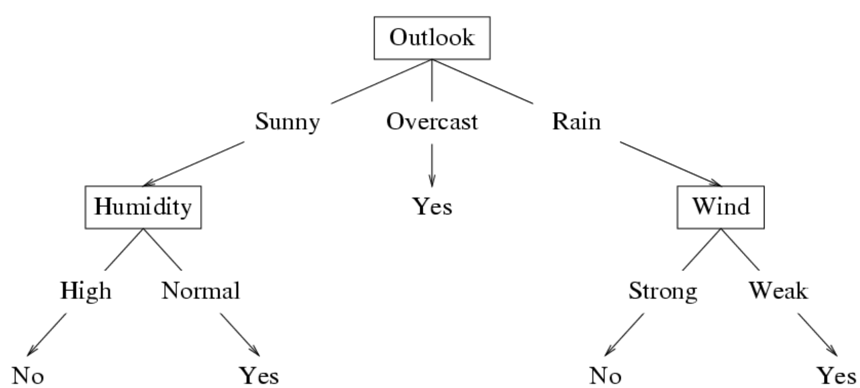
NEURAL NETWORKS

Recap: Decision Trees



Learning a Decision Tree

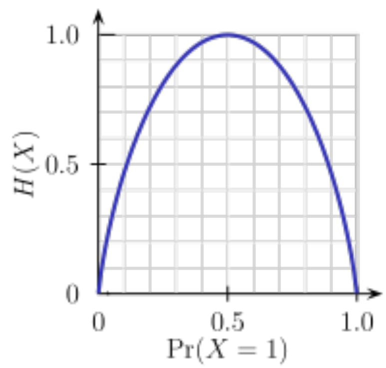
```
function DECISION-TREE-LEARNING (examples, attributes, parents) returns a tree
    if examples is empty return MAJORITY_VOTE(parents)
    else if all examples have same classification return classification
    else if attributes is empty return MAJORITY_VOTE(examples)
    else
        A    ←——  CHOOSE-BEST-ATTRIBUTE (examples)
        tree ←——  a new decision tree with root A
        for each value v_k of A
            S_k     ←—— examples with value v_k for attribute A
            subtree ←—— DECISION-TREE-LEARNING(S_k, attributes-A, examples)
            add branch to tree with label (A=v_k) and subtree
        return tree
```

Choosing the best attribute

- Splitting on a good attribute
  - After the split, the examples at each branch have the same classification
- Splitting on a bad attribute
  - After the split, the examples at each branch have the same proportion of positive and negative examples
- We will use entropy and information gain to formalize what we mean by *good* and *bad* attributes

Entropy

- Entropy measures the uncertainty of a random variable
  - How many bits are needed to efficiently encode the possible values (outcomes) of a random variable?
- Introduced by Shannon in 1948 paper
- Example: flipping a coin
  - A completely biased coin requires 0 bits of entropy
  - A fair coin requires 1 bit of entropy
  - How many bits are need to encode the outcome of flipping a fair coin twice?



Entropy and Information Gain

- Let A be a random variable with values v_k
- Each value v_k occurs with probability $p(v_k)$
- Then the entropy of A is defined as

$$H(A) = \sum_k p(v_k)\, log_2\left(\frac{1}{p(v_k)}\right)$$
$$= -\sum_k p(v_k)\, log_2\, p(v_k)$$

- (Apply this notion of entropy to choosing the best attribute)

Entropy and Information Gain

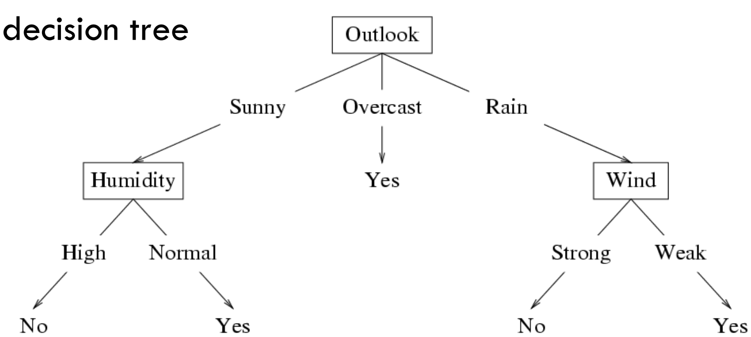
$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

