Reinforcement Learning and Neural networks

October 20, 2005

CS 886

University of Waterloo

Outline (Part I)

- Russell & Norvig Sect 21.1-21.3
- What is reinforcement learning
- Temporal-Difference learning
- Q-learning

Machine Learning

- Supervised Learning
  - Teacher tells learner what to remember
- Reinforcement Learning
  - Environment provides hints to learner
- Unsupervised Learning
  - Learner discovers on its own

What is RL?

- Reinforcement learning is learning what to do so as to maximize a numerical reward signal
  - Learner is not told what actions to take, but must discover them by trying them out and seeing what the reward is

What is RL

- Reinforcement learning differs from supervised learning

Supervised learning

Reinforcement learning

Ouch!

Animal Psychology

- Negative reinforcements:
  - Pain and hunger
- Positive reinforcements:
  - Pleasure and food
- Reinforcements used to train animals

- Let's do the same with computers!

RL Examples

- Game playing (backgammon, solitaire)
- Operations research (pricing, vehicule routing)
- Elevator scheduling
- Helicopter control

- http://neuromancer.eecs.umich.edu/cgi-bin/twiki/view/Main/SuccessesOfRL

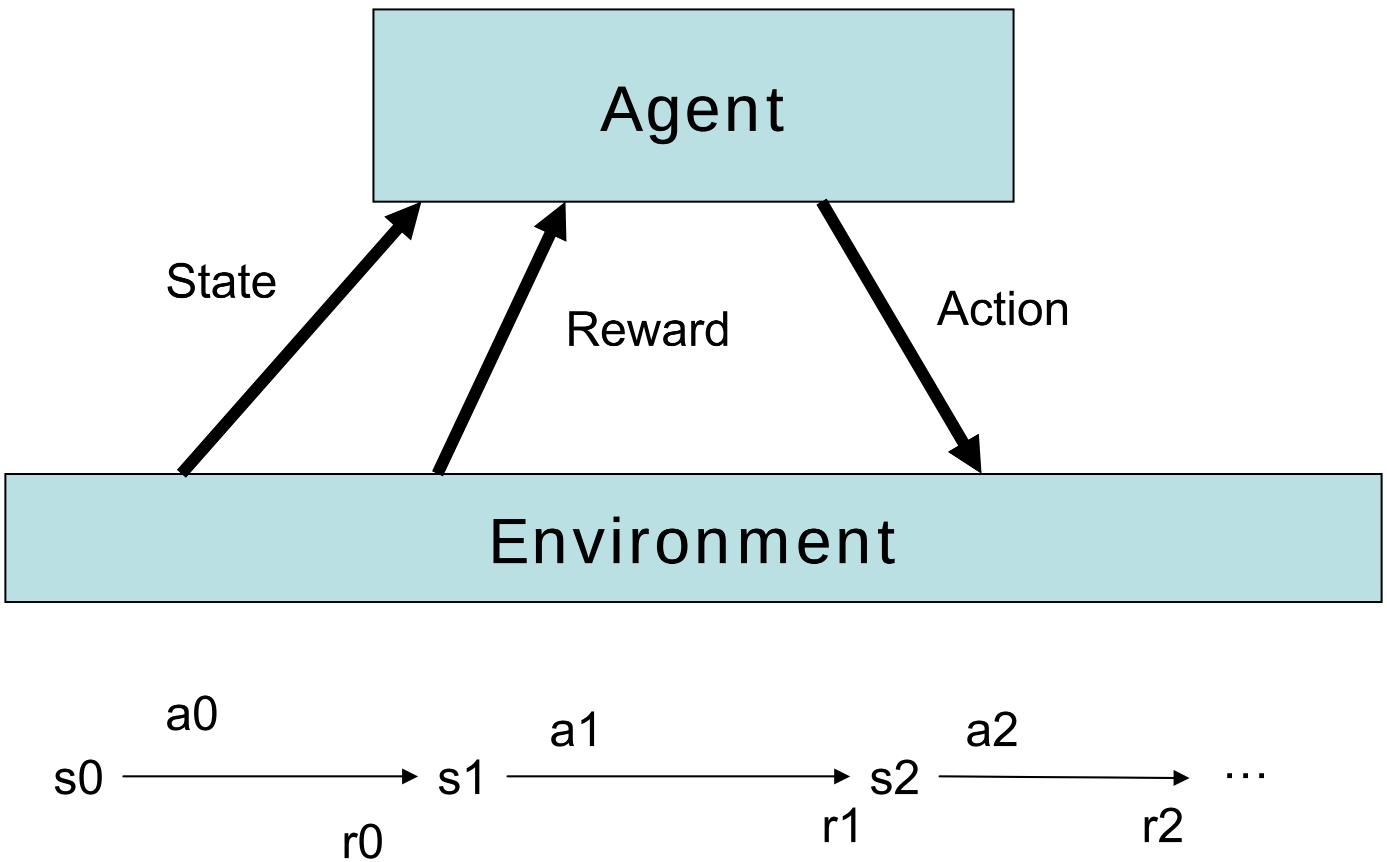
Reinforcement Learning

- Definition:
  - Markov decision process with unknown transition and reward models
- Set of states S
- Set of actions *A*
  - Actions may be stochastic
- Set of reinforcement signals (rewards)
  - Rewards may be delayed

Policy optimization

- Markov Decision Process:
  - Find optimal policy given transition and reward model
  - Execute policy found
- Reinforcement learning:
  - Learn an optimal policy while interacting with the environment

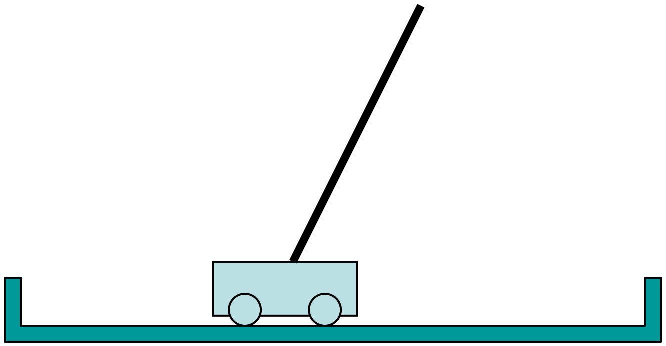
Reinforcement Learning Problem



Goal: Learn to choose actions that maximize $r_0+\gamma\, r_1+\gamma^2 r_2+\ldots$, where $0 \leq \gamma < 1$

