BITS-F459: Computer Vision

07 Perceptron Introduction

Dr. Kamlesh Tiwari
Associate Professor, Department of CSIS,
BITS Pilani, Pilani Campus, Rajasthan-333031 INDIA

Jan 24, 2025 ON-CAMPUS NAB 6108 @ BITS-Pilani [Jan-May 2025]

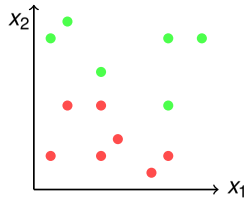
http://ktiwari.in/cv

Linear Classification

Consider Following data

x_1	x_2	y
1	9	green
10	9	green
4	7	green
4	5	red
5	3	red
8	9	green
4	2	red
2	5	red
7	1	red
2	10	green
8	5	green
1	2	red
8	2	red

Data is in 2D, so let us visualize



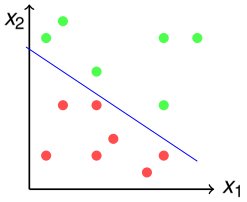
- Data looks linearly separable
- What is the decision boundary?

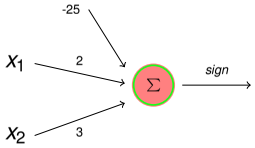
Many Possibilities, such as

if $(2x_1 + 3x_2 - 25 > 0)$ it is green
otherwise red

What about this arrangement?

With chosen *decision boundary* $2x_1 + 3x_2 - 25 = 0$





- This illustration is called as **perceptron**
- Provides a graphical way to represent the linear boundary
- Values 3, 2, -25 are its parameters or weights

Given a data

"How to find appropriate parameters?" is an important issue

Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

Algorithm 1: Perceptron training rule

```
Begin with random weights w
repeat
    for each misclassified example do
        w_i = w_i + η(t − o)x_i
until all training examples are correctly classified;
return w
```

- **Why would this strategy converge?**
  1. Weight does not change when classification is correct
  2. If perceptron outputs -1 when target is +1: weight increases ↑
  3. If perceptron outputs +1 when target is -1: weight decreases ↓

Conversion with perceptron training rule is subject to linear separability of training example and appropriate η

Example

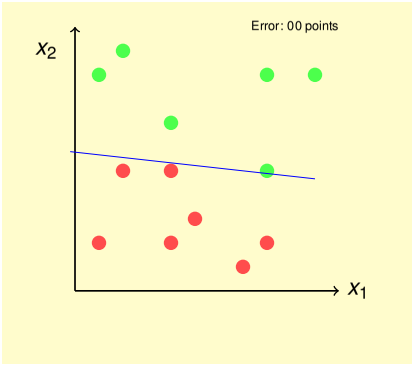
Consider the same data

x_1	x_2	y
1	9	green
10	9	green
4	7	green
4	5	red
5	3	red
8	9	green
4	2	red
2	5	red
7	1	red
2	10	green
8	5	green
1	2	red
8	2	red

$\eta = 0.01$

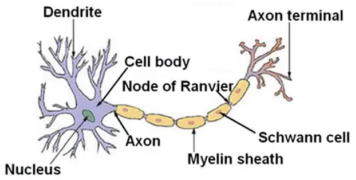
w0=0.500, w1=0.500, w2=0.500	err=7
w0=0.360, w1=-0.120, w2=0.100	err=6
w0=0.300, w1=-0.180, w2=0.060	err=5
w0=0.240, w1=-0.140, w2=0.140	err=4
w0=0.180, w1=-0.200, w2=0.100	err=5
w0=0.120, w1=-0.160, w2=0.180	err=4
w0=0.080, w1=-0.060, w2=0.180	err=5
w0=0.020, w1=-0.120, w2=0.140	err=4
w0=-0.040, w1=-0.180, w2=0.100	err=5
w0=-0.100, w1=-0.140, w2=0.180	err=4
w0=-0.140, w1=-0.040, w2=0.180	err=5
w0=-0.200, w1=-0.100, w2=0.140	err=3
w0=-0.260, w1=-0.160, w2=0.100	err=4
w0=-0.320, w1=-0.120, w2=0.180	err=3
w0=-0.360, w1=-0.020, w2=0.180	err=3
w0=-0.420, w1=-0.080, w2=0.140	err=2
w0=-0.420, w1=-0.080, w2=0.240	err=2
Fourteen more iterations	
w0=-0.900, w1=-0.020, w2=0.180	err=1
w0=-0.900, w1=-0.020, w2=0.240	err=2
w0=-0.920, w1=0.020, w2=0.220	err=2
w0=-0.960, w1=-0.020, w2=0.220	err=3
w0=-0.980, w1=0.020, w2=0.200	err=2
w0=-1.000, w1=0.060, w2=0.180	err=2
w0=-1.040, w1=0.020, w2=0.180	err=0

Visual Interpretation



- Conversion is not gradual. (Error is NOT reducing monotonically)
- It is difficult to decide when to stop if data is not linearly separable

Neural Network (NN)



- Cell, Axon, Synopses, Molicules, and Dendrites
- Humans have 10^{11} neurons, each connected to 10^4 others, switches in 10^{-3} sec

NN is biologically motivated learning model that mimic human brain

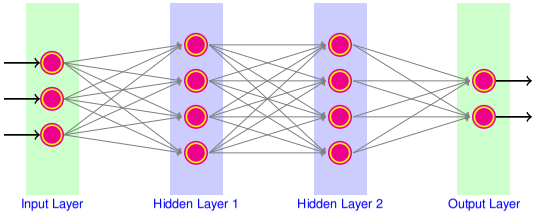
- Started by *W. McCulloch* study on working of neurons in 1943
- MADALINE (1959), an adaptive filter that eliminates echoes on phone lines was the first neural network
- Popularity of Neural Network diminished in 90's but, due to advances in **processing power** and availability of **large data** it again became state-of-the-art

