

1



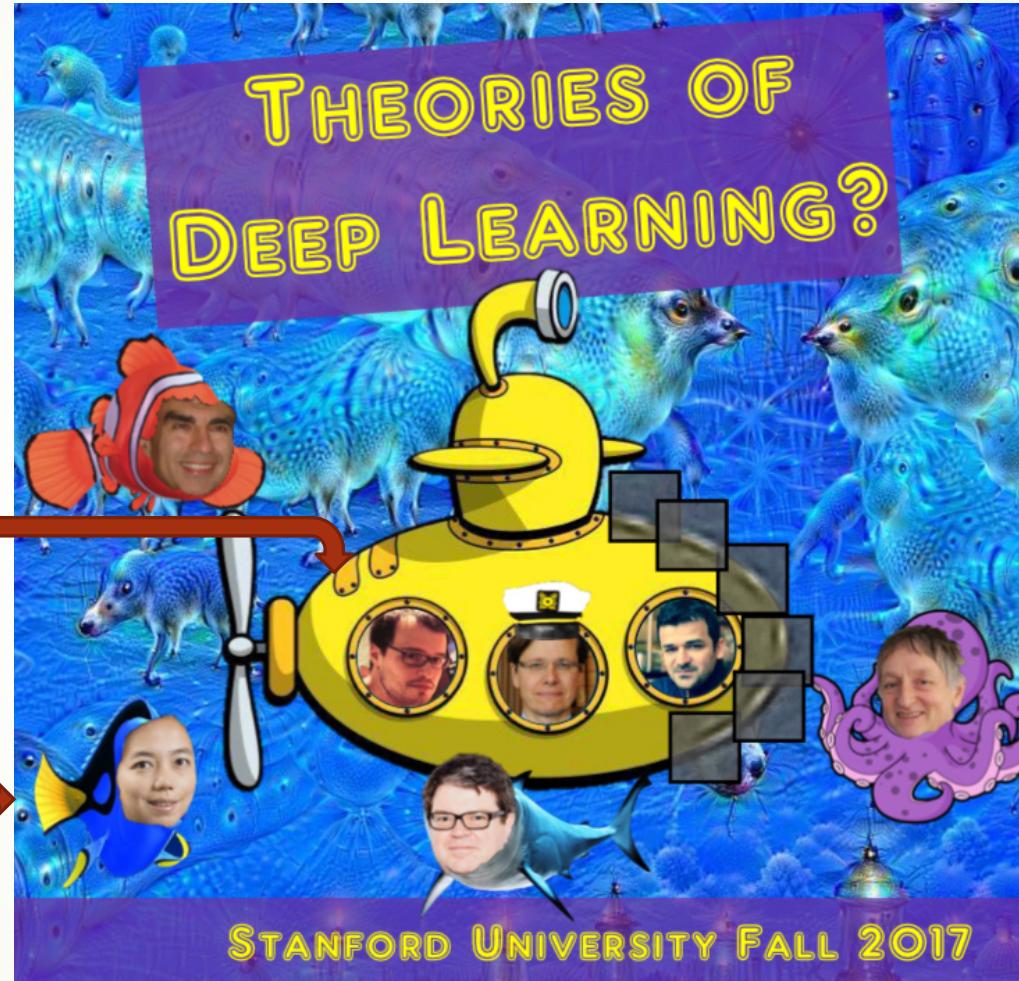
# Deep Learning: Towards Deeper Understanding