Example: Inverted Pendulum

- State: $x(t)$, $x'(t)$, $\theta(t)$, $\theta'(t)$
- Action: Force F
- Reward: 1 for any step where pole balanced



Problem: Find $\delta: S \rightarrow A$ that maximizes rewards

Rl Characterisitics

- Reinforcements: rewards
- Temporal credit assignment: when a reward is received, which action should be credited?
- Exploration/exploitation tradeoff: as agent learns, should it exploit its current knowledge to maximize rewards or explore to refine its knowledge?
- Lifelong learning: reinforcement learning

Types of RL

- Passive vs Active learning
  - Passive learning: the agent executes a fixed policy and tries to evaluate it
  - Active learning: the agent updates its policy as it learns

- Model based vs model free
  - Model-based: learn transition and reward model and use it to determine optimal policy
  - Model free: derive optimal policy without learning the model

Passive Learning

- Transition and reward model known:
  - Evaluate δ:
  - $V^{\delta}(s) = R(s) + \gamma \Sigma_{s'} Pr(s'|s,\delta(s)) V^{\delta}(s')$
- Transition and reward model unknown:
  - Estimate policy value as agent executes policy: $V^{\delta}(s) = E_{\delta}[\Sigma_t \gamma^t R(s_t)]$
  - Model based vs model free

Passive learning

3	r	r	r	+1
2	u		u	-1
1	u	l	l	l
	1	2	3	4

$\gamma = 1$

$r_i = -0.04$ for non-terminal states

Do not know the transition probabilities

$(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (3,3) \rightarrow (4,3)_{+1}$
$(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (3,3) \rightarrow (3,2) \rightarrow (3,3) \rightarrow (4,3)_{+1}$
$(1,1) \rightarrow (2,1) \rightarrow (3,1) \rightarrow (3,2) \rightarrow (4,2)_{-1}$

What is the value V(s) of being in state s?

Passive ADP

- Adaptive dynamic programming (ADP)
  - Model-based
  - Learn transition probabilities and rewards from observations
  - Then update the values of the states

ADP Example

$\gamma = 1$

3	r	r	r	+1
2	u		u	-1
1	u	l	l	l
	1	2	3	4

$r_i = -0.04$ for non-terminal states

$$V^{\delta}(s) = R(s) + \gamma \Sigma_{s'} \Pr(s'|s,\delta(s))\, V^{\delta}(s')$$

(1,1)→ (1,2)→ **(1,3)**→ (1,2)→ **(1,3)**→ (2,3)→ (3,3)→ $(4,3)_{+1}$
(1,1)→ (1,2)→ **(1,3)**→ (2,3)→ (3,3)→ (3,2)→ (3,3)→ $(4,3)_{+1}$
(1,1)→ (2,1)→ (3,1)→ (3,2)→ $(4,2)_{-1}$

$$\left.\begin{array}{l} P((2,3)|(1,3),r) = 2/3 \\ P((1,2)|(1,3),r) = 1/3 \end{array}\right\} \text{Use this information in}$$

We need to learn all the transition probabilities!

Passive TD

- Temporal difference (TD)
  - Model free
- At each time step
  - Observe: s,a,s',r
  - Update $V^\delta(s)$ after each move
  - $V^\delta(s) = V^\delta(s) + \alpha \underbrace{(R(s) + \gamma V^\delta(s') - V^\delta(s))}_{\text{Temporal difference}}$

Learning rate

TD Convergence

Thm: If α is appropriately decreased with number of times a state is visited then $V^{\delta}(s)$ converges to correct value

- α must satisfy:
  - $\Sigma_t \alpha_t \rightarrow \infty$
  - $\Sigma_t (\alpha_t)^2 < \infty$
- Often $\alpha(s) = 1/n(s)$
  - $n(s)$ = # of times s is visited

Active Learning

- Ultimately, we are interested in improving δ
- Transition and reward model known:
  - $V^*(s) = \max_a R(s) + \gamma \Sigma_{s'} \Pr(s'|s,a) V^*(s')$
- Transition and reward model unknown:
  - Improve policy as agent executes policy
  - Model based vs model free

Q-learning (aka active temporal difference)

- Q-function: $Q: S \times A \rightarrow \Re$
  - Value of state-action pair
  - Policy $\delta(s) = \text{argmax}_a\, Q(s,a)$ is the optimal policy

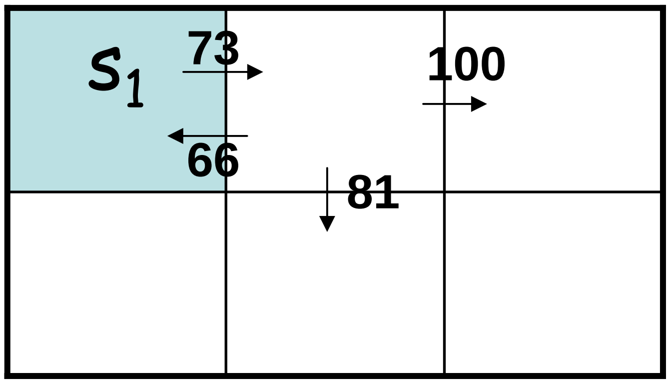
- Bellman's equation:

