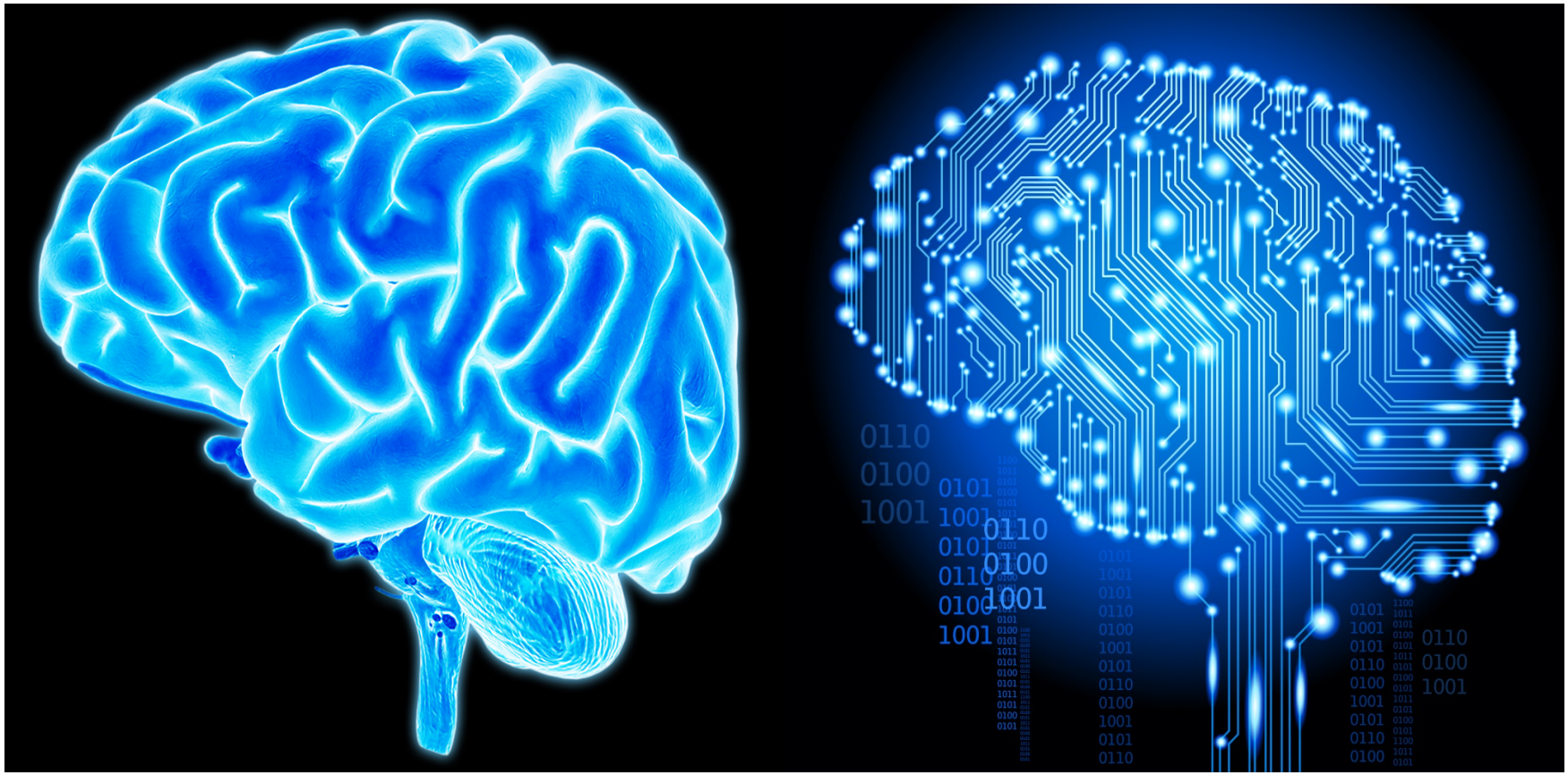
An Introduction to Deep Learning

Nan Ye

Mathematical Sciences School
Queensland University of Technology

A Historical Perspective

Brain vs. Computer



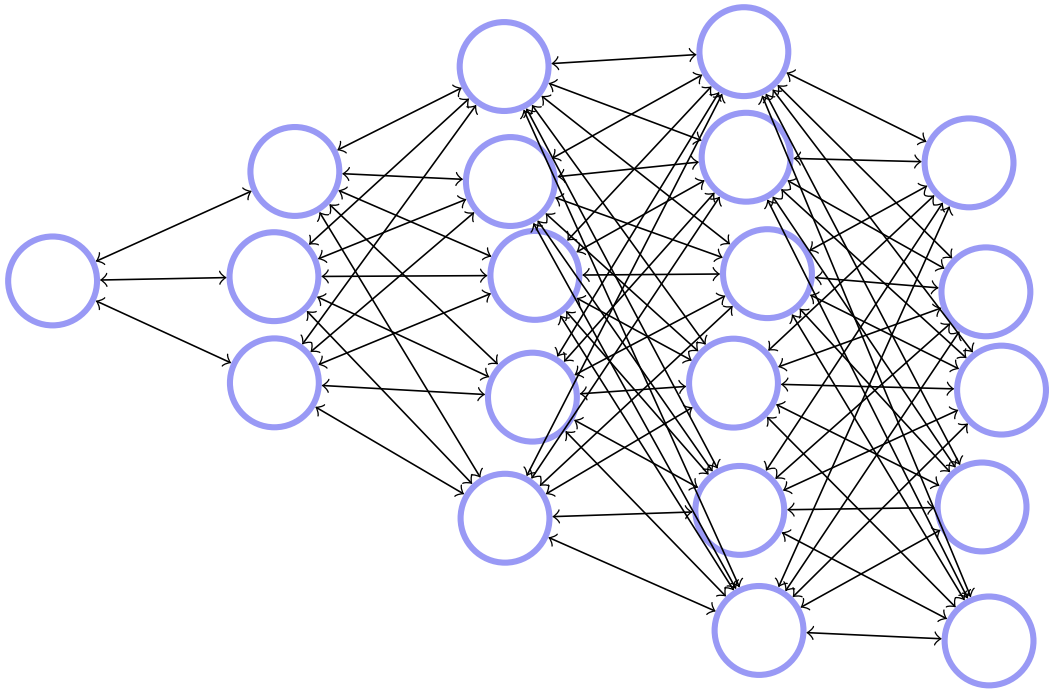
Strengths and weaknesses

- Computers are good at numerical and symbolic problems, but very vulnerable.
- Human brains are good at perceptual problems, and robust.

Architectural differences

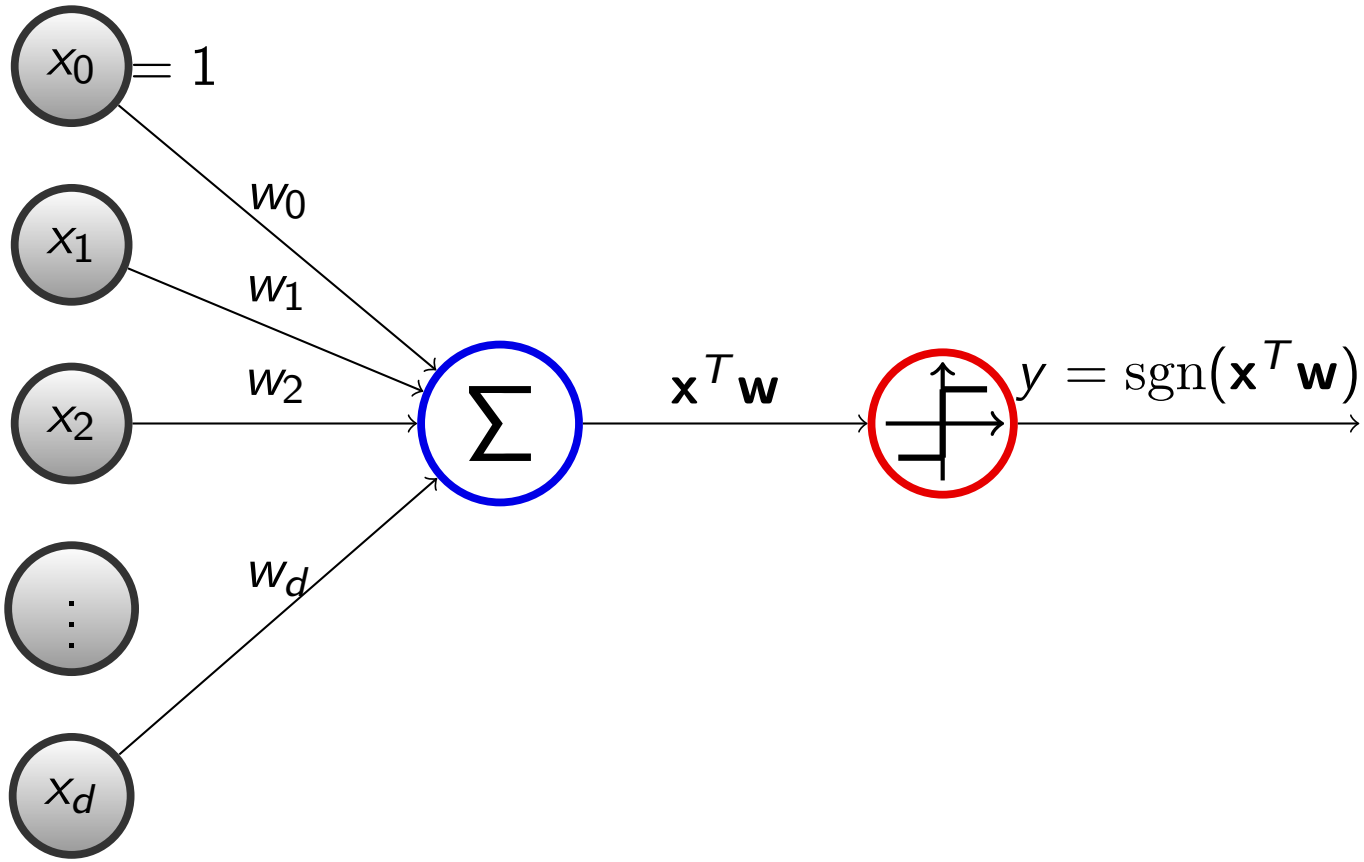
	von Neumann computer	**human brain**
processor	complex	simple
	high speed	low speed
	one or a few	many
memory	separate from processor	integrated into processor
	localized	distributed
	non-content addressable	content addressable
computation	centralized	distributed
	sequential	parallel
	stored programs	self-learning

A simplistic view of the brain



- An information processing system consisting of simple connected neurons.
- An information processing task is achieved by neurons receiving inputs from some neurons and sending their outputs to some neurons.

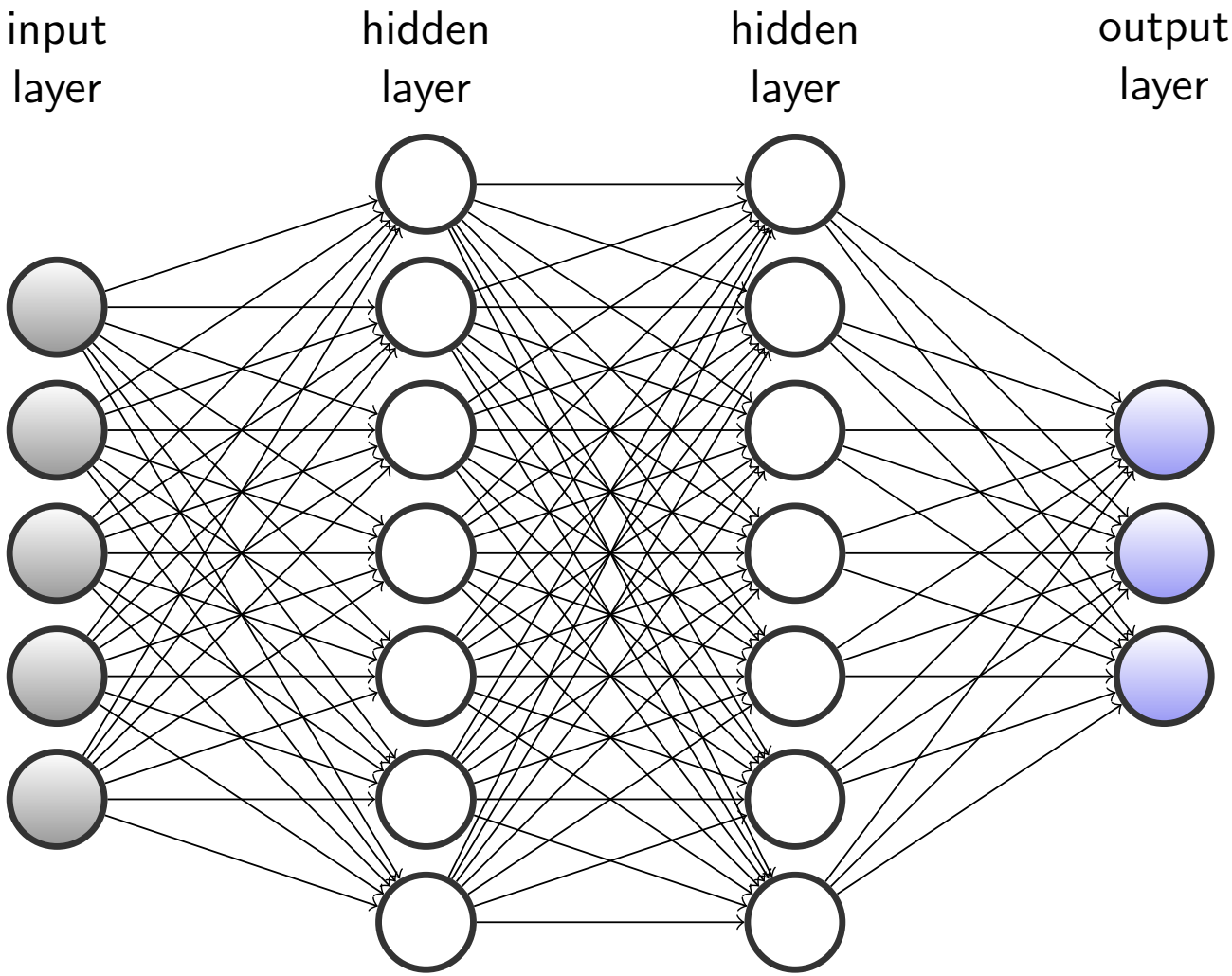
1960s: Rosenblatt's Perceptron



A single-neuron network with activation function $f(\mathbf{x}) = \text{sgn}(\mathbf{w}^T\mathbf{x})$

- The perceptron was the first learning system constructed.
- It finds a separating hyperplane if there is one.
- However, it is incapable of representing even simple functions like XOR.

1980s: Shallow Networks



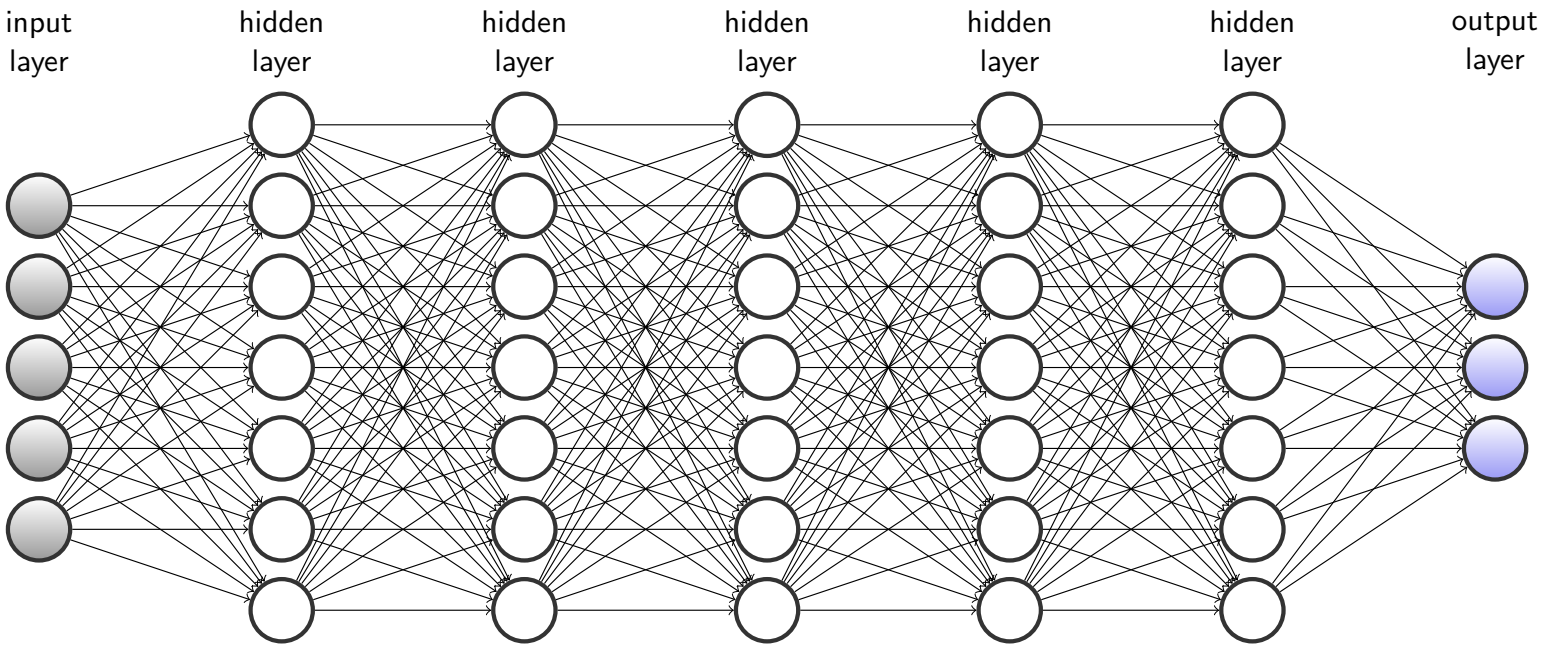
Depth generally not more than 3.

- The discovery of the backprogation algorithm by several authors in the 1970s and the 1980s made it possible to train shallow multi-layer networks.
- By the 1990's people had largely given up on the backpropagation algorithm because it did not seem to work for more complex networks.

Why does backpropagation fail?

