10-701 Introduction to Machine Learning (PhD)
Lecture 8: Perceptron and Neural Networks

Leila Wehbe
Carnegie Mellon University
Machine Learning Department

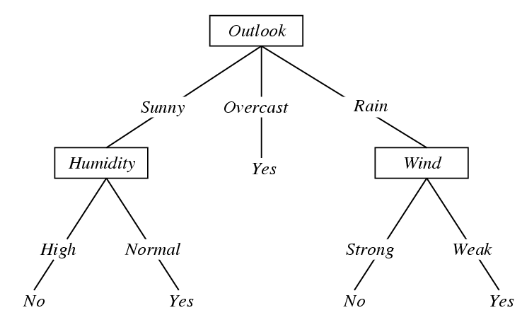
Slides partially based on Tom Mitchell's
10-701 Spring 2016 material
Readings: Tom Mitchell Chapter 3
Hal Daumé III Chapter 4

Decision Trees Review

Simple Training Data Set

Day	Outlook	Temperature	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

A Decision tree for
f: (Outlook, Temperature, Humidity, Wind) → PlayTennis?
(X_1 X_2 X_3 X_4) → Y



Each internal node: test one discrete-valued attribute X_i

Each branch from a node: selects one value for X_i

Each leaf node: predict Y (or $P(Y|X \in \text{leaf})$)

Entropy

Entropy $H(X)$ of a random variable X

$$\boxed{H(X) = -\sum_{i=1}^{n} P(X=i)\log_2 P(X=i)}$$

Specific conditional entropy $H(X|Y=v)$ of X given $Y=v$:

$$H(X|Y=v) = -\sum_{i=1}^{n} P(X=i|Y=v)\log_2 P(X=i|Y=v)$$

Conditional entropy $H(X|Y)$ of X given Y :

$$H(X|Y) = \sum_{v \in values(Y)} P(Y=v)H(X|Y=v)$$

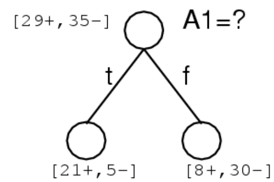
Mutual information (aka Information Gain) of X and Y :

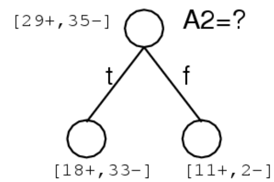
$$I(X,Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

Information Gain is the mutual information between input attribute A and target variable Y

Information Gain is the expected reduction in entropy of target variable Y for data sample S, due to sorting on variable A

$$Gain(S,A) = I_S(A,Y) = H_S(Y) - H_S(Y|A)$$





Example in class

- Try to find IG(Y,X1), IG(Y,X2) and IG(Y,X3)

X1	X2	X3	Y
1	1	1	+
1	1	0	+
0	0	1	-
1	0	0	-

Example in class

- H(Y) = 1
- H(Y|X1=1) = -1/3 log2(1/3) - 2/3 log2(2/3) = 0.92
- H(Y|X1=0) = -1log2(1) = 0
- H(Y|X1) = 3/4 * H(Y|X1=1) +1/4* H(Y|X1=0) ~ 0.92
- IG(Y,X1) ~ 0.31

X1	X2	X3	Y
1	1	1	+
1	1	0	+
0	0	1	-
1	0	0	-

Example in class

- H(Y) = 1
- H(Y|X2=1) = -1log2(1) = 0
- H(Y|X2=0) = -1log2(1) = 0
- H(Y|X2) = 1/2 * H(Y|X2=1) +1/2* H(Y|X2=0)=0
- IG(Y,X2) = 1

- Pick X2!

X1	X2	X3	Y
1	1	1	+
1	1	0	+
0	0	1	-
1	0	0	-

Example in class

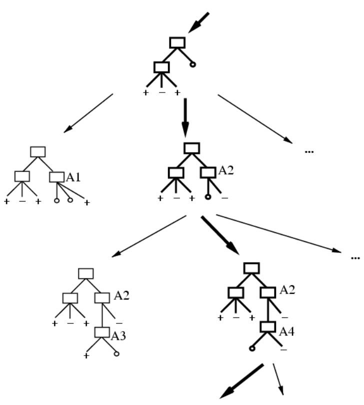
- H(Y) = 1
- H(Y|X3=1) = -1/2log2(1/2) -1/2log2(1/2) = 1
- H(Y|X3=0) = -1/2log2(1/2) -1/2log2(1/2) = 1
- H(Y|X3) = 1/2 * H(Y|X3=1) +1/2* H(Y|X3=0)=1
- IG(Y,X3) = 0

- X3 doesn't help at all at this step

X1	X2	X3	Y
1	1	1	+
1	1	0	+
0	0	1	-
1	0	0	-

Which Tree Should We Output?

