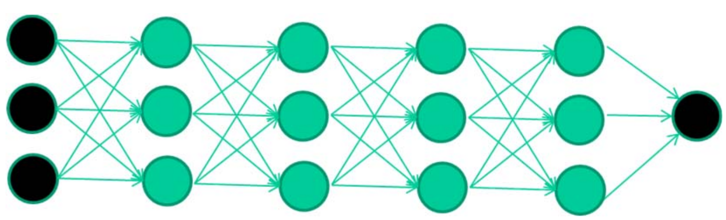
# Deep Learning

- Main idea
  - Use unsupervised learning to initialize weights
  - Seek to learn rich hierarchical representations (i.e. features) automatically through multiple stage of feature learning process.

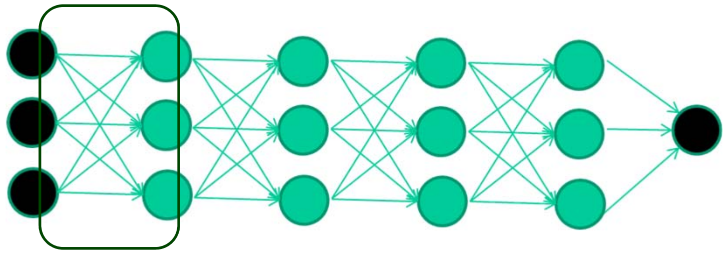



33

# The new way to train multi-layer NNs…

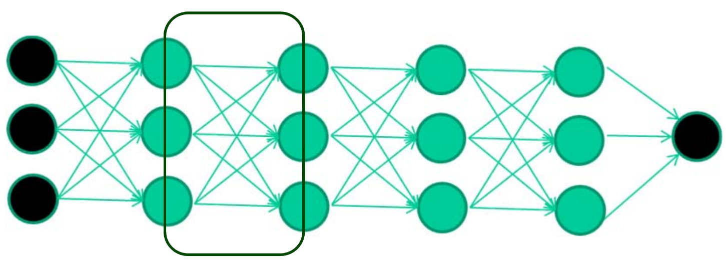


## The new way to train multi-layer NNs…



**Train this layer first**

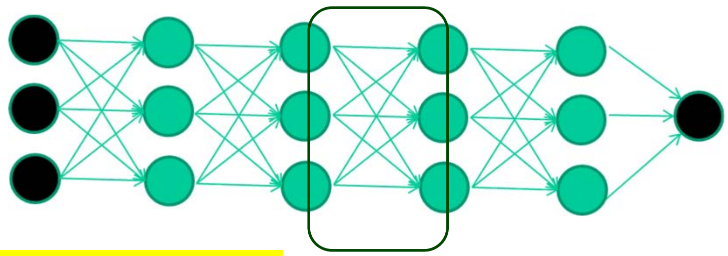
## The new way to train multi-layer NNs…