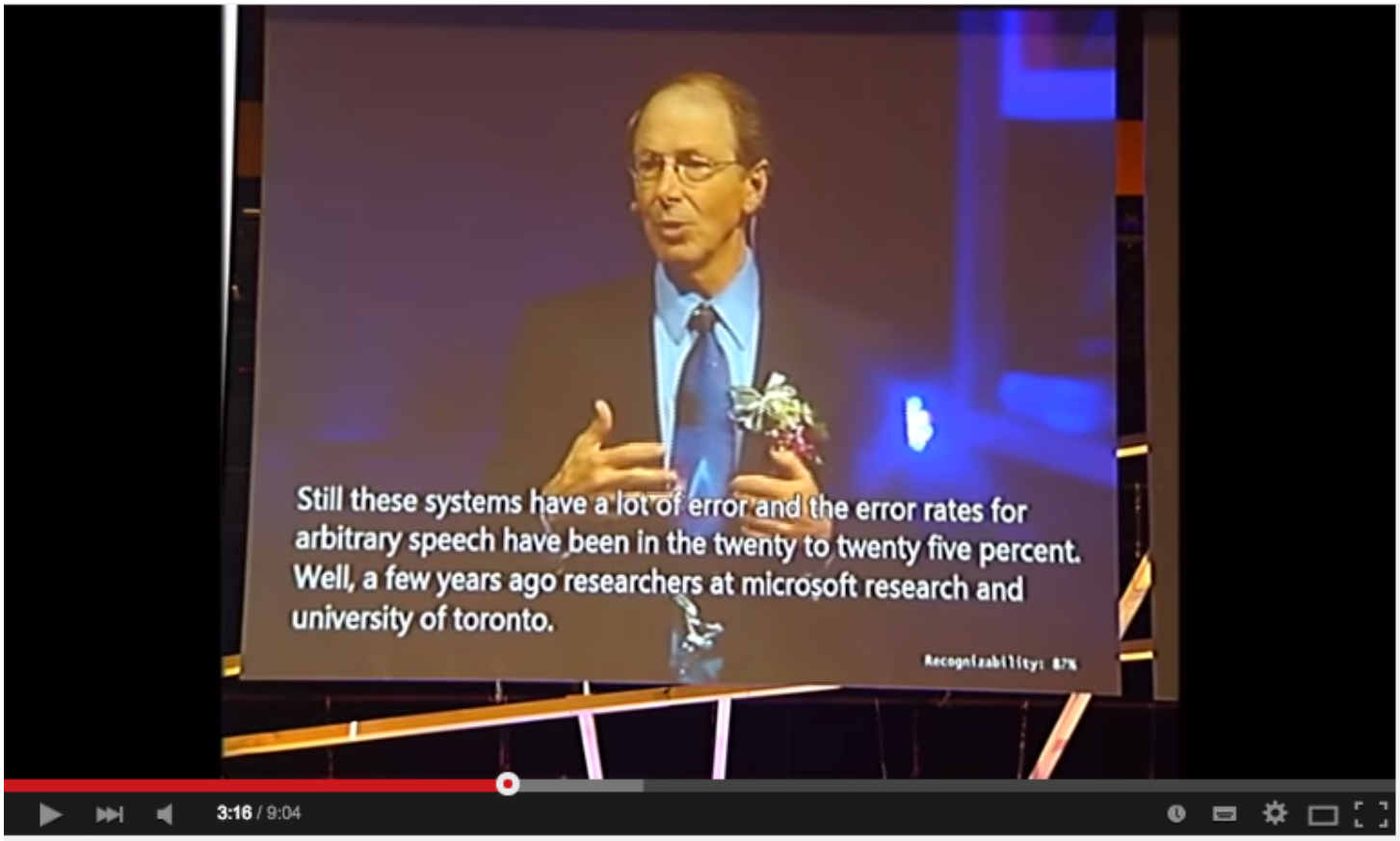
- It requires labeled training data.
- It requires a lot of computation power.
- It can get stuck in poor local optima.

2010s: Deep Architectures



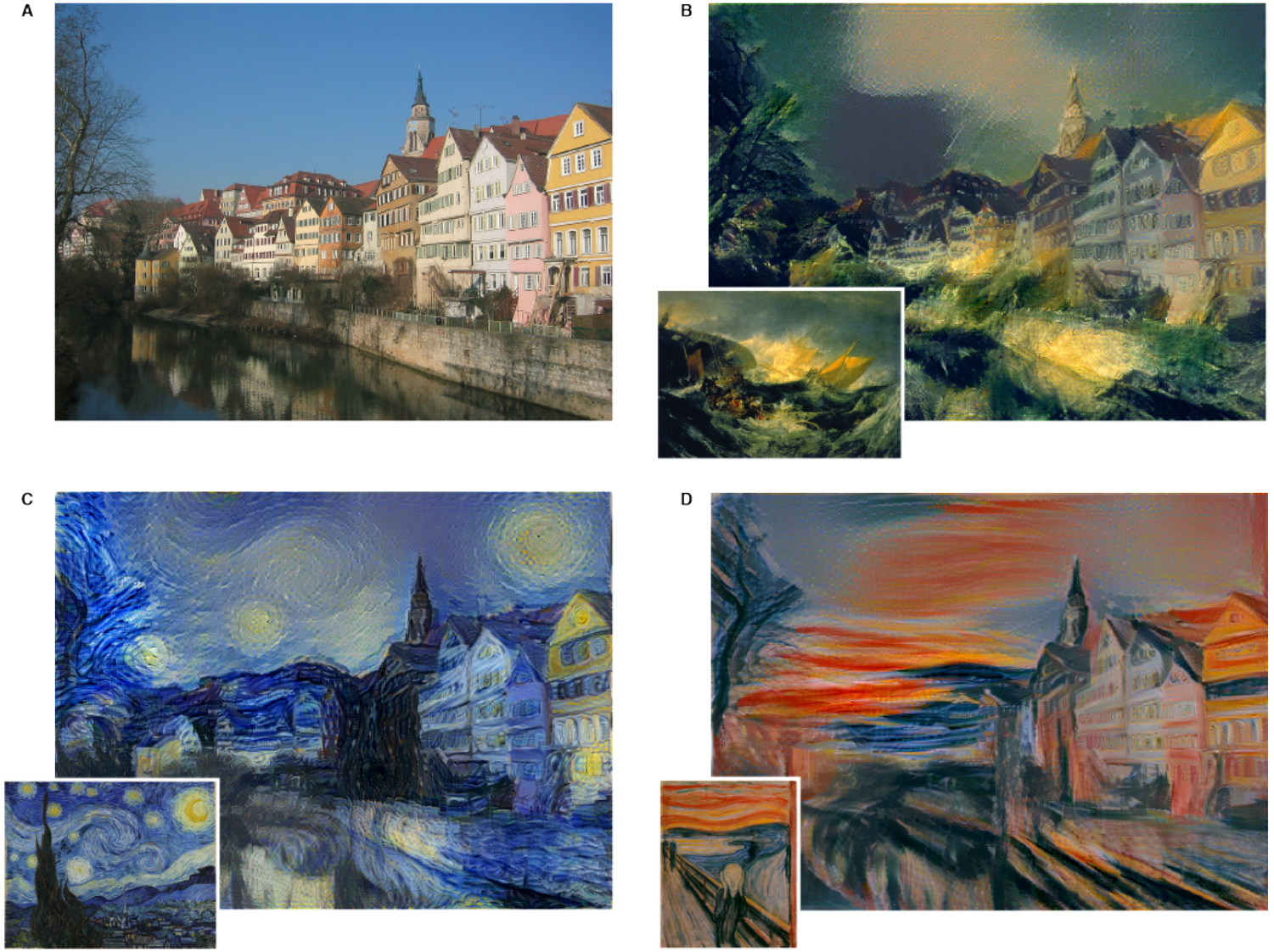
Behind the resurgence

- Many enormous datasets were prepared.
- Computers became much faster.
- Many technical improvements, including better activation functions, improved initialization techniques, and many trainable deep architectures.



First big success

Human-level Go AI



Neural artist

Gatys, Ecker, and Bethge, "A neural algorithm of artistic style", 2015

Tutorial Objective

Why deep architectures?

Understand three core ideas in machine learning

Statistics: what is a good model for a training set

Regularization: how to embed prior knowledge

Optimization: how to compute the desired model

Understand common shallow and deep neural networks

Perceptrons, multi-layer perceptrons, convolutional neural networks, deep belief networks...

Outline

Why deep architectures?

First principles: statistics, optimization, regularization

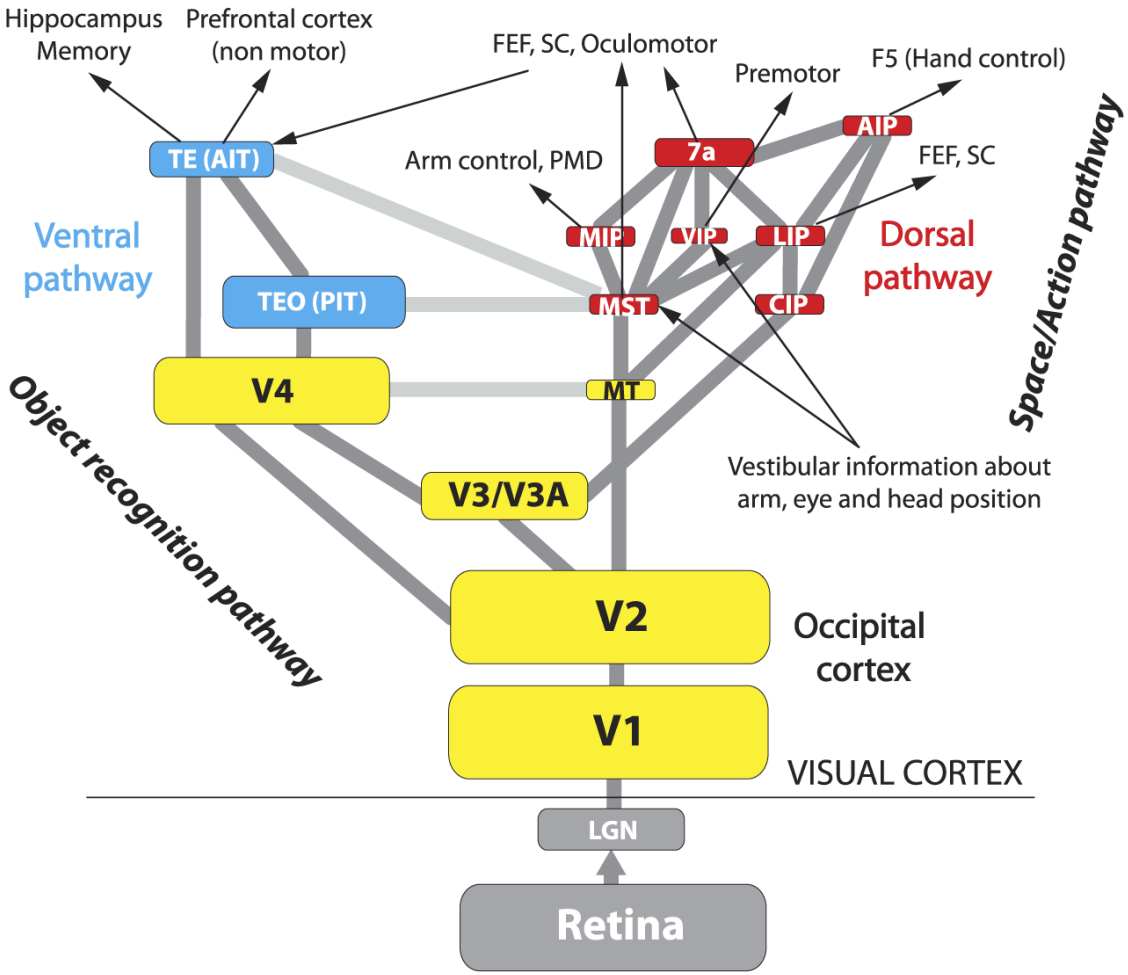
- Illustrations
- Statistical learning theory
- Regularization
- Optimization

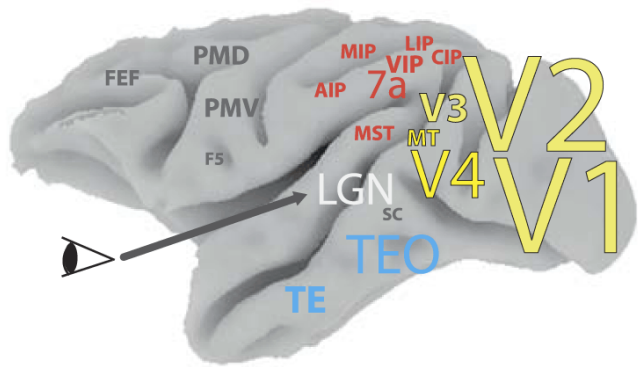
From shallow to deep networks

- Single-Neuron networks
- Multilayer perceptrons
- Convolutional neural networks
- Deep belief networks
- Other deep architectures

Why Deep Architectures?

Inspiration from Nature





The primate visual cortex is hierarchical

Kruger et al., "Deep hierarchies in the primate visual cortex: What can we learn for computer vision?", 2013

Deeper Can Be More Compact

Representational power of neural nets

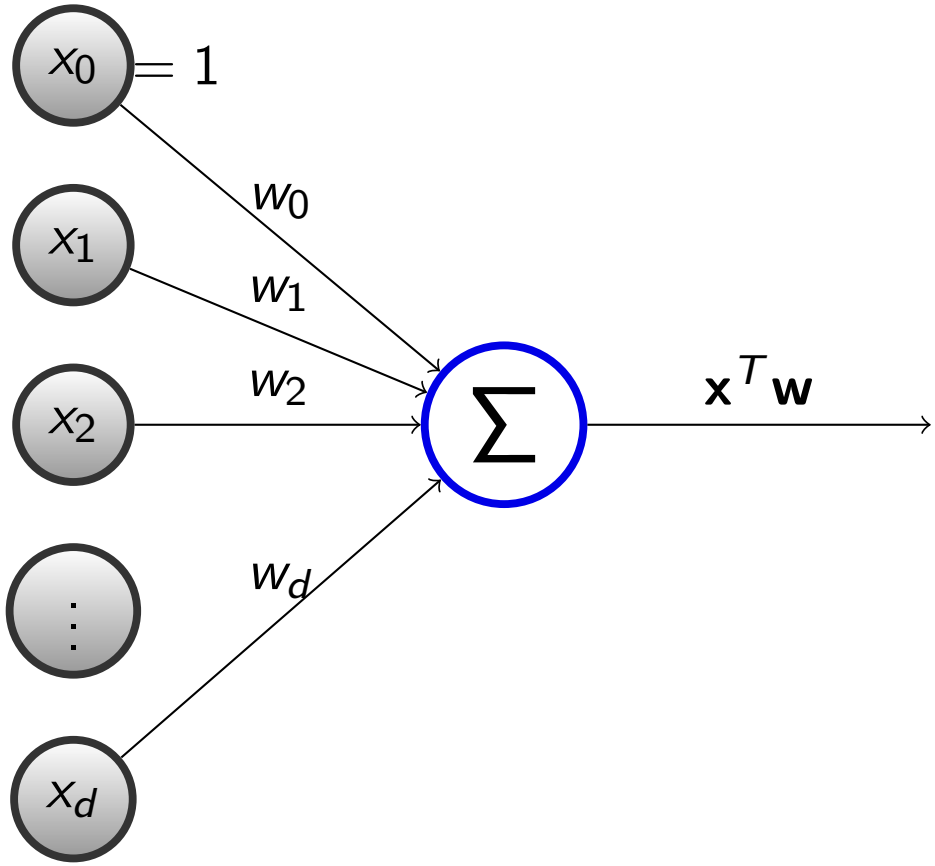
- Every boolean function can be represented by network with single hidden layer but might require exponential (in number of inputs) hidden units.
- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer.
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers.

Deeper can be more compact

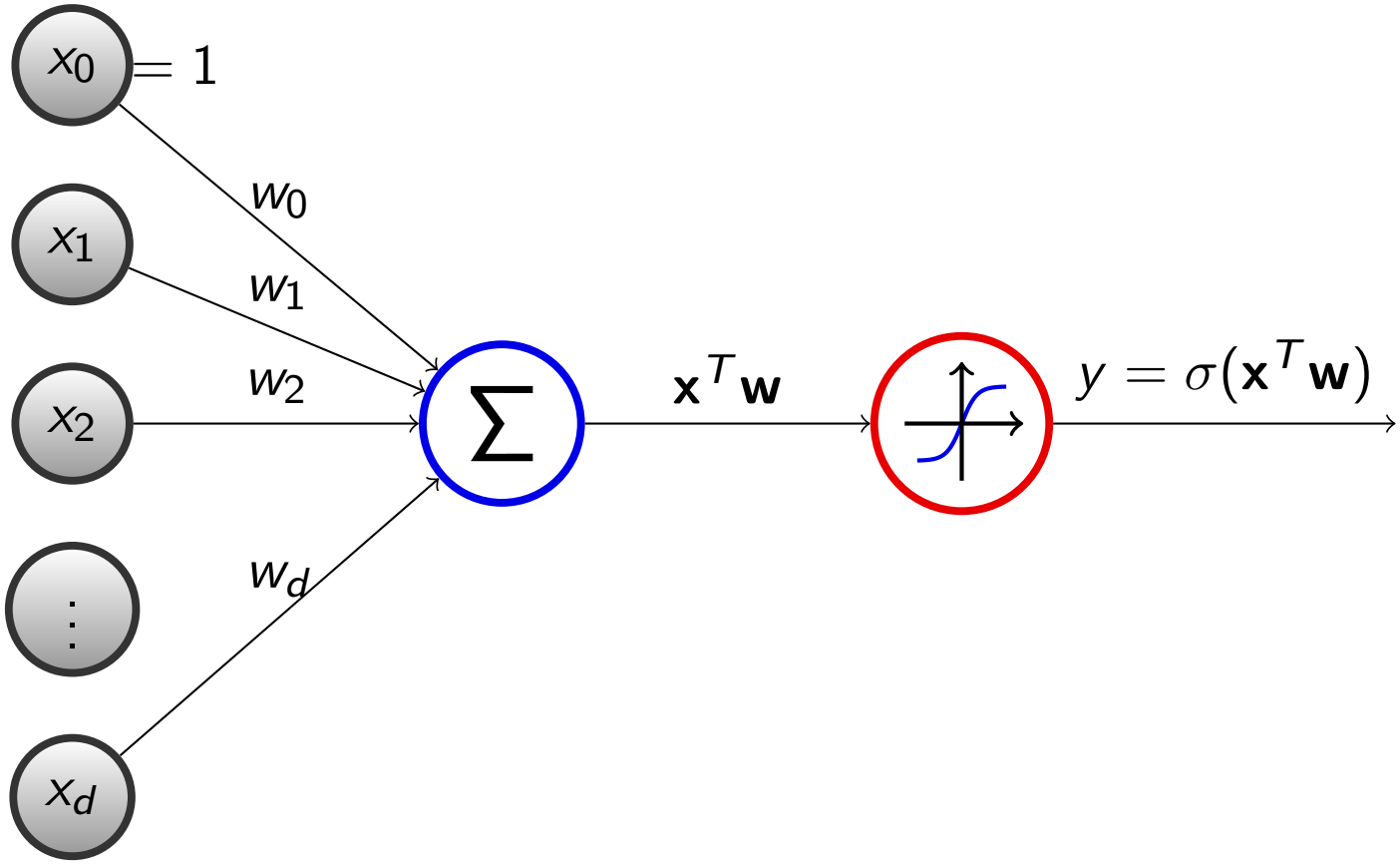
- When a function can be compactly represented by a deep network, it may need a very large shallow to represent it.
- *E.g. There are functions computable with a depth k network consisting of a polynomially many perceptron units that require exponentially many perceptron units when using a depth $k-1$ network.*

