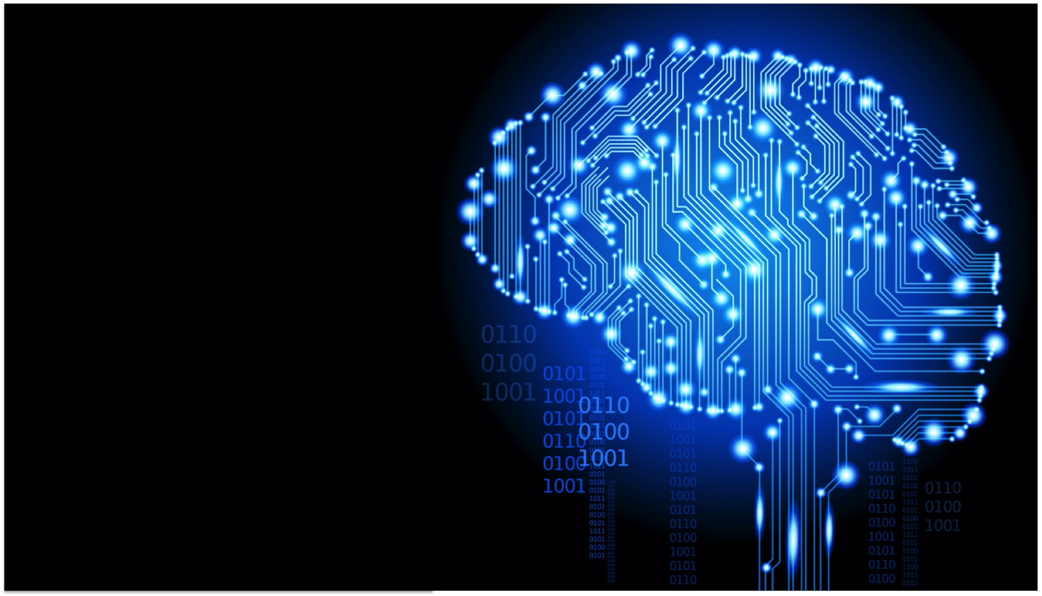
Convex Optimization for Machine Learning



Sunghee Yun (sunghee.yun@gmail.com)

Software R&D Center
Samsung Electronics

About the speaker

- Sunghee Yun
  - B.S., Electrical Engineering @ Seoul National University
  - M.S., Electrical Engineering @ Stanford University
  - Ph.D., Electrical Engineering @ Stanford University
  - CAE Team @ Semiconductor R&D Center
  - Design Technology Team @ DRAM Development Lab.
  - Memory Sales & Marketing Team @ Memory Business Unit
  - (currently) Software R&D Center
- Specialties
  - convex optimization
  - decentralized deep learning

Today

- Convex optimization
- Machine learning
  - four perspectives: statistics, computer science, numerical algorithms, hardware
- Deep learning
  - CNN & RNN
- AI Applications
  - image classification, self-driving cars, security, IoT, bio-medical

Prerequisite for the talk

This talk will assume the audience

- has been exposed to basic linear algebra
- can distinguish between componentwise inequality and that for positive semidefiniteness, $i.e.$,

$$Ax \preceq b \Leftrightarrow \begin{bmatrix} a_1^T \\ \vdots \\ a_m^T \end{bmatrix} x \preceq \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} \Leftrightarrow a_i^T x \leq b_i \text{ for } i = 1, \ldots, m,$$

but,

$$A \succeq 0 \Leftrightarrow A = A^T \text{ and } x^T A x \geq 0 \text{ for all } x \in \mathbf{R}^n$$
$$A \succ 0 \Leftrightarrow A = A^T \text{ and } x^T A x > 0 \text{ for all nonzero } x \in \mathbf{R}^n$$

Why convex optimization?

- many machine learning algorithms (inherently) depend on convex optimization
- one of few optization class that can be actually solved
- a number of engineering and scientific problems can be cast into convex optimization problems
- many more can be approximated to convex optimization
- convex optimization sheds lights on intrinsic property and structure of many optimization, hence, machine learning algorithms

Why convex optimization?

- many machine learning algorithms (inherently) depend on convex optimization
- one of few optization class that can be actually solved
- a number of engineering and scientific problems can be cast into convex optimization problems
- many more can be approximated to convex optimization
- convex optimization sheds lights on intrinsic property and structure of many optimization, hence, machine learning algorithms

Why convex optimization?

- many machine learning algorithms (inherently) depend on convex optimization
- one of few optization class that can be actually solved
- a number of engineering and scientific problems can be cast into convex optimization problems
- many more can be approximated to convex optimization
- convex optimization sheds lights on intrinsic property and structure of many optimization, hence, machine learning algorithms

Why convex optimization?

- many machine learning algorithms (inherently) depend on convex optimization
- one of few optization class that can be actually solved
- a number of engineering and scientific problems can be cast into convex optimization problems
- many more can be approximated to convex optimization
- convex optimization sheds lights on intrinsic property and structure of many optimization, hence, machine learning algorithms

Why convex optimization?

- many machine learning algorithms (inherently) depend on convex optimization
- one of few optization class that can be actually solved
- a number of engineering and scientific problems can be cast into convex optimization problems
- many more can be approximated to convex optimization
- convex optimization sheds lights on intrinsic property and structure of many optimization, hence, machine learning algorithms

Why convex optimization?

- many machine learning algorithms (inherently) depend on convex optimization
- one of few optization class that can be actually solved
- a number of engineering and scientific problems can be cast into convex optimization problems
- many more can be approximated to convex optimization
- convex optimization sheds lights on intrinsic property and structure of many optimization, hence, machine learning algorithms

Mathematical optimization

- mathematical optimization problem:

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \;\; i = 1, \ldots, m \\ & h_i(x) = 0, \;\; i = 1, \ldots, p \end{array}$$

