Introduction to Deep Networks

It starts with regression: Gauss - 1800 - predicting the position of the planetoid Ceres.

Set of measurements \( \{(X_n, Y_n) : n = 1 \text{ to } N\} \)

Model \( Y = ax + b + \varepsilon \)

Fit data by least squares

\[
E[a,b] = \frac{1}{N} \sum_{n=1}^{N} (y_n - ax_n - b)\]

minimize \( \hat{a}, \hat{b} = \arg \min_{a,b} E[a,b] \)

Probabilistic Regression

\[
P(y \mid x; \theta) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y - ax - b)^2}{2\sigma^2}}
\]

Estimate parameters by Maximum Likelihood (ML)

\[
(\hat{a}, \hat{b}, \hat{\varepsilon}) = \arg \max_{a,b,\sigma} \prod_{n=1}^{N} P(y_n \mid x_n; \sigma, a, b)
\]

\[
= \arg \min_{a,b} \left\{ -\frac{N}{2} \log \sigma - \frac{1}{2\sigma^2} \sum_{n=1}^{N} (y_n - ax_n - b)^2 \right\}
\]

\( \text{reduces to } \hat{[a,b]} = \arg \min_{a,b} E[a,b], \text{ as before, } \hat{\varepsilon}^2 = \frac{1}{N} \sum_{n=1}^{N} (y_n - ax_n - b)^2 \)
Introduction to Deep Networks

Perception → Rosenblatt → 1950's

Task: classification

Data \( \{(y^n, x^n) : n = 1 \leq N\} \)

Try to learn a classifier/decision rule.

\[ \hat{y}(x) = +1, \text{ if } \mathbf{w} \cdot x + w_0 > 0 \]
\[ \hat{y}(x) = -1, \text{ if } \mathbf{w} \cdot x + w_0 < 0 \]
\[ \mathbf{w} \cdot x + w_0 = 0 \]

Here + is a datapoint \((y^n, x^n)\) with \(y^n = 1\)
- is a datapoint \((y^n, x^n)\) with \(y^n = -1\).

The perceptron algorithm learns the parameters \(\mathbf{w}, w_0\) from the data \(\{(y^n, x^n) : n = 1 \leq N\}\).

The perceptron algorithm was motivated by simplified models of neurons.

Support Vector Machines (SVM) are descended from Perceptions. But now we follow another path.
The Reformulate perceptions as logistic regression.

\[ P(y \mid x) = \frac{e^{\langle \omega, x \rangle + c_0}}{1 + e^{\langle \omega, x \rangle + c_0}} \]

Input data

\[ \{(y^n, x^n) : n = 1 \ldots N\} \]

Estimate \( \hat{\omega}, \hat{c}_0 \) by Maximum Likelihood

\[ \hat{\omega}, \hat{c}_0 = \arg \min_{\omega, c_0} -\frac{1}{N} \sum_{n=1}^{N} \log P(y^n \mid x^n, \omega) \]

The parameters \( \omega, c_0 \) can be estimated by steepest descent / gradient descendent.

\[ \omega^{t+1} = \omega^t - \eta + \frac{2}{\eta \omega} \left( -\frac{1}{N} \sum_{n=1}^{N} \log P(y^n \mid x^n, \omega) \right) \]

or. stochastic gradient descent.

At time \( t \) pick \( (x^n, y^n) \)

\[ \omega^{t+1} = \omega^t - \eta + \frac{2}{\eta \omega} \left( -\log P(y^n \mid x^n, \omega) \right) \]
Introduction to Deep Networks

Alternative perspective:

This is regression with

\[ P(y | x; \omega, \varphi) = \frac{e^{\omega \cdot \varphi(x, \varphi)}}{\sum_{y} e^{\omega \cdot \varphi(x, \varphi)}} \]

where \( \varphi(x, \varphi) = \tanh(\varphi(x) \cdot x) \).

Learn the parameters \( \omega, \varphi \) by maximizing likelihood (ML).

\[ \hat{\omega}, \hat{\varphi} = \arg \min_{\omega, \varphi} \left\{ -\frac{1}{n} \sum_{i=1}^{n} \log P(y_i | x_i; \omega, \varphi) \right\} \]

stochastic gradient descent

\[ \omega^{t+1} = \omega^t - \eta_t \frac{\partial}{\partial \omega} \left\{ -\frac{1}{n} \sum_{i=1}^{n} \log P(y_i | x_i; \omega, \varphi) \right\} \]

or gradient descent

\[ \omega^{t+1} = \omega^t - \eta_t \frac{\partial}{\partial \omega} \left\{ -\frac{1}{n} \sum_{i=1}^{n} \log P(y_i | x_i; \omega, \varphi) \right\} \]

where \((y_i, x_i)\) are the training pair selected at time \(t\).
For logistic regression, the steepest descent algorithm is guaranteed to converge to an optimal solution. Technically, because \(-\log p(y|x, \omega)\) is a convex function of \(\omega\), which is bounded below, so it has a single minimum.

In the 1980's researchers extended perceptrons to multi-layer perceptron:

\[
y = f(\omega, R, x)
\]

\[
y = \tanh(\omega \cdot z)
\]

\[
z = (z_1, z_2, z_3)
\]

\[
z_i = \tanh(\omega_i \cdot x)
\]

\[
\tanh(z_i) = \frac{e^{z_i} - e^{-z_i}}{e^{z_i} + e^{-z_i}}
\]

Key result: the output \(y\) is a differentiable function of the parameters \(\omega\) & \(R\).
For multi-layer perceptrons, there is no guarantee of convergence to the global minima of $\frac{1}{n} \sum_{i=1}^{n} \log P(y_i | x_i, w, b)$. This function is not convex.

Easy for a learning algorithm to get trapped in local minimum.

But, in practice, algorithms often converge to good solutions. Perhaps because there are many local minima, all of which are equally good solutions. (cf. Ref. Rene Vidal)

Multilayer Perceptrons can represent many more decision rules than Perceptrons, so they are more effective. But they still had limited performance.
Introduction to Deep Networks

Deep networks extend multi-layer perceptrons by having convolutional filters & filter-banks (see handout slides).

They are more effective than 1980's multilayer perceptrons because:

(i) they are much bigger

AlexNet → 650,000 neurons
60,000,000 parameters
630,000,000 connections

(ii) they can be implemented in Graphical Processing Units (GPUs), which were developed for video games

(iii) there is now enough data to train them \( \{ (x_i, y_i) : n = 10, N \} \)

\[ N = 1,000,000 \] or more.

Researches in the 1980's did not have the technology - computers, data.