[SKT AI Course: Deep Learning Basics]

Gentle Introduction to Machine Learning

Heung-Il Suk

hisuk@korea.ac.kr

http://www.ku-milab.org

Department of Brain and Cognitive Engineering,
Korea University

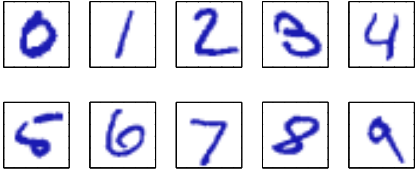
September 18, 2017

Contents

Machine Learning In a Nutshell

Example: Handwritten Digits

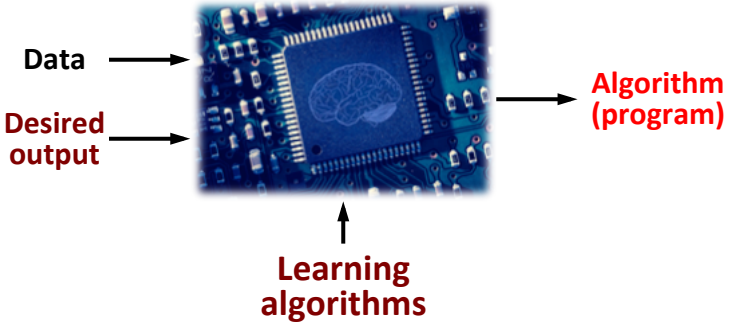
Goal: to build a machine that will produce the identity of the digit as the output



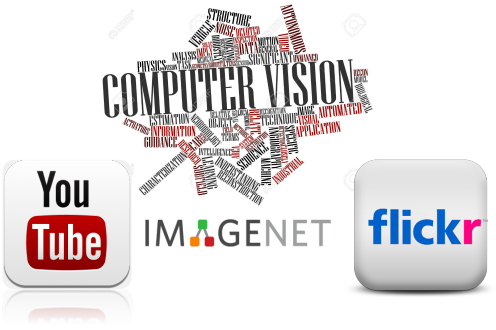
28×28 pixel image: 784 real numbers

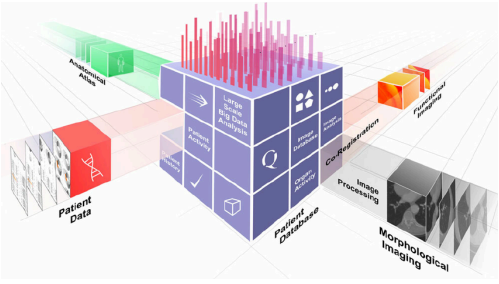
- Handcrafted rules or heuristics: shapes of the strokes
- Leads to a proliferation of rules, exceptions
- Nontrivial due to wide variability of handwriting

Traditional Programming vs. ML



- The more data we have, the more we can learn!!!
- No data? $\rightarrow$ nothing to learn
- big data? $\rightarrow$ lots to learn





Listen to data and let them speak mostly!!!

Why Machine Learning?

- Humans are unable to explain their expertise (speech/action recognition)
- Human expertise does not exist (navigation on Mars)
- Solution changes in time (routing on a computer network)
- Solution needs to be adapted to particular cases (user biometrics)
- etc.

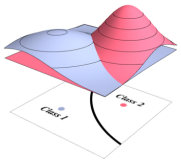
“Learning” in ML

- Induction: process of extracting general rules from a set of particular examples
- Learning: build computer models that can analyze data and extract information automatically from them
- Most cases, data are cheap and abundant but knowledge is expensive and scarce
- Build a model that is a good and useful approximation to the data

- Pattern recognition approach
  - Collect a large set of N digits, *training set*, $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$
  - Express the category of a digit using a *target vector* $\mathbf{t}$
  - Determine a function $f(\mathbf{x})$, *training* or *learning*
    - Generates an output vector $\mathbf{y}$, encoded in the same way as the target vector $\mathbf{t}$

$$\mathbf{x} \quad \Rightarrow \quad f(\mathbf{x};\theta) \quad \Rightarrow \quad \mathbf{y}$$