Yuan YAO  
HKUST

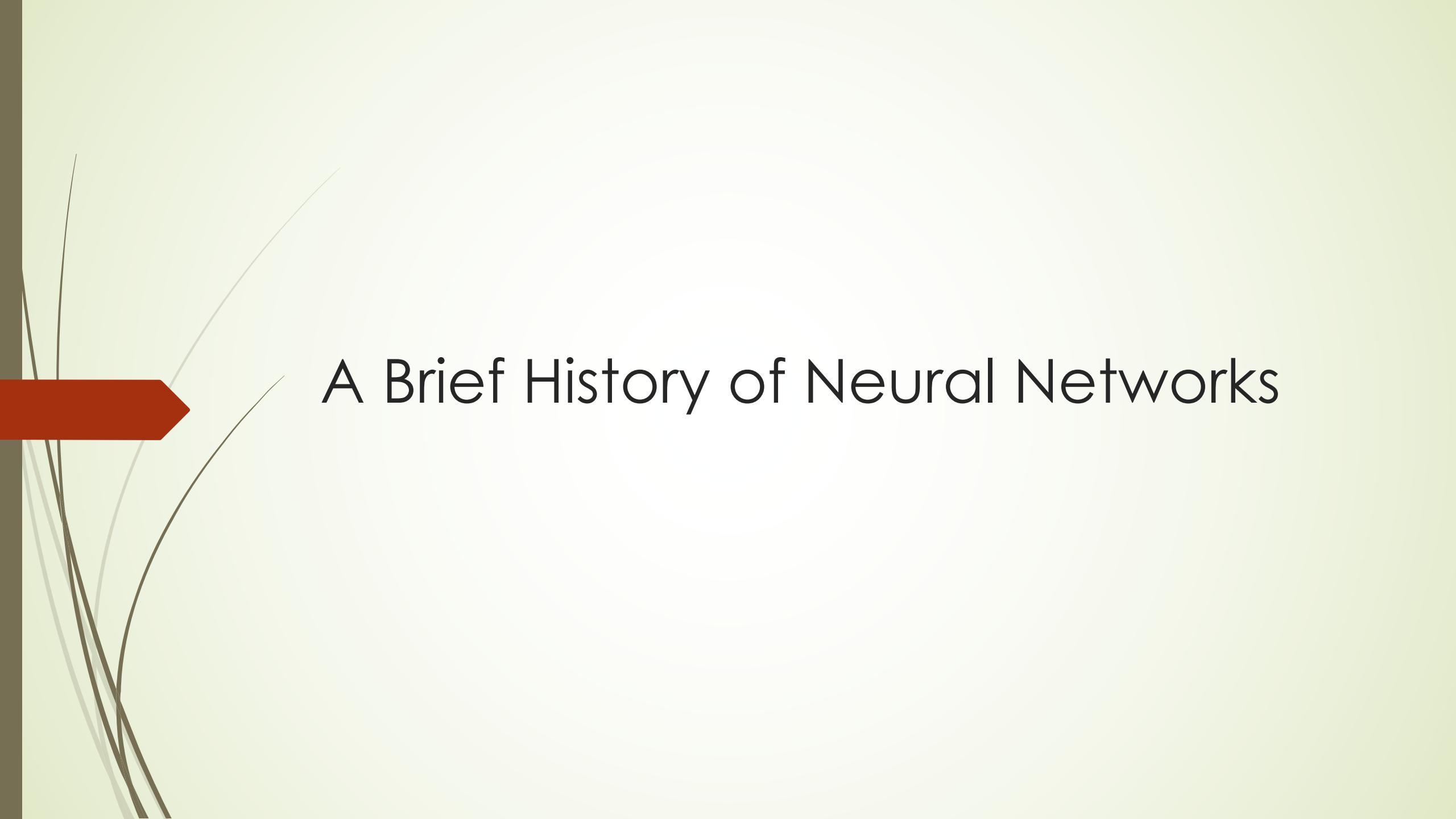
# Acknowledgement

<https://stats385.github.io/>

<http://cs231n.github.io/>



A following-up course at HKUST: <https://deeplearning-math.github.io/>

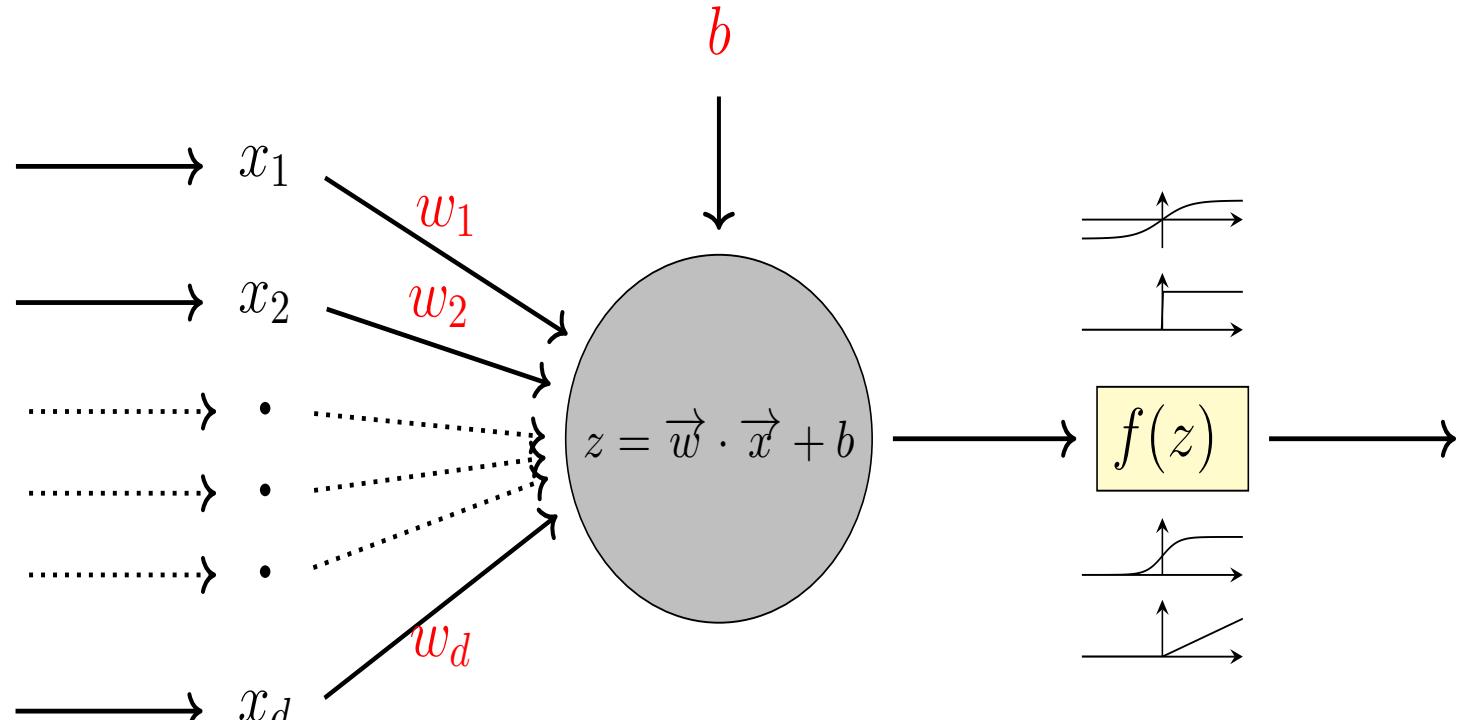


# A Brief History of Neural Networks

# Perceptron: single-layer



- Invented by Frank Rosenblatt (1957)



# Hilbert's 13th Problem

Algebraic equations (under a suitable transformation) of degree up to 6 can be solved by functions of two variables. What about

$$x^7 + ax^3 + bx^2 + cx + 1 = 0?$$

Hilbert's conjecture:  $x(a, b, c)$  cannot be expressed by a superposition (sums and compositions) of bivariate functions.

**Question:** can every continuous (analytic,  $C^\infty$ , etc) function of  $n$  variables be represented as a superposition of continuous (analytic,  $C^\infty$ , etc) functions of  $n - 1$  variables?

**Theorem (D. Hilbert)**

*There is an analytic function of three variables that cannot be expressed as a superposition of bivariate ones.*

# Kolmogorov's Superposition Theorem

Theorem (A. Kolmogorov, 1956; V. Arnold, 1957)

Given  $n \in \mathbb{Z}^+$ , every  $f_0 \in C([0, 1]^n)$  can be represented as

$$f_0(x_1, x_2, \dots, x_n) = \sum_{q=1}^{2n+1} g_q \left( \sum_{p=1}^n \phi_{pq}(x_p) \right),$$

where  $\phi_{pq} \in C[0, 1]$  are increasing functions independent of  $f_0$  and  $g_q \in C[0, 1]$  depend on  $f_0$ .

- Can choose  $g_q$  to be all the same  $g_q \equiv g$  (Lorentz, 1966).
- Can choose  $\phi_{pq}$  to be Hölder or Lipschitz continuous, but not  $C^1$  (Fridman, 1967).
- Can choose  $\phi_{pq} = \lambda_p \phi_q$  where  $\lambda_1, \dots, \lambda_n > 0$  and  $\sum_p \lambda_p = 1$  (Sprecher, 1972).

If  $f$  is a multivariate continuous function, then  $f$  can be written as a superposition of composite functions of mixtures of continuous functions of single variables:  
finite **composition** of continuous functions of a **single variable** and the **addition**.

# Kolmogorov's Exact Representation is Irrelevant

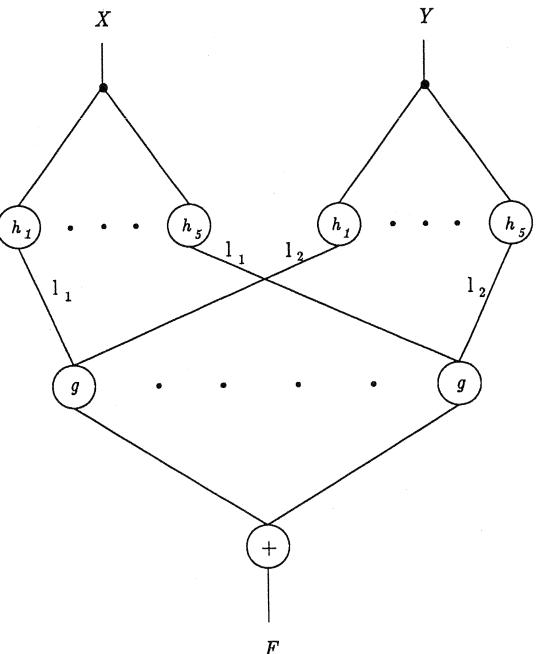


Figure 1: The network representation of an improved version of Kolmogorov's theorem, due to Kahane (1975). The figure shows the case of a bivariate function. The Kahane's representation formula is  $f(x_1, \dots, x_n) = \sum_{q=1}^{2n+1} g[\sum_{p=1}^n l_p h_q(x_p)]$  where  $h_q$  are strictly monotonic functions and  $l_p$  are strictly positive constants smaller than 1.

- ▶ [Girosi-Poggio'1989] Representation Properties of Networks:  
Kolmogorov's Theorem Is Irrelevant,  
<https://www.mitpressjournals.org/doi/pdf/10.1162/neco.1989.1.4.465>
- ▶ Lacking smoothness in  $h$  and  $g$   
[Vitushkin'1964] fails to guarantee the **generalization ability (stability)** against noise and perturbations
- ▶ The representation is **not universal** in the sense that  $g$  and  $h$  both depend on the function  $F$  to be represented.