**Train this layer first**

**then this layer**

## The new way to train multi-layer NNs…

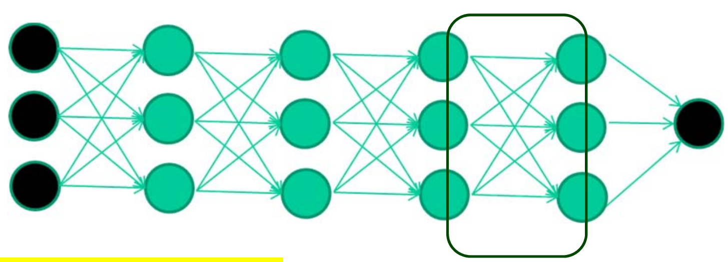


**Train this layer first**

**then this layer**

**then this layer**

## The new way to train multi-layer NNs…



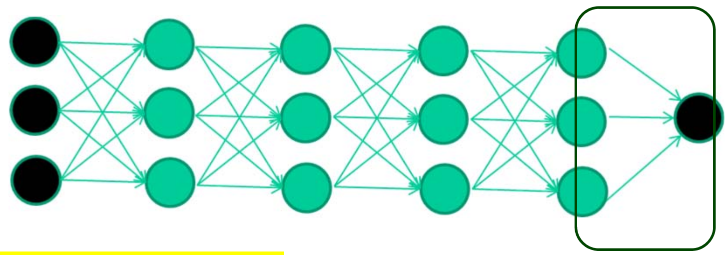
**Train this layer first**

**then this layer**

**then this layer**

**then this layer**

## The new way to train multi-layer NNs…



**Train this layer first**

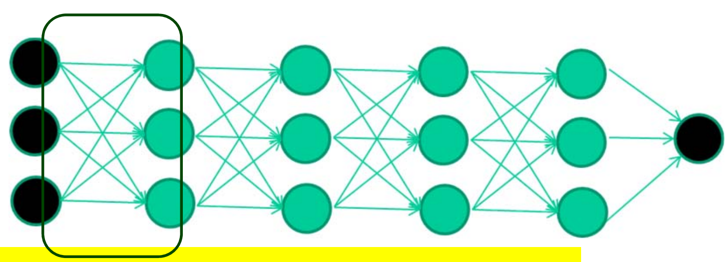
**then this layer**

**then this layer**

**then this layer**

**finally this layer**

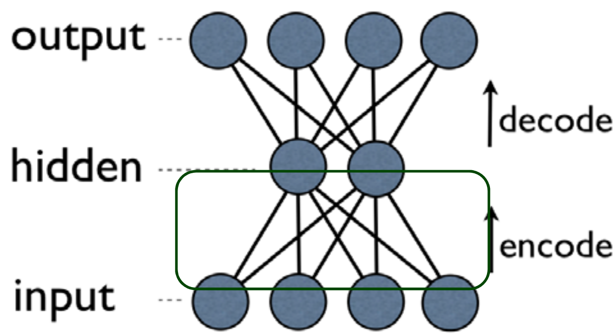
## The new way to train multi-layer NNs…



***EACH of the (non-output) layers is trained to be, e.g., an auto-encoder***

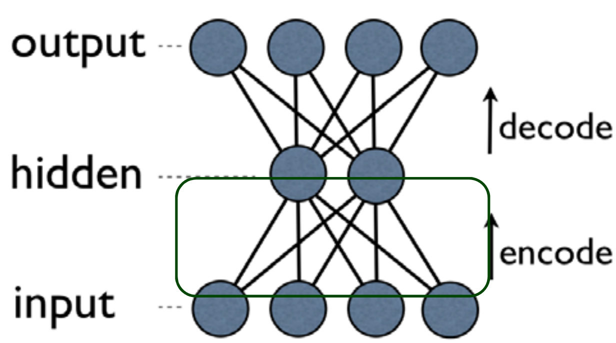
***Basically, it is forced to learn good features that describe what comes from the previous layer***

# Auto-encoder



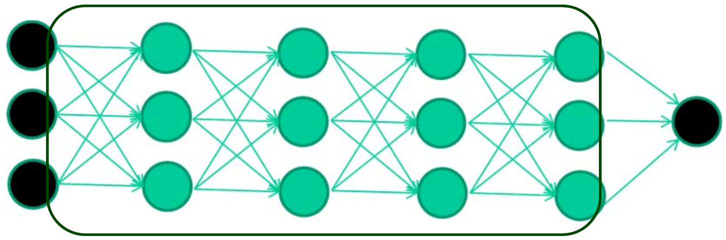

**An auto-encoder is trained, with an absolutely standard weight-adjustment algorithm to <u>reproduce the input</u>**

# Auto-encoder



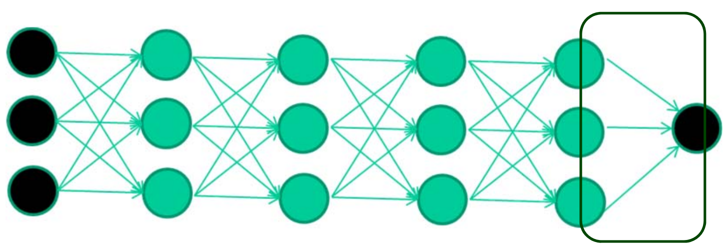

**By making this happen with (many) fewer units than the inputs, this forces the 'hidden layer' units to become good feature detectors**

## Deep Learning – Intermediate Layers



Intermediate layers are each trained to be auto encoders (or similar)

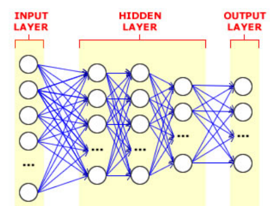
## Deep Learning – Final Layer



Final layer trained to predict class based on outputs from previous layers

## Deep Learning

- That's the basic idea
- There are many types of deep learning
- Different kinds of autoencoder, variations on architectures and training algorithms, etc…
- Very fast growing area …




## Deep Learning Architectures

- Deep belief networks (to learn representations layer by layer, like auto-encoder)
- Recurrent neural networks
- Convolutional deep neural networks
- Residual neural networks

## Next Class

- Clustering