$$p(y|\mathbf{x};\mu,\Sigma)$$

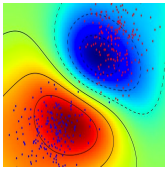


Bayes classifier

$$S_j^{\downarrow}\left[F_j\left[\sum_j w_{ij} * X_j\right]\right]$$

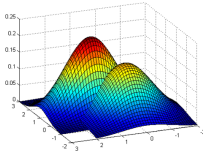
Convolutional neural network

$$\sum_{m \in S} a_m k(\mathbf{x}, \mathbf{x}_m) + b$$



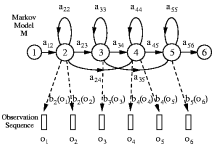
Support vector machine

$$\sum_{k=1}^{K} \pi_k N(\mathbf{x}|\mu_k, \Sigma_k)$$



Gaussian mixture model

$$p(S_t|S_{t-1})p(\mathbf{x}_t|S_t)$$



Hidden Markov model

Generalization: the ability to categorize correctly new examples that differ from those used for training

Feature Extraction/Representation

- To transform the original input variables into some new space of variables
- Hope to be easier to solve the problem in the new space
- To lessen computational burden (in real-time applications)
- Careful not to discard the useful discriminator information
- New test data must be preprocessed using the same steps as the training data

PRML Overview

Training session

1. Collecting training samples
2. Preprocessing
3. Feature extraction/representation
4. Feature selection
5. Classifier/regressor learning

Testing session

1. Given testing samples
2. Preprocessing
3. Feature extraction/representation
4. Feature selection
5. Outputs from classifier/regressor

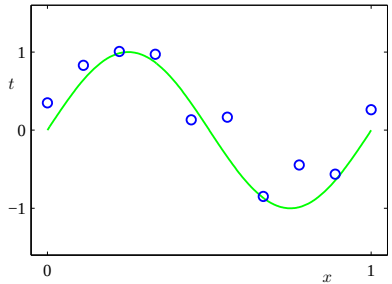
Terminology

- Supervised learning
  - Regression: continuous outputs
  - Classification: discrete or category outputs
- Unsupervised learning
  - Clustering
  - Density estimation
  - Visualization
- Reinforcement learning
  - Finding suitable actions to take in a given situation in order to maximize a reward
  - No optimal outputs are given, but must discover them by a process of trial and error

Machine Learning Overview

Example: Polynomial Curve Fitting

- N observations of $x \in \mathbb{R}$: $\mathbf{x} \equiv (x_1, \ldots, x_N)^\top$
- Corresponding target values of $t \in \mathbb{R}$: $\mathbf{t} \equiv (t_1, \ldots, t_N)^\top$
- Goal: to exploit the training set to predict value of $\hat{t}$ from x



10 samples generated from $\sin(2\pi x)$ by adding Gaussian noise

- Polynomial function

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

  - M: order of the polynomial

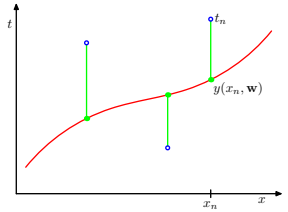
- Non-linear function of the input x
- Linear function of the coefficients $\mathbf{w} = [w_0, w_1, \ldots, w_M]^\top$

Linear model

[Error Function]

- Sum of squares of the errors between the predictions $y(x_n, \mathbf{w})$ for each data point x_n and target value t_n
  - Motivation for this choice of error function: discussed later

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}^2$$



- Solve the problem by choosing the value of $\mathbf{w}$ for which $E(\mathbf{w})$ is as small as possible

$$\min_{\mathbf{w}} E(\mathbf{w})$$

Optimization!!!

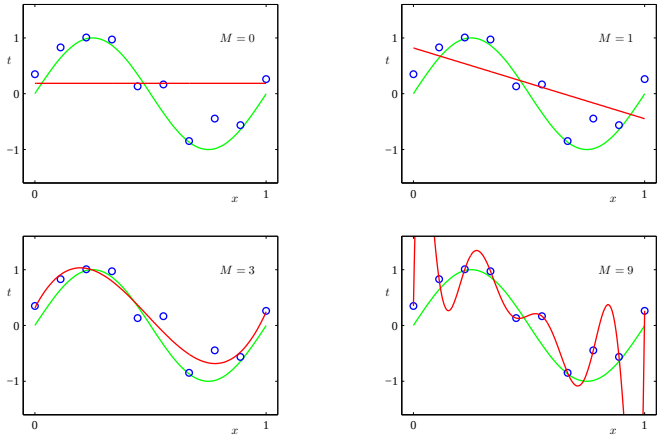
[Minimization of Error Function]