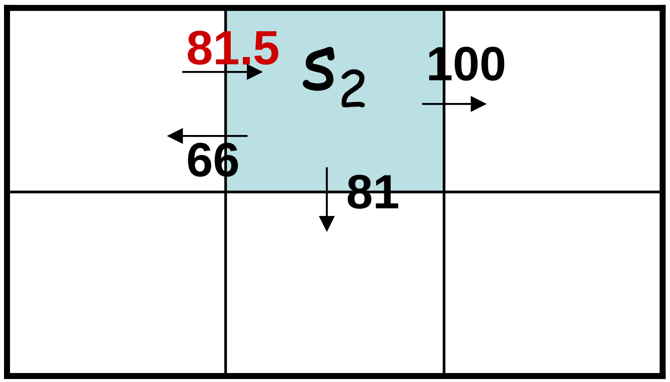
$$\boxed{Q^*(s,a) = R(s) + \gamma\, \Sigma_{s'}\, \Pr(s'|s,a)\, \max_{a'}\, Q^*(s',a')}$$

Q-learning

- For each state s and action a initialize $Q(s,a)$ (0 or random)
- Observe current state
- Loop
  - Select action a and execute it
  - Receive immediate reward r
  - Observe new state s'
  - Update $Q(a,s)$
    - $Q(s,a) = Q(s,a) + \alpha(r(s)+\gamma \max_{a'} Q(s',a') - Q(s,a))$
  - $s=s'$

Q-learning example





r=0 for non-terminal states
$\gamma=0.9$
$\alpha=0.5$

$$
\begin{aligned}
Q(s_1,right) &= Q(s_1,right) + \alpha\,(r(s_1) + \gamma \max_{a'} Q(s_2,a') - Q(s_1,right)) \\
&= 73 + 0.5\,(0 + 0.9 \max[66,81,100] - 73) \\
&= 73 + 0.5\,(17) \\
&= 81.5
\end{aligned}
$$

Q-learning

- For each state s and action a initialize Q(s,a) (0 or random)
- Observe current state
- Loop
  - Select action a and execute it
  - Receive immediate reward r
  - Observe new state s'
  - Update Q(a,s)
    - $Q(s,a) = Q(s,a) + \alpha(r(s)+\gamma \max_{a'} Q(s',a') - Q(s,a))$
  - s=s'

Exploration vs Exploitation

- If an agent always chooses the action with the highest value then it is exploiting
  - The learned model is not the real model
  - Leads to suboptimal results
- By taking random actions (pure exploration) an agent may learn the model
  - But what is the use of learning a complete model if parts of it are never used?
- Need a balance between exploitation and exporation

Common exploration methods

- ε-greedy:
  - With probability ε execute random action
  - Otherwise execute best action a*
    $a^* = \text{argmax}_a\, Q(s,a)$

- Boltzmann exploration

$$P(a) = \frac{e^{Q(s,a)/T}}{\Sigma_a e^{Q(s,a)/T}}$$

Exploration and Q-learning

- Q-learning converges to optimal Q-values if
  - Every state is visited infinitely often (due to exploration)
  - The action selection becomes greedy as time approaches infinity
  - The learning rate a is decreased fast enough but not too fast

A Triumph for Reinforcement Learning: TD-Gammon

- Backgammon player: TD learning with a neural network representation of the value function:

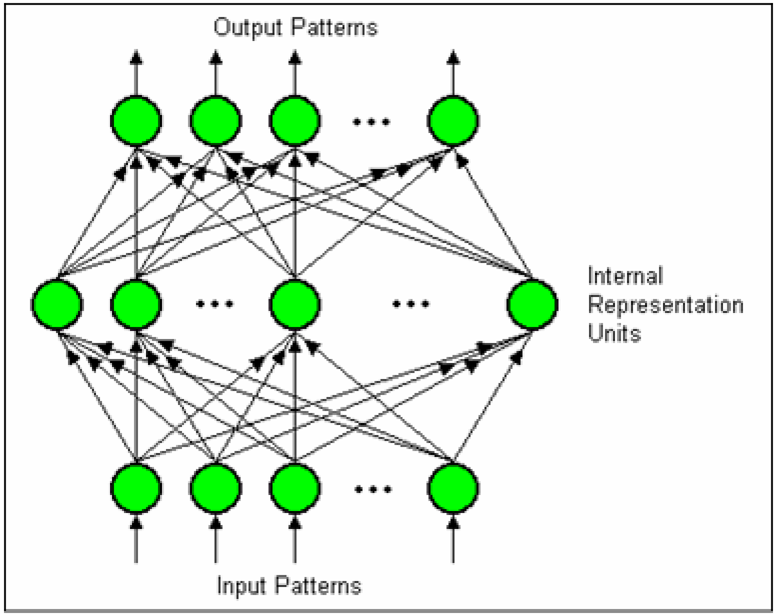


Figure 1. An illustration of the multilayer perception architecture used in TD-Gammon's neural network. This architecture is also used in the popular backpropagation learning procedure. Figure reproduced from [9].

Outline (Part II)