- ID3 performs heuristic search through space of decision trees
- It stops at smallest acceptable tree. Why?



Occam's razor: prefer the simplest hypothesis that fits the data

Why Prefer Short Hypotheses? (Occam's Razor)

Argument in favor:

- Fewer short hypotheses than long ones

→ a short hypothesis that fits the data is less likely to be a statistical coincidence

→ highly probable that a sufficiently complex hypothesis will fit the data

Argument opposed:

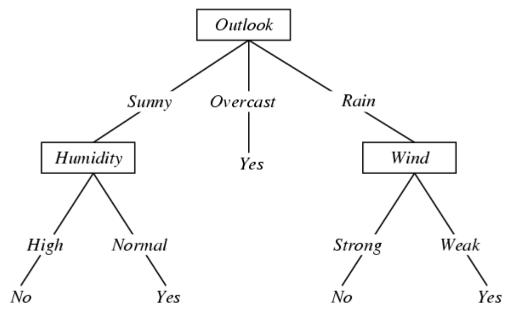
- Also fewer hypotheses with prime number of nodes and attributes beginning with "Z"
- What's so special about "short" hypotheses?

Overfitting in Decision Trees

Consider adding noisy training example #15:

$Sunny,\ Hot,\ Normal,\ Strong,\ PlayTennis = No$

What effect on earlier tree?



Overfitting

Consider a hypothesis h and its

- Error rate over training data: $error_{train}(h)$
- True error rate over all data: $error_{true}(h)$

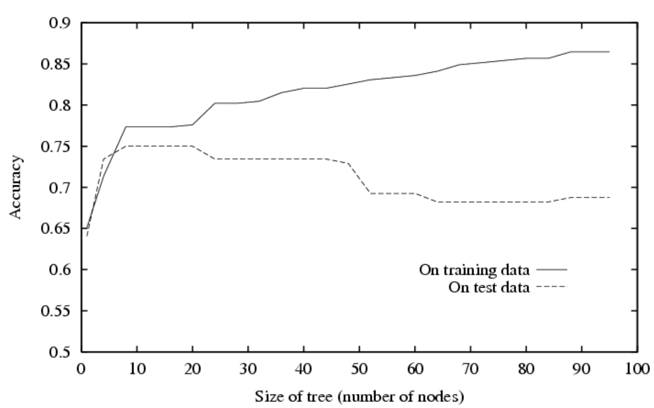
We say h overfits the training data if

$$error_{true}(h) > error_{train}(h)$$

Amount of overfitting =

$$error_{true}(h) - error_{train}(h)$$

Overfitting in Decision Tree Learning



Avoiding Overfitting

How can we avoid overfitting?

- stop growing when data split not statistically significant
- grow full tree, then post-prune

Avoiding Overfitting

How can we avoid overfitting?

- stop growing when data split not statistically significant
- grow full tree, then post-prune

How to select "best" tree:

- Measure performance over training data
- Measure performance over separate validation data set
- MDL: minimize $size(tree) + size(misclassifications(tree))$

Reduced-Error Pruning

Split data into *training* and *validation* set

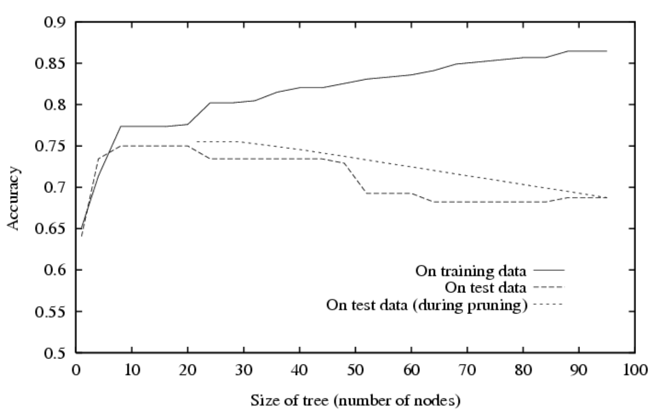
Create tree that classifies *training* set correctly

Do until further pruning is harmful:

1. Evaluate impact on *validation* set of pruning each possible node (plus those below it)
2. Greedily remove the one that most improves *validation* set accuracy

- produces smallest version of most accurate subtree
- What if data is limited?