- Quadratic in coefficients $\mathbf{w}$
- Derivative w.r.t. coefficients will be linear in elements of $\mathbf{w}$
- Unique solution!!! $\mathbf{w}^{\star}$

$$
\begin{aligned}
E(\mathbf{w}) &= \frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\mathbf{w})-t_n\}^2 \\
y(x,\mathbf{w}) &= \sum_{j=0}^{M} w_j x^j \\
\frac{\partial E(\mathbf{w})}{\partial w_i} &= \sum_{n=1}^{N}\left\{\sum_{j=0}^{M} w_j x_n^j - t_n\right\} x_n^i = 0 \\
\sum_{n=1}^{N}\sum_{j=0}^{M} w_j x_n^{i+j} &= \sum_{n=1}^{N} t_n x_n^i
\end{aligned}
$$

[Choosing the Order of M]

- Model comparison or model selection



[Generalization Performance]

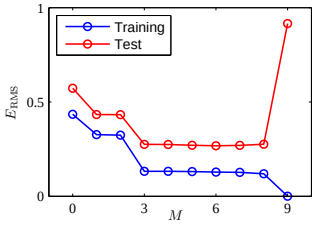
- Consider a separate test set of 100 points
- For each value of M, evaluate the error function for training data and test data

$$E\left(\mathbf{w}^{\star}\right)=\frac{1}{2}\sum_{n=1}^{N}\left\{y\left(x_n,\mathbf{w}^{\star}\right)-t_n\right\}^2$$

- Use Root-Mean-Square (RMS) error

$$E_{\mathrm{RMS}}=\sqrt{2E\left(\mathbf{w}^{\star}\right)/N}$$

  - Division by N allows different sizes of N to be compared on equal footing
  - Square root ensures E_{RMS} is measured in same units as t



Paradoxical !?

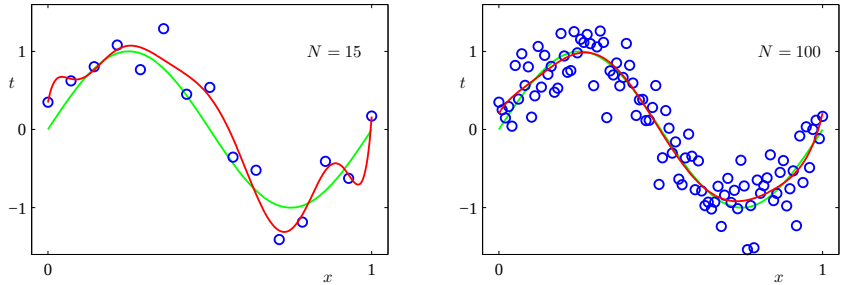
Table of the coefficients $\mathbf{w}^\star$ for polynomials of various order. Observe how the typical magnitude of the coefficients increases dramatically as the order of the polynomial increases.

	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

- As M increases, magnitude of coefficients increases
- At $M = 9$ finely tuned to random noise in target values

More flexible polynomials with large values of M are
becoming increasingly tuned to the random noise on the target values.

[Increasing Data Set Size]



- For a given model complexity, overfitting problem is less severe as the size of a data set increases
- The larger the data set, the more complex we can afford to fit the data
- (Heuristic) The number of data points should be no less than some multiple (say 5 or 10) of the number of adaptive parameters in the model.

[Regularization]

- Using relatively complex models with data sets of limited size

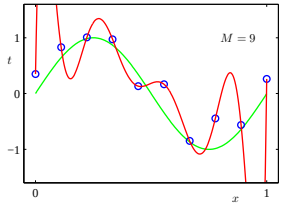
Table of the coefficients $\mathbf{w}^\star$ for polynomials of various order. Observe how the typical magnitude of the coefficients increases dramatically as the order of the polynomial increases.

	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

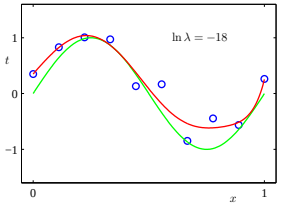
- Add a penalty term to error function to discourage coefficients from reaching large values

$$\tilde{E}(\mathbf{w}) = \underbrace{\frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2}_{\text{goodness-of-fit}} + \underbrace{\frac{\lambda}{2}\|\mathbf{w}\|^2}_{\text{penalty}}$$

