Neural networks

Chapter 20, Section 5

Outline

♦ Brains
♦ Neural networks
♦ Perceptrons
♦ Multilayer perceptrons
♦ Applications of neural networks

Brains

10^{14} neurons of > 20 types, 10^{14} synapses, 1ms–10ms cycle time
Signals are noisy “spike trains” of electrical potential

McCulloch–Pitts “unit”

Output is a “squashed” linear function of the inputs:

\[ a_i \leftarrow g(\text{in}_i) = g(\sum_j W_{j,i} a_j) \]

A gross oversimplification of real neurons, but its purpose is to develop understanding of what networks of simple units can do

Activation functions

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

Implementing logical functions

McCulloch and Pitts: every Boolean function can be implemented
Network structures

Feed-forward networks:
- single-layer perceptrons
- multi-layer perceptrons

Feed-forward networks implement functions, have no internal state

Recurrent networks:
- Hopfield networks have symmetric weights ($W_{i,j} = W_{j,i}$)
  
  $g(x) = \text{sign}(x)$, $a_i = \pm 1$; holographic associative memory
- Boltzmann machines use stochastic activation functions,
  $\approx \text{MCMC in Bayes nets}$
- recurrent neural nets have directed cycles with delays
  $\Rightarrow$ have internal state (like flip-flops), can oscillate etc.

Chapter 20, Section 5

Feed-forward example

$$W_{1,3} \quad W_{1,4} \quad W_{2,3} \quad W_{2,4} \quad W_{3,5} \quad W_{4,5} \quad W_{5,3}$$

Feed-forward network = a parameterized family of nonlinear functions:

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

Adjusting weights changes the function: do learning this way!

Expressiveness of perceptrons

Consider a perceptron with $g = \text{step function}$ (Rosenblatt, 1957, 1960)

Can represent AND, OR, NOT, majority, etc., but not XOR

Represents a linear separator in input space:

$$\sum_j W_{j,i} x_j > 0 \quad \text{or} \quad W \cdot x > 0$$

Minsky & Papert (1969) pricked the neural network balloon

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

$$E = \frac{1}{2} \text{Err}^2 \equiv \frac{1}{2} (y - h_W(x))^2$$

Perform optimization search by gradient descent:

$$\frac{\partial E}{\partial W_j} = \text{Err} \times \frac{\partial \text{Err}}{\partial W_j} = \text{Err} \times \frac{\partial}{\partial W_j} (y - g(\sum_j W_{j,i} x_j))$$

$$= -\text{Err} \times g'(m) \times x_j$$

Simple weight update rule:

$$W_j \leftarrow W_j + \alpha \times \text{Err} \times g'(m) \times x_j$$

E.g., +ve error $\Rightarrow$ increase network output

$\Rightarrow$ increase weights on +ve inputs, decrease on -ve inputs

Perceptron learning contd.

Perceptron learning rule converges to a consistent function for any linearly separable data set

Perceptron learns majority function easily, DTL is hopeless

DTL learns restaurant function easily, perceptron cannot represent it

Single-layer perceptrons

Output units all operate separately—no shared weights

Adjusting weights moves the location, orientation, and steepness of cliff

Chapter 20, Section 5
**Multilayer perceptrons**

Layers are usually fully connected; numbers of hidden units typically chosen by hand

Output units \( a_i \)

\[ W_{ij} \]

Hidden units \( a_j \)

\[ W_{kj} \]

Input units \( a_k \)

**Expressiveness of MLPs**

All continuous functions w/ 2 layers, all functions w/ 3 layers

![Graph showing the expressiveness of MLPs](image)

Combine two opposite-facing threshold functions to make a ridge

Combine two perpendicular ridges to make a bump

Add bumps of various sizes and locations to fit any surface

Proof requires exponentially many hidden units (cf DTL proof)

**Back-propagation learning**

Output layer: same as for single-layer perceptron,

\[ W_{ij} \leftarrow W_{ij} + \alpha \times a_j \times \Delta_i \]

where \( \Delta_i = \text{error from the output layer} \)

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

\[ \Delta_j = g'(m_j) \sum W_{ij} \Delta_i \]

Update rule for weights in hidden layer:

\[ W_{kj} \leftarrow W_{kj} + \alpha \times a_k \times \Delta_j \]

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

**Back-propagation derivation**

The squared error on a single example is defined as

\[ E = \frac{1}{2} \sum (y_i - a_i)^2 \]

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

\[ \frac{\partial E}{\partial W_{ji}} = -(y_i - a_i) \frac{\partial a_i}{\partial W_{ji}} = -(y_i - a_i) \frac{\partial g(m_i)}{\partial W_{ji}} \]

\[ = -(y_i - a_i) g'(m_i) \frac{\partial m_i}{\partial W_{ji}} = -(y_i - a_i) g'(m_i) \frac{\partial}{\partial W_{ji}} \left( \sum W_{ij} a_j \right) \]

\[ = -(y_i - a_i) g'(m_i) a_j = -a_j \Delta_i \]

**Back-propagation derivation contd.**

\[ \frac{\partial E}{\partial W_{kj}} = \sum (y_i - a_i) \frac{\partial a_i}{\partial W_{kj}} = \sum (y_i - a_i) \frac{\partial g(m_i)}{\partial W_{kj}} \]

\[ = \sum (y_i - a_i) g'(m_i) \frac{\partial m_i}{\partial W_{kj}} = \sum (y_i - a_i) g'(m_i) \frac{\partial}{\partial W_{kj}} \left( \sum W_{ij} a_j \right) \]

\[ = \sum (y_i - a_i) g'(m_i) \frac{\partial}{\partial W_{kj}} \left( \sum W_{ij} a_j \right) \]

\[ = \sum (y_i - a_i) g'(m_i) a_j = -a_j \Delta_i \]

**Back-propagation learning contd.**

At each epoch, sum gradient updates for all examples and apply

Training curve for 100 restaurant examples: finds exact fit

Typical problems: slow convergence, local minima
Back-propagation learning contd.

Learning curve for MLP with 4 hidden units:

MLPs are quite good for complex pattern recognition tasks, but resulting hypotheses cannot be understood easily.

Handwritten digit recognition

3-nearest-neighbor = 2.4% error
400–300–10 unit MLP = 1.6% error
LeNet: 768–192–30–10 unit MLP = 0.9% error
Current best (kernel machines, vision algorithms) ≈ 0.6% error

Summary

Most brains have lots of neurons; each neuron ≈ linear–threshold unit (?)
Perceptrons (one-layer networks) 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