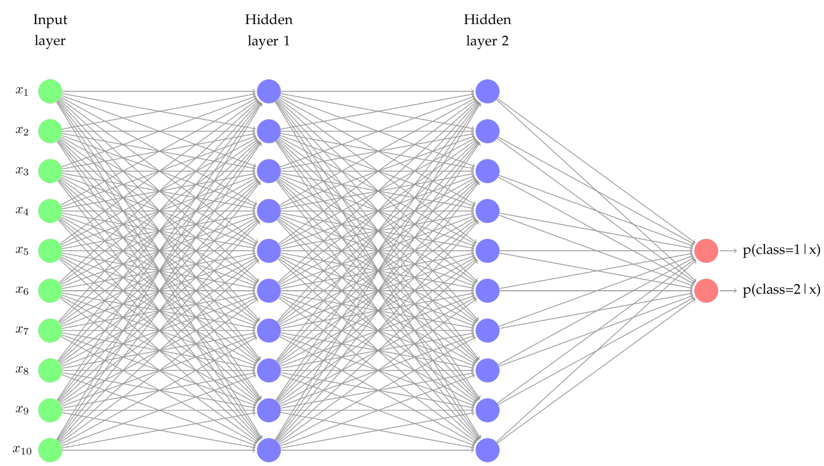
  - $x = \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix}^T \in \mathbf{R}^n$ is the (vector) optimization variable
  - $f_0 : \mathbf{R}^n \to \mathbf{R}$ is the objective function
  - $f_i : \mathbf{R}^n \to \mathbf{R}$ are the inequality constraint functions
  - $h_i : \mathbf{R}^n \to \mathbf{R}$ are the equality constraint functions

Optimization examples

- circuit optimization
  - optimization variables: transistor widths, resistances, capacitances, inductances
  - objective: operating speed (or equivalently, maximum delay)
  - constraints: area, power consumption
- portfolio optimization
  - optimization variables: amounts invested in different assets
  - objective: expected return
  - constraints: budget, overall risk, return variance

Optimization examples

- machine learning
  - optimization variables: model parameters ($e.g.$, connection weights)
  - objective: squared error (or loss function)
  - constraints: network architecture



Solution methods

- for general optimization problems
  - extremly difficult to solve (practically impossible to solve)
  - most methods try to find (good) suboptimal solutions, $e.g.$, using heuristics
- some exceptions
  - least-squares (LS)
  - liner programming (LP)
  - semidefinite programming (SDP)

Least-squares (LS)

- least-squares (LS) problem:

$$\text{minimize} \quad \|Ax - b\|_2^2 = \sum_{i=1}^{m} (a_i^T x - b_i)^2$$

  - analytic solution: any solution satisfying $(A^T A)x^* = A^T b$
  - extremely reliable and efficient algorithms
  - has been there at least since Gauss
- applications
  - LS problems are easy to recognize
  - has huge number of applications, $e.g.$, line fitting

Linear programming (LP)

- linear program (LP):

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \preceq b \end{array}$$

  - no analytic solution
  - reliable and efficient algorithms exist, *e.g.*, simplex method, interiorpoint method
  - has been there at least since Fourier
  - systematical algorithm existed since World War II
- applications
  - less obvious to recognize (than LS)
  - lots of problems can be cast into LP, *e.g.*, network flow problem

Semidefinite programming (SDP)

- semidefinite program (SDP):

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & F_0 + x_1 F_1 + \cdots + x_n F_n \succeq 0 \end{array}$$

  - no analytic solution
  - but, reliable and efficient algorithms exist, $e.g.$, interior-point method
  - recent technology

- applications
  - never easy to recognize
  - lots of problems, $e.g.$, optimal control theory, can be cast into SDP
  - extremely non-obvious, but convex, hence global optimality easily achieved!

Max-det problem (extension of SDP)

- max-det program:

$$\begin{array}{ll} \text{minimize} & c^T x + \log\det(F_0 + x_1 F_1 + \cdots + x_n F_n) \\ \text{subject to} & G_0 + x_1 G_1 + \cdots + x_n G_n \succeq 0 \end{array}$$

  - no analytic solution
  - but, reliable and efficient algorithms exist, $e.g.$, interior-point method
  - recent technology
- applications
  - never easy to recognize
  - lots of stochastic optimization problems, $e.g.$, every covariance matrix is positive semidefinite
  - again convex, hence global optimality (relatively) easily achieved!

Common features in these Exceptions?

- they are convex optimization problems!
- convex optimization:

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \preceq_{K_i} 0, \;\; i = 1, \ldots, m \\ & Ax = b \end{array}$$

where

- $f_0(\lambda x + (1-\lambda)y) \leq \lambda f_0(x) + (1-\lambda) f_0(y)$ for all $x, y \in \mathbf{R}^n$ and $0 \leq \lambda \leq 1$
- $f_i : \mathbf{R}^n \to \mathbf{R}^{k_i}$ are K_i-convex w.r.t. proper cone $K_i \subseteq \mathbf{R}^{k_i}$
- all equality constraints are linear

Convex optimization

- algorithms
  - classical algorithms like simplex method still work well for many LPs
  - many state-of-the-art algorithms develoled for (even) large-scale convex optimization problems
    * barrier methods
    * primal-dual interior-point methods
- applications
  - huge number of engineering and scientific problems are (or can be cast into) convex optimization problems
  - convex relaxation

What's fuss about convex optimization?

- which one of these problems are easier to solve?
  - (generalized) geometric program with $n = 3,000$ variables and $m = 1,000$ constraints

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{p_0} \alpha_{0,i} x_1^{\beta_{0,i,1}} \cdots x_n^{\beta_{0,i,n}} \\ \text{subject to} & \sum_{i=1}^{p_j} \alpha_{j,i} x_1^{\beta_{j,i,1}} \cdots x_n^{\beta_{j,i,n}} \leq 1, \;\; j = 1, \ldots, m \end{array}$$

    with $\alpha_{j,i} \geq 0$ and $\beta_{j,i,k} \in \mathbf{R}$
    $\Rightarrow$ can be solved within 1 minute *globally* in your laptop computer
  - minimization of 10th order polynomial of $n = 20$ variables with no constraint

$$\text{minimize} \quad \sum_{i_1=1}^{10} \cdots \sum_{i_n=1}^{10} c_{i_1,\ldots,i_n} x_1^{i_1} \cdots x_n^{i_n}$$

    with $c_{i_1,\ldots,i_n} \in \mathbf{R}$
    $\Rightarrow$ you *cannot* solve!

What's fuss about convex optimization?