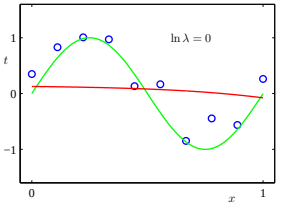
  - λ: governing the relative importance of the regularization term
  - Can be minimized exactly in a closed form
  - a.k.a. 'shrinkage' in statistics, 'weight decay' in neural networks



No regularization ($\lambda = 0$)



Optimal



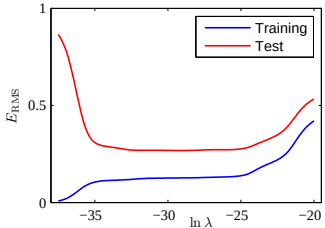
Too large

Table of the coefficients $\mathbf{w}^\star$ for $M = 9$ polynomials with various values for the regularization parameter λ. Note that $\ln \lambda = -\infty$ corresponds to a model with no regularization, i.e., to the graph at the bottom right in Figure 1.4. We see that, as the value of λ increases, the typical magnitude of the coefficients gets smaller.

	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

Impact of Regularization

- λ: controls the complexity of the model and hence degree of overfitting
  - Analogous to the choice of M
- Suggestion: partition data into two sets
  - Training set: to determine coefficients $\mathbf{w}$
  - Validation set: to optimize model complexity (M or λ)
  - BUT, too wasteful of valuable training data; need to seek more sophisticated approaches



RMS error vs. $\ln \lambda$ for $M = 9$

Interim Summary

Given *i.i.d.* samples $\mathcal{D} = \{\mathbf{x}_n, y_n\}_{n=1}^{N}$,
the aim is to build a good and useful approximation to y_n

$$\mathbf{x} \quad \Rightarrow \quad f\left(\mathbf{x}|\theta\right) \quad \Rightarrow \quad y$$

1. Model: $f\left(\mathbf{x}_n|\theta\right) \in \mathcal{F} \rightarrow$ capacity
2. Loss function: $J(\theta|\mathcal{D}) \rightarrow$ sufficient # of data, regularization
3. Learning: $\theta^* = \underset{\theta}{\operatorname{argmin}}\, J(\theta|\mathcal{D}) \rightarrow$ optimization

Linear Basis Function Models

Linear Basis Function Models

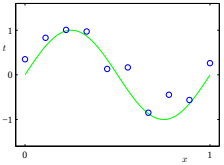
- Linear regression: simplest model for regression
  - Linear combination of input variables

$$y(\mathbf{x}, \mathbf{w}) = \sum_{d=1}^{D} w_d x_d + w_0$$

  - Limited as practical techniques for pattern recognition (e.g., high dimensionality)
  - Nice analytical properties; foundation for more sophisticated models

- More useful form: polynomial curve fitting

$$y(x, \mathbf{w}) = \sum_{j=1}^{M-1} w_j x^j + w_0$$



- Linear combination of non-linear functions of input variables $\phi(\mathbf{x})$, called '*basis functions*'

$$\begin{aligned} y(x, \mathbf{w}) &= \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x}) + w_0 = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) \\ &= \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}) \end{aligned}$$

where $\mathbf{w} = [w_0, w_1, \ldots, w_{M-1}], \boldsymbol{\phi}(\mathbf{x}) = [\phi_0 = 1, \phi_1, \ldots, \phi_{M-1}]$

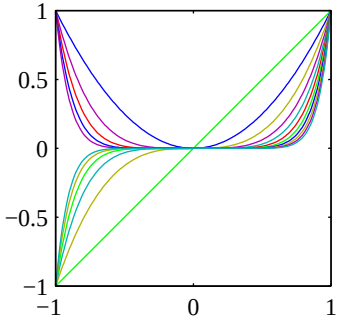
  - $\boldsymbol{\phi}(\mathbf{x})$: fixed preprocessing or feature extraction

Linear functions of parameters (still analytic); Yet, non-linear with respect to the input variables

(Recap.) polynomial curve fitting

$$y(x, \mathbf{w}) = \sum_{j=1}^{M-1} w_j x^j + w_0$$

- Global function of the input variables: changes in one region of input space affect all other regions
- Difficult to formulate: number of polynomials/coefficients increases exponentially with M

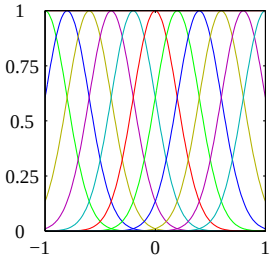


Divide the input space into regions and use different polynomials in each region!!!

