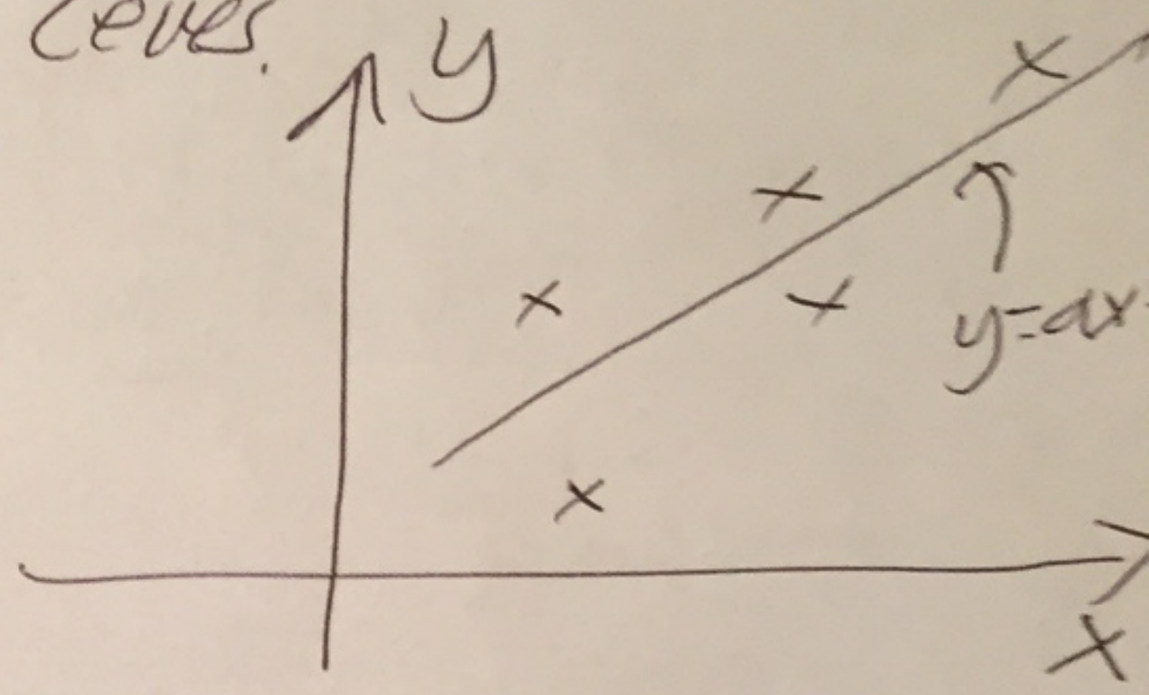
# Introduction to Deep Networks

Yuille

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
$$y = ax + b + \epsilon$$
noise.

Fit data by least squares.
$$E[a,b] = \frac{1}{N}\sum_{n=1}^{N}\{y_n - ax_n - b\}^2$$

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

Probabilistic Regression.
$$P(y \mid x; b, a, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{(y-ax-b)^2/2\sigma^2}$$

Estimate parameters by Maximum Likelihood (ML)
$$(\hat{a}, \hat{b}, \hat{\sigma}) = \arg\max_{(a,b,\sigma)} \prod_{n=1}^{N} P(y_n \mid x_n; b, a, \sigma)$$
$$= \arg\min_{(a,b,\sigma)} \left\{ -\sum_{n=1}^{N} \log P(y_n \mid x_n; b, \sigma, \sigma) \right\}$$

reduces to $[\hat{a}, \hat{b}] = \arg\min_{a,b} E[a,b]$, as before, $\hat{\sigma}^2 = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{a}x_n - \hat{b})^2$

