

Introduction to Machine Learning and Data Mining Lecture-4: Reviews and Neural Networks

Prof. Eugene Chang

Today

- Review of previous topics
 - Bayes probability
 - Python language
 - Spyder & Ipython
 - Numpy & Scikit-learn
 - Regression and Decision Tree example
- Neural Networks

Revisit Bayes Probability

- Example: drawing a fruit from 2 color boxes
- Here the Box is the class (source), and the fruit is a feature, or observation.

	Orange	Apple	
$\frac{1}{3}$ Blue box	1	3	4
$\frac{2}{3}$ Red box	6	2	8
	7	5	12

Interpretation of Bayes Rule

The diagram illustrates the components of Bayes' Rule. It features three blue rectangular boxes: 'Posterior' on the left, 'Likelihood' in the center, and 'Prior' on the right. The equation $p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)}$ is displayed below the 'Likelihood' box. A blue hand-drawn line connects the 'Posterior' box to the left side of the equation, the 'Likelihood' box to the numerator $p(X|Y)p(Y)$, and the 'Prior' box to the denominator $p(X)$. The entire equation is also enclosed in a blue hand-drawn oval.

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)}$$

- **Prior:** Information we have before observation.
- **Posterior:** The distribution of Y after observing X
- **Likelihood:** The likelihood of observing X given Y
- $P(X) = \sum_i P(X|Y_i)P(Y_i)$
- Example
 - $P(Y_0)$ red box Prior probability = $2/3$, $P(Y_1)$ blue box Prior probability = $1/3$
 - $P(X_0|Y_0)$ orange given red box = $1/4$, $P(X_1|Y_0)$ apple given red box = $3/4$
 - $P(X_0|Y_1)$ orange given blue box = $3/4$, $P(X_1|Y_1)$ apple given blue box = $1/4$

The Correct Posterior Probability Calculation

$$P(X_0) = \sum_i P(X_0|Y_i)P(Y_i) = \frac{1}{4} * \frac{2}{3} + \frac{3}{4} * \frac{1}{3} = \frac{5}{12}$$

- After drawing an orange (X_0), the probability that it comes from red box (Y_0)

$$P(Y_0|X_0) = \frac{P(X_0|Y_0)P(Y_0)}{P(X_0)} = \frac{\frac{1}{4} * \frac{2}{3}}{\frac{5}{12}} = \frac{2}{5}.$$

- After drawing an orange (X_0), the probability that it comes from blue box (Y_1)

$$P(Y_1|X_0) = \frac{P(X_0|Y_1)P(Y_1)}{P(X_0)} = \frac{\frac{3}{4} * \frac{1}{3}}{\frac{5}{12}} = \frac{3}{5}.$$

Python in one slide

- Interpreted language
- Comments: # and '''
- Variables are assigned, not declared
- Use indentation to determine code block boundary (scope)
- Data types: numbers, strings, lists, tuples, dictionary, array
- Control statements: if elif else, for, while
- Expressions
- Functions: built-in
 - Functions: `def function_name(arguments):`
 - Anonymous function: `lamda arguments: simple_expression`
- Modules: import

Python Development Environments

- Anaconda
 - <http://continuum.io/downloads>
 - Package python core + libraries + tools in one download
 - Update and install: “conda update *module*” and “conda install *module*”
- Spyder
 - Integrated Development Environment
 - Code editor, object information, lpython consoles
- Ipython
 - Qt based graphical shell with improved commands
 - Browser based **notebook** interface
- Libraries
 - numpy, scikit-learn, scipy, matplotlib, pandas

Numpy

- N-d array: `a = array([[0, 1, 2, 3], [10,11,12,13]])`
- Indexing
 - Start at 0, -1: last element, -2: last element - 1
- Shape functions
 - `a.shape` -> (row, column) = (2, 4)
 - Resize, swapaxes, flatten
- Slicing
 - `a[2::2,::2]` -> `array([[20, 22, 24], [40, 42, 44]])`
 - `a[:,2]` -> `array([2,12,22,32,42,52])`
 - `a[0,3:5]` -> `array([3, 4])`
 - `a[4:,4:]` -> `array([[44, 45], [54, 55]])`
- Array methods
 - Calculation
 - Statistics

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

Scikit-learn

- Cover many standard machine learning techniques
- Training
 - `estimator.fit(X_train, y_train)`
- Classification, regression, clustering
 - `y_pred = estimator.predict(X_test)`
- Predictive models, density estimation
 - `test_score = estimator.score(X_test)`
- Filters, dimension reduction, latent variables
 - `X_new = estimator.transform(X_test)`
- Code sample

```
>>> from sklearn import linear_model
>>> regr= linear_model.LinearRegression()
>>> regr.fit(X_train, y_train)
>>> y_pred = regr.predict(X_test)
```

Example: hw1sample.py

- Create np array
- Create training data and targets
- Apply linear regression
- Print out coefficients
- Try to add noise