# A Simplified illustration by David McAllester

## A Simpler, Similar Theorem

For (possibly discontinuous)  $f : [0, 1]^N \rightarrow \mathbb{R}$  there exists (possibly discontinuous)  $g, h_i : \mathbb{R} \rightarrow \mathbb{R}$ .

$$f(x_1, \dots, x_N) = g \left( \sum_i h_i(x_i) \right)$$

Proof: Select  $h_i$  to spread out the digits of its argument so that  $\sum_i h_i(x_i)$  contains all the digits of all the  $x_i$ .

# Universal Approximate Representation

[Cybenko'1989, Hornik et al. 1989, Poggio-Girosi'1989, ...]

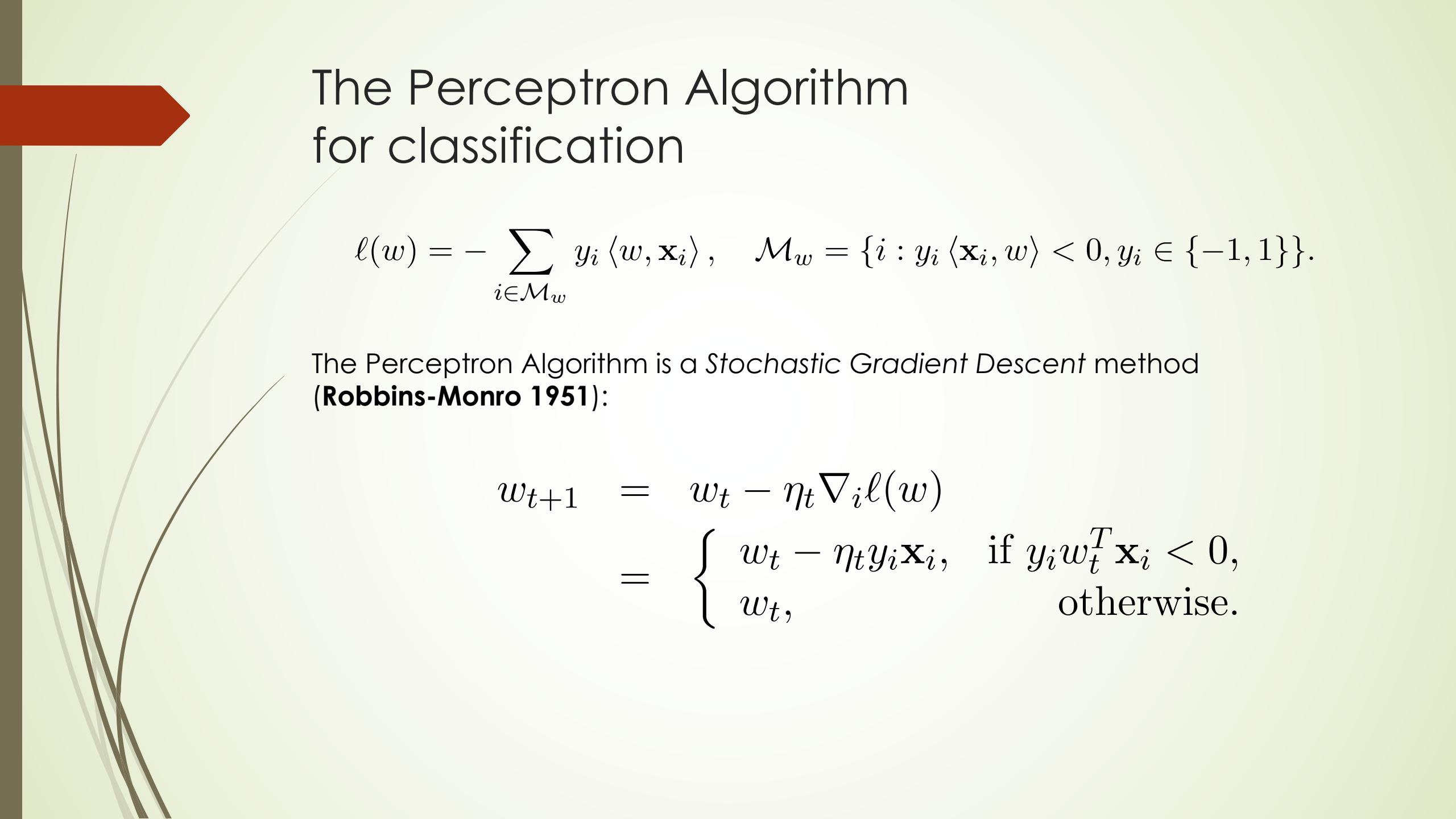
For continuous  $f : [0, 1]^N \rightarrow \mathbb{R}$  and  $\varepsilon > 0$  there exists

$$F(x) = \alpha^\top \sigma(Wx + \beta)$$

$$= \sum_i \alpha_i \sigma \left( \sum_j W_{i,j} x_j + \beta_i \right)$$

such that for all  $x$  in  $[0, 1]^N$  we have  $|F(x) - f(x)| < \varepsilon$ .

Complexity (regularity, smoothness) thereafter becomes the central pursuit in Approximation Theory.



# The Perceptron Algorithm for classification

$$\ell(w) = - \sum_{i \in \mathcal{M}_w} y_i \langle w, \mathbf{x}_i \rangle, \quad \mathcal{M}_w = \{i : y_i \langle \mathbf{x}_i, w \rangle < 0, y_i \in \{-1, 1\}\}.$$

The Perceptron Algorithm is a *Stochastic Gradient Descent* method  
**(Robbins-Monro 1951)**:

$$\begin{aligned} w_{t+1} &= w_t - \eta_t \nabla_i \ell(w) \\ &= \begin{cases} w_t - \eta_t y_i \mathbf{x}_i, & \text{if } y_i w_t^T \mathbf{x}_i < 0, \\ w_t, & \text{otherwise.} \end{cases} \end{aligned}$$

# Finiteness of Stopping Time and Margin

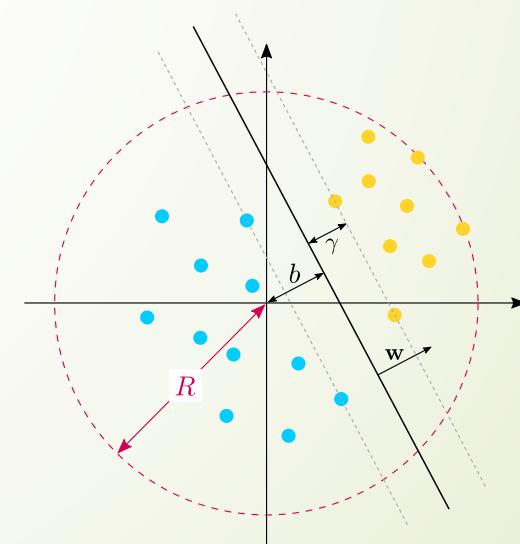
The perceptron convergence theorem was proved by [Block \(1962\)](#) and [Novikoff \(1962\)](#). The following version is based on that in [Cristianini and Shawe-Taylor \(2000\)](#).

**Theorem 1** (Block, Novikoff). *Let the training set  $S = \{(\mathbf{x}_1, t_1), \dots, (\mathbf{x}_n, t_n)\}$  be contained in a sphere of radius  $R$  about the origin. Assume the dataset to be linearly separable, and let  $\mathbf{w}_{\text{opt}}$ ,  $\|\mathbf{w}_{\text{opt}}\| = 1$ , define the hyperplane separating the samples, having functional margin  $\gamma > 0$ . We initialise the normal vector as  $\mathbf{w}_0 = \mathbf{0}$ . The number of updates,  $k$ , of the perceptron algorithms is then bounded by*

$$k \leq \left( \frac{2R}{\gamma} \right)^2. \quad (10)$$

Input ball:  $R = \max_i \|\mathbf{x}_i\|$ .