# Introduction to Deep Networks

Perceptron → Rosenblatt → 1950's

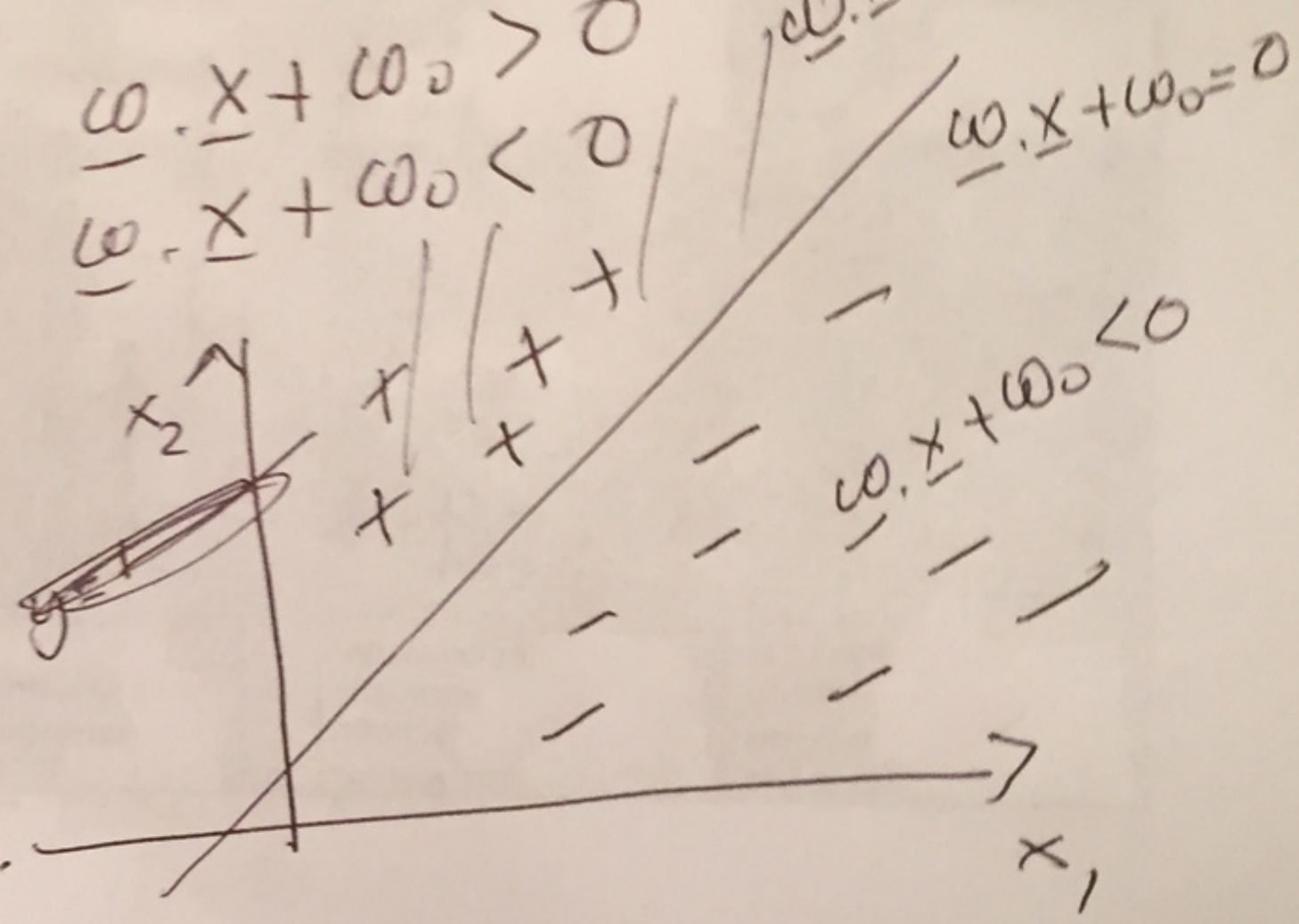
Task: classification. $y \in \{\pm 1\}$

Data $\{(y^n, \underline{x}^n) : n = 1 \text{ to } N\}$,

Try to learn a classifier / decision rule.

$\hat{y}(\underline{x}) = +1$, if $\underline{\omega} \cdot \underline{x} + \omega_0 > 0$

$\hat{y}(\underline{x}) = -1$, if $\underline{\omega} \cdot \underline{x} + \omega_0 < 0$