Brief History

- 1872 Staining/Reticular Theory of Nervous Tissue
- 1943 McCulloch & Pitt (Neuron Model)
- 1947 Donald Hebb (Hebbian Learning)
- 1948 Nerbert (Cybernetics, optimal filter, feedback)
- 1954 Frank Rosenblatt (Perceptron)
- MADALINE, Checkers: 1959
- 1962 Hubel & Wiesel (Visual Cortex Model)
- 1965 Frank Rosenblatt (MLP: Multi Layer Perceptron)
- 1969 Minsky (Limitations of Perceptron)
- 1986 (Backpropagation)
- 1989 Universal Approximation Theorem
- Deep Blue: 1997 LSTM
- 1998 LeCun (ConvNet for MNIST digit)
- 2006 Unsupervised Pre Training
- 2010 Dahl. (Speech Recognition)
- 2012 AlexNet (ImageNet: Computer Vision) 26→16 … →12% zfNet
- 2013 VGG →7.3
- 2014 Google LeNet →6.7
- Waymo, AlphaGo, Yolo: 2015 ResNet →3.6
- 2017 DenseNet
- Deepfake, StyleGAN: 2018 Transformers, U-Net segmentation
- ChatGPT, AlphaFold: 2022 ChatGPT

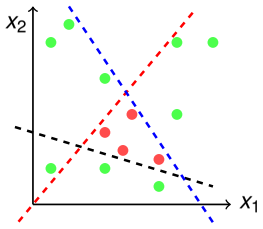
Neural Network

When neurons are interconnected in layers



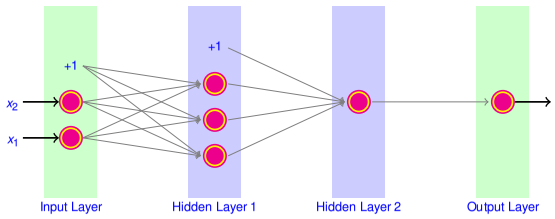
- Number of layers may differ
- Nodes in each intermediate layers may also differ
- Multiple output neurons are used for different class
- **Two levels deep** NN can represent any boolean function

More Example: Design NN for the following data



Whether is it green?

Red-line	Blue-line	Black-line	Color
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



Note: Weights and activation in subsequent layers add power to the model in terms of non linearity.

Neural Network Applications

NN is appropriate for problems with the following characteristics:

- Instances are provided by many attribute-value pairs (more data)
- The target function output may be discrete-valued, real-valued, or a vector of several real or discrete valued attributes
- The training examples may contain errors
- Long training times are acceptable
- Fast evaluation of the target function may be required
- The ability of humans to understand the learned target function is not important

Perceptron Training (delta rule)

When data is not linearly-separable; error fluctuates with *parameter training* updates. It is difficult to decide, when to stop?

- Delta rule converges to a best-fit approximation of the target
- Uses gradient descent
- Consider unthresholded perceptron, $o(\vec{x}) = \vec{w}.\vec{x}$
- Training error is defined as

$$E(\vec{w}) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$$

- Gradient would specify direction of steepest increase $\nabla E(\vec{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_n}\right]$
- Weights can be learned as $w_i = w_i - \eta \frac{\partial E}{\partial w_i}$
- It can be seen that $\frac{\partial E}{\partial w_i} = \sum_{d \in D}(t_d - o_d)(-x_{id})$

Perceptron Training (delta rule)

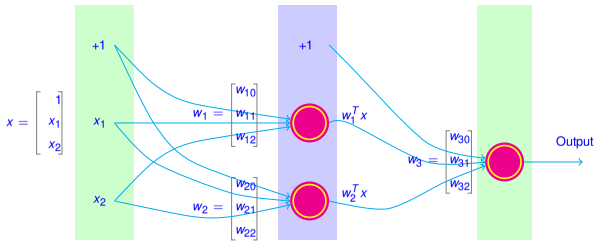
Algorithm 2: Gradient Descent (D, η)

Initialize w_i with random weights
repeat
  For each w_i, initialize $\triangle w_i = 0$
  for *each training example* $d \in D$ **do**
    Compute output o using model for d whose target is t
    For each w_i, update $\triangle w_i = \triangle w_i + \eta(t-o)x_i$
  For each w_i, set $w_i = w_i + \triangle w_i$
until *termination condition is met*;
return w

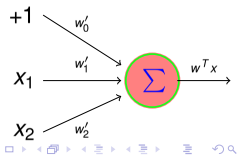
- A date item $d \in D$, is supposed to be multidimensional $d = (x_1, x_2, ..., x_n, t)$
- Algorithm converges toward the minimum error hypothesis.
- Linear programming can also be an approach

Linear Activation is Not Much Interesting

NN with perceptrons have limited capability, even with many layers



$$
\begin{aligned}
Output &= w_{30} \times 1 + w_{31} \times (w_1^T x) + w_{32} \times (w_2^T x) \\
&= w_{30} \times 1 + w_{31} \times [w_{10} \times 1 + w_{11} \times x_1 + w_{12} \times x_2] \\
&\quad + w_{32} \times [w_{20} \times 1 + w_{21} \times x_1 + w_{22} \times x_2] \\
&= (w_{30} + w_{31} w_{10} + w_{32} w_{20}) + (w_{31} w_{11} + w_{32} w_{21}) \times x_1 \\
&\quad + (w_{31} w_{12} + w_{32} w_{22}) \times x_2 \\
&= w'_0 + w'_1 \times x_1 + w'_2 \times x_2
\end{aligned}
$$



Expression of single perceptron

Thank You!

Thank you very much for your attention!