Lecture 14b: Neural Network Learning

CSCI 360

Introduction to Artificial Intelligence

USC

Urban legend (likely)

https://www.jefftk.com/p/detecting-tanks

Once upon a time, the US Army wanted to use neural networks to automatically detect camouflaged enemy tanks. The researchers trained a neural net on 50 photos of camouflaged tanks in trees, and 50 photos of trees without tanks. Using standard techniques for supervised learning, the researchers trained the neural network to a weighting that correctly loaded the training set—output "yes" for the 50 photos of camouflaged tanks, and output "no" for the 50 photos of forest. This did not ensure, or even imply, that *new* examples would be classified correctly. The neural network might have "learned" 100 special cases that would not generalize to any new problem. Wisely, the researchers had originally taken 200 photos, 100 photos of tanks and 100 photos of trees. They had used only 50 of each for the training set. The researchers ran the neural network on the remaining 100 photos, and without further training the neural network classified all remaining photos correctly. Success confirmed! The researchers handed the finished work to the Pentagon, which soon handed it back, complaining that in their own tests the neural network did no better than chance at discriminating photos.

Why?

Urban legend (likely)

https://www.jefftk.com/p/detecting-tanks

Once upon a time, the US Army wanted to use neural networks to automatically detect camouflaged enemy tanks. The researchers trained a neural net on 50 photos of camouflaged tanks in trees, and 50 photos of trees without tanks. Using standard techniques for supervised learning, the researchers trained the neural network to a weighting that correctly loaded the training set—output "yes" for the 50 photos of camouflaged tanks, and output "no" for the 50 photos of forest. This did not ensure, or even imply, that *new* examples would be classified correctly. The neural network might have "learned" 100 special cases that would not generalize to any new problem. Wisely, the researchers had originally taken 200 photos, 100 photos of tanks and 100 photos of trees. They had used only 50 of each for the training set. The researchers ran the neural network on the remaining 100 photos, and without further training the neural network classified all remaining photos correctly. Success confirmed! The researchers handed the finished work to the Pentagon, which soon handed it back, complaining that in their own tests the neural network did no better than chance at discriminating photos.

It turned out that in the researchers' dataset, photos of camouflaged tanks had been taken on cloudy days, while photos of plain forest had been taken on sunny days. The neural network had learned to distinguish cloudy days from sunny days, instead of distinguishing camouflaged tanks from empty forest.

Here is where we are…

	3/1		Project 2 Out	
9	3/4 3/6	3/5 3/7	Quantifying Uncertainty Bayesian Networks	[Ch 13.1-13.6] [Ch 14.1-14.2]
10	3/11 3/13	3/12 3/14	*(spring break, no class)* *(spring break, no class)*	
11	3/18 3/20	3/19 3/21	Inference in Bayesian Networks Decision Theory	[Ch 14.3-14.4] [Ch 16.1-16.3 and 16.5]
	3/23		Project 2 Due	
12	3/25 3/27	3/26 3/28	***Advanced topics*** *(Chao traveling to National Science Foundation)* ***Advanced topics*** *(Chao traveling to National Science Foundation)*	
	3/29		Homework 2 Out	
13	4/1 4/3	4/2 4/4	Markov Decision Processes Decision Tree Learning	[Ch 17.1-17.2] [Ch 18.1-18.3]
	4/5 4/5		Homework 2 Due Project 3 Out	
14	4/8 4/10	4/9 4/11	Perceptron Learning Neural Network Learning	[Ch 18.6] [Ch 18.7]
15	4/15 4/17	4/16 4/18	Statistical Learning Reinforcement Learning	[Ch 20.2.1-20.2.2] [Ch 21.1-21.2]
16	4/22 4/24	4/23 4/25	Artificial Intelligence Ethics Wrap-Up and Final Review	
	4/26		Project 3 Due	
	5/3	5/2	**Final Exam** (2pm-4pm)	

Outline

- What is AI?
- Part I: Search
- Part II: Logical reasoning
- Part III: Probabilistic reasoning
- **Part IV: Machine learning**
  - Decision Tree Learning
  - Perceptron Learning
  - **Neural Network Learning**
  - Statistical Learning
  - Reinforcement Learning

Recap: *Forms of learning – Feedback to learn from*

- **Unsupervised** learning
  - Learn “patterns in the input” without explicit feedback

- **Supervised** learning
  - Given example input-output pairs, learn an input-output function
  - Decision Tree / Regression / Classification

- **Reinforcement** learning
  - Learn from reinforcements (rewards or punishments)

Recap: *Forms of learning – Feedback to learn from*

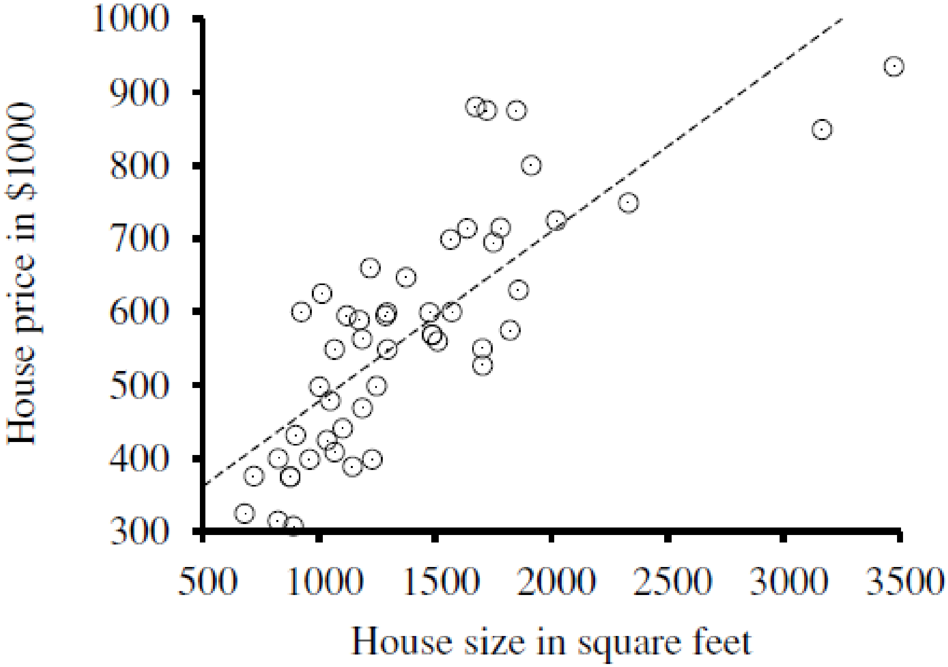
- **Unsupervised** learning
  - Learn “patterns in the input” without explicit feedback
- **Supervised** learning
  - Given example input-output pairs, learn an input-output function
  - Decision Tree / Regression / Classification
- **Reinforcement** learning
  - Learn from reinforcements (rewards or punishments)

Recap: *Linear regression*

- Finding $h_w(x)$ that best fits the training data { (x, $f(x)$) }

$$h_{\mathbf{w}}(x) = w_1 x + w_0$$

Find the values of $[w_0, w_1]$ that minimize empirical loss



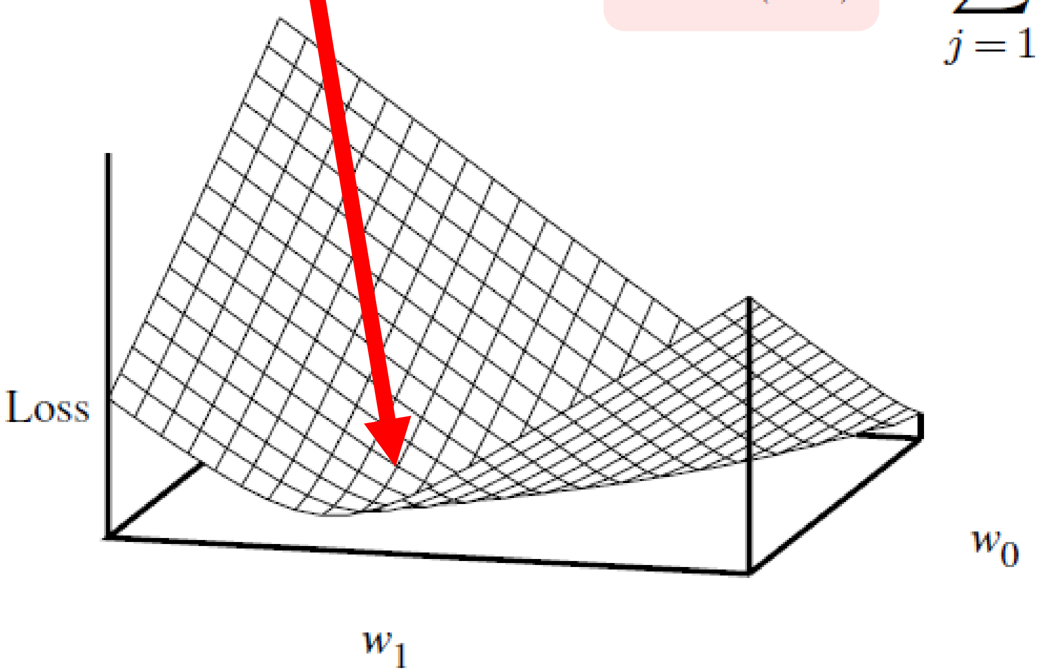
$$\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} Loss(h_{\mathbf{w}}).$$

$$Loss(h_{\mathbf{w}}) = \sum_{j=1}^{N} L_2(y_j, h_{\mathbf{w}}(x_j)) = \sum_{j=1}^{N} (y_j - h_{\mathbf{w}}(x_j))^2$$

Recap: *Minimizing square loss (L_2)*

- **Gauss**: If the *f(x)* values have normally distributed noise, the most likely values of w_1 and w_0 can be obtained by minimizing the **sum of the squares of the errors**

$$\boxed{Loss(h_{\mathbf{w}})} = \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$



"Gradient descent"

Recap: *Computing the partial derivative*

$$\frac{\partial}{\partial w_0} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

$$\frac{\partial}{\partial w_1} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

Recap: *Computing the partial derivative*

$$\frac{\partial}{\partial w_0} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

$$\frac{\partial}{\partial w_1} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

$$\begin{aligned} \frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x))^2 \\ &= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x)) \\ &= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - (w_1 x + w_0)) \end{aligned}$$

$$\frac{\partial}{\partial w_0} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x))$$

Recap: *Computing the partial derivative*

$$\frac{\partial}{\partial w_0}\sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2$$

$$\frac{\partial}{\partial w_1}\sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2$$

$$\begin{aligned}
\frac{\partial}{\partial w_i}Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x))^2 \\
&= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x)) \\
&= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - (w_1 x + w_0))
\end{aligned}$$

$$\frac{\partial}{\partial w_0}Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x))$$

$$\frac{\partial}{\partial w_1}Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \times x$$

Recap: *Minimizing L_2 – gradient descent*

- General search technique: a **hill-climbing** algorithm that follows the **gradient** of the function to be optimized
  - **Initialization**: choose any starting point (w_0, w_1)
  - **Iteration**: move to a neighboring point that is downhill

$\mathbf{w} \leftarrow$ any point in the parameter space

loop until convergence **do**

  for each w_i **in w do**

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$

(**α**) is **step size** (or **learning rate**)

$$w_0 \leftarrow w_0 + \alpha \left(y - h_{\mathbf{w}}(x)\right);$$

$$w_1 \leftarrow w_1 + \alpha \left(y - h_{\mathbf{w}}(x)\right) \times x$$

Recap: *Minimizing L_2 – gradient descent (multiple training examples)*

$$Loss(h_{\mathbf{w}}) = \sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2$$

- General search technique: a **hill-climbing** algorithm that follows the **gradient** of the function to be optimized
  - **Initialization**: choose any starting point (w_0, w_1)
  - **Iteration**: move to a neighboring point that is downhill

w $\leftarrow$ any point in the parameter space
loop until convergence **do**
  for each w_i **in w do**
$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$

