Machine Learning

Neural Networks, part 1

Jeff Abrahamson

octobre 2020

Perceptron

Gradient Descent

Perceptron

- Supervised
- Binary linear classifier
- Online learning

Perceptron

Beginnings:

- One of first artificial neural networks (ANN's)
- Developed in 1957 by Frank Rosenblatt, Cornell University Aeronautical Laboratory
- First implemented in software (IBM 704)
- Intended to be a machine
- Designed for image recognition

Perceptron

Controversy:

- 1958, press conference, NYT
- Rosenblatt too optimistic
- 1969, Minsky and Papert

Perceptron

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

The algorithm terminates if and only if the data is linearly separable.

Perceptron

- Also called “single-layer perceptron”
- Not related to multi-layer perceptron
- Feedforward neural network

Perceptron

The training data is

$$D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$$

Perceptron

The training data is

$$D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$$

$x_{j,i}$ is the value of the ith feature of the jth training input vector

$x_{j,0} = 1$ (the bias is thus w_0 rather than b)

w_i is the weight on the ith feature

Perceptron

Start by setting the weight vector to zero (or to some small random noise).

For each input vector j in turn:

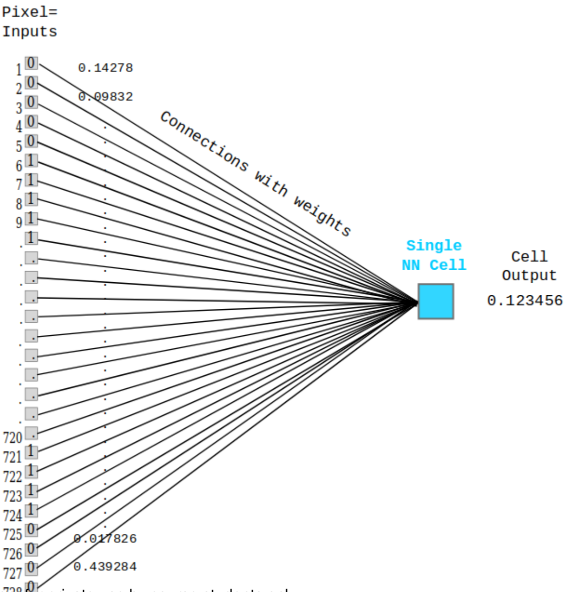
1. Compute $\hat{y} = f(w \cdot x)$
2. Update the weights: $w_i = w_i + (y_j - \hat{y}_j)x_{j,i}$

Multiclass Perceptron

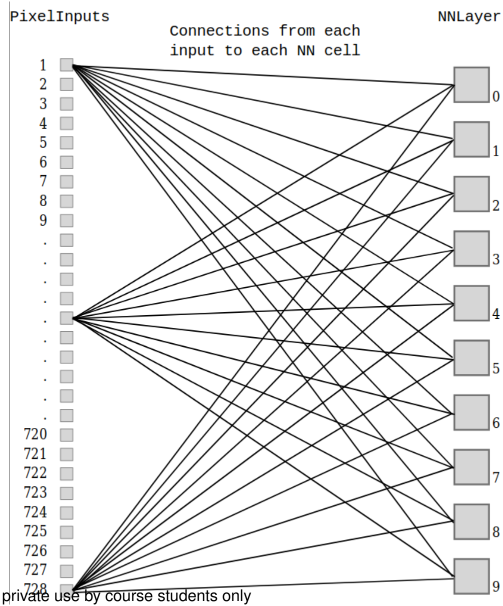
$$\hat{y} = \text{argmax}_y f(x, y) \cdot w$$

$$w = w + f(x, y) - f(x, \hat{y})$$

Multiclass Perceptron: Example



Multiclass Perceptron: Example



Perceptron History

- Perceptron is an example of SPR for image recognition
- Initially very promising
- IBM 704 (software implementation of algorithm)

- Mark 1 Perceptron at the Smithsonian Institution
- 400 photocells randomly connected to neurons.
- Weights encoded in potentiometers, updated during learning by electric motors

Frank Rosenblatt, The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, Cornell Aeronautical Laboratory, Psychological Review, v65, No. 6, pp. 386-408, 1958. doi:10.1037/h0042519

Perceptron History

- Minksy and Papert showed perceptrons are incapable of recognizing certain classes of images
- AI community mistakenly over-generalized to all NN's
- So NN research stagnated for some time

- Single layer perceptrons only recognize linearly separable input
- Hidden layers overcome this problem