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

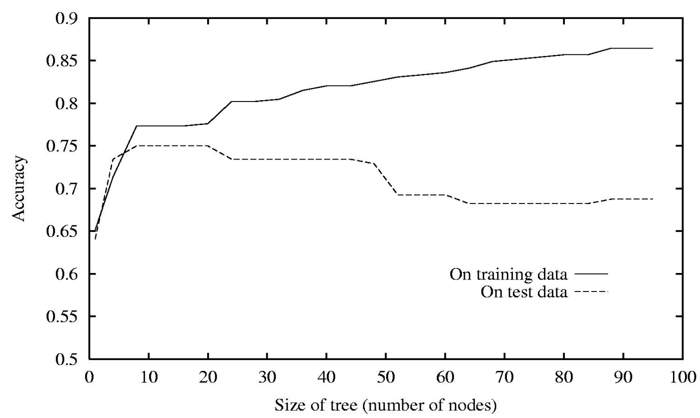
Which is a better feature: wind or humidity?

Decision Trees: additional considerations

- Overfitting can be caused by many factors
  - Noisy data, irrelevant attributes, spurious correlations, non-determinism
- Can cause additional nodes to be added to the decision tree



Decision Trees: additional considerations



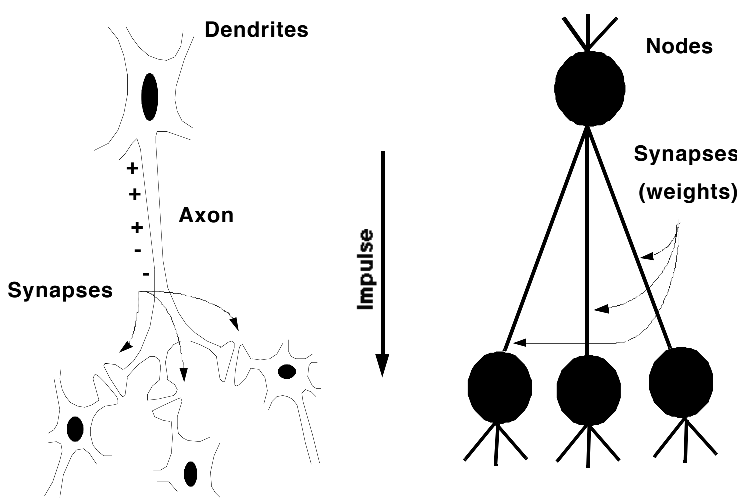
Decision Trees: additional considerations

- Overfitting
  - Can post-process the learned decision tree and prune using significance testing at final nodes
  - Cross-validation using validity set
- Continuous or integer-valued attributes
  - Use ranges
- Continuous label y
  - Combination of splitting and linear regression

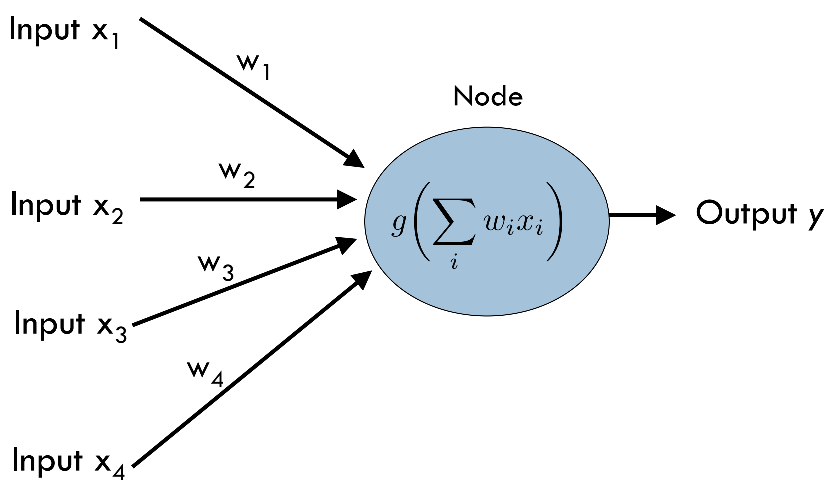
Today

- Reading
  - AIMA 18.6-18.8
  - Note: 18.6 covers regression but also sets up the mathematical background/notation for neural networks
- Goals
  - Perceptron (networks)
  - Perceptron training rule
  - Feed-forward neural networks
  - (Backpropagation)