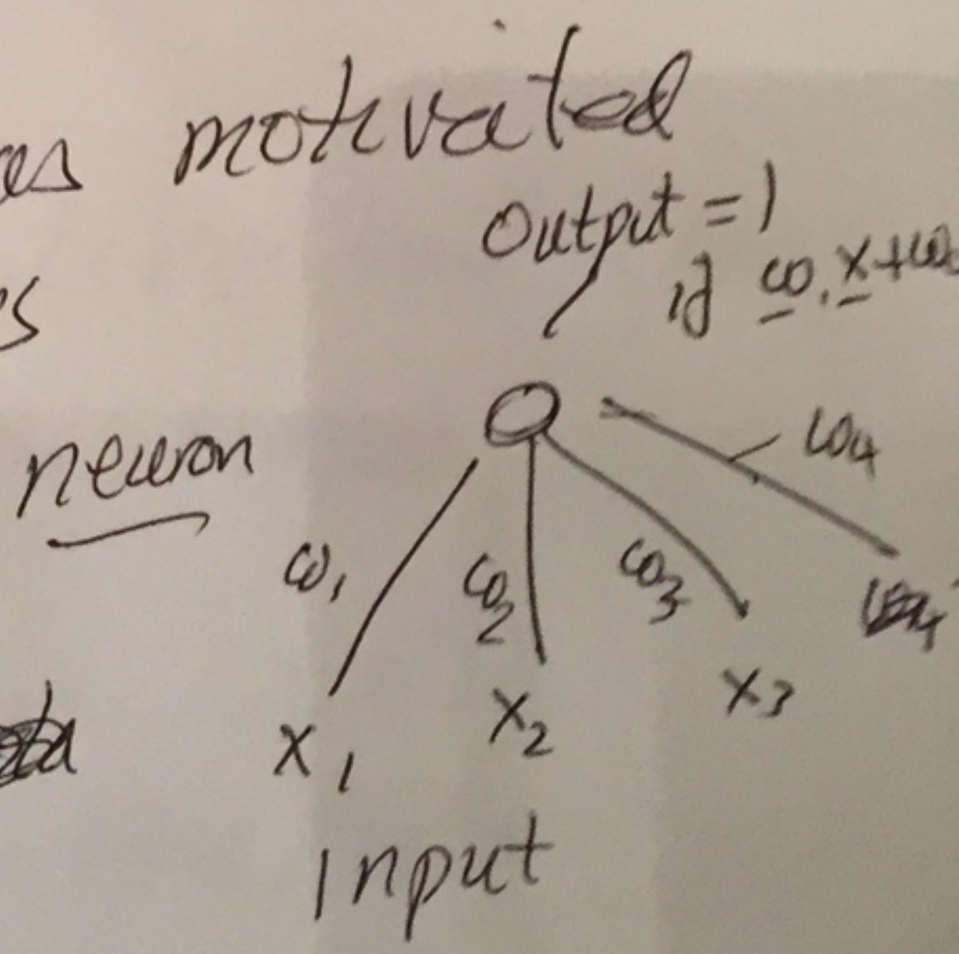



Here + is a datapoint $(y^n, \underline{x}^n)$ with $y^n = 1$

– is a datapoint $(y^n, \underline{x}^n)$ with $y^n = -1$.

The perceptron algorithm learns the parameters $\underline{\omega}, \omega_0$ from the data $\{(y^n, \underline{x}^n) : n = 1 \text{ to } N\}$

The perceptron algorithm was motivated by simplified modes of neurons




Support Vector Machines (SVM) are descended from Perceptrons.

But now we follow another path.

# Introduction to Deep Networks



~~The~~ Reformulate perceptrons as logistic regression.

$$P(y \mid \underline{x}) = \frac{e^{y\{\underline{\omega}.\underline{x}+\omega_0\}}}{e^{\underline{\omega}.\underline{x}+\omega_0} + e^{-(\underline{\omega}.\underline{x}+\omega_0)}}$$

input data

$\{(y^n, \underline{x}^n) : n = 1 \text{ to } N\}$

Estimate $\underline{\omega}, \omega_0$ by Maximum Likelihood

$$\hat{\underline{\omega}}, \hat{\omega}_0 = \arg\min_{\hat{\underline{\omega}}, \omega_0} \left\{ -\sum_{n=1}^{N} \log P(y_n \mid \underline{x}_n, \underline{\omega}) \right\}$$

~~This~~

The parameters $\underline{\omega}, \omega_0$ can be estimated by steepest descent / gradient descent

$$\underline{\omega}^{t+1} = \underline{\omega}^t - \eta_t \frac{\partial}{\partial \underline{\omega}} \left( -\sum_{n=1}^{N} \log P(y^n \mid \underline{x}^n, \underline{\omega}) \right)$$

BATCH MODE.

