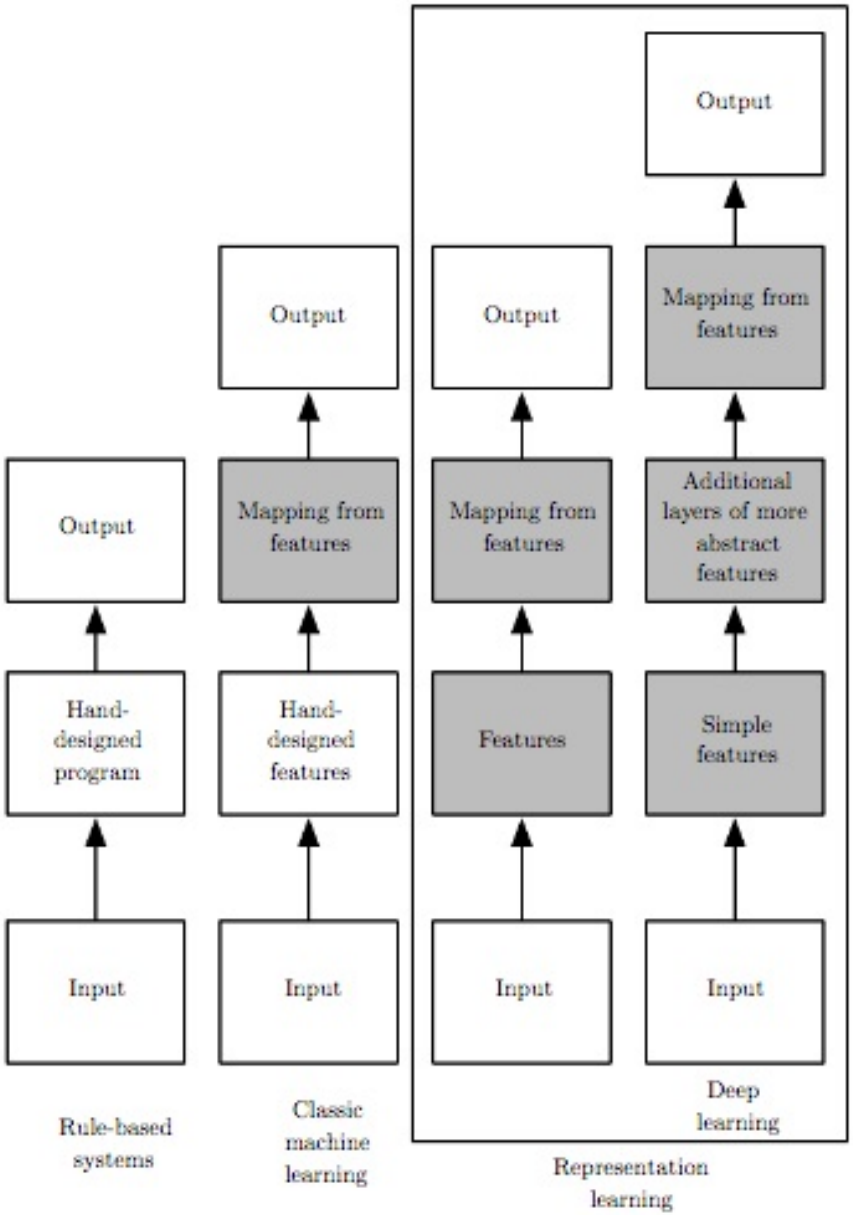
INSEAD PhD Course

Foundations of Machine Learning and AI

Theodoros Evgeniou - Nicolas Vayatis

Sessions 9-10: Deep Learning

"Big picture" of Learning



Machine Learning

What we have seen so far

Supervised machine learning
Setup

- Data: $(X_1, Y_1), \ldots, (X_1, Y_1)$ with X_i being a vector of variables (factors) for observation i, and Y_i being the label of X_i
- Hypothesis class: set of functions $h \in \mathcal{H}$
- Loss of a function h at a data point (X, Y):

$$\ell(h(X), Y) \geq 0$$

- *Empirical risk* of a function h over the data:

$$\widehat{L}_n(h) = \frac{1}{n} \sum_{i=1}^{n} \ell(h(X_i), Y_i)$$

Machine Learning Methods
Examples seen so far

ML Method: Minimize some error (loss) on the data + Hypothesis space controlled using Regularization. Solution using an Optimization method.

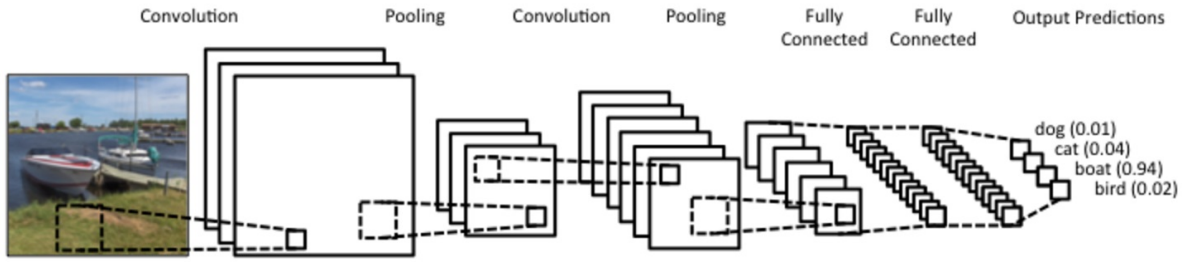
- General hypothesis space with function selected through *Empirical Risk Minimization* (ERM) (could be shallow or deep)
- (Sparse) Linear models with parameters estimated through (penalized) least square minimization
- Kernel ridge regression - Exercise: What are the hypothesis space, the loss, the regularization, and the optimization method in that case?

The latter is an example of popular and efficient shallow learning method.

Deep Learning:

Introducing the main concepts

Deep Feedforward Network

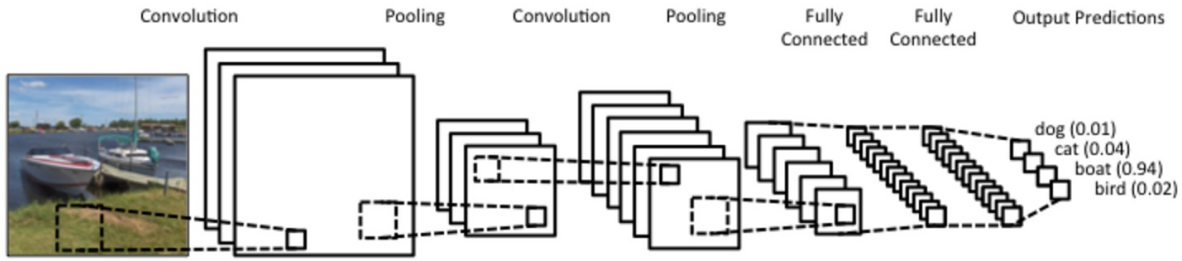


- Hypothesis space: functions of the form

$$h(x, \theta) = \sigma_m \circ A_m \circ \sigma_{m-1} \circ \ldots \circ A_2 \circ \sigma_1 \circ A_1 x$$

  where $\theta = \left(A_1, \ldots, A_m\right)$ sequence of parameters to be estimated through learning
- We denote by $\sigma = \left(\sigma_1, \ldots, \sigma_m\right)$ the so-called activation functions which are hyperparameters related to the choice of a network architecture (which includes the number and size of the layers - see below).