- which one of these problems are easier to solve?
  - (generalized) geometric program with $n = 3,000$ variables and $m = 1,000$ constraints

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{p_0} \alpha_{0,i} x_1^{\beta_{0,i,1}} \cdots x_n^{\beta_{0,i,n}} \\ \text{subject to} & \sum_{i=1}^{p_j} \alpha_{j,i} x_1^{\beta_{j,i,1}} \cdots x_n^{\beta_{j,i,n}} \leq 1, \; j = 1, \ldots, m \end{array}$$

    with $\alpha_{j,i} \geq 0$ and $\beta_{j,i,k} \in \mathbf{R}$
    $\Rightarrow$ can be solved within 1 minute *globally* in your laptop computer
  - minimization of 10th order polynomial of $n = 20$ variables with no constraint

$$\text{minimize} \quad \sum_{i_1=1}^{10} \cdots \sum_{i_n=1}^{10} c_{i_1,\ldots,i_n} x_1^{i_1} \cdots x_n^{i_n}$$

    with $c_{i_1,\ldots,i_n} \in \mathbf{R}$
    $\Rightarrow$ you *cannot* solve!

What's fuss about convex optimization?

- which one of these problems are easier to solve?
  - (generalized) geometric program with $n = 3,000$ variables and $m = 1,000$ constraints

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{p_0} \alpha_{0,i} x_1^{\beta_{0,i,1}} \cdots x_n^{\beta_{0,i,n}} \\ \text{subject to} & \sum_{i=1}^{p_j} \alpha_{j,i} x_1^{\beta_{j,i,1}} \cdots x_n^{\beta_{j,i,n}} \leq 1, \; j = 1, \ldots, m \end{array}$$

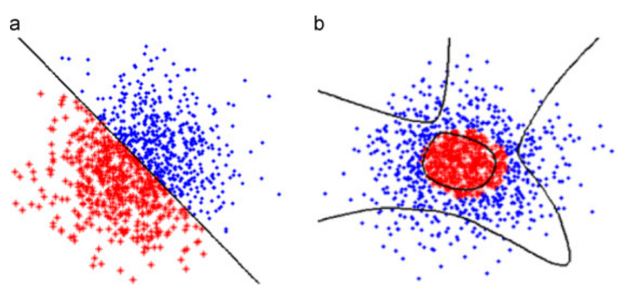
    with $\alpha_{j,i} \geq 0$ and $\beta_{j,i,k} \in \mathbf{R}$
    $\Rightarrow$ can be solved within 1 minute *globally* in your laptop computer
  - minimization of 10th order polynomial of $n = 20$ variables with no constraint

$$\text{minimize} \quad \sum_{i_1=1}^{10} \cdots \sum_{i_n=1}^{10} c_{i_1,\ldots,i_n} x_1^{i_1} \cdots x_n^{i_n}$$

    with $c_{i_1,\ldots,i_n} \in \mathbf{R}$
    $\Rightarrow$ you *cannot* solve!

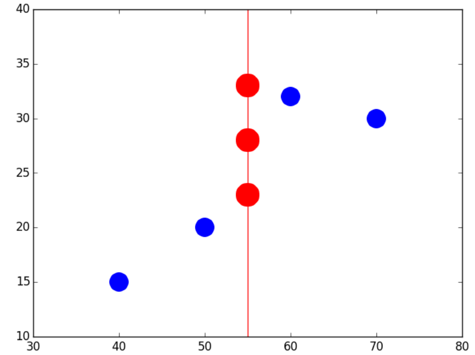
What is machine learning?

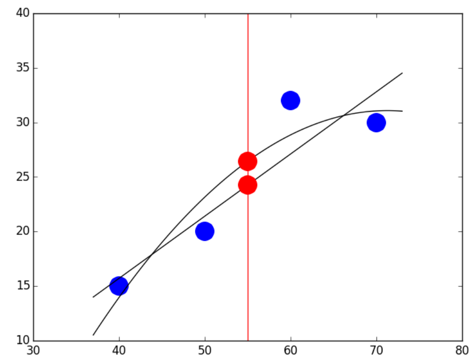
- machine learning
  - is the subfield of computer science that "gives computers the ability to learn without being explicitly programmed." (Arthur Samuel, 1959)
  - learns from data and predicts on data
- applications
  - spam fitering, search engine
  - detection of network intruders (or malicious insiders)
  - computer vision, speach recognition, natural language processing



ML example: regression

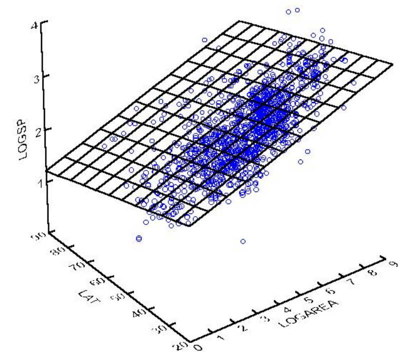
- problem: what is a reasonable price for a house?
  - what would a rational (or rather normal) human being do?
  - ML approach:
    - collect data: x: size, y: price
    - train model: draw a line to represent (typical) trend
    - predict a price from the line



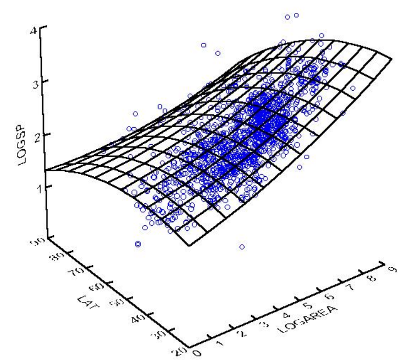


ML example: multi-variate regression

- what if we have more than one x? or rather more than two x's?



- what if highly nonlinera and nonconvex fitting function is needed?



Mathematical formulation for (supervised) ML

- given training set, $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$, where $x^{(i)} \in \mathbf{R}^p$ and $y^{(i)} \in \mathbf{R}^q$
- want to find function $g_\theta : \mathbf{R}^p \to \mathbf{R}^q$ with learning parameter, $\theta \in \mathbf{R}^n$
  - $g_\theta(x)$ desired to be as close as possible to y for future $(x, y) \in \mathbf{R}^p \times \mathbf{R}^q$
  - *i.e.*, $g_\theta(x) \sim y$
- define a loss function $l : \mathbf{R}^q \times \mathbf{R}^q \to \mathbf{R}_+$
- solve the optimization problem:

$$\begin{array}{ll} \text{minimize} & f(\theta) = \frac{1}{m} \sum_{i=1}^m l(g_\theta(x^{(i)}), y^{(i)}) \\ \text{subject to} & \theta \in \Theta \end{array}$$