Effect of Reduced-Error Pruning



Random Forests

Key idea:

1. learn a collection of many trees
2. classify by taking a weighted vote of the trees

Empirically successful. Widely used in industry.

- human pose recognition in Microsoft kinect
- medical imaging – cortical parcellation
- classify disease from gene expression data

How to train different trees

1. Train on different random subsets of data
2. Randomize the choice of decision nodes

Random Forests

Key idea:

1. learn a collection of many trees
2. classify by taking a weighted vote of the trees

more to come

later lecture on boosting

Emp

- hu
- m
- classify disease from gene expression data

How to train different trees

- Train on different random subsets of data
- Randomize the choice of decision nodes

Questions to think about (1)

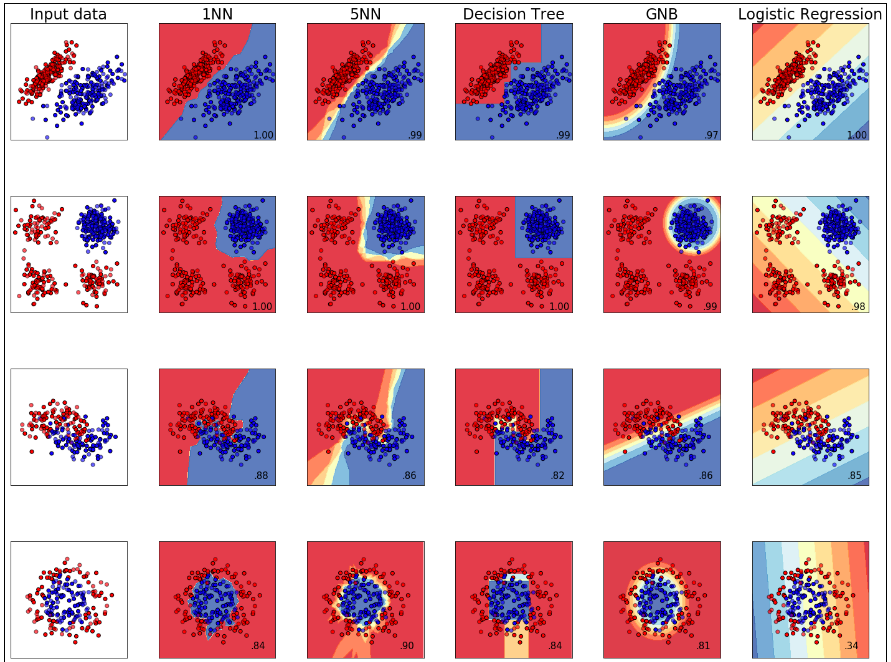
- Consider target function f: (x1,x2) → y, where x1 and x2 are real-valued, y is boolean. What is the set of decision surfaces describable with decision trees that use each attribute at most once?

Questions to think about (2)

- ID3 and C4.5 are heuristic algorithms that search through the space of decision trees. Why not just do an exhaustive search?

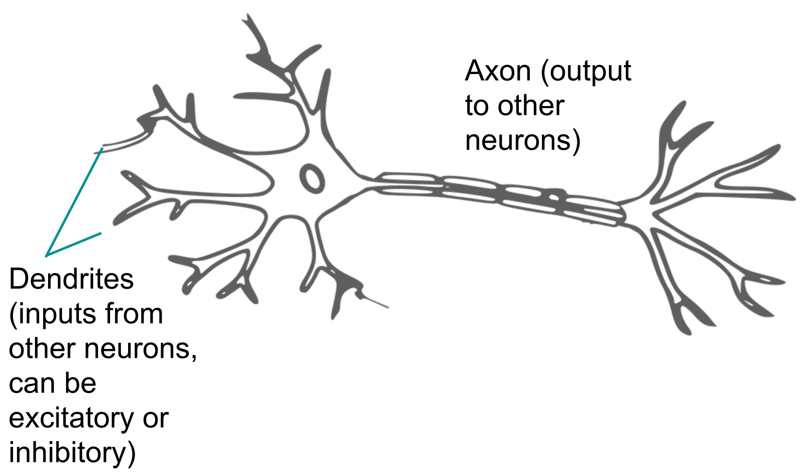
Questions to think about (3)

- Why use Information Gain to select attributes in decision trees? What other criteria seem reasonable, and what are the tradeoffs in making this choice?

