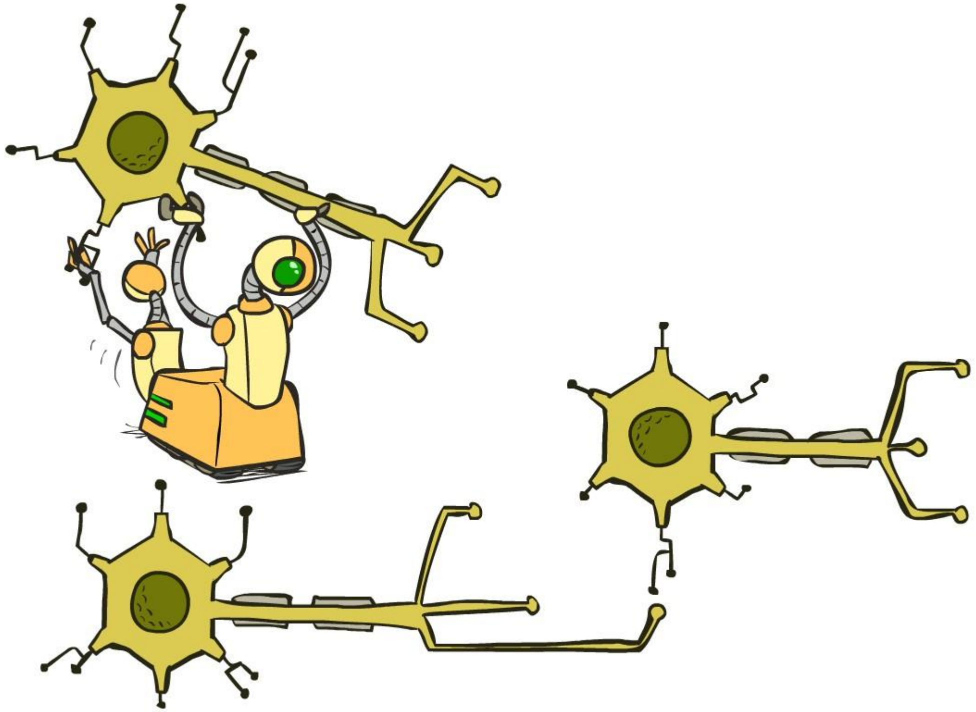
Formalizing Learning
Decision Trees, Neural Networks



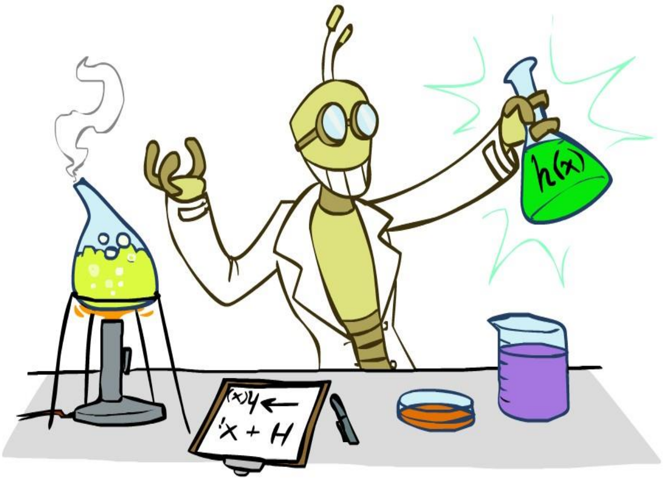
These slides are based on the slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley - http://ai.berkeley.edu.

Today

- Formalizing Learning
- Decision Trees
- Neural Nets

Inductive Learning (Science)

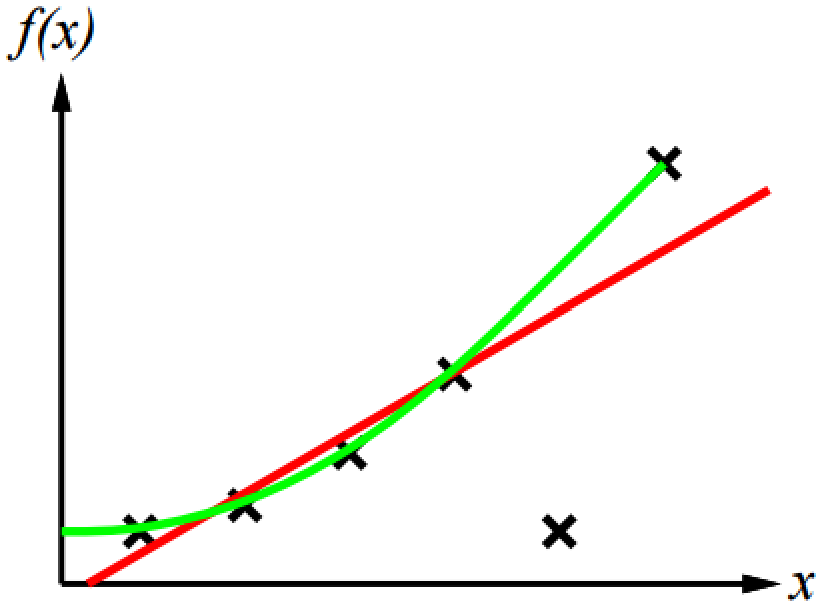
- Simplest form: learn a function from examples
  - Given an unknown function *f*
  - Training examples are pairs: (x, *f*(x))
  - Example: x is an email, *f*(x) is Ham or Spam
  - Example: x is a house and *f*(x) is its price



- Goal:
  - Given a training data set, discover the best hypothesis *h* that approximates the true function *f* best.
- Includes:
  - Classification: *f* (x) is in a finite set of values
  - Regression: *f* (x) can be any number

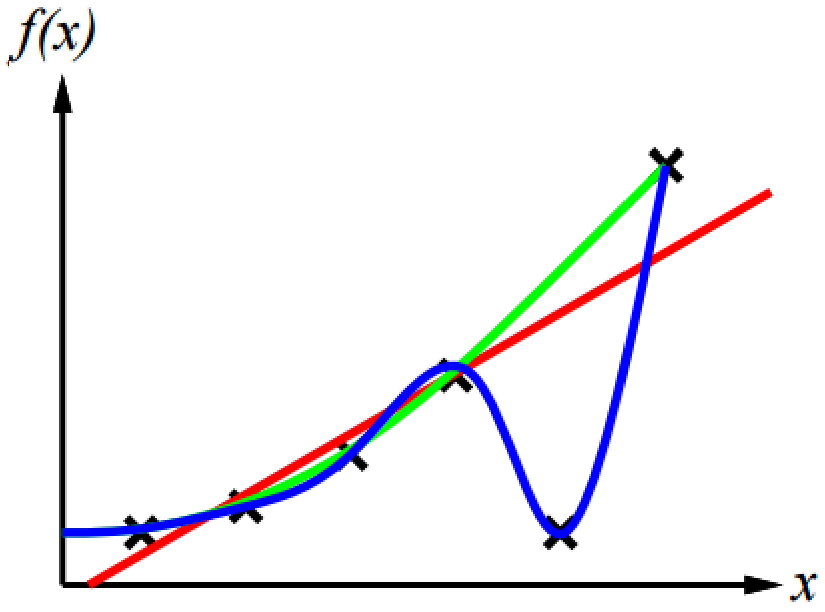
Inductive Learning

- Construct/adjust *h* to agree with *f* on training set
- *h* is consistent if it agrees with *f* on all examples
- curve fitting (regression)



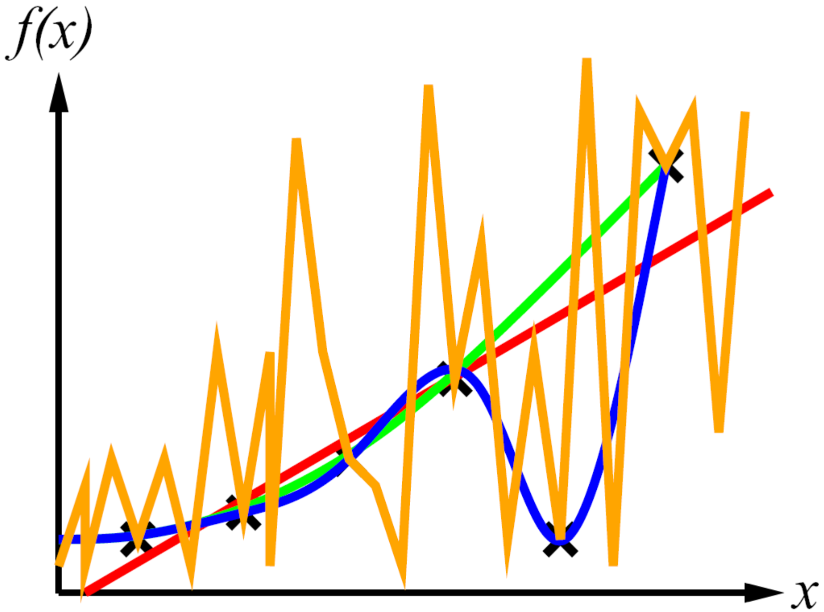
Inductive Learning

- Construct/adjust *h* to agree with *f* on training set
- *h* is consistent if it agrees with *f* on all examples
- curve fitting (regression)



Inductive Learning

- Construct/adjust *h* to agree with *f* on training set
- *h* is consistent if it agrees with *f* on all examples
- curve fitting (regression)



Consistency vs. simplicity

The blue and orange curves are consistent.

The red and green ones are simpler.

Consistency vs Simplicity

Ockham's razor: simpler is better

'Simpler' solutions generalize better – avoid overfitting

Several ways to "simplify"

- Reduce the hypothesis space
  - Assume more: independence assumptions, as in naïve Bayes
  - Have fewer (better) features
- Regularization
  - Smoothing
  - Other generalization parameters

Decision Trees

Decision Tree Representation

A decision tree represents a function that takes as input a feature vector and returns a ‘decision’. A decision is a single output value.

Input and output values may be discrete or continuous.

When the output is one of two possible values, we get Boolean classification.

Example: Will a customer wait for a table?

We have 12 training examples: 6 True (customer will wait) and 6 False.

Example	Attributes aka Features										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

Example: Will a customer wait for a table?

Example	Attributes aka Features										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*T*	*F*	*F*	*T*	*Some*	*\$\$\$*	*F*	*T*	*French*	*0–10*	*T*
X_2					*Full*	*\$*	*F*	*F*	*Thai*	*30–60*	*F*
X_3					*Some*	*\$*	*F*	*F*	*Burger*	*0–10*	*T*
X_4					*Full*	*\$*	*F*	*F*	*Thai*	*10–30*	*T*
X_5					*Full*	*\$\$\$*	*F*	*T*	*French*	*>60*	*F*
X_6					*Some*	*\$\$*	*T*	*T*	*Italian*	*0–10*	*T*
X_7	*F*	*T*	*F*	*F*	*None*	*\$*	*T*				*F*
X_8	*F*	*F*	*F*	*T*	*Some*	*\$\$*	*T*				*T*
X_9	*F*	*T*	*T*	*F*	*Full*	*\$*	*T*				*F*
X_{10}	*T*	*T*	*T*	*T*	*Full*	*\$\$\$*	*F*				*F*
X_{11}	*F*	*F*	*F*	*F*	*None*	*\$*	*F*		*Thai*	*0–10*	*F*
X_{12}	*T*	*T*	*T*	*T*	*Full*	*\$*	*F*	*F*	*Burger*	*30–60*	*T*

Features have discrete values (True, False, Some, etc…)

Classification of examples is positive (T) or negative (F)

Example: Will a customer wait for a table?