Gifts I

- genetic algorithm learning how to swing

- multi-class classification using deep learning

Linear regression

- (simple) linear regression is a ML method when
  - $q = 1$, $i.e.$, the output is scalar
  - $g_\theta(x) = \theta^T \begin{bmatrix} 1 \\ x \end{bmatrix} = \theta_0 + \theta_1 x_1 + \cdots + \theta_p x_p$, $i.e.$, $n = p + 1$
  - $l : \mathbf{R} \times \mathbf{R} \to \mathbf{R}_+$ is defined by $l(y_1, y_2) = (y_1 - y_2)^2$
  - $\Theta = \mathbf{R}^{p+1}$, $i.e.$, parameter domain is all the real numbers
- formulation

$$\text{minimize} \quad f(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(\theta^T \begin{bmatrix} 1 \\ x^{(i)} \end{bmatrix} - y^{(i)} \right)^2$$

Solution method for linear regression

- linear regression is nothing but LS since

$$
\begin{aligned}
mf(\theta) &= \sum_{i=1}^{m} \left(\theta^T \begin{bmatrix} 1 \\ x^{(i)} \end{bmatrix} - y^{(i)} \right)^2 = \left\| \begin{bmatrix} 1 & {x^{(1)}}^T \\ \vdots & \vdots \\ 1 & {x^{(m)}}^T \end{bmatrix} \theta - \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \right\|_2^2 \\
&= \|X\theta - y\|_2^2
\end{aligned}
$$

- convex in θ, hence obtains its global optimality when the gradient vanishes, $i.e.$,

$$
m\nabla f(\theta) = 2X^T(X\theta - y) = 2((X^T X)\theta - X^T y) = 0
$$

- analytic solution exists and in practice,
  - QR decomposition or single value decomposition (SVD) can be used

Multiple output linear regression

- multiple output linear regression is a ML method when
  - $g_\theta(x) = \theta^T \begin{bmatrix} 1 \\ x \end{bmatrix} = \begin{bmatrix} \theta_{1,0} + \theta_{1,1}x_1 + \cdots + \theta_{1,p}x_p \\ \vdots \\ \theta_{q,0} + \theta_{q,1}x_1 + \cdots + \theta_{q,p}x_p \end{bmatrix}$
  - $l : \mathbf{R}^q \times \mathbf{R}^q \to \mathbf{R}_+$ is defined by $l(y_1, y_2) = \|y_1 - y_2\|_2^2$
  - $\Theta = \mathbf{R}^{(p+1)\times q}$, *i.e.*, parameter domain is all the real numbers
- formulation

$$\text{minimize} \quad f(\theta) = \frac{1}{m}\sum_{i=1}^{m} \left\| \theta^T \begin{bmatrix} 1 \\ x^{(i)} \end{bmatrix} - y^{(i)} \right\|_2^2$$

Solution method for multiple output linear regression

- linear regression is nothing but LS since

$$
\begin{aligned}
mf(\theta) &= \sum_{i=1}^{m} \left\| \theta^T \begin{bmatrix} 1 \\ x^{(i)} \end{bmatrix} - y^{(i)} \right\|_2^2 \\
&= \left\| \begin{bmatrix} 1 & {x^{(1)}}^T & \cdots & 1 & {x^{(1)}}^T \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & {x^{(m)}}^T & \cdots & 1 & {x^{(m)}}^T \end{bmatrix} \tilde{\theta} - \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \right\|_2^2 \\
&= \|\tilde{X}\tilde{\theta} - y\|_2^2
\end{aligned}
$$

where $\tilde{X} \in \mathbf{R}^{m \times q(p+1)}$ and $\tilde{\theta} \in \mathbf{R}^{q(p+1)}$

- hence, the same method applies

Linear regression with constraints

- what if we have one constraint?

$$\begin{array}{ll} \text{minimize} & f(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T \begin{bmatrix} 1 \\ x^{(i)} \end{bmatrix} - y^{(i)}\right)^2 \\ \text{subject to} & \theta_1 \geq 0 \end{array}$$

- no analytic solution exists (with only one constraint) in general
- however, convex optimization algorithms solve it (almost) as easily as original problem
- but, now with *any* number of convex constraints

$$\begin{array}{ll} \text{minimize} & f(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T \begin{bmatrix} 1 \\ x^{(i)} \end{bmatrix} - y^{(i)}\right)^2 \\ \text{subject to} & g_i(\theta) \leq 0 \text{ for } i = 1, \ldots, l \\ & A\theta = b \end{array}$$

Linear regression with constraints

- what if we have one constraint?

$$\begin{array}{ll} \text{minimize} & f(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T \begin{bmatrix} 1 \\ x^{(i)} \end{bmatrix} - y^{(i)}\right)^2 \\ \text{subject to} & \theta_1 \geq 0 \end{array}$$

- no analytic solution exists (with only one constraint) in general
- however, convex optimization algorithms solve it (almost) as easily as original problem
- but, now with *any* number of convex constraints

$$\begin{array}{ll} \text{minimize} & f(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T \begin{bmatrix} 1 \\ x^{(i)} \end{bmatrix} - y^{(i)}\right)^2 \\ \text{subject to} & g_i(\theta) \leq 0 \text{ for } i = 1, \ldots, l \\ & A\theta = b \end{array}$$

Linear regression with constraints

- what if we have one constraint?

$$\begin{array}{ll} \text{minimize} & f(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(\theta^T \left[\begin{array}{c} 1 \\ x^{(i)} \end{array} \right] - y^{(i)} \right)^2 \\ \text{subject to} & \theta_1 \geq 0 \end{array}$$

- no analytic solution exists (with only one constraint) in general
- however, convex optimization algorithms solve it (almost) as easily as original problem
- but, now with *any* number of convex constraints

$$\begin{array}{ll} \text{minimize} & f(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(\theta^T \left[\begin{array}{c} 1 \\ x^{(i)} \end{array} \right] - y^{(i)} \right)^2 \\ \text{subject to} & g_i(\theta) \leq 0 \text{ for } i = 1, \ldots, l \\ & A\theta = b \end{array}$$

Linear regression with constraints

- what if we have one constraint?

$$\begin{array}{ll} \text{minimize} & f(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T \left[\begin{array}{c} 1 \\ x^{(i)} \end{array}\right] - y^{(i)}\right)^2 \\ \text{subject to} & \theta_1 \geq 0 \end{array}$$

- no analytic solution exists (with only one constraint) in general
- however, convex optimization algorithms solve it (almost) as easily as original problem
- but, now with *any* number of convex constraints

$$\begin{array}{ll} \text{minimize} & f(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T \left[\begin{array}{c} 1 \\ x^{(i)} \end{array}\right] - y^{(i)}\right)^2 \\ \text{subject to} & g_i(\theta) \leq 0 \text{ for } i = 1, \ldots, l \\ & A\theta = b \end{array}$$