For N training examples, the **batch gradient descent** converges to the unique global minimum, but is very slow

Stochastic gradient descent, which considers only a single training point at a time, is often faster, but doesn't guarantee convergence.

$$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j))\,; \quad w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j)) \times x_j\,.$$

Recap: *Forms of learning – Feedback to learn from*

- **Unsupervised** learning
  - Learn “patterns in the input” without explicit feedback

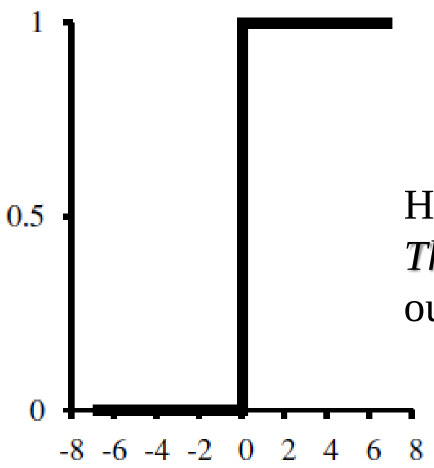
- **Supervised** learning
  - Given example input-output pairs, learn an input-output function
  - Decision Tree / Regression / Classification

- **Reinforcement** learning
  - Learn from reinforcements (rewards or punishments)

Recap: *Classifier $h_w(x)$*

- The classifier is meant to return either **1 (true)** or **0 (false)**
  - *Think of it as the result of passing the linear function (w*x) through a threshold function*

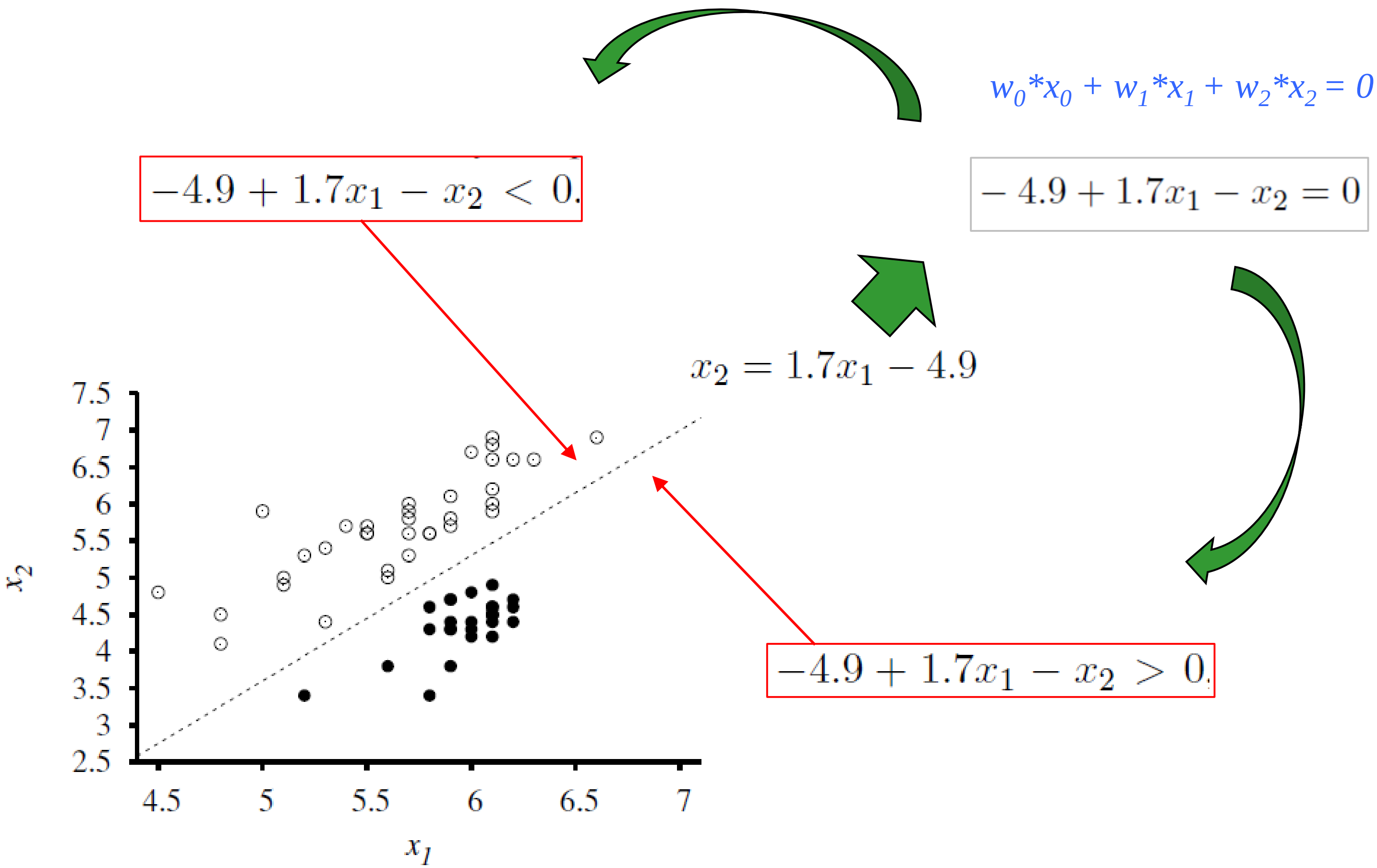
$$\boxed{h_{\mathbf{w}}(\mathbf{x}) = Threshold(\mathbf{w} \cdot \mathbf{x})} \text{ where } Threshold(z) = 1 \text{ if } z \geq 0 \text{ and } 0 \text{ otherwise.}$$



Hard threshold function *Threshold(z)* with 0/1 output

Recap: *Linear classifiers with a hard threshold*

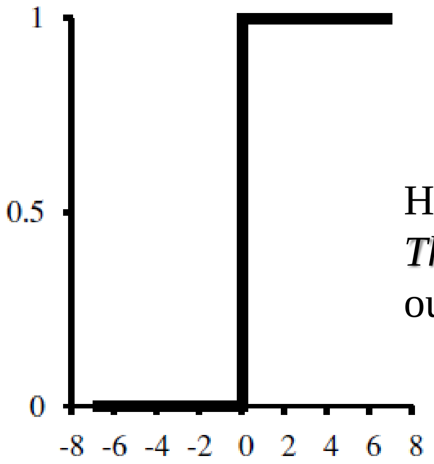
- A **decision boundary** is a line (or surface) that separates the two classes.



Recap: *Learning a classifier $h_w(x)$*

- The classifier is meant to return either **1 (true)** or **0 (false)**
  - *Think of it as the result of passing the linear function (**w***x) through a threshold function*

$$\boxed{h_{\mathbf{w}}(\mathbf{x}) = Threshold(\mathbf{w} \cdot \mathbf{x})} \text{ where } Threshold(z) = 1 \text{ if } z \geq 0 \text{ and } 0 \text{ otherwise.}$$



Hard threshold function *Threshold(z)* with 0/1 output

Perceptron learning rule:

$$\boxed{w_i \leftarrow w_i + \alpha\,(y - h_{\mathbf{w}}(\mathbf{x})) \times x_i}$$

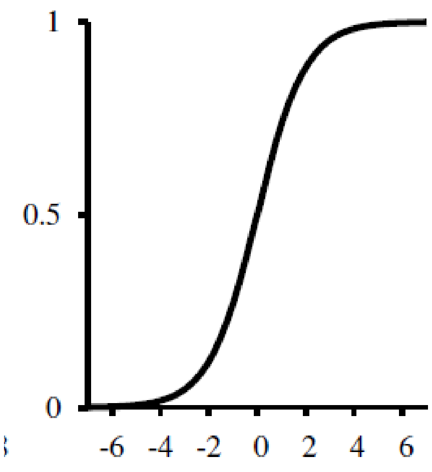
- **0** if the output is correct: **no update**
- **+1** if y is 1 but $h_w(x)$ is 0: **increase w_i**
- **-1** if y is 0 but $h_w(x)$ is 1: **decrease w_i**

Recap: *Logistic regression*

- Derivation of the gradient

$$\begin{aligned}\frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x}))^2 \\ &= 2(y - h_{\mathbf{w}}(\mathbf{x})) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x}))\end{aligned}$$

$$\boxed{w_i \leftarrow w_i + \alpha\,(y - h_{\mathbf{w}}(\mathbf{x})) \times h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x})) \times x_i}$$

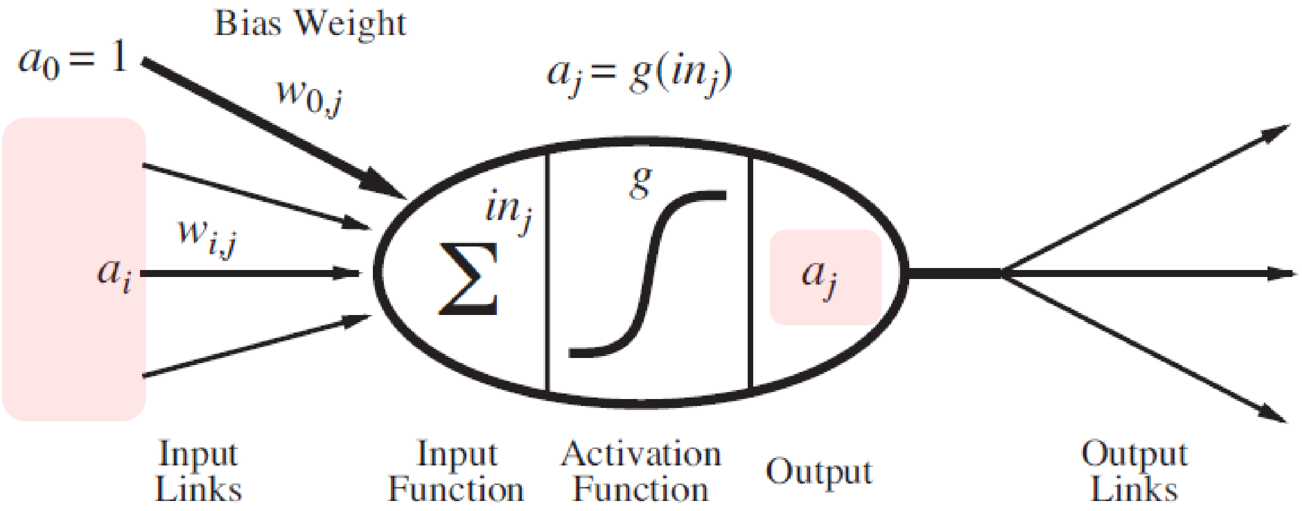


$$\boxed{h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w} \cdot \mathbf{x})} = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}} \,.$$