Features: Engineering to Learning

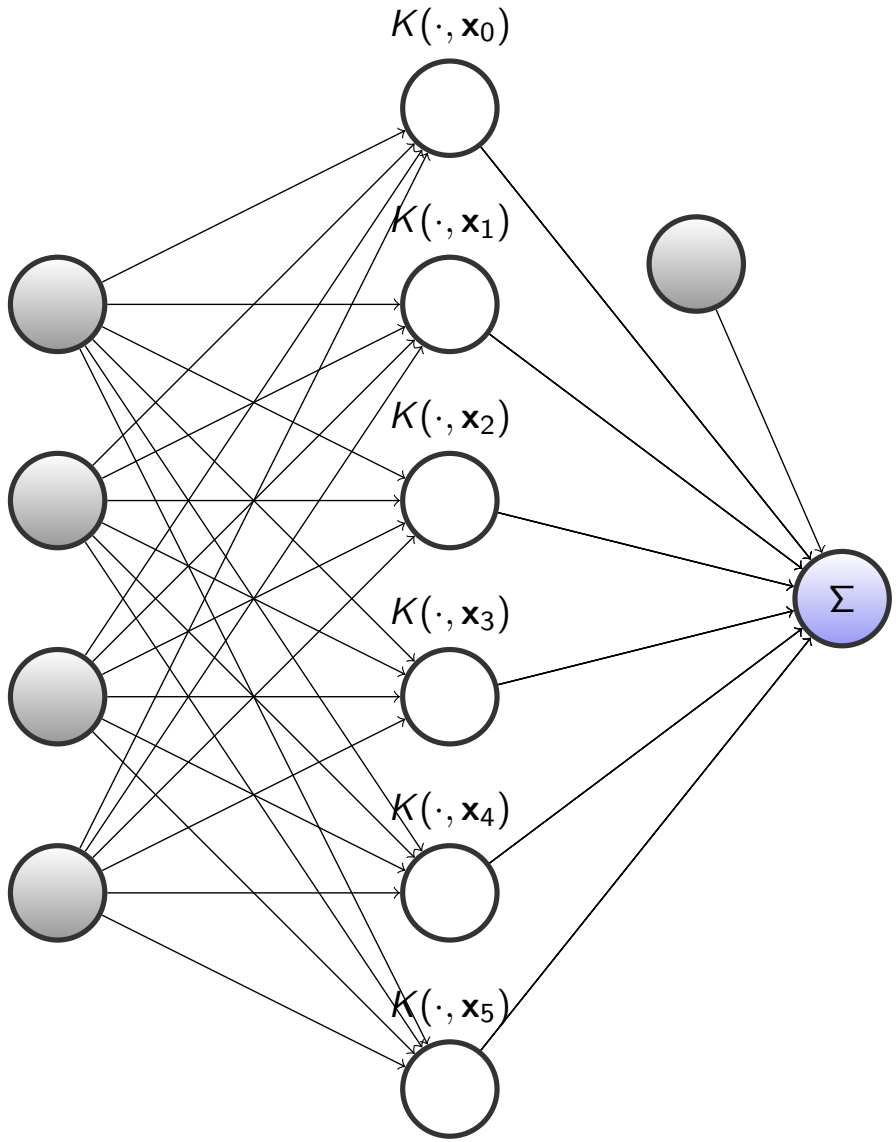
Traditional models as neural nets



Least squares regression $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$



Logistic regression $f(\mathbf{x}) = p(y = 1 \mid \mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$

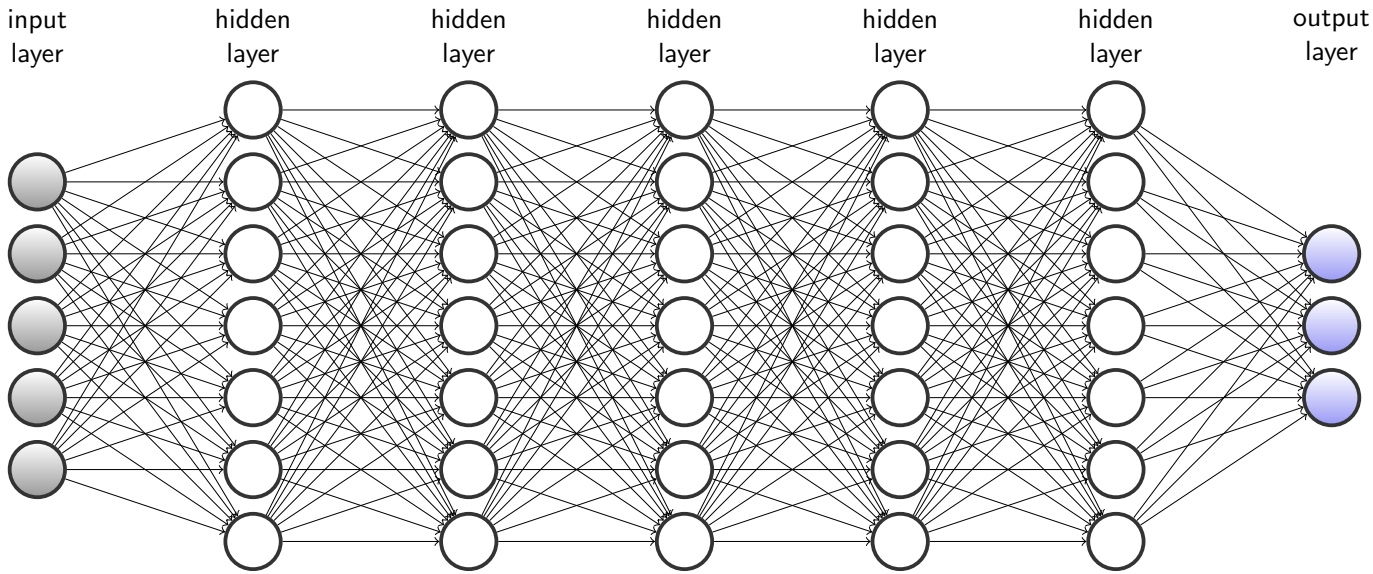


Support vector machines $f(\mathbf{x}) = b_0 + \sum_i \alpha_i K(\mathbf{x}, \mathbf{x}_i)$

Traditional learning: handcrafted features + classifier learning

- Many other traditional learning algorithms can be seen as neural networks.
- They build classifiers using *handcrafted* features.

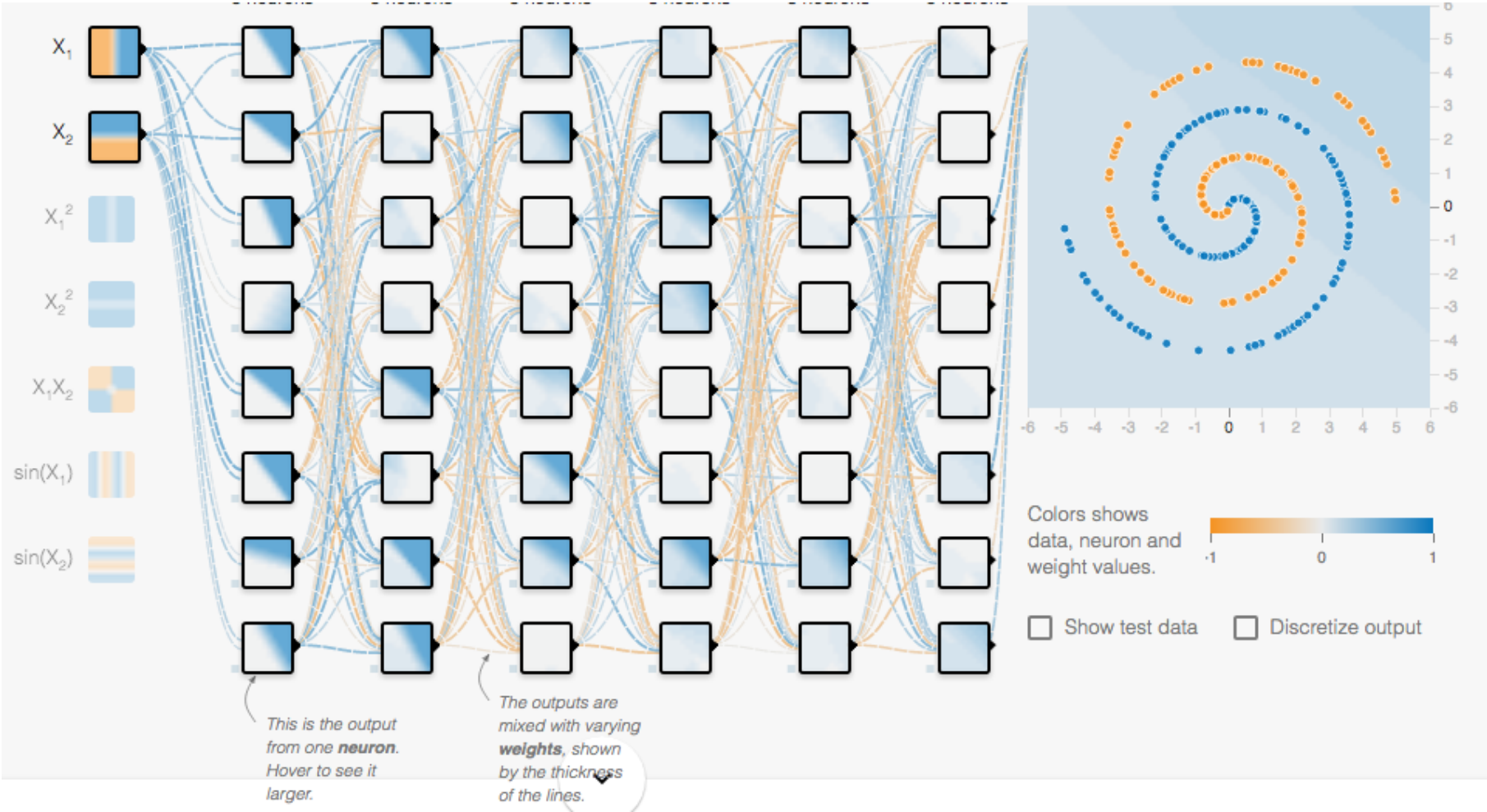
Deep learning: feature learning + classifer learning



- Deep learning uses deep architectures to additionally learn features.
- Deeper layers build abstract representations of previous layers.

  e.g. pixels $\rightarrow$ *edges* $\rightarrow$ *noses, eyes, ears* $\rightarrow$ *face*

A demo



# of hidden layers: 1, 2, 3, 4, 5, 6.

What You Need to Know...

Three motivations on using deep networks:

- Inspiration from nature: primate visual cortex has a deep hierarchy.
- Deeper can be more compact.
- Replacing feature engineering by feature learning.

Outline

Why deep architectures?

First principles: statistics, optimization, regularization

- Illustrations
- Statistical learning theory
- Regularization
- Optimization

From shallow to deep networks

- Single-Neuron networks
- Multilayer perceptrons
- Convolutional neural networks
- Deep belief networks
- Other deep architectures

Statistics, Optimization and Regularization

Illustrations

- Learning a binomial distribution
- Learning a Gaussian distribution

Learning a Binomial Distribution

I pick a coin with the probability of heads being θ. I flip it 100 times for you and you see a dataset D of 70 heads and 30 tails, can you learn θ?

Learning a Binomial Distribution

I pick a coin with the probability of heads being θ. I flip it 100 times for you and you see a dataset D of 70 heads and 30 tails, can you learn θ?

Maximum likelihood estimation
The likelihood of θ is

$$P(D \mid \theta) = \theta^{70}(1-\theta)^{30}.$$

Learning θ is an optimization problem.

$$\begin{aligned}\theta_{ml} &= \arg\max_{\theta} \; P(D \mid \theta) \\ &= \arg\max_{\theta} \; \ln P(D \mid \theta) \\ &= \arg\max_{\theta} \; (70 \ln \theta + 30 \ln(1-\theta)).\end{aligned}$$

$$\theta_{ml} = \arg\max_{\theta} \left(70 \ln \theta + 30 \ln(1-\theta)\right).$$

Set derivative of log-likelihood to 0,

$$\frac{70}{\theta} - \frac{30}{1-\theta} = 0,$$

we have

$$\theta_{ml} = 70/(70+30).$$

Regularization

If your friend told you that he has played the game with me and his estimate is 0.6, and you want to make use of this prior knowledge...

Regularization

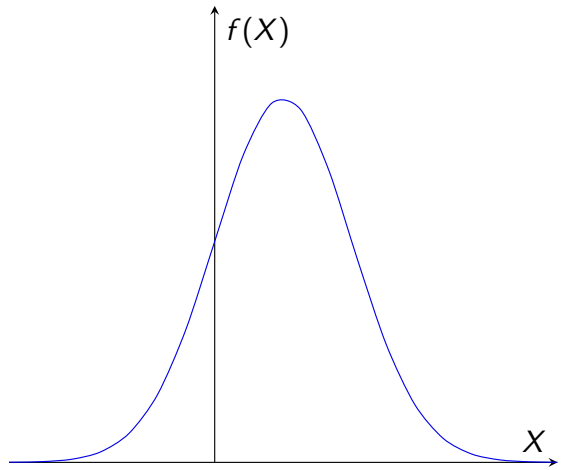
If your friend told you that he has played the game with me and his estimate is 0.6, and you want to make use of this prior knowledge...

$$\hat{\theta} = \arg\max_{\theta} \left(70 \ln \theta + 30 \ln(1-\theta) - \overbrace{\lambda(\theta - 0.6)^2}^{\text{regularizer}} \right)$$

- The *regularizer* favors estimates similar to your friend's.
- A larger $\lambda > 0$ means you have more trust for your friend's estimate.

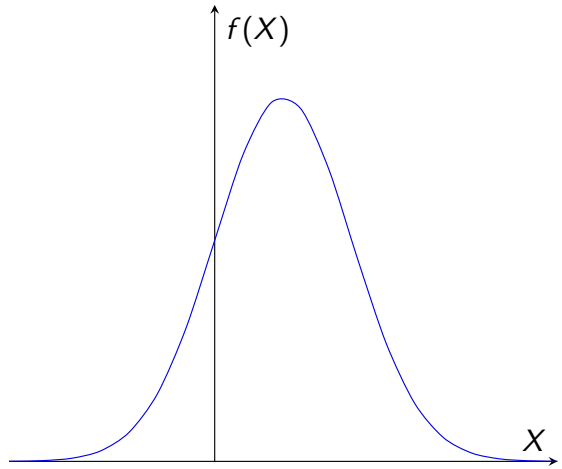
Learning a Gaussian distribution

I pick a Gaussian $N(\mu, \sigma^2)$ and give you a bunch of data $D = \{x_1, \ldots, x_n\}$ independently drawn from it. Can you learn μ and σ.



Learning a Gaussian distribution

I pick a Gaussian $N(\mu, \sigma^2)$ and give you a bunch of data $D = \{x_1, \ldots, x_n\}$ independently drawn from it. Can you learn μ and σ.



$$P(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

Maximum likelihood estimation

$$\ln P(D \mid \mu, \sigma) = \ln \left(\frac{1}{\sigma\sqrt{2\pi}} \right)^n \exp \left(- \sum_i \frac{(x_i - \mu)^2}{2\sigma^2} \right)$$
$$= -n \ln(\sigma\sqrt{2\pi}) - \sum_i \frac{(x_i - \mu)^2}{2\sigma^2}.$$

Set derivative w.r.t. μ to 0,

$$- \sum_i \frac{x_i - \mu}{\sigma^2} = 0 \quad \Rightarrow \quad \mu_{ml} = \frac{1}{n} \sum_i x_i$$

Set derivative w.r.t. σ to 0,

$$- \frac{n}{\sigma} + \frac{(x_i - \mu)^2}{\sigma^3} = 0 \quad \Rightarrow \quad \sigma^2_{ml} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu_{ml})^2.$$

Regularization

If your friend told you that he has played the game with me and he knows σ and his estimate of μ is c, and you want to make use of this prior knowledge...

Regularization

If your friend told you that he has played the game with me and he knows σ and his estimate of μ is c, and you want to make use of this prior knowledge...

$$\hat{\mu} = \arg\max_{\mu} \left(\ln P(D \mid \mu) - \overbrace{\frac{1}{2\sigma^2}(\mu - c)^2}^{\text{regularizer}} \right) = \frac{1}{n+1}(c + \sum_i x_i).$$

- The regularized estimate $\hat{\mu}$ has a smaller variance than μ_{ml}.
- However, $\hat{\mu}$ has a larger bias (expected difference between the estimate and the true mean) than μ_{ml}.

What You Need to Know...

Learning is...

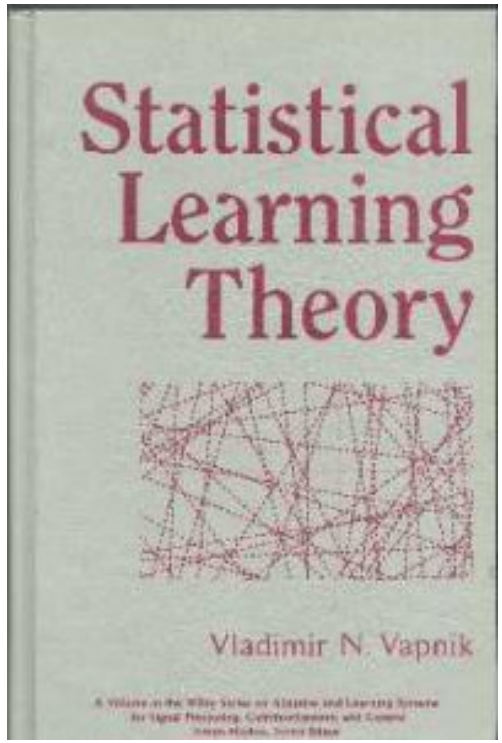
- Collect some data, e.g. coin flips.
- Choose a hypothesis class, e.g. binomial distribution.
- Choose a loss function, e.g. negative log-likelihood.
- Choose an optimization procedure, e.g. set derivative to 0.
- Regularization may be used to encode prior knowledge.

Statistics, optimization and regularization provide powerful tools for formulating and solving machine learning problems.

Statistical Learning Theory

- The framework
- Applications in classification, regression, and density estimation
- Does empirical risk minimization work?

There is nothing more practical than a good theory.
Kurt Lewin



...at least in the problems of statistical inference.
Vladimir Vapnik

Learning...

- H. Simon: Any process by which a system improves its performance.
- M. Minsky: Learning is making useful changes in our minds.
- R. Michalsky: Learning is constructing or modifying representations of what is being experienced.
- L. Valiant: Learning is the process of knowledge acquisition in the absence of explicit programming.

A Probabilistic Framework

Data

Training examples $z_1, \ldots, z_n$ are i.i.d. drawn from a *fixed* but *unknown* distribution $P(Z)$ on $\mathcal{Z}$.

e.g. outcomes of coin flips.

Hypothesis space $\mathcal{H}$

e.g. head probability $\theta \in [0, 1]$.