Deep Feedforward Network



- Optimization objective (far from convex! where is the regularizer?):

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \ell(h(X_i, \theta), Y_i)$$

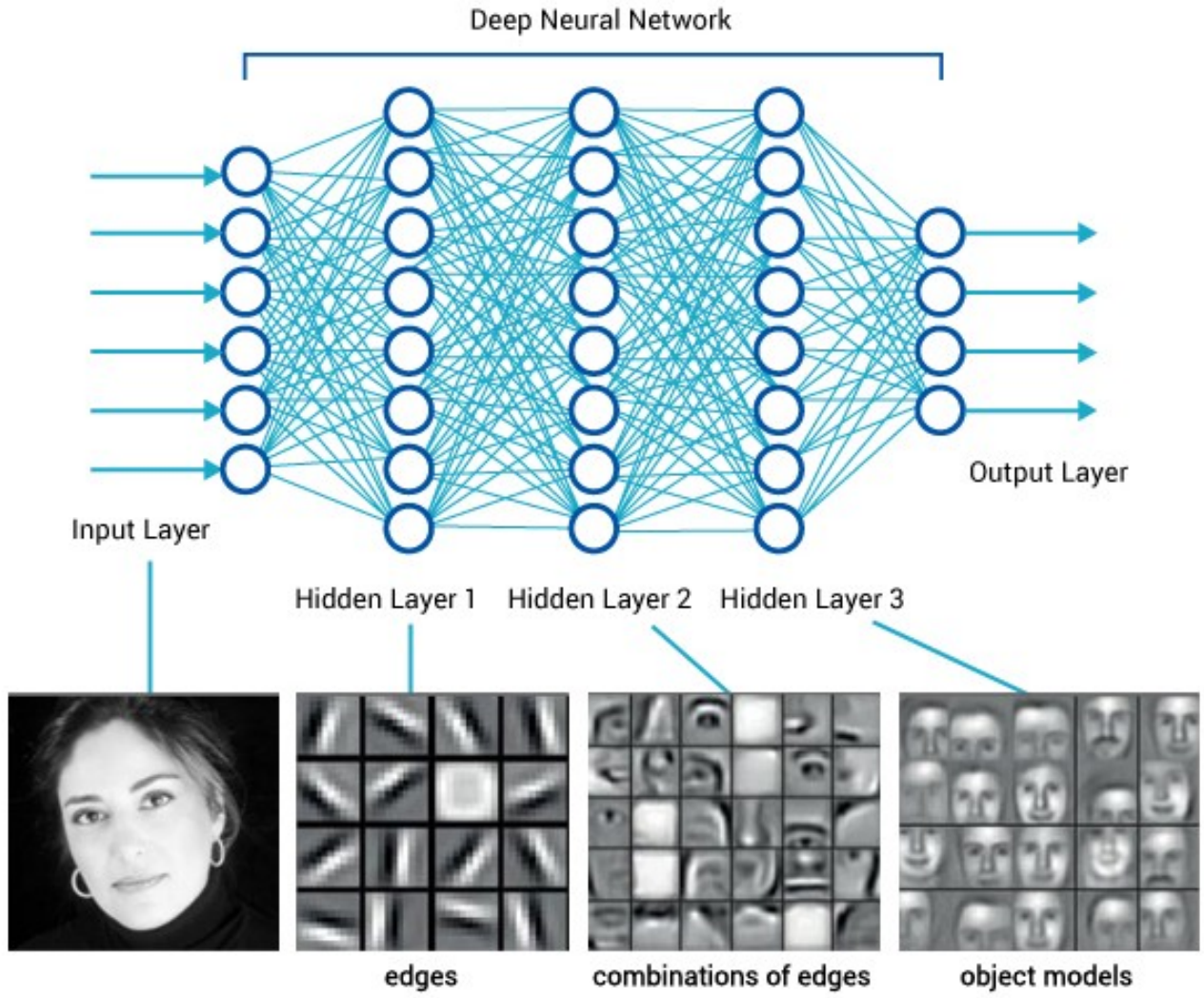
- Optimization method based on stochastic gradient descent (iterates over data points)

$$\theta_{i+1} = \theta_i - \eta \frac{\partial \ell(h(X_i, \theta), Y_i)}{\partial \theta}(\theta_i)$$

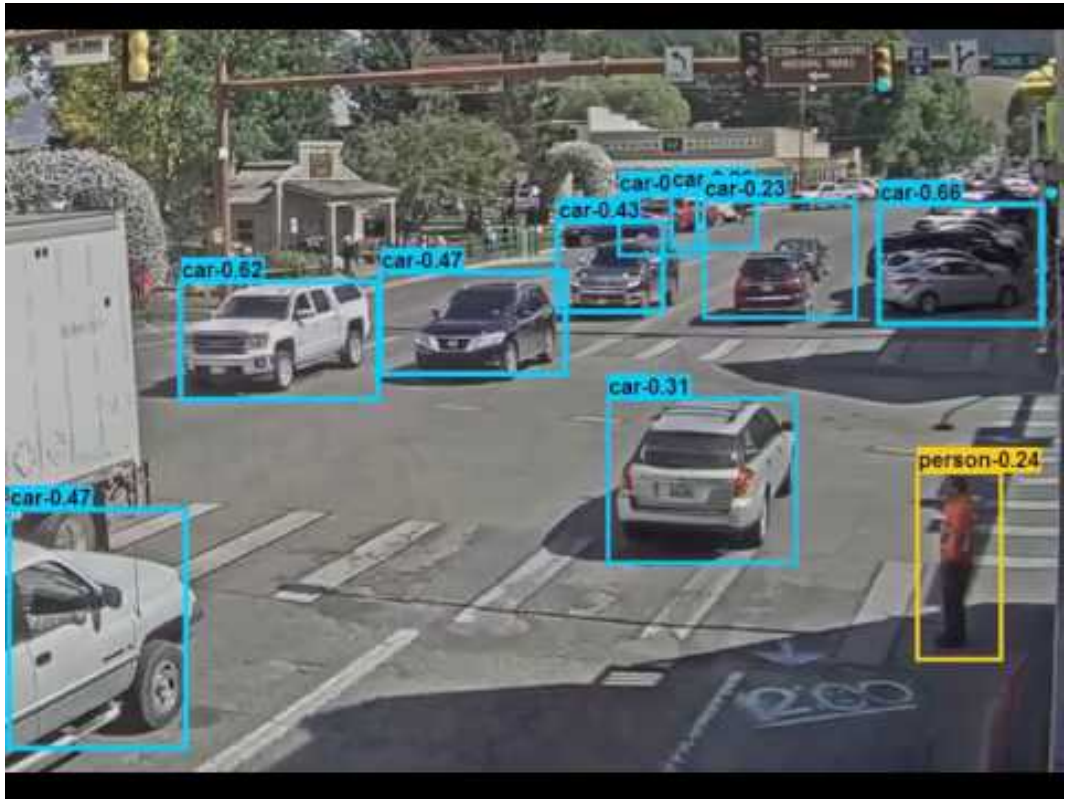
Deep Learning:

Why is it popular?