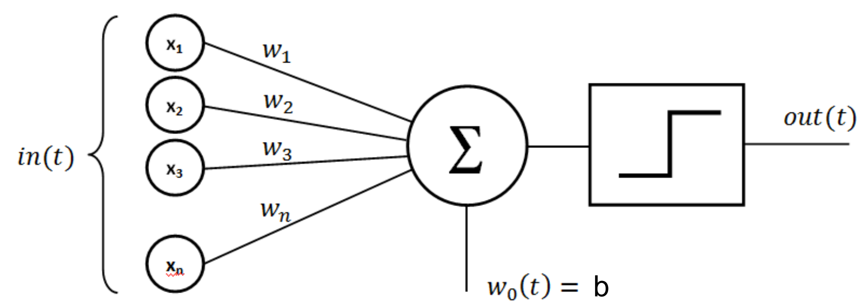


The Perceptron

Inspired by biological neurons

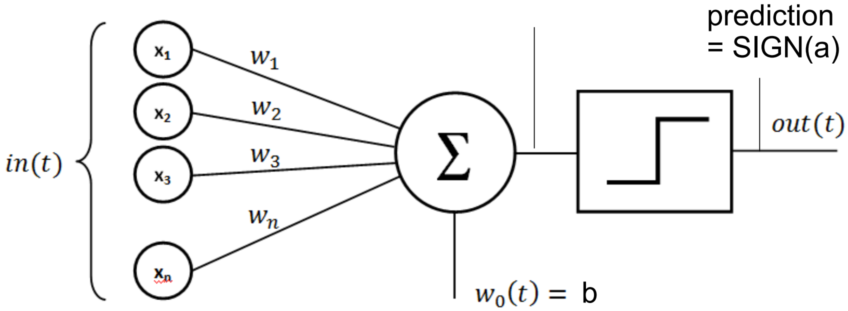


Perceptron



Perceptron

$$a = b + \sum_{d=1}^{D} w_d x_d$$



Error driven learning

$$a = b + \sum_{d=1}^{D} w_d x_d$$

- At each step, return SIGN(a)
- if SIGN(a)≠ y update parameter
- otherwise don't change

Algorithm 5 PERCEPTRONTRAIN($\mathbf{D}$, $MaxIter$)

$w_d \leftarrow 0$, for all $d = 1 \ldots D$ // initialize weights
$b \leftarrow 0$ // initialize bias
for $iter = 1 \ldots MaxIter$ **do**
  for all $(x,y) \in \mathbf{D}$ **do**
    $a \leftarrow \sum_{d=1}^{D} w_d\, x_d + b$ // compute activation for this example
    if $ya \leq 0$ **then**
      $w_d \leftarrow w_d + yx_d$, for all $d = 1 \ldots D$ // update weights
      $b \leftarrow b + y$ // update bias
    end if
  end for
end for
return $w_0, w_1, \ldots, w_D, b$

from http://ciml.info/dl/v0_99/ciml-v0_99-ch08.pdf

Example: y = 1 and prediction is -1

- update **w'** = **w** +y**x** = **w** + **x**
- b' = b + y = b+1

Does this move a in the right direction?

- update **w'** = **w** +y**x** = **w** + **x**
- b' = b + y = b+1

$$\begin{aligned} a' &= \sum_{d=1}^{D} w'_d x_d + b' \\ &= \sum_{d=1}^{D} (w_d + x_d) x_d + (b+1) \\ &= \sum_{d=1}^{D} w_d x_d + b + \sum_{d=1}^{D} x_d x_d + 1 \\ &= a + \sum_{d=1}^{D} x_d^2 + 1 \quad > \quad a \end{aligned}$$

Does this move a in the right direction?