Support vector machine

- problem definition:
  - given $x^{(i)} \in \mathbf{R}^p$: input data, and $y^{(i)} \in \{-1, 1\}$: output labels
  - find hyperplane which separates two different classes as distinctively as possible (in some measure)
- (typical) formulation:

$$
\begin{array}{ll}
\text{minimize} & \|a\|_2^2 + \gamma \sum_{i=1}^m u_i \\
\text{subject to} & y^{(i)}(a^T x^{(i)} + b) \geq 1 - u_i, \; i = 1, \ldots, m \\
& u \succeq 0
\end{array}
$$

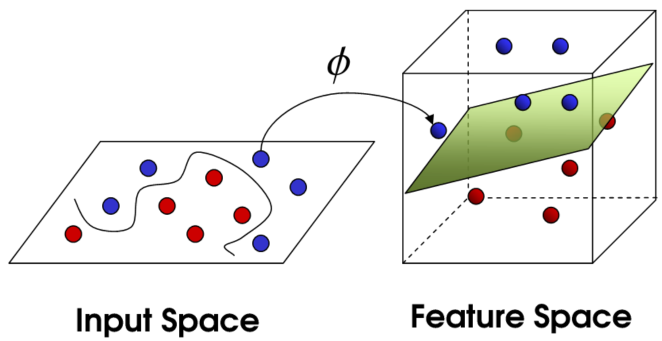
  - convex optimization problem, hence stable and efficient algorithms exist even for very large problems
  - has worked extremely well in practice (until... deep learning boom)

Support vector machine with kernels

- use feature transformation $\phi : \mathbf{R}^p \to \mathbf{R}^q$ (with $q > p$)
- formulation:

$$\begin{array}{ll} \text{minimize} & \|\tilde{a}\|_2^2 + \gamma \sum_{i=1}^m \tilde{u}_i \\ \text{subject to} & y^{(i)}(\tilde{a}^T \phi(x^{(i)}) + \tilde{b}) \geq 1 - \tilde{u}_i, \;\; i = 1, \ldots, m \\ & \tilde{u} \succeq 0 \end{array}$$

- still convex optimization problem



Different perspectives on machine learning

- statistical view: Frequentist or Bayesian?
- computer scientific perspective
- numerical algorithmic perspective
- performance acceleration using hardware parallelism with GPGPUs

Statistical perspective

- suppose data set $X_m = \{x^{(1)}, \ldots, x^{(m)}\}$
  - drawn independently from (true, but unknown) data generating distribution $p_{\text{data}}(x)$
- Maximum Likelihood Estimation (MLE) is to solve

$$\text{maximize} \quad p_{\text{data}}(X; \theta) = \prod_{i=1}^{m} p_{\text{data}}(x^{(i)}; \theta)$$

- equivalent, but numerically friendly formulation:

$$\text{maximize} \quad \log p_{\text{data}}(X; \theta) = \sum_{i=1}^{m} \log p_{\text{data}}(x^{(i)}; \theta)$$

Equivalence of MLE to KL divergence

- in information theory, Kullback-Leibler (KL) divergence defines distance between two probability distributions, p and q:

$$D_{\mathrm{KL}}(p\|q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)}\, dx$$

- KL divergence between data distribution, p_{data}, and model distribution, p_{model}, can be approximated by Monte Carlo method as

$$D_{\mathrm{KL}}(p_{\mathrm{data}}\|p_{\mathrm{model}}) \simeq \frac{1}{m}\sum_{i=1}^{m}(\log p_{\mathrm{data}}(x^{(i)}) - \log p_{\mathrm{model}}(x^{(i)};\theta))$$

- hence, *minimizing the KL divergence is equivalent to maximizing the log-likelihood!*

Equivalence of MLE to MSE

- assume the model is Gaussian, *i.e.*, $y \sim \mathcal{N}(g_\theta(x), \Sigma)$:

$$p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}^p |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}\left(y^{(i)} - g_\theta(x^{(i)})\right)^T \Sigma^{-1}\left(y^{(i)} - g_\theta(x^{(i)})\right)\right)$$

- assuming that $\Sigma = I_p$, the log-likelihood becomes

$$\sum_{i=1}^m \log p(y^{(i)}|x^{(i)};\theta) = -\sum_{i=1}^m \|y^{(i)} - g_\theta(x^{(i)})\|_2^2/2 - \frac{pm}{2}\log(2\pi)$$

- hence, *maximizing log-likelihood is equivalent to minimizing mean-square-error (MSE)!*

Other statistical factors

- overfitting problems
- training and test
- cross-validation
- regularization
- drop-out

Computer scientific perspectives

- neural network architectures
- hyper parameter optimization
- double/single precision representation
- low-power machine learning (especially for inference)

Numerical algorithmic perspectives

- basic formulation:

$$\text{minimize} \quad f(\theta) = \frac{1}{m} \sum_{i=1}^{m} l(g_\theta(x^{(i)}), y^{(i)})$$

- formulation with regularization:

$$\text{minimize} \quad f(\theta) = \frac{1}{m} \sum_{i=1}^{m} l(g_\theta(x^{(i)}), y^{(i)}) + \gamma r(\theta)$$

