Neural Networks

Computer Science 367
Patricia J Riddle
Some History of Neural Networks

McCulloch and Pitts [1943]: Model of artificial neurons

Hebb [1949]: Simple updating rule

Minsky and Edmonds [1951]: First neural network computer

Rosenblatt [1962]: Perceptrons (the model)

Minsky and Papert [1969]: Perceptrons (the book)
Revival of Neural Networks

Recently, there has been a resurgence of interest in neural networks for the following reasons:

- Faster digital computers to simulate larger networks
- Interest in building massively parallel computers
- New neural network architectures
- Powerful Learning algorithms
Characteristics of Neural Networks

A large number of very simple neuronlike processing elements

A large number of weighted connections between the elements

Highly parallel, distributed control

An emphasis on learning internal representations automatically
The 100-Time-Steps Requirement

Individual neurons are extremely slow devices (compared to digital computers), operating in the millisecond range.

Yet, humans can perform extremely complex tasks in just a tenth of a second.

This means, humans do in about a hundred steps what current computers cannot do in 10 million steps.

Look for massively parallel algorithms that require no more than 100 time steps.
Failure Tolerance

On the one hand, neurons are constantly dying, and their firing patterns are irregular.

On the other hand, components in digital computers must operate perfectly.

With current technology, it is:

- Easier to build a billion-component IC in which 95% of the components work correctly.
- More difficult to build a million-component IC that functions perfectly.
Fuzziness

People seem to be able to do better than computers in fuzzy situations.

We have very large memories of visual, auditory, and problem-solving episodes.

Key operation in solving new problems is finding closest matches to old situations.
Hopfield Networks

Theory of memory

Hopfield introduced this type of neural network as a theory of memory (1982).

Distributed representation

A memory is stored as a pattern of activation across a set of processing elements.

Furthermore, memories can be superimposed on one another; different memories are represented by different patterns over the same set of processing elements.
Hopfield Networks (cont’d)

Distributed, asynchronous control
Each processing element makes decisions based only on its own local situation. All the local actions add up to a global solution.

Content-addressable memory
A number of patterns can be stored in a network. To retrieve a pattern, we need only specify a portion of it. The network automatically finds the closest match.

Fault tolerance
If a few processing elements misbehave or fail completely, the network will still function properly.
Technical Details of Hopfield Networks

Processing elements (units) are either in state active (1) or passive (-1).

Units are connected to each other with weighted, symmetric connections (recurrent network).

A positively (negatively) weighted connection indicates that the two units tend to activate (deactivate) each other.
Parallel Relaxation in Hopfield Networks

A random unit is chosen.

If any of its neighbors are active, the unit computes the sum of the weights on the connections to those active neighbors.

If the sum is positive, the unit becomes active; otherwise it becomes inactive.

The process (parallel relaxation) is repeated until the network reaches a stable state.
Example of a Hopfield Network
Another Example

Initial state

Stable State
Stability

Given any set of weights and any initial state, parallel relaxation eventually steers the network into a stable state.

It will only stabilize if parallel relaxation is used (asynchronous) (it is also important that the arcs are bidirectional)

If a synchronous update is done then it will either stabilize or oscillate between two (and only two) units.
Some Features of Hopfield Networks

The network can be used as a **content-addressable memory** by setting the activities of the units to correspond to a **partial pattern**. To retrieve the pattern, we need only supply a portion of it.

**Parallel relaxation** is nothing more than **search**, albeit of a different style. The stable states correspond to local minima in the search space.

The network **corrects errors**: if the initial state contains inconsistencies, the network will settle into the solution that violates the fewest constraints offered by the inputs.
Perceptrons

This type of network was invented by Rosenblatt [1962].

A perceptron models a neuron by taking a **weighted sum** of its inputs and sending the output 1 if the sum is greater than or equal to some **adjustable threshold value**; otherwise it sends 0.

The **connections** in a perceptron, unlike in a Hopfield network, are **unidirectional** (feedforward network).

**Learning** in perceptrons means adjusting the **weights** and the **threshold**.

A perceptron computes a binary function of its input.

Perceptrons can be combined to compute more complex functions.
A Perceptron

\[ o = \begin{cases} 
1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\
-1 & \text{otherwise} 
\end{cases} \]

**FIGURE 4.2**
A perceptron.
Activation Function

Input:
\[ \mathbf{x} = (x_1, \ldots, x_n) \quad x_0 = 1 \]

Output with explicit threshold:
\[
g(\mathbf{x}) = \begin{cases} 
1 & \text{if } \sum_{i=1}^{n} w_i x_i \geq t \\
0 & \text{otherwise}
\end{cases}
\]

Output with implicit threshold:
\[
g(\mathbf{x}) = \begin{cases} 
1 & \text{if } \sum_{i=0}^{n} w_i x_i \geq 0 \\
0 & \text{otherwise}
\end{cases}
\]

\[ t = -w_0 \times x_0 = -w_0 \times 1 \]
What Perceptrons Can Represent

Linearly Separable Function

Input: \((x_1, x_2)\)

Output: \(g(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2\)

Decision Surface: \(g(x_1, x_2) = 0 \iff x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}\)
Limitations of Perceptrons

If a decision surface does not exist, the perceptron cannot learn the function.

An example is the XOR function:
Perceptron Learning Method

Start with randomly assigned weights.

For each example $\vec{x}$ do:

- Let $o$ be the computed output $g(\vec{x})$
- Let $t$ be the expected (target) output.
- Update the weights based on $\vec{x}$, $o$, and $t$.