Loss function

$L(z, h)$ measures the penalty for hypothesis h on example z.

e.g. log-loss $L(z, \theta) = -\ln P(z \mid \theta) = \begin{cases} -\ln(\theta), & z = H, \\ -\ln(1-\theta), & z = T. \end{cases}$

Expected risk

- The expected risk of h is $\mathbb{E}(L(Z, h))$.
- We want to find the hypothesis with minimum expected risk,

$$\arg\min_{h \in \mathcal{H}} \mathbb{E}(L(Z, h)).$$

Empirical risk minimization (ERM)

Minimize empirical risk $R_n(h) \stackrel{\text{def}}{=} \frac{1}{n} \sum_i L(z_i, h)$ over $h \in \mathcal{H}$.

e.g. choose θ to minimize $-70 \ln \theta - 30 \ln(1 - \theta)$.

This provides a unified formulation for many machine learning problems, which differ in

- the data domain $\mathcal{Z}$,
- the choice of the hypothesis space $\mathcal{H}$, and
- the choice of loss function L.

Most algorithms that we see later can be seen as special cases of ERM.

Classification
predict a discrete class

Digit recognition: image to $\{0, 1, \ldots, 9\}$.

Spam filter: email to {spam, not spam}.

Given $D = \{(x_1, y_1), \ldots, (x_n, y_n)\} \subseteq \mathcal{X} \times \mathcal{Y}$, find a classifier f that maps an input $x \in \mathcal{X}$ to a *class* $y \in \mathcal{Y}$.

We usually use the 0/1 loss

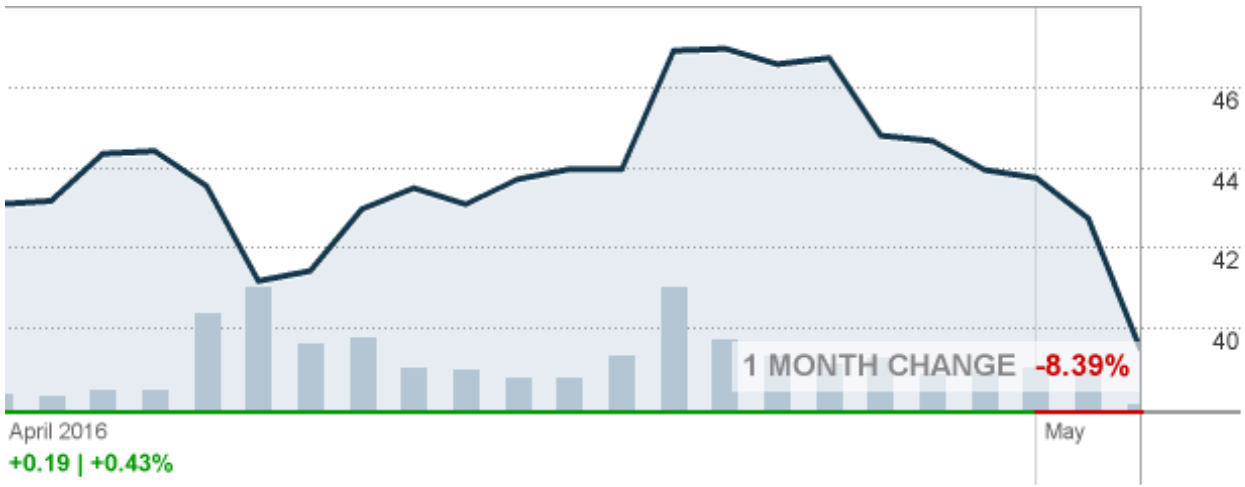
$$L((x, y), h) = \mathsf{I}(h(x) = y) = \begin{cases} 1, & h(x) \neq y, \\ 0, & h(x) = y. \end{cases}.$$

ERM chooses the classifier with minimum classification error

$$\min_{h \in \mathcal{H}} \frac{1}{n} \sum_i \mathsf{I}(h(x_i) = y_i).$$

Regression
predict a numerical value

Stock market prediction: predict stock price using recent trading data



Given $D = \{(x_1, y_1), \ldots, (x_n, y_n)\} \subseteq \mathcal{X} \times \mathbb{R}$, find a function f that maps an input $x \in \mathcal{X}$ to a *value* $y \in \mathbb{R}$.

We usually use the quadratic loss

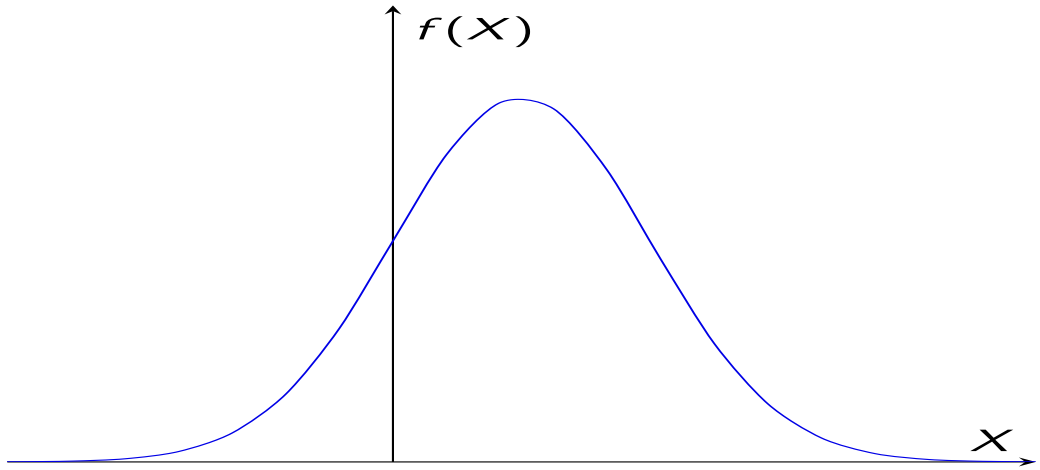
$$L((x, y), h) = (y - h(x))^2.$$

ERM is often called the method of least squares in this case

$$\min_{h \in \mathcal{H}} \frac{1}{n} \sum_i (y_i - h(x_i))^2.$$

Density Estimation

E.g. learning a binomial distribution, or a Gaussian distribution.



We often use the log-loss

$$L(x, h) = -\ln p(x \mid h).$$

ERM is MLE in this case.

Does ERM Work?

Estimation error

- How does the empirically best hypothesis $h_n = \arg\min_{h \in \mathcal{H}} R_n(h)$ compare with the best in the hypothesis space? Specifically, how large is the estimation error $R(h_n) - \inf_{h \in \mathcal{H}} R(h)$?
- **Consistency**: Does $R(h_n)$ converge to $\inf_{h \in \mathcal{H}} R(h)$ as $n \to \infty$?

Does ERM Work?

Estimation error

- How does the empirically best hypothesis $h_n = \arg\min_{h \in \mathcal{H}} R_n(h)$ compare with the best in the hypothesis space? Specifically, how large is the estimation error $R(h_n) - \inf_{h \in \mathcal{H}} R(h)$?
- **Consistency**: Does $R(h_n)$ converge to $\inf_{h \in \mathcal{H}} R(h)$ as $n \to \infty$?

If $|\mathcal{H}|$ is finite, ERM is likely to pick the function with minimal expected risk when n is *large*, because then $R_n(h)$ is close to $R(h)$ for all $h \in \mathcal{H}$.

If $|\mathcal{H}|$ is infinite, we can still show that ERM is likely to choose a near-optimal hypothesis if $\mathcal{H}$ has finite complexity (such as VC-dimension).

Approximation error

How good is the best hypothesis in $\mathcal{H}$? That is, how large is the approximation error $\inf_{h \in \mathcal{H}} R(h) - \inf_h R(h)$?

Approximation error

How good is the best hypothesis in $\mathcal{H}$? That is, how large is the approximation error $\inf_{h \in \mathcal{H}} R(h) - \inf_h R(h)$?

Trade-off between estimation error and approximation error:

- Larger hypothesis space implies smaller approximation error, but larger estimation error.
- Smaller hypothesis space implies larger approximation error, but smaller estimation error.

Optimization error

Is the optimization algorithm computing the empirically best hypothesis exact?

Optimization error

Is the optimization algorithm computing the empirically best hypothesis exact?

While ERM can be efficiently implemented in many cases, there are also computationally intractable cases, and efficient approximations are sought. The performance gap between the sub-optimal hypothesis and the empirically best hypothesis is the optimization error.

An Error Bound

If $|\mathcal{H}|$ is finite, and the loss function is bounded in $[0, 1]$, then with probability $1 - \delta$,

$$R(h_n) - \inf_{h \in \mathcal{H}} R(h) \leq 2\sqrt{\frac{\ln |\mathcal{H}| + \ln(2/\delta)}{2n}}$$

- The error decreases at a rate of $O(\frac{1}{\sqrt{n}})$.
- As $n \to \infty$, the RHS tends to 0, thus ERM is consistent in this case.

The case with $|\mathcal{H}| = \infty$

- Error bound depends on more sophisticated complexity measure (such as VC-dimension, see Vapnik's book).
- The number of parameters does not measure the complexity of the model family.

What You Need to Know...

Recognise machine learning problems as special cases of the general statistical learning problem.

Understand that the performance of ERM depends on the approximation error, estimation error and optimization error.

Outline

Why deep architectures?

First principles: statistics, optimization, regularization

- Illustrations
- Statistical learning theory
- Regularization
- Optimization

From shallow to deep networks

- Single-Neuron networks
- Multilayer perceptrons
- Convolutional neural networks
- Deep belief networks
- Other deep architectures

Regularization

- Regularized ERM
- Regularization and stability
- Regularization and overfitting
- Regularization as Occam's razor
- Regularization as Bayesian prior
- Regularization as constraint

Regularized ERM

We choose a hypothesis such that

$$h_n = \arg\min_{h \in \mathcal{H}} (nR_n(h) + r(h)),$$

where $r(h) \geq 0$ is a penalty of choosing h, and is generally called the regularizer.

Some common regularizers (model parametrized by w)

- ℓ_0 regularizer: $r(\mathbf{w}) = ||\mathbf{w}||_0$, induces sparsity but hard to optimize.
- ℓ_1 regularizer: $r(\mathbf{w}) = ||\mathbf{w}||_1$, induces sparsity and easier to optimize.
- ℓ_2 regularizer: $r(\mathbf{w}) = ||\mathbf{w}||_2^2$, shrinks coefficients towards 0.

Recall

Regularized MLE for learning Bernoulli distribution

$$\hat{\theta} = \arg\max_{\theta} \left(70 \ln \theta + 30 \ln(1-\theta) - \lambda(\theta - 0.6)^2\right)$$

Regularized MLE for learning Gaussian mean

$$\hat{\mu} = \arg\max_{\mu} \left(\ln P(D \mid \mu) - \frac{1}{2\sigma^2}(\mu - c)^2\right)$$

Regularization and Stability

Stability

- A problem is stable if its solution does not change much when it is perturbed.

Regularization and Stability

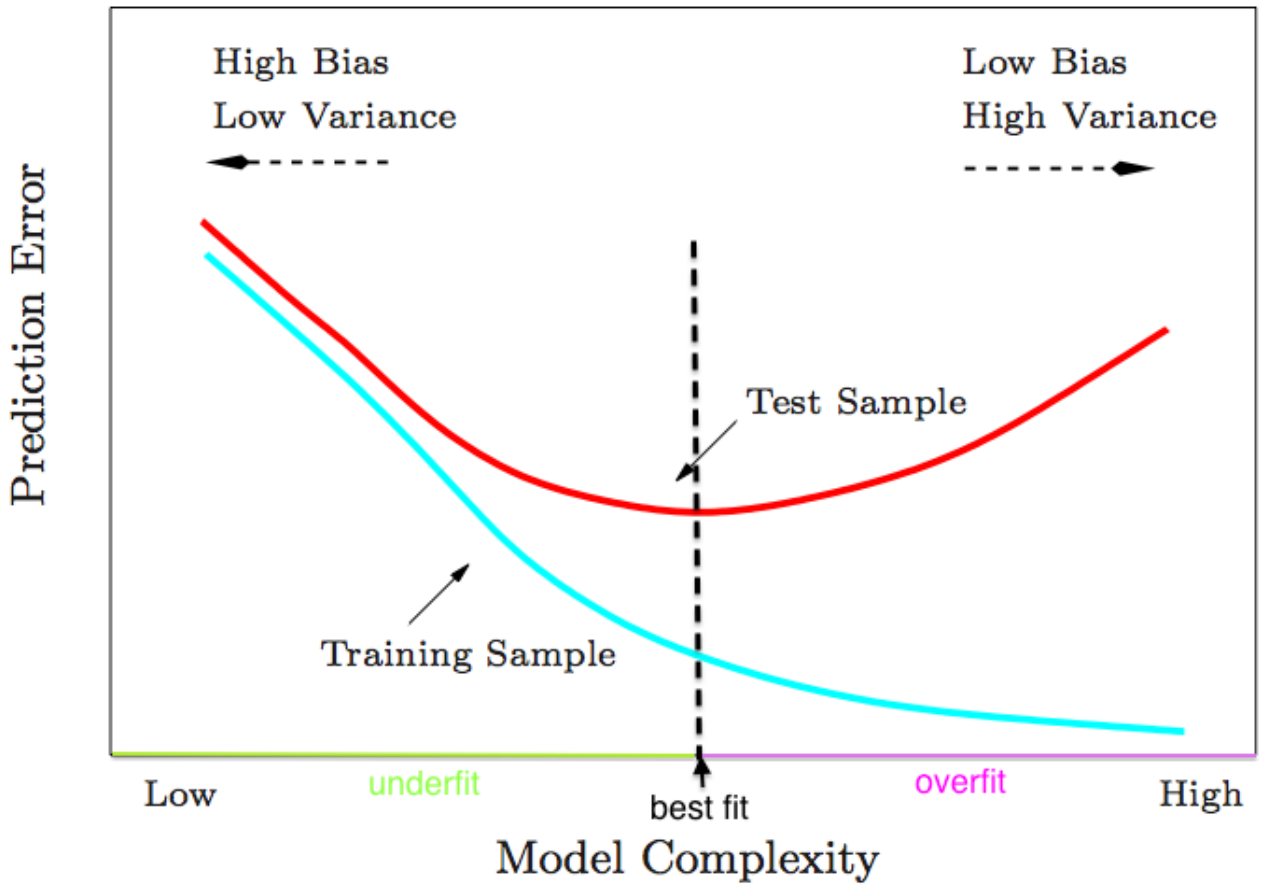
Stability

- A problem is stable if its solution does not change much when it is perturbed.

Regularization increases stability

- Regularization makes learning more stable as the regularized estimates change less when data is changed.
- Regularized estimate has smaller variance but larger bias.
- Examples
  - Recall: Unregularized and regularized MLE for Gaussian mean.
  - Ridge regression: $\arg\min_{\mathbf{w}} \left(\sum_i ||\mathbf{x}_i^T \mathbf{w} - y_i||_2^2 + \lambda ||\mathbf{w}||^2\right)$.

Regularization and Overfitting



Regularization decreases model complexity and make overfitting less likely.

Regularization as Occam's Razor

Occam's razor

- Prefer simpler explanations.

Regularization as Occam's Razor

Occam's razor

- Prefer simpler explanations.

Regularization favors simpler hypothesis

- The regularizer $r(h)$ can be thought of as a measure of the complexity of a hypothesis.
- Simpler hypothesis is thus preferred over more complex ones.

Regularization as Bayesian Prior

Bayesian MAP (maximum a posterior) hypothesis

- Assume a prior distribution $P(h)$, a likelihood function $P(D \mid h)$, then the posterior probability $P(h \mid D)$ is given by

$$P(h \mid D) = P(h)P(D \mid h)/P(D),$$

- $h_{MAP} = \arg\max_{h \in \mathcal{H}} P(h \mid D)$ is called the MAP hypothesis.

Regularization as Bayesian prior

Suppose the prior $P(h)$ and the likelihood function $P(D \mid h)$ are given by

$$P(h) \propto e^{-r(h)},$$
$$P(D \mid h) \propto e^{-nR_n(h)},$$

Then we have

$$h_{MAP} = \arg\max_{h \in \mathcal{H}} P(h \mid D) = \arg\min_{h \in \mathcal{H}} \left(nR_n(h) + r(h) \right).$$

Thus the regularizer can be interpreted as a prior distribution $P(h)$.

Regularization as Constraint

There exists a constant t such that

$$\arg\min_{h \in \mathcal{H}} \left(n R_n(h) + r(h) \right) = \arg \min_{h \in \mathcal{H}, r(h) \leq t} R_n(h).$$

Thus regularization restricts model fitting to hypothesis with complexity below a threhold.

What You Need to Know...

- Regularized ERM
- Regularization increases stability.
- Regularization alleviates overfitting.
- Regularization can be seen as Occam's razor.
- Regularization can be seen as a Bayesian prior.
- Regularization can be seen as a constraint on hypothesis space.

A Primer On Numerical Optimization

- Optimization problems
- Gradient descent
- Stochastic gradient descent
- Other methods

Optimization Problem

$$\begin{aligned} \min \quad & f(\mathbf{w}) \\ \text{s.t.} \quad & h_i(\mathbf{w}) \leq b_i, i = 1, \ldots, m, \end{aligned}$$

- $\mathbf{w} = (w_1, \ldots, w_d) \in \mathbb{R}^d$: optimization variable.
- $f : \mathbb{R}^d \to \mathbb{R}$: the objective function.
- $h_i : \mathbb{R}^d \to \mathbb{R}$: (inequality) constraint functions.
- Feasible set: the set of points satisfying all constraints.

Recall: Learning as Optimization

ERM (a.k.a M-estimators)

Given examples $z_1, \ldots, z_n \in \mathcal{Z}$, solve

$$\mathbf{w}_n = \arg\min_{\mathbf{w} \in \mathbb{R}^d} \sum_{i=1}^{n} L(z_i, \mathbf{w}),$$

Regularization and constraint

- A regularizer is often added to the objective to avoid over-fitting

$$\mathbf{w}_n = \arg\min_{\mathbf{w} \in \mathbb{R}^d} \left(\sum_i L(z_i, \mathbf{w}) + \lambda r(\mathbf{w}) \right).$$

- This is equivalent to a constrained optimization problem $\arg\min_{r(\mathbf{w}) \leq r} \sum_i L(z_i, \mathbf{w})$.

Optimization is Hard in General

We can prove that in general we cannot find the optimizer of a function $f(\mathbf{w})$ efficiently.

Optimization is Hard in General

We can prove that in general we cannot find the optimizer of a function $f(\mathbf{w})$ efficiently.

There are interesting classes of functions that can be efficiently optimized.

Optimization is Hard in General

We can prove that in general we cannot find the optimizer of a function $f(\mathbf{w})$ efficiently.

There are interesting classes of functions that can be efficiently optimized.

Deep learning involves optimizing functions which are considered hard to optimize, and recent successes motivate researches into new ways to look at these optimization problems.