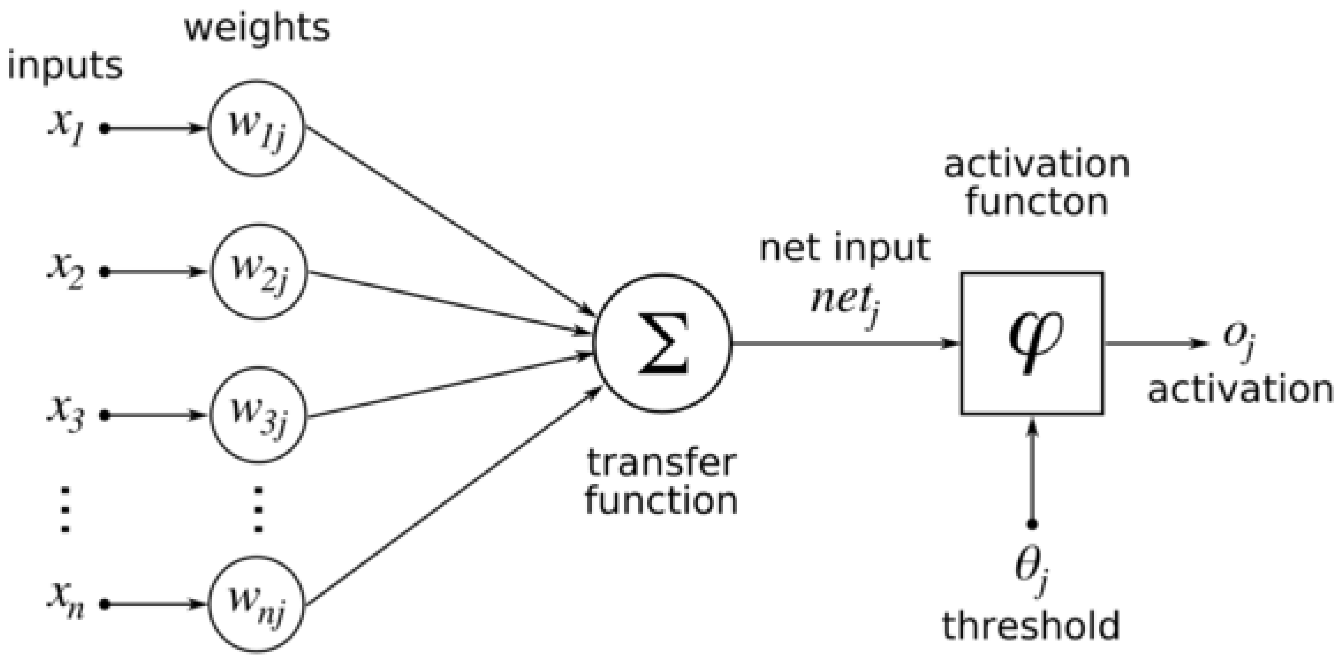
- stochastic gradient descent (SGD):

$$\theta^{(k+1)} = \theta^{(k)} - \alpha_k \nabla f(\theta)$$

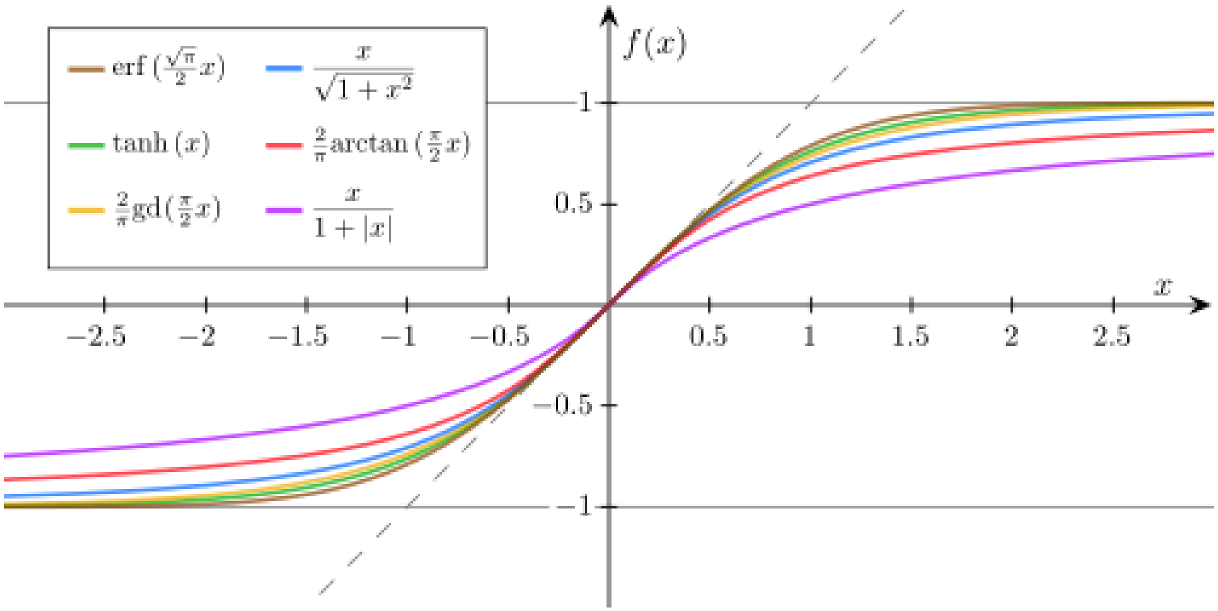
Backpropagation for training neural network?

- assuming that
  - the dimension of the feature space (or input space) is p
  - the dimension of the output space is q
  - a loss function $l : \mathbf{R}^q \times \mathbf{R}^q \to \mathbf{R}_+$
  - a neural network has d layers or it is of depth d
  - $z^{\{i\}} \in \mathbf{R}^{n_i}$ is the input to the perceptrons in the ith layer
  - $y^{\{i\}} \in \mathbf{R}^{n_i}$ is the output of the perceptrons in the ith layer
  - $W^{\{i\}} \in \mathbf{R}^{n_i \times n_{i-1}}$ is the weights of the connections between $i-1$th layer and ith layer
  - $w^{\{i\}} \in \mathbf{R}^{n_i \times n_{i-1}}$ is the bias weights for the ith layer
  - $\phi^{\{i\}} : \mathbf{R}^{n_i} \to \mathbf{R}^{n_i}$ represents the activation functions of the ith layer.

Basic unit comprising a general neural network



Activation function



Backpropagation for training neural network?

- modeling function for the (deep) neural network $g_\theta : \mathbf{R}^p \to \mathbf{R}^q$ defined by

$$g_\theta = \phi^{\{d\}} \circ \psi^{\{d\}} \circ \cdots \circ \phi^{\{1\}} \circ \psi^{\{1\}}$$

  or equivalently

$$g_\theta(x) = \phi^{\{d\}}(\psi^{\{d\}}(\cdots(\phi^{\{1\}}(\psi^{\{1\}}(x)))))$$

  for all $x \in \mathbf{R}^p$

- affine transmation $\psi^{\{i\}} : \mathbf{R}^{n_{i-1}} \to \mathbf{R}^{n_i}$ defined by

$$\psi^{\{i\}}(y^{\{i-1\}}) = W^{\{i\}} y^{\{i-1\}} + w^{\{i\}}.$$

Recall the chain rule from college calculus class

- if we have two functions $f : \mathbf{R}^n \to \mathbf{R}^m$ and $g : \mathbf{R}^m \to \mathbf{R}^p$, and the Jacobian matrices of f and g are $D_f : \mathbf{R}^n \to \mathbf{R}^{m\times n}$ and $D_g : \mathbf{R}^m \to \mathbf{R}^{p\times m}$ respectively, then the Jacobian matrix of $D_h : \mathbf{R}^n \to \mathbf{R}^{p\times n}$ of the composite function $h = g \circ f$ is

$$D_h(x) = D_g(f(x))D_f(x) \in \mathbf{R}^{p\times n}$$

- hence, if $p = 1$, we have

$$\nabla h(x) = D_f(x)^T \nabla g(f(x)) \in \mathbf{R}^n$$

Following math logics gives back propagation formula!

