

Lecture 8: Perceptrons, Neural Networks

CS480/680 Intro to Machine Learning

2023-2-2

Pascal Poupart
David R. Cheriton School of Computer Science



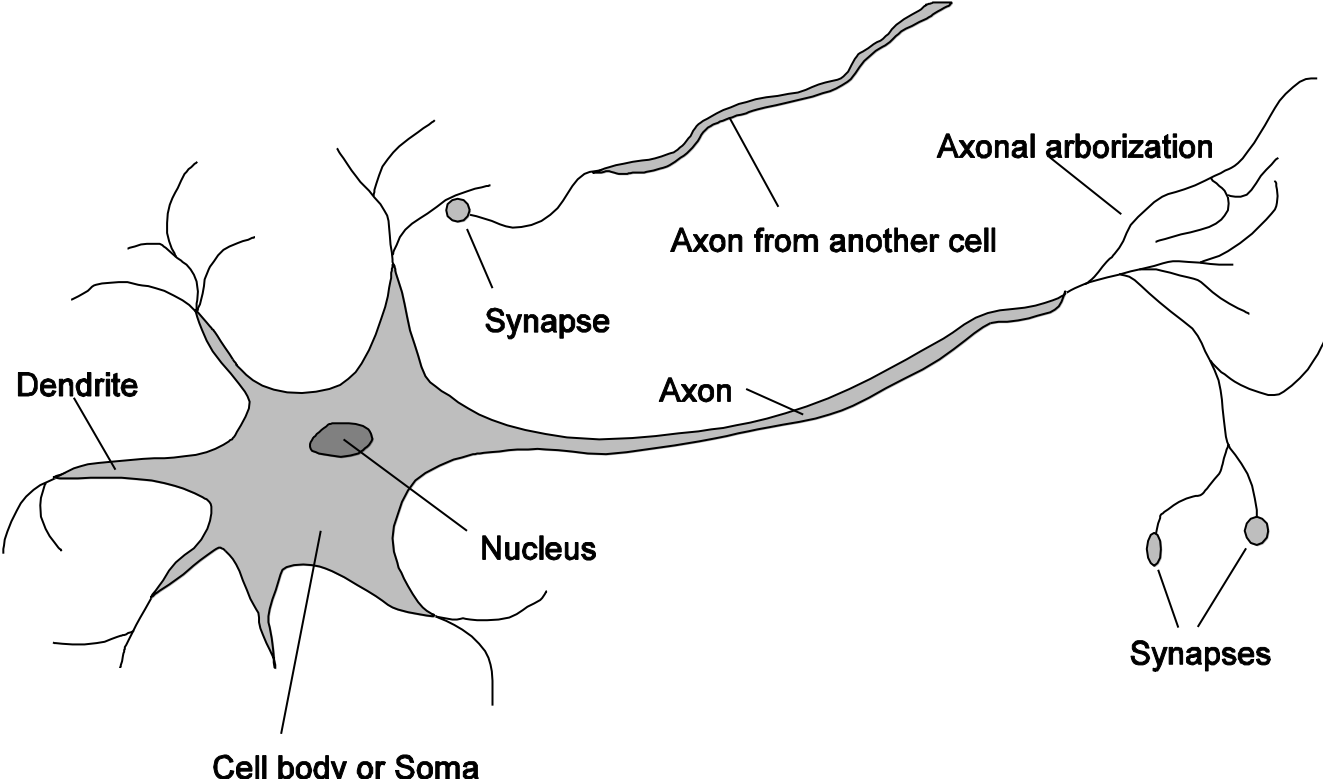
Outline

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

Brain

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

Neuron



Comparison

- Brain
 - Network of neurons
 - Nerve signals propagate in a neural network
 - Parallel computation
 - **Robust (neurons die everyday without much impact)**

- Computer
 - Bunch of gates
 - Electrical signals directed by gates
 - Sequential and parallel 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**

- Strength of the link from unit i to unit j
- Input signals x_i weighted by W_{ji} and linearly combined:

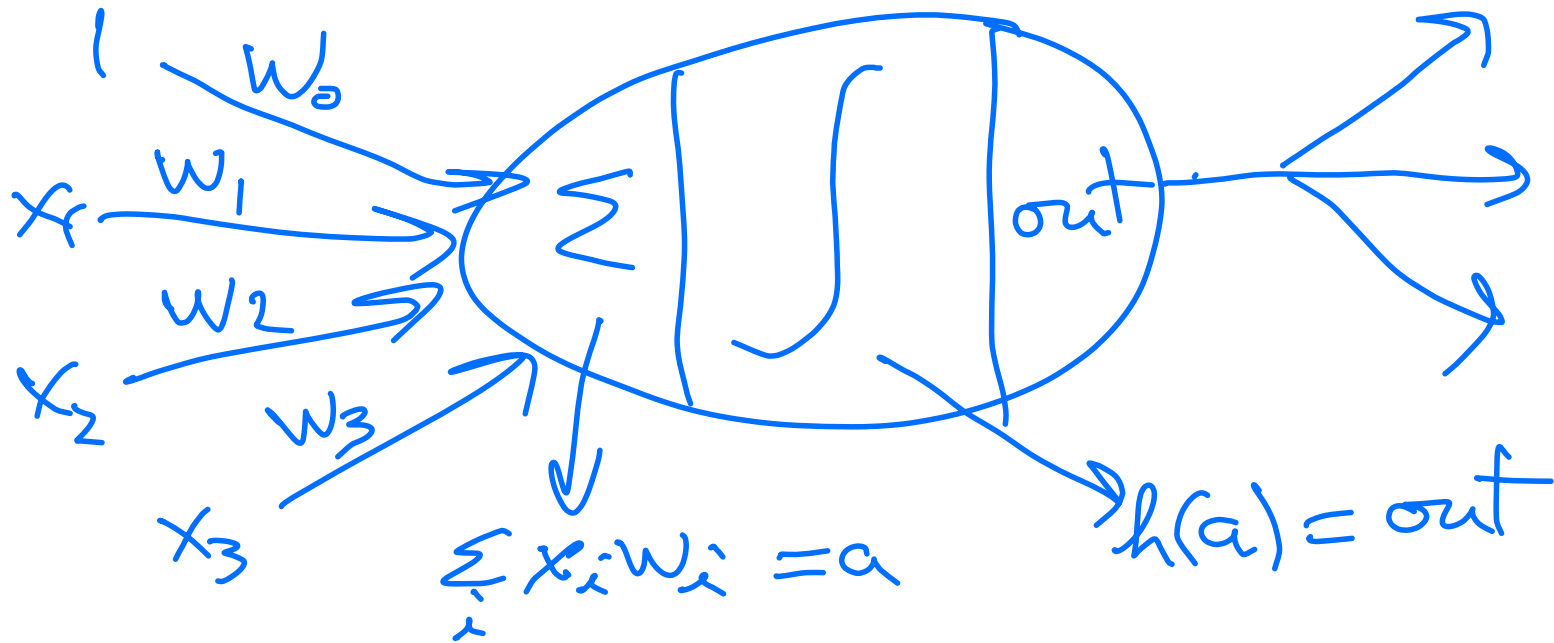
$$a_j = \sum_i W_{ji} x_i + W_{j0} = \mathbf{W}_j \bar{\mathbf{x}}$$