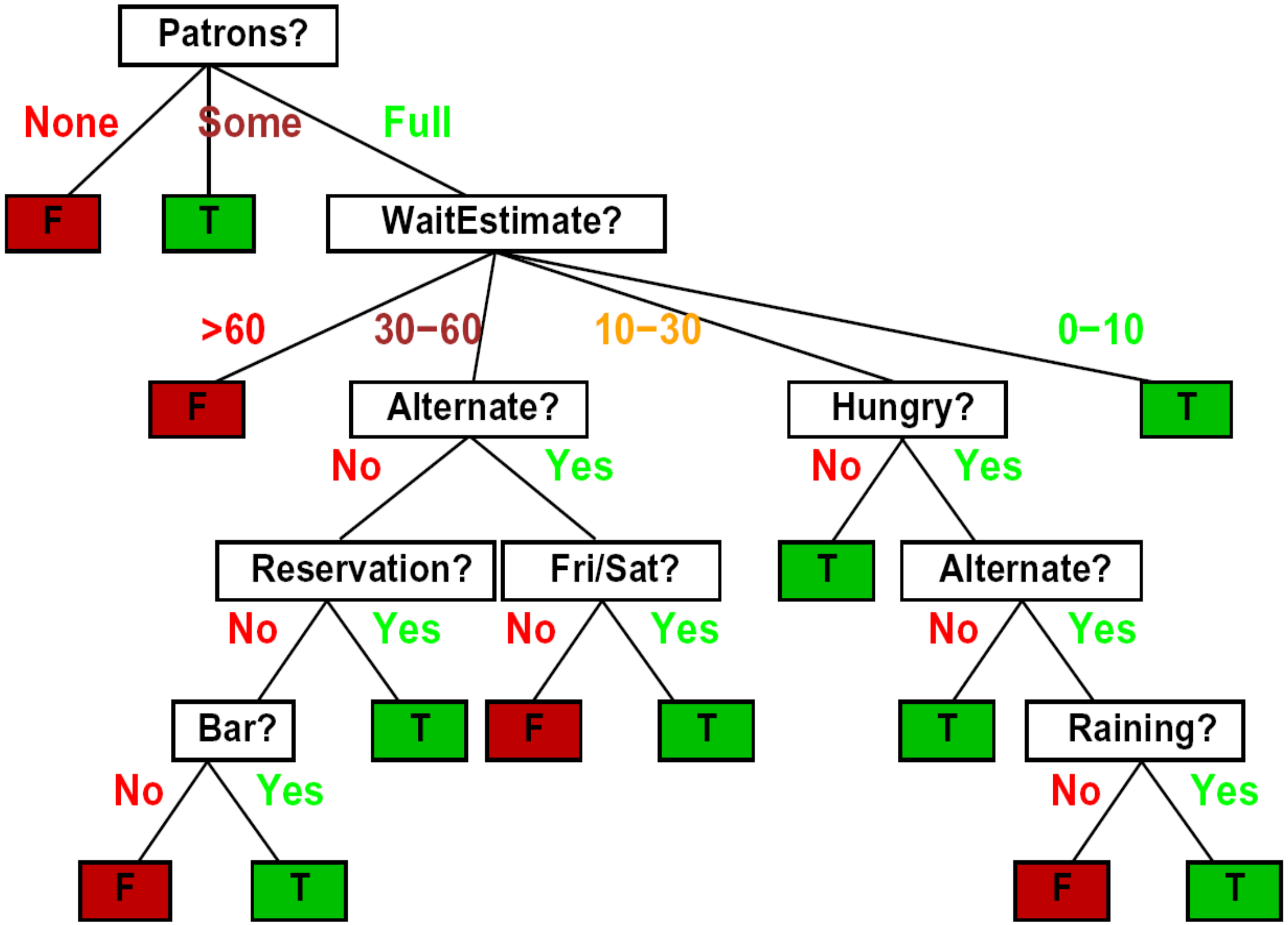
Our goal is to be able to predict whether a customer will wait today.

It is hard to guess. Let's try to understand.

Example	Attributes aka Features										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

Decision Tree

Decision tree representing the 'true' function:



Expressiveness of Decision Trees

Decision trees can express any function of the features

Example:

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F

Trivially, there is a consistent decision tree for any training set with one path to leaf for each example (unless *f* is nondeterministic) but this tree probably won't generalize to new examples.

Prefer to find more compact decision trees.

Comparison: Decision Trees vs Perceptrons

- What is the expressiveness of a perceptron over these features?

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*T*	*F*	*F*	*T*	*Some*	*$$$*	*F*	*T*	*French*	*0–10*	*T*
X_2	*T*	*F*	*F*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*30–60*	*F*

- For a perceptron, a feature's contribution is either positive or negative
- If we want one feature's effect to depend on another, we have to explicitly add a new feature that is the conjunction of two features.

  For example adding "PATRONS=full and WAIT = 60" allows a perceptron to model the interaction between the two atomic features.
- Decision Trees automatically conjoin features .
- Features can have different effects in different branches of the tree!

Hypothesis Spaces

How many distinct decision trees with n Boolean features?

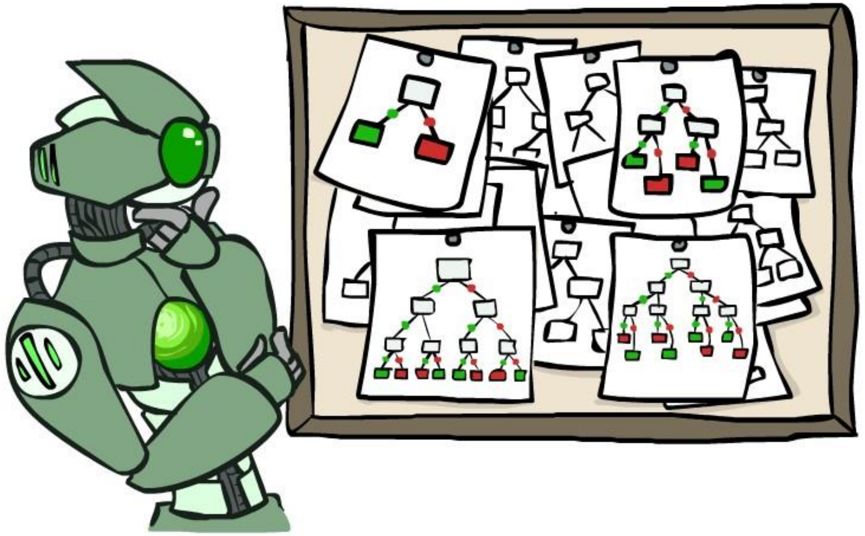
= number of Boolean functions

= number of distinct truth tables with 2^n rows

= 2^{2^n}

With 10 Boolean features:

2^{1024} trees or about 10^{308} trees



Decision Tree Learning

Aim: find a small tree consistent with the training examples

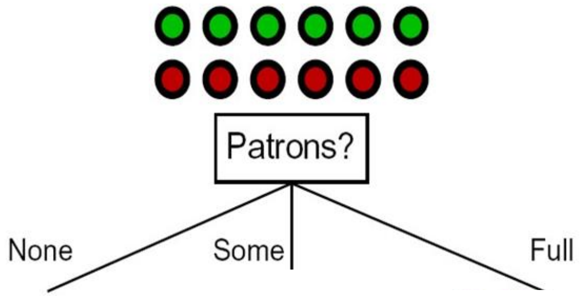
Idea: (recursively) choose "most significant" feature/attribute as root

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return MODE(examples)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value v_i of best do
            examples_i ← {elements of examples with best = v_i}
            subtree ← DTL(examples_i, attributes − best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

Choosing a Feature/Attribute

Goal: find a small tree

How do we choose the "most significant" feature?

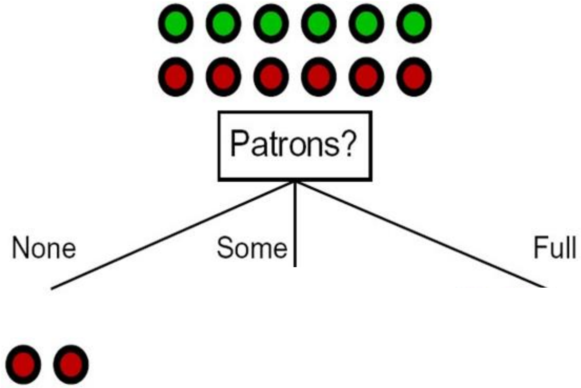


Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

Choosing a Feature/Attribute

Goal: find a small tree

How do we choose the "most significant" feature?

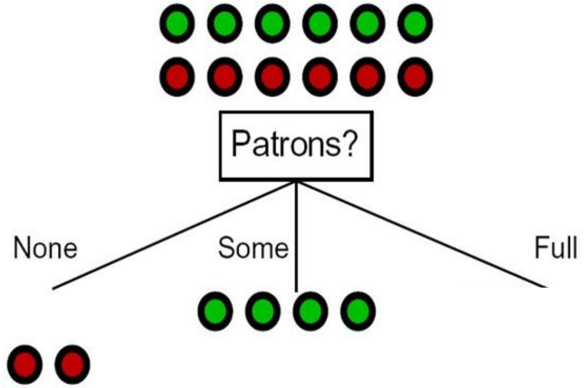


Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	T	F	F	T	Some	$\$\$\$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$\$$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$\$$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$\$$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$\$\$\$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$\$\$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$\$$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$\$\$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$\$$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$\$\$\$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$\$$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$\$$	F	F	Burger	30–60	T

Choosing a Feature/Attribute

Goal: find a small tree

How do we choose the "most significant" feature?

