**Perceptrons**

**and**

**Multi layered Perceptrons**

Single Neuron Modeling

Action Potential generation

Hodgkin-Huxley Model

More

Detailed

Less

Detailed

Biophysical Modeling Dendritic Processing Axonal Processing

Synaptic Transmission

Simplified Neuron Models FitzHugh-Nagumo model Morris-Lecar Model Izhikevich Models

Integrate and fire neuron Dynamic Binary Neuron McCulloch-Pitts neuron

Single Neuron Modeling

Action Potential generation

Hodgkin-Huxley Model

More

Detailed

Less

Detailed

Biophysical Modeling Dendritic Processing Axonal Processing

Synaptic Transmission

Simplified Neuron Models FitzHugh-Nagumo model Morris-Lecar Model Izhikevich Models

Integrate and fire neuron Dynamic Binary Neuron **McCulloch-Pitts neuron**

**Perceptrons**

MP neuron

• McCulloch-Pitts neuron model (1943) takes inputs from many neurons and produces a single output.

• If the net effect of the external inputs is greater than a threshold, the neuron goes into excited state (1), else it remains in its resting state (0).

Basic Neuron Model

w1

x1

y

xi

*n*

∑

= −

*y g wi xi b*

( )

xn

g(h)

0

*i*

org(h)

=

1

MP neuron model

• McCulloch and Pitts thought that brain works like a computer and neurons are like logic gates • Since the MP neurons are binary units it seemed worthwhile to check if the basic logical operations can be performed by these neurons. • McCulloch and Pitts quickly showed that the MP neuron can implement the basic logic gates AND, OR and NOT simply by proper choice of the weights:

A Theory of how the Brain doesn’t work

• McCulloch-Pitts neurons can implement logic

gates.

• OR Gate: y = g (x1 + x2 – 0.5) table: • AND Gate: y = g(x1 + x2 – 1.5) table: • NOT Gate: y = g(-x + 0.5) table:

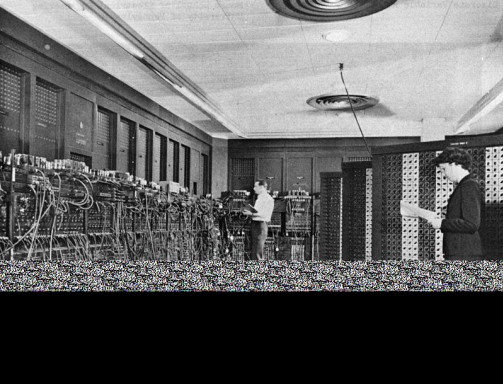
| X1 | X2 | Y |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| X1 | X2 | Y |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| X | Y |
| --- | --- |
| 0 | 1 |
| 1 | 0 |

• Hence Brain is a logical machine. (Wrong!)

ENIAC (1945)

(**Electronic Numerical Integrator and Computer**)https://en.wikipedia.org/wiki/ENIAC

• The idea of considering neurons as logic gates and the brain itself as a large Boolean circuit *does not* satisfy other important requirements of a good theory of the brain.

The Brain and the Computer

• Computer Brain

• Rigid inorganic 2D sheets matter Soft organic 3D tissue • Powered by DC mains Powered by ATP • Signals: 5volt DC 100 mV pulsed • Extreme sensitivity to faults Very fault tolerant, • Clock cycle: O(10-9) sec 1 mS (200-300Hz) • Centralized clock No centralized clock • Power dissipation: 10-5 watts 10-12 watts

The Perceptron – A NN that learns (1960’s)

• Learns to recognize simple image patterns • Uses a single layer of MP neurons

A

• Limitations:

– Can classify only “linearly separable” patterns.

Weight ‘W’ of ‘m\*n’

dimension

Bias ‘b’

X (to each neuron)

y=actual output (‘n’ dimension)

(‘m’ dimension)

.

.

.

.

‘n’ neurons

MP Neuron divides the input space into two semi-infinite halves∑*w x b* − =

*y g w x b* = − ∑0 *i i*

( ) *i i*

*i*

*i*

*y* =1

*y* = 0

**AN EXAMPLE OF HOW A PERCEPTRON LEARNS**

Function to be learnt: OR gate

| **X1**  0  0 | **X2**  0  1 | **d**  0  1 |
| --- | --- | --- |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Presenting (0,0): wrongly classified

Presenting (0,1): correctly classified

Presenting (1,0): correctly classified

Presenting (1,1): correctly classified



3 out of 4 correctly classified



Presenting (0,0): correctly classified



Presenting (0,1): correctly classified



Presenting (1,0): correctly classified



Presenting (1,1): correctly classified



. When the training data is linearly separable, there can be an infinite number of solutions.

|  |
| --- |

• In other words, a

Perceptron classifies input patterns by

dividing the input space into two semi-infinite regions using a

hyperplane.

|  |
| --- |

Perceptron Learning Rule:

• It is also called the LMS Rule or Delta Rule or Widrow-Hoff Rule.