- **Activation function: h**

- Numerical signal produced: $y_j = h(a_j)$

ANN Unit

- Picture

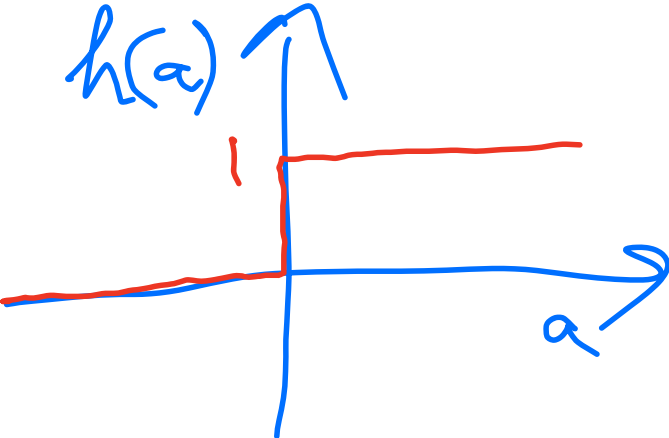


Activation Function

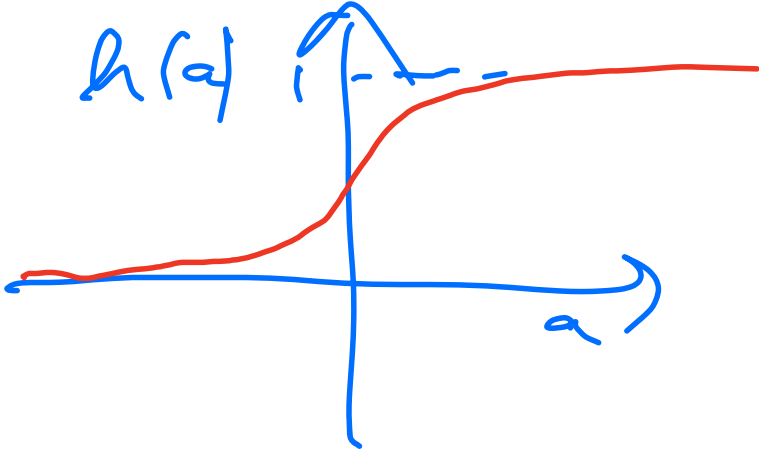
- Should be nonlinear
 - Otherwise, network is just a linear function
- Often chosen to mimic firing in neurons
 - Unit should be “active” (output near 1) when fed with the “right” inputs
 - Unit should be “inactive” (output near 0) when fed with the “wrong” inputs

Common Activation Functions

Threshold



Sigmoid



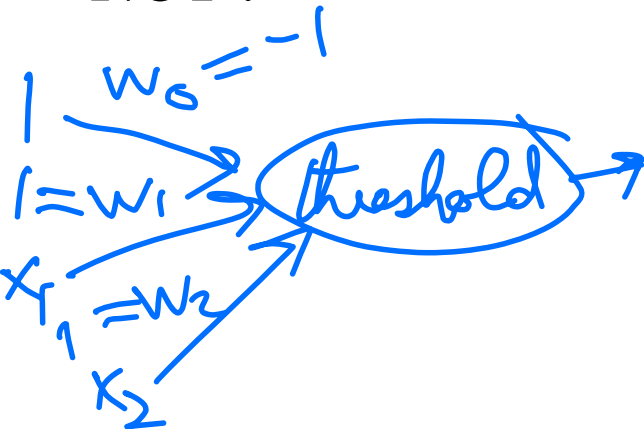
Logic Gates

- McCulloch and Pitts (1943)

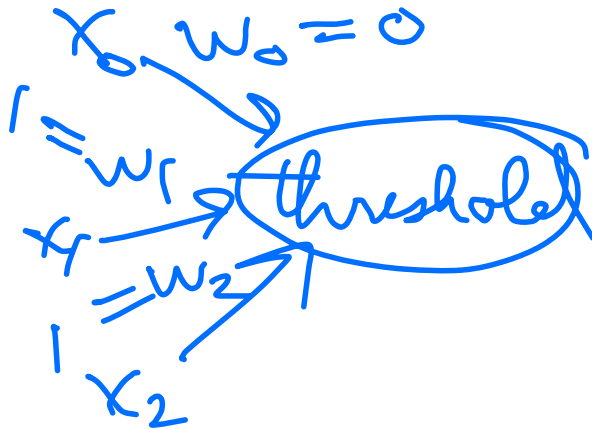
- Design ANNs to represent Boolean functions

- What should be the weights of the following units to code AND, OR, NOT ?