Margin:  $\gamma := \min_i y_i f(\mathbf{x}_i)$

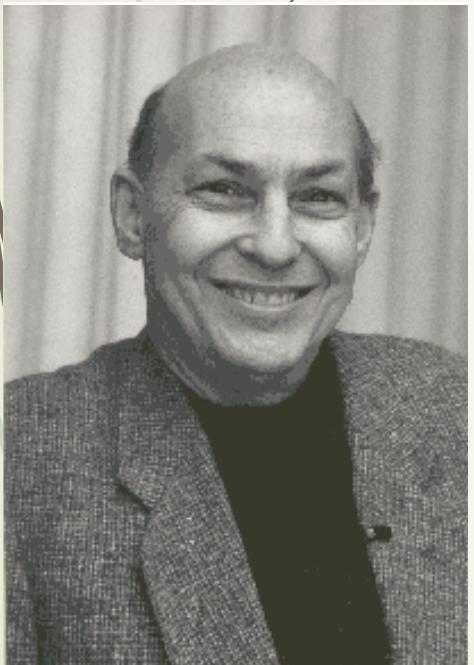


# Locality or Sparsity of Computation

Minsky and Papert, 1969

Perceptron can't do **XOR** classification

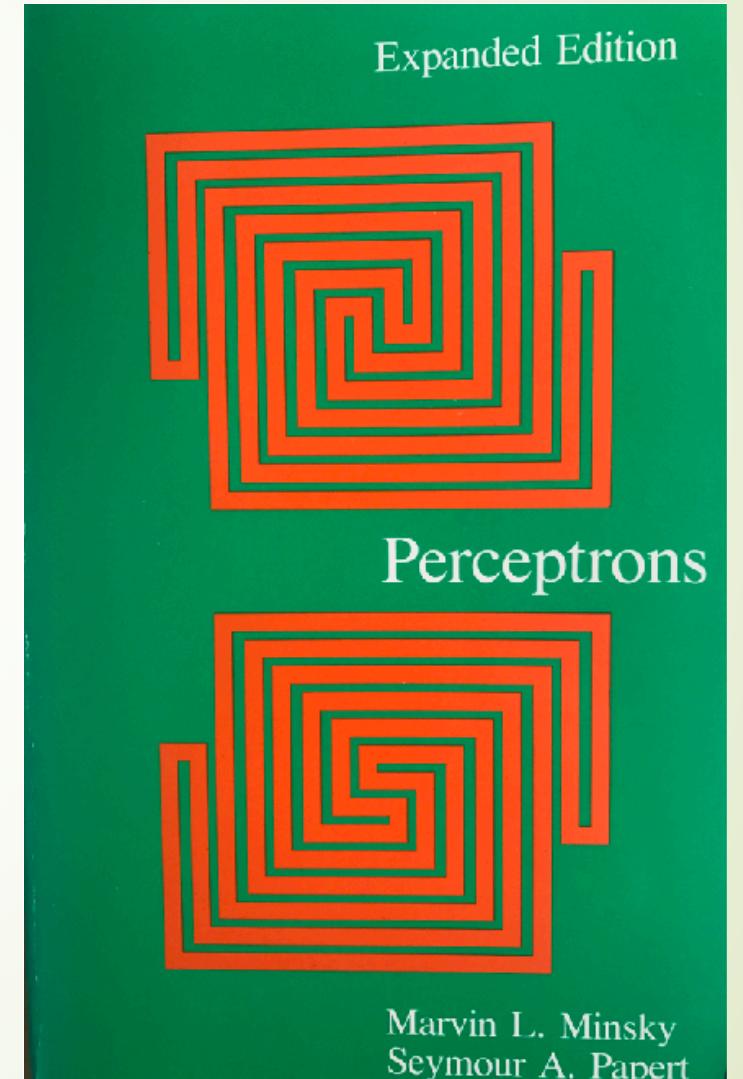
Perceptron needs infinite global  
information to compute **connectivity**



**Locality or Sparsity** is important:

Locality in time?

Locality in space?



Marvin L. Minsky  
Seymour A. Papert

# Multilayer Perceptrons (MLP) and Back-Propagation (BP) Algorithms

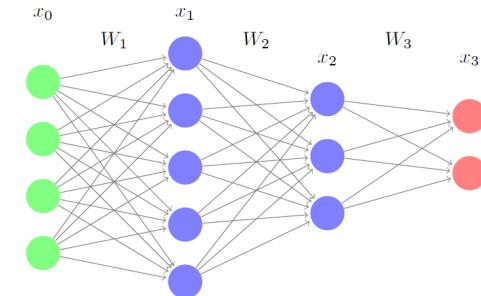
Rumelhart, Hinton, Williams (1986)

Learning representations by back-propagating errors, Nature, 323(9): 533-536

BP algorithms as **stochastic gradient descent** algorithms (**Robbins–Monro 1950; Kiefer–Wolfowitz 1951**) with Chain rules of Gradient maps

MLP classifies **XOR**, but the global hurdle on topology (**connectivity**) computation still exists: condition number in **Blum-Shub-Smale** real computation models helps.

NATURE VOL. 323 9 OCTOBER 1986 LETTERS TO NATURE 533



## Learning representations by back-propagating errors

David E. Rumelhart\*, Geoffrey E. Hinton† & Ronald J. Williams\*

\* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA  
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

more difficult when we introduce hidden units whose actual or desired states are not specified by the task. (In perceptrons, there are 'feature analysers' between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning units should be active in order to help achieve the desired input-output behaviour. This amounts to deciding what these units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate learned representations.

The simplest form of the learning procedure is for layered networks which have a layer of input units at the bottom, any number of intermediate layers, and a layer of output units at the top. Connections within a layer or from higher to lower layers are forbidden, but connections can skip intermediate layers. An input vector is presented to the network by setting the states of the input units. Then the states of the units in each layer are determined by applying equations (1) and (2) to the connections coming from lower layers. All units within a layer have their states set in parallel, but different layers have their states set sequentially, starting at the bottom and working upwards until the states of the output units are determined.

The total input,  $x_j$ , to unit  $j$  is a linear function of the outputs,  $y_i$ , of the units that are connected to  $j$  and of the weights,  $w_{ij}$ , on these connections

$$x_j = \sum_i y_i w_{ij} \quad (1)$$

Units can be given biases by introducing an extra input to each unit which always has a value of 1. The weight on this extra input is called the bias and is equivalent to a threshold of the opposite sign. It can be treated just like the other weights.

A unit has a real-valued output,  $y_j$ , which is a non-linear function of its total input

$$y_j = \frac{1}{1 + e^{-x_j}} \quad (2)$$

\* To whom correspondence should be addressed

# BP Algorithm: Forward Pass

- Cascade of repeated [linear operation followed by coordinatewise nonlinearity]'s
- Nonlinearities: sigmoid, hyperbolic tangent, (recently) ReLU.