Other Basis Functions

- (Gaussian) Radial Basis Functions (RBF)

$$\phi_j(x) = \exp\left\{\frac{(x-\mu_j)^2}{2s^2}\right\}$$



  - μ_j: governing the locations of the basis functions in input space
    - Can be arbitrary points in the data
  - s: governing the spatial scale
    - Can be chosen from the data set, *e.g.*, average variance

- Not required to have a probabilistic interpretation (normalization term is unimportant)

- Sigmoidal Basis Function

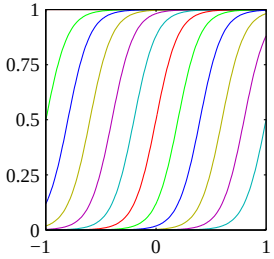
$$\phi_j(x) = \sigma\left(\frac{x - \mu_j}{s}\right)$$

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

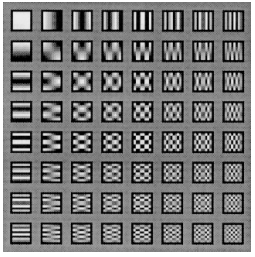
- Equivalently, 'tanh' function, which is related to the logistic sigmoid

$$\tanh(a) = 2\sigma(a) - 1$$

  - A general linear combination of logistic sigmoid functions is equivalent to a general linear combination of 'tanh' functions.

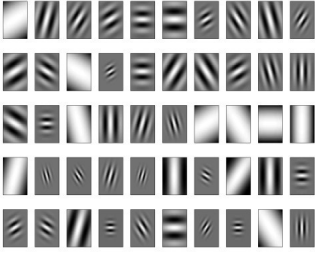


- Fourier
  - Expansion in sinusoidal functions
  - Infinite spatial extent



e.g., DCT Fourier basis

- Wavelet
  - Localized in both space and frequency
  - Useful for lattices such as images and time series



e.g., Gabor wavelet basis

Regularized Least Squares

Regularization term in order to control overfitting

- Error function to minimize

$$E(\mathbf{w}) = E_D(\mathbf{w}) + \lambda E_{\mathbf{w}}(\mathbf{w})$$

  - λ (regularization coefficient): controls relative importance of a data-dependent error $E_D(\mathbf{w})$ and a regularization term $E_{\mathbf{w}}(\mathbf{w})$

- "Quadratic" regularizer

$$E(\mathbf{w}) = \underbrace{\frac{1}{2}\sum_{n=1}^{N}\left\{t_n - \mathbf{w}^\top \phi(\mathbf{x}_n)\right\}^2}_{E_D(\mathbf{w})} + \lambda \underbrace{\frac{1}{2}\mathbf{w}^\top \mathbf{w}}_{E_{\mathbf{w}}(\mathbf{w})}$$

  - a.k.a., '*weight decay*': in sequential learning, encouraging weight values to decay towards zero, unless supported by data
  - Error function remains a quadratic function of $\mathbf{w} \rightarrow$ exact minimizer can be found in a closed form

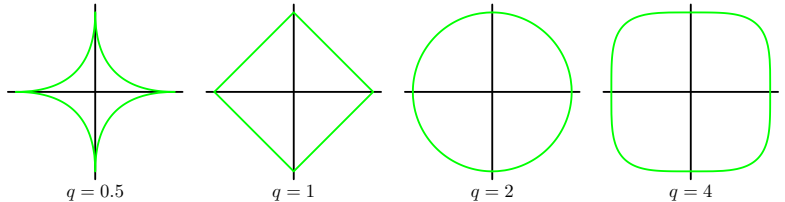
- Taking derivative w.r.t. $\mathbf{w}$ and setting to zero

$$\nabla_{\mathbf{w}}^{R} \ln p(\mathbf{t}|\mathbf{w}, \beta) \Rightarrow \mathbf{w}_{ML}^{R} = \left(\Phi^\top \Phi + \lambda I\right)^{-1} \Phi^\top \mathbf{t}$$