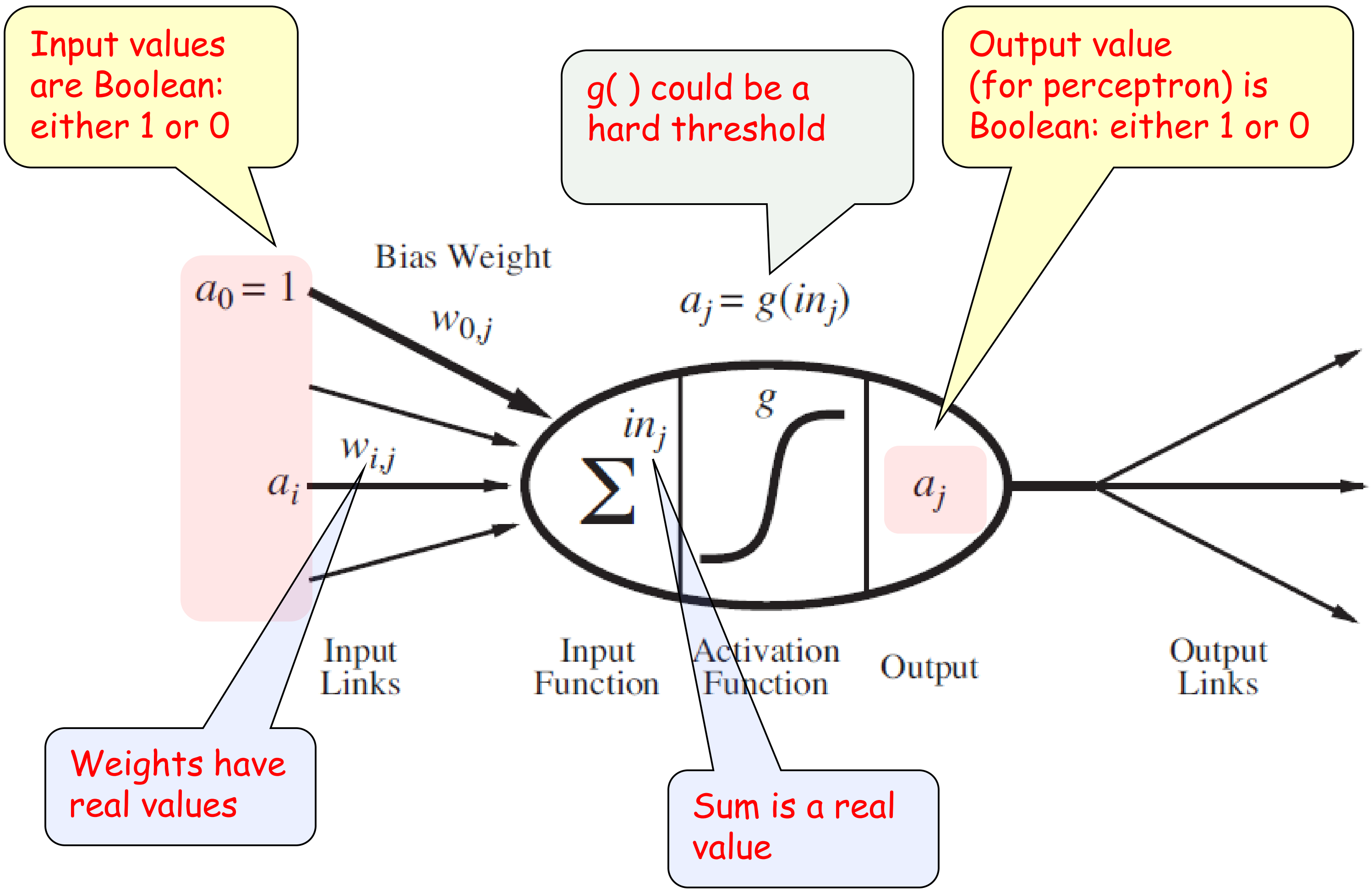
Outline of today's lecture

- **Single-layer neural networks**
- Multi-layer neural networks
- Learning in neural networks
  - Back propagation

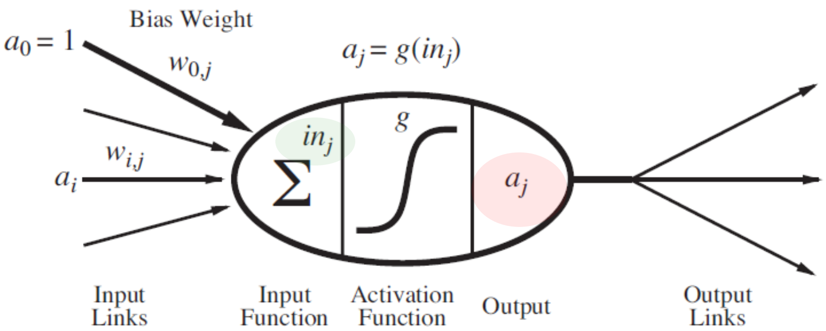
Node (or unit, or neuron)



Node (or unit, or neuron)



Perceptron and sigmoid perceptron



- **Hard threshold** (g)
  Perceptron
- **Logistic function** (g)
  Sigmoid perceptron

Weighted sum

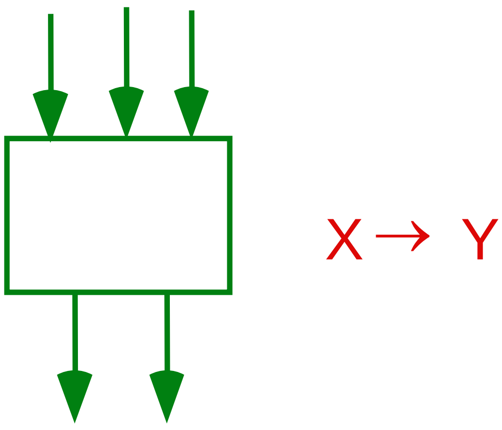
$$in_j = \sum_{i=0}^{n} w_{i,j} a_i$$

Threshold function g( )

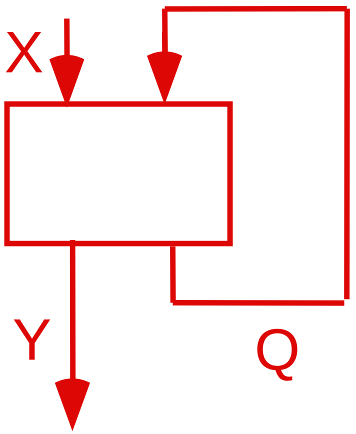
$$a_j = g(in_j) = g\left(\sum_{i=0}^{n} w_{i,j} a_i\right)$$

Neural network structure

- **Feed-forward** network
  - Forms a directed acyclic graph (DAG)
  - Has no internal state



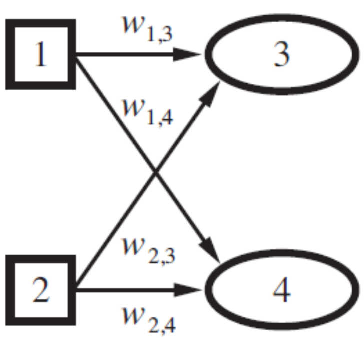
- **Recurrent** network
  - Feeds its output back into its own input
  - Acts as short-term memory



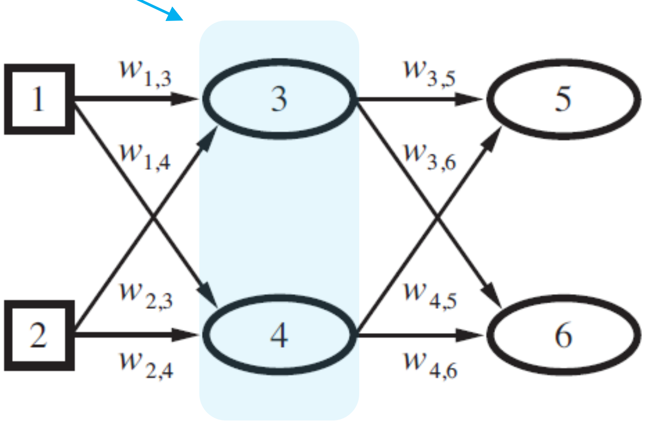
Feed-forward network

- Arranged in layers
  - **Input** layer
  - **Output** layer
  - **Hidden** layer

Perceptron network
(single-layer network)



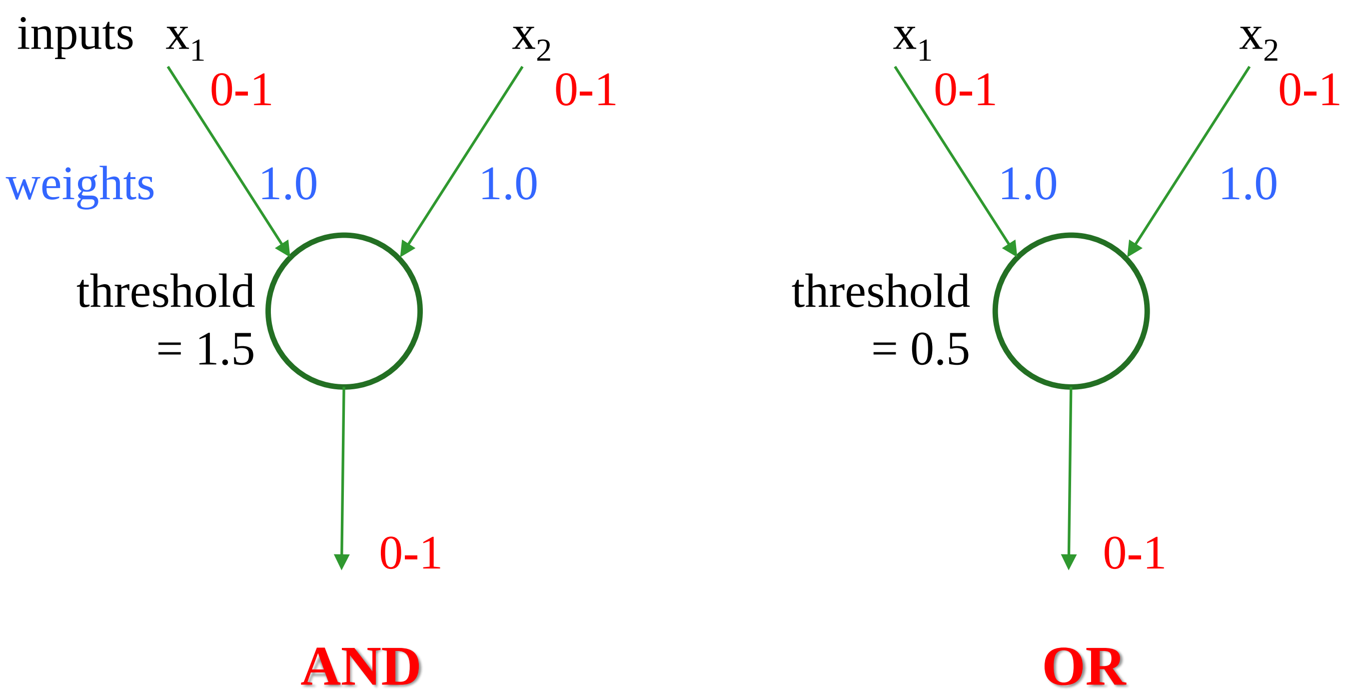
Multi-layer neural network



Single-layer networks *(perceptrons)*

- Can express any linearly separable function

- **Examples**



Expressiveness of perceptrons

- Can **perceptrons** represent all Boolean functions?