- update **w'** = **w** +y**x** = **w** + **x**
- b' = b + y = b+1

$$\begin{aligned} a' &= \sum_{d=1}^{D} w'_d x_d + b' \\ &= \sum_{d=1}^{D} (w_d + x_d) x_d + (b+1) \\ &= \sum_{d=1}^{D} w_d x_d + b + \sum_{d=1}^{D} x_d x_d + 1 \\ &= a + \sum_{d=1}^{D} x_d^2 + 1 \quad > \quad a \end{aligned}$$

a becomes more positive
(not guaranteed that a>0)

Does this move a in the right direction?

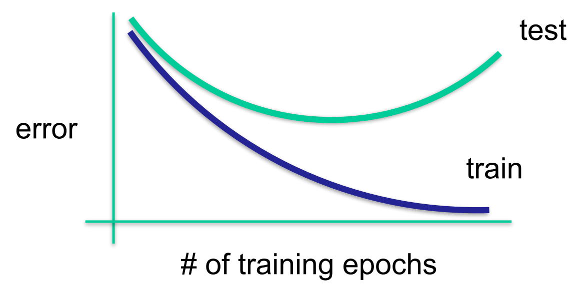
- update **w'** = **w** +y**x** = **w** + **x**
- b' = b + y = b+1

What is the update if
y=-1 and we predict 1

$$\begin{aligned} a' &= \sum_{d=1}^{D} w'_d x_d + b' \\ &= \sum_{d=1}^{D} (w_d + x_d) x_d + (b+1) \\ &= \sum_{d=1}^{D} w_d x_d + b + \sum_{d=1}^{D} x_d x_d + 1 \\ &= a + \sum_{d=1}^{D} x_d^2 + 1 \quad > \quad a \end{aligned}$$

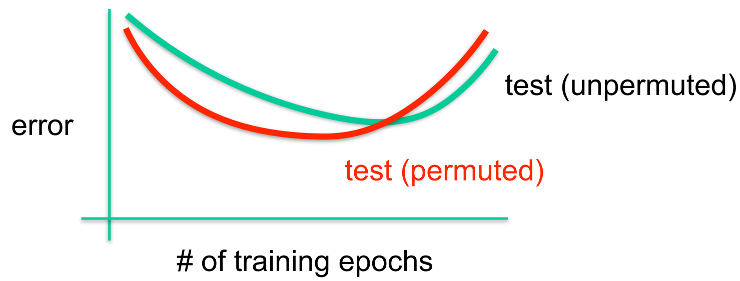
When do we stop?

- Hyperparameter MaxIter
- training too long could lead to overfitting
- training for too few steps could lead to underfitting



Randomizing samples helps

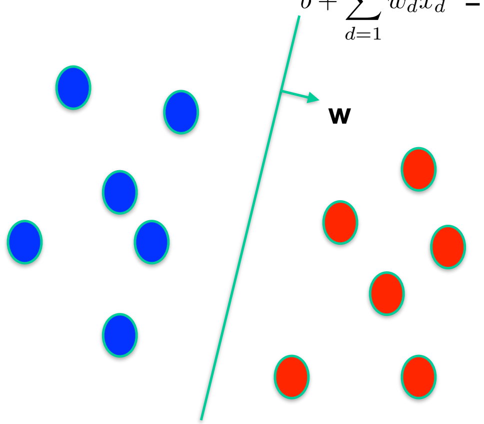
- permute the samples before starting
- even better: permute the samples for each iteration



What is the decision boundary?

What is the decision boundary?

$$b + \sum_{d=1}^{D} w_d x_d = 0$$



How good is this algorithm?

- Convergence: an entire pass without changing the weights.
- If the data is linearly separable, the algorithm will converge. But not necessarily to the “best” boundary

Notion of margin

$$margin(\mathbf{D}, \boldsymbol{w}, b) = \begin{cases} \min_{(x,y)\in\mathbf{D}} y(\boldsymbol{w}\cdot\boldsymbol{x} + b) & \text{if } \boldsymbol{w} \text{ separates } \mathbf{D} \\ -\infty & \text{otherwise} \end{cases}$$

Notion of margin

$$margin(\mathbf{D}, \boldsymbol{w}, b) = \begin{cases} \min_{(x,y)\in\mathbf{D}} y(\boldsymbol{w}\cdot\boldsymbol{x} + b) & \text{if } \boldsymbol{w} \text{ separates } \mathbf{D} \\ -\infty & \text{otherwise} \end{cases}$$

$$margin(\mathbf{D}) = \sup_{\boldsymbol{w},b} margin(\mathbf{D}, \boldsymbol{w}, b)$$