learning rate

or. ~~stoch~~ stochastic gradient descent

At time $t$ pick ~~n(t)~~ $(\underline{x}^{n(t)}, y^{n(t)})$

$$\underline{\omega}^{t+1} = \underline{\omega}^t - \eta_t \frac{\partial}{\partial \underline{\omega}} \left( -\log P(y^n \mid \underline{x}^n, \underline{\omega}) \right)$$

# Introduction to Deep Networks

Yuille 5

## Alternative perspective:

This is regression with

$$P(y \mid \underline{x}; \underline{\omega}, \underline{\Omega}) = \frac{e^{y\{\underline{\omega}\cdot\underline{z}(\underline{x},\underline{\Omega})\}}}{e^{\underline{\omega}\cdot\underline{z}(\underline{x},\underline{\Omega})} + e^{-\underline{\omega}\cdot\underline{z}(\underline{x},\underline{\Omega})}}$$

where $\underline{z}(\underline{x},\underline{\Omega}) = \tanh(\underline{\Omega}_i \cdot \underline{x})$.

Learn the parameters $\underline{\omega}, \underline{\Omega}$ by maximum likelihood (ML).

$$\hat{\underline{\omega}}, \hat{\underline{\Omega}} = \arg\min_{\underline{\omega},\underline{\Omega}} \left\{ -\sum_{n=1}^{N} \log P(y_n \mid \underline{x}_n; \underline{\omega}, \underline{\Omega}) \right\}$$

stochastic gradient descent

$$\underline{\omega}^{t+1} = \underline{\omega}^{t} - \eta_t \frac{\partial}{\partial \underline{\omega}} \left\{ -\log P(y_{n(t)} \mid \underline{x}_{n(t)}; \underline{\omega}, \underline{\Omega}) \right\}$$

where $\{y_{n(t)}, \underline{x}_{n(t)}\}$ are the training pair selected at time $t$.

or gradient decent.

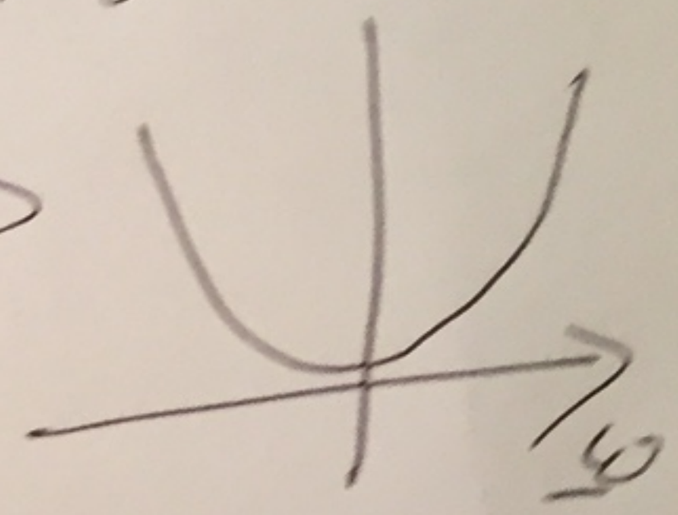
$$\underline{\omega}^{t+1} = \underline{\omega}^{t} - \eta_t \frac{\partial}{\partial \underline{\omega}} \left\{ -\sum_{n=1}^{N} \log P(y_n \mid \underline{x}_n; \underline{\omega}, \underline{\Omega}) \right\}$$

# Introduction to Deep Networks



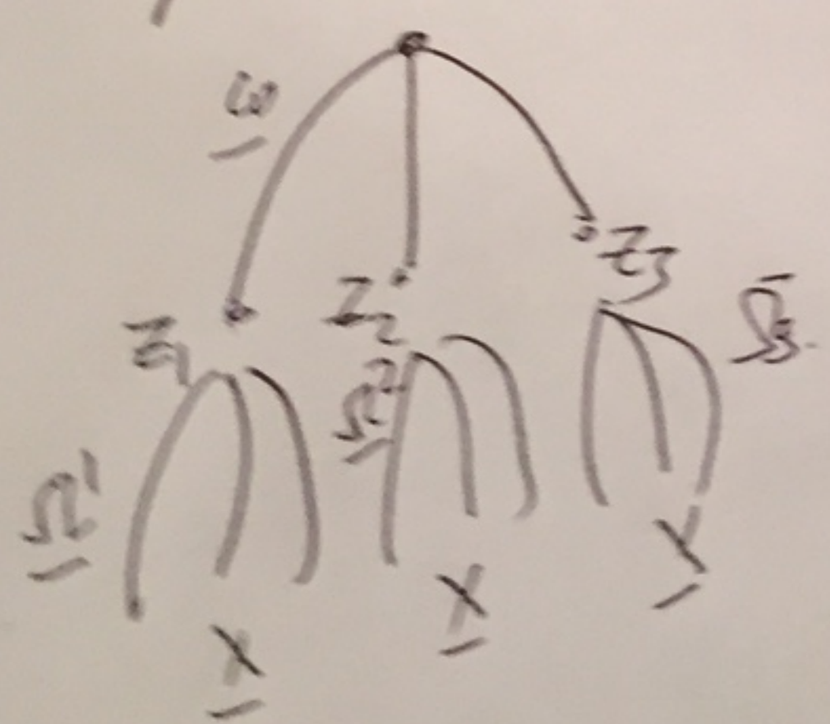
For logistic regression the steepest descent algorithm is guaranteed to converge to the optimal solution

