Section 18.7 Artificial Neural Networks

CS4811 - Artificial Intelligence

Nilufer Onder
Department of Computer Science
Michigan Technological University

Outline

Brains

Regression problems

Neural network structures

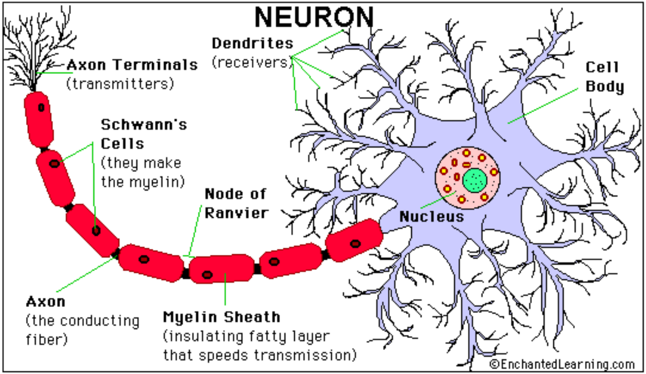
- Single-layer perceptrons
- Multilayer perceptrons (MLPs)

Back-propagation learning

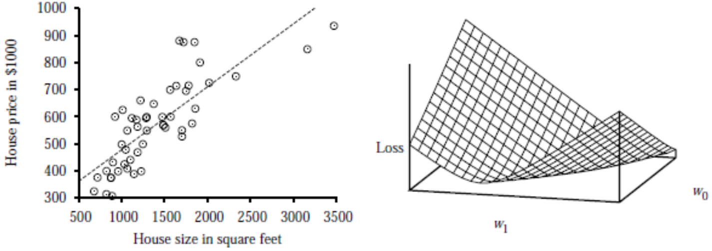
Applications of neural networks

Brains

- 10^{11} neurons of > 20 types, 1ms-10ms cycle time
- Signals are noisy "spike trains" of electrical potential



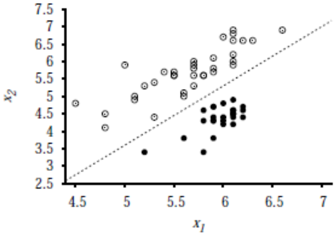
Linear regression



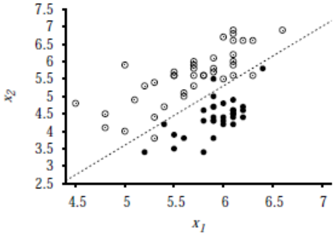
(a) (b)

- The graph in (a) shows the data points of price (y) versus floor space (x) of houses for sale in Berkeley, CA, in July 2009.
- The dotted line is a linear function hypothesis that minimizes squared error: $y = 0.232x + 246$
- The graph in (b) is the plot of the loss function $\sum_j (w_1 x_j + w_0 - y_j)^2$ for various values of w_0 and w_1.
- Note that the loss function is convex, with a single global mimimum.

Linear classifiers with a hard threshold



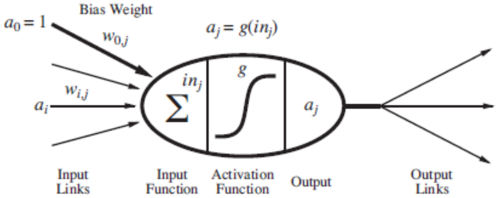
(a)



(b)

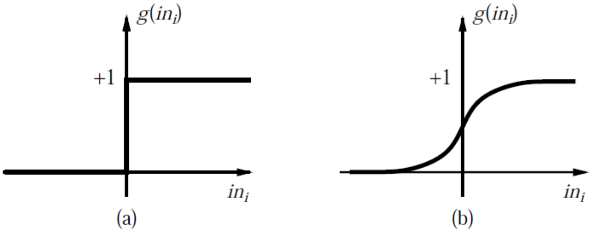
- The plots show two seismic data parameters, body wave magnitude x_1 and surface wave magnitute x_2.
- Nuclear explosions are shown as black circles. Earthquakes (not nuclear explosions) are shown as white circles.
- In graph (a), the line separates the positive and negative examples.