- assume that the cost function of the deep neural network is

$$f(\theta) = \frac{1}{m} \sum_{i=1}^{m} l(g_\theta(x^{(i)}), y^{(i)}).$$

- hence, the gradient is

$$m \nabla f(\theta) = \sum_{i=1}^{m} \nabla_\theta l(g_\theta(x^{(i)}), y^{(i)}) = \sum_{i=1}^{m} \nabla_\theta l(g_\theta(x^{(i)}), y^{(i)})$$

$$= \quad \sum_{i=1}^{m} D_\theta g_\theta(x^{(i)})^T \nabla_{y_1} l(g_\theta(x^{(i)}), y^{(i)})$$

$$= \sum_{i=1}^{m} \left(D_{\phi^{\{d\}}}(z^{\{d\}}) D_{\psi^{\{d\}}}(y^{\{d-1\}}) \cdots D_{\phi^{\{1\}}}(z^{\{1\}}) D_{\psi^{\{1\}}}(x^{(i)}) \right)^T \nabla_{y_1} l(g_\theta(x^{(i)}), y^{(i)})$$

$$= \sum_{i=1}^{m} D_{\psi^{\{1\}}}(x^{(i)})^T D_{\phi^{\{1\}}}(z^{\{1\}})^T \cdots D_{\psi^{\{d\}}}(y^{\{d-1\}})^T D_{\phi^{\{d\}}}(z^{\{d\}})^T \nabla_{y_1} l(g_\theta(x^{(i)}), y^{(i)})$$

(having assumed that $l(y_1, y_2) = \|y_1 - y_2\|_2^2$)

$$\nabla_\theta l(g_\theta(x^{(i)}), y^{(i)}) = 2 \begin{bmatrix} y_1^{\{d\}} - y_1^{(i)} \\ y_2^{\{d\}} - y_2^{(i)} \\ \vdots \\ y_q^{\{d\}} - y_q^{(i)} \end{bmatrix} \in \mathbf{R}^q,$$

$$D_{\psi^{\{i\}}}(y^{\{i-1\}})^T = {W^{\{i\}}}^T \in \mathbf{R}^{n_{i-1} \times n_i},$$

$$D_{\phi^{\{i\}}}(z^{\{i\}})^T = \begin{bmatrix} \frac{d}{dz}\phi_1^{\{i\}}(z_1^{\{i\}}) & 0 & \cdots & 0 \\ 0 & \frac{d}{dz}\phi_2^{\{i\}}(z_2^{\{i\}}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{d}{dz}\phi_{n_i}^{\{i\}}(z_{n_i}^{\{i\}}) \end{bmatrix} \in \mathbf{R}^{n_i \times n_i}.$$

Acceleration using hardware parallelism

- general-purpose computing on GPU (GPGPU)
  - maximizes parallelism for scientific computing
  - can fully utilize GPU-CPU framwork
  - is efficient for matrix multiplication, LU factorization, *etc.*
- history
  - becomes popular after 2001
  - two major APIs: OpenGL and DirectX
  - CUDA allowing users to ignore underlying graphical concepts
  - newer: Microsoft's DirectComputer, Apple/Khronos Group's OpenCL

Gifts II

- Google DeepMind's deep Q-learning to play a computer game

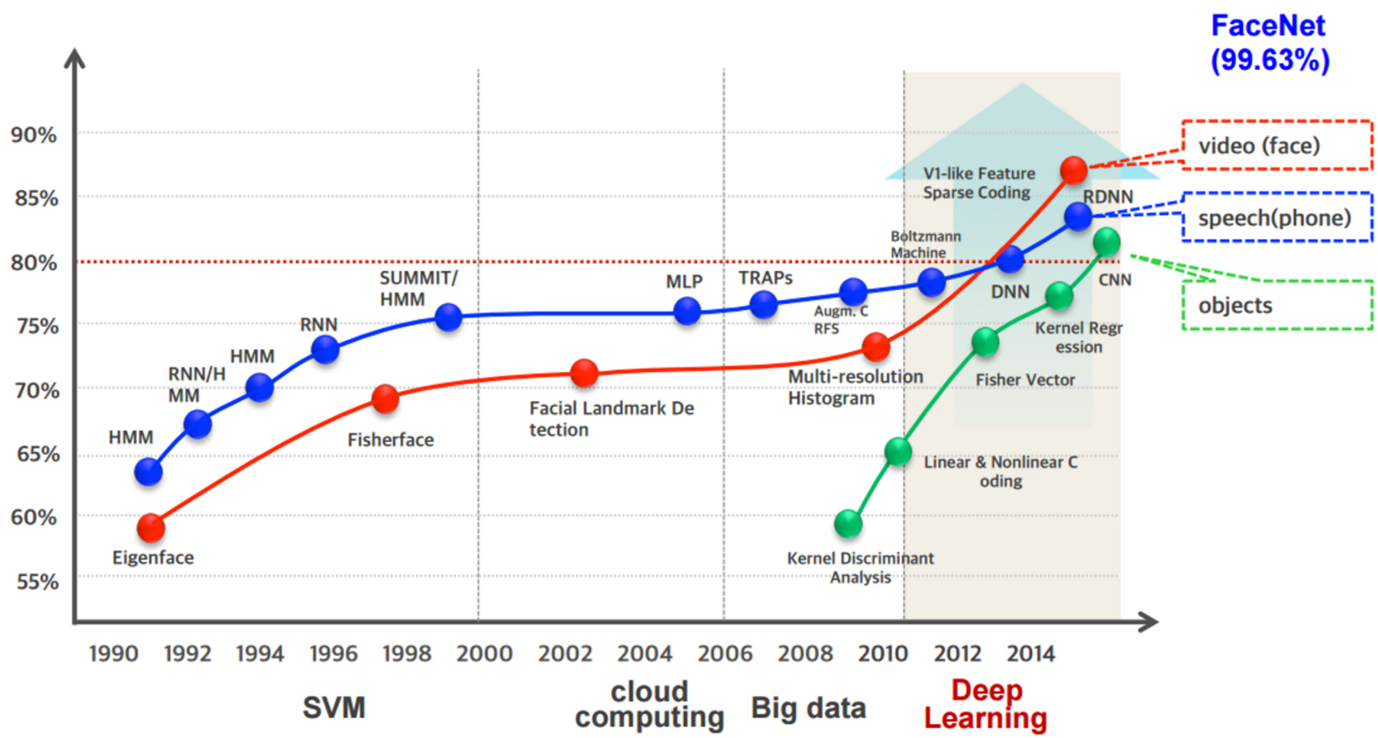
- Nvidia's self-driving technology demo @ CES 2017

What is deep learning (DL)?