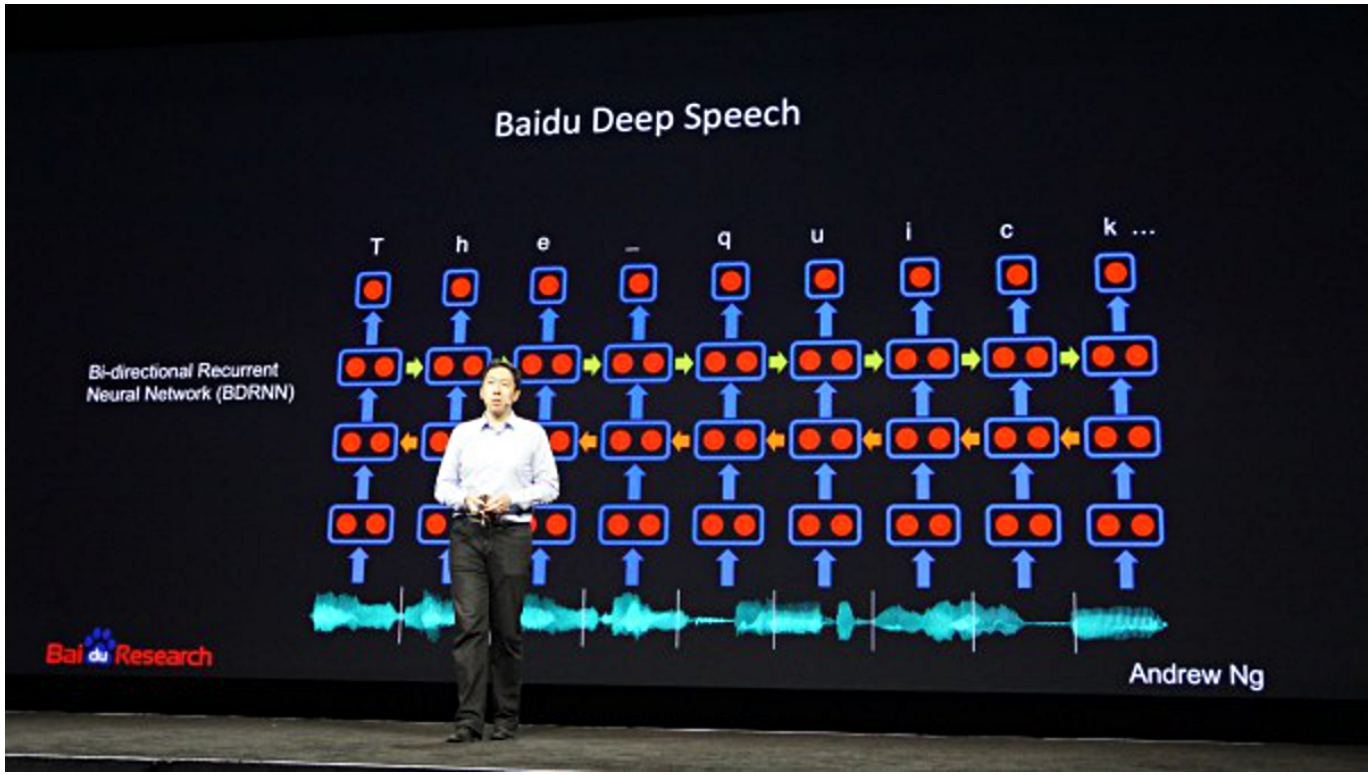
Deep means 'many layers' (compositions) between input and output...



Success of deep learning (1/3)
Computer Vision

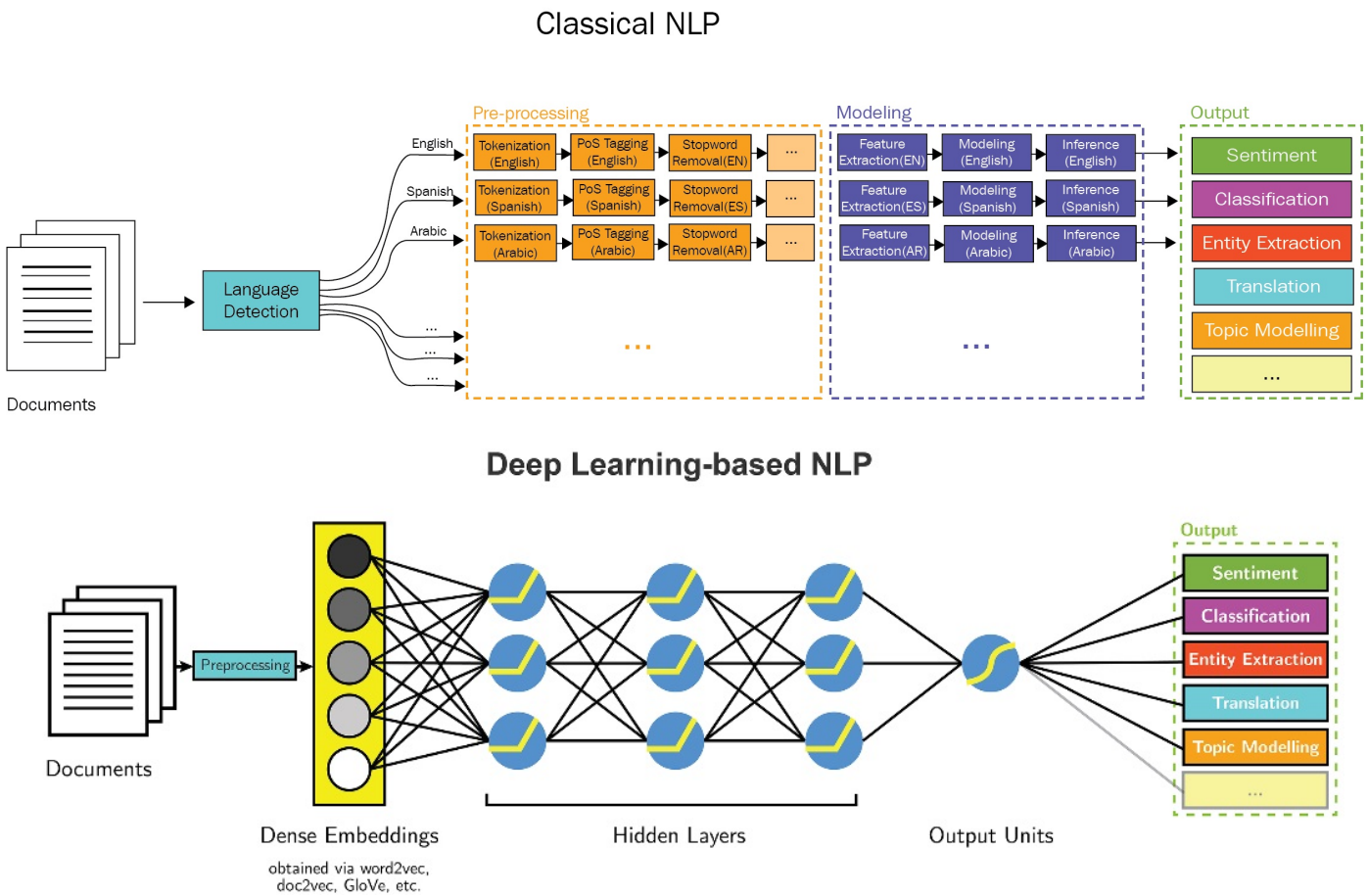


Success of deep learning (2/3)
Speech recognition



System	Clean (94)	Noisy (82)	Combined (176)
Apple Dictation	14.24	43.76	26.73
Bing Speech	11.73	36.12	22.05
Google API	6.64	30.47	16.72
wit.ai	7.94	35.06	19.41
Deep Speech	**6.56**	**19.06**	**11.85**