Repeat the process (i.e., go through another epoch) until all examples are correctly predicted or the stopping criterion is reached.
Updating Rule

The error is the difference between the expected output and the computed output:

\[ err = t - o \]

If the error is positive (negative), \( o \) must be increased (decreased).

Each input \( x_i \) contributes \( w_i x_i \) to the total input.

If \( x_i \) is positive (negative), increasing \( w_i \) will increase (decrease) \( o \).

The desired effect can be achieved with the following rule (\( \eta \) is the learning rate):

\[ w_i \leftarrow w_i + \eta \cdot x_i \cdot err \]
Multilayer Feed-Forward Networks

Input units are connected to hidden units.

Hidden units are connected to other hidden units.

Hidden units are connected to output units.
Example of a Two Layer Feed-Forward Network

Output units $O_i$

$W_{j,i}$

Hidden units $a_j$

$W_{k,j}$

Input units $I_k$
The Idea of Back-Propagation Learning

Compute the output for a given input and compare it with the expected output.

Assess the blame for an error and divide it among the contributing weights.

Start with the second layer (hidden units to output units) and then continue with the first layer (input units to hidden units).

Repeat this for all examples and for as many epochs as it takes for the network to converge.
Activation Function

Backpropagation requires the derivative of the activation function $g$.

The sign function (used in Hopfield networks) and the step function (used in Perceptrons) are not differentiable.

Usually, backpropagation networks use the sigmoid function:

![Sigmoid Function Graph]
Sign And Step Functions

Sign Function

Step Function

9/30/15 367-ANN
Sigmoid Unit

\[ o = \sigma(\text{net}) = \frac{1}{1 + e^{-\text{net}}} \]

\[ \text{net} = \sum_{i=0}^{n} w_i x_i \]
Backpropagation Update Rules
(2\textsuperscript{nd} Layer)

Let $Err_k$ be the error $(t_k - o_k)$ at the output node.

Let $in_k$ be the weighted sum $\sum_k w_{k,i} x_k$ of inputs to unit $k$.

Let $\Delta_k$ be the new error term $Err_k g'(in_k)$.

Then the weights in the second layer are updated as follows:

$$w_{j,k} \leftarrow w_{j,k} + \eta \cdot x_j \cdot \Delta_k$$
Backpropagation Update Rules
(1\textsuperscript{st} Layer)

Let $\Delta_j$ be the new error term for the first layer:

$$\Delta_j = g'(in_j) \sum_k w_{j,k} \Delta_k$$

Then the weights in the first layer are updated as follows:

$$w_{i,j} \leftarrow w_{i,j} + \eta \cdot x_i \cdot \Delta_j$$
Differences from the book

I use \( \mu \) instead of \( \alpha \) (for learning rate)

I use \( x \) instead of \( a \) (for data values)
Pros and Cons of Backpropagation

Cons

Backpropagation might get stuck in a local minimum that does not solve the problem.
Even for simple problems like the XOR problem, the speed of learning is slow.

Pros

Fortunately, Backpropagation does not get stuck very often,
Backpropagation is inherently a parallel, distributed algorithm.
What is the ANN hypothesis Space?
What is the ANN hypothesis Space?

- The N-dimensional of real numbers

\[ w_i = <0.8, 0.7, 0.2> \]

goes to

\[ w_{i+1} = <0.6, 0.8, 0.2> \]
Hypothesis Space

Every possible assignment of network weights represents a syntactically different hypothesis.

N-dimensional Euclidean space of the n network weights.
This hypothesis space is continuous.

Since E is differentiable with respect to the continuous parameters, we have a well-defined error gradient.
Inductive Bias

Inductive Bias depends on interplay between gradient descent search and the way the weight space spans the space of representable functions.

Roughly - smooth interpolation between data points

Given two positive training instances with no negatives between them, Backpropagation will tend to label the points between as positive.
Multilayer Networks and Nonlinear Surfaces

FIGURE 4.5
Hidden Layer Representations

Backpropagation can discover useful intermediate representations at the hidden unit layers.

It is a way to make implicit concepts explicit.

Discovering binary encoding
Backprop in action

<table>
<thead>
<tr>
<th>Input</th>
<th>Hidden Values</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000000</td>
<td>.89 .04 .08</td>
<td>10000000</td>
</tr>
<tr>
<td>01000000</td>
<td>.15 .99 .99</td>
<td>01000000</td>
</tr>
<tr>
<td>00100000</td>
<td>.01 .97 .27</td>
<td>00100000</td>
</tr>
<tr>
<td>00010000</td>
<td>.99 .97 .71</td>
<td>00010000</td>
</tr>
<tr>
<td>00001000</td>
<td>.03 .05 .02</td>
<td>00001000</td>
</tr>
<tr>
<td>00000100</td>
<td>.01 .11 .88</td>
<td>00000100</td>
</tr>
<tr>
<td>00000010</td>
<td>.80 .01 .98</td>
<td>00000010</td>
</tr>
<tr>
<td>00000001</td>
<td>.60 .94 .01</td>
<td>00000001</td>
</tr>
</tbody>
</table>
ALVINN’ s ANN
Learned Hidden Representations

30 x 32 resolution input images

Network weights after 1 iteration through each training example

Network weights after 100 iterations through each training example
Error Plots

Error versus weight updates (example 1)

Error versus weight updates (example 2)