M. L. Minsky and S. A. Papert, Perceptrons. Cambridge, MA: MIT Press. 1969.

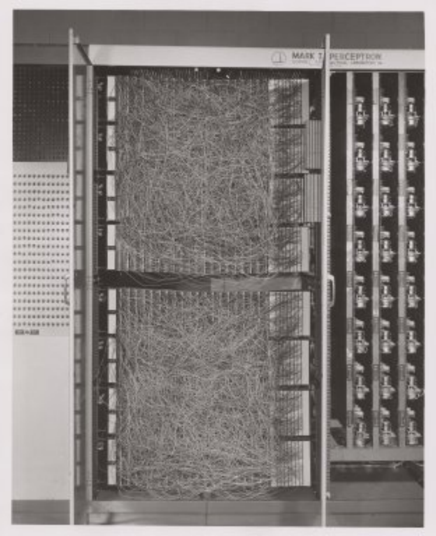
Perceptron History

- ANN's were slow.
- Vanishing gradient problem (Sepp Hochreiter)
- Support vector machines (SVN) were faster

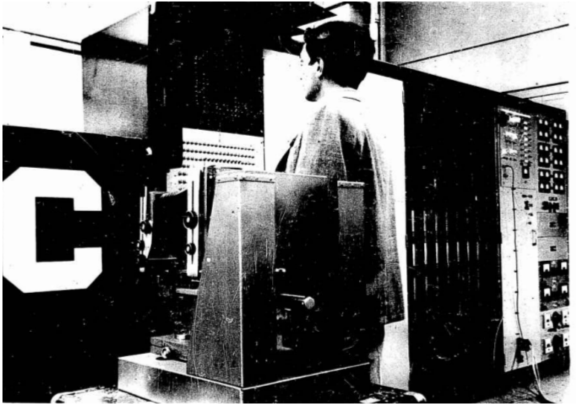
Perceptron History

- Inputs = 400 CdS photocells
- Weights = potentiometers
- Tuning = electric motors

Perceptron History



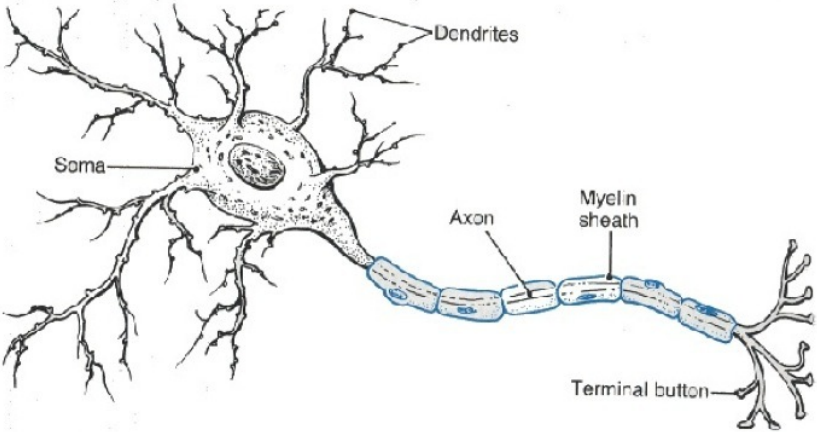
Perceptron History



Frank Rosenblatt, Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms, Report No. 1196-G-8, 15 March 1961, Cornell Aeronautical Laboratory

Neurons

Inspired by biology



. . . but only inspired

Linear neuron

$$y = b + \sum_i x_i w_i$$

Linear neuron

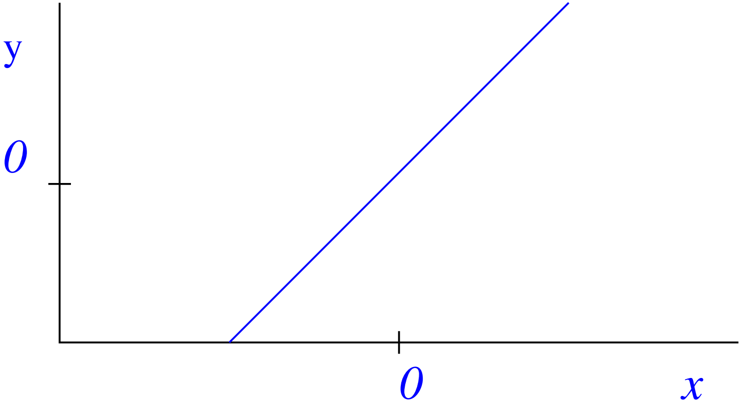
$$y = b + \sum_i x_i w_i$$

where

$$\begin{aligned} y &= \text{output} \\ b &= \text{bias} \\ x_i &= i^{\text{th}} \text{ input} \\ w_i &= \text{weight on } i^{\text{th}} \text{ input} \end{aligned}$$