McCulloch-Pitts "unit"



- Output is a "squashed" linear function of the inputs
  $a_i \leftarrow g(in_i) = g\left(\sum_j W_{j,i} a_j\right)$
- It is a gross oversimplification of real neurons, but its purpose is to develop an understanding of what networks of simple units can do

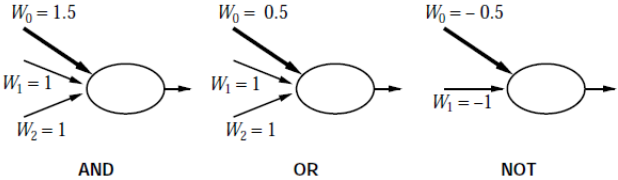
Activation functions



- (a) is a step function or threshold function
- (b) is a sigmoid function $1/(1 + e^{-x})$
- Changing the bias weight $W_{0,i}$ moves the threshold location

Implementing logical functions

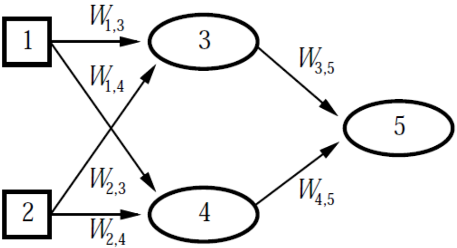
McCulloch and Pitts: every Boolean function can be implemented



Neural Network structures

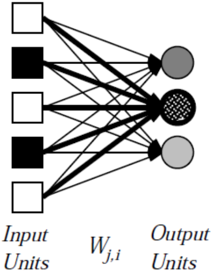
- Feed-forward networks: implement functions, no internal state
  - single-layer perceptrons
  - multi-layer perceptrons
- Recurrent networks: have directed cycles with delays, have internal state, can oscillate
  - (Hopfield networks)
  - (Boltzmann machines)

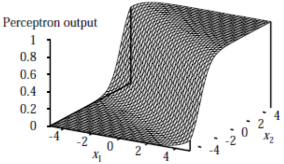
Feed-forward example



- Feed-forward network: parameterized family of nonlinear functions
- Output of unit 5 is $a_5 = g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4)$
  $= g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2))$
- Adjusting the weights changes the function:
  do learning this way!

Single-layer perceptrons

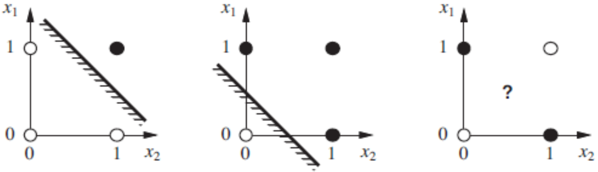




- Output units all operate separately – no shared weights
- Adjusting the weights moves the location, orientation, and steepness of cliff

Expressiveness of perceptrons

- Consider a perceptron where g is the step function (Rosenblatt, 1957, 1960)
- It can represent AND, OR, NOT, but not XOR
- Minsky & Papert (1969) pricked the neural network balloon
- A perceptron represents a *linear separator* in input space: $\sum_j W_j x_j > 0$ or $\mathbf{W} \cdot \mathbf{x} > 0$



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

Perceptron learning

- Learn by adjusting weights to reduce *error* on training set
- The squared error for an example with input x and true output y is
  $E = \frac{1}{2}Err^2 \equiv \frac{1}{2}(y - h_{\mathbf{W}}(\mathbf{x}))^2$

Perceptron learning (cont'd)

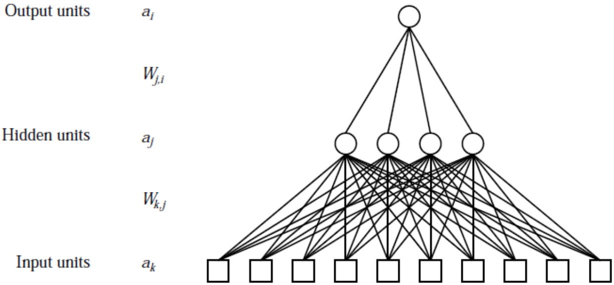
- Perform optimization search by gradient descent:

$$\begin{aligned}\frac{\partial E}{\partial W_j} &= Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j}\left(y - g(\sum_{j=0}^{n} W_j x_j)\right) \\ &= -Err \times g'(in) \times x_j\end{aligned}$$

- Simple weight update rule: $W_j \leftarrow W_j + (\alpha \times g'(in)) \times Err \times x_j$
- $Err = y - h_{\mathbf{W}} = 1 - 1 = 0 \Rightarrow$ no change
- $Err = y - h_{\mathbf{W}} = 1 - 0 = 1 \Rightarrow$ increase w_i when x_i is positive, decrease otherwise
- $Err = y - h_{\mathbf{W}} = 0 - 1 = -1 \Rightarrow$ decrease w_i when x_i is positive, decrease otherwise
- Perceptron learning rule converges to a consistent function for any linearly separable data set

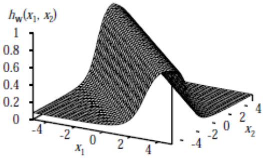
Multilayer perceptrons (MLPs)

- Layers are usually fully connected
- Numbers of *hidden units* are typically chosen by hand