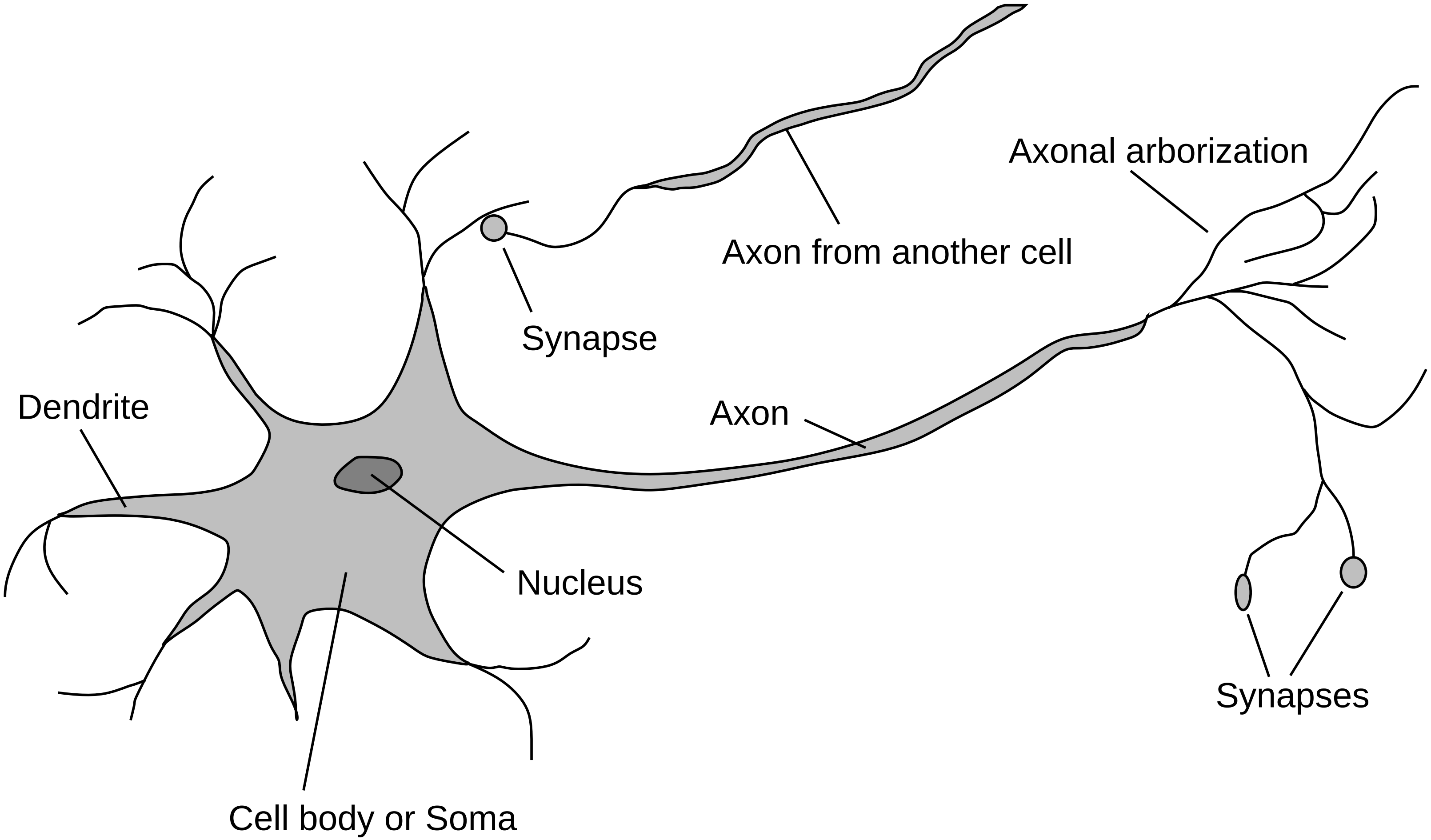
- Neural networks
  - Perceptron
  - Supervised learning algorithms for neural networks

- Reading: R&N Ch 20.5

Brain

- Seat of human intelligence
- Where memory/knowledge resides
- Responsible for thoughts and decisions
- Can learn
- Consists of nerve cells called neurons

Neuron



Axonal arborization
Axon from another cell
Synapse
Dendrite
Axon
Nucleus
Synapses
Cell body or Soma

Comparison

- Brain
  - Network of neurons
  - Nerve signals propagate in a neural network
  - Parallel computation
  - Robust (neurons die everyday without any impact)
- Computer
  - Bunch of gates
  - Electrical signals directed by gates
  - Sequential computation
  - Fragile (if a gate stops working, computer crashes)