Linear neuron

$$y = b + \sum_i x_i w_i$$



Binary threshold neuron

$$z = \sum_i x_i w_i$$

$$y = \begin{cases} 1 \text{ if } z \geqslant 0 \\ 0 \text{ otherwise} \end{cases}$$

Binary threshold neuron

$$z = \sum_i x_i w_i$$

$$y = \begin{cases} 1 \text{ if } z \geqslant 0 \\ 0 \text{ otherwise} \end{cases}$$

where

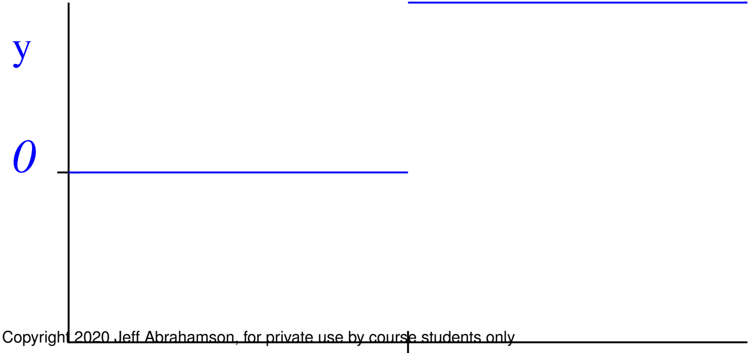
$$\begin{aligned} z &= \text{total input} \\ y &= \text{output} \\ x_i &= i^{\text{th}} \text{ input} \\ w_i &= \text{weight on } i^{\text{th}} \text{ input} \end{aligned}$$

W. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, 7:115–133, 1943.

Binary threshold neuron

$$z = \sum_i x_i w_i$$

$$y = \begin{cases} 1 \text{ if } z \geqslant 0 \\ 0 \text{ otherwise} \end{cases}$$



Rectified linear neuron

$$z = b + \sum_i x_i w_i$$

$$y = \begin{cases} z \text{ if } z \geqslant 0 \\ 0 \text{ otherwise} \end{cases}$$

Rectified linear neuron

$$z = b + \sum_i x_i w_i$$

$$y = \begin{cases} z \text{ if } z \geqslant 0 \\ 0 \text{ otherwise} \end{cases}$$

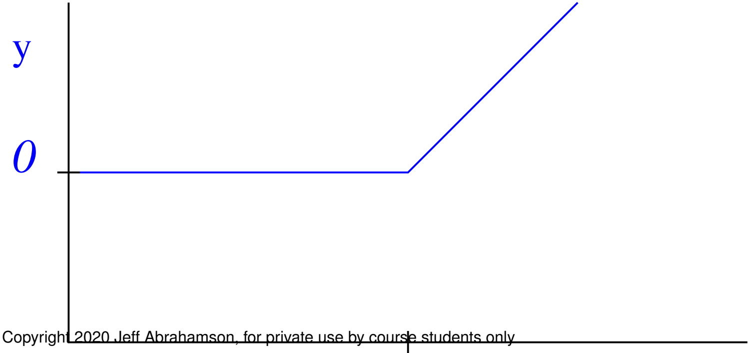
where

$$\begin{aligned} z &= \text{total input} \\ y &= \text{output} \\ b &= \text{bias} \\ x_i &= i^{\text{th}} \text{ input} \\ w_i &= \text{weight on } i^{\text{th}} \text{ input} \end{aligned}$$

Rectified linear neuron

$$z = b + \sum_i x_i w_i$$

$$y = \begin{cases} z \text{ if } z \geqslant 0 \\ 0 \text{ otherwise} \end{cases}$$



Sigmoid neuron

$$z = b + \sum_i x_i w_i$$

$$y = \frac{1}{1 + e^{-z}}$$

Sigmoid neuron

$$z = b + \sum_i x_i w_i$$

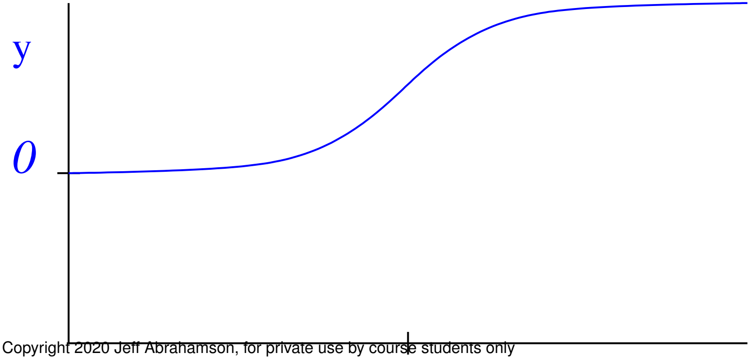
$$y = \frac{1}{1 + e^{-z}}$$

(It's differentiable!)