Success of deep learning (3/3)
Natural Language processing



Shallow vs. Deep Learning
Vapnik vs. LeCun

The first algorithms to reach human performance on a visual task

- LeCun, Boser, et al. (1989). Backpropagation Applied to Handwritten Zip Code Recognition, in Neural Computation.

  Architecture: 1000 units - 70,000 connections

- C. Cortes and V. Vapnik (1995). Support-Vector Networks, in Machine Learning Journal.

  Architecture: 1 kernel - 2 parameters

USPS ZIP code database

Goal for the class today

- Develop insights about deep learning and neural networks: when it works and when it does not work, and what it means to "work" (open discussion)
- Practical guide to deep learning optimization and engineering
- Learn about the three mysteries of deep learning... and connect to the machine learning concepts seen so far (such as approximation error, complexity, and regularization)

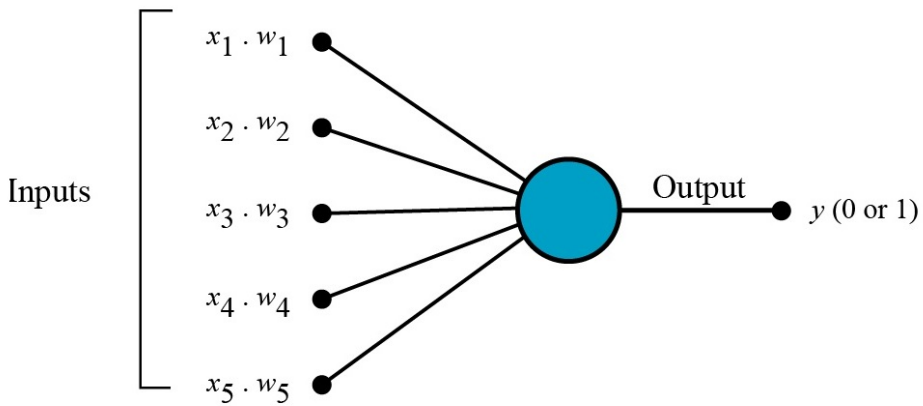
Historical perspective on neural networks

- Cybernetics (1940s-1960s)
  - Achievement: modeling and training one neuron
  - Key algorithm: Perceptron
  - Paper: Rosenblatt (1958)
- Connectionism (1980s)
  - Achievement: training one or two hidden layers
  - Key algorithm: Backpropagation
  - Paper: Rumelhart-Hinton-Williams (1986)
- Deep Learning (2007-....)
  - Achievement: training multiple layers of representation
  - Key algorithm: Stochastic gradient
  - Papers: Hinton (2006), Bengio-LeCun (2007)

First wave: 1960s

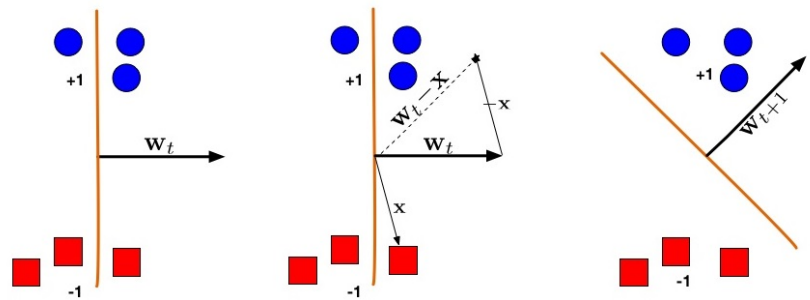
The Perceptron

Primitive neural network
Single neuron Perceptron



Batch Perceptron

input: A training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
initialize: $\mathbf{w}^{(1)} = (0, \ldots, 0)$
  for $t = 1, 2, \ldots$
    if $(\exists\ i \text{ s.t. } y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle \leq 0)$ then
      $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + y_i \mathbf{x}_i$
    else
      output $\mathbf{w}^{(t)}$



Second wave: 1980's

Multilayer perceptrons

1. Theory: Universal approximators
2. Algorithm: Backpropagation algorithm

Second wave: 1980's

Multilayer perceptrons

1. Existence theorems of universal approximators

Stone-Weierstrass theorem

- Consider any continuous function $f : [a, b] \to \mathbb{R}$, then for any $\varepsilon > 0$, there exists a polynomial P such that:

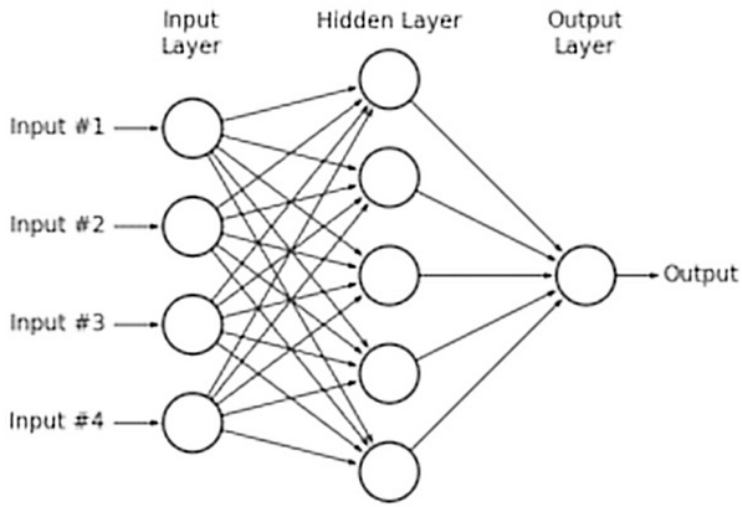
$$\sup_{x \in [a,b]} |f(x) - P(x)| < \varepsilon \ .$$

Single-Layer Neural Network Definition

- Single-layer neural network: Let σ a 'smooth' activation function. A single-layer neural network with N units and an *activation function* σ, is a function of this form:

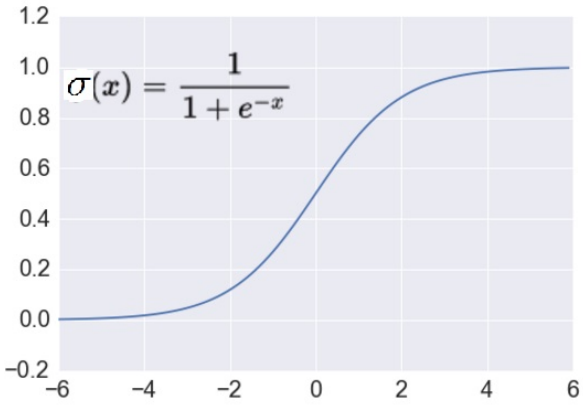
$$h(x) = \sum_{k=1}^{N} \sigma(a_k^T x + b), \ \forall x \in \mathbb{R}^d$$

where $a \in \mathbb{R}^d$, $b \in \mathbb{R}$, N integer. The number N corresponds to the number of *units* in the hidden layer of the network.