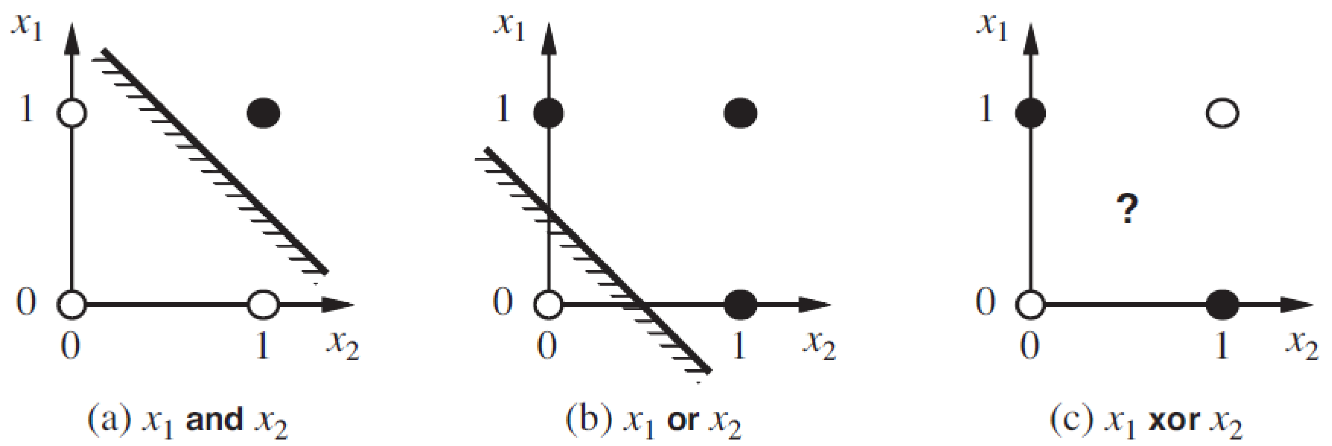
  f (Feature_1, …, Feature_n) ≡ some propositional sentence

Expressiveness of perceptrons

- Can **perceptrons** represent all Boolean functions?

  f (Feature_1, …, Feature_n) ≡ some propositional sentence

- Linearly separable
  - Need to find an **n-dimensional plane** that separates the labeled examples (*true* from *false*)

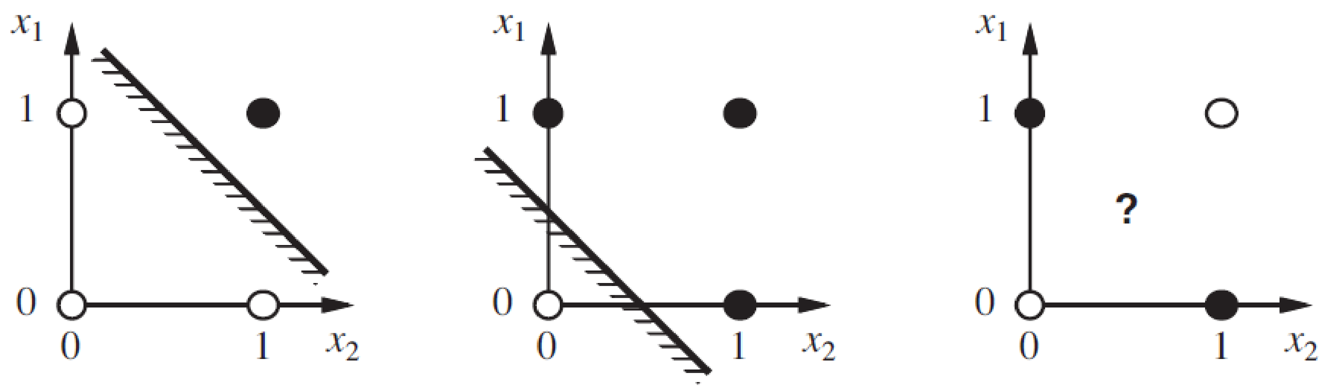


Expressiveness of perceptrons

- Can **perceptrons** represent all Boolean functions? **NO**

  f (Feature_1, …, Feature_n) ≡ some propositional sentence

- An XOR **cannot** be represented with a perceptron (or any single-layer neural network)!



(a) x_1 **and** x_2 (b) x_1 **or** x_2 (c) x_1 **xor** x_2

Expressiveness of perceptrons

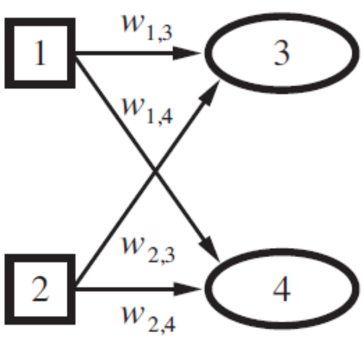
- Can **perceptrons** represent all Boolean functions? **NO**

  f (Feature_1, …, Feature_n) ≡ some propositional sentence

- An XOR **cannot** be represented with a perceptron (or any single-layer neural network)!

This **two-bit adder** function can not be expressed by perceptrons, because (y4) is XOR

x_1	x_2	y_3 (carry)	y_4 (sum)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Expressiveness of perceptrons

- Can **perceptrons** represent all Boolean functions? **NO**

  f (Feature_1, …, Feature_n) ≡ some propositional sentence

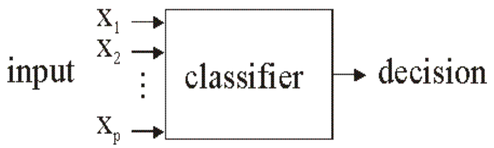
- An XOR **cannot** be represented with a perceptron (or any single-layer neural network)!

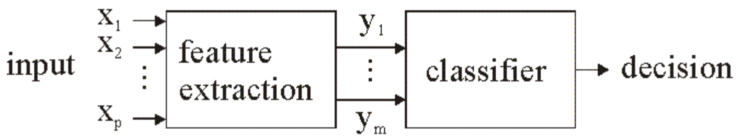
- However, since
  - XOR(x, y) ≡ (x AND NOT y) OR (NOT x AND y)
  - AND, OR and NOT can be represented by perceptrons

- XOR can be expressed in a multi-layer neural network

Outline of today's lecture

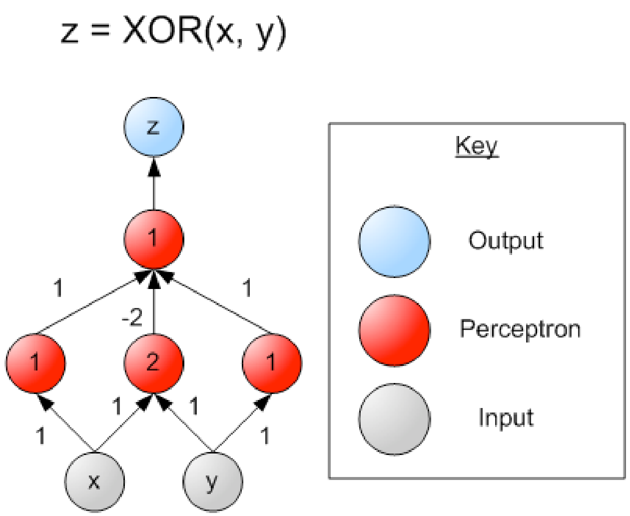
- Single-layer neural networks
  - perceptron networks
- **Multi-layer neural networks**
- Learning in neural networks
  - Back propagation





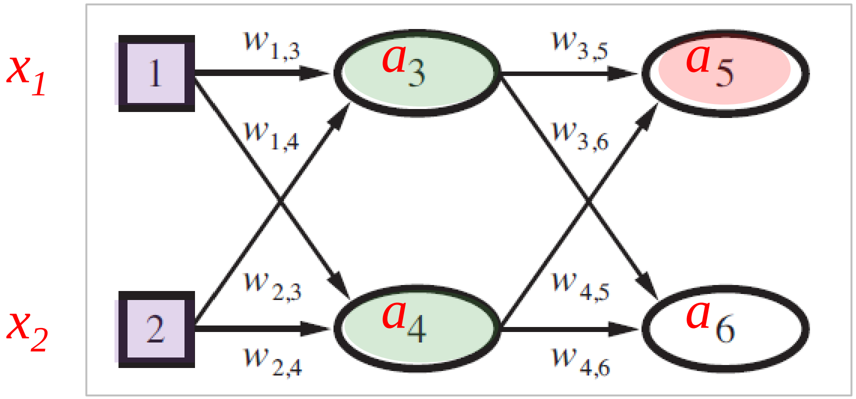
Multi-layer neural network

- Perceptrons can only represent **linear** decision surfaces
- Multi-layer network represent **non-linear** decision surfaces



Multi-layer neural network

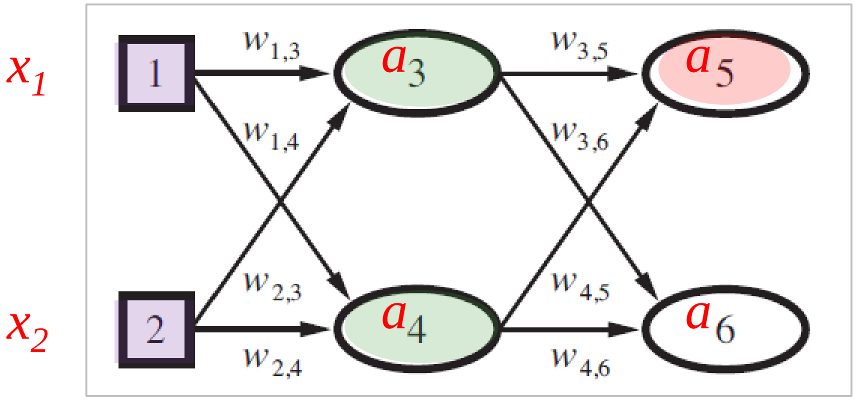
- Can express any Boolean function
  - Can be viewed as a tool for doing **nonlinear regression** when it is equipped with soft threshold functions



$$a_5 = g(w_{0,5,+} w_{3,5}\, a_3 + w_{4,5}\, a_4)$$

Multi-layer neural network

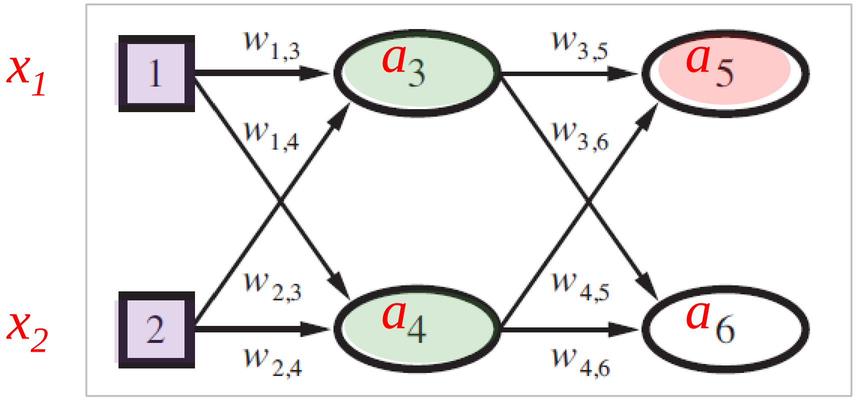
- Can express any Boolean function
  - Can be viewed as a tool for doing **nonlinear regression** when it is equipped with soft threshold functions



$$
\begin{aligned}
a_5 &= g(w_{0,5,}+w_{3,5}\, a_3 + w_{4,5}\, a_4) \\
&= g(w_{0,5,}+w_{3,5}\, g(w_{0,3} + w_{1,3}\, a_1 + w_{2,3}\, a_2) + w_{4,5}\, g(w_0 4 + w_{1,4}\, a_1 + w_{2,4}\, a_2))
\end{aligned}
$$

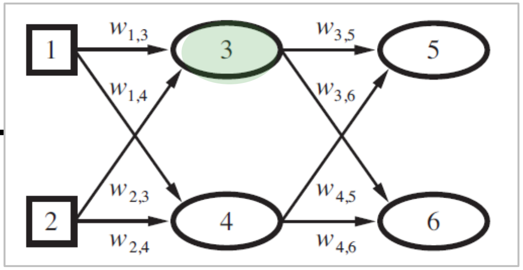
Multi-layer neural network

- Can express any Boolean function
  - Can be viewed as a tool for doing **nonlinear regression** when it is equipped with soft threshold functions

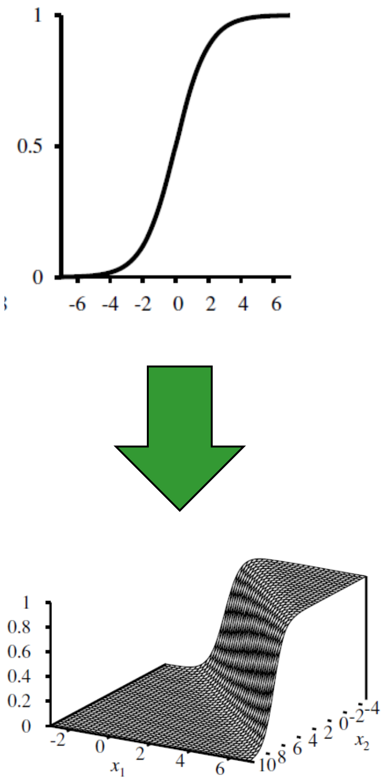


$$\begin{aligned} a_5 &= g(w_{0,5,+} w_{3,5}\, a_3 + w_{4,5}\, a_4) \\ &= g(w_{0,5,+} w_{3,5}\, g(w_{0,3} + w_{1,3}\, a_1 + w_{2,3}\, a_2) + w_{4,5}\, g(w_0 4 + w_{1,4}\, a_1 + w_{2,4}\, a_2)) \\ &= g(w_{0,5,+} w_{3,5}\, g(w_{0,3} + w_{1,3}\, x_1 + w_{2,3}\, x_2) + w_{4,5}\, g(w_0 4 + w_{1,4}\, x_1 + w_{2,4}\, x_2)). \end{aligned}$$