Example: regr_dtree.py

ID	Gender	Age	Income	Degree	Credit Rate	Use X	Use Y	Use Z
1	0	35	120000	2	600	0	0	1
2	1	28	115000	3	800	1	1	1
3	0	45	138000	2	610	0	0	0
4	1	65	69000	2	600	1	0	1
5	1	67	35000	1	600	0	0	0
6	1	68	42000	1	800	1	0	0
7	1	36	28000	1	800	1	1	1
8	0	25	67000	2	600	0	1	1
9	0	70	31000	2	630	1	0	1
10	1	23	61000	1	600	0	1	0
11	1	44	62000	1	800	1	1	1
12	1	50	65000	1	800	0	0	1
13	0	51	125000	3	600	1	1	0
14	0	69	80000	1	800	0	0	0
15	0	38	59000	2	800	0	1	1
16	1	46	33000	1	600	1	0	1

Gender	Male	Female		
	0	1		
Age	Child	Youth	Middle	Senior
>=	0	20	40	65
Income	Poor	Low	Medium	High
>=	0	20000	50000	100000
Credit	Poor	Fair	Good	Excellecnt
>=	0	550	650	750
Degree	No	High school	College	Post grad
	0	1	2	3

Example: regr_dtree.py

- Read in inputs
- Pre-process data
- Create training and test sets
- Linear regression
 - Fit
 - Predict
 - Evaluation
- Decision tree
 - Fit
 - Predict
 - Evaluation

Neural Network

Review: Fitting Polynomial Functions

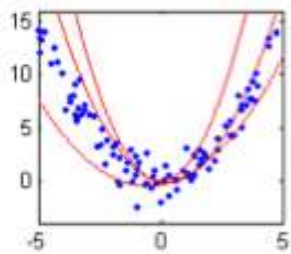
- Fitting nonlinear 1-D functions

- Polynomial:

$$f(x, \vec{w}) = \sum_{d=1}^D w_d x^d + w_0 \quad f(x, \theta) = \sum_{d=1}^D \theta_d x^d + \theta_0$$

vector
↓
singular
↓

- Risk:



$$R_{emp}(\vec{w}) = \frac{1}{2N} \sum_{i=0}^{N-1} \left(t_i - \begin{bmatrix} 1 & x_i \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \right)^2$$

$$R(\theta) = \frac{1}{2N} \left\| \begin{bmatrix} t_0 \\ t_1 \\ \vdots \\ t_{N-1} \end{bmatrix} - \begin{bmatrix} 1 & x_0 \\ 1 & x_1 \\ \vdots & \vdots \\ 1 & x_{N-1} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \right\|^2$$

$$= \frac{1}{2N} \left\| \vec{t} - \vec{X} \vec{w} \right\|^2$$

Review: Fitting Polynomial Functions

- Order-D polynomial regression for 1D variables is the same as D-dimensional linear regression
- Extend the feature vector from a scalar.

- More generally

$$\vec{x}_i = [x_i^0 \quad x_i^1 \quad x_i^2 \quad \dots \quad x_i^D]^T$$

Handwritten note: A blue circle around the T superscript with an arrow pointing to it and the text "vector denote" written below.

$$\vec{x}_i = [\phi_0(x_i) \quad \phi_1(x_i) \quad \phi_2(x_i) \quad \dots \quad \phi_D(x_i)]^T$$

Introduction

- What are Neural Networks?
 - Neural networks are a new method of programming computers.
 - They are exceptionally good at performing pattern recognition and other tasks that are very difficult to program using conventional techniques.
 - Programs that employ neural nets are also capable of learning on their own and adapting to changing conditions.

Background

- An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the biological nervous systems, such as the human brain's information processing mechanism.
- The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. NNs, like people, learn by example.
- An NN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of NNs as well.

Neural Network in Use

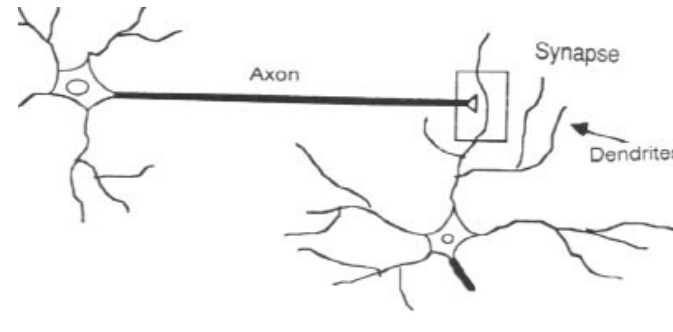
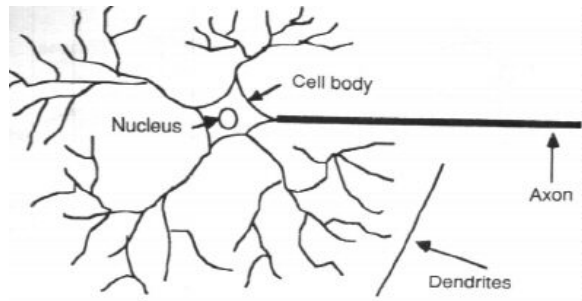
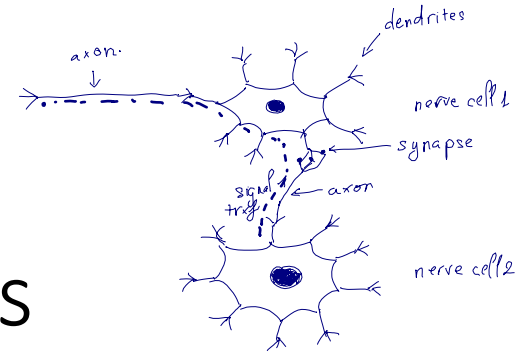
Since neural networks are best at identifying patterns or trends in data, they are well suited for prediction or forecasting needs including:

- sales forecasting
- industrial process control
- customer research
- data validation
- risk management

ANN are also used in the following specific paradigms

recognition of speakers in communications; diagnosis of hepatitis; undersea mine detection; texture analysis; three-dimensional object recognition; hand-written word recognition; and facial recognition.