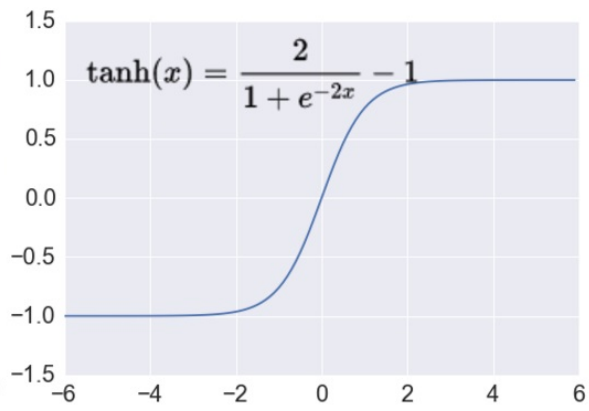


Activation function
Examples

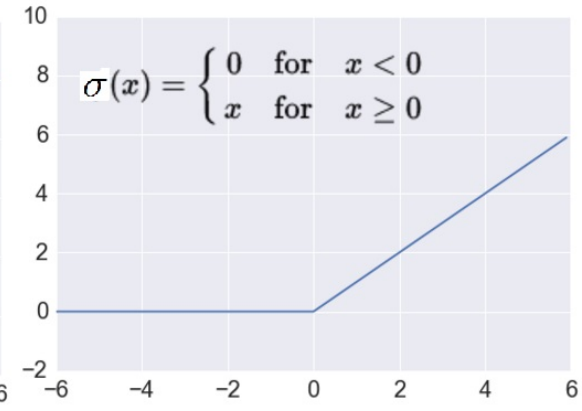
Sigmoid



TanH



ReLU



Single-Layer Neural Network are universal approximators

- Cybenko's theorem: consider any continuous function $f : [0,1]^d \to \mathbb{R}$, then for any $\varepsilon > 0$, there exists a single-layer neural network $h(x) = \sum_{k=1}^{N} \sigma(a_k^T x + b)$ (i.e. some N, a, b) such that:

$$\sup_{x \in [a,b]} |f(x) - h(x)| < \varepsilon \ .$$

- Further work by Hornik-Stinchcombe-White (1989), Barron (1993).

2. Backpropagation algorithm:

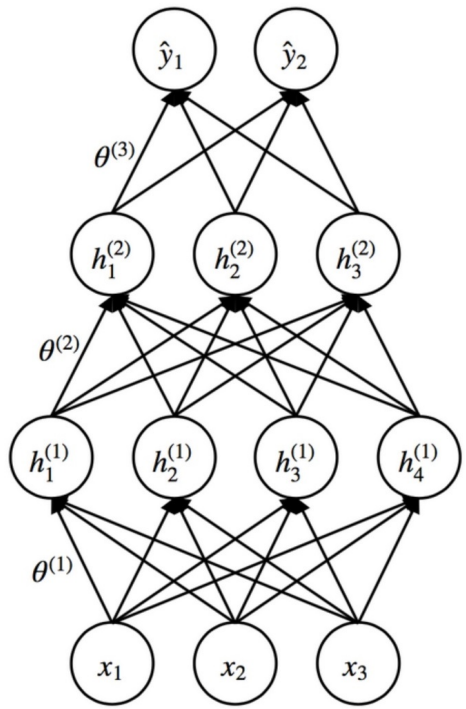
The key to multilayer perceptron calibration

Multilayer perceptron
More than one hidden layer!

- Hypothesis space: functions of the form

$$h(x, \theta) = \sigma \circ A_m \circ \sigma \circ \ldots \circ A_2 \circ \sigma \circ A_1 x$$

where $\theta = (A_1, \ldots, A_m)$ sequence of parameters to be estimated through learning and σ activation function applied componentwise



Backpropagation Principle

- Consider the square loss, then given a weight vector θ_1, we can evaluate the error as:

$$\mathcal{L}(\theta_1) = \frac{1}{n}\sum_{i=1}^{n}\big(h(X_i, \theta_1) - Y_i\big)^2$$

- The idea is to propagate the error backwards in the network to update θ_1 by the following rule:

$$\theta_2 = \theta_1 - \eta \nabla_\theta \mathcal{L}(\theta_1)$$

where η is the so-called learning rate.

Backpropagation
Background: Chain rule

- Consider the composition of three functions:

$$f(u) = \ell \circ \sigma \circ g(x)$$

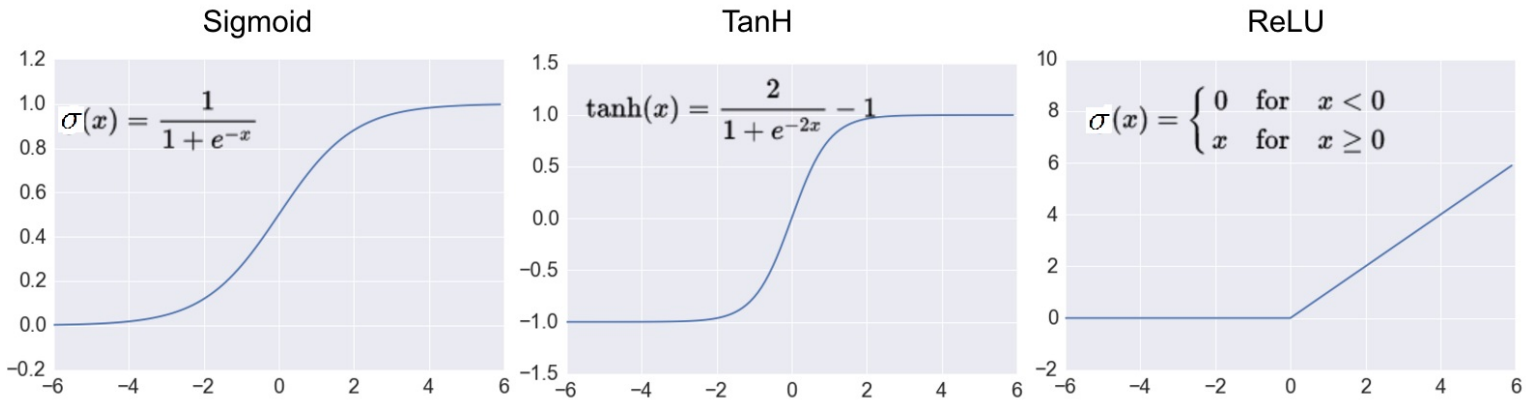
with $t = \sigma \circ g(u)$ and $z = g(u)$ (everything in $\mathbb{R}$ here)