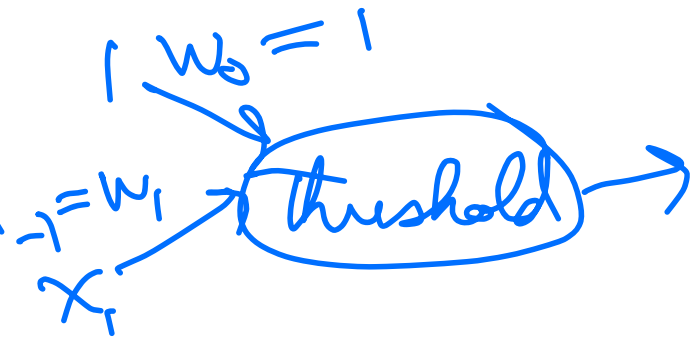
$$a = w_0 + x_1 w_1 + x_2 w_2$$
$$h(a) = \begin{cases} 0 & \text{if } a \leq 0 \\ 1 & \text{otherwise} \end{cases}$$



AND



OR



NOT

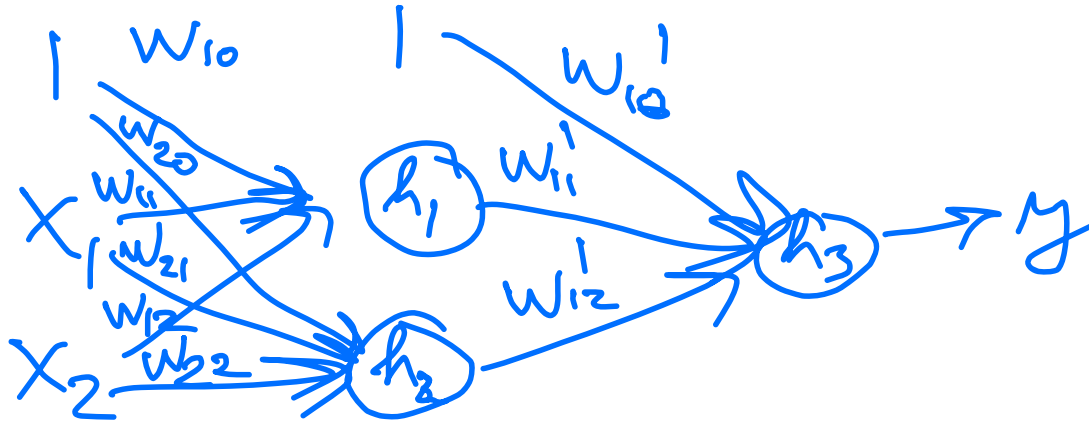
Network Structures

- **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

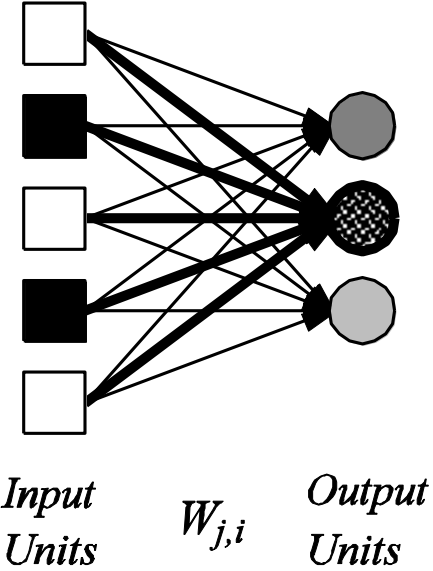
Feed-forward network

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



Perceptron

- Single layer feed-forward network



Supervised Learning

- Given list of (\mathbf{x}, \mathbf{y}) pairs
- Train feed-forward ANN
 - To compute proper outputs \mathbf{y} when fed with inputs \mathbf{x}
 - Consists of adjusting weights W_{ji}
- **Simple learning algorithm for threshold perceptrons**