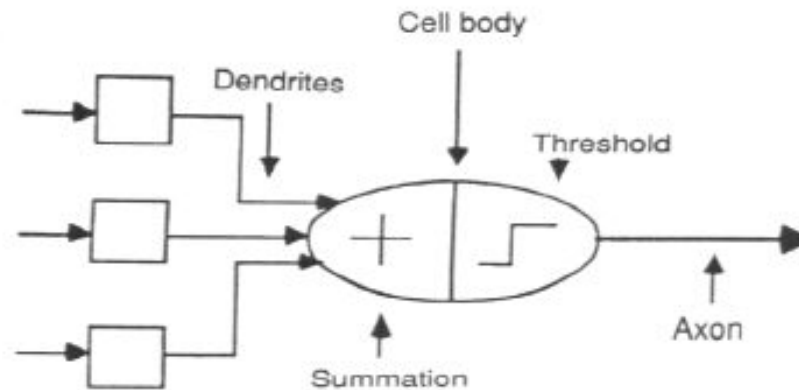
How the Human Brain learns



- In the human brain, a typical neuron collects signals from others through a host of fine structures called *dendrites*.
- The neuron sends out spikes of electrical activity through a long, thin stand known as an *axon*, which splits into thousands of branches.
- At the end of each branch, a structure called a *synapse* converts the activity from the axon into electrical effects that inhibit or excite activity in the connected neurons.

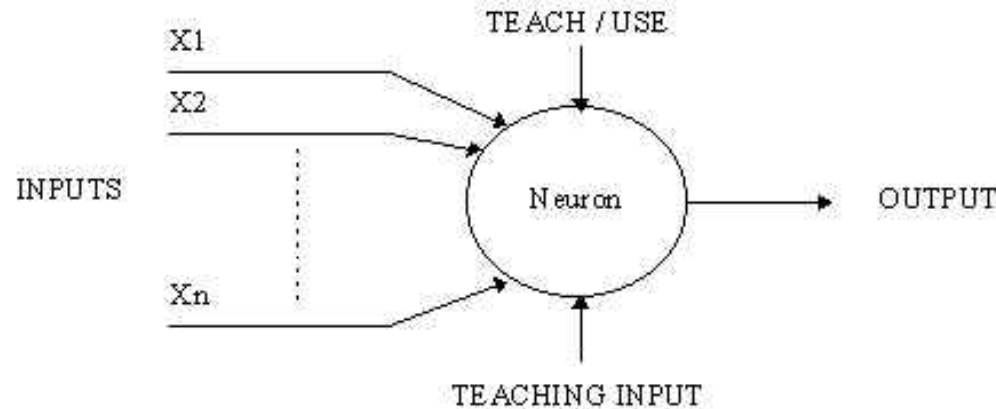
A Neuron Model

- When a neuron receives excitatory input that is sufficiently large compared with its inhibitory input, it sends a spike of electrical activity down its axon. Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes



- We conduct these neural networks by first trying to deduce the essential features of neurons and their interconnections.
- We then typically program a computer to simulate these features.

A Simple Neuron



- An artificial neuron is a device with many inputs and one output.
- The neuron has two modes of operation;
 - the training mode and
 - the using mode.

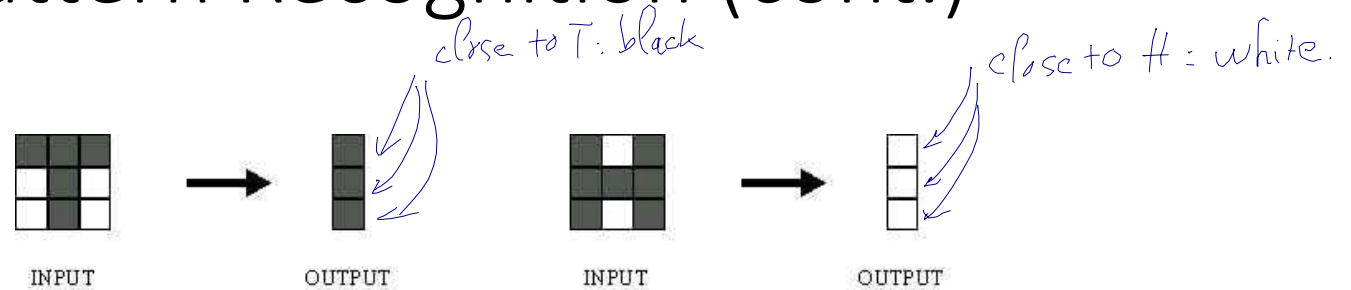
A Simple Neuron (Cont.)