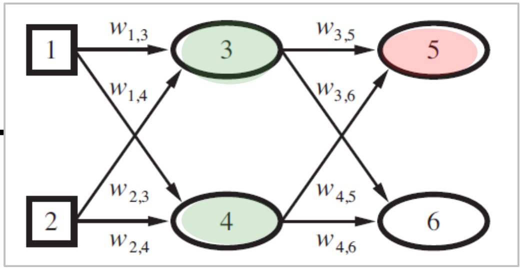
Multi-layer neural network



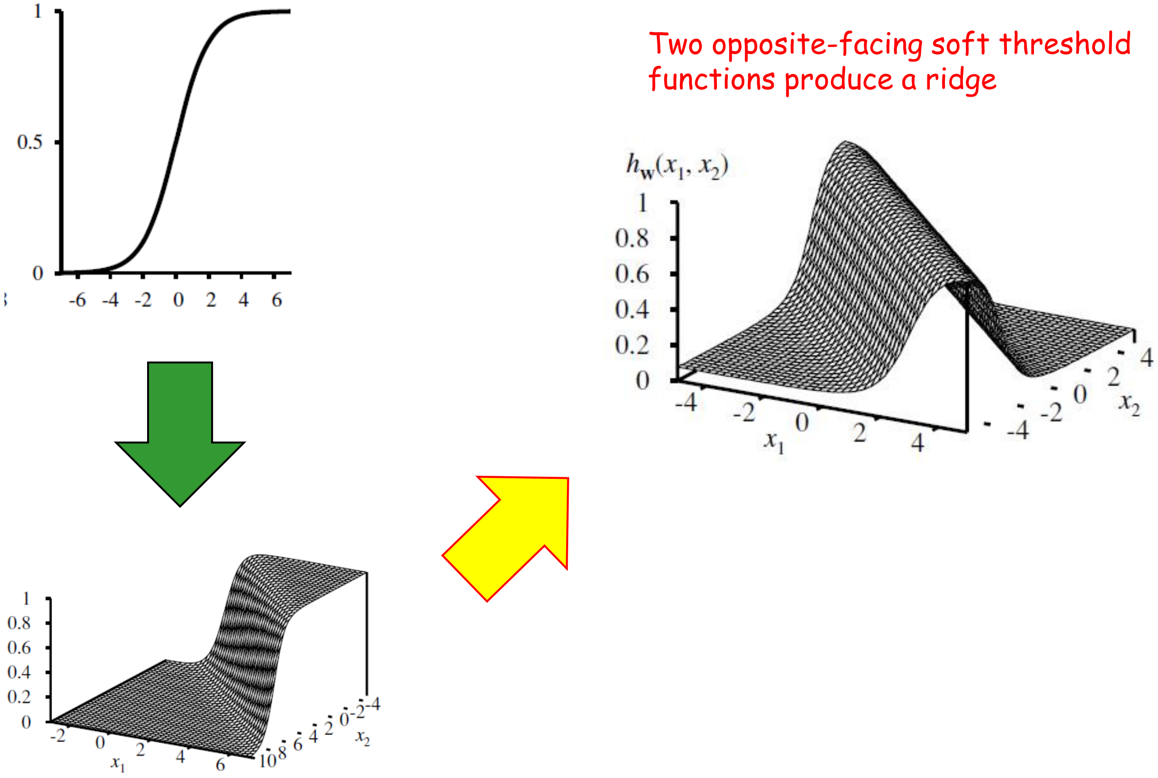
- Can express any Boolean function
  - Can be viewed as a tool for doing **nonlinear regression** when it is equipped with soft threshold functions



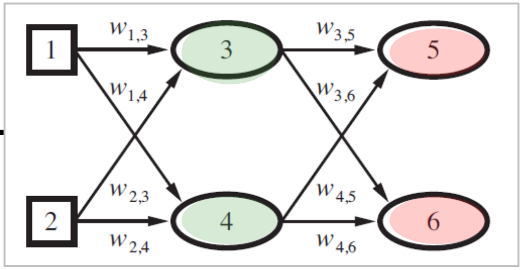
Multi-layer neural network



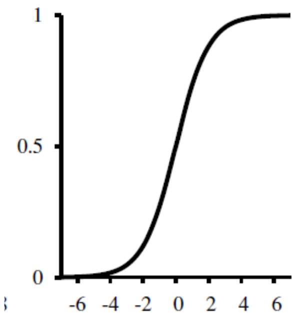
- Can express any Boolean function
  - Can be viewed as a tool for doing **nonlinear regression** when it is equipped with soft threshold functions

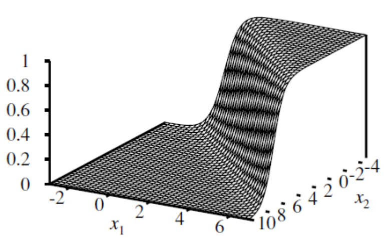


Multi-layer neural network

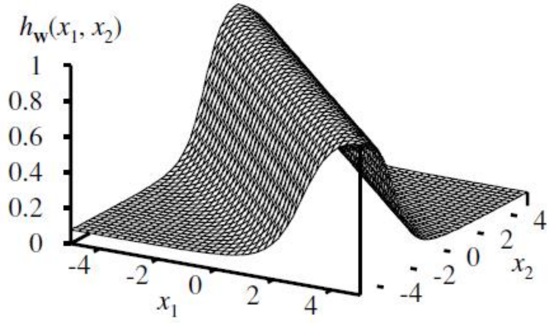


- Can express any Boolean function
  - Can be viewed as a tool for doing **nonlinear regression** when it is equipped with soft threshold functions

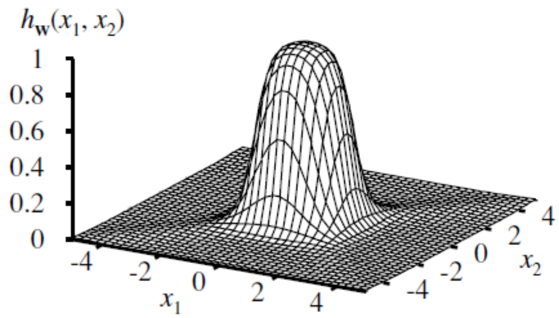




Two opposite-facing soft threshold functions produce a ridge

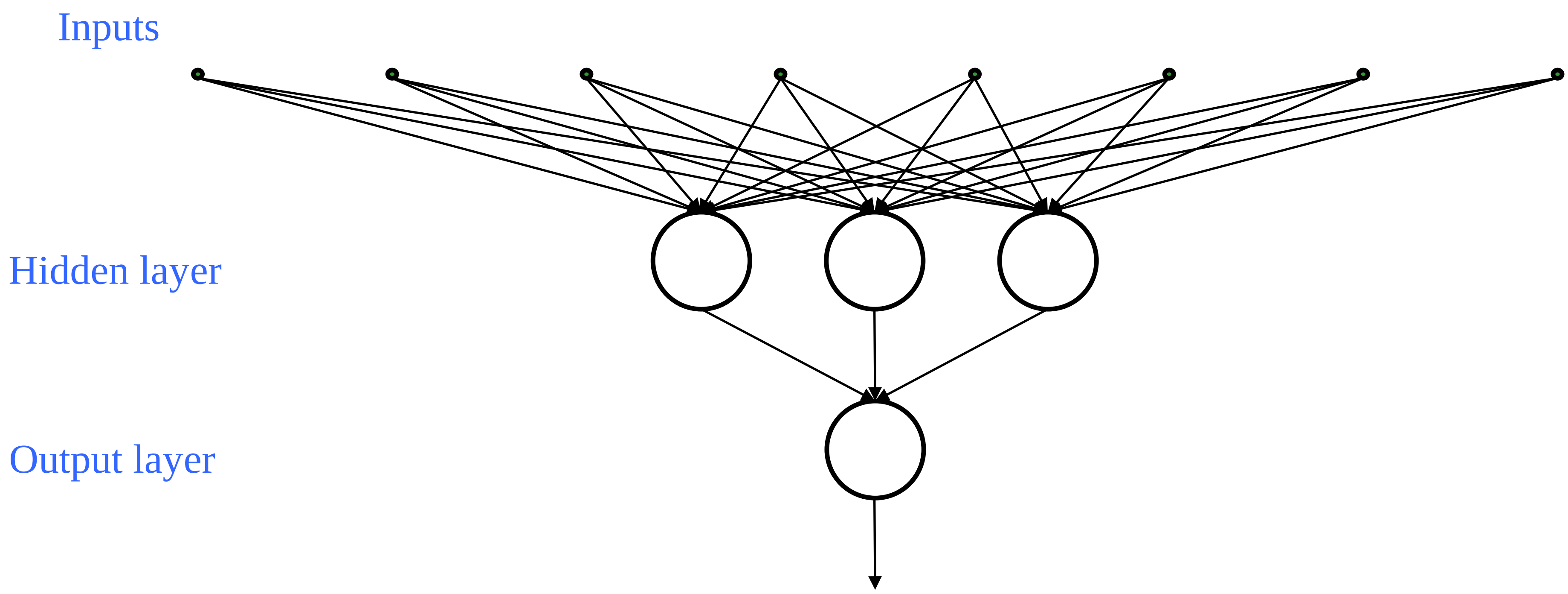


Two opposite-facing ridges produce a bump



Neural network

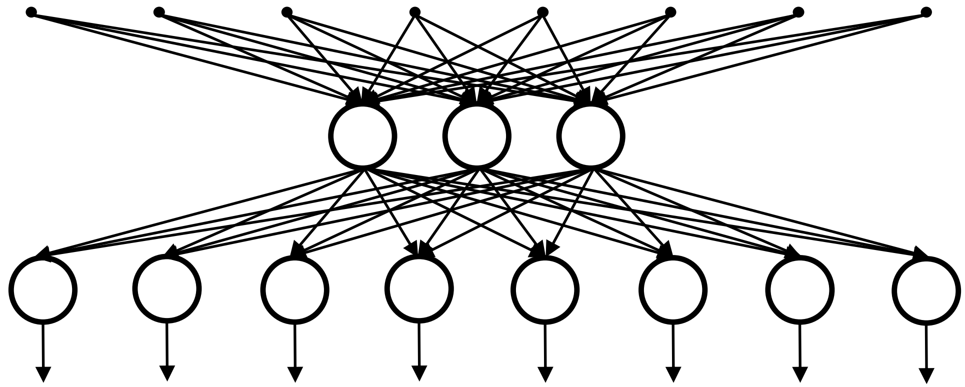
- We use “three-layer” feed-forward networks as network topology.