Threshold Perceptron Learning

- Learning is done separately for each unit j
 - Since units do not share weights

Perceptron learning for unit j :

- For each (\mathbf{x}, y) pair do:
 - Case 1 (correct output produced): $\forall_i W_{ji} \leftarrow W_{ji}$
 - Case 2 (output produced is 0 instead of 1): $\forall_i W_{ji} \leftarrow W_{ji} + x_i$
 - Case 3 (output produced is 1 instead of 0): $\forall_i W_{ji} \leftarrow W_{ji} - x_i$
- Until correct output for all training instances

Threshold Perceptron Learning

- Dot products: $\bar{\mathbf{x}}^T \bar{\mathbf{x}} \geq 0$ and $-\bar{\mathbf{x}}^T \bar{\mathbf{x}} \leq 0$
- Perceptron computes
 - 1 when $\mathbf{w}^T \bar{\mathbf{x}} = \sum_i x_i w_i + w_0 > 0$
 - 0 when $\mathbf{w}^T \bar{\mathbf{x}} = \sum_i x_i w_i + w_0 < 0$
- If output should be 1 instead of 0 then
 - $\mathbf{w} \leftarrow \mathbf{w} + \bar{\mathbf{x}}$ since $(\mathbf{w} + \bar{\mathbf{x}})^T \bar{\mathbf{x}} \geq \mathbf{w}^T \bar{\mathbf{x}}$
- If output should be 0 instead of 1 then
 - $\mathbf{w} \leftarrow \mathbf{w} - \bar{\mathbf{x}}$ since $(\mathbf{w} - \bar{\mathbf{x}})^T \bar{\mathbf{x}} \leq \mathbf{w}^T \bar{\mathbf{x}}$

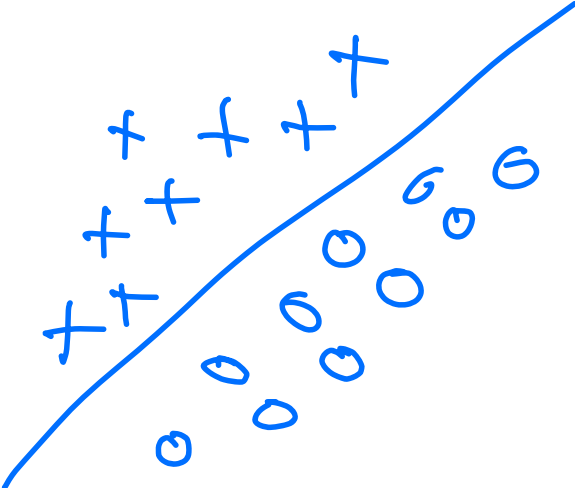
Threshold Perceptron Hypothesis Space

- Hypothesis space $h_{\mathbf{w}}$:
 - All binary classifications with parameters \mathbf{w} s.t.
$$\mathbf{w}^T \bar{\mathbf{x}} > 0 \rightarrow +1$$
$$\mathbf{w}^T \bar{\mathbf{x}} < 0 \rightarrow -1$$
- Since $\mathbf{w}^T \bar{\mathbf{x}}$ is linear in \mathbf{w} , perceptron is called a **linear separator**
- **Theorem:** Threshold perceptron learning converges if and only if the data is linearly separable

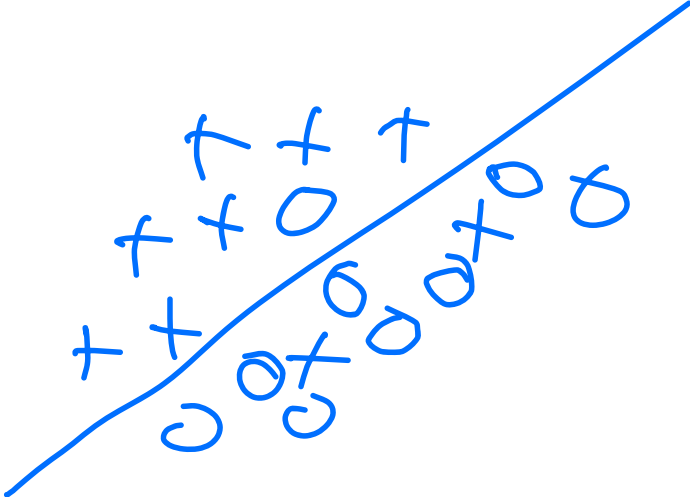
Linear Separability

- Examples:

Linearly separable

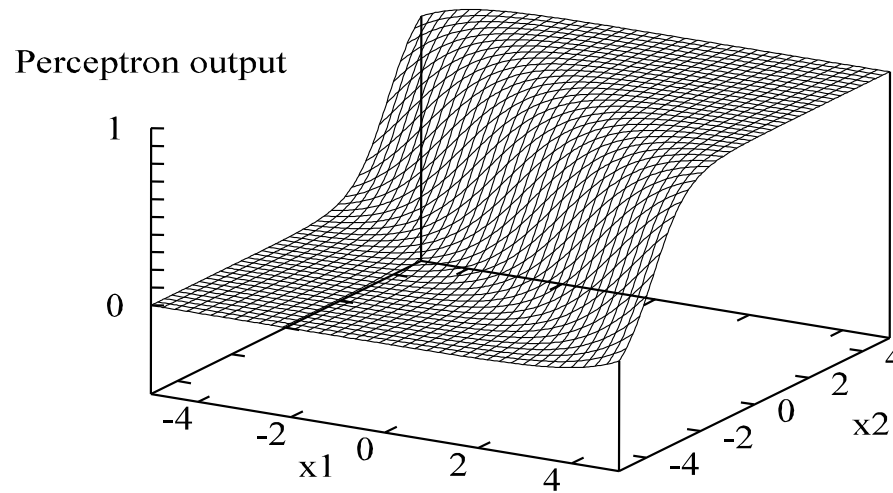


Non-linearly separable



Sigmoid Perceptron

- Represent “soft” linear separators
- **Same hypothesis space as logistic regression**



Sigmoid Perceptron Learning

- Possible objectives
 - **Minimum squared error**

$$E(\mathbf{w}) = \frac{1}{2} \sum_n E_n(\mathbf{w})^2 = \frac{1}{2} \sum_n \left(y_n - \sigma(\mathbf{w}^T \bar{\mathbf{x}}_n) \right)^2$$

- **Maximum likelihood**
 - Same algorithm as for logistic regression
- Maximum a posteriori hypothesis
- Bayesian Learning

Gradient of Squared Error

- Gradient:

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \sum_n E_n(w) \frac{\partial E_n}{\partial w_i} \\ &= - \sum_n E_n(w) \sigma'(w^T \bar{x}_n) x_i\end{aligned}$$

Recall that $\sigma' = \sigma(1 - \sigma)$

$$= - \sum_n E_n(w) \sigma(w^T \bar{x}_n) (1 - \sigma(w^T \bar{x}_n)) x_i$$

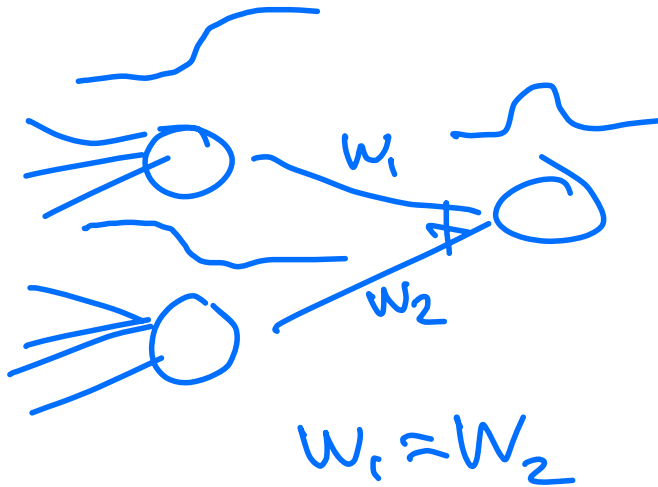
Sequential Gradient Descent

Perceptron-Learning(training_set, network)