- In the training mode, the neuron can be trained to fire (or not), for particular input patterns.
- In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output. If the input pattern does not belong in the taught list of input patterns, the firing rule is used to determine whether to fire or not.
- The firing rule is an important concept in neural networks and accounts for their high flexibility. A firing rule determines how one calculates whether a neuron should fire for any input pattern. It relates to all the input patterns, not only the ones on which the node was trained on previously.

Pattern Recognition

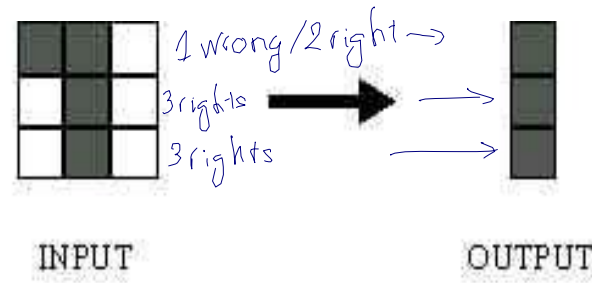
- An important application of neural networks is pattern recognition. Pattern recognition can be implemented by using a feed-forward neural network that has been trained accordingly.
- During training, the network is trained to associate outputs with input patterns.
- When the network is used, it identifies the input pattern and tries to output the associated output pattern.
- The power of neural networks comes to life when a pattern that has no output associated with it, is given as an input.
- In this case, the network gives the output that corresponds to a taught input pattern that is least different from the given pattern.

Pattern Recognition (cont.)



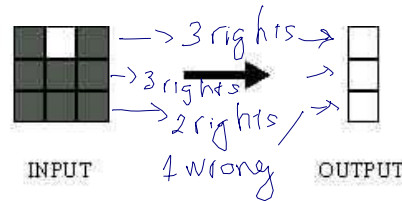
- Suppose a network is trained to recognize the patterns T and H. The associated patterns are all black and all white respectively as shown above.

Pattern Recognition (cont.)



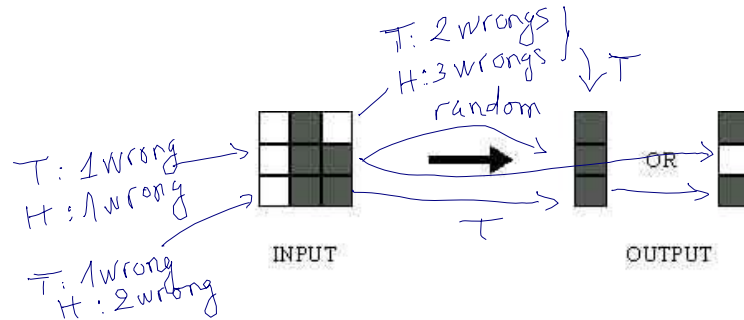
Since the input pattern looks more like a 'T', when the network classifies it, it sees the input closely resembling 'T' and outputs the pattern that represents a 'T'.

Pattern Recognition (cont.)



The input pattern here closely resembles 'H' with a slight difference. The network in this case classifies it as an 'H' and outputs the pattern representing an 'H'.

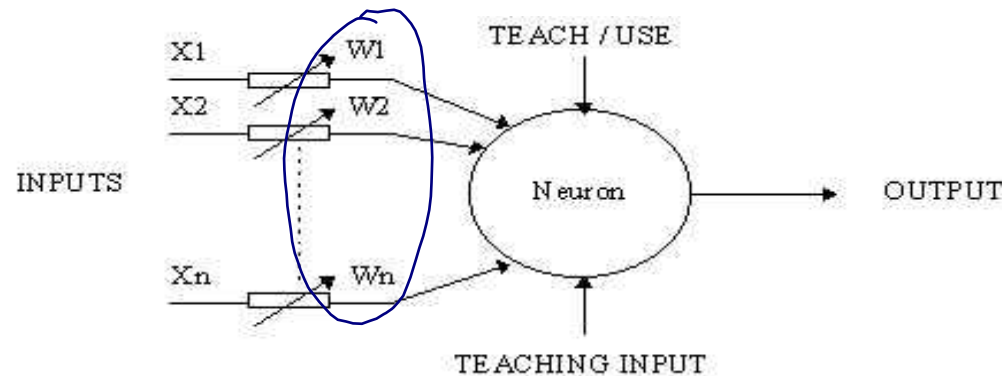
Pattern Recognition (cont.)



- Here the top row is 2 errors away from a 'T' and 3 errors away from an H. So the top output is a black.
- The middle row is 1 error away from both T and H, so the output is random.
- The bottom row is 1 error away from T and 2 away from H. Therefore the output is black.
- Since the input resembles a 'T' more than an 'H' the output of the network is in favor of a 'T'.

A Complicated Perceptron

the weight influences the output



- A more sophisticated Neuron is known as the McCulloch and Pitts model (MCP).
- The difference is that in the MCP model, the inputs are weighted and the effect that each input has at decision making, is dependent on the weight of the particular input.
- The weight of the input is a number which is multiplied with the input to give the weighted input.