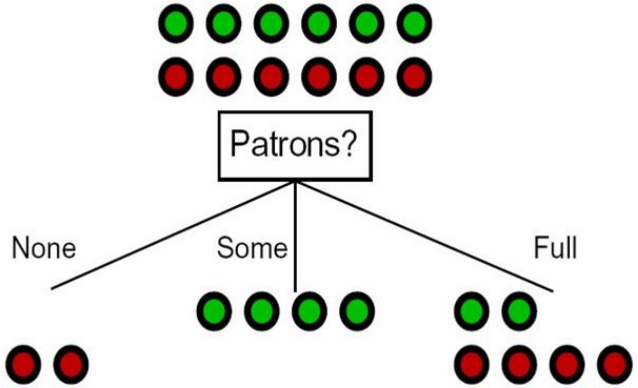


Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

Choosing a Feature/Attribute

Goal: find a small tree

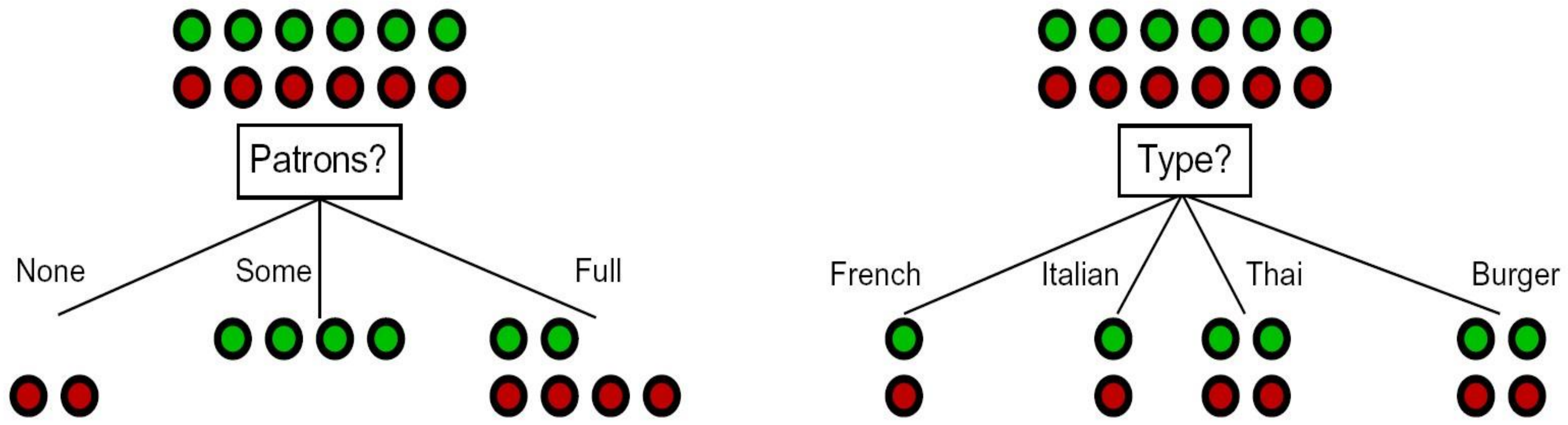
How do we choose the "most significant" feature?



Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0–10	T
X_2	T	F	F	T	Full	\$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	\$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	\$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	\$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	\$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30–60	T

Choosing a Feature/Attribute

Idea: a good attribute splits the examples into subsets where each subset is (ideally) "all positive" or "all negative"



Patrons? is a better choice - gives more information about the classification

We need a measure of how "good" a split is, even if the results aren't perfectly separated out

Information and Entropy

- Information answers questions
- The more clueless I am about the answer initially, the more information is contained in the answer.

Information and Entropy

- A domain variable with only one value such as a coin that always comes up heads, has no uncertainty. Its entropy is defined as zero: we gain no information by observing its value.
- A two-sided fair coin gives one bit of information if we know its value (1 out of two possible values).
- A pair of coins gives two bits of information (1 out of 4 possible values).

Information and Entropy

- Measure: bits needed to represent the answer
  - Answer to Boolean question with prior <1/2, 1/2>?

    1 bit
  - Answer to 4–way question with prior <1/4, 1/4, 1/4, 1/4>?

    2 bits

A uniform distribution of size n (n possible values, p = 1/n)

- The information can be represented with a code of length:

  $\log_2 n = \log_2 1/p = -\log_2 p$

Information and Entropy

What if the distribution is not uniform?

Answer to 4–way question with prior <1/2, 1/4, 1/8, 1/8>?

Information in an answer when prior is <P1, . . . , Pn>:

$H(<P_1, . . . , P_n>) = \sum_{i=1}^{n} - Pi \log_2 Pi$

This is called entropy of the prior.

Information and Entropy

Entropy: H(<P_1, . . . , P_n>) = $\sum_{i=1}^{n} - P_i \log_2 P_i$

Answer to 4–way question with prior <1/2, 1/4, 1/8, 1/8>?

$$H = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{4}\log_2\frac{1}{4} - \frac{1}{8}\log_2\frac{1}{8} - \frac{1}{8}\log_2\frac{1}{8}$$

$$= \frac{1}{2} + \frac{1}{2} + \frac{3}{8} + \frac{3}{8} = \frac{7}{4}$$

= 1.75

compared to 2 for the uniform distribution H (<1/4, 1/4, 1/4, 1/4>)

$$\log_2\frac{1}{2} \text{ is } -1$$

$$\log_2\frac{1}{4} \text{ is } -2$$

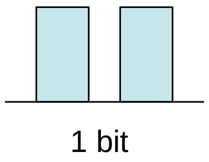
$$\log_2\frac{1}{8} \text{ is } -3$$

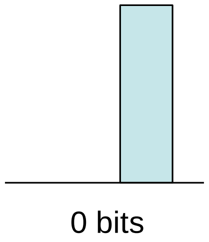
Information and Entropy

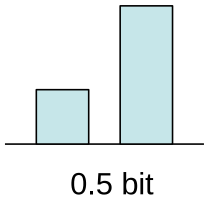
More uniform = higher entropy

More values = higher entropy

More peaked = lower entropy





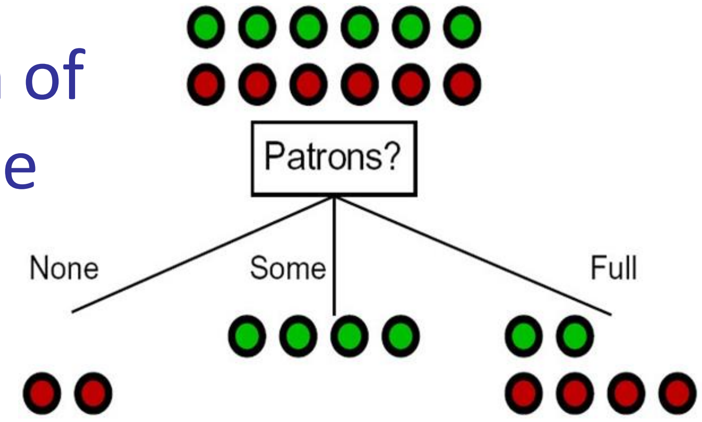


Choosing the Best Feature

We have p positive and n negative examples at the root.

H(< $\frac{p}{p+n}$, $\frac{n}{p+n}$ >) bits are needed to classify a new example.

An attribute splits the examples E into *i* subsets E_i, each of which (we hope) needs less information to complete the classification.



Let E_i have p_i positive and n_i negative examples:

H(< $\frac{p_i}{p_i+n_i}$, $\frac{n_i}{p_i+n_i}$ >) are bits needed to classify a new example in branch i.

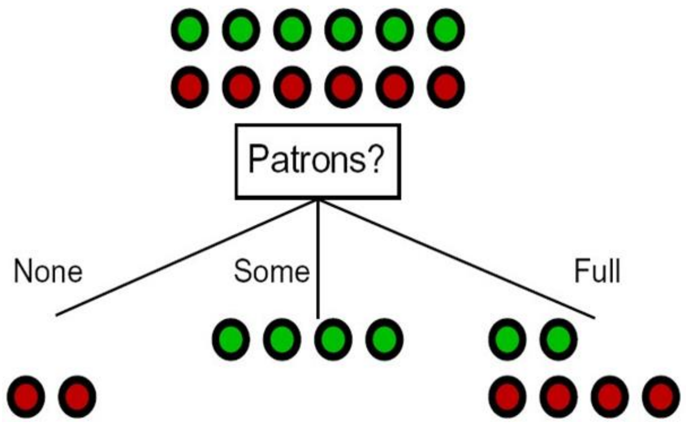
Expected number of bits per example over all branches is:

$\sum_i \frac{p_i+n_i}{p+n}$ H(< $\frac{p_i}{p_i+n_i}$, $\frac{n_i}{p_i+n_i}$ >)

Choosing he Best Feature

Expected entropy per example over all branches is

$$\sum_i \frac{p_i+n_i}{p+n} H(<\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}>)$$



For Patrons?:

$$= \frac{2}{12} H_{None} + \frac{4}{12} H_{Some} + \frac{1}{2} H_{Full}$$

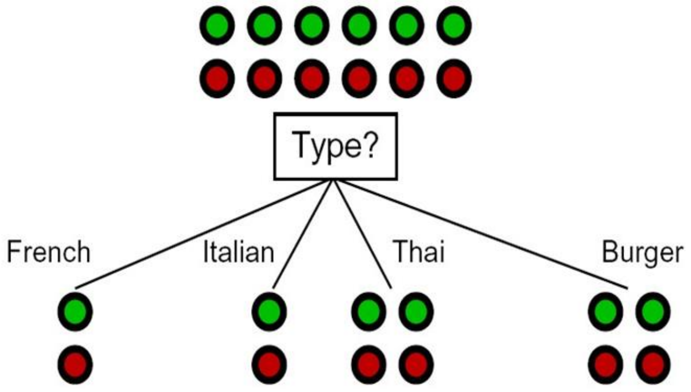
$$= 0 + 0 + \frac{1}{2}\ H(<\frac{2}{6}, \frac{4}{6}>) = \frac{1}{2}(- \frac{2}{6} \log_2 \frac{2}{6}\ - \frac{4}{6} \log_2 \frac{4}{6})$$

$$= 0.46$$

Choosing the Best Feature

Expected entropy per example over all branches is

$$\sum_i \frac{p_i+n_i}{p+n} H(<\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}>)$$



For Type?:

$$\frac{2}{12} H(<\frac{1}{2}, \frac{1}{2}>) + \frac{2}{12} H(<\frac{1}{2}, \frac{1}{2}>) + \frac{4}{12} H(<\frac{1}{2}, \frac{1}{2}>) + \frac{4}{12} H(<\frac{1}{2}, \frac{1}{2}>)$$

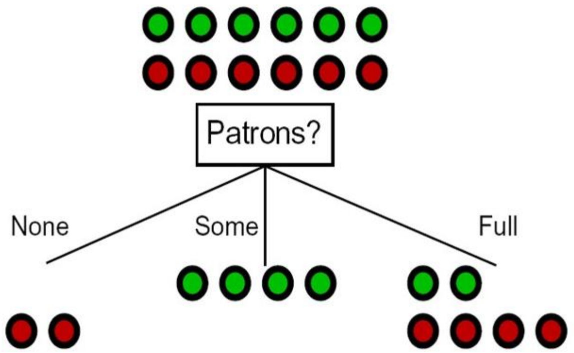
$= 1$

We choose the attribute that minimizes the remaining information needed: Patrons?

Next Step: Recurse

Now we need to keep growing the tree!

Two branches are done (why?)