Neural network learning

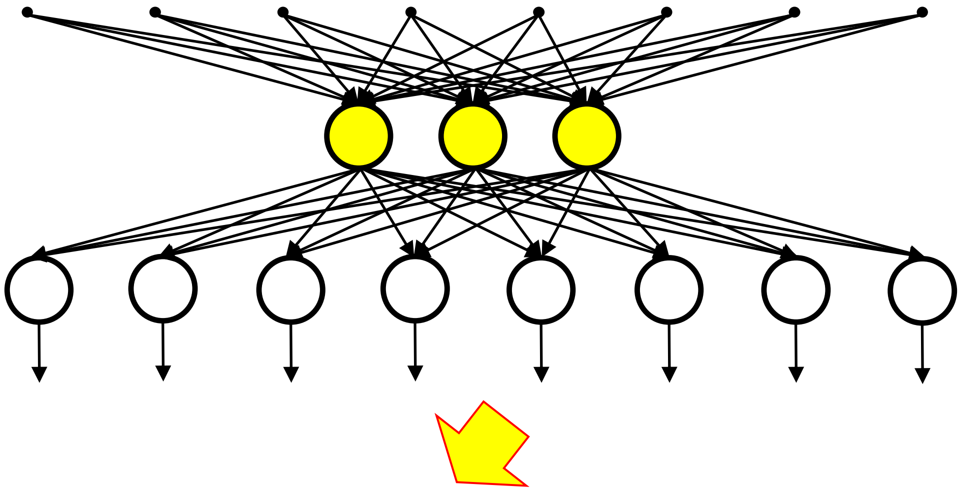
- Given a proper topology, neural networks can discover useful representations **automatically**, but
  - If there are **too few perceptrons** in the hidden layer, it may not learn a function that is consistent with the training examples.
  - If there are **too many perceptrons** in the hidden layer, it may not generalize well, i.e., make few mistakes on the test examples.

Example: Neural Network



input		output
10000000		10000000
01000000		01000000
00100000		00100000
00010000		00010000
00001000		00001000
00000100		00000100
00000010		00000010
00000001		00000001

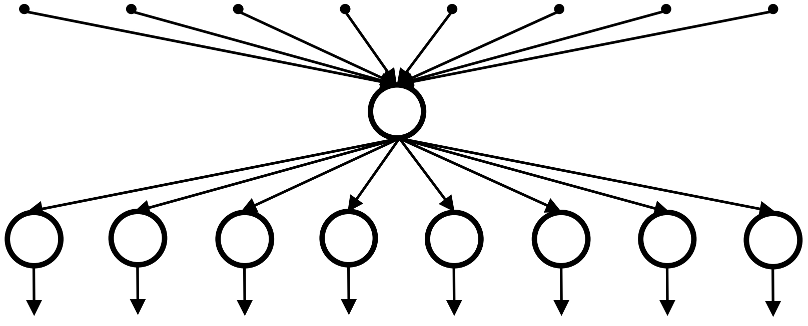
Example: Neural Network



input	hidden values			output
10000000	0.89	0.04	0.08	10000000
01000000	0.15	0.99	0.99	01000000
00100000	0.01	0.97	0.27	00100000
00010000	0.99	0.97	0.71	00010000
00001000	0.03	0.05	0.02	00001000
00000100	0.01	0.11	0.88	00000100
00000010	0.80	0.01	0.98	00000010
00000001	0.60	0.94	0.01	00000001

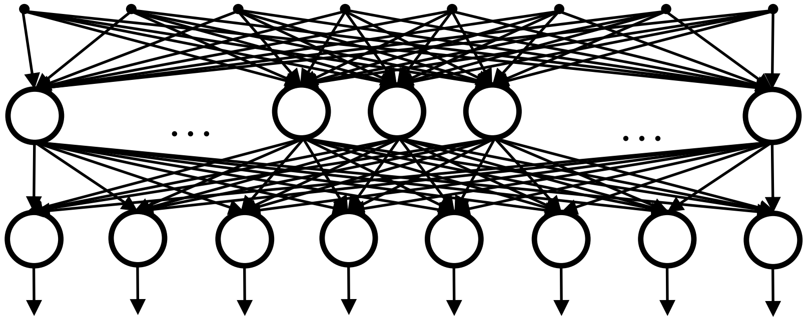
Example: Neural Network

- Too few hidden units (hard to be consistent with training examples)



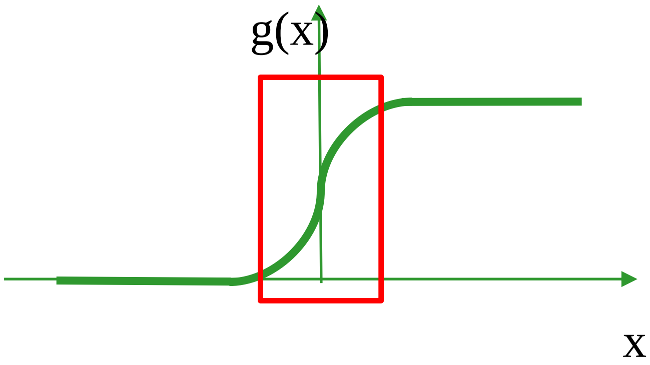
Example: Neural Network

- Too many hidden units (may not generalize well)



Real-valued inputs

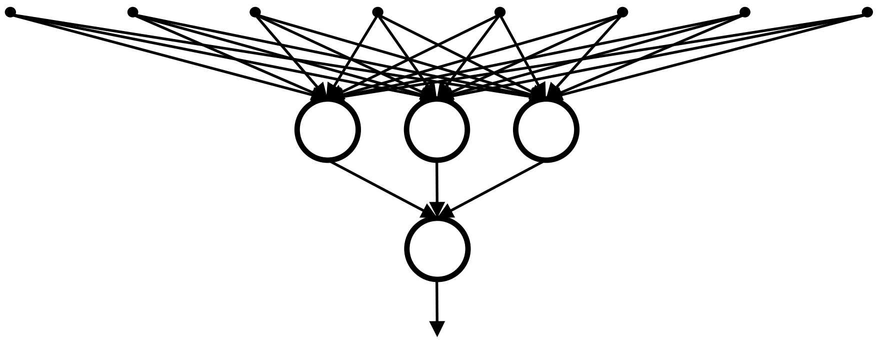
- One can use non-binary inputs. However, avoid operating in the (red) region where small changes in the input cause large changes in the output.
- Rather, use several outputs by using several perceptrons in the output layer.



Example: Neural Network

- Example with real-values inputs and outputs: early autonomous driving

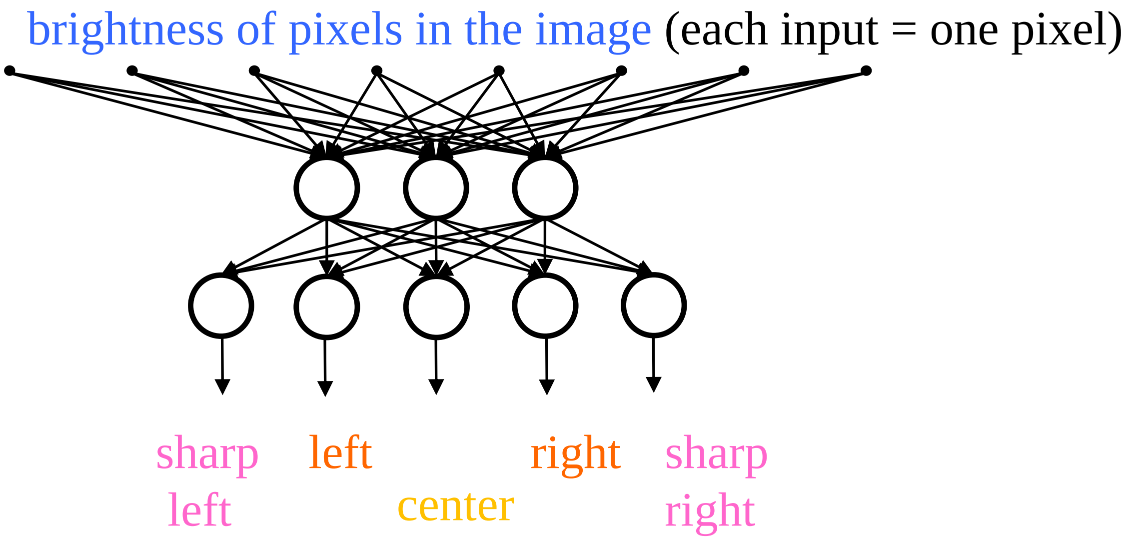
brightness of pixels in the image (each input = one pixel)



steering direction [0 = sharp left .. 1 = sharp right]

Example: Neural Network

- Example with real-values inputs and outputs: early autonomous driving

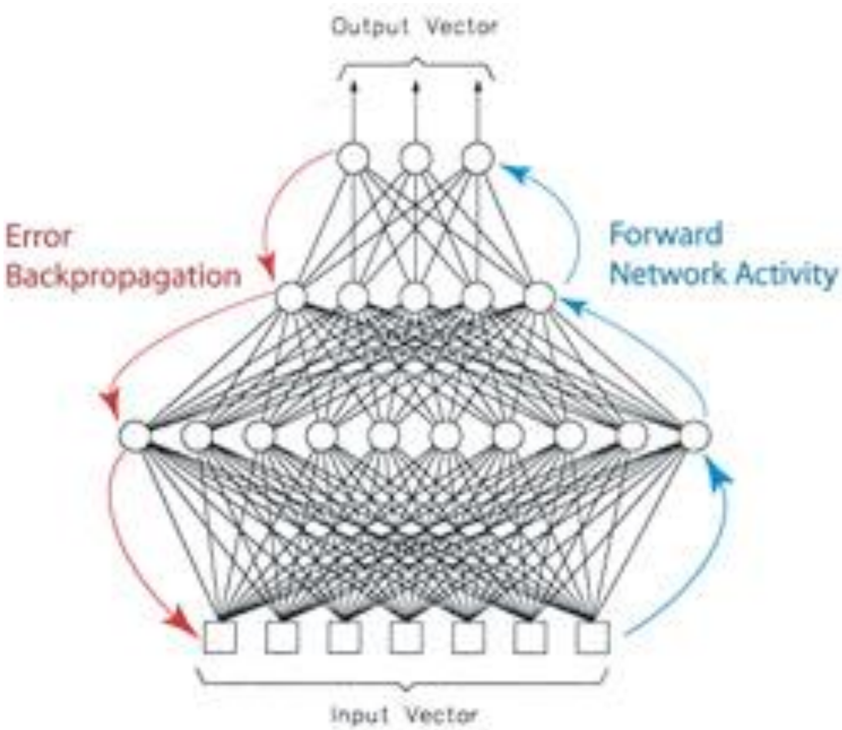


Outline of today's lecture

- Single-layer neural networks
  - perceptron networks
- Multi-layer neural networks
- **Learning in neural networks**
  - **Back propagation**

Training algorithm

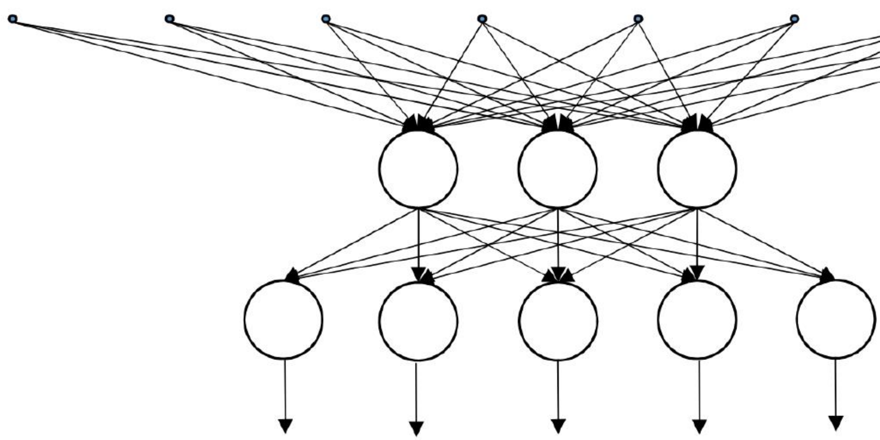
- Start with random weights
- Apply training examples
- Compute output error
- Adjust the weights
- Check if average system error is acceptable