A Complicated Perceptron

- The weighted inputs are then added together and if they exceed a pre-set threshold value, the perceptron / neuron fires.
- Otherwise it will not fire and the inputs tied to that perceptron will not have any effect on the decision making.
- In mathematical terms, the neuron fires if and only if;
$$X_1W_1 + X_2W_2 + X_3W_3 + \dots > T$$
 threshold
- The MCP neuron has the ability to adapt to a particular situation by changing its weights and/or threshold.
- Various algorithms exist that cause the neuron to 'adapt'; the most used ones are the Delta rule and the back error propagation.

Different types of Neural Networks

- **Feed-forward networks**

- Feed-forward NNs allow signals to travel one way only; from input to output. There is no feedback (loops) i.e. the output of any layer does not affect that same layer.
- Feed-forward NNs tend to be straight forward networks that associate inputs with outputs. They are extensively used in pattern recognition.
- This type of organization is also referred to as bottom-up or top-down.

- **Feedback networks**

- Feedback networks can have signals traveling in both directions by introducing loops in the network.
- Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point.
- Feedback architectures are also referred to as interactive or recurrent, although the latter term is often used to denote feedback connections in single-layer organizations.

Diagram of an NN

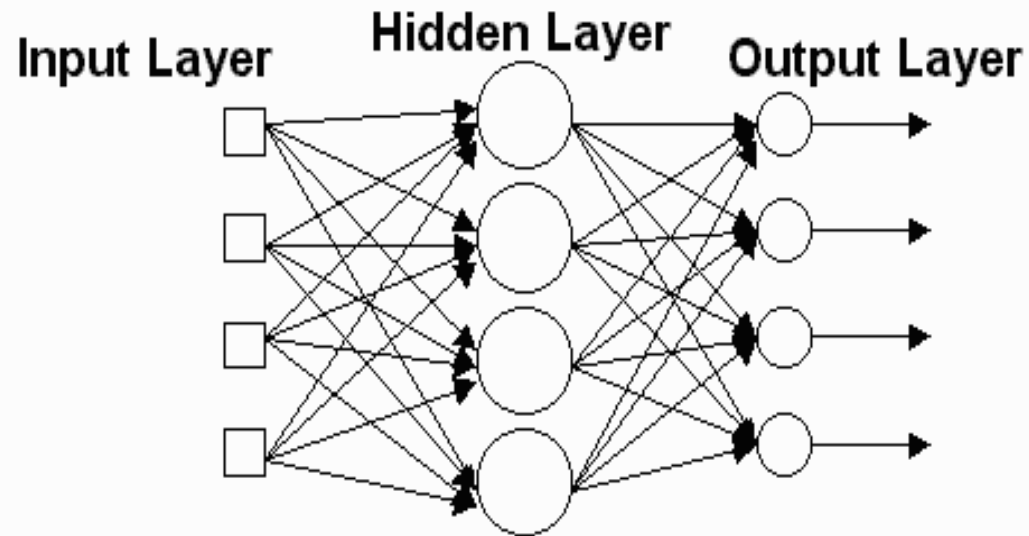


Figure 2 The anatomy of a neural network.

Network Layers

- Input Layer - The activity of the input units represents the raw information that is fed into the network.
- Hidden Layer - The activity of each hidden unit is determined by the activities of the input units and the weights on the connections between the input and the hidden units.
- Output Layer - The behavior of the output units depends on the activity of the hidden units and the weights between the hidden and output units.

Network Parameters

- How are the weights initialized?
- How many hidden layers and how many neurons?
- How many examples in the training set?

Weights

- In general, initial weights are randomly chosen, with typical values between -1.0 and 1.0 or -0.5 and 0.5.
- There are two types of NNs. The first type is known as
 - Fixed Networks – where the weights are fixed
 - Adaptive Networks – where the weights are changed to reduce prediction error.

Size of Training Data

- Rule of thumb:
 - the number of training examples should be at least five to ten times the number of weights of the network.
- Other rule:

$$N > \frac{|W|}{(1 - a)}$$

$|W|$ = number of weights

a = expected accuracy on test set

Perceptron Training

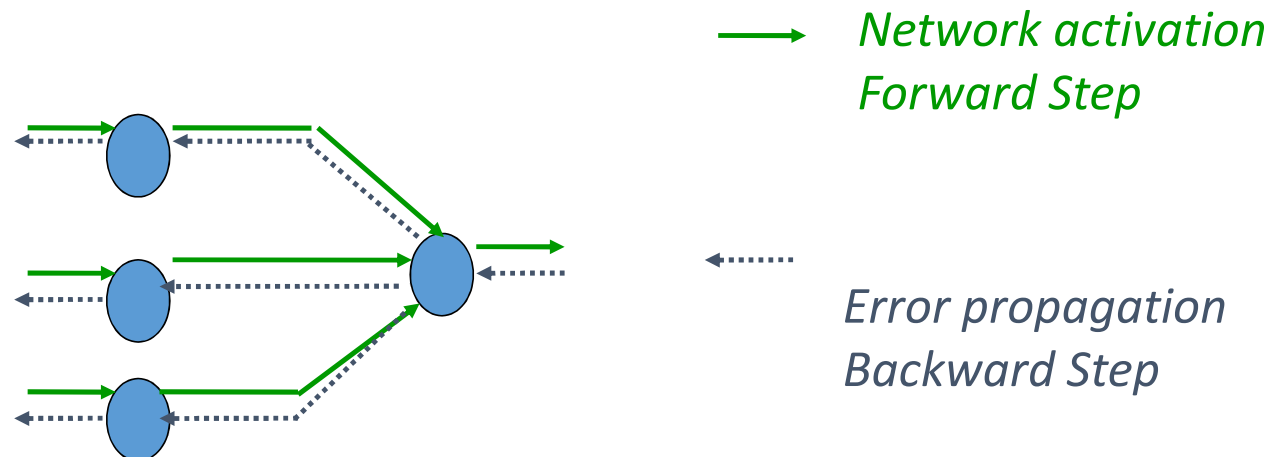
- The most basic method of training a neural network is trial and error. (Supervised Learning)
- If the network isn't behaving the way it should, change the weighting of a random link by a random amount. If the accuracy of the network declines, undo the change and make a different one.
- Learn synaptic weights so that unit produces the correct output for each example.
- Perceptron uses iterative update algorithm to learn a correct set of weights.

Training: Backprop algorithm

- The Backprop algorithm searches for weight values that minimize the total error of the network over the set of training examples (training set).
- Backprop consists of the repeated application of the following two passes:
 - **Forward pass**: in this step the network is activated on one example and the error of (each neuron of) the output layer is computed.
 - **Backward pass**: in this step the network error is used for updating the weights. Starting at the output layer, the error is propagated backwards through the network, layer by layer. This is done by recursively computing the local gradient of each neuron.

Back Propagation

- Back-propagation training algorithm



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

Perceptron Learning Rule

- Update weights by:

$$w_{ji} = w_{ji} + \eta(t_j - o_j)o_i$$

where η is the “learning rate”

t_j is the teacher specified output for unit j .

- Equivalent to rules:
 - If output is correct do nothing.
 - If output is high, lower weights on active inputs
 - If output is low, increase weights on active inputs
- Also adjust threshold to compensate:

$$T_j = T_j - \eta(t_j - o_j)$$

Perceptron Learning Algorithm

- Iteratively update weights until convergence.

Initialize weights to random values

Until outputs of all training examples are correct

For each training pair, E , do:

 Compute current output o_j for E given its inputs

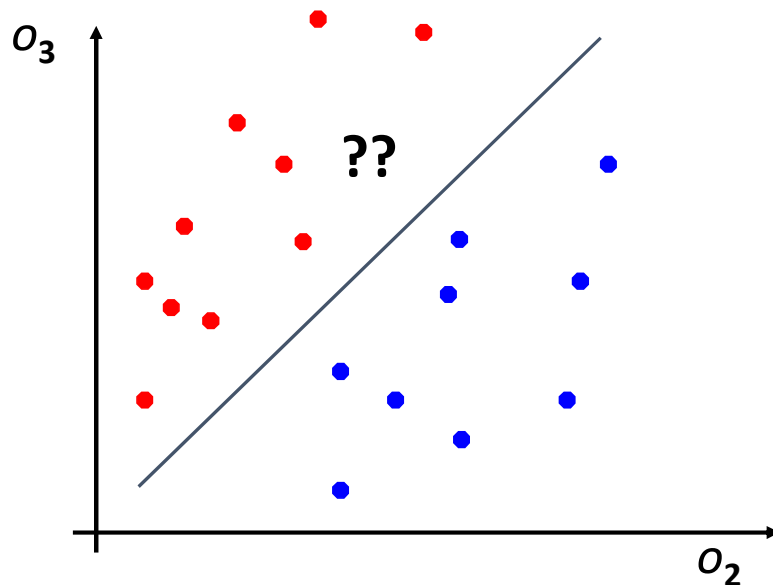
 Compare current output to target value, t_j , for E

 Update synaptic weights and threshold using learning rule

- Each execution of the outer loop is typically called an *epoch*.

Perceptron as a Linear Separator

- Since perceptron uses linear threshold function, it is searching for a linear separator that discriminates the classes.