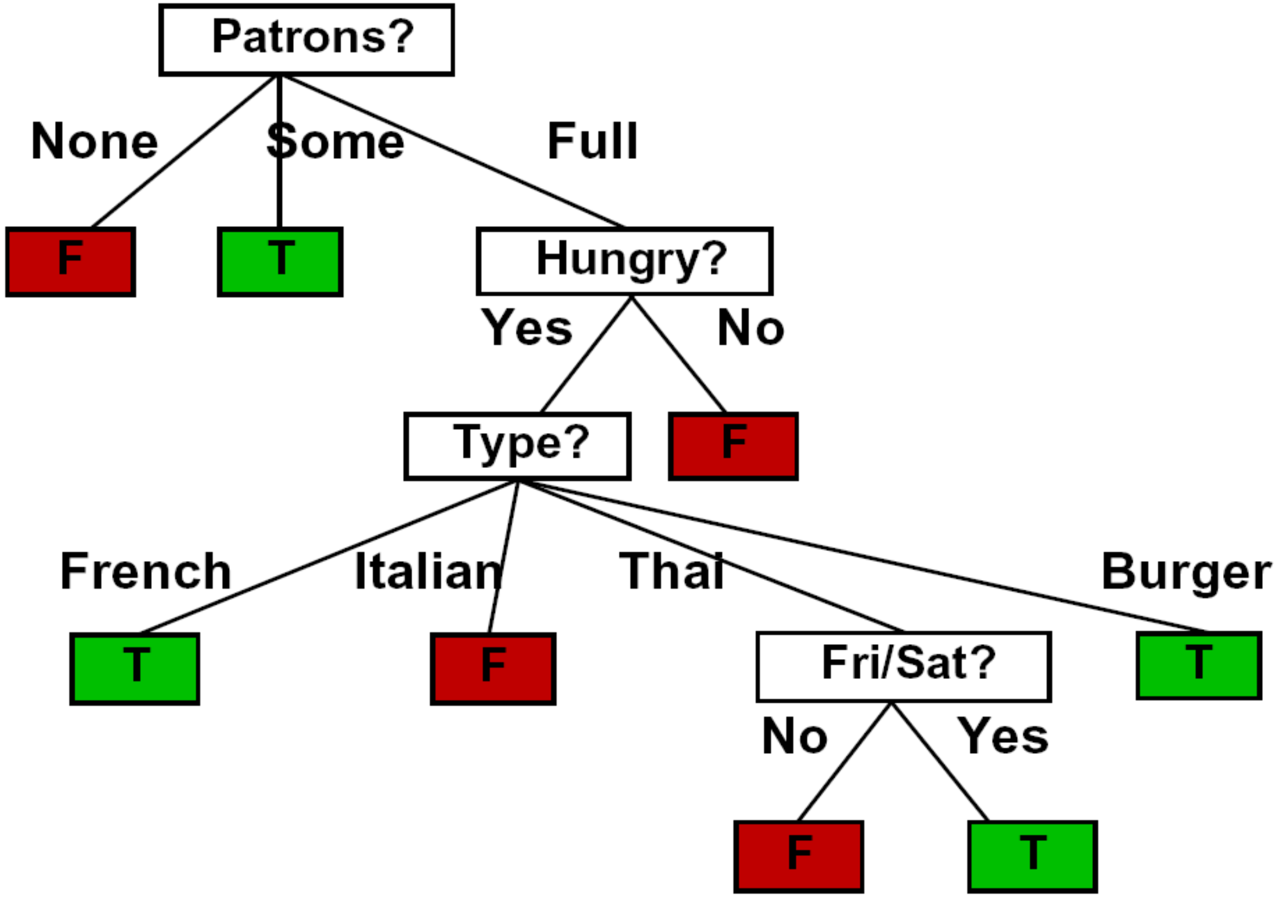


Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

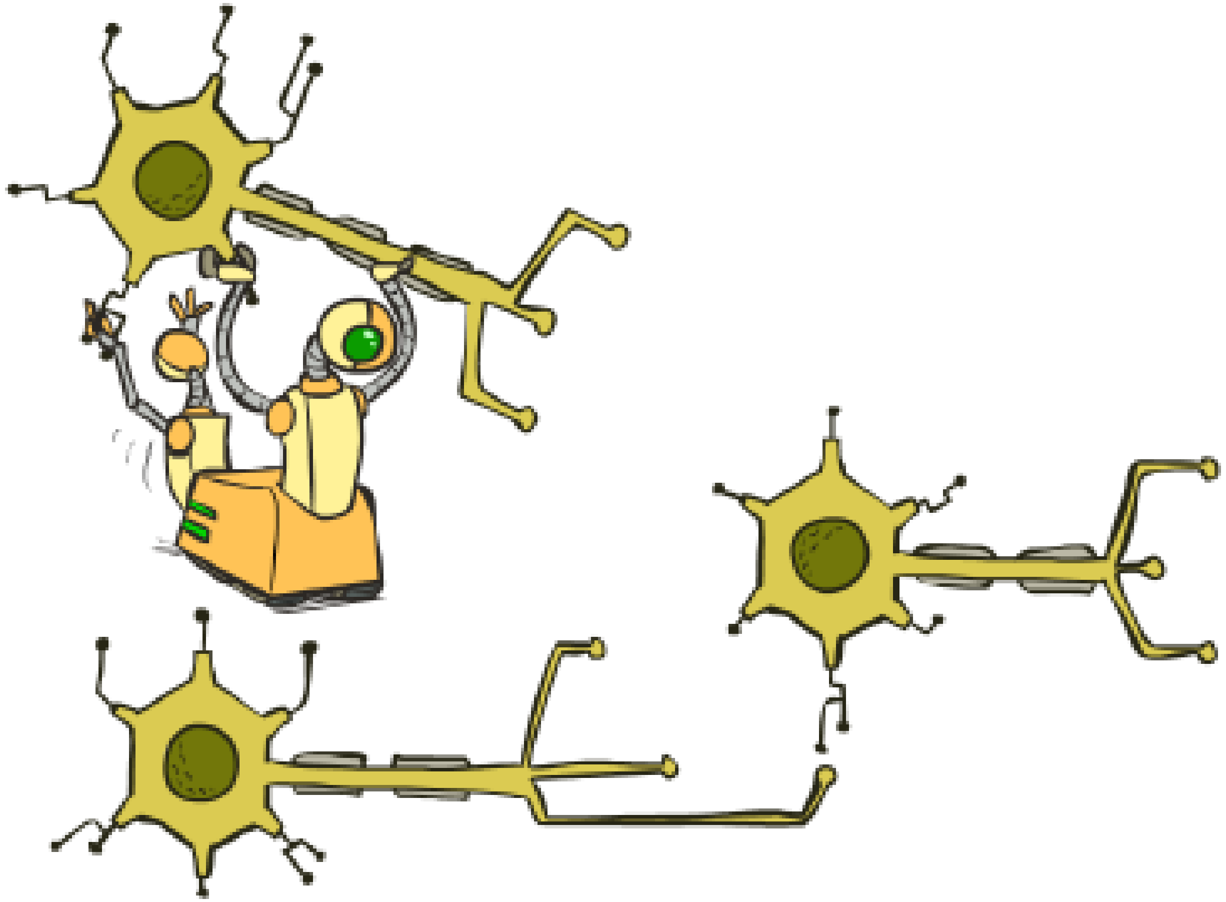
Choosing the Best Feature

Decision tree learned from these 12 examples:

Substantially simpler than ‘true’ tree.

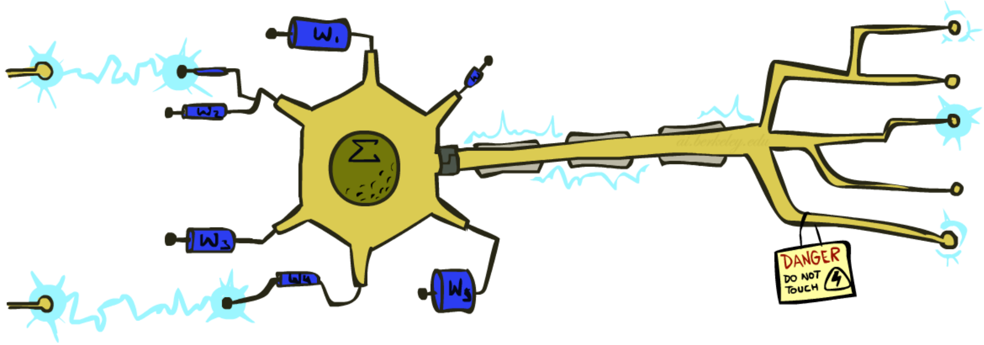


Artificial Neural Networks



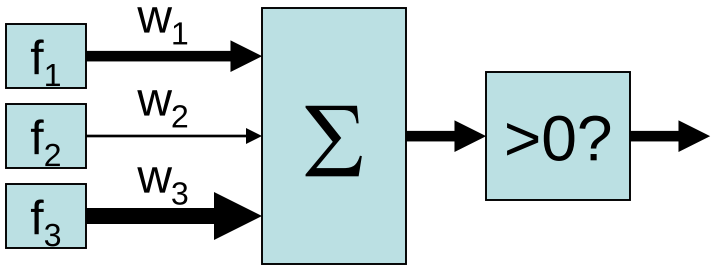
Reminder: Perceptrons

- Inputs are feature values (vectors)
- Each feature has a weight
- Sum is the activation