Back propagation

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

$$\boxed{g'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = g(x)(1-g(x))}$$



a_k

w_{kj}

$in_j = \Sigma_k\, w_{kj}\, a_k \qquad a_j = g(in_j)$

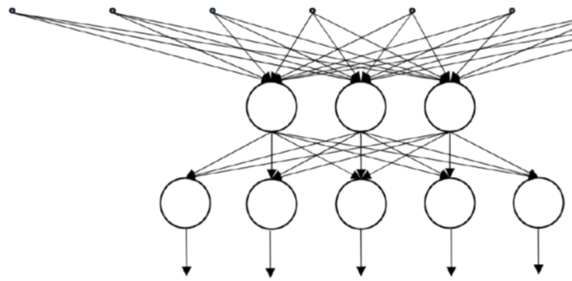
w_{ji}

$in_i = \Sigma_j\, w_{ji}\, a_j \qquad a_i = g(in_i)$

k = index of inputs
j = index of hidden units
i = index of outputs

$$\boxed{\text{Error} = \frac{1}{2}\sum_i (y_i - a_i)^2}$$

Back propagation



a_k

w_{kj}

$in_j = \Sigma_k \, w_{kj} \, a_k \qquad a_j = g(in_j)$

w_{ji}

$in_i = \Sigma_j \, w_{ji} \, a_j \qquad a_i = g(in_i)$

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

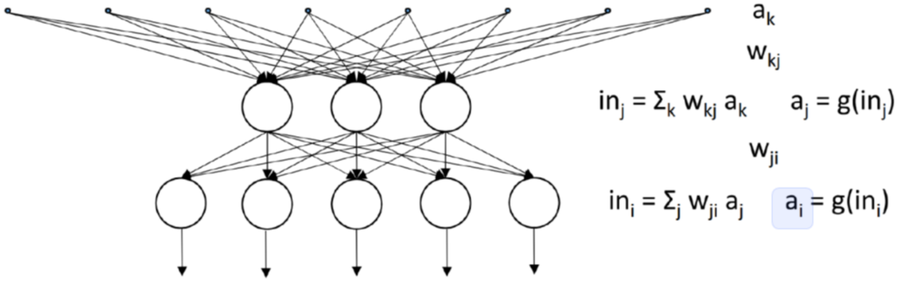
$\boxed{g'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = g(x)(1-g(x))}$

k = index of inputs
j = index of hidden units
i = index of outputs

$$\boxed{\text{Error}=\frac{1}{2}\sum_i (y_i - a_i)^2}$$

$$\frac{d\text{Error}}{dw_{ji}} = -(y_i - a_i)\frac{da_i}{dw_{ji}}$$

Back propagation



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

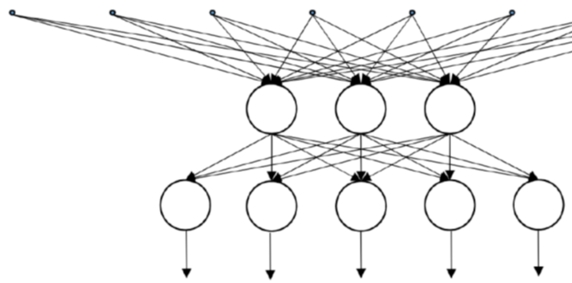
$$g'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = g(x)(1-g(x))$$

k = index of inputs
j = index of hidden units
i = index of outputs

$$\text{Error}=\frac{1}{2}\sum_i (y_i - a_i)^2$$

$$\frac{d\text{Error}}{dw_{ji}} = -(y_i - a_i)\frac{da_i}{dw_{ji}} = -(y_i - a_i)\frac{dg(in_i)}{dw_{ji}}$$

Back propagation



a_k

w_{kj}

$in_j = \Sigma_k\, w_{kj}\, a_k \qquad a_j = g(in_j)$

w_{ji}

$in_i = \Sigma_j\, w_{ji}\, a_j \qquad a_i = g(in_i)$

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

$$\boxed{g'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = g(x)(1-g(x))}$$

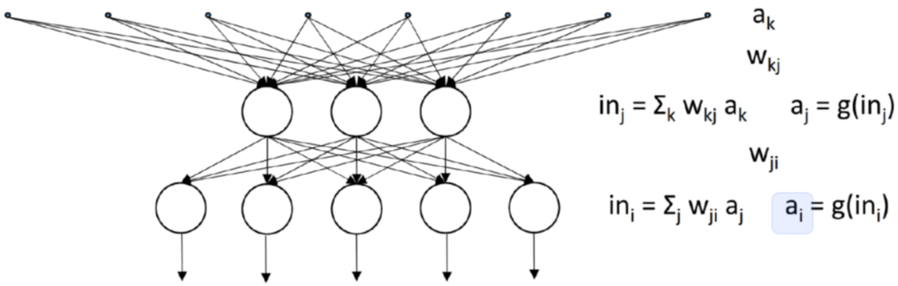
k = index of inputs
j = index of hidden units
i = index of outputs

$$\boxed{\text{Error}=\frac{1}{2}\sum_i (y_i - a_i)^2}$$

$$\frac{d\text{Error}}{dw_{ji}} = -(y_i - a_i)\frac{da_i}{dw_{ji}} = -(y_i - a_i)\frac{dg(in_i)}{dw_{ji}}$$

$$= -(y_i - a_i)g'(in_i)\frac{din_i}{dw_{ji}}$$

Back propagation



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

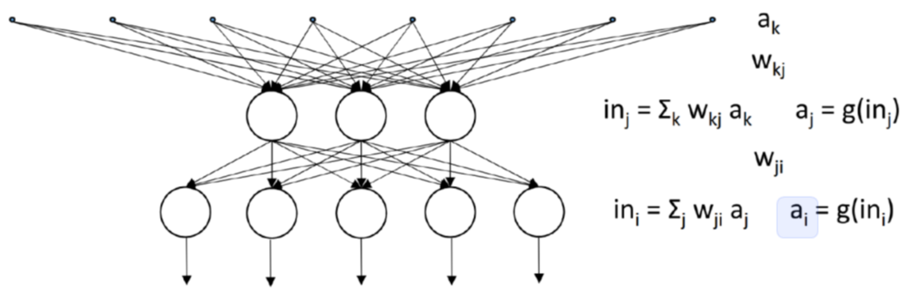
$$g'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = g(x)(1-g(x))$$

k = index of inputs
j = index of hidden units
i = index of outputs

$$\text{Error} = \frac{1}{2}\sum_i (y_i - a_i)^2$$

$$\begin{aligned}\frac{d\text{Error}}{dw_{ji}} &= -(y_i - a_i)\frac{da_i}{dw_{ji}} = -(y_i - a_i)\frac{dg(in_i)}{dw_{ji}} \\ &= -(y_i - a_i)g'(in_i)\frac{din_i}{dw_{ji}} \\ &= -(y_i - a_i)g'(in_i)\frac{d\sum_j w_{ji}a_j}{dw_{ji}}\end{aligned}$$

Back propagation



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

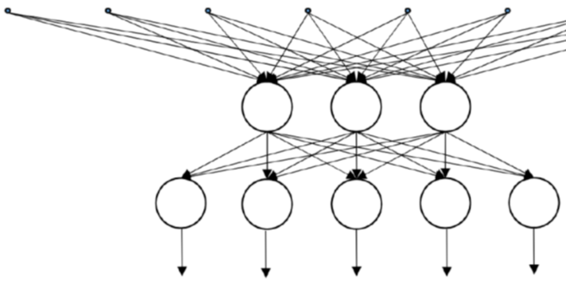
$g'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = g(x)(1 - g(x))$

k = index of inputs
j = index of hidden units
i = index of outputs

$$\text{Error} = \tfrac{1}{2} \textstyle\sum_i (y_i - a_i)^2$$

$$\begin{aligned}
\frac{d\text{Error}}{dw_{ji}} &= -(y_i - a_i)\frac{da_i}{dw_{ji}} = -(y_i - a_i)\frac{dg(in_i)}{dw_{ji}} \\
&= -(y_i - a_i)g'(in_i)\frac{din_i}{dw_{ji}} \\
&= -(y_i - a_i)g'(in_i)\frac{d\sum_j w_{ji}a_j}{dw_{ji}} \\
&= -(y_i - a_i)g'(in_i)a_j
\end{aligned}$$

Back propagation



a_k

w_{kj}

$in_j = \Sigma_k\, w_{kj}\, a_k \qquad a_j = g(in_j)$

w_{ji}

$in_i = \Sigma_j\, w_{ji}\, a_j \qquad a_i = g(in_i)$

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

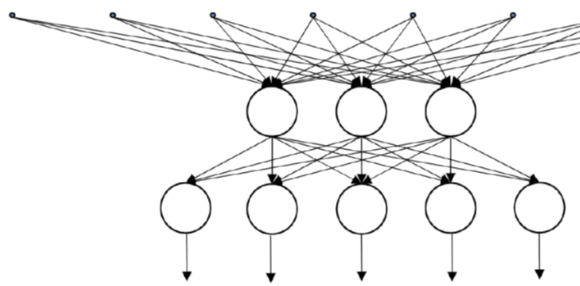
$$\boxed{g'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = g(x)(1-g(x))}$$

k = index of inputs
j = index of hidden units
i = index of outputs

$$\boxed{\text{Error} = \frac{1}{2}\sum_i (y_i - a_i)^2}$$

$$\begin{aligned}
\frac{d\text{Error}}{dw_{ji}} &= -(y_i - a_i)\frac{da_i}{dw_{ji}} = -(y_i - a_i)\frac{dg(in_i)}{dw_{ji}} \\
&= -(y_i - a_i)g'(in_i)\frac{din_i}{dw_{ji}} \\
&= -(y_i - a_i)g'(in_i)\frac{d\sum_j w_{ji}a_j}{dw_{ji}} \\
&= -(y_i - a_i)g'(in_i)a_j
\end{aligned}$$

Back propagation



a_k

w_{kj}

$in_j = \Sigma_k w_{kj} a_k \quad a_j = g(in_j)$

w_{ji}

$in_i = \Sigma_j w_{ji} a_j \quad a_i = g(in_i)$

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

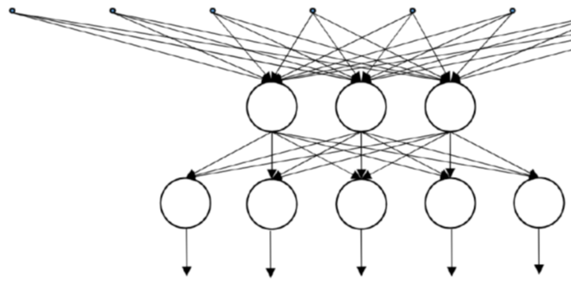
$$g'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = g(x)(1-g(x))$$

k = index of inputs
j = index of hidden units
i = index of outputs

$$\text{Error}=\frac{1}{2}\sum_i (y_i - a_i)^2$$

$$\begin{aligned}
\frac{d\text{Error}}{dw_{ji}} &= -(y_i - a_i)\frac{da_i}{dw_{ji}} = -(y_i - a_i)\frac{dg(in_i)}{dw_{ji}} \\
&= -(y_i - a_i)g'(in_i)\frac{din_i}{dw_{ji}} \\
&= -(y_i - a_i)g'(in_i)\frac{d\sum_j w_{ji}a_j}{dw_{ji}} \\
&= -(y_i - a_i)g'(in_i)a_j \\
&= -\Delta[i]a_j
\end{aligned}$$