Efficiently optimizable functions

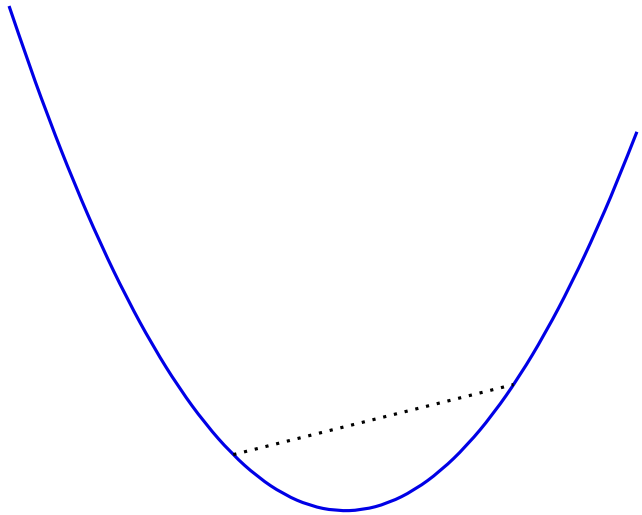
- Linear optimization: linear objective, linear constraints.
- Convex optimization: convex objective, convex feasible set.
- Some non-convex functions.

$$\text{linear} \subset \text{convex} \subset \text{general}$$

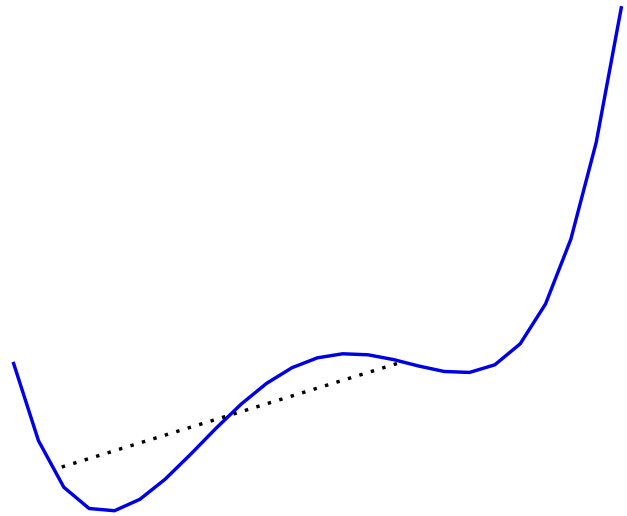
A function f is convex if

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$$

for all $\lambda \in [0,1]$ and x, y.



Convex

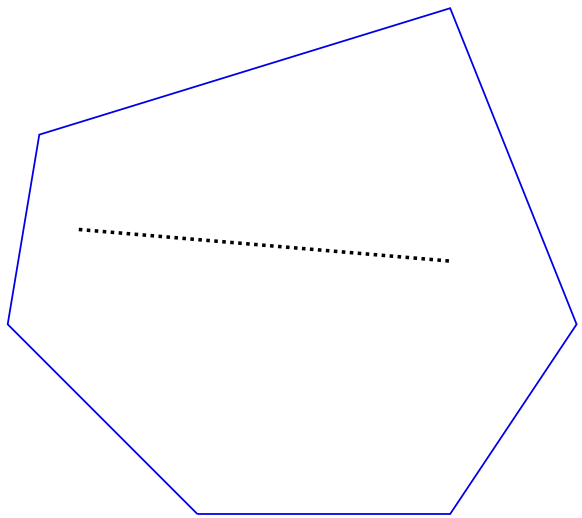


Nonconvex

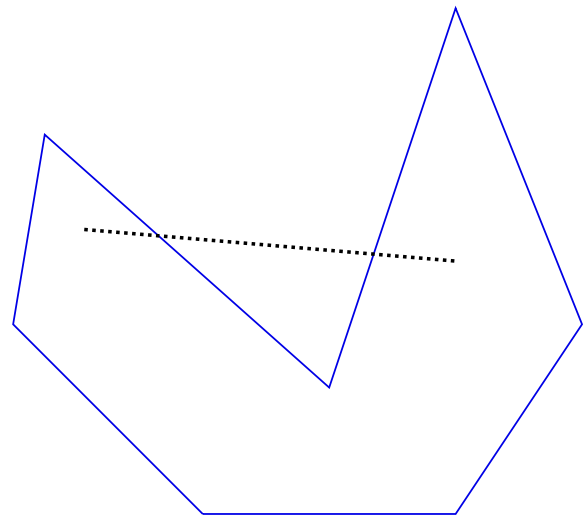
A set S is convex if

$$\lambda x + (1 - \lambda) y \in S$$

for any $\lambda \in [0, 1]$, any $x, y \in S$.



Convex



Nonconvex

Examples of convex learning problems

- Ridge regression

$$\min_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{n} (x_i^T \mathbf{w} - y_i)^2 + \lambda ||\mathbf{w}||_2^2.$$

- Logistic regression

$$\min_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{n} \ln(1 + e^{-y_i x_i^T \mathbf{w}}) + \lambda ||\mathbf{w}||_2^2.$$

- Support vector machines

$$\min_{\mathbf{w}\in\mathbb{R}^d, b\in\mathbb{R}} \frac{1}{2}||\mathbf{w}||_2^2 + \sum_{i=1}^{n} \max(0, 1 - y_i(x_i^T \mathbf{w} + b)).$$

Gradient Descent

General scheme

$\mathbf{w}_1 \in \mathbb{R}^d$

$\mathbf{w}_{s+1} = \mathbf{w}_s - \eta_s \nabla f(\mathbf{w}_s)$, where $\eta_s > 0$ is a step size.

Intuition

$f(\mathbf{w}_s - \eta_s d) \approx f(\mathbf{w}_s) - \eta_s \nabla f(\mathbf{w}_s)^T d$

$d = \nabla f(\mathbf{w}_s)$ yields fastest descent in the linear approximation.

Example

$$\min_{x \in \mathbb{R}} f(x, y) = x^2 + y^2$$

Constant step size $\eta_s = 0.25$ for all s.

- $(x_1, y_1) = (2, 3)$.
- $(x_2, y_2) = (x_1, y_1) - 0.25 \cdot 2(x_1, y_1) = 0.5(x_1, y_1)$.
- $(x_3, y_3) = (x_2, y_2) - 0.25 \cdot 2(x_2, y_2) = 0.5(x_2, y_2) = 0.5^2(x_1, y_1)$.
- $\ldots$
- $(x_s, y_s) = 0.5^{s-1}(x_1, y_1) \to (0, 0)$ as $s \to \infty$.

Example

$$\min_{x \in \mathbb{R}} f(x, y) = x^2 + y^2$$

Constant step size $\eta_s = 1$ for all s.

- $(x_1, y_1) = (2, 3)$.
- $(x_2, y_2) = (x_1, y_1) - 1 \cdot 2(x_1, y_1) = -(x_1, y_1)$.
- $(x_3, y_3) = (x_2, y_2) - 1 \cdot 2(x_2, y_2) = -(x_2, y_2) = (x_1, y_1)$.
- $\ldots$
- $(x_s, y_s) = (-1)^{s-1}(x_1, y_1)$ does not converge to the minimizer.

Example

$$\min_{x \in \mathbb{R}} f(x, y) = x^2 + y^2$$

Constant step size $\eta_s = 1$ for all s.

- $(x_1, y_1) = (2, 3)$.
- $(x_2, y_2) = (x_1, y_1) - 1 \cdot 2(x_1, y_1) = -(x_1, y_1)$.
- $(x_3, y_3) = (x_2, y_2) - 1 \cdot 2(x_2, y_2) = -(x_2, y_2) = (x_1, y_1)$.
- $\ldots$
- $(x_s, y_s) = (-1)^{s-1}(x_1, y_1)$ does not converge to the minimizer.

Step size is important.

Step size rules

- Chosen in advance, such as setting η_k to a constant or $\eta_k = \frac{1}{\sqrt{k+1}}$.

Step size rules

- Chosen in advance, such as setting η_k to a constant or $\eta_k = \frac{1}{\sqrt{k+1}}$.
- Chosen to optimize $f(\mathbf{w}_s - \eta_s \nabla f(\mathbf{w}_s))$.

Step size rules

- Chosen in advance, such as setting η_k to a constant or $\eta_k = \frac{1}{\sqrt{k+1}}$.
- Chosen to optimize $f(\mathbf{w}_s - \eta_s \nabla f(\mathbf{w}_s))$.
- Chosen according to Goldstein-Armijo rule: Find $x_{k+1} = x_k - \eta \nabla f(x_k)$ such that

$$\alpha \nabla f(x_k)^\top (x_k - x_{k+1}) \leq f(x_k) - f(x_{k+1}),$$
$$\beta \nabla f(x_k)^\top (x_k - x_{k+1}) \geq f(x_k) - f(x_{k+1}),$$

where $0 < \alpha < \beta < 1$.

Projected Gradient Descent

Consider the following constrained optimization problem

$$\min_{\mathbf{w} \in W} f(\mathbf{w})$$

General scheme

$\mathbf{w}_1 \in W$

$\mathbf{w}_{s+1} = \Pi_W(\mathbf{w}_s - \eta_s \nabla f(\mathbf{w}_s))$, where $\eta_s > 0$ is a step size, and

$$\Pi_W(\mathbf{v}) = \arg\min_{\mathbf{w} \in W} ||\mathbf{w} - \mathbf{v}||_2.$$

Remarks

- First move along the negative gradient direction, and then projects the solution back to the feasible set.
- Unconstrained gradient descent corresponds to $W = \mathbb{R}^d$.

Convergence Rates

Assume convex W, $\mathbf{w}^* = \arg\min_{\mathbf{w} \in W} f(\mathbf{w})$.

Convex functions

If $0 \preceq \nabla^2 f(\mathbf{w})$, f is L-Lipschitz, and $\eta_s = \frac{1}{\sqrt{t}}$ for $1 \leq s \leq t$, then

$$f\left(\frac{1}{t}\sum_{s=1}^{t}\mathbf{w}_s\right) - f(\mathbf{w}^*) \leq \frac{||\mathbf{w}_1 - \mathbf{w}^*||_2^2 + L^2}{\sqrt{t}}.$$

More on function properties

- Convexity: $0 \preceq \nabla^2 f(x)$.

  A local minimum is a global minimum.

More on function properties

- Convexity: $0 \preceq \nabla^2 f(x)$.

  A local minimum is a global minimum.

- Strong convexity: $\alpha I \preceq \nabla^2 f(x)$ for some $\alpha > 0$.

  Quadratic lower bound $f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{\alpha}{2} ||y - x||_2^2$.

More on function properties

- Convexity: $0 \preceq \nabla^2 f(x)$.

  A local minimum is a global minimum.

- Strong convexity: $\alpha I \preceq \nabla^2 f(x)$ for some $\alpha > 0$.

  Quadratic lower bound $f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{\alpha}{2} ||y - x||_2^2$.

- Smoothness: $||\nabla f(y) - \nabla f(x)||_2 \leq \beta ||y - x||_2$ ($\Rightarrow \nabla^2 f(x) \preceq \beta I$).

  Quadratic upper bound $f(y) \leq f(x) + \nabla f(x)^T (y - x) + \frac{\beta}{2} ||y - x||_2^2$.

A strongly convex smooth function

For logistic regression

$$f(\mathbf{w}) = \sum_{i=1}^{n} \ln(1 + e^{-y_i x_i^T \mathbf{w}}) + \lambda ||\mathbf{w}||_2^2,$$

we have $\alpha = 2\lambda$ and $\beta = \frac{||\mathbf{X}||_2^2}{4} + 2\lambda$.

Strongly convex smooth functions

If $\alpha I \preceq \nabla^2 f(\mathbf{w}) \preceq \beta I$ with $\alpha > 0$, and $\eta_s = \frac{1}{\beta}$ for all $s \geq 1$. Then

$$f(\mathbf{w}_t) - f(\mathbf{w}^*) \leq \left(1 - \frac{\alpha}{\beta}\right)^{t-1} (f(\mathbf{w}_1) - f(\mathbf{w}^*)).$$

Faster Algorithms

Accelerated gradient descent

- Add a *momentum* term to gradient descent

$$\mathbf{w}_{s+1} = \mathbf{w}_s - \eta_s \nabla f(\mathbf{w}_s) + \alpha_s(\mathbf{w}_s - \mathbf{w}_{s-1}).$$

- The momentum term keeps going on the previous descent direction.

Using second-order information

- Newton's method uses the Hessian and converges within fewer iterations, but each iteration is more expensive.
- Quasi-Newton methods approximates the Hessian using previous gradients, converges within fewer iterations and has similar computational efficiency as gradient descent at each iteration.

Lower Bounds and Optimal Algorithms

Consider algorithms satisfying $\mathbf{w}_{s+1} \in \text{Span}(\nabla f(\mathbf{w}_1), \ldots, \nabla f(\mathbf{w}_s))$, how large is $f(\mathbf{w}_t) - f(\mathbf{w}^*)$ in the worst case?

Lower Bounds and Optimal Algorithms

Consider algorithms satisfying $\mathbf{w}_{s+1} \in \text{Span}(\nabla f(\mathbf{w}_1), \ldots, \nabla f(\mathbf{w}_s))$, how large is $f(\mathbf{w}_t) - f(\mathbf{w}^*)$ in the worst case?

Recall: We have seen upper bounds, e.g., for strongly convex and smooth f, $f(\mathbf{w}_t) - f(\mathbf{w}^*)$ is upper bounded by $O(c^t)$ for some constant $c < 1$.

Lower Bounds and Optimal Algorithms

Consider algorithms satisfying $\mathbf{w}_{s+1} \in \text{Span}(\nabla f(\mathbf{w}_1), \ldots, \nabla f(\mathbf{w}_s))$, how large is $f(\mathbf{w}_t) - f(\mathbf{w}^*)$ in the worst case?

Recall: We have seen upper bounds, e.g., for strongly convex and smooth f, $f(\mathbf{w}_t) - f(\mathbf{w}^*)$ is upper bounded by $O(c^t)$ for some constant $c < 1$.

Can we establish a lower bound? Can we say something for other function classes?

function	lower bound	optimal algorithm
convex, Lipschitz	$O(\frac{1}{\sqrt{t}})$	gradient descent
convex, smooth	$O(\frac{1}{t^2})$	accelerated gradient descent
strongly, Lipschitz	$O(\frac{1}{t})$	gradient descent
strongly, smooth	$O((1-\sqrt{\frac{\alpha}{\beta}})^t)$	accelerated gradient descent

The optimal algorithms achieve the lower bounds.

Optimization of Sums

- For machine learning, the objective function often has the form

$$f(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} f_i(\mathbf{w})$$

- Going through the whole dataset is expensive for big dataset.
- Idea: repeatedly update $\mathbf{w}$ using one or more randomly chosen f_i's.

Stochastic Gradient Descent (SGD)

General scheme

$\mathbf{w}_1 \in W$.

$\mathbf{w}_{s+1} = \Pi_W(\mathbf{w}_s - \eta_s g_s)$, where $g_s = \nabla f_{i_s}(\mathbf{w}_s)$ for an i_s randomly drawn from $[n]$.

A convergence result

If f is α-strongly convex, and $L^2 \geq \mathbb{E}\,||g_s||_2^2$ for all s, then SGD with $\eta_s = \frac{2}{(s+1)\alpha}$ satisfies

$$\mathbb{E}\, f(\bar{x}_t) - f(x^*) \leq \frac{2L^2}{(t+1)\alpha} \in O\left(\frac{1}{t}\right),$$

where $\bar{x}_t = \frac{2}{t(t+1)} \sum_{s=1}^{t} s x_s$.

Comparing non-stochastic and stochastic optimization

	lower bound	SGD
convex, Lipschitz	$O(\frac{1}{\sqrt{t}})$	$O(\frac{1}{\sqrt{t}})$
convex, smooth	$O(\frac{1}{t^2})$	$O(\frac{1}{\sqrt{t}})$
strongly, Lipschitz	$O(\frac{1}{t})$	$O(\frac{1}{t})$
strongly, smooth	$O((1-\sqrt{\frac{\alpha}{\beta}})^t)$	$O(\frac{1}{t})$

- Good news: SGD is as fast as deterministic for general non-smooth problems.
- Bad news: SGD is slow for smooth functions.

Variance reduction

- The slow convergence of SGD is due to large variance in the stochastic gradient.
- The idea is to reduce the variance by averaging.
  - Stochastic average gradient descent (SAG), stochastic variance reduced gradient descent (SVRG).
  - For the smooth case, can achieve $O(1/t)$ rate for convex functions and linear rate for strongly convex functions.

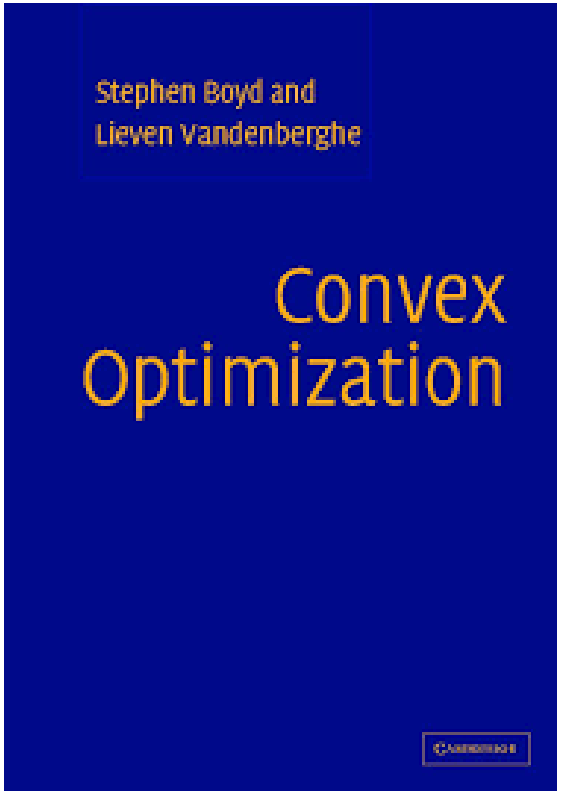
What You Need to Know...

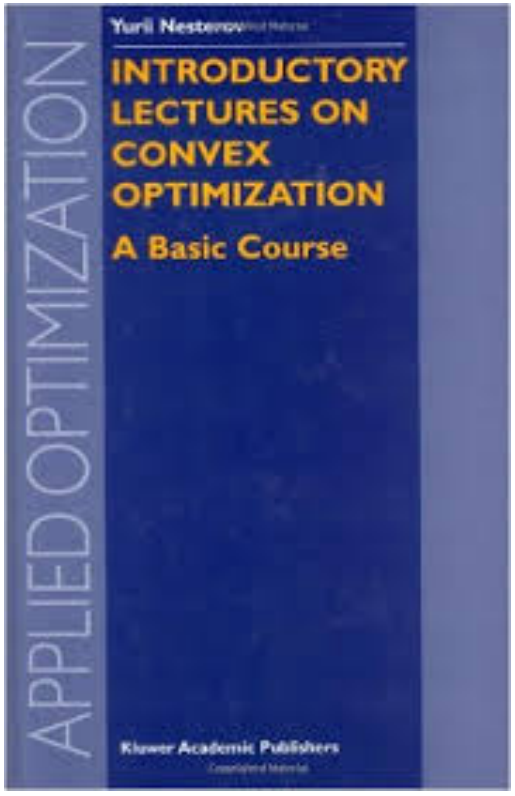
- Classes of optimization problems
- Gradient descent
- Stochastic gradient descent

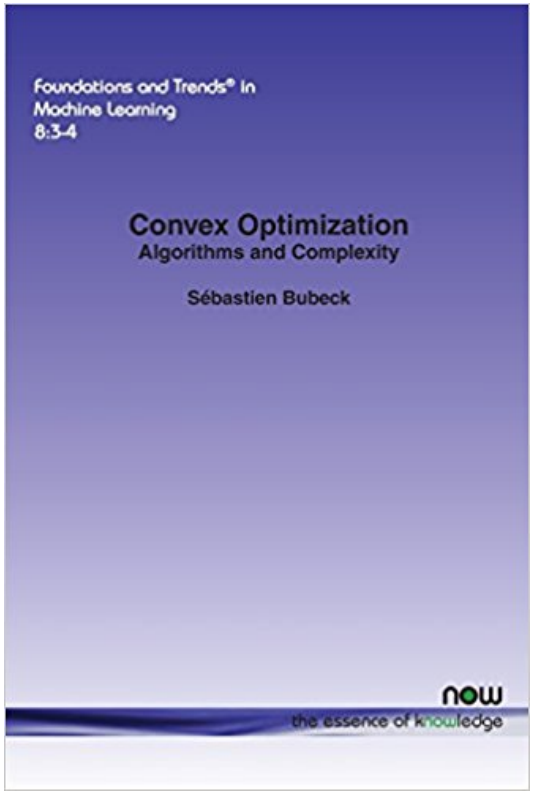
What's Going On

- Stochastic versions for quasi-Newton and Newton methods.
- Algorithms and theory for non-convex optimization.
- Explaining the effectiveness of the optimization algorithms for deep networks.