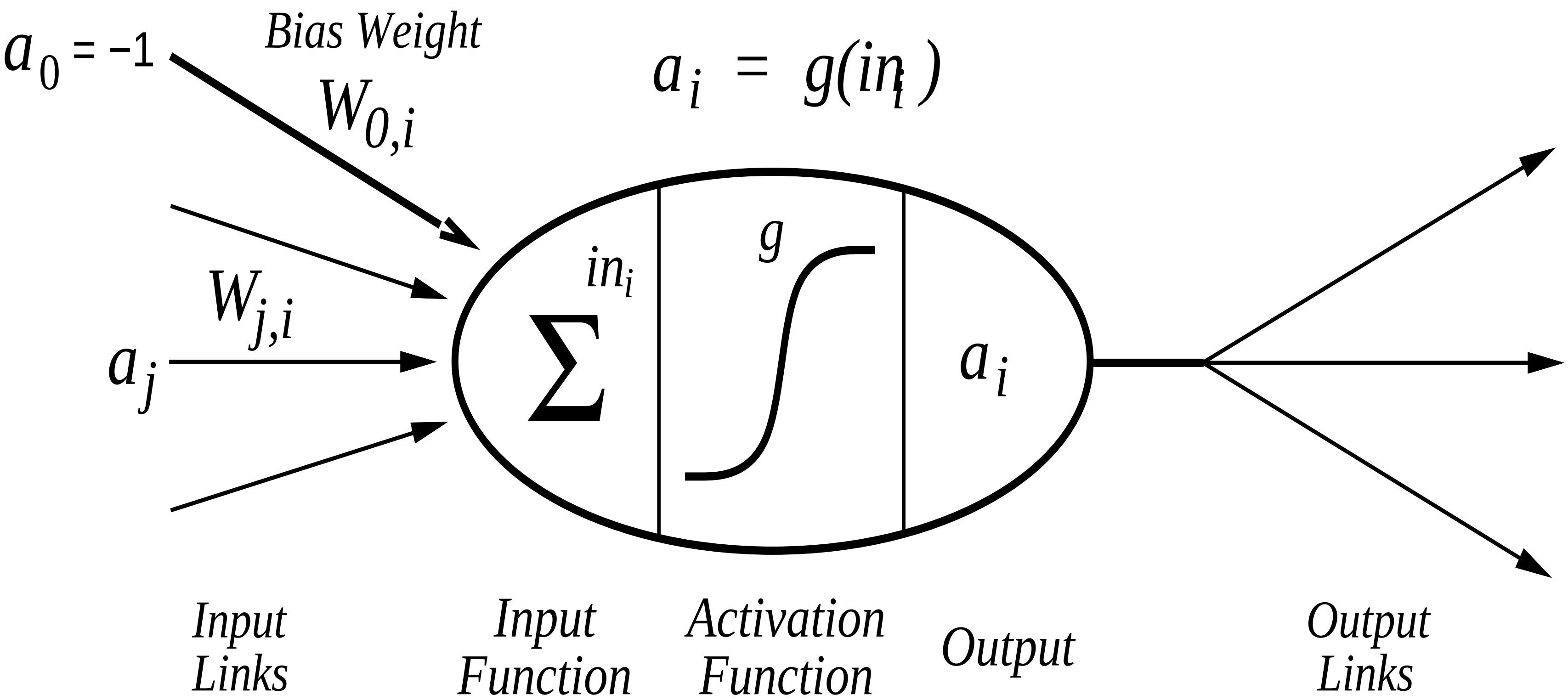
Artificial Neural Networks

- Idea: mimic the brain to do computation
- Artificial neural network:
  - Nodes (a.k.a units) correspond to neurons
  - Links correspond to synapses
- Computation:
  - Numerical signal transmitted between nodes corresponds to chemical signals between neurons
  - Nodes modifying numerical signal corresponds to neurons firing rate

ANN Unit

- For each unit i:
- Weights: W_{ji}
  - Strength of the link from unit j to unit i
  - Input signals a_j weighted by W_{ji} and linearly combined: $in_i = \Sigma_j W_{ji} a_j$
- Activation function: g
  - Numerical signal produced: $a_i = g(in_i)$

ANN Unit



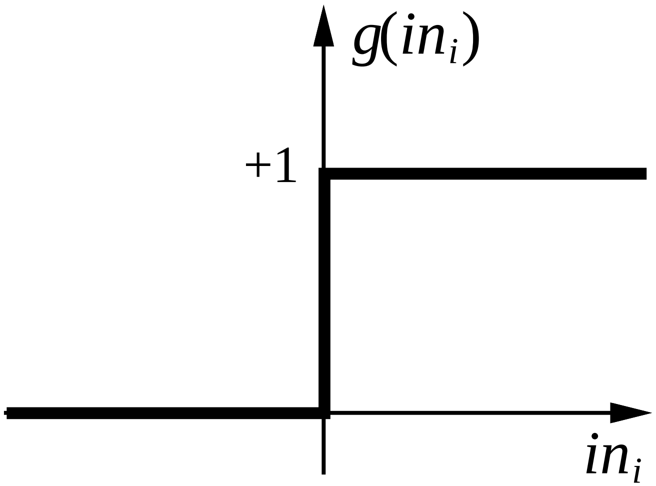
$a_0 = -1$
Bias Weight
$W_{0,i}$
$a_i = g(in_i)$
$W_{j,i}$
a_j
in_i
Σ
g
a_i
Input Links
Input Function
Activation Function
Output
Output Links

Activation Function

- Should be nonlinear
  - Otherwise network is just a linear function
- Often chosen to mimic firing in neurons
  - Unit should be "active" (output near 1) when fed with the "right" inputs
  - Unit should be "inactive" (output near 0) when fed with the "wrong" inputs

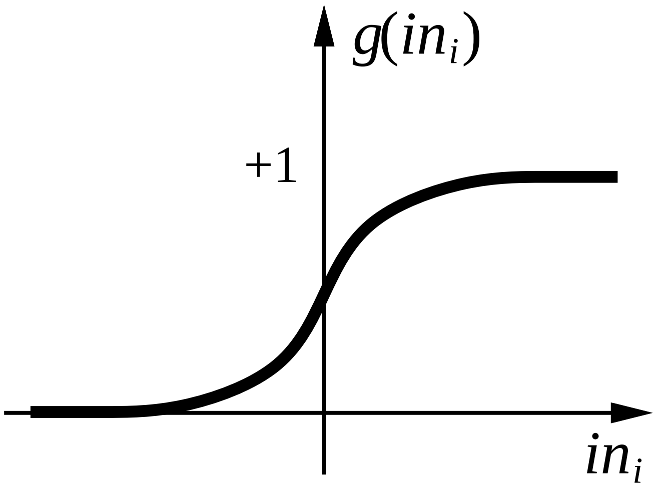
Common Activation Functions

Threshold



(a)

Sigmoid

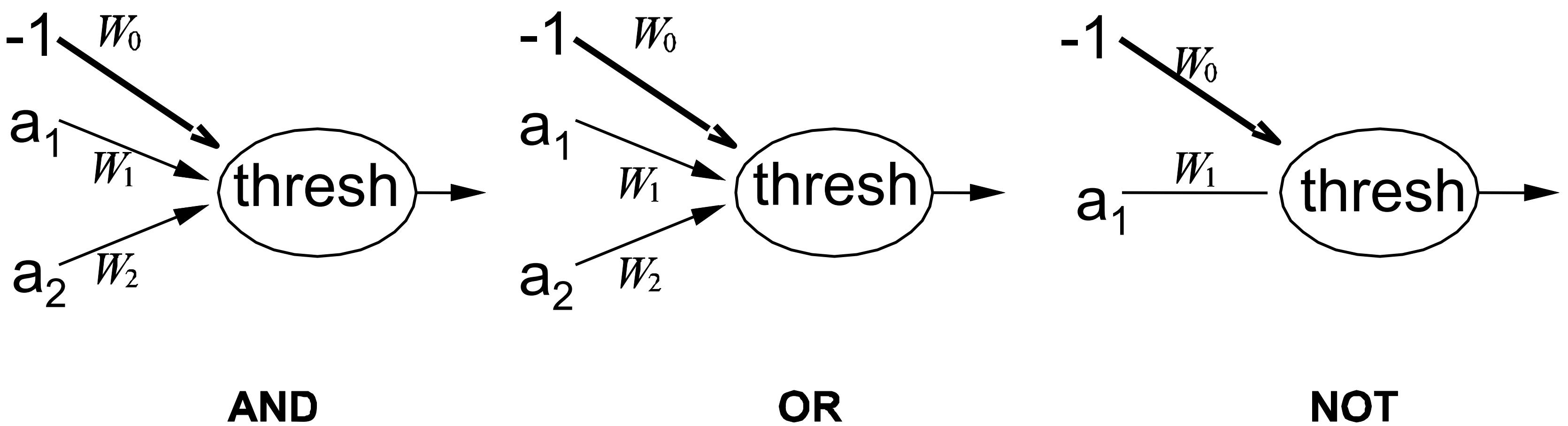


(b)

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

Logic Gates

- McCulloch and Pitts (1943)
  - Design ANNs to represent Boolean fns
- What should be the weights of the following units to code AND, OR, NOT ?