---

## Algorithm 1 Forward pass

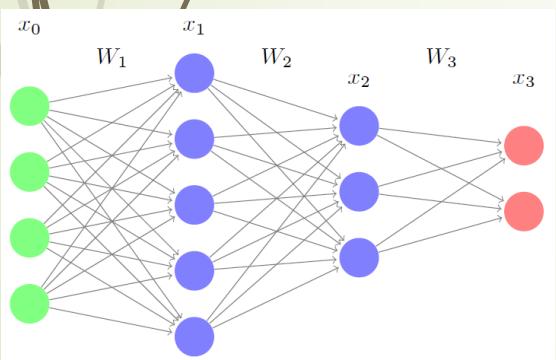
---

**Input:**  $x_0$

**Output:**  $x_L$

```
1: for  $\ell = 1$  to  $L$  do
2:    $x_\ell = f_\ell(W_\ell x_{\ell-1} + b_\ell)$ 
3: end for
```

---



# BP algorithm = Gradient Descent Method

- Training examples  $\{x_0^i\}_{i=1}^n$  and labels  $\{y^i\}_{i=1}^n$
- Output of the network  $\{x_L^i\}_{i=1}^m$
- Objective    Square loss, cross-entropy loss, etc.

$$J(\{W_l\}, \{b_l\}) = \frac{1}{n} \sum_{i=1}^n \frac{1}{2} \|y^i - x_L^i\|_2^2 \quad (1)$$

- Gradient descent

$$W_l = W_l - \eta \frac{\partial J}{\partial W_l}$$

$$b_l = b_l - \eta \frac{\partial J}{\partial b_l}$$

In practice: use Stochastic Gradient Descent (SGD)

# Derivation of BP: Lagrangian Multiplier

LeCun et al. 1988

Given  $n$  training examples  $(I_i, y_i) \equiv (\text{input}, \text{target})$  and  $L$  layers

- Constrained optimization

$$\min_{W,x} \quad \sum_{i=1}^n \|x_i(L) - y_i\|_2$$

$$\begin{aligned} \text{subject to} \quad & x_i(\ell) = f_\ell \left[ W_\ell x_i(\ell-1) \right], \\ & i = 1, \dots, n, \quad \ell = 1, \dots, L, \quad x_i(0) = I_i \end{aligned}$$

- Lagrangian formulation (Unconstrained)

$$\min_{W,x,B} \mathcal{L}(W, x, B)$$

$$\begin{aligned} \mathcal{L}(W, x, B) = \sum_{i=1}^n & \left\{ \|x_i(L) - y_i\|_2^2 + \right. \\ & \left. \sum_{\ell=1}^L B_i(\ell)^T \left( x_i(\ell) - f_\ell \left[ W_\ell x_i(\ell-1) \right] \right) \right\} \end{aligned}$$

# back-propagation – derivation

- $\frac{\partial \mathcal{L}}{\partial B}$

## Forward pass

$$x_i(\ell) = f_\ell \left[ \underbrace{W_\ell x_i(\ell-1)}_{A_i(\ell)} \right] \quad \ell = 1, \dots, L, \quad i = 1, \dots, n$$

- $\frac{\partial \mathcal{L}}{\partial x}, z_\ell = [\nabla f_\ell] B(\ell)$

## Backward (adjoint) pass

$$z(L) = 2\nabla f_L \left[ A_i(L) \right] (y_i - x_i(L))$$

$$z_i(\ell) = \nabla f_\ell \left[ A_i(\ell) \right] W_{\ell+1}^T z_i(\ell+1) \quad \ell = 0, \dots, L-1$$

- $W \leftarrow W + \lambda \frac{\partial \mathcal{L}}{\partial W}$

## Weight update

$$W_\ell \leftarrow W_\ell + \lambda \sum_{i=1}^n z_i(\ell) x_i^T(\ell-1)$$

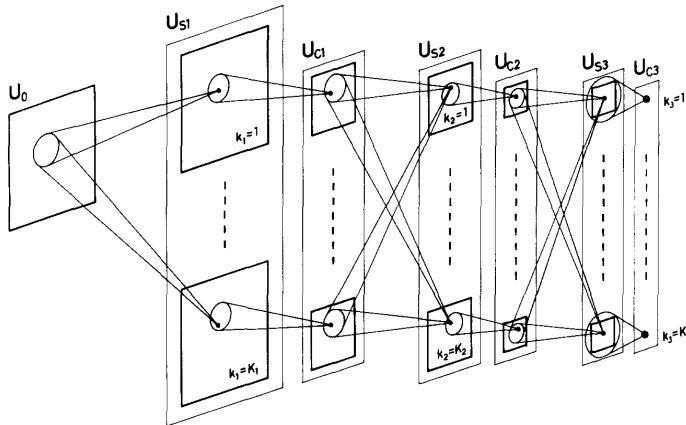
# Convolutional Neural Networks: shift invariances and locality

Biol. Cybernetics 36, 193–202 (1980)

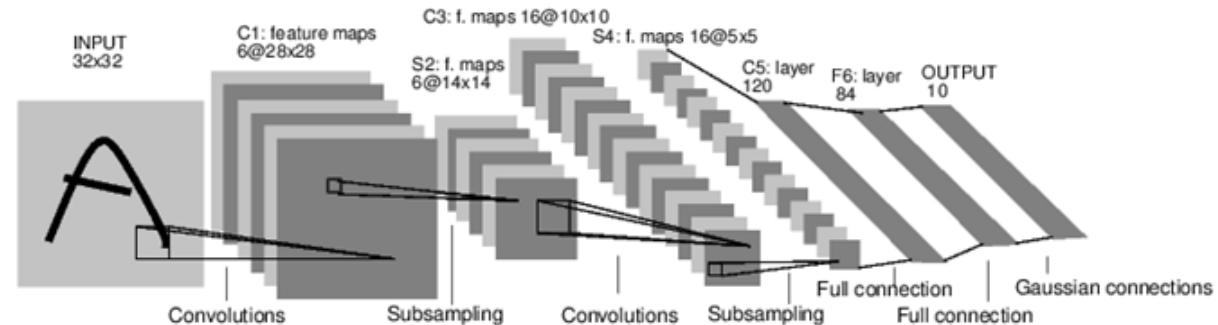
**Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position**

Kunihiko Fukushima

NHK Broadcasting Science Research Laboratories, Kinuta, Setagaya, Tokyo, Japan



- Can be traced to *Neocognitron* of Kunihiko Fukushima (1979)
- Yann LeCun combined convolutional neural networks with back propagation (1989)
- Imposes **shift invariance** and **locality** on the weights
- Forward pass remains similar
- Backpropagation slightly changes – need to sum over the gradients from all spatial positions