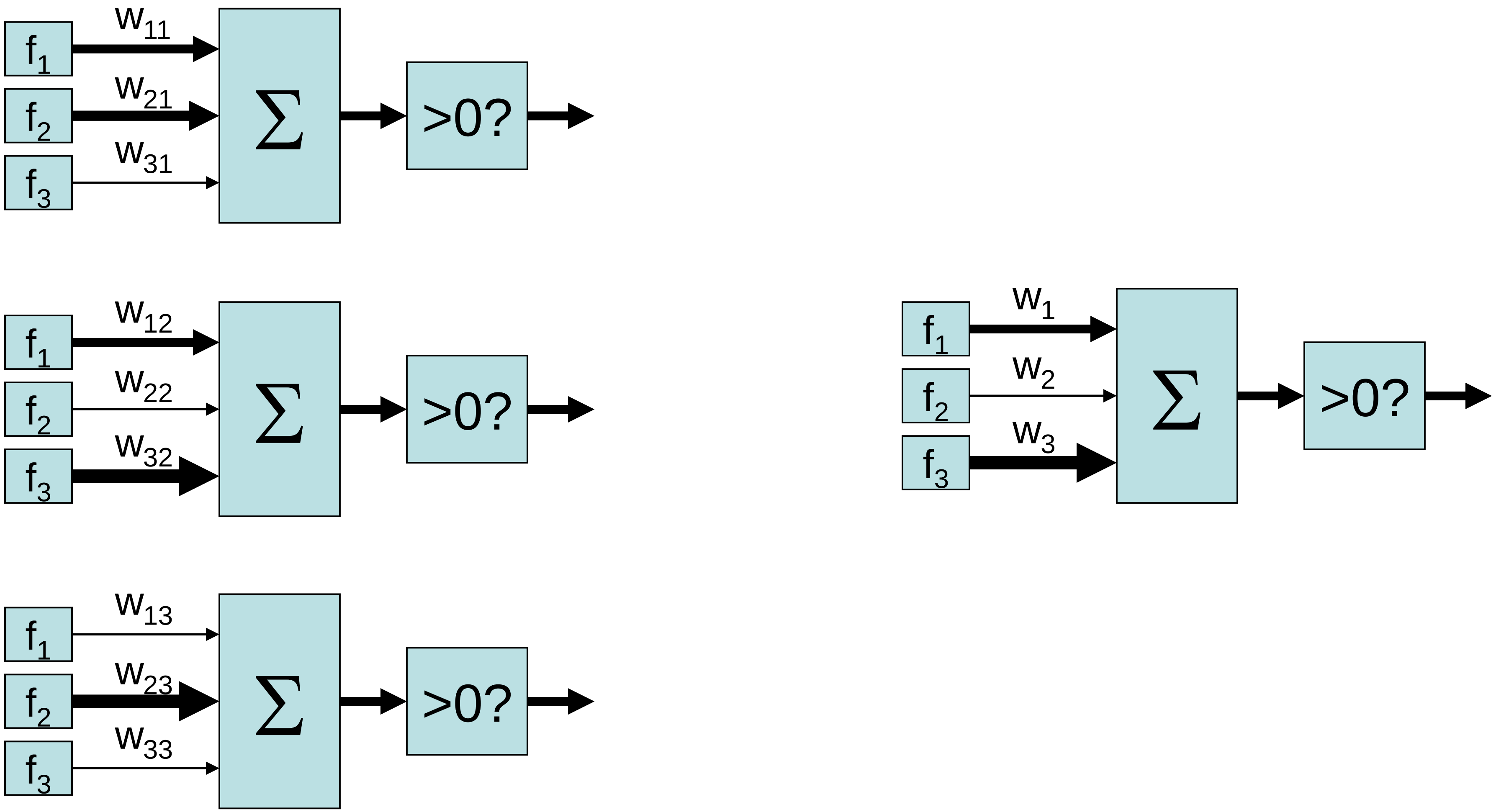


$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

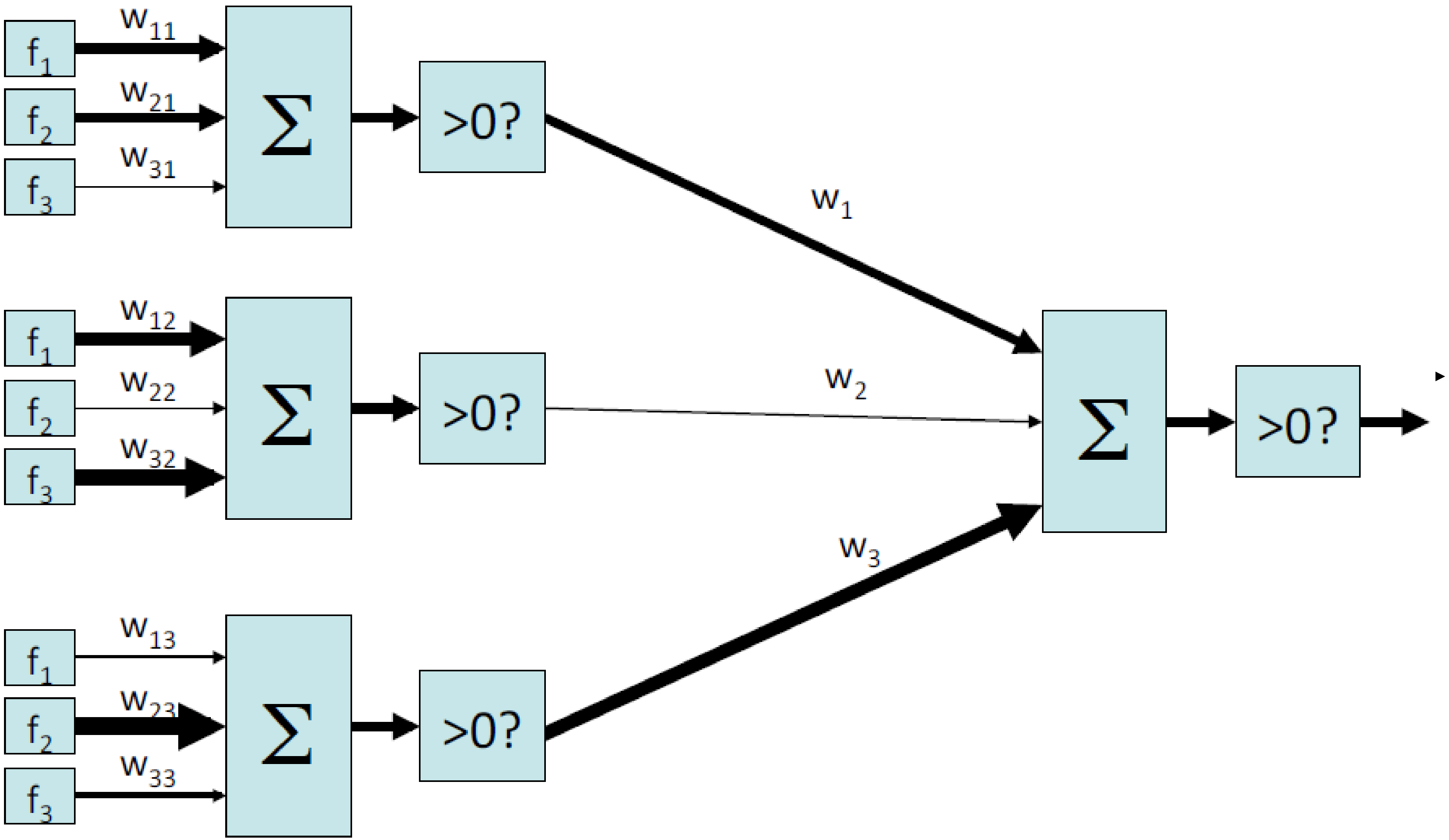
- If the activation is:
  - Positive, output +1
  - Negative, output -1



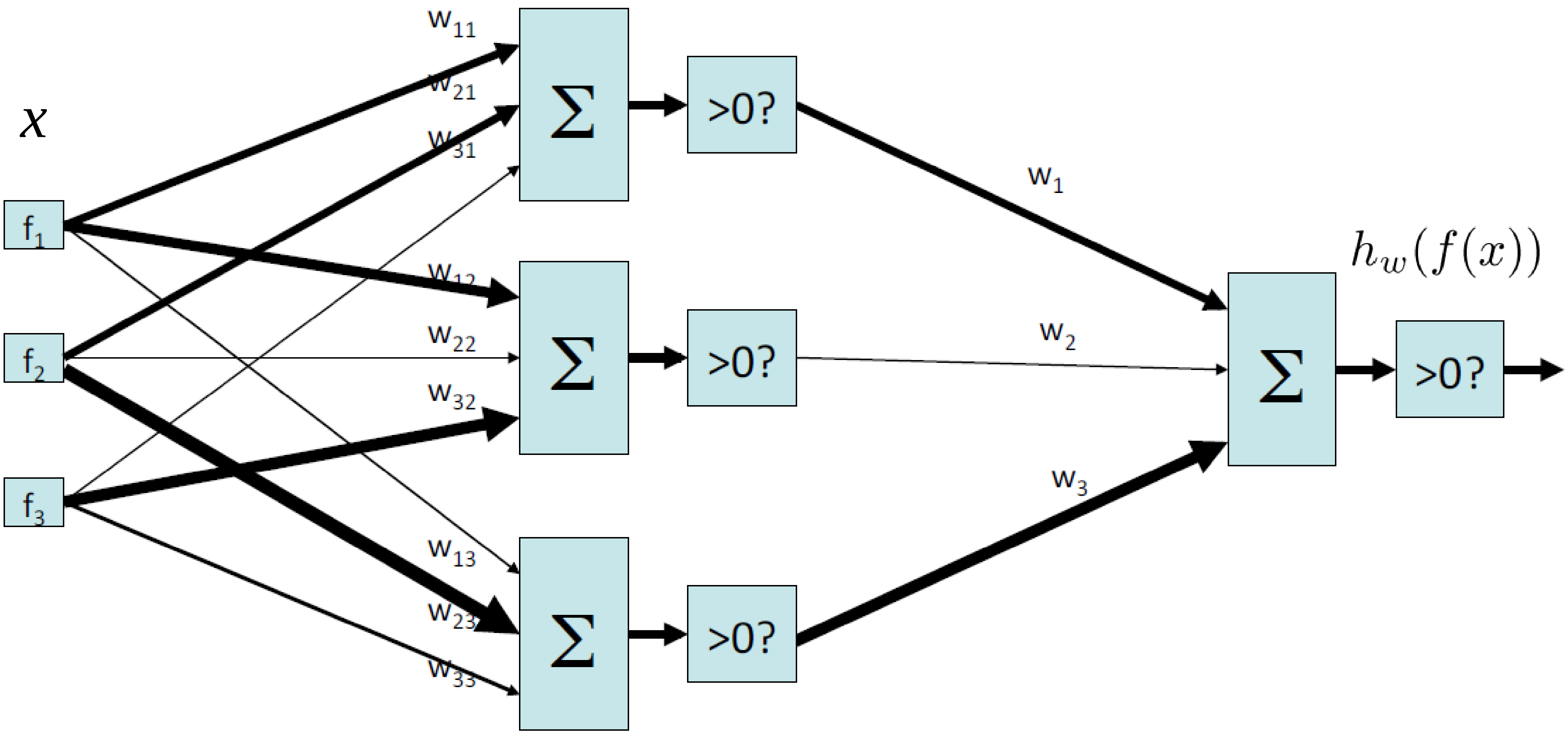
Two-Layer Perceptron Network



Two-Layer Perceptron Network



Two-Layer Perceptron Network



Learning w

- Training Examples:

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$$

- Goal: find the weight vector *w* that minimizes the error (loss)

$$\min_w \sum_{i=1}^{m} \left(y^{(i)} - h_w(f(x^{(i)})) \right)^2$$

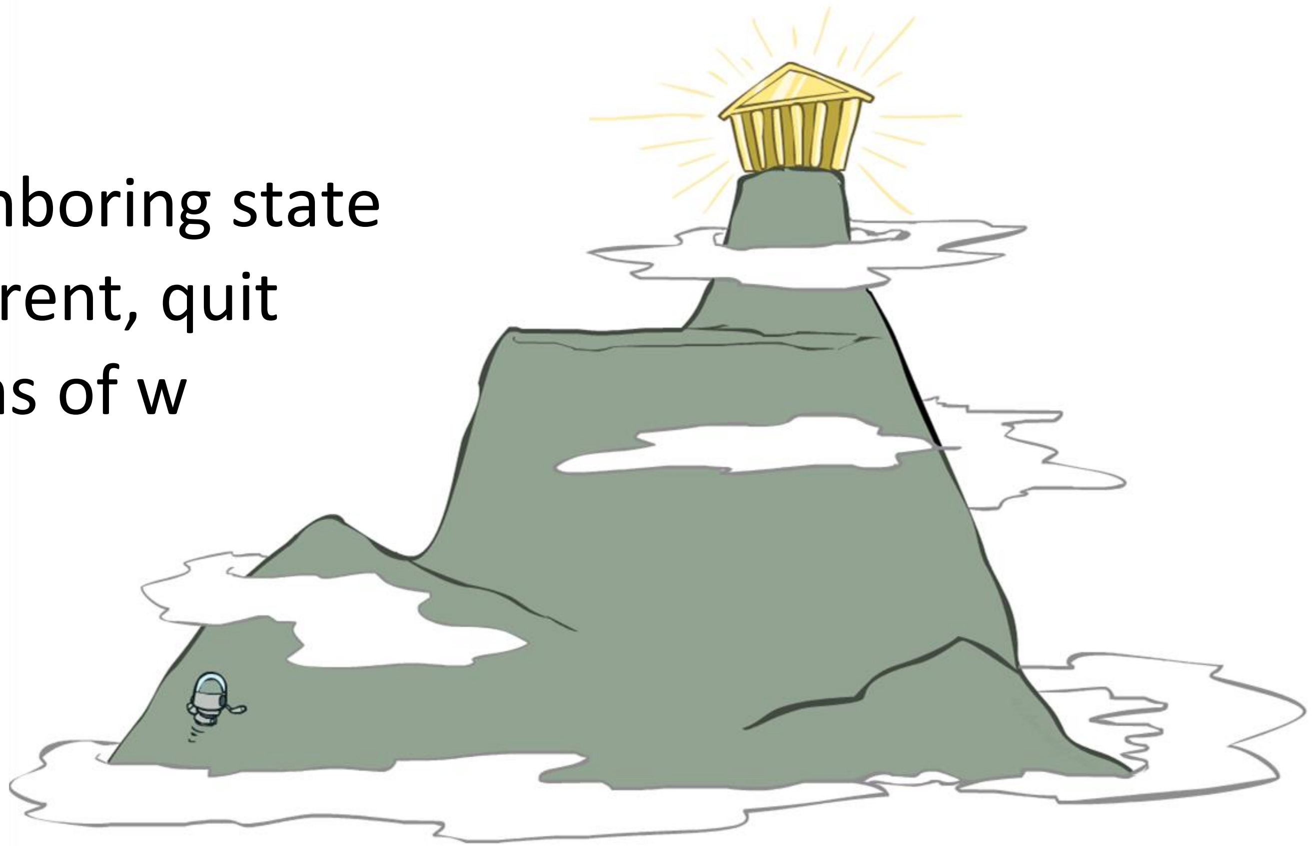
$$w = \begin{pmatrix} W_{11} \\ W_{21} \\ W_{31} \\ W_{12} \\ W_{22} \\ W_{32} \\ W_{13} \\ W_{23} \\ W_{33} \\ W_1 \\ W_2 \\ W_3 \end{pmatrix}$$

- How?
  - Basic idea: hill climbing (gradient descent)

Hill Climbing (gradient descent)

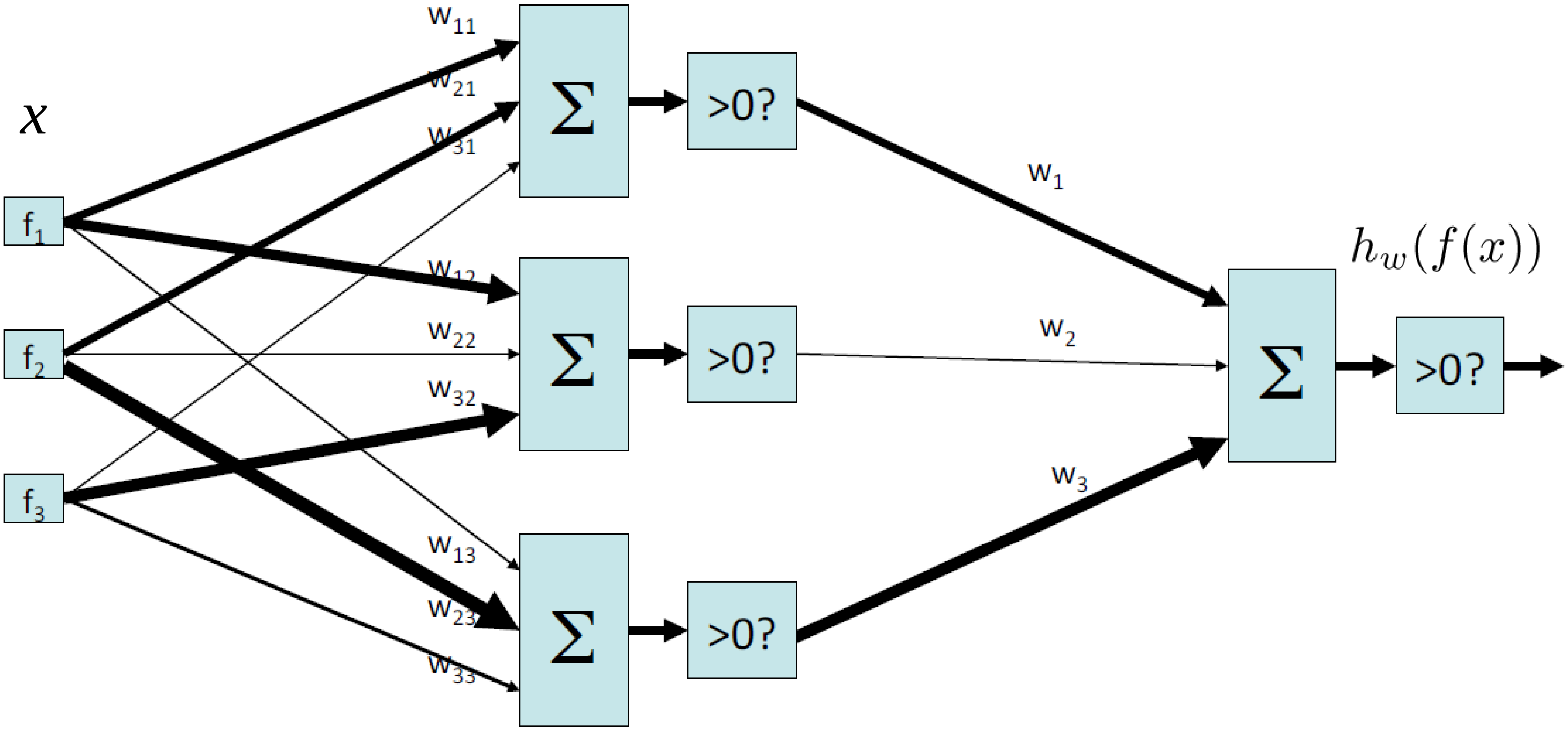
Simple, general idea:

- Start wherever
- Repeat: move to the best neighboring state
- If no neighbors better than current, quit
- Neighbors = small perturbations of w

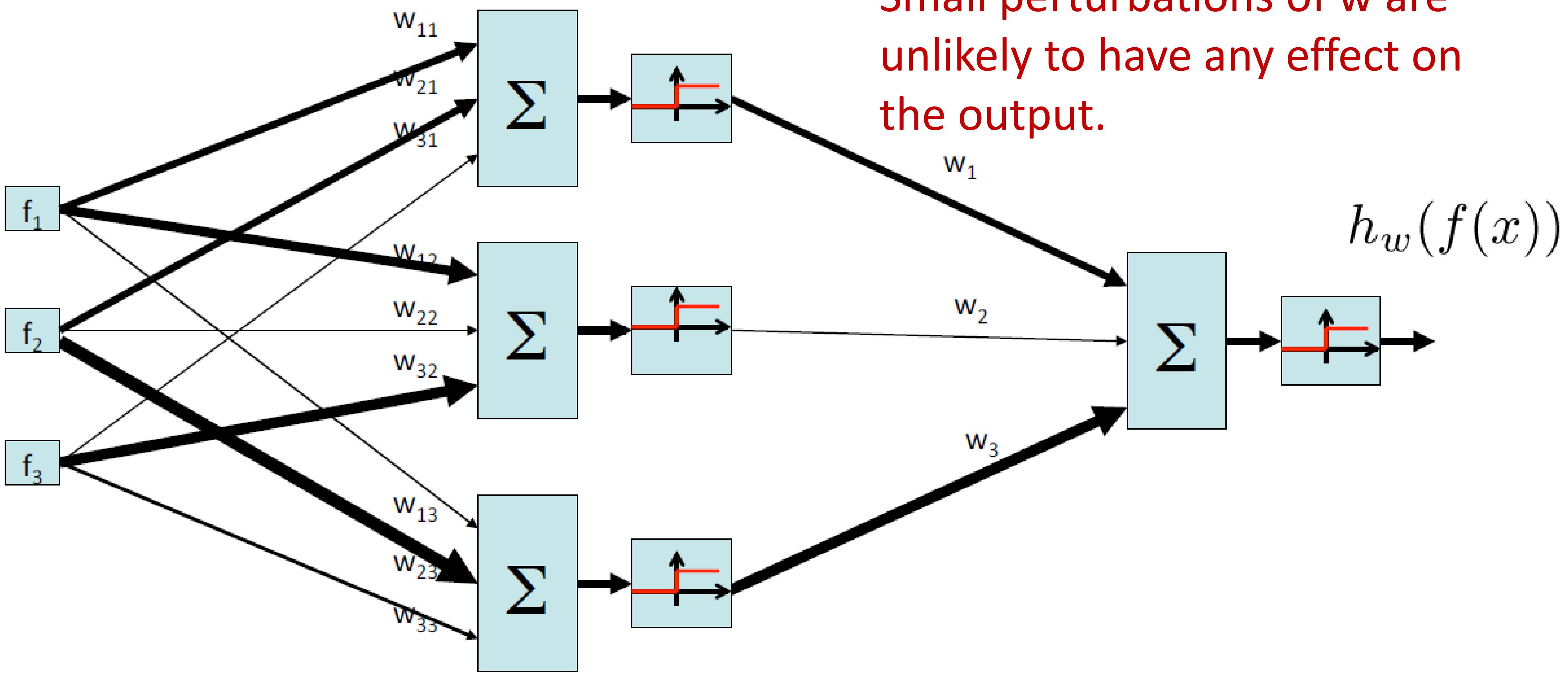


What's tricky when hill-climbing for the multi-layer perceptron?