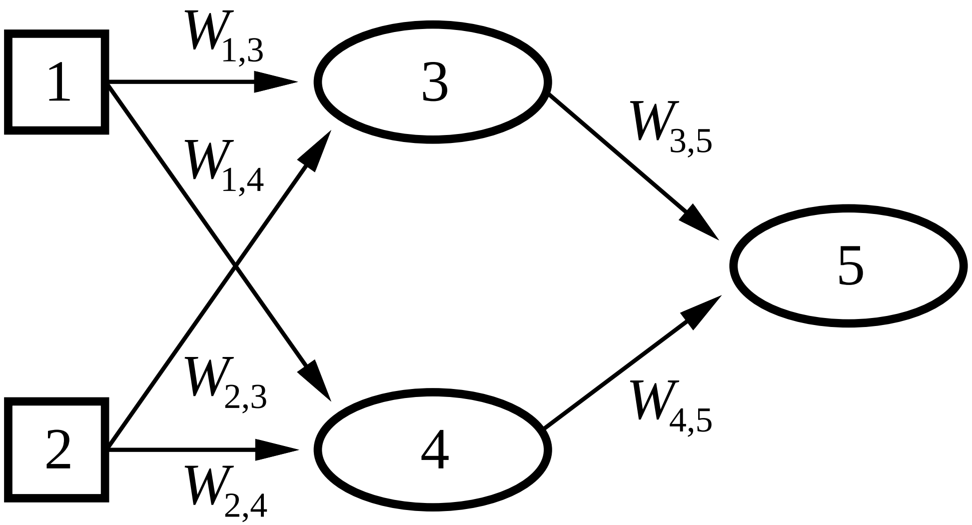


Network Structures

- Feed-forward network
  - Directed acyclic graph
  - No internal state
  - Simply computes outputs from inputs
- Recurrent network
  - Directed cyclic graph
  - Dynamical system with internal states
  - Can memorize information

Feed-forward network

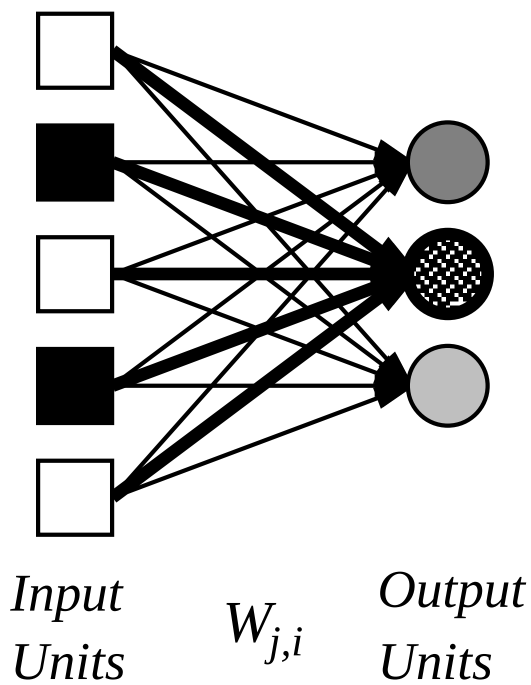
- Simple network with two inputs, one hidden layer of two units, one output unit



$$a_5 = g(W_{3,5}a_3 + W_{4,5}a_4)$$
$$= g(W_{3,5}g(W_{1,3}a_1 + W_{2,3}a_2) + W_{4,5}g(W_{1,4}a_1 + W_{2,4}a_2))$$

Perceptron

- Single layer feed-forward network



Supervised Learning

- Given list of <input,output> pairs
- Train feed-forward ANN
  - To compute proper outputs when fed with inputs
  - Consists of adjusting weights W_{ji}

- Simple learning algorithm for threshold perceptrons

Threshold Perceptron Learning

- Learning is done separately for each unit
  - Since units do not share weights
- Perceptron learning for unit i:
  - For each <inputs,output> pair do:
    - Case 1: correct output produced
      - $\forall_j \; W_{ji} \leftarrow W_{ji}$
    - Case 2: output produced is 0 instead of 1
      - $\forall_j \; W_{ji} \leftarrow W_{ji} + a_j$
    - Case 3: output produced is 1 instead of 0
      - $\forall_j \; W_{ji} \leftarrow W_{ji} - a_j$
  - Until correct output for all training instances

Threshold Perceptron Learning

- Dot products: $a \bullet a \geq 0$ and $-a \bullet a \leq 0$
- Perceptron computes
  - 1 when $a \bullet W = \Sigma_j a_j W_{ji} > 0$
  - 0 when $a \bullet W = \Sigma_j a_j W_{ji} < 0$
- If output should be 1 instead of 0 then
  - $W \leftarrow W+a$ since $a \bullet (W+a) \geq a \bullet W$
- If output should be 0 instead of 1 then
  - $W \leftarrow W-a$ since $a \bullet (W-a) \leq a \bullet W$