• The steps involved in Perceptron learning are as follows:

1. Initialization of weights: Set the initial values of the weights to 0. **w**(0) = 0.

2. Present the pth training pattern, x, and calculate the network output, y.

3. Using the desired output, d, for the input pattern x, adjust the weights by a small amount using the following learning rule:

*w t w t d t y t x t*

( 1) ( ) [ ( ) ( )] ( )

+ = + − η

*where*

*b*(*t* +1) = *b*(*t*) −η[*d*(*t*) − *y*(*t*)]

,

*d n x t C*

( ) 1, ( ) 1

= + ∈

*d n x t C*

( ) 1, ( ) 2

= − ∈

4. Go back to step 2 and continue until the network output error, e = d-y, is 0 for all patterns.

• The above training process will converge after Nmax iterations where,

*w x n*

α

=

min *x n C*

( ) 1 ∈

*T* 0

( )

β

=

max *x k C*

( ) 1

∈

*x k* ( )

2

*N w*

β α

=

max 0

2

/

• See (Haykin 1999, Chapter 3, Section 3.9) for proof of convergence.

• Range of η :

0 < η ≤ 1

• Averaging of past inputs leads to stable weight dynamics, which requires small η.

• Fast adaptation requires large η.

• Learning rule can also be derived from an error

function:

Δ = − ∇ *w E* η *w i*

η∂

*E*

∂ ∂

Δ = −∂

*ww i*

*E y d y d y g x*

= − − = − −

[ ] [ ] ' *i*

*w w*

*i i*

where E denotes the squared error over all patterns. The learning rule may be derived by performing gradient descent over the error function.

• The last term in the above equation has g’, which is zero everywhere except at the origin if g is a hardlimiting nonlinearity.

• But if we take a smoother version of g(), which saturate at +1 and -1, like the tanh() function, the learning rule becomes,

[ ] ' Δ = − *w d y g x i i* η

• Since g’ > 0 always for tanh() function, we can absorb it into h, considering it as a quantity that varies with x. We then have,

'[ ] Δ = − *w d y x i i* η

• Which is identical to the Perceptron learning rule given before above.

A not-very-constructive

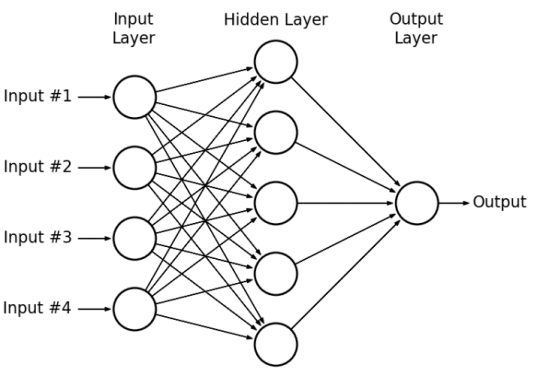
Criticism

• Minsky and Papert on Perceptron: – A single layer perceptron is so limited in its capabilities

– It may not be worth studying its multilayer counterparts.

**The Multi-layered Perceptron**

• Improvements over Perceptron: – Smooth nonlinearity – sigmoid – 1 or more hidden layers



The Hidden Layer

• The perceptron, which has no hidden layers, can classify only linearly separable patterns. • The MLP, with at least 1 hidden layer can classify *any* linearly non-separable classes also.

• It can easily be shown that the XOR problem which was not solvable by a Perceptron can be solved by a MLP with a single hidden layer containing two neurons.

Multi Layer Perceptron (MLP)

• Neurons are organized as layers

– Input, output and hidden layer(s)

• Feedforward NN: Flow of info from input to output layer

XY

V1 V2 x1 x2

MLP solves the EXOR problem 1 1 2 *V g x x* = + − ( 0.5)

2 1 2 *V g x x* = + − ( 1.5)

1 2 *y g V V* = − − ( 0.5)

MLP can model an XOR gate

V1 V2 y=1

y=1

V1-V2 y=0

y=0

x1+x2-0.5=0x1+x2-1.5=0 y=0

1 2 *y g V V* = − − ( 0.5)

XOR gate!

y=1

y=0

MLP

• An MLP can approximate any continuous multivariate function to any degree of accuracy, provided there are sufficiently many hidden neurons (Cybenko, 1988; Hornik et al, 1989). A more precise formulation is given below.