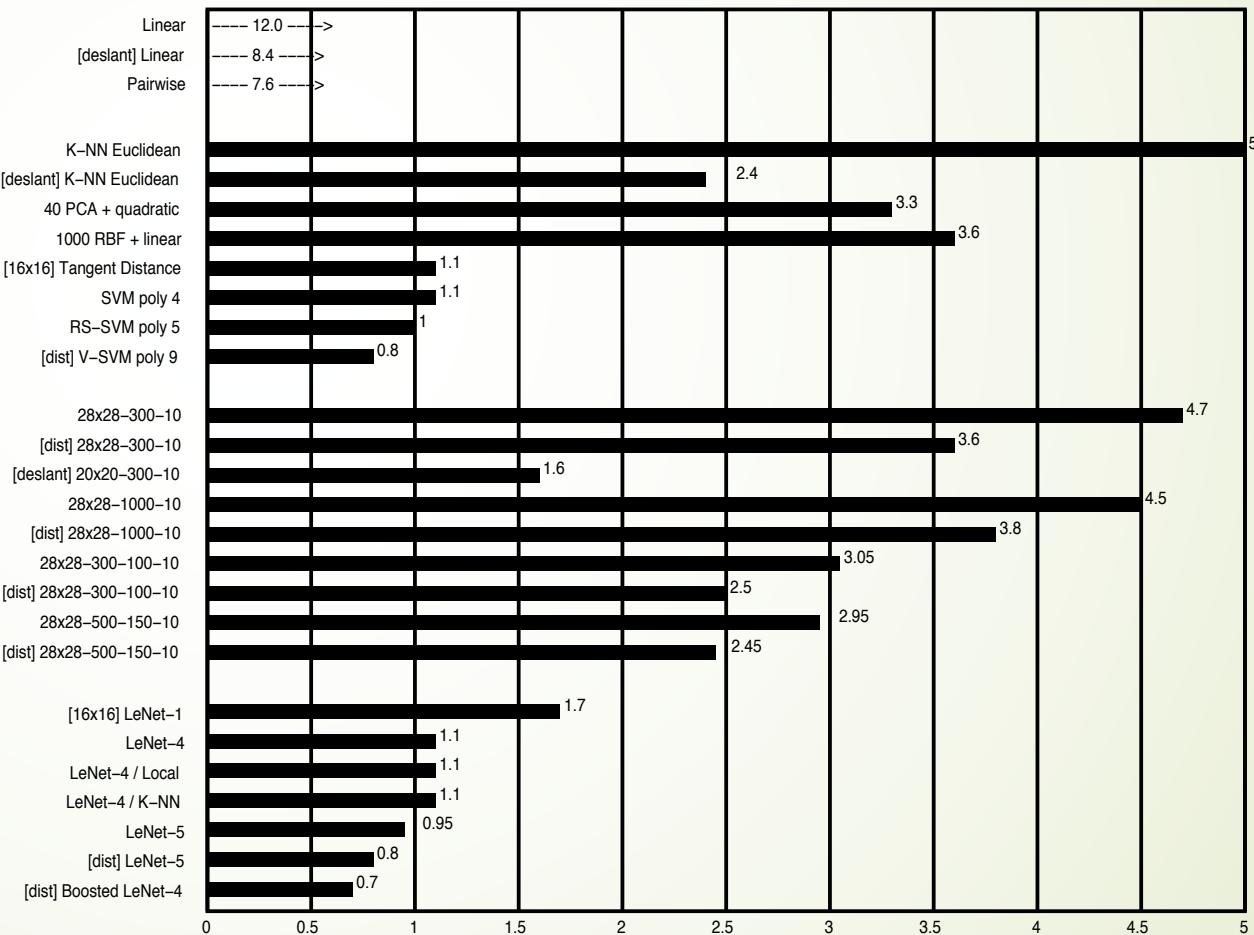


# MNIST Dataset Test Error

## LeCun et al. 1998

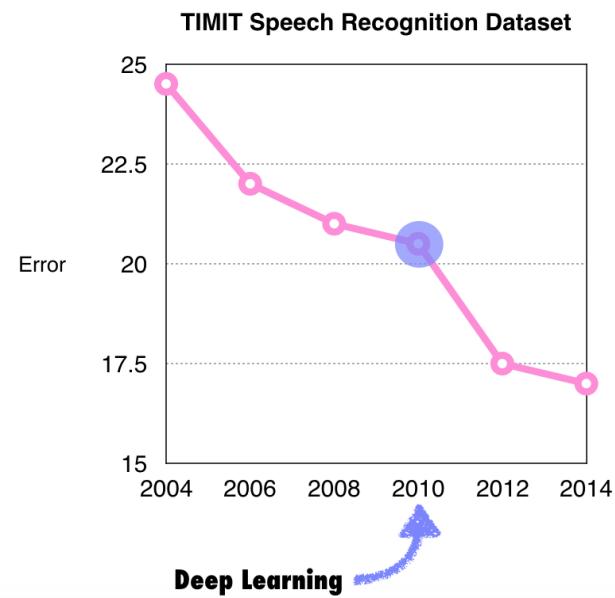
Simple SVM performs  
as well as Multilayer  
Convolutional Neural  
Networks which need  
careful tuning (LeNets)