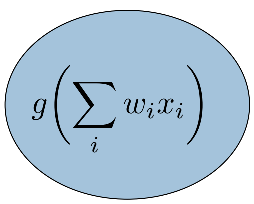
Our Nervous System

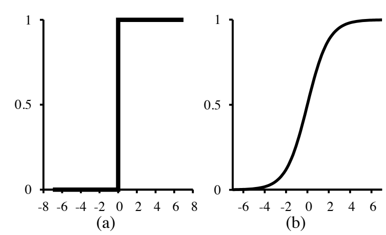


A single perceptron



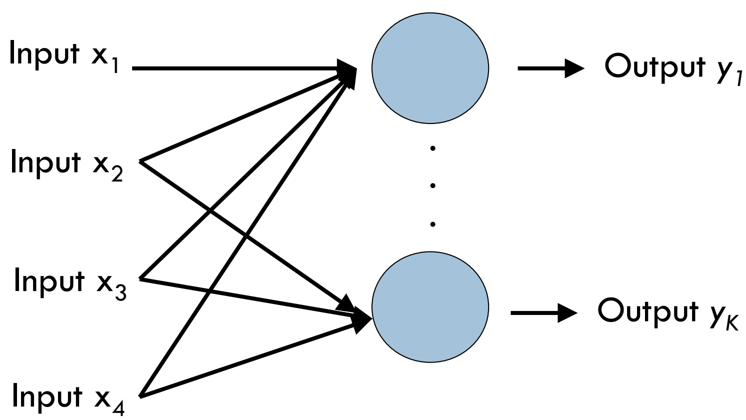
Activation function





- The activation function determines if the "electrical signal" entering the neuron is sufficient to cause it to fire
  - Threshold function – range is {0,1}
  - Sigmoid function – range [0,1]
  - Hyperbolic tangent function – range [-1,1]

A perceptron network



Reduces to K independent perceptrons

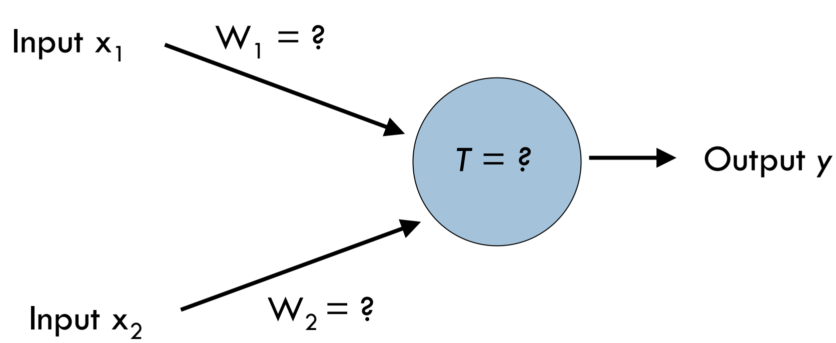
Example: logical operators

- **AND**: If all inputs are 1, return 1. Otherwise return 0
- **OR**: If at least one input is 1, return 1. Otherwise return 0
- **NOT**: Return the opposite of the input
- **XOR**: If exactly one input is 1, then return 1. Otherwise return 0