Notion of margin

$$margin(\mathbf{D}, \boldsymbol{w}, b) = \begin{cases} \min_{(x,y)\in\mathbf{D}} y(\boldsymbol{w}\cdot\boldsymbol{x} + b) & \text{if } \boldsymbol{w} \text{ separates } \mathbf{D} \\ -\infty & \text{otherwise} \end{cases}$$

$$margin(\mathbf{D}) = \sup_{\boldsymbol{w},b} margin(\mathbf{D}, \boldsymbol{w}, b)$$

If data is linearly separable with margin γ and $||\mathbf{x}|| \leq 1$, then algorithm will converge in $\frac{1}{\gamma^2}$ updates

Relationship to stochastic gradient descent

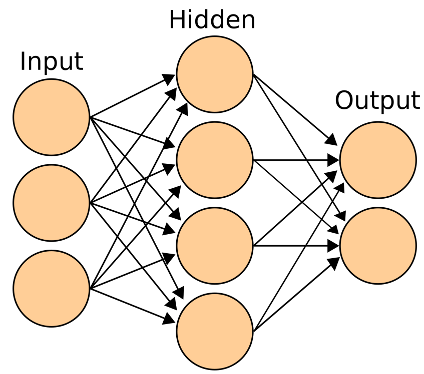
- We can write the loss function of the perceptron as:

  $$L(y, \hat{y}) = \max(0, -y(b + \sum_d w_d x_d))$$

- This is not differentiable, we need to learn more about sub-gradient methods
- At each step, we update using only one datapoint

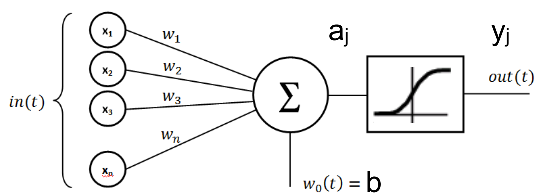
Neural Networks

Every node is analogous to a neuron

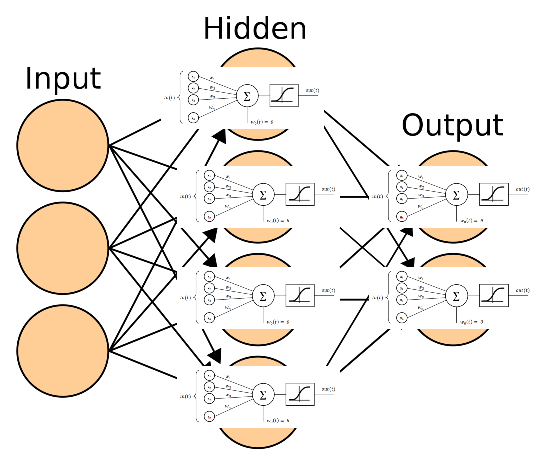


Every node is analogous to a neuron

sigmoid unit



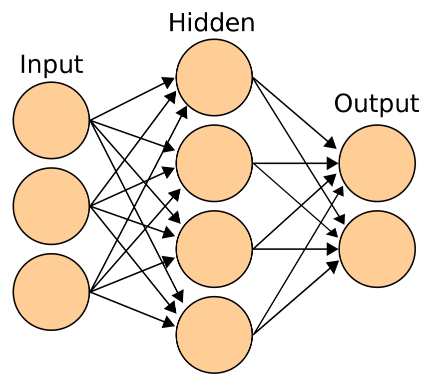
Every node is analogous to a neuron



How to train

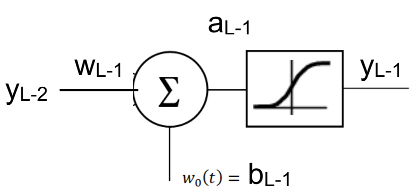
- Calculate each output
- Calculate output error E
- Back-propagate E (weighting it by the gradient of previous layer and activation function)
- Calculate the gradients dE/dw and dE/db
- Update the parameters

Every node is analogous to a neuron



Backprop with one node per layer

sigmoid unit



Backprop with one node per layer

sigmoid unit