Some Reference Books







Outline

Why deep architectures?

First principles: statistics, optimization, regularization

- Illustrations
- Statistical learning theory
- Regularization
- Optimization

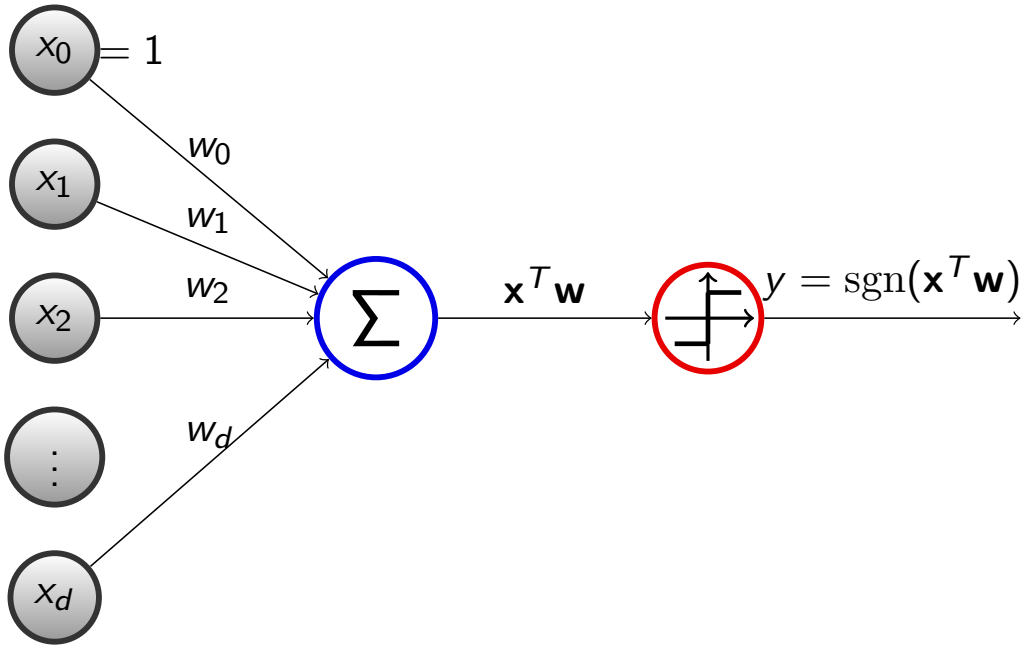
From shallow to deep networks

- Single-Neuron networks
- Multilayer perceptrons
- Convolutional neural networks
- Deep belief networks
- Other deep architectures

Single-Neuron networks

- The Perceptron
- Least squares regression
- Logistic regression

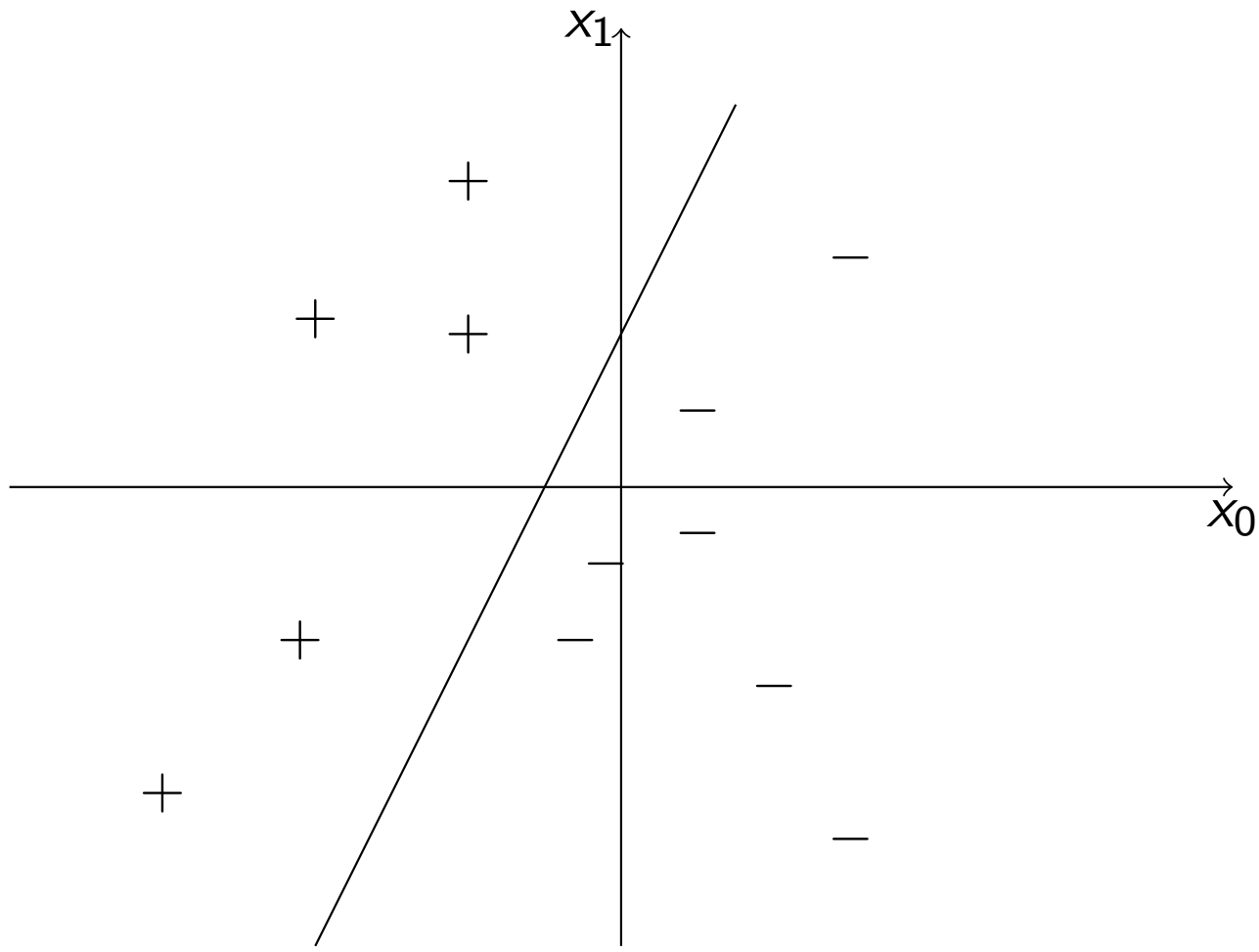
The Perceptron



A perceptron (linear threshold unit) maps an input $\mathbf{x} \in \mathbb{R}^{d+1}$ to

$$h(x) = \text{sgn}(\mathbf{x}^T\mathbf{w}) = \begin{cases} 1, & \mathbf{x}^T\mathbf{w} > 0, \\ 0, & \mathbf{x}^T\mathbf{w} = 0, \\ -1, & \mathbf{x}^T\mathbf{w} < 0. \end{cases}.$$

Here $\mathbf{x} = (1, x_1, \ldots, x_d)$ includes a dummy variable 1.



A perceptron corresponds to a linear decision boundary (i.e., the boundary between the regions for examples of the same class).

Perceptron Algorithm

Require: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \in \mathbb{R}^{d+1} \times \{-1, +1\}$, $\eta \in (0, 1]$.
Ensure: Weight vector $\mathbf{w}$.

Randomly or smartly initialize $\mathbf{w}$.
while there is any misclassified example **do**
  Pick a misclassified example $(\mathbf{x}_i, y_i)$.
  $\mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i$.

Why the update rule $\mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i$?

- $\mathbf{w}$ classifies $(\mathbf{x}_i, y_i)$ correctly if and only if $y_i \mathbf{x}_i^T \mathbf{w} > 0$.
- If $\mathbf{w}$ classifies $(\mathbf{x}_i, y_i)$ wrongly, then the update rule moves $y_i \mathbf{x}_i^T \mathbf{w}$ towards positive, because

$$y_i \mathbf{x}_i^T (\mathbf{w} + \eta y_i \mathbf{x}_i) = y_i \mathbf{x}_i^T \mathbf{w} + \eta ||\mathbf{x}_i||^2 > y_i \mathbf{x}_i^T \mathbf{w}.$$

Perceptron convergence theorem

If the training data is linearly separable (i.e., there exists some $\mathbf{w}^*$ such that $y_i \mathbf{x}_i^T \mathbf{w}^* > 0$ for all i), then the perceptron algorithm terminates with all training examples correctly classified.

Perceptron convergence theorem

If the training data is linearly separable (i.e., there exists some $\mathbf{w}^*$ such that $y_i \mathbf{x}_i^T \mathbf{w}^* > 0$ for all i), then the perceptron algorithm terminates with all training examples correctly classified.

Proof. Suppose $\mathbf{w}^*$ separates the data. We can scale $\mathbf{w}^*$ such that $|\mathbf{x}_i^T \mathbf{w}^*| \geq ||\mathbf{x}_i||_2^2$.

If $\mathbf{w}$ classifies $(\mathbf{x}_i, y_i)$ wrongly, then it will be updated to $\mathbf{w}' = \mathbf{w} + \eta y_i \mathbf{x}_i$.

We show that $||\mathbf{w}^* - \mathbf{w}'||_2^2 \leq ||\mathbf{w}^* - \mathbf{w}||_2^2 - \eta R^2$, where $R = \min_i ||\mathbf{x}_i||_2$. This implies that only finitely many updates is possible.

The inequality can be shown as follows.

$$
\begin{aligned}
||\mathbf{w}^* - \mathbf{w}'||_2^2 &= ||\mathbf{w}^* - \mathbf{w} - \eta y_i \mathbf{x}_i||_2^2 \\
&= ||\mathbf{w}^* - \mathbf{w}||_2^2 - 2\eta y_i \mathbf{x}_i^T (\mathbf{w}^* - \mathbf{w}) + \eta^2 y_i^2 ||\mathbf{x}_i||_2^2 \\
&\leq ||\mathbf{w}^* - \mathbf{w}||_2^2 - 2\eta (|\mathbf{x}_i^T \mathbf{w}^*| + |\mathbf{x}_i^T \mathbf{w}|) + \eta ||\mathbf{x}_i||_2^2 \\
&\leq ||\mathbf{w}^* - \mathbf{w}||_2^2 - 2\eta |\mathbf{x}_i^T \mathbf{w}^*| + \eta ||\mathbf{x}_i||_2^2 \\
&\leq ||\mathbf{w}^* - \mathbf{w}||_2^2 - \eta |\mathbf{x}_i^T \mathbf{w}^*| \\
&\leq ||\mathbf{w}^* - \mathbf{w}||_2^2 - \eta R^2.
\end{aligned}
$$

Starting with 0

Let $R = \max_i ||\mathbf{x}_i||$ be the radius of a ball centered at the origin and enclosing all the training data, and $\gamma = \min_i y_i \mathbf{w}^{*\top} \mathbf{x}_i / ||\mathbf{w}^*||$ be the margin of the the hyperplane $\mathbf{w}^*$. Suppose $\mathbf{w}_1 = 0$, then the Perceptron algorithm converges after at most R^2/γ^2 updates.

Starting with 0

Let $R = \max_i ||\mathbf{x}_i||$ be the radius of a ball centered at the origin and enclosing all the training data, and $\gamma = \min_i y_i \mathbf{w}^{*\top} \mathbf{x}_i / ||\mathbf{w}^*||$ be the margin of the the hyperplane $\mathbf{w}^*$. Suppose $\mathbf{w}_1 = 0$, then the Perceptron algorithm converges after at most R^2/γ^2 updates.

Proof. We have $\mathbf{w}_t^\top \mathbf{w}^* \geq t\eta\gamma||\mathbf{w}^*||$ because

$$\mathbf{w}_{t+1}^\top \mathbf{w}^* = \mathbf{w}_t^\top \mathbf{w}^* + \eta y_i \mathbf{x}_i^\top \mathbf{w}^* \geq \mathbf{w}_t^\top \mathbf{w}^* + \eta\gamma||\mathbf{w}^*||,$$

In addition, we have $||\mathbf{w}_t||^2 \leq t\eta^2 R^2$ because

$$||\mathbf{w}_{t+1}||^2 = ||\mathbf{w}_t + \eta y_i \mathbf{x}_i||^2 = ||\mathbf{w}_t||^2 + 2\eta y_i \mathbf{w}_t \mathbf{x}_i + \eta^2 ||\mathbf{x}_i||^2 \leq ||\mathbf{w}_t||^2 + \eta^2 R^2,$$

Combining the above two inequalities, we have

$$t\eta\gamma||\mathbf{w}^*|| \leq \mathbf{w}_t^\top \mathbf{w}^* \leq ||\mathbf{w}_t|| \cdot ||\mathbf{w}^*|| \leq \sqrt{t}\eta R||\mathbf{w}^*||.$$

Hence $t \leq R^2/\gamma^2$.

SGD interpretation

- Consider the loss function

$$L((\mathbf{x}, y), \mathbf{w}) = (-y\mathbf{w}^\top \mathbf{x})_+,$$

  for the classifier $h_{\mathbf{w}}(\mathbf{x}) = \text{sgn}(\mathbf{w}^\top \mathbf{x})$.
- The loss is 0 for correct classification, and $-y\mathbf{w}^\top x$ for incorrect classification.
- The empirical risk is

$$R_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} (-y_i \mathbf{w}^\top \mathbf{x}_i)_+, \tag{7.1}$$

- The Perceptron algorithm is SGD applied to $R_n(\mathbf{w})$.

Weaknesses of the algorithm

- When the data is separable, the hyperplane found by the perceptron algorithm depends on the initial weight and is thus arbitrary.
- Convergence can be very slow, especially when the gap between the positive and negative examples is small.
- When the data is not separable, the algorithm does not stop, but this can be difficult to detect.

It is NP-hard to minimize the empirical $0/1$ loss of a perceptron, that is, given a training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, it is NP-hard to solve

$$\min_{\mathbf{w}} \frac{1}{n} \mathsf{I}(\mathrm{sgn}(\mathbf{x}_i^T \mathbf{w}) \neq y_i)$$

What You Need to Know...

- The Perceptron
- The Perceptron algorithm
- Convergence, SGD interpretation, and weaknesses

Dealing with Intractability

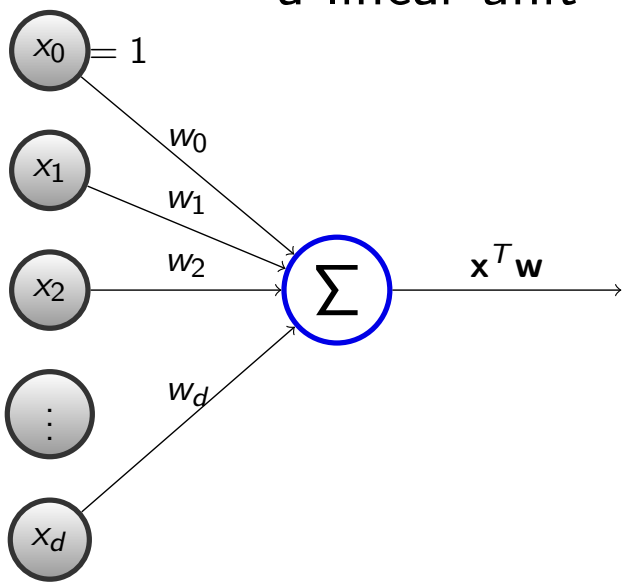
Idea: approximate the optimization problem using a surrogate loss.

- Least squares linear regression
- Logistic regression

First approximation

Least squares linear regression

a linear unit



$$\min_{\mathbf{w}} \sum_i (\mathbf{w}^T\mathbf{x}_i - y_i)^2$$

- Use a linear output $\hat{y} = \mathbf{w}^T\mathbf{x}$ instead of $\hat{y} = \text{sgn}(\mathbf{w}^T\mathbf{x})$.
- Use squared loss $(y - \hat{y})^2$ instead of 0/1 loss $\mathsf{I}(y \neq \hat{y})$.

SGD training

- Stochastic gradient

$$\nabla f_i(\mathbf{w}) = \nabla(y_i - \mathbf{w}^T\mathbf{x})^2 = -2(y_i - \mathbf{w}^T\mathbf{x})\mathbf{x}$$

.

- Update rule

$$\mathbf{w}_{s+1} = \mathbf{w}_s + 2\eta_s(y_i - \mathbf{w}^T\mathbf{x}_i)\mathbf{x}_i$$

.

- Also known as the delta rule, LMS rule, Adaline rule, Widrow-Hoff rule.

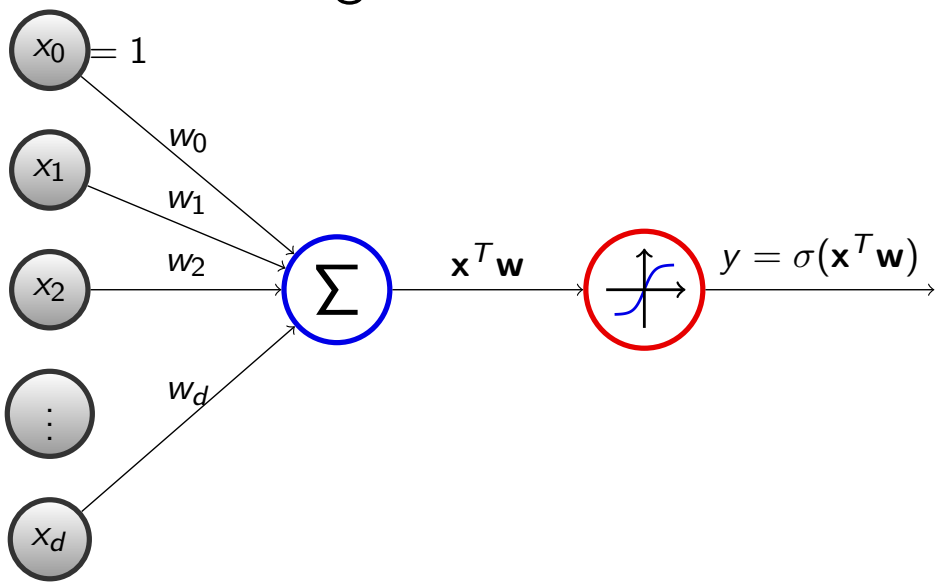
Weaknesses

- Least squares fitting may not find a separating hyperplane when there is one.
- In fact, it may give an error arbitrary close to 0.5 when there is a separating hyperplane.