• A serious limitation disappears suddenly by adding a single hidden layer.

All-powerful MLP

• MLP is a *universal approximator*

• “An MLP with a single hidden layer can approximate any continuous I/O function with arbitrary accuracy, over a finite input domain, given enough number of nodes in the hidden layer.” (Cybenko, 1988)

NN as an I/O Machine

•NN is a parametric model that can model arbitrary I/O relationships

X Y

Learns from Experience

• NN can learn the unknown I/O relationship from sample data

Input

X

Target Output T

NN

+- Error

T-Y

Actual Output Y

NN vs. Algorithmic Approach

• Traditional or Algorithmic Approach – Study and find out the steps to produce a final output from a given input

• NN Approach:

– Learn from Experience

Training the hidden layer

• Not obvious how to train the hidden layer parameters. • The error term is meaningful only to the weights connected to the output layer – *credit assignment* problem.

• In a large network with many layers, this implies that information is exchanged over distant elements of the network though they are not directly connected.

• Such an algorithm may be mathematically valid, but is biologically unrealistic. (locality rule is violated)

**The Backpropagation Algorithm**

• **History:**

• Fist described by Paul Werbos (1974) in his PhD thesis at MIT. • Rediscovered by Rumelhart, McClelland and Williams (1986) • Also discovered by Parker (1985) and LeCun (1985) in the same year.

• As in Perceptron, this training algorithm involves 2 passes: The forward pass – outputs of various layers are computed The backward pass – weight corrections are computed

Back Propagation Algorithm

• Consider a simple 3-layer network with a single neuron in each layer.

*E E* = ∑

Total output error over all patterns:*p*

*p*

Squared Output error for the p’th pattern: 1 2

*E e* = ∑

*p i*

2

*i*

Output error for the p’th pattern: ei = di - yi Network output: ( )*s*

*i i y g h* =

*h w V* = − ∑ θ

Net input to the output layer:*s s s i ij j i*

*j*

Output of the hidden layer: ( ) *f f V g h j j*

=

*h w x* = − ∑ θ

Net input of the hidden layer:*f f f j jk k j*

*k*

BPA

• Update rule for the weights using gradient descent:

η∂

*E*

∂

*E*

Δ = −∂;*s p s p*

*ww*

*ij s ij*

η∂

*E*

Δ = −∂

θ ηθ

*i s i*

∂

*E*

Δ = −∂;*f p f p*

*ww*

*jk f*

*jk*

**Updating** *s*

*wij***:**

∂ ∂ ∂ ∂ ∂

Δ = −∂

θ ηθ

*j f j*

*E E e y h s*

*p p i i i*

=

*s s s* ∂ ∂ ∂ ∂ ∂

*w e y h w ij i i i ij* ( 1) '( )*si j*

= − *e g h V*

• **weight correction = (learning rate) \* (local** δ **from ‘top’) \* (activation from ‘bottom’)**

Training MLP

• **Backpropagation**

Y Error, δ=T-Y

δi

wij

Vj

X

Δ wij = η δi Vj

Training and Testing a MLP

**Traning:**

• Randomly initialize weights.

• Train network using backprop eqns. • Stop training when error is sufficiently low and freeze the weights.

**Testing**

• Start using the network.

Merits of MLP

**Merits of MLP trained by BP:**

• A general solution to a large class of problems. • With sufficient number of hidden layer nodes, MLP can approximate arbitrary target functions.

• Backprop applies for arbitrary number of layers, partial connectivity (no loops).

• Training is local both in time and space – parallel implementation made easy.

• Hidden units act as “feature detectors.”

• Good when no model is available

Demerits of MLP by BPA

**Problems with MLP trained by BP**: • Blackbox approach

• Limits of generalization not clear • Hard to incorporate prior knowledge of the model into the network

• slow training

• local minima

Development of an NN Application

• The Stages

– Feasibility study

– Data collection

– Designing/Training the Network

– Testing the Network

– Deployment...

Feasibility Study

• Do you really need a NN?

• Is it an I/O problem?

• What are the inputs and the outputs? • Are inputs and outputs uniquely related? • Do I have enough domain knowledge? • Do I have enough Data?

Domain Knowledge

• Can never overemphasize the value of Domain Knowledge

• Domain Knowledge is used to -

– Breaking up the problem

– Map problem onto NN(s)

– Data collection

– Data preprocessing