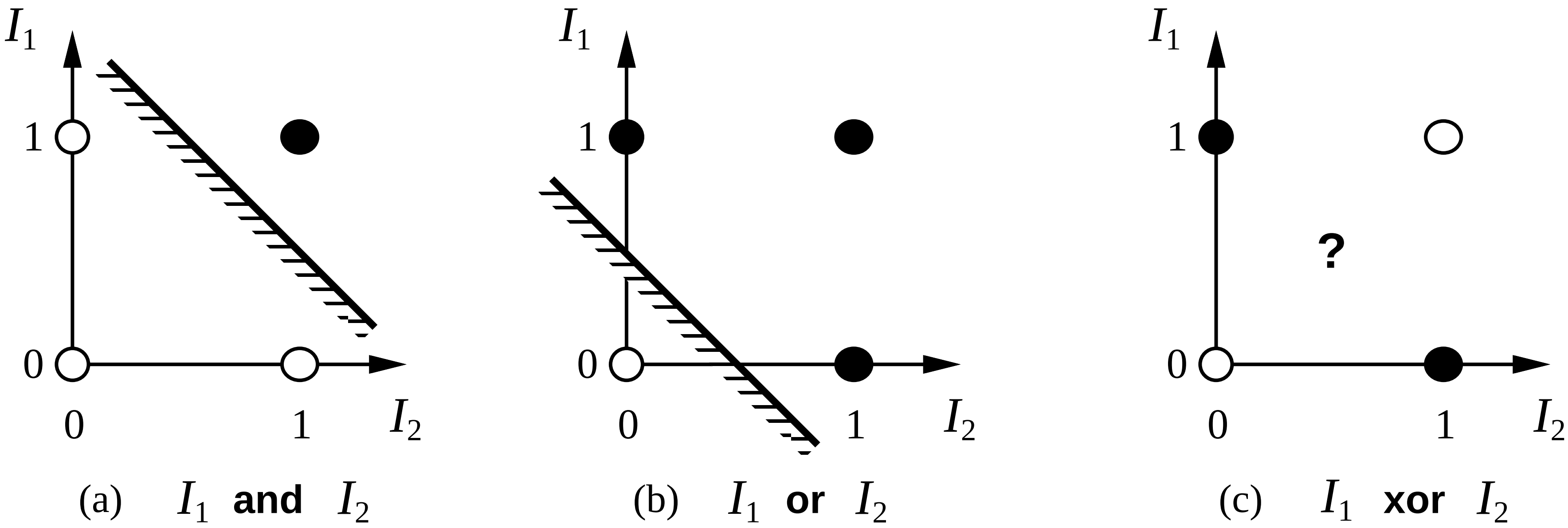
Threshold Perceptron Hypothesis Space

- Hypothesis space h_W:
  - All binary classifications with param. W s.t.
    - a•W > 0 → 1
    - a•W < 0 → 0

- Since a•W is linear in W, perceptron is called a linear separator

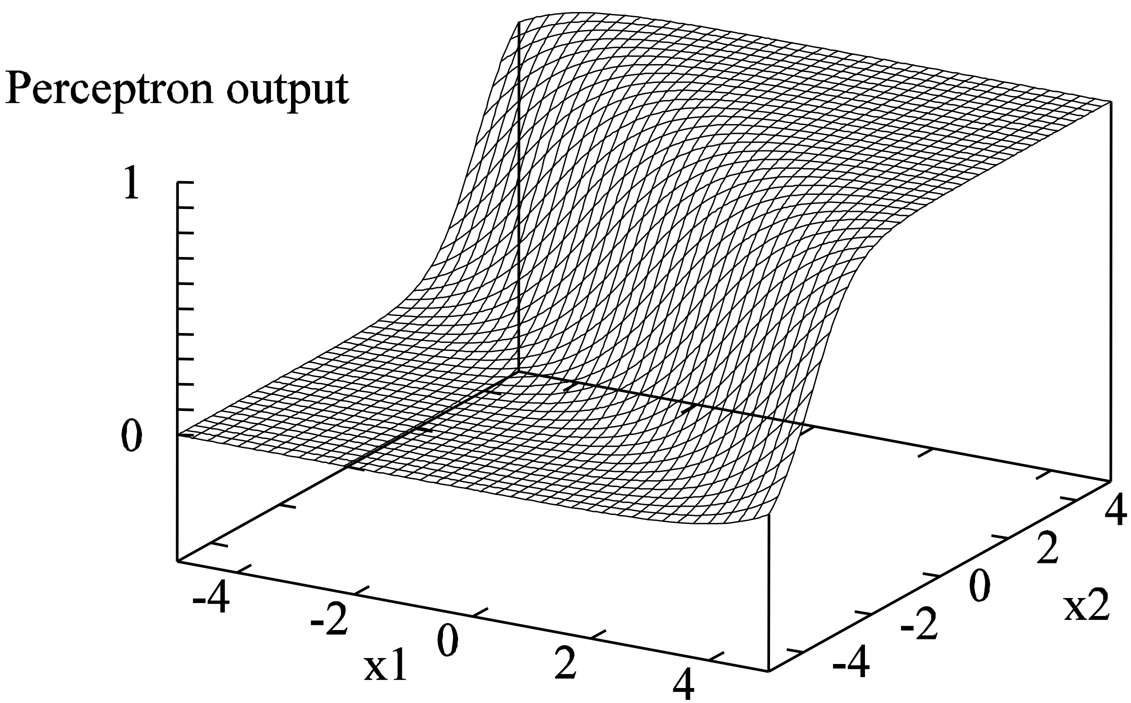
Threshold Perceptron Hypothesis Space

- Are all Boolean gates linearly separable?



Sigmoid Perceptron

- Represent "soft" linear separators



Sigmoid Perceptron Learning

- Formulate learning as an optimization search in weight space
  - Since g differentiable, use gradient descent

- Minimize squared error:
  - $E = 0.5\ Err^2 = 0.5\ (y - h_W(\mathbf{x}))^2$
    - $\mathbf{x}$: input
    - y: target output
    - $h_W(\mathbf{x})$: computed output

Perceptron Error Gradient

- $E = 0.5\ Err^2 = 0.5\ (y - h_{\mathbf{W}}(\mathbf{x}))^2$

- $\partial E/\partial W_j = Err \times \partial Err/\partial W_j$
  $= Err \times \partial(y - g(\Sigma_j W_j x_j))$
  $= -Err \times g'(\Sigma_j W_j x_j) \times x_j$

- When g is sigmoid fn, then $g' = g(1-g)$

Perceptron Learning Algorithm

- Perceptron-Learning(examples,network)
  - Repeat
    - For each e in examples do
      - in $\leftarrow \Sigma_j W_j x_j[e]$
      - Err $\leftarrow y[e] - g(in)$
      - $W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j[e]$
  - Until some stopping criteria satisfied
  - Return learnt network
- N.B. α is a learning rate corresponding to the step size in gradient descent