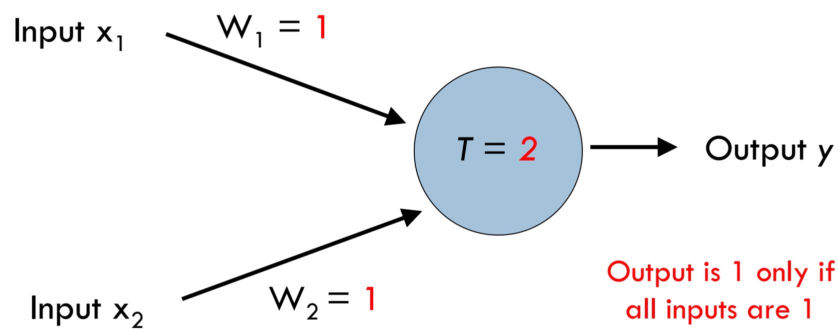
AND

x_1	x_2	x_1 **and** x_2
0	0	0
0	1	0
1	0	0
1	1	1

AND

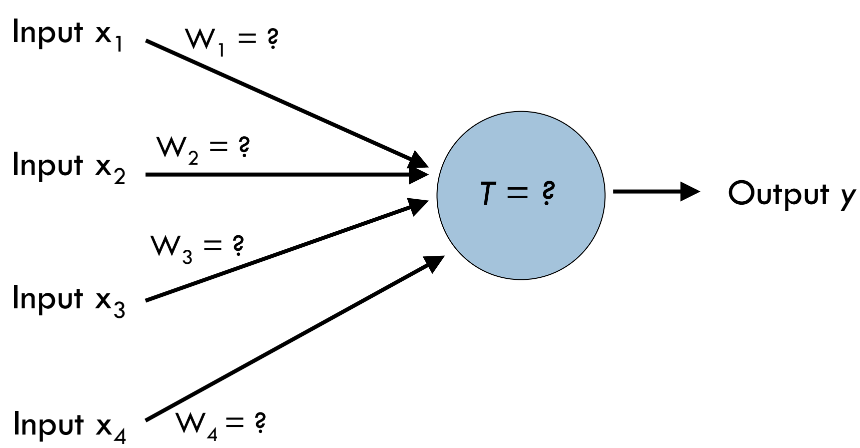


AND

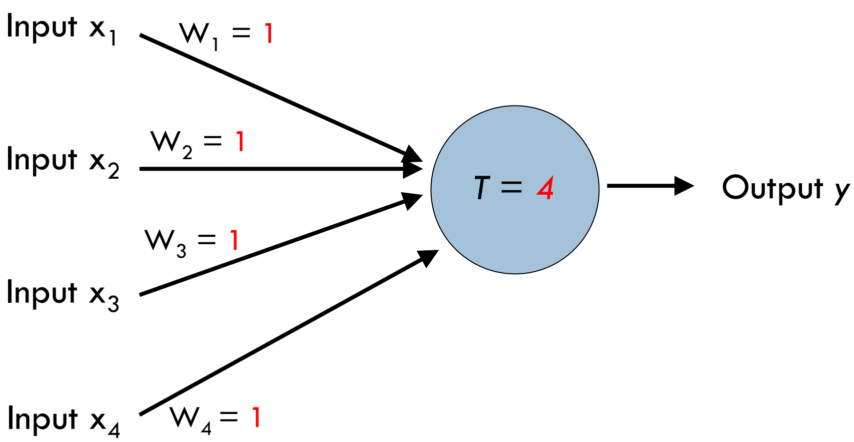


Inputs are 0 or 1

AND



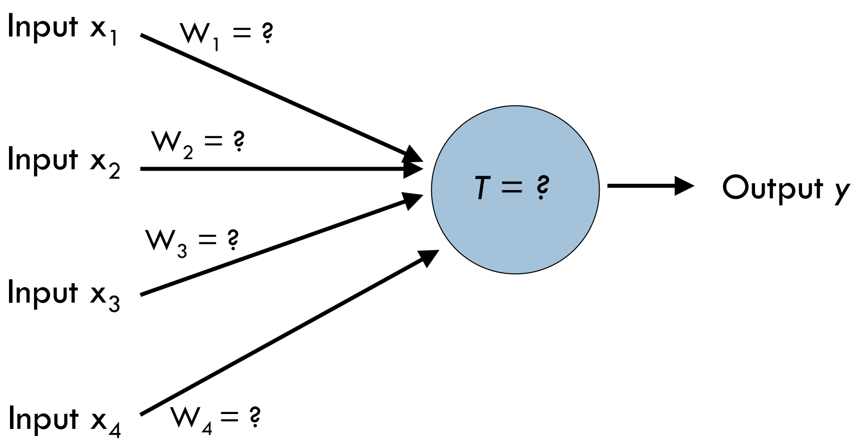
AND



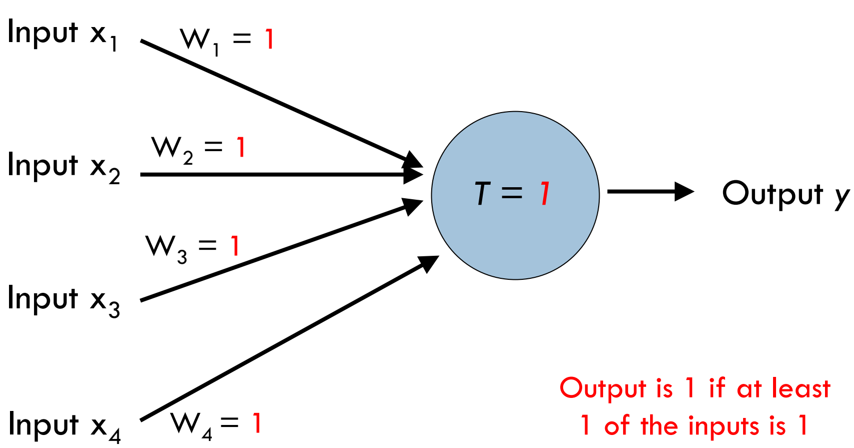
OR

x_1	x_2	x_1 **or** x_2
0	0	0
0	1	1
1	0	1
1	1	1

OR



OR