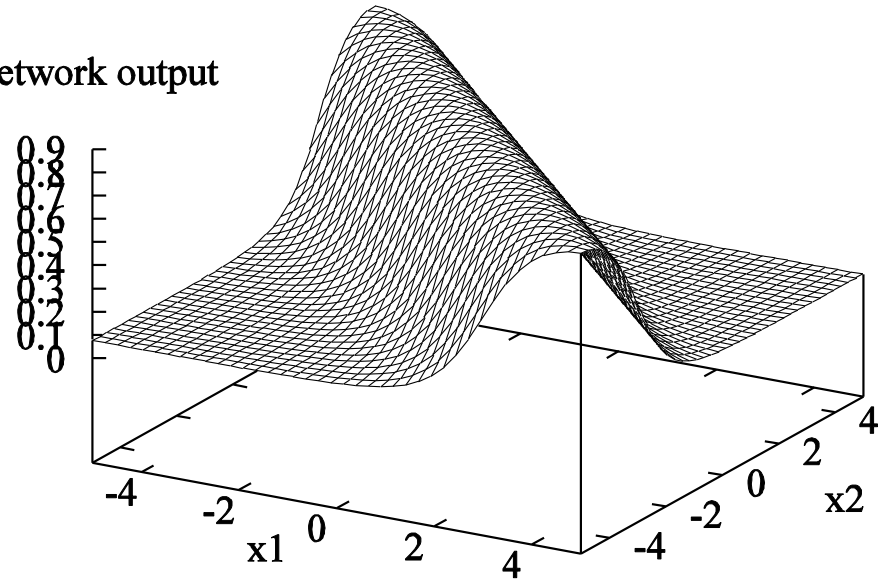
- Repeat
 - For each (\mathbf{x}_n, y_n) in training_set do
$$E_n \leftarrow y_n - \sigma(\mathbf{w}^T \bar{\mathbf{x}}_n)$$
$$\mathbf{w} \leftarrow \mathbf{w} + \eta E_n \sigma(\mathbf{w}^T \bar{\mathbf{x}}_n) (1 - \sigma(\mathbf{w}^T \bar{\mathbf{x}}_n)) \bar{\mathbf{x}}_n$$
 - Until some stopping criterion satisfied
 - Return learnt network
- N.B. η is a learning rate corresponding to the step size in gradient descent