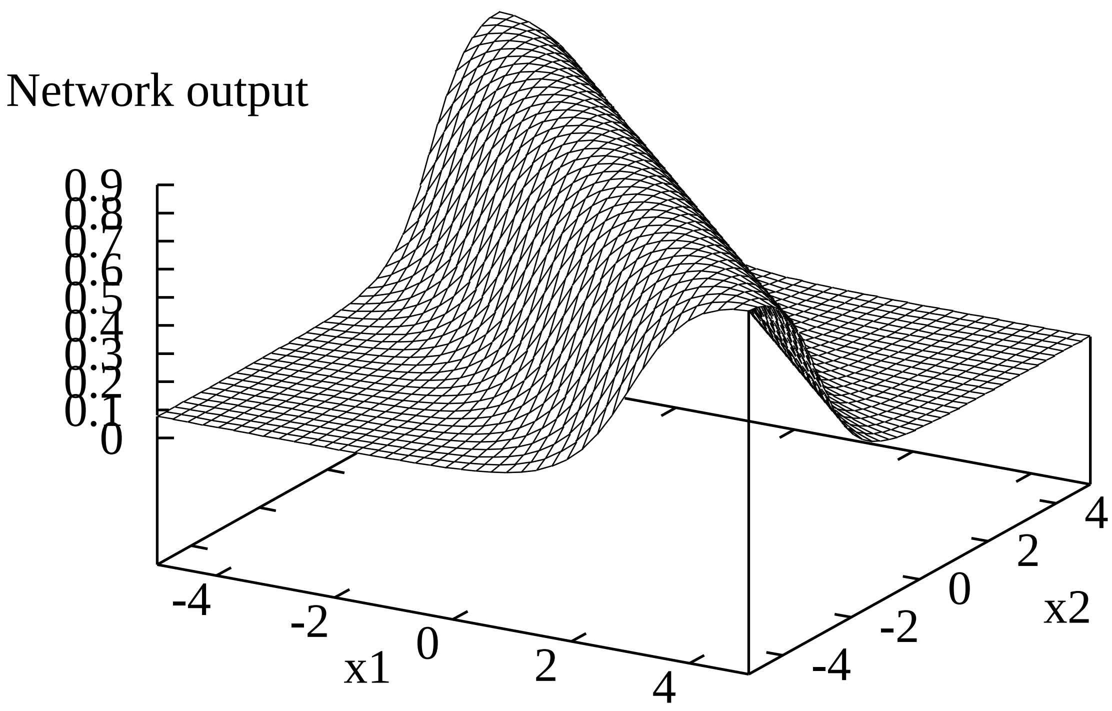
Multilayer Feed-forward Neural Networks

- Perceptron can only represent (soft) linear separators
  - Because single layer

- With multiple layers, what fns can be represented?
  - Virtually any function!

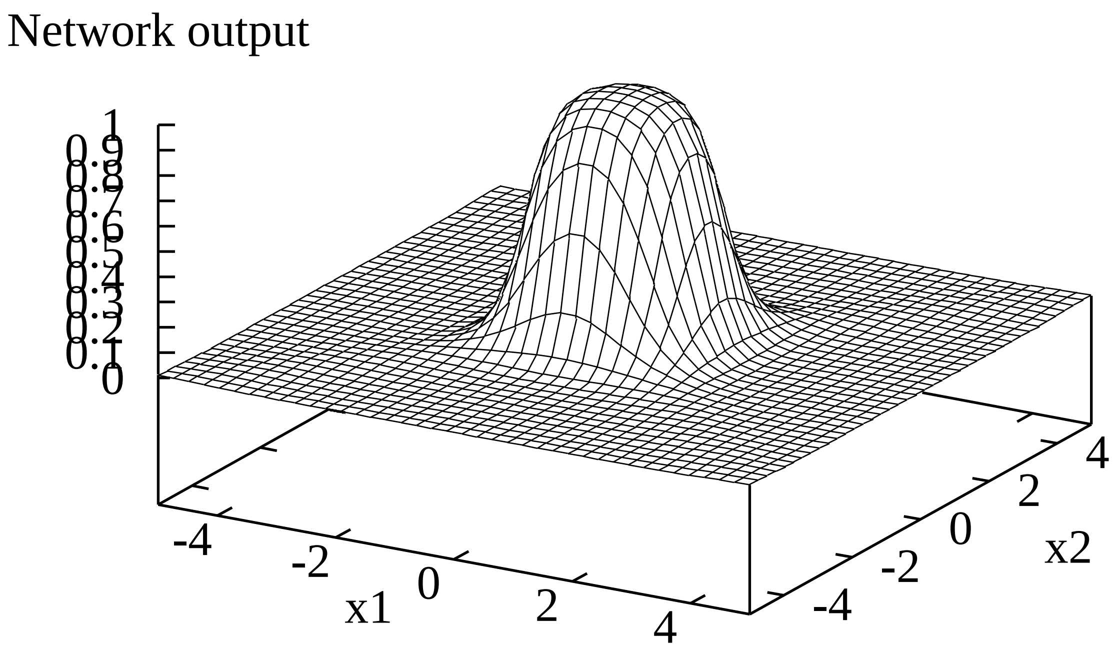
Multilayer Networks

- Adding two sigmoid units with parallel but opposite "cliffs" produces a ridge



Multilayer Networks

- Adding two intersecting ridges (and thresholding) produces a bump



Multilayer Networks

- By tiling bumps of various heights together, we can approximate any function

- Training algorithm:
  - Back-propagation
  - Essentially gradient performed by propagating errors backward into the network
  - See textbook for derivation

Neural Net Applications

- Neural nets can approximate any function, hence 1000's of applications
  - NETtalk for pronouncing English text
  - Character recognition
  - Paint-quality inspection
  - Vision-based autonomous driving
  - Etc.

Neural Net Drawbacks

- Common problems:
  - How should we interpret units?
  - How many layers and units should a network have?
  - How to avoid local optimum while training with gradient descent?