- The chain rule provides the expression for the derivative of f:

$$\frac{df}{du}(u) = \frac{d\ell}{dt}(t)\frac{d\sigma}{dz}(z)\frac{dg}{du}(u) = \ell'(\sigma \circ g(u))\sigma'(g(u))g'(u)$$

Backpropagation
Background: Activation function

- Typical examples:



- For the logistic activation function: $\sigma(z) = \dfrac{1}{1+e^{-z}}$, we have by standard algebra:

$$\sigma'(z) = \frac{d\sigma}{dz}(z) = \sigma(z)(1-\sigma(z))$$

Backpropagation
Toy example: the single unit case

- Consider a single unit (neuron): $h(x, a) = \sigma(a^T x)$ which is connected to the output Y of the network
- The error of the predictions produced by this neuron on the training data is the following:

$$\mathcal{L}(a) = \frac{1}{n} \sum_{i=1}^{n} \ell(\sigma(a^T X_i), Y_i)$$

where $\ell(t, y) = (t - y)^2$ considering the square loss here.

Backpropagation
Gradient computation

- Apply the chain rule with three compositions in the case where the last function is linear
- The gradient of $\mathcal{L}$ wrt a is given by:

$$\frac{\partial \mathcal{L}}{\partial a_j}(a) = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial \ell}{\partial t}(\sigma(a^T X_i), Y_i)\sigma'(a^T X_i)X_{ij}$$

Backpropagation
Weight update

- Special case here: square loss, logistic activation function

$$\frac{\partial \ell}{\partial t}(t, Y_i) = 2(t - Y_i) \quad \text{and } \sigma'(z) = \sigma(z)(1 - \sigma(z))$$

- The gradient update applied at input 'neurons' is the following

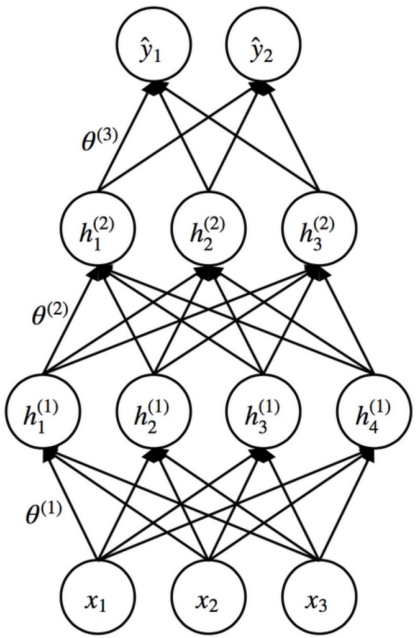
$$\frac{\partial \mathcal{L}}{\partial a_j}(a) = \frac{2}{n} \sum_{i=1}^{n} (z_i - Y_i) z_i (1 - z_i) X_{ij}$$

where $z_i = \sigma(a^T X_i)$

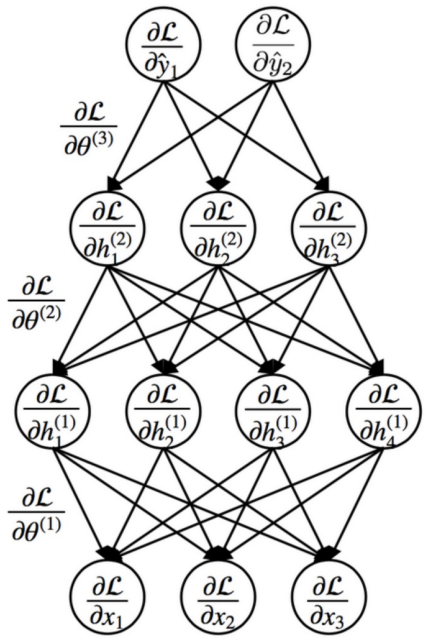
Backpropagation
General case

- In case of multiple layers, it suffices to apply the chain rule upstream
  For more details: check Lecture notes by Jake Abernethy

  `https://nbviewer.jupyter.org/format/slides/github/thejakeyboy/umich-eecs545-lectures/`



Forward



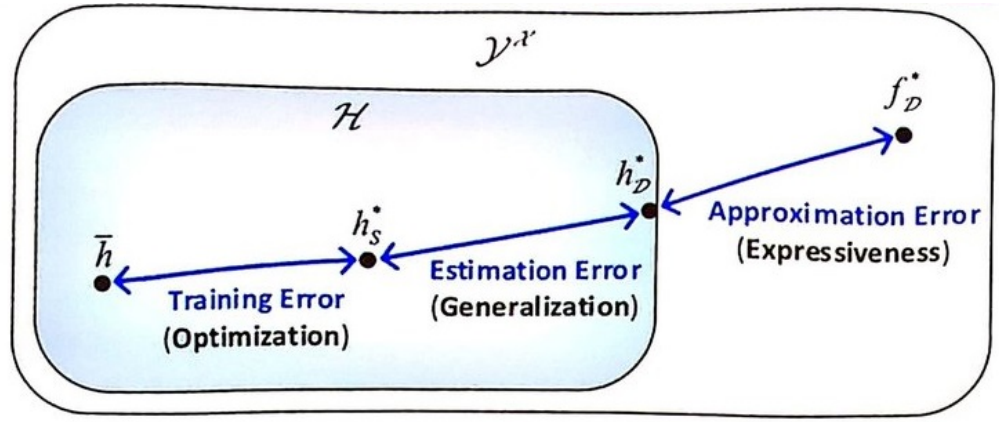
Backward

- But does it converge to the optimal solution?

A new trade-off in Machine Learning
The three terms

$$\begin{aligned}\mathcal{E} &= \mathbb{E}\left[E(f_{\mathcal{F}}^*) - E(f^*)\right] + \mathbb{E}\left[E(f_n) - E(f_{\mathcal{F}}^*)\right] + \mathbb{E}\left[E(\tilde{f}_n) - E(f_n)\right] \\ &= \mathcal{E}_{\text{app}} + \mathcal{E}_{\text{est}} + \mathcal{E}_{\text{opt}}.\end{aligned}$$

		$\mathcal{F}$	n	ρ
$\mathcal{E}_{\text{app}}$	(approximation error)	$\searrow$		
$\mathcal{E}_{\text{est}}$	(estimation error)	$\nearrow$	$\searrow$	
$\mathcal{E}_{\text{opt}}$	(optimization error)	$\cdots$	$\cdots$	$\nearrow$
T	(computation time)	$\nearrow$	$\nearrow$	$\searrow$



$f_{\mathcal{D}}^*$ – ground truth ($\text{argmin}_{f \in \mathcal{Y}^{\mathcal{X}}} L_{\mathcal{D}}(f)$)

$h_{\mathcal{D}}^*$ – optimal hypothesis ($\text{argmin}_{h \in \mathcal{H}} L_{\mathcal{D}}(h)$)

h_S^* – empirically optimal hypothesis ($\text{argmin}_{h \in \mathcal{H}} L_S(h)$)