- DL can be defined by
  - *deep* artificial neural network
- DL can be characterized by
  - many layers of processing for feature extraction and transformation
  - learning of multiple levels of features or representations of the data
  - learning representations of data
  - multiple levels corresponding to different levels of abstraction
- two interpretations
  - universal approximation theorem interpretation
  - probabilistic interpretation

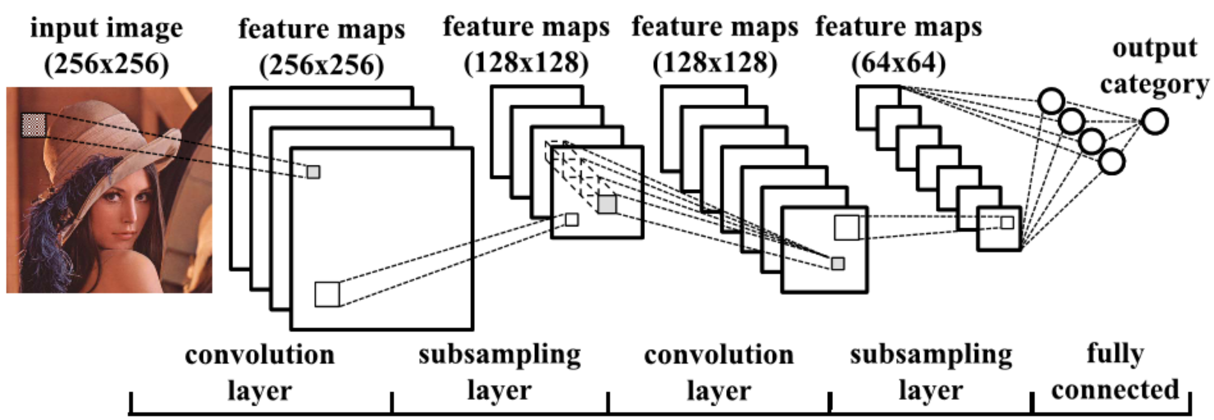
Recent advances in deep learning

- upheaval in pattern recognition due to deep learning (H. Choi)



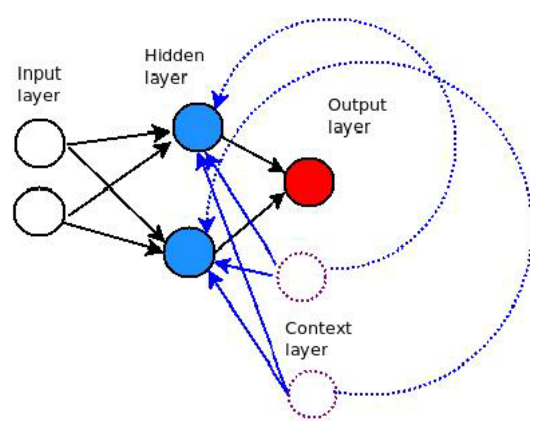
Convolutional neural network (CNN)

- CNN (or ConvNet) is
  - a type of feed-forward artificial neural network
  - inspired by animal visual cortex
- individual cortical neurons respond to restricted region of space
- applications in image and video recognition, recommender systems, and natural language processing



Recurrent neural network (RNN)

- RNN
  - is a class of artificial neural network where connections between units form a directed cycle
  - creates an internal state of the network which allows it to exhibit dynamic temporal behavior
- applicable to handwriting recognition or speech recognition
  - neural history compressor, long short-term memory



Special consideration: how to learn a deep neural net from rules

- create generative model: use rules to generate $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$
  - want to find function $g_\theta : \mathbf{R}^p \to \mathbf{R}^q$ with learning parameter, $\theta \in \mathbf{R}^n$, but this time, we want to use it for another purpose
  - define a loss function $l : \mathbf{R}^q \times \mathbf{R}^q \to \mathbf{R}_+$ for the purpose
- now do the usual, *i.e.*, learn a deep neural net using the set as training set
- how is this different from the rule-based approach?
  - what are the advantages?

AI Applications

- big data: medical, bio, finance
- auto industry: self-driving (or assisted driving) algorithm
- IoT: smart machines, smart algorithms
- securities

Image classification

- today's largest network
  - 10 layers, 1B parameters, 10M images
  - 30 exa flops
- human brain has trillions of parameters - only $1,000$ times more

Machine learning and security

- Is ML pipe dream of cybersecurity?
  - "there's no silver bullet in security."
- Is ML cybersecurity's answer to detecting advanced breaches?
  - it will shine as IT envinronments "grow increasingly complex."
- Will AI replace cybersecurity experts?

Machine learning and IoT

- IoT market will grow to $>$ \$1.7 trillion by 2020 with CAGR of 16.9%
  - purpose-built platforms, storage, networking, security
  - application software and service offerings
- # IoT connected devices (cars, refrigerators, . . .) will climb to 30 billion
- *e.g.*, General Electric, Philips, Ford Motor, Rio Tinto Group, and Stanley Black & Decker being a few of the companies with huge support from
  - companies like Dell, Hewlett Packard Enterprise, IBM, AT&T, Verizon Communications, Intel, ARM
  - small/startup companies than can be counted

(source: Forbes article)

Machine learning and medical applications

- demand: people increasingly interested in longer and healthier life
- technology
  - data from huge number of patients needed
  - size of DNA sequence huge!

Machine learning and bio applications

- origin: perceptron constituted an attempt to model actual neuronal behavior

- analysis of translation initiation sequences employed the perceptron to define criteria for start sites in Escherichia coli

- medical service, applications like emotion detection

Who will be the winner in the era of Deep Learning and AI?

- Google
- Amazon
- Nvidia
- Facebook, Twitter, LinkedIn, Uber, *etc.*.
- Samsung

Thank you!

Sunghee Yun (sunghee.yun@gmail.com)