Sigmoid neuron

$$z = b + \sum_i x_i w_i$$

$$y = \frac{1}{1 + e^{-z}}$$



Stochastic binary neuron

$$z = b + \sum_i x_i w_i$$

$$p = \frac{1}{1 + e^{-z}}$$

$$y = \begin{cases} 1 \text{ with probability } p \\ 0 \text{ with probability } 1 - p \end{cases}$$

Stochastic binary neuron

$$z = b + \sum_i x_i w_i$$

$$p = \frac{1}{1 + e^{-z}}$$

$$y = \begin{cases} 1 \text{ with probability } p \\ 0 \text{ with probability } 1 - p \end{cases}$$

(a probability distribution)

Stochastic binary neuron

$$z = b + \sum_i x_i w_i$$

$$p = \frac{1}{1 + e^{-z}}$$

$$y = \begin{cases} 1 \text{ with probability } p \\ 0 \text{ with probability } 1 - p \end{cases}$$

Can also do something similar with rectified linear neurons, produce spikes with probability p with a Poisson distribution.

Neural Networks

Architecture

It's how we connect the dots (the states).

Architecture

Feedforward neural networks

- Flow is unidirectional
- No loops

Makes linear separators (perceptron).

Architecture

Idea: maybe add some layers in the middle

Architecture

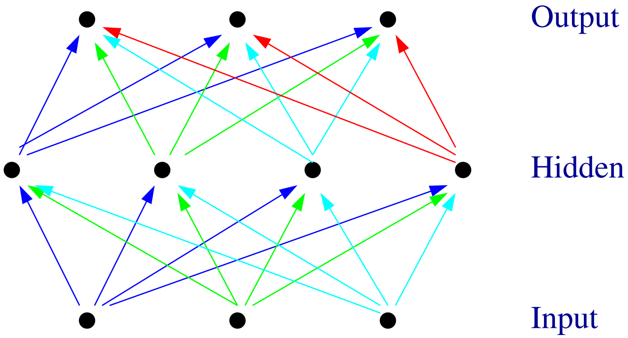
Idea: maybe add some layers in the middle

What would we put there?

Maybe choose not to care, call them “hidden layers”.

Layers

Neuron activity at each layer must be a non-linear function of previous layer



If more than two hidden layers, then we call it deep

Recurrent neural networks (RNN)

- Cycles
- Memory
- Oscilalations
- Harder to train

Recurrent neural networks (RNN)

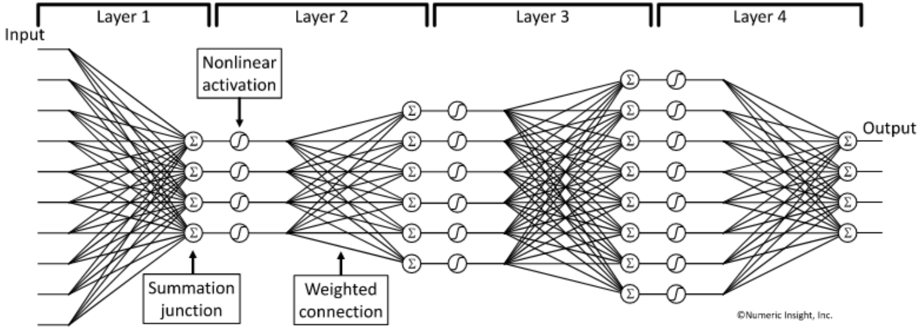
Video time

So how would we train such a thing?

So how would we train such a thing?

It turns out that the perceptron algorithm is unstable and prone to many problems on deep or non-linear networks.

Answer: Backpropagation



$$(10 \times 4) + (4 \times 6) + (6 \times 8) + (8 \times 4) = 144 \text{ connections}$$

Shashi Sathyanarayana, A Gentle Introduction to Backpropagation, 22 July 2014.

Question: How do we find the weights on those 144 connections?

Question: How do we find the weights on those 144 connections?

Need a way of refining an initial (random) guess.

Question: How do we find the weights on those 144 connections?

Need a way of refining an initial (random) guess.

Feedforward is not stable.

Question: How do we find the weights on those 144 connections?

So work backwards.

Backprogation

- Modify weights at output layer by an amount proportional to the partial-derivative of the error with respect to that weight.
- Then do next layer.
- Continue through all layers, recomputing partial derivatives at each step.

Backprogation

- Modify weights at output layer by an amount proportional to the partial-derivative of the error with respect to that weight.
- Then do next layer.
- Continue through all layers, recomputing partial derivatives at each step.

Repeat.

Backprogation

This was hard to learn to do right.

Next week: architectures, examples, and code

questions?