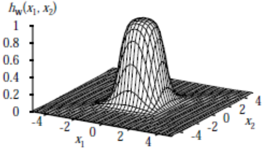


Expressiveness of MLPs

- All continuous functions with 2 layers,
  all functions with 3 layers
- Ridge: Combine two opposite-facing threshold functions
- Bump: Combine two perpendicular ridges
- Add bumps of various sizes and locations to fit any surface
- Proof requires exponentially many hidden units



(a)



(b)

Back-propagation learning

Output layer: same as for single-layer perceptron,

$$\mathbf{W}_{j,i} \leftarrow \mathbf{W}_{j,i} + \alpha \times a_j \times \Delta_i$$
where $\Delta_i = Err_i \times g'(in_i)$

Hidden layer: back-propagate the error from the output layer:

$$\Delta_j = g'(in_j) \sum_i \mathbf{w}_{j,i} \Delta_i .$$

Update rule for weights in hidden layer:

$$\mathbf{W}_{k,j} \leftarrow \mathbf{W}_{k,j} + \alpha \times a_k \times \Delta_j .$$

(Most neuroscientists deny that back-propagation occurs in the brain)

Back-propagation derivation

The squared error on a single example is defined as

$$E = \frac{1}{2}\sum_i (y_i - a_i)^2 \ ,$$

where the sum is over the nodes in the output layer.

$$\begin{aligned}
\frac{\partial E}{\partial \mathbf{W}_{j,i}} &= -(y_i - a_i)\frac{\partial a_i}{\partial \mathbf{W}_{j,i}} = -(y_i - a_i)\frac{\partial g(in_i)}{\partial \mathbf{W}_{j,i}} \\
&= -(y_i - a_i)g'(in_i)\frac{\partial in_i}{\partial \mathbf{W}ji} \\
&= -(y_i - a_i)g'(in_i)\frac{\partial}{\partial \mathbf{W}_{j,i}}\left(\sum_j \mathbf{W}_{j,i} a_j\right) \\
&= -(y_i - a_i)g'(in_i)a_j = -a_j\Delta_i
\end{aligned}$$

Back-propagation derivation (cont'd)

$$
\begin{aligned}
\frac{\partial E}{\partial \mathbf{W}_{k,j}} &= -\sum_i (y_i - a_i) \frac{\partial a_i}{\partial \mathbf{W}_{k,j}} = -\sum_i (y_i - a_i) \frac{\partial g(in_i)}{\partial \mathbf{W}_{k,j}} \\
&= -\sum_i (y_i - a_i) g'(in_i) \frac{\partial in_i}{\partial \mathbf{W}_{k,j}} = -\sum_i \Delta_i \frac{\partial}{\partial \mathbf{W}_{k,j}} \left(\sum_j \mathbf{W}_{y,i} a_j \right) \\
&= -\sum_i \Delta_i \mathbf{W}_{y,i} \frac{\partial a_j}{\partial \mathbf{W}_{k,j}} = -\sum_i \Delta_i \mathbf{W}_{y,i} \frac{\partial g(in_j)}{\partial \mathbf{W}_{k,j}} \\
&= -\sum_i \Delta_i \mathbf{W}_{y,i} g' J in_j) \frac{\partial in_j}{\partial \mathbf{W}_{k,j}} \\
&= -\sum_i \Delta_i \mathbf{W}_{y,i} g'(in_j) \frac{\partial}{\partial \mathbf{W}_{k,j}} \left(\sum_k \mathbf{W}_{k,j} a_k \right) \\
&= -\sum_i \Delta_i \mathbf{W}_{y,i} g'(in_j) a_k = -a_k \Delta_j
\end{aligned}
$$

MLP learners

- MLPs are quite good for complex pattern recognition tasks
- The resulting hypotheses cannot be understood easily
- Typical problems: slow convergence, local minima

Handwritten digit recognition

- 3-nearest-neighbor classifier (stored images) = 2.4% error
- Shape matching based on computer vision = 0.63% error
- 400-300-10 unit MLP = 1.6% error
- LeNet 768-192-30-10 unit MLP = 0.9% error
- Boosted neural network = 0.7% error
- Support vector machine = 1.1% error
- Current best: virtual support vector machine = 0.56% error
- Humans $\approx$ 0.2% error

Summary

- Brains have lots of neurons;
  each neuron $\approx$ linear–threshold unit (?)
- Perceptrons (one-layer networks) are insufficiently expressive
- Multi-layer networks are sufficiently expressive; can be trained by gradient descent, i.e., error back-propagation
- Many applications: speech, driving, handwriting, fraud detection, etc.
- Engineering, cognitive modelling, and neural system modelling subfields have largely diverged

Sources for the slides

- AIMA textbook (3^{rd} edition)
- AIMA slides:
  http://aima.cs.berkeley.edu/
- Neuron cell:
  http://www.enchantedlearning.com/subjects/anatomy/brain/Neuron.shtml
  (Accessed December 10, 2011)