Technically, because $-\log P(y \mid \underline{x}, \underline{\omega})$ is a convex function of $\underline{\omega}$ which is bounded below. So it has a single minimum.




---

In the 1990's researchers extended perceptrons to multi-layer perceptron



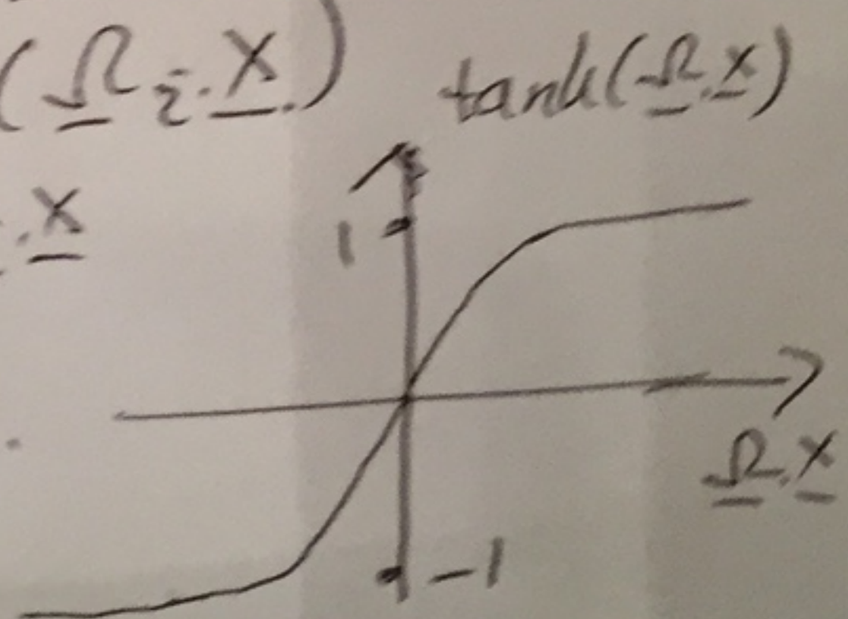

$$y = f(\underline{\omega}, \underline{\Omega}, \underline{x})$$

$$y = \tanh(\underline{\omega} \cdot \underline{z})$$

$$\underline{z} = (z_1, z_2, z_3)$$

$$z_i = \tanh(\underline{\Omega}_i \cdot \underline{x})$$

$$\tanh(\underline{\Omega}_i \cdot \underline{x}) = \frac{e^{\underline{\Omega}_i \cdot \underline{x}} - e^{-\underline{\Omega}_i \cdot \underline{x}}}{e^{\underline{\Omega}_i \cdot \underline{x}} + e^{\underline{\Omega}_i \cdot \underline{x}}}.$$




Key result: the output $y$ is a differentiable function of the parameters $\underline{\omega}$ & $\underline{\Omega}$