Second Approximation

Logistic regression

a sigmoid unit



$$\min_{\mathbf{w}} \sum_{\mathbf{w}} -\ln \frac{1}{1+e^{-y_i \mathbf{w}^T \mathbf{x}_i}}$$

- Use $\hat{y} = \sigma(\mathbf{w}^T\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$ as the output. The output is the probability of positive.
- Use log loss $-\ln \frac{1}{1+e^{-y\mathbf{w}^T\mathbf{x}}}$.
- For prediction, output the most likely label.

SGD training

- Stochastic gradient

$$\nabla f_i(\mathbf{w}) = \nabla \ln(1 + e^{-y_i \mathbf{w}^T \mathbf{x}_i}) = -\sigma(-y_i \mathbf{w}^T \mathbf{x}_i) y_i \mathbf{x}_i$$

.

- Update rule

$$\mathbf{w}_{s+1} = \mathbf{w}_s + 2\eta_s \, \sigma(-y_i \mathbf{w}^T \mathbf{x}_i) y_i \mathbf{x}_i$$

.

Demo

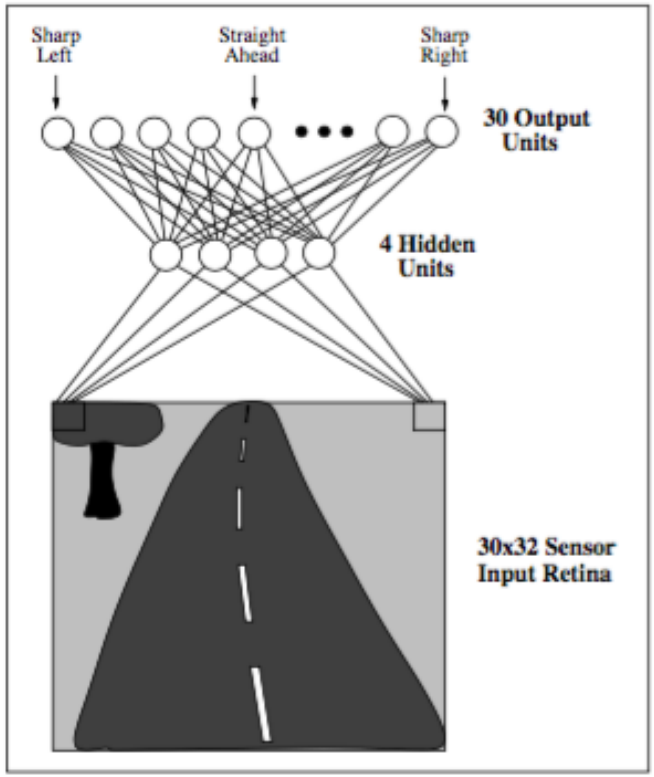
```
1  from numpy import random, dot, sign, zeros, ones, around, exp
2  from scipy.special import expit
3
4  # generate a random dataset consisting of 200 examples
5  n = 200
6  d = 10
7  X = random.rand(n, d) - 0.5
8  beta = ones(d)
9  Y = sign(dot(X, beta))
```

```
w = zeros(d)
for s in range(1000):
  i = random.randint(n)
  if Y[i] != sign(dot(w, X[i,])):
    w += 0.2 * Y[i] * X[i,]
print 'Perceptron error:', sum(abs(sign(dot(X, w)) - Y))/(2*n)

w = zeros(d)
for s in range(1000):
  i = random.randint(n)
  w += 0.2 * (Y[i] - dot(w, X[i,])) * X[i,]
print 'Least squares error:', sum(abs(sign(dot(X, w)) - Y))/(2*n)

w = zeros(d)
for s in range(1000):
  i = random.randint(n)
  w += 0.2 * Y[i] * X[i,] * expit(-Y[i] * dot(w, X[i,]))
print 'Logistic regression error:', sum(abs(2*around(expit(dot(X,
    w))) -1 - Y))/(2*n)
```

Multi-layer Perceptrons

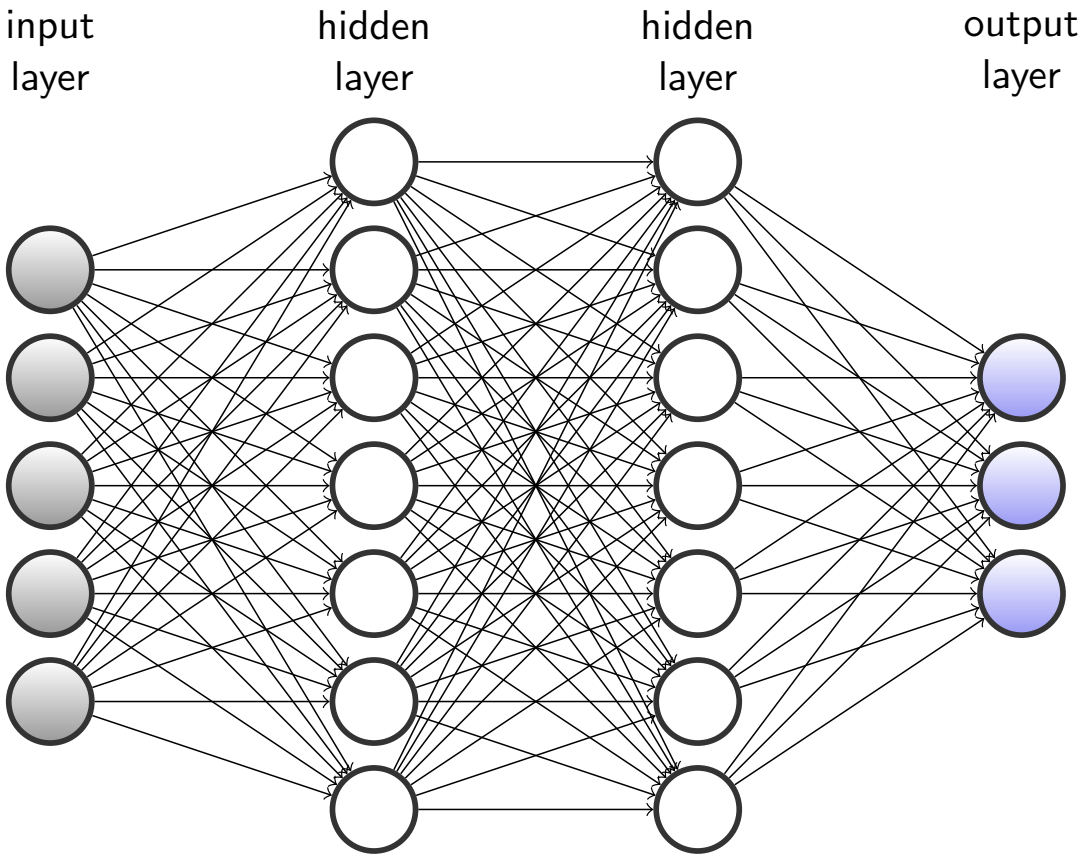
- Multi-layer perceptrons (MLPs)
- Backpropagation

ALVINN Driving at 70 MPH (1993)



Pomerleau, "Knowledge-based training of artificial neural networks for autonomous robot driving", 1993

MLP



a.k.a. multi-layer feedforward neural networks

Compute the output of a sigmoid MLP

- $P(j)$: the set of parents of j.
- o_i: the output of unit i. For an input neuron, o_i denotes its input.
- w_{ij}: weight on the directed edge (i, j).

For each neuron j,

$$o_j \leftarrow \sigma\Big(\sum_{i \in P(j)} w_{ij} o_i\Big)$$

when o_i's are computed.

Backprop

Train a single output sigmoid MLP $f(\mathbf{x}; \mathbf{w})$ for regression

$$\min_{\mathbf{w}} \sum_i \frac{1}{2}(f(\mathbf{x}_i; \mathbf{w}) - y_i)^2$$

Backprop is gradient descent for optimizing the network weights.

Gradient computation

For each (i, j), $g_{ij} \leftarrow 0$.
for each example $(\mathbf{x}, y)$ **do**
Compute all o_i's.
For the output unit k, $\delta_k \leftarrow o_k(1 - o_k)(y - o_k)$.
For each (i, j), $\delta_i \leftarrow o_i(1 - o_i) \sum_{j \in C(i)} w_{ij}\delta_j$ when δ_j's are computed.
For each (i, j), $g_{ij} \leftarrow g_{ij} + \delta_j o_i$.

Derivation

- Let $L(o_k, y) = \frac{1}{2}(o_k - y)^2$ be the loss function.
  Let $n_j = \sum_{i \in P(j)} w_{ij} o_i$ (linear combination of the inputs to i).
- Define $\delta_i = \partial L / \partial n_i$.

For the output unit k,

$$\delta_k = \frac{\partial L}{\partial n_k} = \frac{\partial L}{\partial o_k} \frac{\partial o_k}{\partial n_k} = 2(o_k - y) o_k (1 - o_k).$$

This is because $o_k = \sigma(n_k)$ and $\sigma'(n_k) = o_k(1 - o_k)$.

Derivation

- Let $L(o_k, y) = \frac{1}{2}(o_k - y)^2$ be the loss function.
  Let $n_j = \sum_{i \in P(j)} w_{ij} o_i$ (linear combination of the inputs to i).
- Define $\delta_i = \partial L / \partial n_i$.

We also have the following recurrence

$$\delta_i = \frac{\partial L}{\partial n_i} = \sum_{j \in C(i)} \frac{\partial L}{\partial n_j} \frac{\partial n_j}{\partial o_i} \frac{\partial o_i}{\partial n_i} = o_i(1 - o_i) \sum_{j \in C(i)} w_{ij} \delta_j.$$

Derivation

- Let $L(o_k, y) = \frac{1}{2}(o_k - y)^2$ be the loss function.
  Let $n_j = \sum_{i \in P(j)} w_{ij} o_i$ (linear combination of the inputs to i).
- Define $\delta_i = \partial L / \partial n_i$.

In addition, we have

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial n_j} \frac{\partial n_j}{\partial w_{ij}} = \delta_j o_i.$$

Consider a general loss $L(o_k, y)$, and an MLP with unit j having output $o_j = \sigma_j(n_j)$.

Gradient computation

For each (i, j), $g_{ij} \leftarrow 0$.
for each example $(\mathbf{x}, y)$ **do**
Compute all o_i's.
For the output unit k, $\delta_k \leftarrow \frac{\partial L}{\partial o_k} \sigma_k'(n_k)$.
For each (i, j), $\delta_i \leftarrow \sigma_i'(n_i) \sum_{j \in C(i)} w_{ij} \delta_j$ when δ_j's are computed.
For each (i, j), $g_{ij} \leftarrow g_{ij} + \delta_j o_i$.

Extensions

- We can extend the backpropagation algorithm to handle multiple output units.
- By choosing different loss functions and using multiple output neurons, we can train an MLP for classification and density estimation.

More on optimization

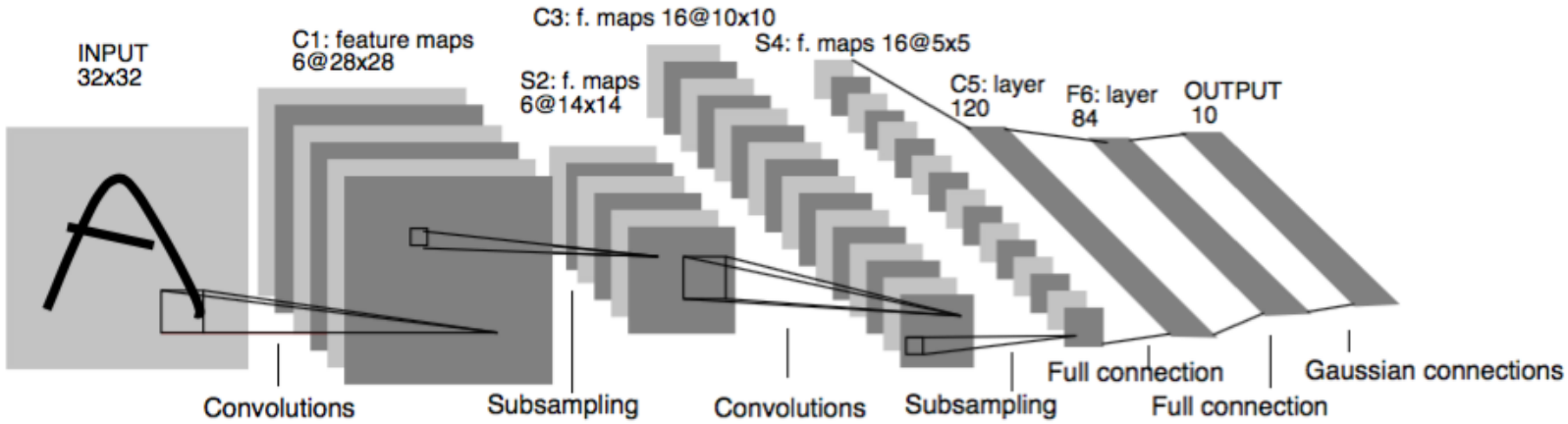
- In practice, we often use accelerated gradient descent to train the network.
- We can perform stochastic gradient descent instead of full gradient descent.
- Early stopping has a regularization effect.
  - A good fit on the training has not been found, and thus this alleviates overfitting.
  - Vanilla SGD might be better than variance reduced versions.

Backprop on deep networks

- While backprop on shallow networks found empirical successes, it was found to be trapped in local optima for deep networks.
- It only uses labeled examples, but such examples were few.
- Starting at a good initial weight is important.

Convolutional Neural Networks

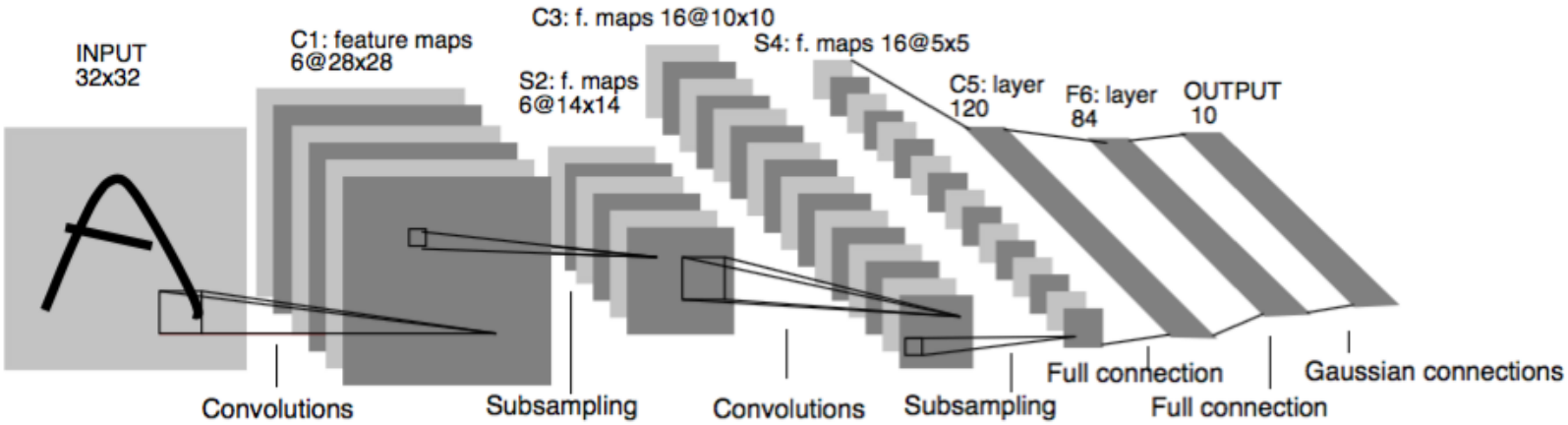
LeNet-5 (1989)



- 7 layers (excluding input layer)
- Layer 1,3,5 are convolution layers
- Layer 2,4 are sub-sampling layers
- Layer 6 is fully-connected
- Layer 7 is the output layer

LeCun et al., "Gradient-based learning applied to document recognition", 1998

LeNet-5 (1989)

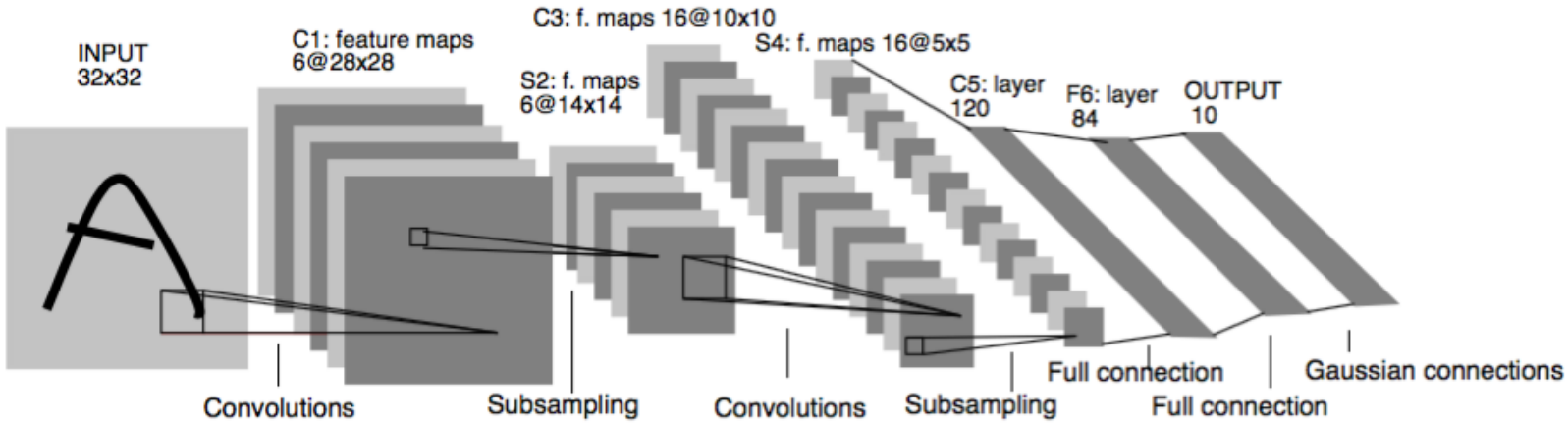


Convolutional layers

- Each convolutional layer has units organized as several 2D arrays.
- Each 2D array represents the states of applying a local filter (feature) by sliding, or convolving, it through the input space.
  - $h_{ij}^k = \sigma(W^k \cdot x_{ij} + b_k)$
  - The receptive fields of C1 consists of 5x5 local in the input image...