Dark era for NN: 1998-2012

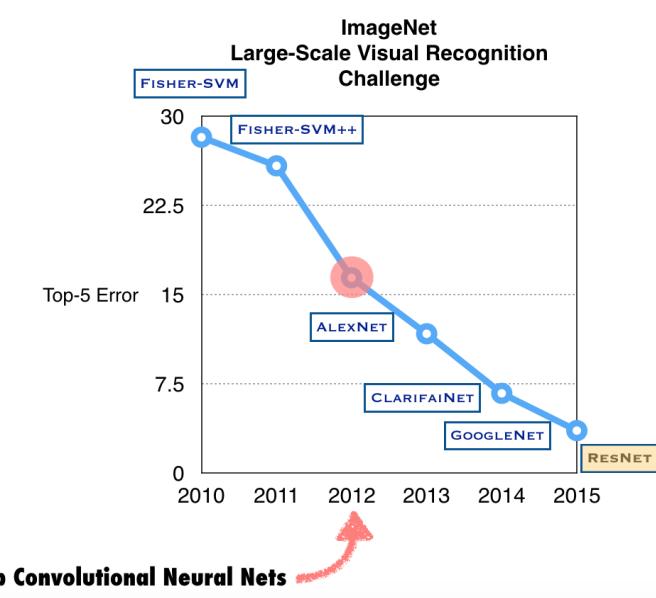


# Around the year of 2012: return of NN as `deep learning'

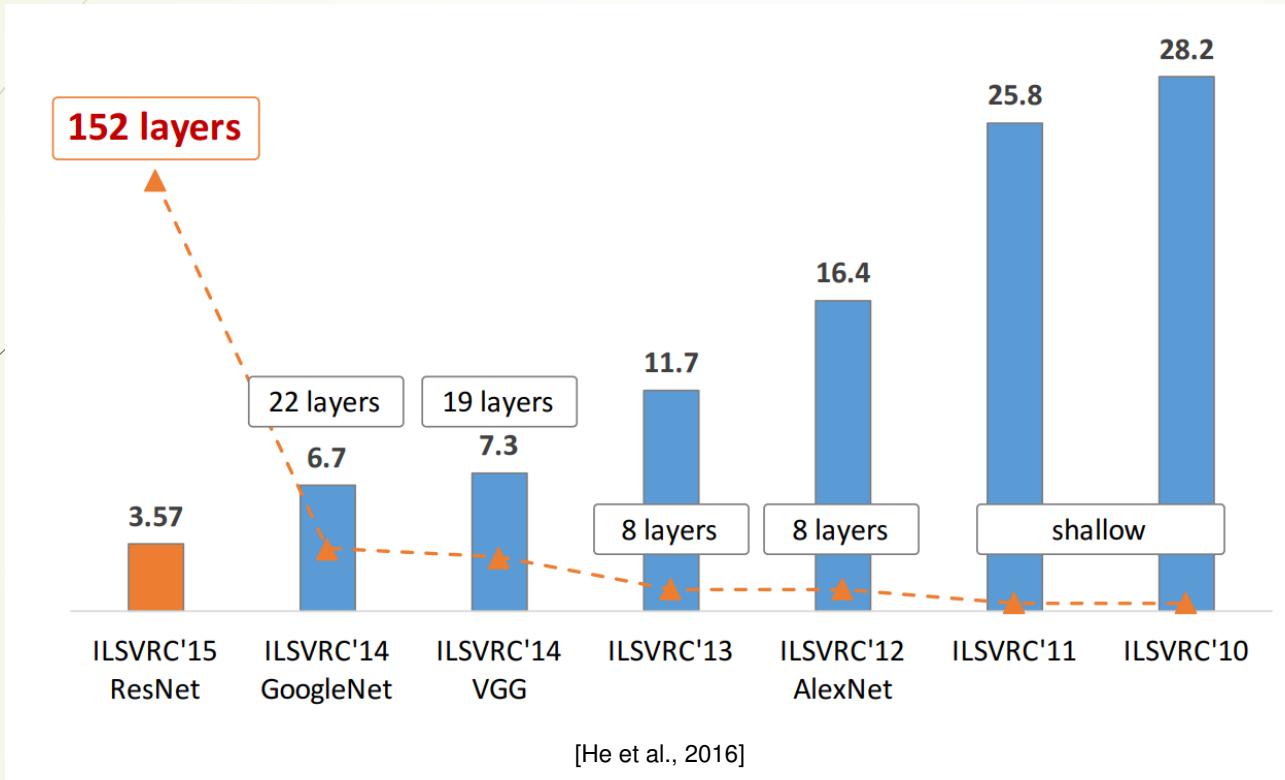
Speech Recognition: TIMIT



Computer Vision: ImageNet

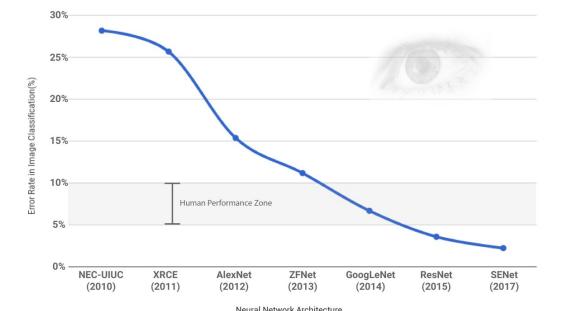


# Depth as function of year



## ILSVRC ImageNet Top 5 errors

- ImageNet (subset):
  - 1.2 million training images
  - 100,000 test images
  - 1000 classes
- ImageNet large-scale visual recognition Challenge

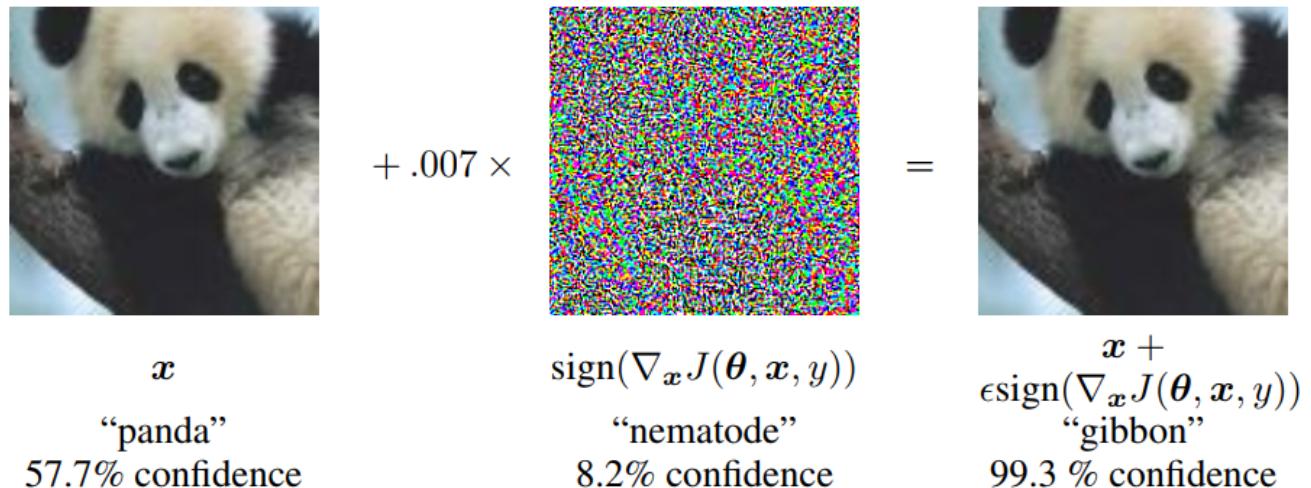


# Some Cold Water: Tesla Autopilot Misclassifies Truck as Billboard



**Problem:** Why? How can you trust a blackbox?

# Deep Learning may be fragile in generalization against noise!



[Goodfellow et al., 2014]

- Small but malicious perturbations can result in severe misclassification
- Malicious examples generalize across different architectures
- What is source of instability?
- Can we robustify network?

# What's wrong with deep learning?

**Ali Rahimi** NIPS'17: Machine (deep) Learning has become **alchemy**.  
<https://www.youtube.com/watch?v=ORHFOnaEzPc>

**Yann LeCun** CVPR'15, invited talk: **What's wrong with deep learning?**  
One important piece: **missing some theory!**

<http://techtalks.tv/talks/whats-wrong-with-deep-learning/61639/>



Being alchemy is certainly not a shame, not wanting to work on advancing to chemistry is a shame! -- **by Eric Xing**

# Generalization Ability

Why over-parameterized models may generalize well without overfitting?

# Generalization Error

- Consider the empirical risk minimization under i.i.d. samples

$$\hat{R}_n(\theta) = \frac{1}{n} \sum_{i=1}^n \ell(y_i, f(x_i; \theta)) + \mathcal{R}(\theta)$$

- The population risk with respect to unknown distribution

$$R(\theta) = \mathbf{E}_{x,y \sim P} \ell(y, f(x; \theta))$$

- Fundamental Theorem of Machine Learning (for 0-1 misclassification loss, called 'errors' below)

$$R(\theta) = \underbrace{\hat{R}_n(\theta)}_{\text{training loss/error}} + \underbrace{R(\theta) - \hat{R}_n(\theta)}_{\text{generalization loss/error}}$$