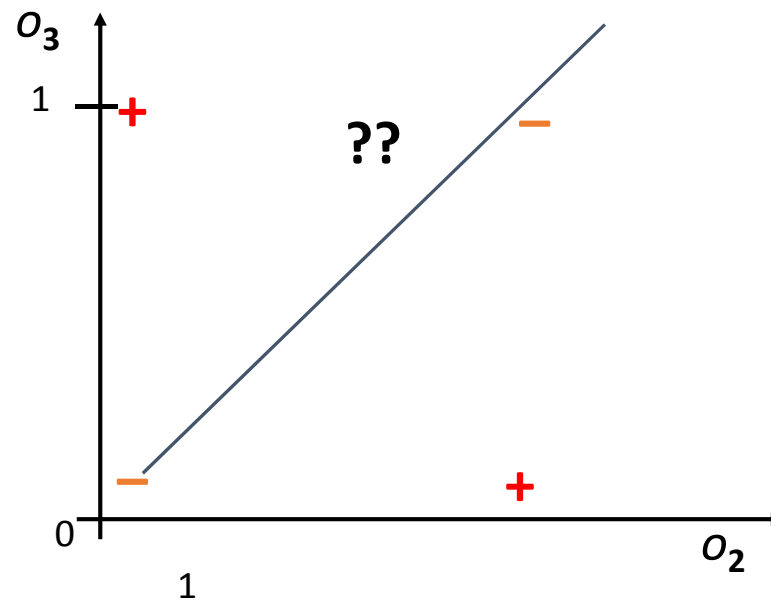
$$w_{12}o_2 + w_{13}o_3 > T_1$$

$$o_3 > -\frac{w_{12}}{w_{13}}o_2 + \frac{T_1}{w_{13}}$$

**Or hyperplane in
n-dimensional space**

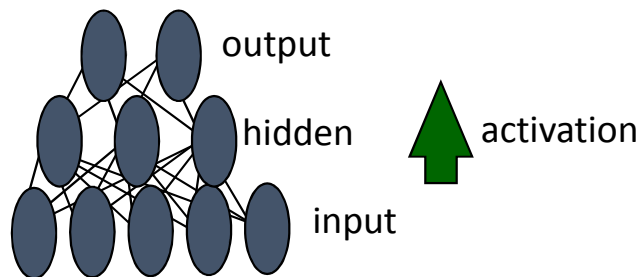
Concept Perceptron Cannot Learn

- Cannot learn exclusive-or, or parity function in general.



Multi-Layer Networks

- Multi-layer networks can represent arbitrary functions, but an effective learning algorithm for such networks was thought to be difficult.
- A typical multi-layer network consists of an input, hidden and output layer, each fully connected to the next, with activation feeding forward.



- The weights determine the function computed. Given an arbitrary number of hidden units, any boolean function can be computed with a single hidden layer.

Backpropagation Learning Rule

- Each weight changed by:

$$\Delta w_{ji} = \overset{\text{learning rate}}{\eta} \overset{\text{error measure}}{\delta_j} \overset{\text{source output}}{o_i}$$

$$\delta_j = o_j(1 - o_j) \underbrace{(t_j - o_j)}_{\substack{\text{differences of teaching output} \\ \text{vs experimental output}}} \quad \text{if } j \text{ is an output unit}$$

$$\delta_j = o_j(1 - o_j) \sum_k \delta_k w_{kj} \quad \text{if } j \text{ is a hidden unit}$$

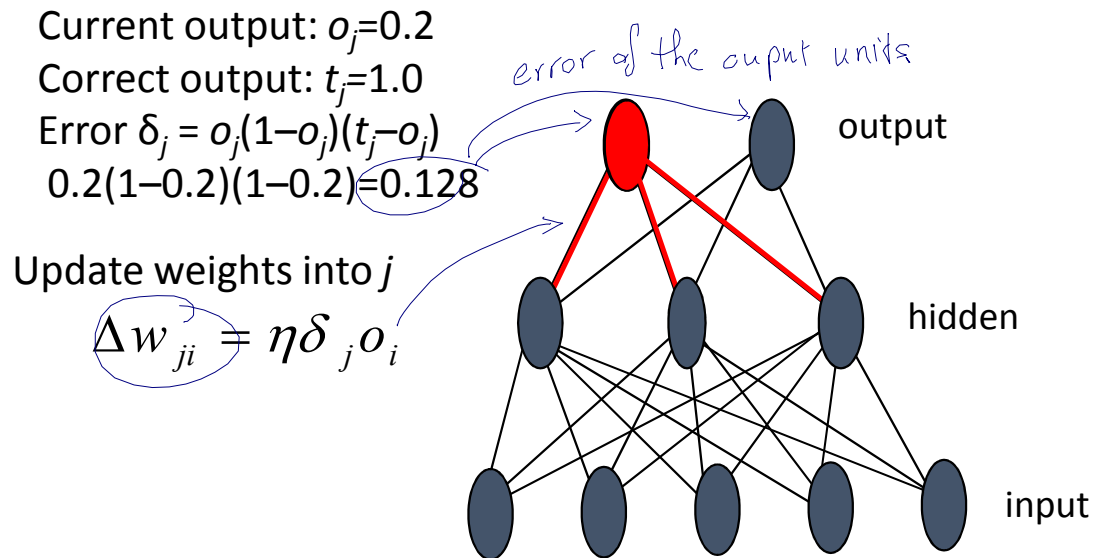
where η is a constant called the learning rate