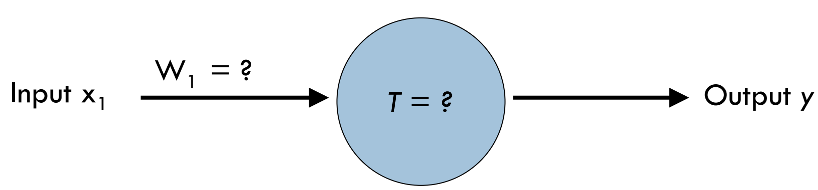


Output is 1 if at least 1 of the inputs is 1

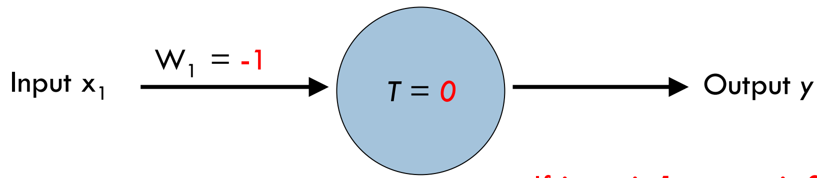
NOT

x_1	**not** x_1
0	1
1	0

NOT



NOT

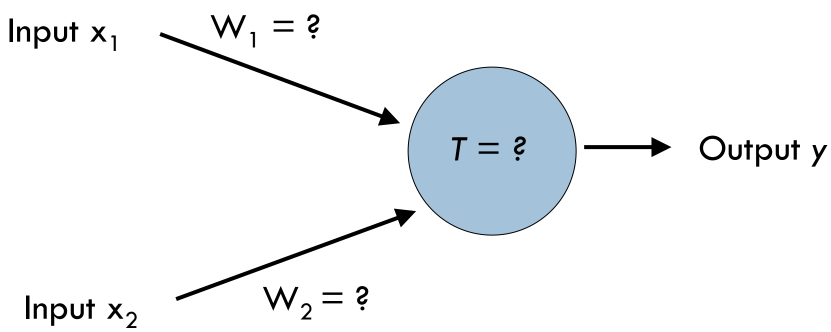


If input is 1, output is 0
If input is 0, output is 1

XOR

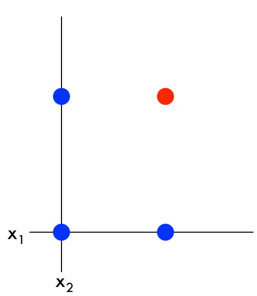
x_1	x_2	x_1 **xor** x_2
0	0	0
0	1	1
1	0	1
1	1	0