# Why big models generalize well?



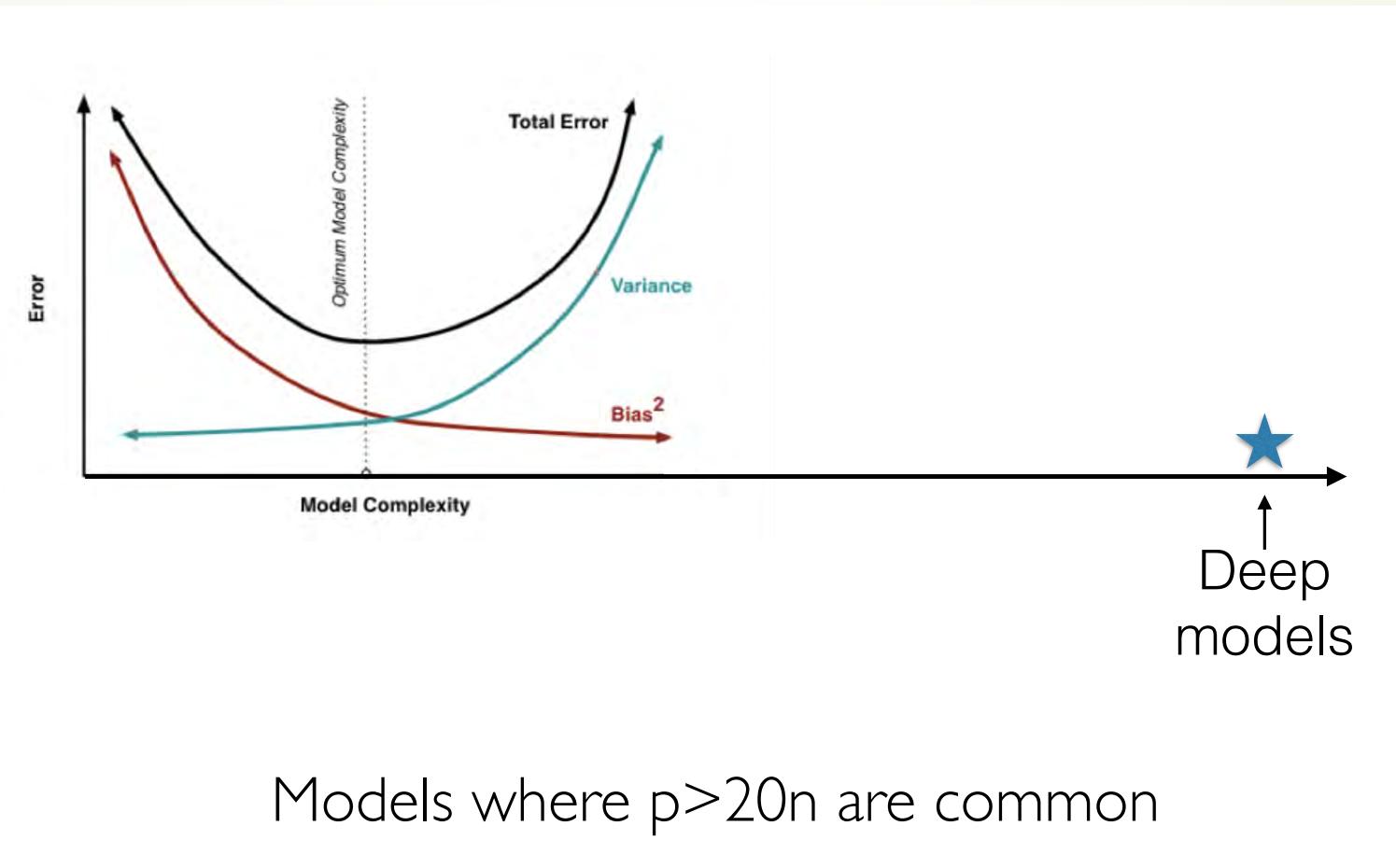
CIFAR10

n=50,000  
d=3,072  
k=10

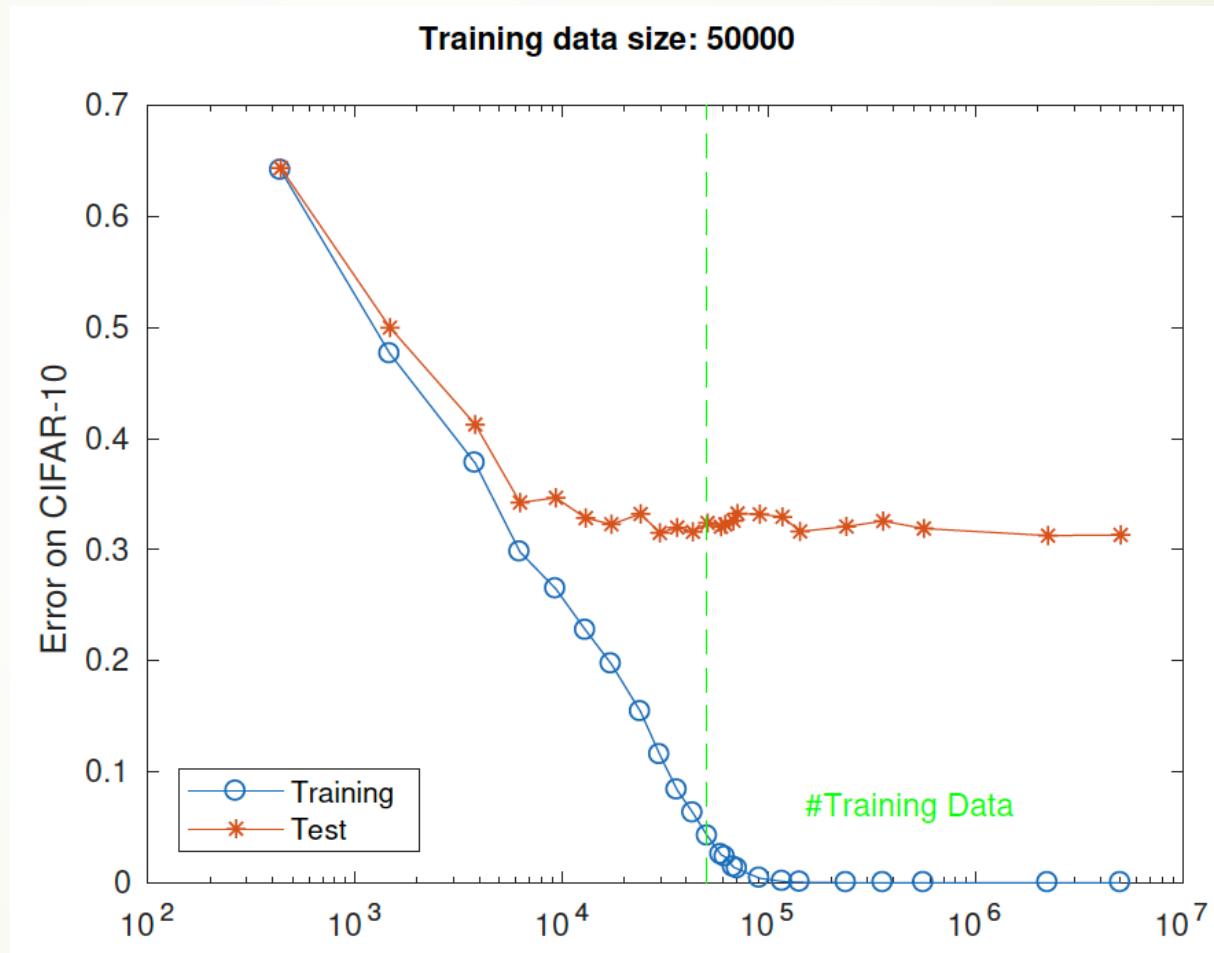
What happens when I turn off the regularizers?

<u>Model</u>	<u>parameters</u>	p/n	Train <b>loss</b>	Test <b>error</b>
CudaConvNet	145,578	2.9	0	23%
CudaConvNet (with regularization)	145,578	2.9	0.34	18%
MicroInception	1,649,402	33	0	14%
ResNet	2,401,440	48	0	13%

# The Bias-Variance Tradeoff?



# Over-parameterized models



As model complexity grows ( $p > n$ ), training error goes down to zero, but test error does not increase. Why overparameterized models do not overfit here? -- Tommy Poggio, 2018

# Some tentative answers:

- ▶ **Belkin** et al.: **Interpolation (overfitting)** has a low generalization error in overparameterization regime
  - ▶ <https://simons.berkeley.edu/talks/tbd-65>
- ▶ For overparameterized linear regression models:
  - ▶ **Peter Bartlett** et al. <https://simons.berkeley.edu/talks/tbd-51>
  - ▶ **Trevor Hastie** et al. asymptotic theory based on random matrix theory
- ▶ For logistic regressions:
  - ▶ **Telgarsky, Srebro**, et al. GD converges to max margin solution
- ▶ Nonlinear neural networks: ???
- ▶ Some warnings on “interpolations”:
  - ▶ Ben Recht: <https://simons.berkeley.edu/talks/tbd-63>

# Some Theoretical Problems

- ▶ Approximation Theory and Harmonic Analysis : **What functions are represented well by deep neural networks, without suffering the curse of dimensionality and better than shallow networks?**
  - ▶ Sparse (local), hierarchical (multiscale), compositional functions avoid the curse of dimensionality
  - ▶ Group (translation, rotational, scaling, deformation) invariances achieved as depth grows
- ▶ Statistics learning: **How can deep learning generalize well without overfitting the noise?**
  - ▶ “Benign overfitting”? ...
- ▶ Optimization: **What is the landscape of the empirical risk and how to optimize it efficiently?**
  - ▶ Wide networks may have simple landscape for GD/SGD algorithms ...

Thank you!