$\bar{h}$ – returned hypothesis

Here: n sample size, ρ numerical tolerance in the optimization

Third wave: 2010s

From shallow to deep networks

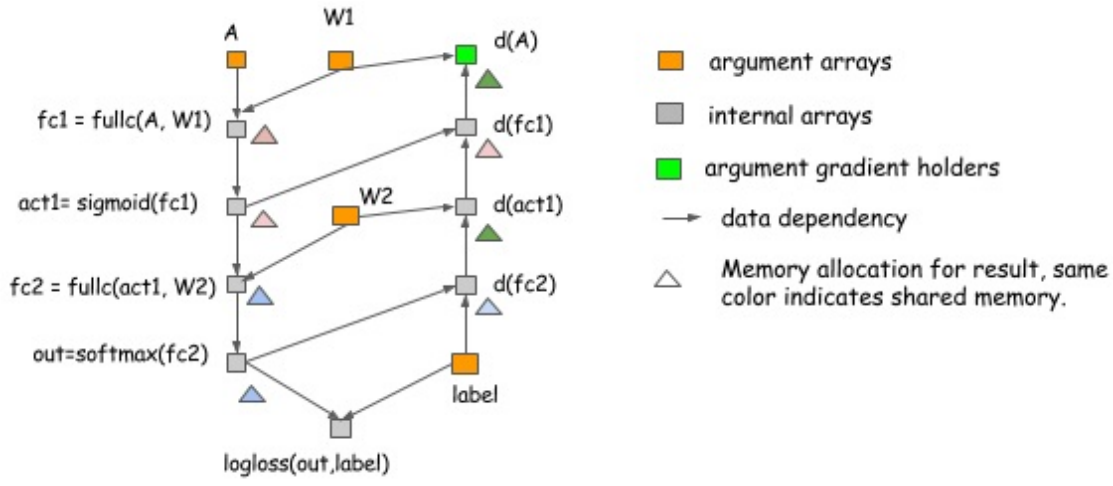
1. How to build deep networks
2. The mysteries of deep learning

From shallow to deep networks

1. How to build deep networks

Engineering of deep learning

- Software environments for deep learning designed as computational graphs (Theano, Keras, TensorFlow...)



- A computational graph is a way to represent a math function in the language of graph theory.
- In a computational graph nodes are either input values or functions for combining values.

Edges receive their weights as the data flows through the graph. Outbound edges from an input node are weighted with that input value; outbound nodes from a function node are weighted by combining the weights of the inbound edges using the specified function.

Regularization in deep learning and why DL theory is difficult

Implicit in the objective, but lots of engineering tricks in the computational graph:

- Weight decay
- Weight sharing
- Early stopping
- Model averaging
- Dropout
- Data augmentation
- Adversarial training

Implementation of Deep Learning Examples on github

- https://github.com/enggen/Deep-Learning-Coursera/
- https://github.com/aymericdamien/TensorFlow-Examples/

The design problem (1/2)
Setup

- Denote by T the structural parameters of the deep network (architecture, activity functions, regularization modes...) and $\hat{f}_T$ the function produced by deep learning given T
- Some estimate of the predictive error $\hat{L}(\hat{f}_T)$ of the function supposed to be available (can be estimated by hold out, cross validation...).
- Finding T is key to address the estimation-approximation tradeoff

The design problem (2/2)
Selecting the structure

- Selecting the structure of a deep network is a meta-learning problem
- The optimal architecture can be obtained if it is possible to solve the following optimization problem:

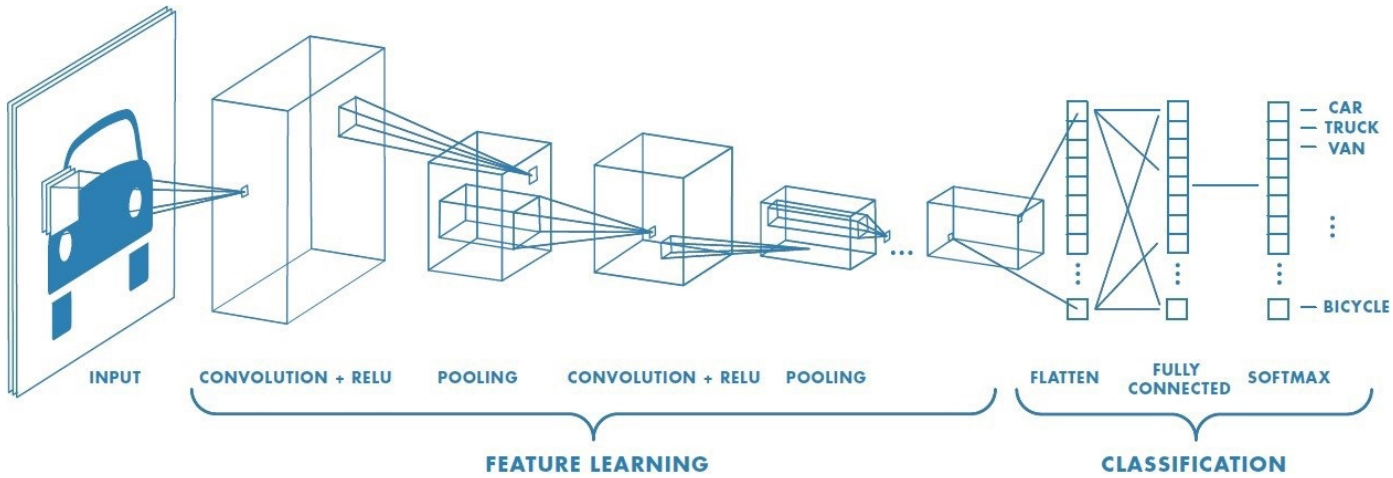
$$\min_{T} \hat{L}(\hat{f}_T)$$

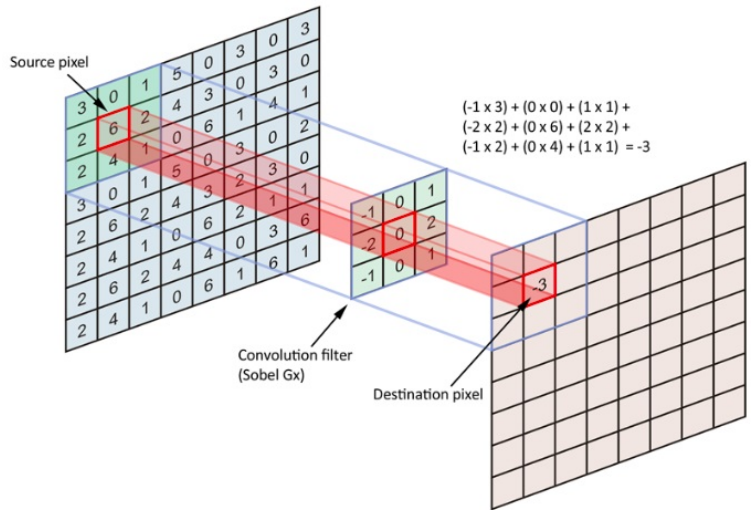
  which is generally nonconvex, nonsmooth
- Main approaches to find T: experience, heuristics, discrete optimization, experimental design?