Back propagation: *updating weights w_{ji}*



a_k

w_{kj}

$in_j = \Sigma_k \, w_{kj} \, a_k \qquad a_j = g(in_j)$

w_{ji}

$in_i = \Sigma_j \, w_{ji} \, a_j \qquad a_i = g(in_i)$

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

$$g'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = g(x)(1-g(x))$$

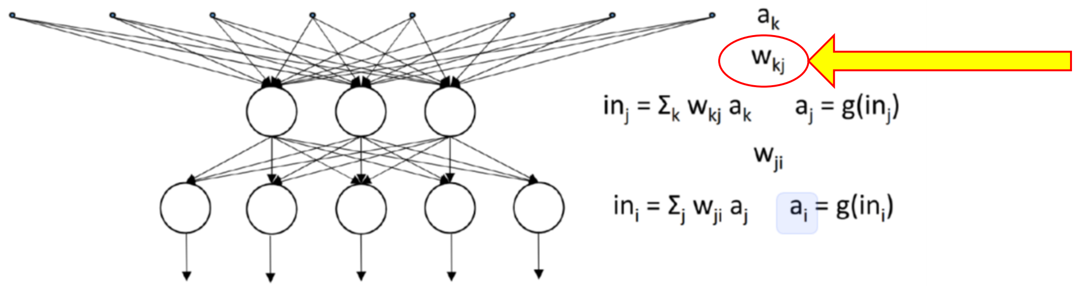
k = index of inputs
j = index of hidden units
i = index of outputs

$$\text{Error} = \frac{1}{2}\sum_i (y_i - a_i)^2$$

Weights at the output layer

$$\boxed{w_{ji} := w_{ji} - \alpha \frac{d\text{Error}}{dw_{ji}} = w_{ji} + \alpha a_j \Delta[i]}$$

Back propagation: *updating weights w_{kj}*



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

$$g'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = g(x)(1-g(x))$$

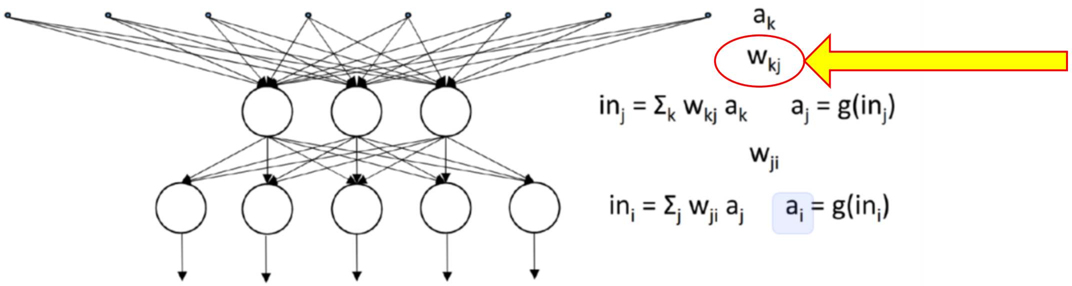
k = index of inputs
j = index of hidden units
i = index of outputs

$$\boxed{\text{Error}=\frac{1}{2}\sum_i (y_i - a_i)^2}$$

$$\begin{aligned}
\frac{d\text{Error}}{dw_{kj}} &= -\sum_i (y_i - a_i)\frac{da_i}{dw_{kj}} \\
&= -\sum_i (y_i - a_i)\frac{dg(in_i)}{dw_{kj}} \\
&= -\sum_i (y_i - a_i)g'(in_i)\frac{din_i}{dw_{kj}} \\
&= -\sum_i \Delta[i]\frac{d(\sum_j w_{ji}a_j)}{dw_{kj}} \\
&= -\sum_i \Delta[i]w_{ji}\frac{da_j}{dw_{kj}}
\end{aligned}$$

$$\begin{aligned}
&= -\sum_i \Delta[i]w_{ji}\frac{dg(in_j)}{dw_{kj}} \\
&= -\sum_i \Delta[i]w_{ji}g'(in_j)\frac{din_j}{dw_{kj}} \\
&= -\sum_i \Delta[i]w_{ji}g'(in_j)\frac{d(\sum_k w_{kj}a_k)}{dw_{kj}} \\
&= -\sum_i \Delta[i]w_{ji}g'(in_j)a_k \\
&= -\Delta[j]a_k, \text{ where } \Delta[j] = \sum_i \Delta[i]w_{ji}g'(in_j)
\end{aligned}$$

Back propagation: *updating weights w_{kj}*



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

$$g'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = g(x)(1-g(x))$$

k = index of inputs
j = index of hidden units
i = index of outputs

$$\boxed{\text{Error} = \frac{1}{2}\sum_i (y_i - a_i)^2}$$

Weights at the hidden layer

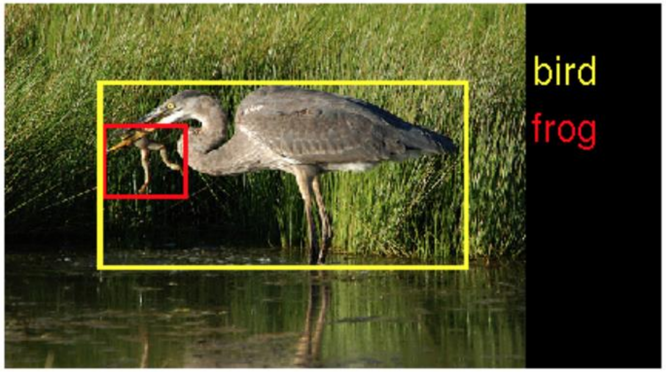
$$\boxed{w_{kj} := w_{kj} - \alpha \frac{d\text{Error}}{dw_{kj}} = w_{kj} + \alpha a_k \Delta[j]}$$

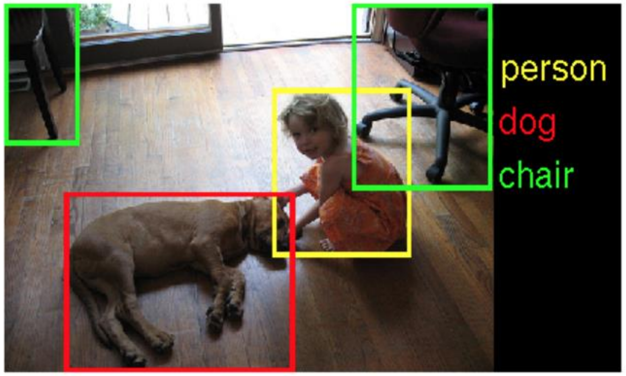
Outline of today's lecture

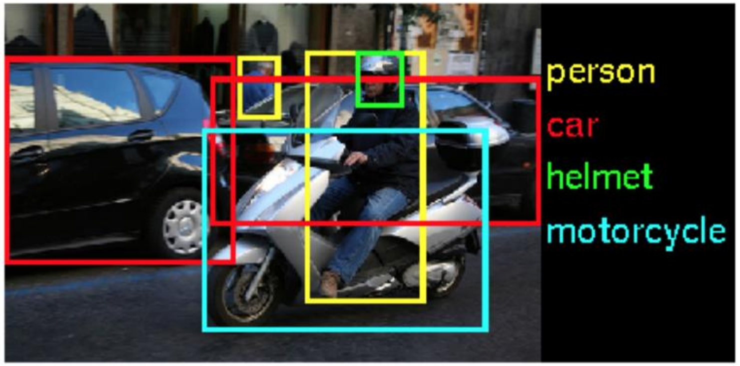
- Single-layer neural networks
  - perceptron networks
- Multi-layer neural networks
- Learning in neural networks
  - Back propagation
- **Applications**

Deep neural networks

- **ImageNet** visual recognition challenge: recognize objects, animals, people, etc (200 categories) in photographs.

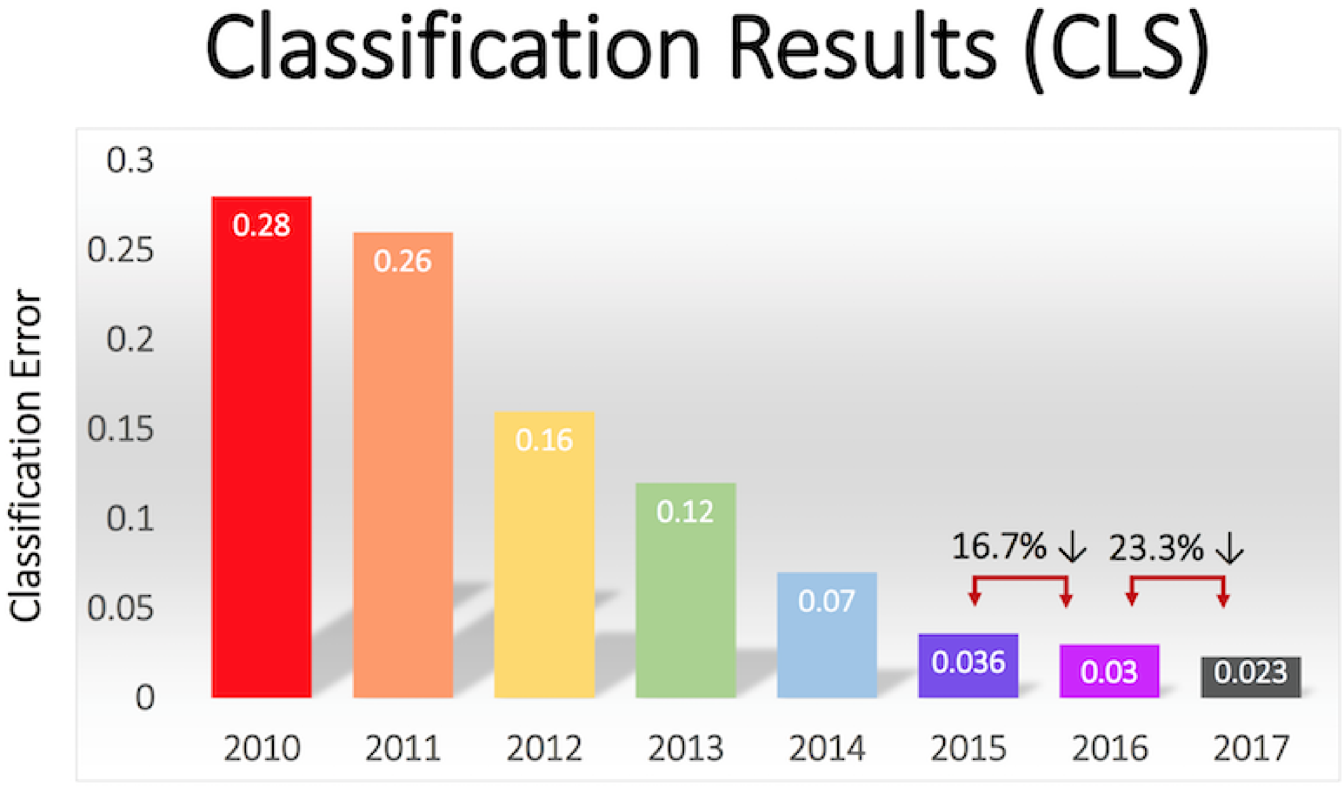






Deep neural networks

- Results keep improving.



Applications: Classification

Business
- Credit rating and risk assessment
- Insurance risk evaluation
- Fraud detection
- Insider dealing detection
- Marketing analysis
- Mailshot profiling
- Signature verification
- Inventory control

Engineering
- Machinery defect diagnosis
- Signal processing
- Character recognition
- Process supervision
- Process fault analysis
- Speech recognition
- Machine vision
- Speech recognition
- Radar signal classification

Security
- Face recognition
- Speaker verification
- Fingerprint analysis

Medicine
- General diagnosis
- Detection of heart defects

Science
- Recognising genes
- Botanical classification
- Bacteria identification

Applications: Modelling

Business

- Prediction of share and commodity prices
- Prediction of economic indicators
- Insider dealing detection
- Marketing analysis
- Mailshot profiling
- Signature verification

Engineering

- Transducer linerisation
- Colour discrimination
- Robot control and navigation
- Process control
- Aircraft landing control
- Car active suspension control
- Printed Circuit auto routing
- Integrated circuit layout
- Image compression

Science

- Prediction of the performance of drugs from the molecular structure
- Weather prediction
- Sunspot prediction

Medicine

- . Medical imaging and image processing

Applications: Forecasting

- Future sales
- Production Requirements
- Market Performance
- Economic Indicators
- Energy Requirements
- Time Based Variables

Applications: Novelty Detection

- Fault Monitoring
- Performance Monitoring
- Fraud Detection
- Detecting Rate Features
- Different Cases

Summary

- Single-layer neural networks
- Multi-layer neural networks
- Learning in neural networks
- Applications