Multilayer Networks

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

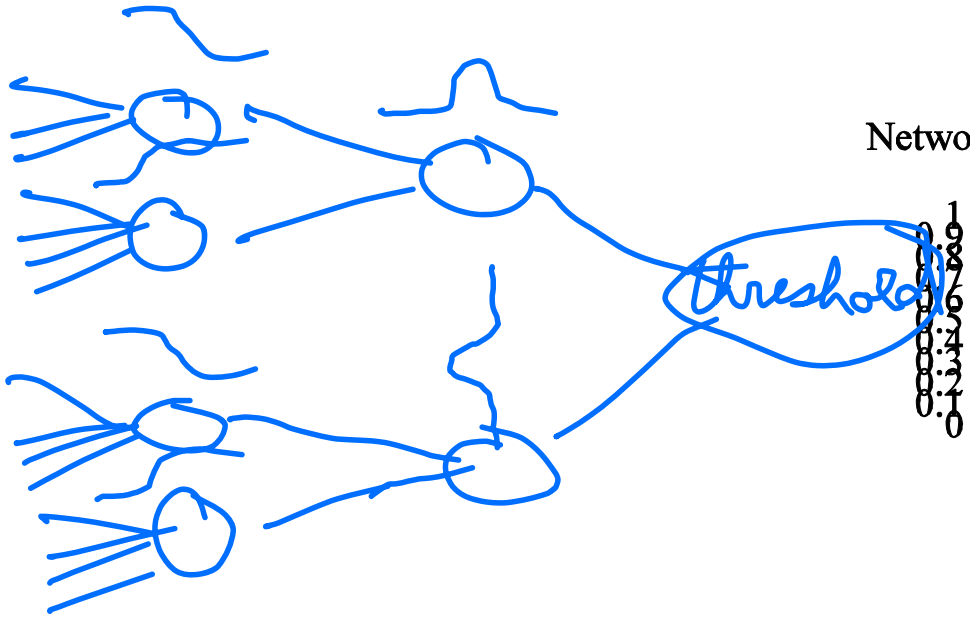


Network output

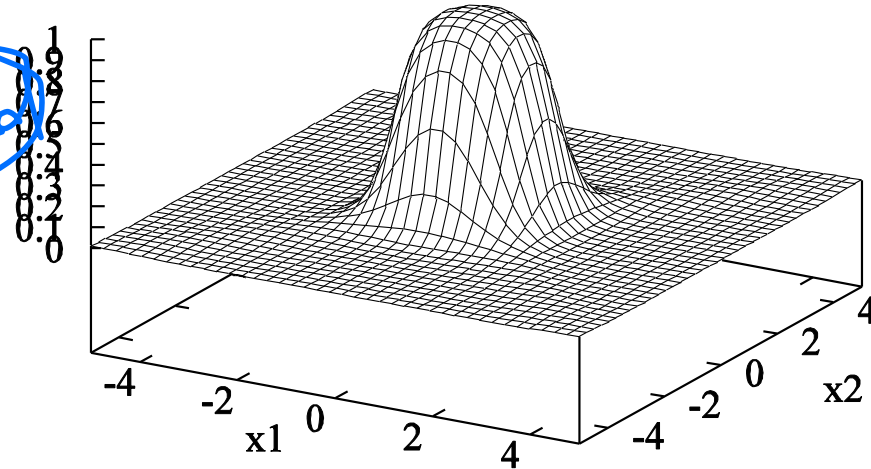


Multilayer Networks

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

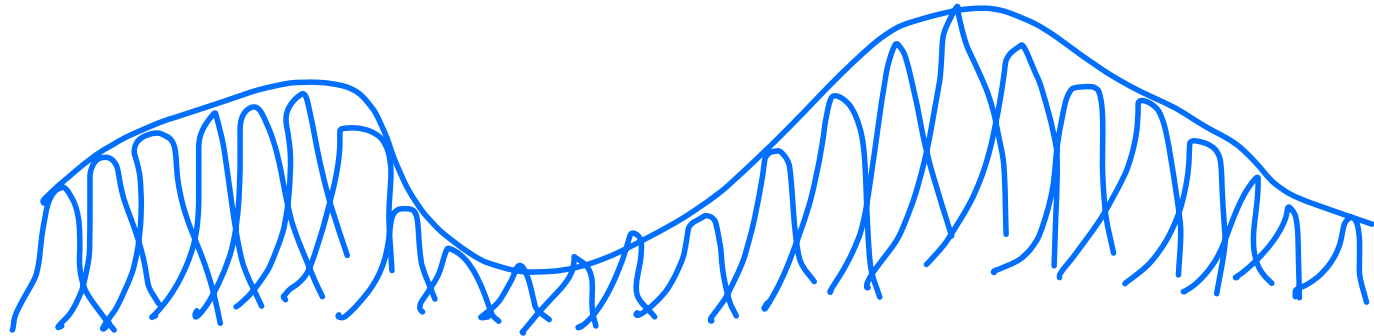


Network output



Multilayer Networks

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



- Training algorithm:
 - **Back-propagation** (gradient descent performed by propagating errors backward into the network)
 - Derivation next class

Neural Net Applications

- Neural nets can approximate any function, hence millions of applications
 - Speech recognition
 - Word embeddings
 - Machine translation
 - Vision-based object recognition
 - Vision-based autonomous driving
 - Etc.