t_j is the correct teacher output for unit j

δ_j is the error measure for unit j

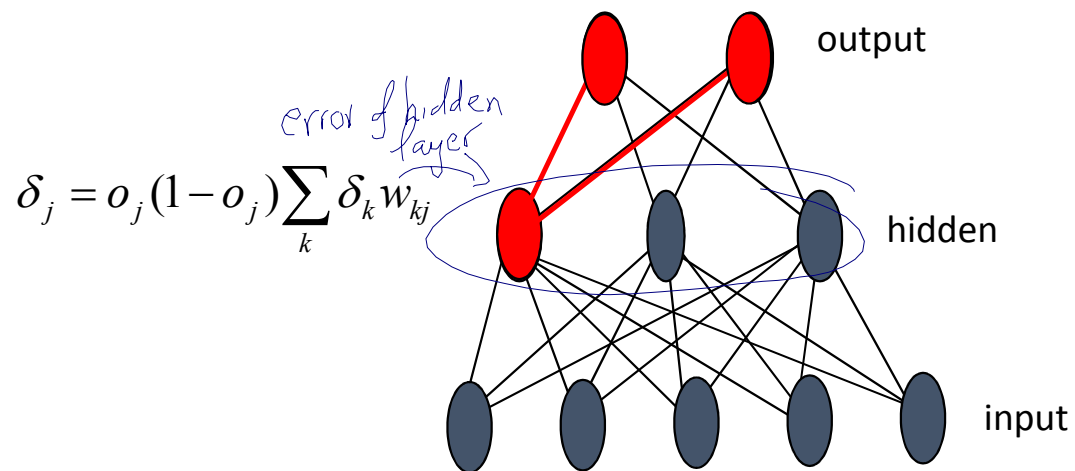
Error Backpropagation

- First calculate error of output units and use this to change the top layer of weights.



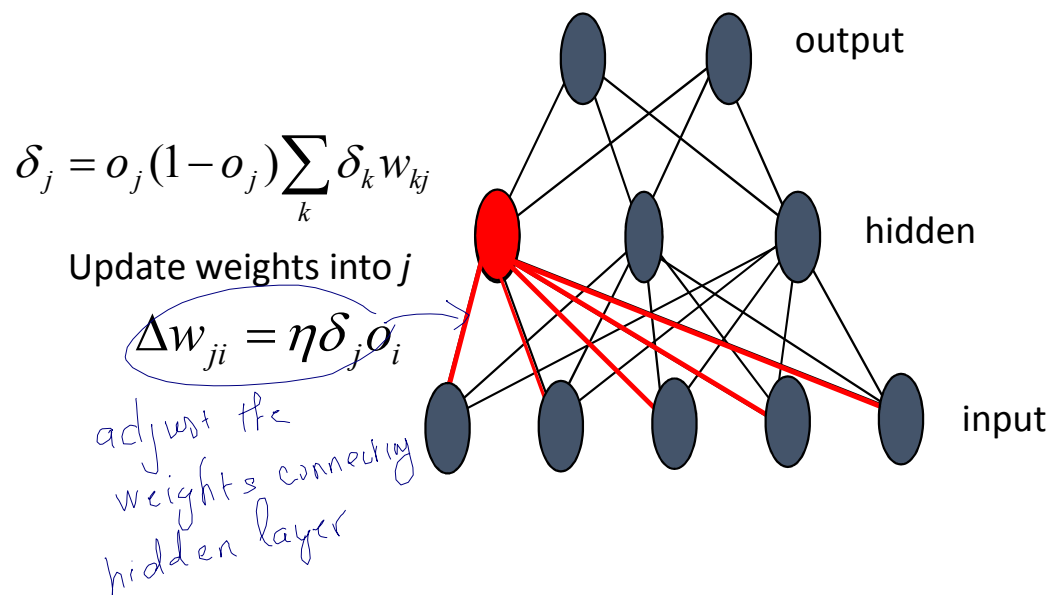
Error Backpropagation

- Next calculate error for hidden units based on errors on the output units it feeds into.



Error Backpropagation

- Finally update bottom layer of weights based on errors calculated for hidden units.



Backpropagation Training Algorithm

Create the 3-layer network with H hidden units with full connectivity between layers. Set weights to small random real values.

Until all training examples produce the correct value (within ϵ), or mean squared error ceases to decrease, or other termination criteria:

Begin epoch

For each training example, d , do:

Calculate network output for d 's input values

Compute error between current output and correct output for d


Update weights by backpropagating error and using learning rule

End epoch

Comments on Training Algorithm

- Not guaranteed to converge to zero training error, may converge to local optima or oscillate indefinitely.
- However, in practice, does converge to low error for many large networks on real data.
- Many epochs (thousands) may be required, hours or days of training for large networks.
- To avoid local-minima problems, run several trials starting with different random weights (*random restarts*).
 - Take results of trial with lowest training set error.
 - Build a committee of results from multiple trials (possibly weighting votes by training set accuracy).

Representational Power

- **Boolean functions:** Any boolean function can be represented by a two-layer network with sufficient hidden units. 
- **Continuous functions:** Any bounded continuous function can be approximated with arbitrarily small error by a two-layer network.
 - Sigmoid functions can act as a set of basis functions for composing more complex functions, like sine waves in Fourier analysis.
- **Arbitrary function:** Any function can be approximated to arbitrary accuracy by a three-layer network.

Evaluation of Classification Results

- Confusion matrix: evaluates classification accuracy
 - Entry i, j in a confusion matrix is the number of observations actually in group i , but predicted to be in group j

- Sample code

```
>>> from sklearn.metrics import confusion_matrix
>>> y_true = [2, 0, 2, 2, 0, 1]
>>> y_pred = [0, 0, 2, 2, 0, 2]
>>> confusion_matrix(y_true, y_pred)
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```

Examples

- `mlp.py`
- `iris.py`
- `logic.py`
- `PNOz.py`
- `mnist.py`