Other popular deep learning architectures

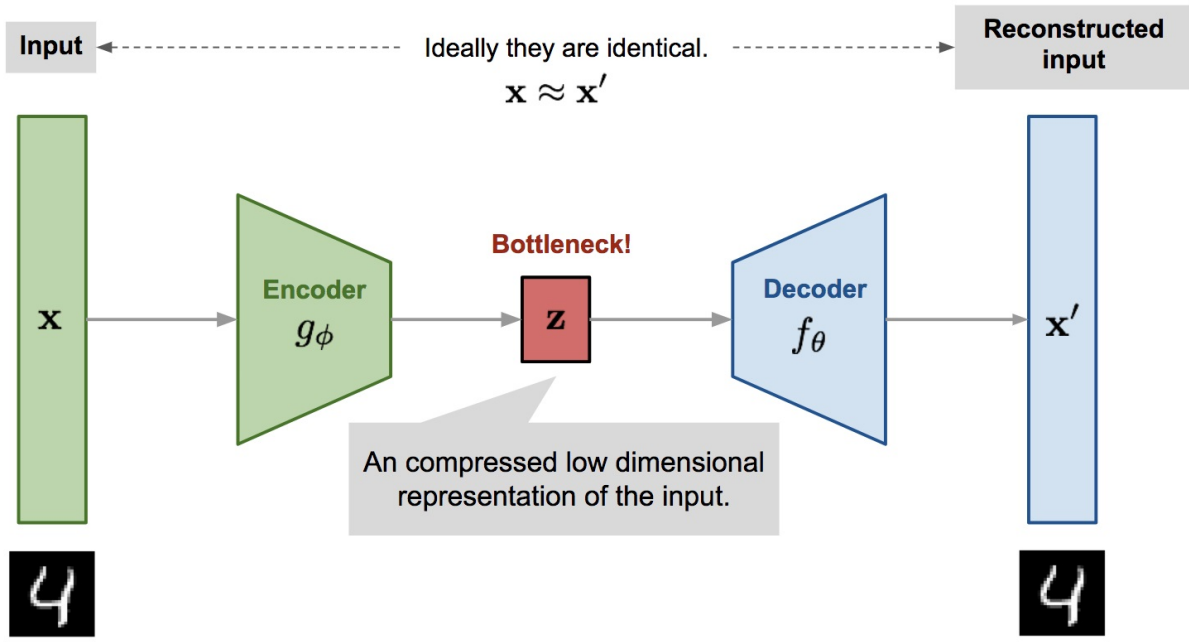
- Convolutional Neural Networks
- Recurrent Neural Networks
- Long Short Term Memory
- Auto-Encoders
- Boltzmann Machines, Belief Networks
- Generative Adversarial Networks

Other popular deep learning architectures
Convolutional Neural Networks





Other popular deep learning architectures
Auto-Encoders



From shallow to deep networks

2. The mysteries of Deep Learning

Mysteries about deep learning

- Approximation: deep better than shallow?
- Optimization: nonconvex with millions of dimensions (!)
- Overfitting: huge complexity

Facts about approximation theory

Comparison of Shallow vs. Deep Networks

- Poggio and Liao (2018): approximation of compositional functions
- Liang and Srikant (2017): approximation of polynomial functions
- Similar findings:

  " the number of neurons needed by a shallow network to approximate a function is exponentially larger than the corresponding number of neurons needed by a deep network for a given degree of function approximation. "

Facts about optimization in Deep Learning

- Under certain conditions, no poor local minima

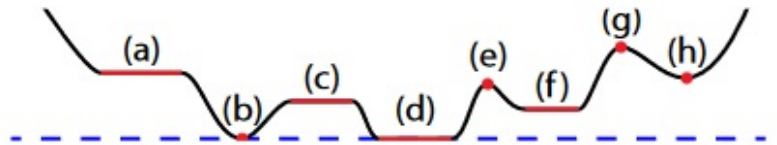


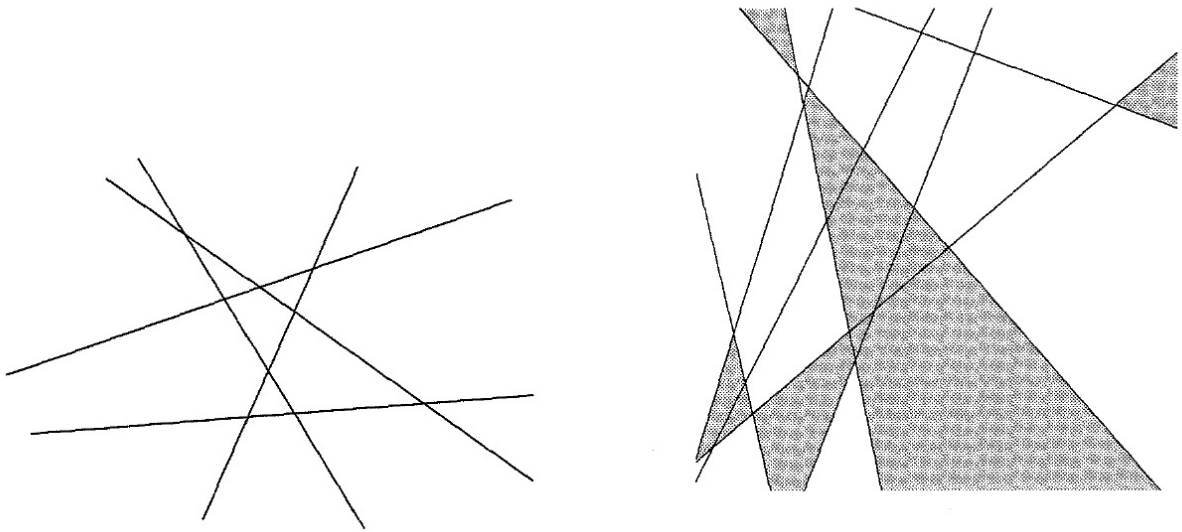
Figure 1. Example critical points of a non-convex function (shown in red). (a,c) Plateaus. (b,d) Global minima. (e,g) Local maxima. (f,h) Local minima.

- SGD avoids bad critical points
- Larger networks are better behaved (local minima are global)

References:

Soudry and Carmon (2016), "No bad local minima: Data independent training error guarantees for multilayer neural networks".
Kawaguchi (2016), "Deep learning without poor local minima".
Haeffele and Vidal (2017), "Global optimality in neural network training".
Janzamin, Sedghi, and Anandkumar (2015), "Beating the perils of non-convexity: Guaranteed training of neural networks using tensor methods".
Panageas and Piliouras (2016), "Gradient descent only converges to minimizers: Non-isolated critical points and invariant regions".
Brutzkus, Alon et al. (2017), "SGD Learns Over-parameterized Networks that Provably Generalize on Linearly Separable Data".

Vapnik's theory applied to Deep Learning
Complexity of arrangements



- VC dimension of multilayer feedforward neural network with ω parameters using step function for activation:

$$V \leq 2\omega \log_2(e\omega) \text{ can be quite huge...}$$

What's next?

- Other methods
- Decision trees, Bagging, Boosting, Random Forests