LeCun et al., "Gradient-based learning applied to document recognition", 1998

LeNet-5 (1989)

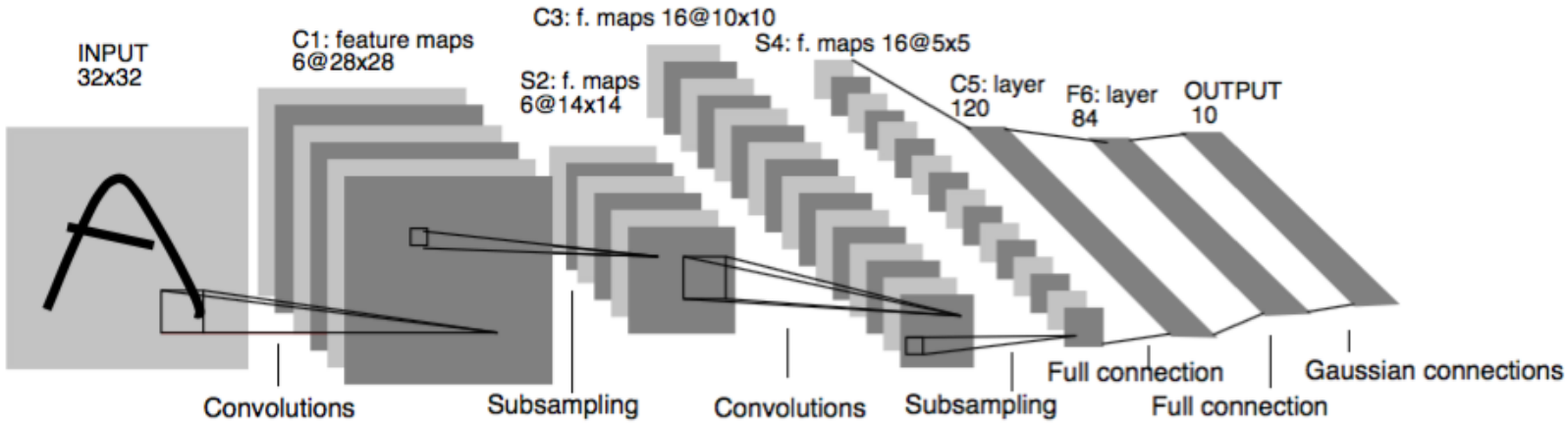


Sub-sampling/pooling layers

- Each sub-sampling layer has units organized as the same number of 2D arrays as previous convolutional layer.
- Reduces each 2D array in the previous convolutional layer to a lower resolution, by taking the sum of each non-overlapping 2x2 neighborhood and adding a bias to it.

LeCun et al., "Gradient-based learning applied to document recognition", 1998

LeNet-5 (1989)



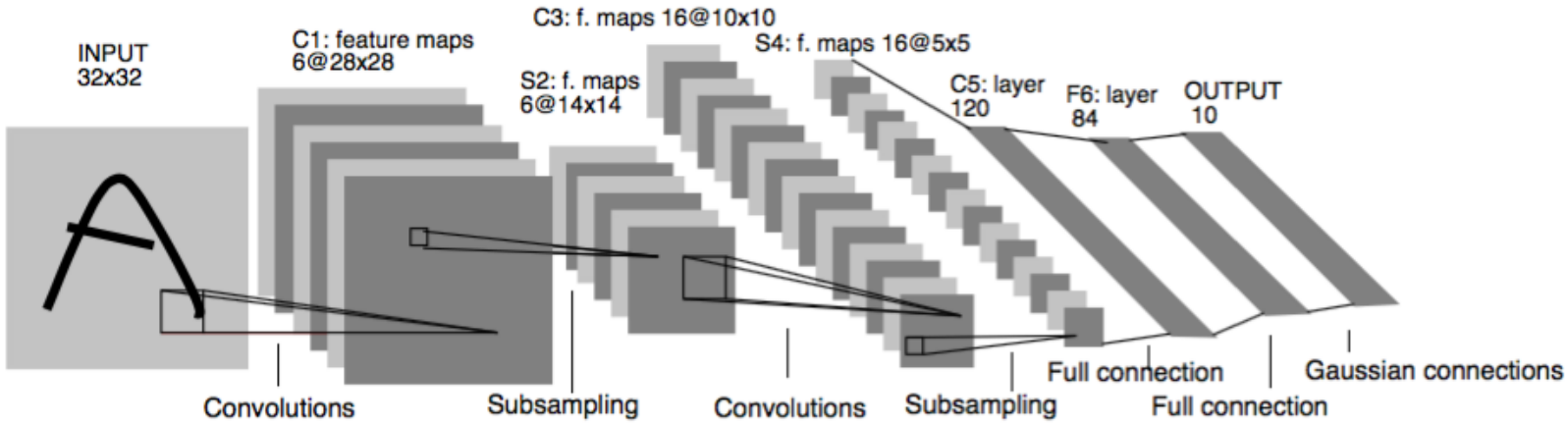
Three architectural ideas

- Local receptive fields
- Shared weights
- Spatial or temporal sub-sampling

These ensure some degree of shift, scale, and distortion invariance.

LeCun et al., "Gradient-based learning applied to document recognition", 1998

LeNet-5 (1989)



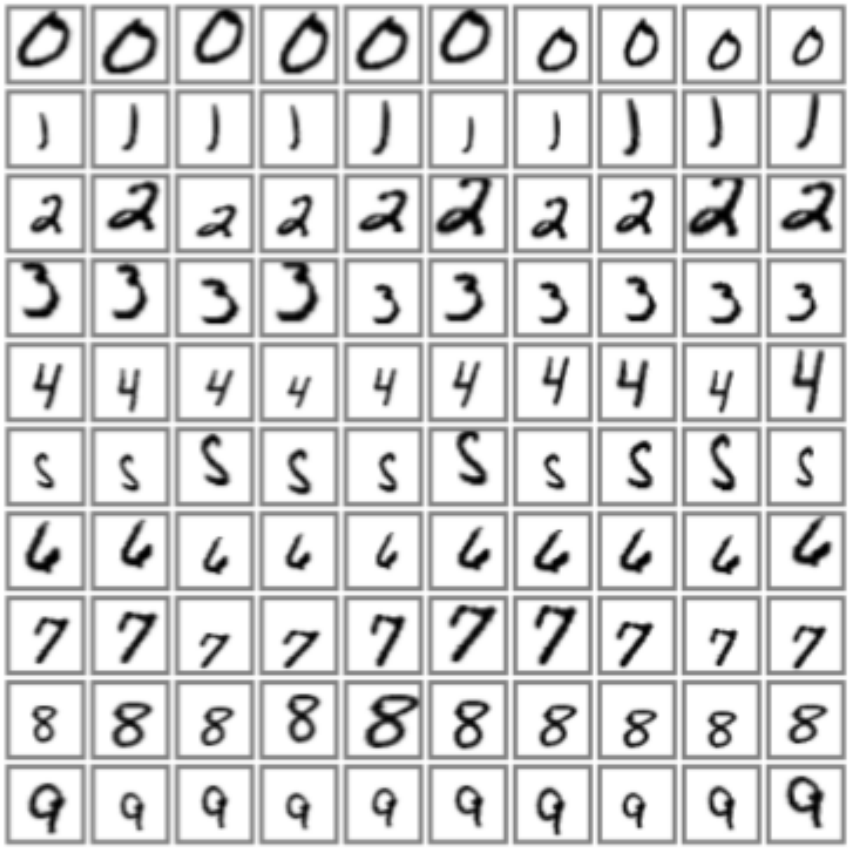
Trainable using backprop.

LeCun et al., "Gradient-based learning applied to document recognition", 1998

Performance

- MNIST dataset: 60,000 training examples, 10,000 test examples, size-normalized to 20x20 and centered by center of mass in 28x28 fields.
- 0.95% error.

Adding distorted training data helps



- Additional 540,000 distorted training examples.
- Error improved to 0.8%.

Errors made by LeNet5

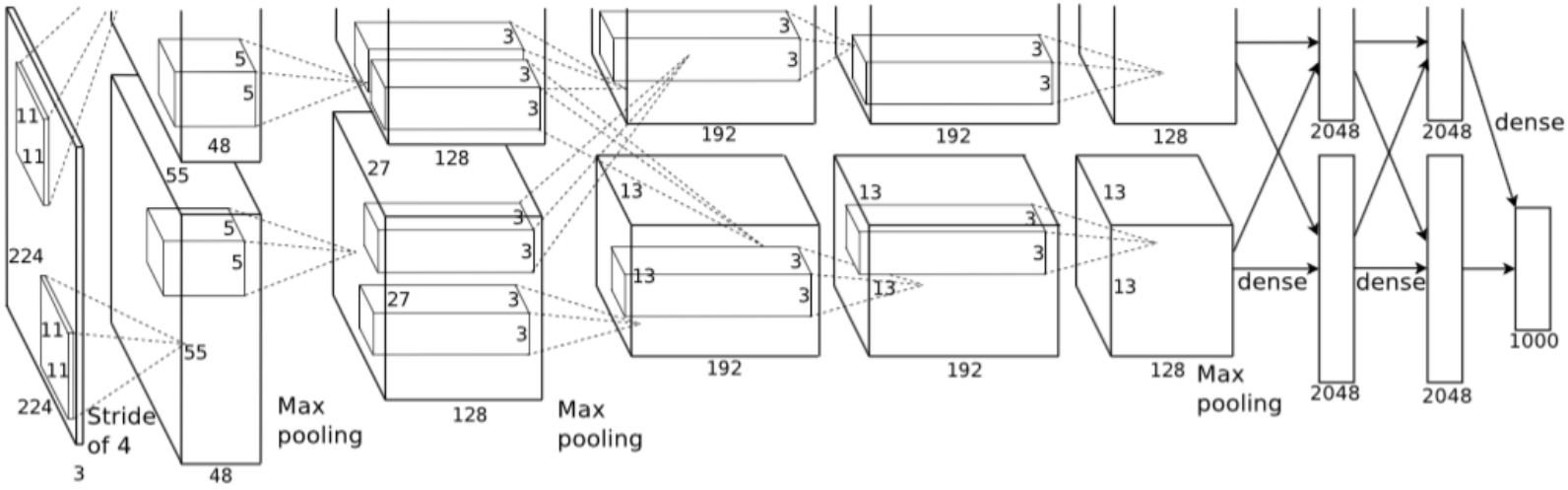
Variants

- Max-pooling (taking maximum in the neighborhood) is found to work better than average-pooling.
- Overlapping pooling.
- Rectified linear unit (ReLU, $\max(0, x)$) instead of sigmoid units ($\tanh(x)$ or $\sigma(x)$).
- Initial training with data from a related domain, followed by fine-tuning using in-domain data.

Regularization using dropout

- Each time we present a training example, randomly omit each hidden unit with probability 0.5.
- All architectures share weights.
- Only a few of the models ever get trained.
- Better regularizer than L2 or L1
- Test: geometric mean of the outputs of many sampled architectures, use all weights with halving of their outgoing weights.

ImageNet Classification with Deep ConvNet (2012)



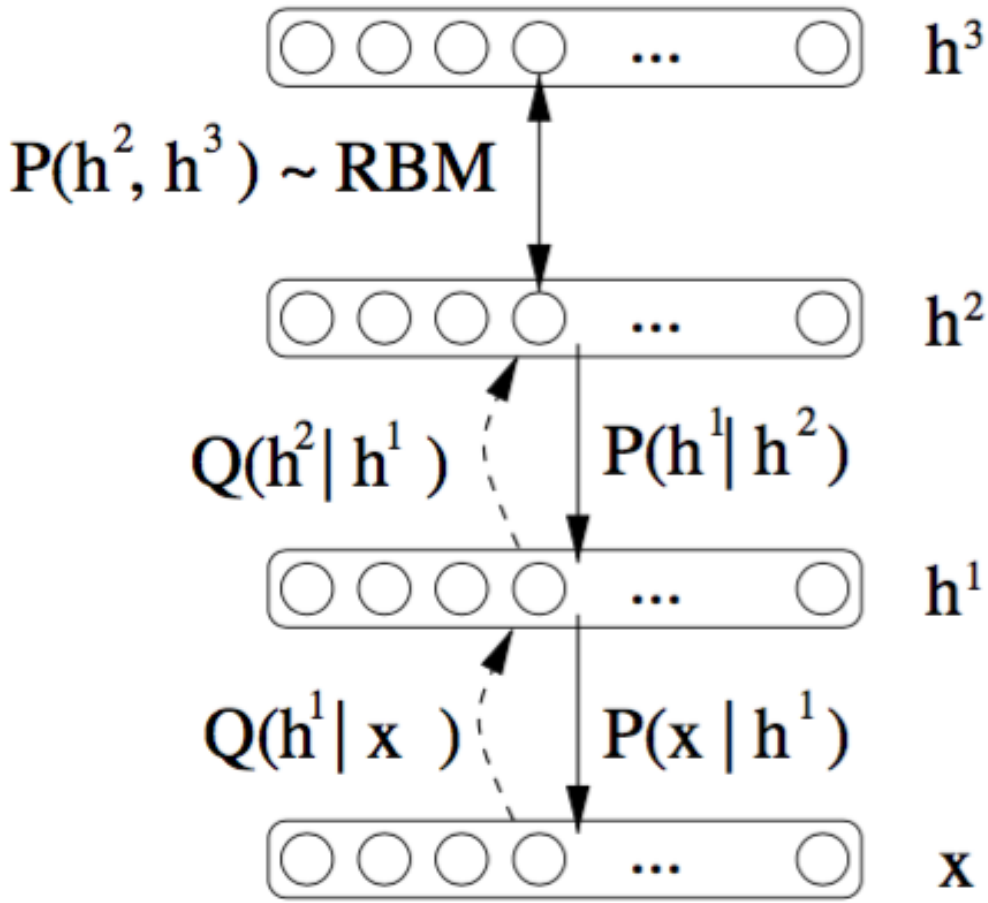
- Five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax.
- 60 million parameters and 650,000 neurons

Wide applications

- Image and video recognition
- Recommender systems
- Natural language processing

Deep Belief Networks

Deep belief network



$$P(\mathbf{x}, \mathbf{h}^1, \ldots, \mathbf{h}^\ell) = \left(\prod_{k=0}^{\ell-2} P(\mathbf{h}^k \mid \mathbf{h}^{k+1}) \right) P(\mathbf{h}^{\ell-1}, \mathbf{h}^\ell), \text{where } \mathbf{h}^0 = \mathbf{x}.$$

Restricted Boltzman Machines

$$P(\mathbf{x}, \mathbf{h}) \propto e^{-\mathbf{b}^T \mathbf{x} - \mathbf{c}^T \mathbf{h} - \mathbf{h}^T W \mathbf{x}}.$$

- A fully connected 2-layer model with an observed layer $\mathbf{x}$ and a hidden layer $\mathbf{h}$.
- Easy to draw samples from $P(\mathbf{x}, \mathbf{h})$ using Gibbs sampling because the factorization makes $P(\mathbf{h} \mid \mathbf{x})$ and $P(\mathbf{x} \mid \mathbf{h})$ easy to compute.

Unsupervied layerwise training

for $k = 1$ to ℓ **do**

- initialize $W^k = 0$, $\mathbf{b}^k = 0$, $\mathbf{c}^k = 0$

while not stopping criterion **do**

- sample $\mathbf{h}^0 = \mathbf{x}$ from $\widehat{P}$

for $i = 1$ to $k - 1$ **do**

if `mean_field_computation` **then**

- **assign** $\mathbf{h}^i_j$ **to** $Q(\mathbf{h}^i_j = 1|\mathbf{h}^{i-1})$, **for all elements** j **of** $\mathbf{h}^i$

else

- **sample** $\mathbf{h}^i_j$ **from** $Q(\mathbf{h}^i_j|\mathbf{h}^{i-1})$, **for all elements** j **of** $\mathbf{h}^i$

end if

end for

- `RBMupdate`$(\mathbf{h}^{k-1}, \epsilon, W^k, \mathbf{b}^k, \mathbf{c}^k)$ {thus providing $Q(\mathbf{h}^k|\mathbf{h}^{k-1})$ for future use}

end while

end for

- Unsupervised training initializes the network parameters in a good region to start with.
- The parameters are then tuned using supervised training.

Contrastive divergence

for all hidden units i **do**
- compute $Q(\mathbf{h}_{1i}=1|\mathbf{x}_1)$ (for binomial units, $\text{sigm}(\mathbf{c}_i+\sum_j W_{ij}\mathbf{x}_{1j})$)
- sample $\mathbf{h}_{1i} \in \{0,1\}$ from $Q(\mathbf{h}_{1i}|\mathbf{x}_1)$

end for
for all visible units j **do**
- compute $P(\mathbf{x}_{2j}=1|\mathbf{h}_1)$ (for binomial units, $\text{sigm}(\mathbf{b}_j+\sum_i W_{ij}\mathbf{h}_{1i})$)
- sample $\mathbf{x}_{2j} \in \{0,1\}$ from $P(\mathbf{x}_{2j}=1|\mathbf{h}_1)$

end for
for all hidden units i **do**
- compute $Q(\mathbf{h}_{2i}=1|\mathbf{x}_2)$ (for binomial units, $\text{sigm}(\mathbf{c}_i+\sum_j W_{ij}\mathbf{x}_{2j})$)

end for
- $W \leftarrow W + \epsilon(\mathbf{h}_1\mathbf{x}_1' - Q(\mathbf{h}_{2\cdot}=1|\mathbf{x}_2)\mathbf{x}_2')$
- $\mathbf{b} \leftarrow \mathbf{b} + \epsilon(\mathbf{x}_1 - \mathbf{x}_2)$
- $\mathbf{c} \leftarrow \mathbf{c} + \epsilon(\mathbf{h}_1 - Q(\mathbf{h}_{2\cdot}=1|\mathbf{x}_2))$

- Contrastive divergence is optimizing $P(\mathbf{x})$ using SGD with a biased stochastic gradient.

DBN achieves 1.25% errors on the MNIST dataset

Other Deep Architectures

- Long Short-Term Memory (LSTM): time series with uncertain time tags between important events, best results in natural language compression, unsegmented connected handwriting recognition, automatic speech recognition.
- Generative adversarial networks (GAN): generative photorealistic images for visualization, model rudimentary patterns of motion in video, reconstruct 3D models of objects from images, improve astronomical images.
- Autoencoders: learn a representation of data for dimensionality reduction.
- Deep Q-networks...

Some Learning Resources

- Awesome Deep Learning: a collection of learning resources, including free e-books, courses, tutorials, websites...
- The most cited deep learning papers: influential papers divided into groups
- Deep learning resources for computer vision
- Deep learning tutorial in Theano

Software

- Caffe: developed by the Berkeley Vision and Learning Center (BVLC).
- TensorFlow: originally developed by Google Brain Team.
- Torch: used by FAIR, IBM, Yandex, Idiap Research Institute.
- Theano: primarily developed by researchers at Université de Montréal.

Tutorial Objective

Why deep architectures?

Understand three core ideas in machine learning

Statistics: what is a good model for a training set

Regularization: how to embed prior knowledge

Optimization: how to compute the desired model

Understand common shallow and deep neural networks

Perceptrons, multi-layer perceptrons, convolutional neural networks, deep belief networks...