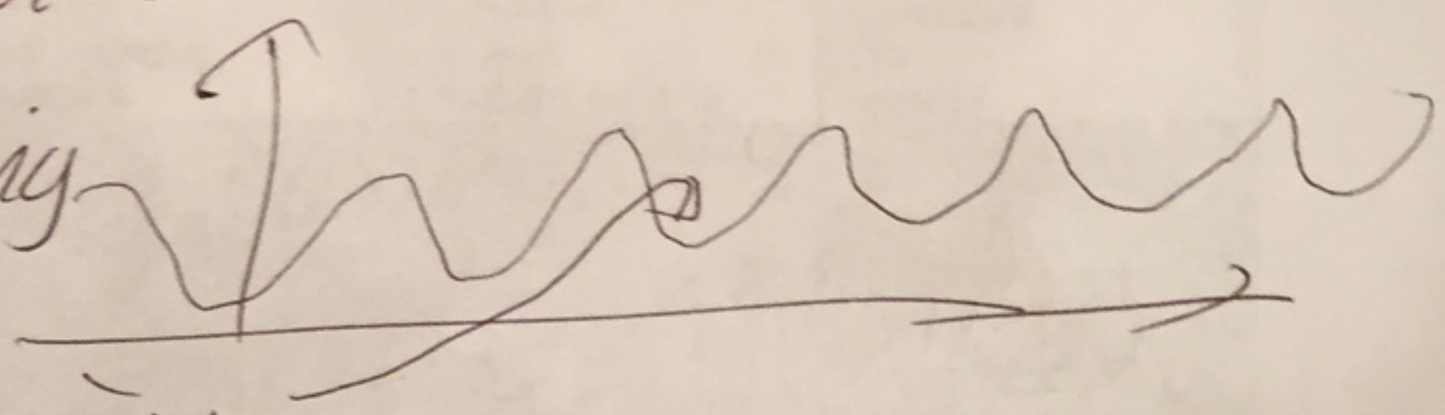
# Introduction to Deep Networks

Yuille 06.

For multilayer perceptron, there is no guarantee of convergence to the global minimum of $-\sum_{n=1}^{N} \log P(y_n | \underline{x}_n; \omega, \beta)$.

This function is not convex.

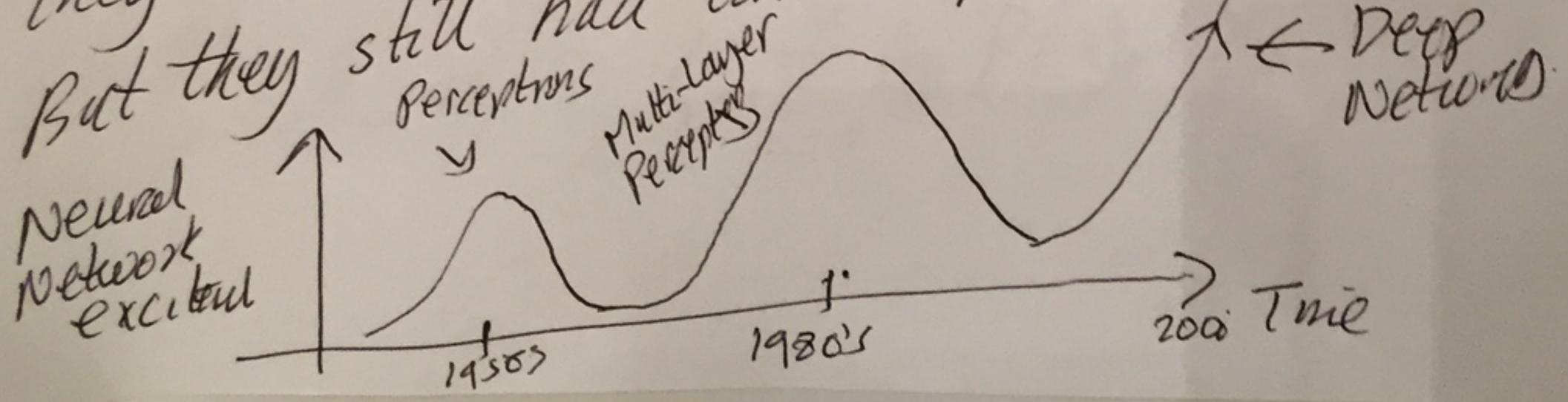
Easy for a learning algorithms to get trapped in local minimum.



But, in practice, algorithms often converge to good solutions. Perhaps because there are many local minima which are all equally good solutions. (c.f. Prof. 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

Yuille 7.

Deep Network extend multilayer perceptrons by having convolutional filters & filterbanks (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 (GPU's), which were developed for video games.

(iii) there is now enough data to train them $\{(\underline{x}_n, y_n) : n = 1 \text{ to } N\}$, $N \approx 1{,}000{,}000$ or more.

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