Two-Layer Perceptron Network



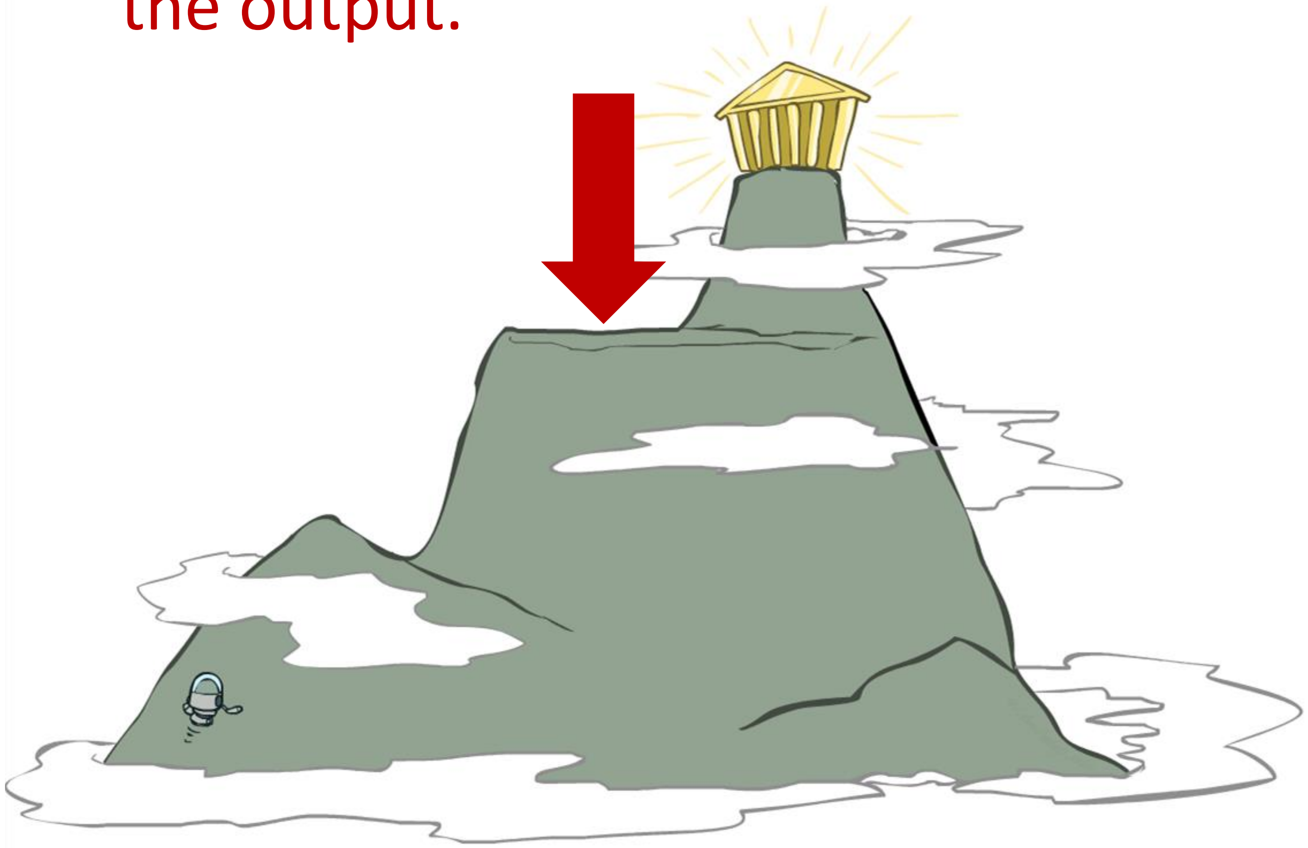
Two-Layer Perceptron Network



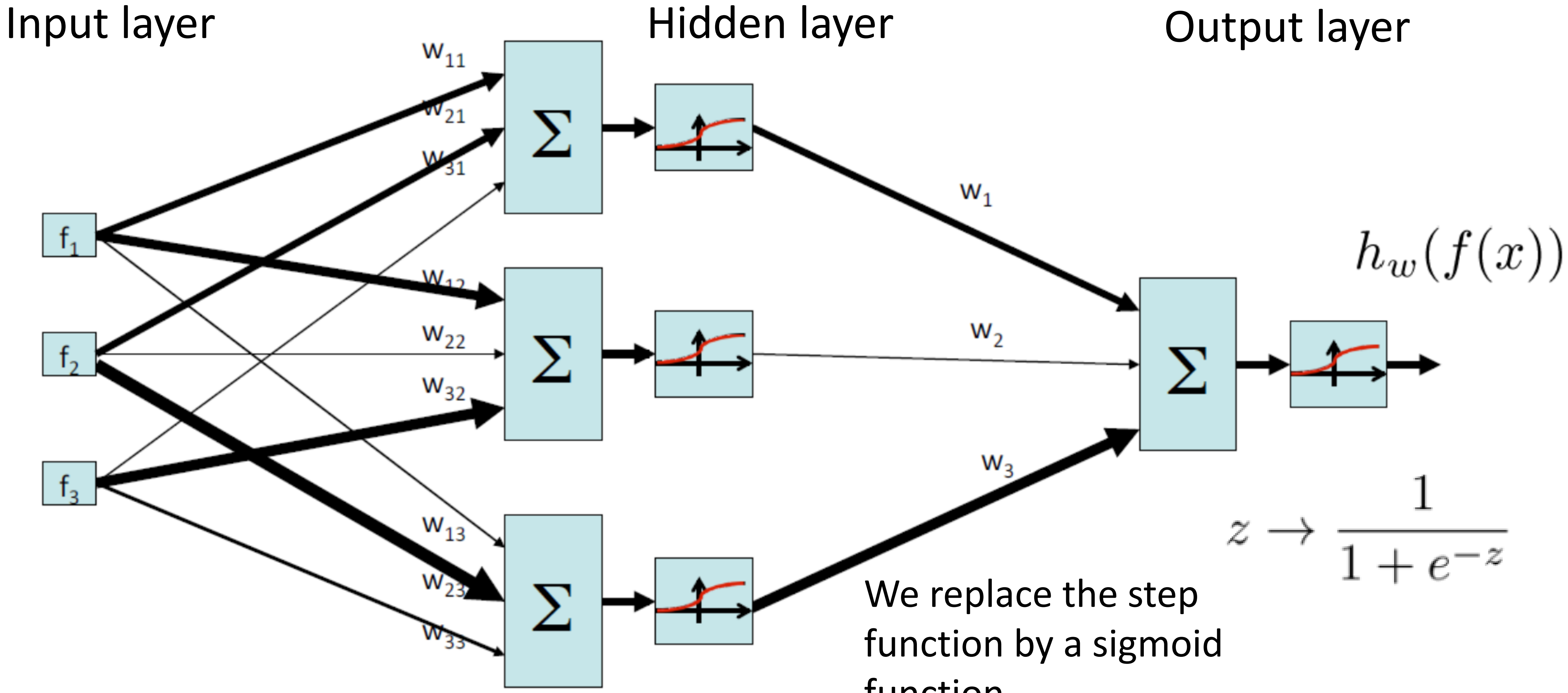
Small perturbations of w are unlikely to have any effect on the output.

Two-Layer Perceptron Network

Small perturbations of w are unlikely to have any effect on the output.