XOR



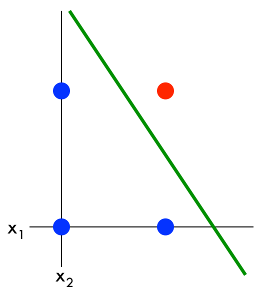
Linearly Separable

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1



Linearly Separable

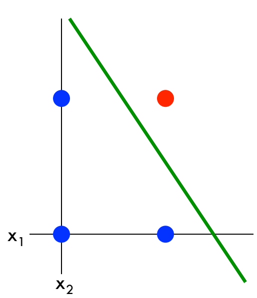
x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

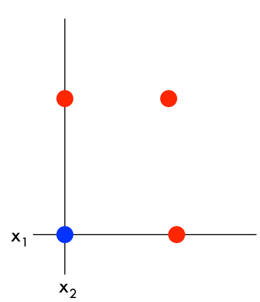


Linearly Separable

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_2	x_1 or x_2
0	0	0
0	1	1
1	0	1
1	1	1

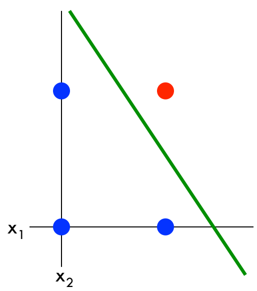


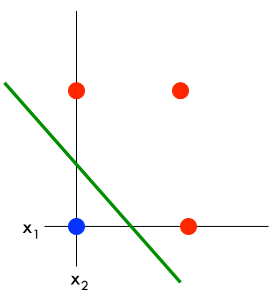


Linearly Separable

x_1	x_2	x_1 **and** x_2
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_2	x_1 **or** x_2
0	0	0
0	1	1
1	0	1
1	1	1



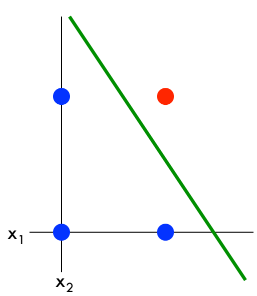


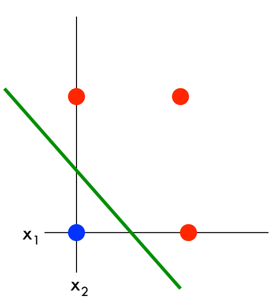
Perceptrons: Linearly separable functions

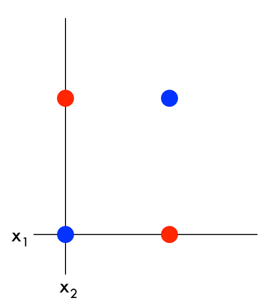
x_1	x_2	x_1 **and** x_2
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_2	x_1 **or** x_2
0	0	0
0	1	1
1	0	1
1	1	1

x_1	x_2	x_1 **xor** x_2
0	0	0
0	1	1
1	0	1
1	1	0







Perceptron Training rule

- Need an algorithm for finding a set of weights w such that
  - The predicted output of the neural network matches the true output for all examples in the training set
  - Predicts a reasonable output for inputs not in the training set

Perceptron Training Rule

1. Begin with randomly initialized weights
2. Apply the perceptron to each training example (each pass through examples is called an epoch)
3. If it misclassifies an example modify the weights
4. Continue until the perceptron classifies all training examples correctly

Perceptron Training Rule

1. Begin with randomly initialized weights
2. Apply the perceptron to each training example (each pass through examples is called an epoch)
3. If it misclassifies an example **modify the weights**
4. Continue until the perceptron classifies all training examples correctly

(Derive gradient-descent update rule)