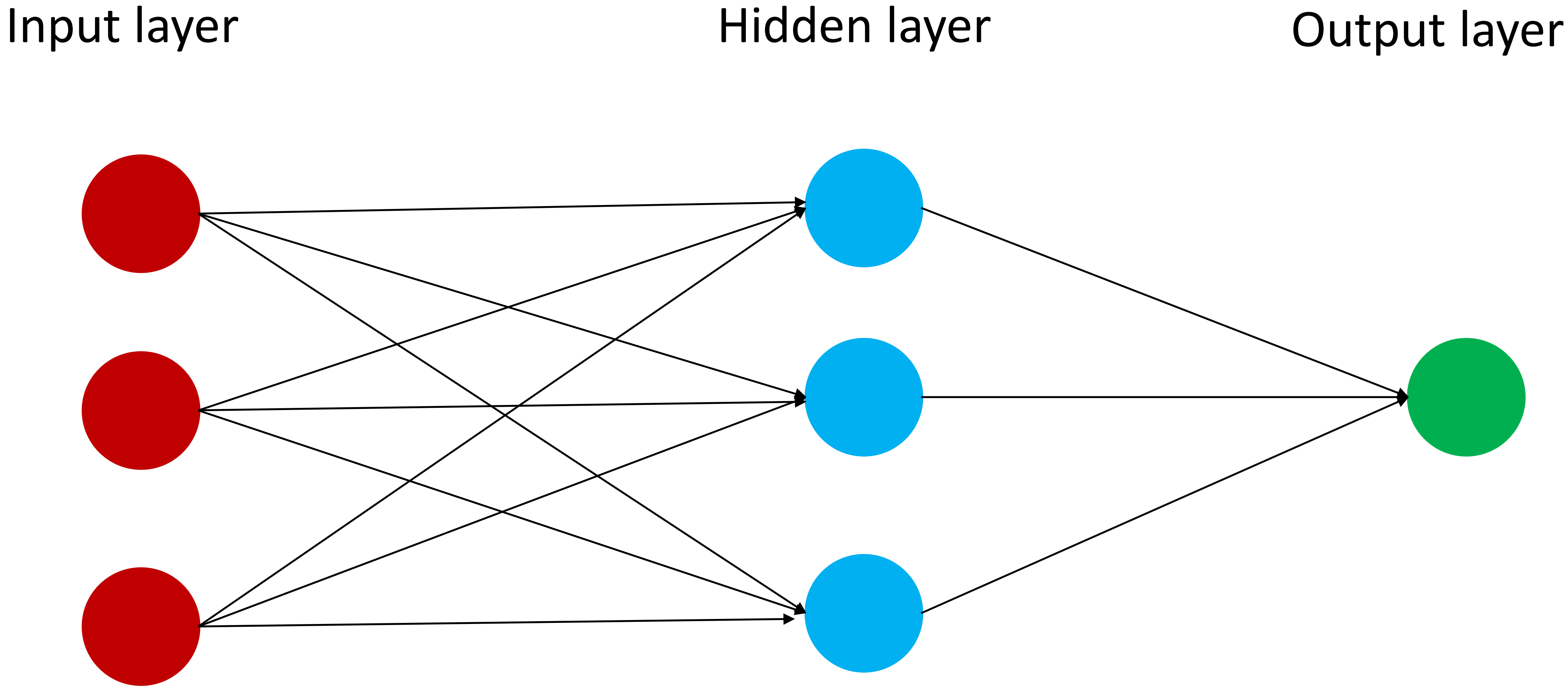
Two-Layer Neural Network



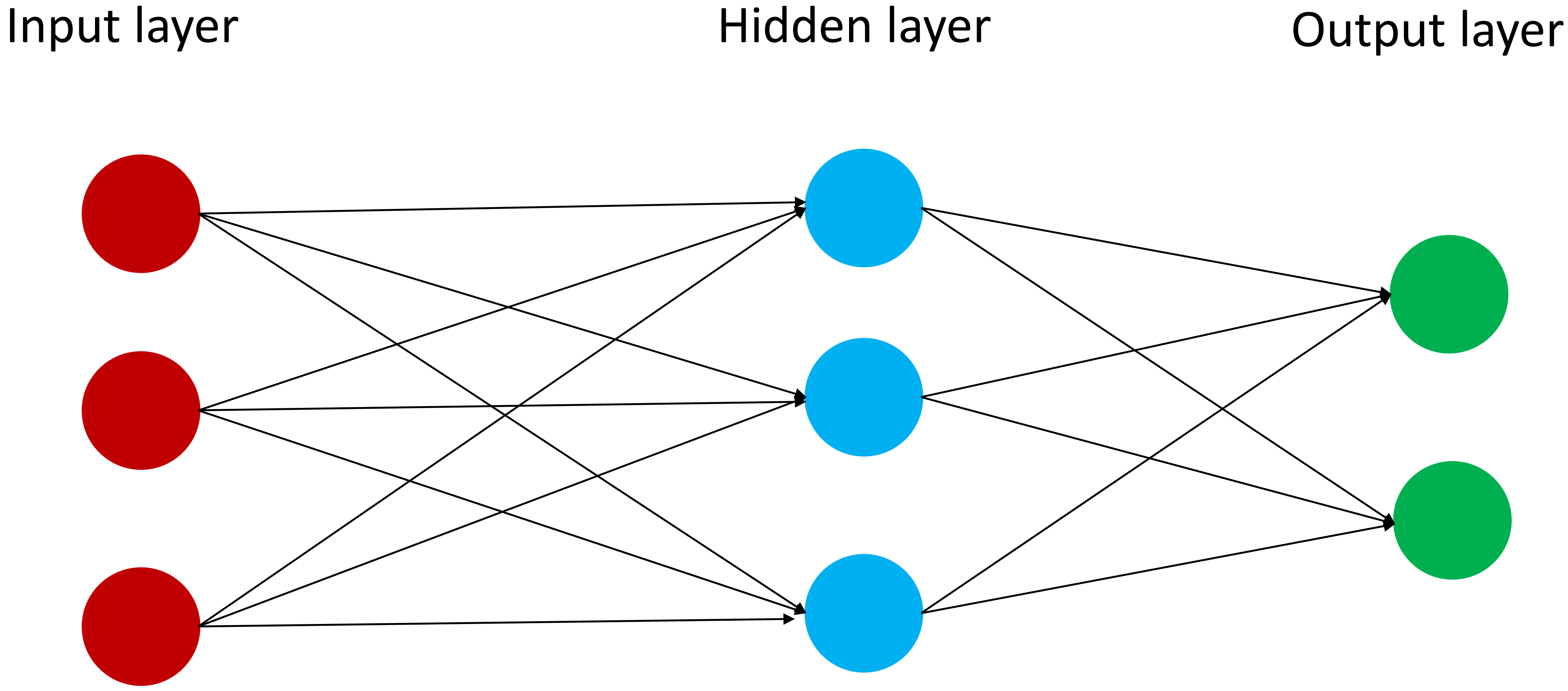
$$z \rightarrow \frac{1}{1+e^{-z}}$$

We replace the step function by a sigmoid function.

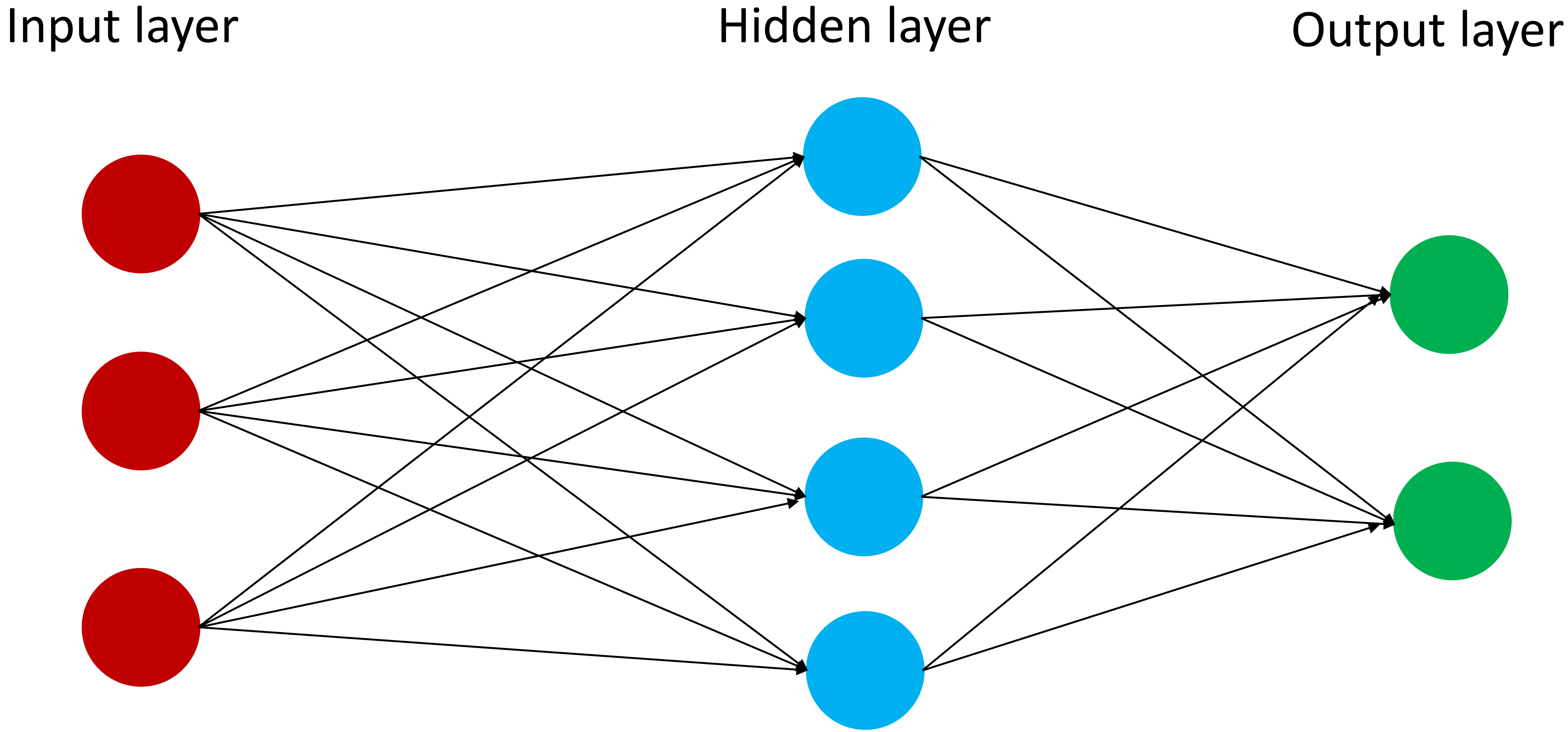
Two-Layer Neural Network



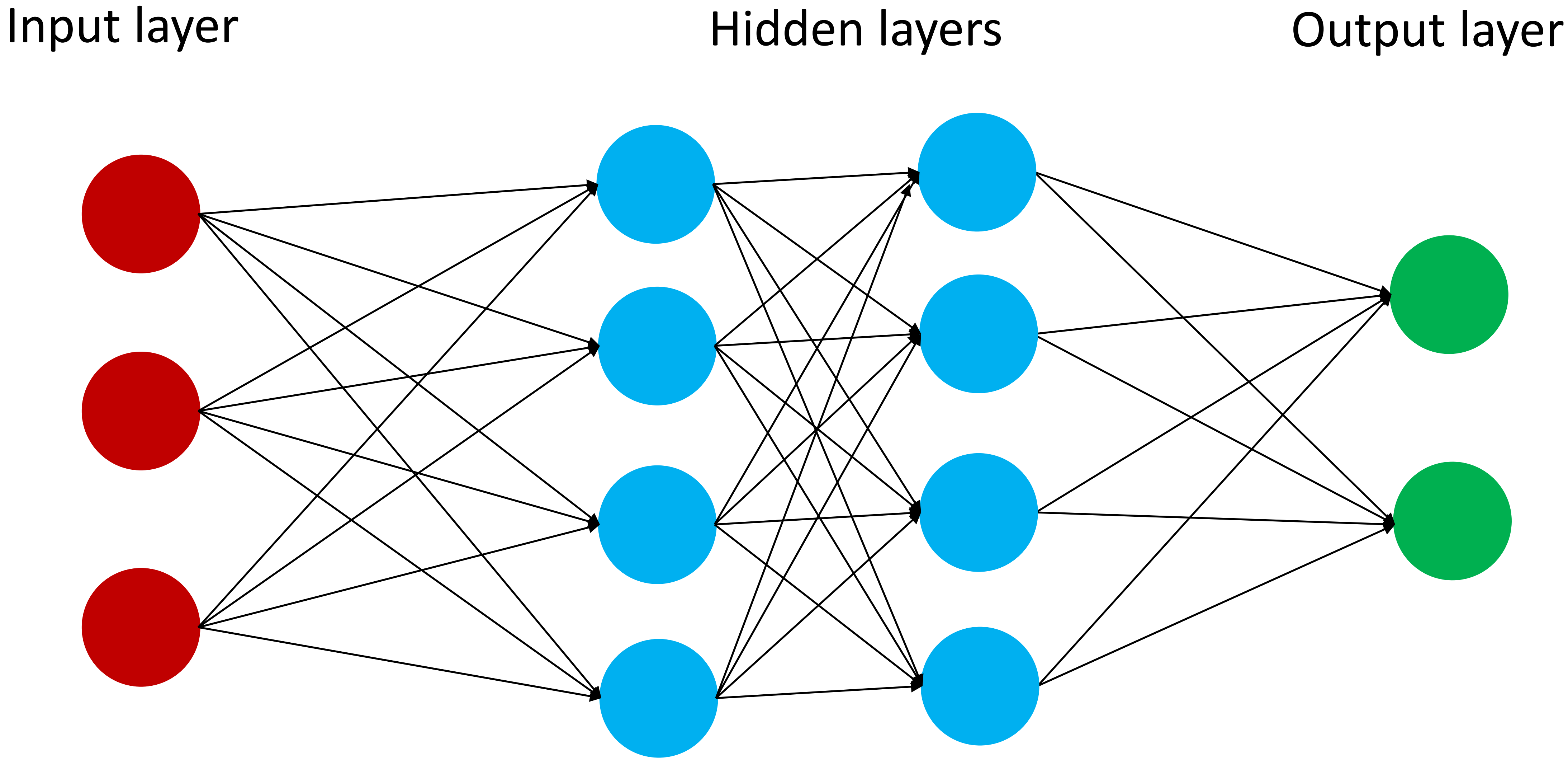
Two-Layer Neural Network



Two-Layer Neural Network



Deep Neural Network



Neural Network Properties

- Theorem (Universal Function Approximators):

  A two-layer neural network with a sufficient number of neurons can approximate any continuous function to any desired accuracy.

- Practical Considerations
  - Can be seen as learning the features
  - Large number of neurons: danger for overfitting
  - Hill-climbing procedure can get stuck in bad local optima