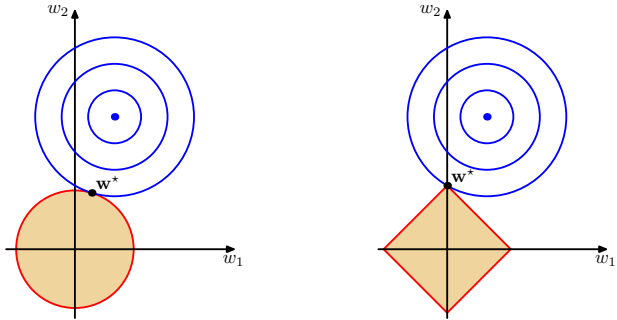
[Generalization of Quadratic Regularizer]

$$E(\mathbf{w}) = \underbrace{\frac{1}{2}\sum_{n=1}^{N}\left\{t_n - \mathbf{w}^\top \phi(\mathbf{x}_n)\right\}^2}_{E_D(\mathbf{w})} + \underbrace{\frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^q}_{E_{\mathbf{w}}(\mathbf{w})}$$

- $q = 2$: quadratic regularizer, *ridge* regressor
- $q = 1$: known as *LASSO* (Least Absolute Shrinkage and Selection Operator)
  - If λ is sufficiently large, some of the coefficients w_j are driven to zero, leading to a *sparse* model in which the corresponding basis functions play no role



$\sqrt{w_1} + \sqrt{w_2} = \text{const}$; $|w_1| + |w_2| = \text{const}$; $w_1^2 + w_2^2 = \text{const}$; $w_1^4 + w_2^4 = \text{const}$



Plot of the contours of the unregularized error function (blue)
along with the constraint region for $q = 2$ (left) and $q = 1$ (right),
in which the optimum value for the parameter vector $\mathbf{w}$ is denoted by $\mathbf{w}^*$.

Minimizing

$$\frac{1}{2}\sum_{n=1}^{N}\left\{t_n - \mathbf{w}^\top \phi\left(\mathbf{x}_n\right)\right\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^q$$

equivalent to minimizing

$$\frac{1}{2}\sum_{n=1}^{N}\left\{t_n - \mathbf{w}^\top \phi\left(\mathbf{x}_n\right)\right\}^2 \text{ s.t. } \sum_{j=1}^{M}|w_j|^q \leq \eta$$

- can be related using Lagrange multipliers
- $\lambda \uparrow \Rightarrow$ an increasing number of parameters $\rightarrow 0$

Regularization

- Allows complex models to be trained on small data sets without severe overfitting
- Limits model complexity, *i.e.*, how many basis functions to use

- The problem of determining the optimal model complexity is shifted from one of finding the appropriate number of basis functions to one of determining suitable value of the regularization coefficient λ

From Linear Regression to Linear Classification

- Linear regression model $y(\mathbf{x}, \mathbf{w})$

$$y(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + w_0$$

- For classification, we wish to obtain a discrete output or posterior probabilities in a range $(0, 1)$

$$y(\mathbf{x}) = f\left(\mathbf{w}^\top \phi(\mathbf{x}) + w_0\right)$$

Generalized Linear Model

$$y(\mathbf{x}) = f\left(\mathbf{w}^{\top}\phi(\mathbf{x}) + w_0\right)$$

- $f(\cdot)$: nonlinear, known as the activation function
- Decision surfaces
  - $y(\mathbf{x}) =$ constant or $\mathbf{w}^{\top}\phi(\mathbf{x}) + w_0 =$ constant
- Decision surfaces are linear functions of $\mathbf{x}$ even if $f(\cdot)$ is nonlinear (generalized linear model)

  [McCullagh and Nelder, 1989]
- Nonlinear in the parameter space $\mathbf{w}$ due to the nonlinear function $f(\cdot)$
  - Leads to more complex models for classification than regression

(Two-Class) Logistic Regression

For a dataset $\{\phi_n = \phi(\mathbf{x}_n), t_n\}$, where $t_n \in \{0, 1\}$ with $n = 1, \ldots, N$

- Likelihood function

$$p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N} y_n^{t_n} \{1 - y_n\}^{1-t_n}$$

  - $y_n = P(C_1|\phi(\mathbf{x}_n)) = \text{sigmoid}(a_n)$, where $a_n = \mathbf{w}^\top \phi(\mathbf{x}_n)$
  - $\mathbf{t} = (t_1, \ldots, t_N)^\top$

- Taking negative logarithm $\rightarrow$ *cross-entropy error function*

$$E(\mathbf{w}) = -\ln p(\mathbf{t}|\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

(Multi-Class) Logistic Regression

Work with a softmax function instead of logistic sigmoid

$$P\left(C_k|\mathbf{x}\right) = y_k\left(\phi\right) = \frac{\exp\left(a_k\right)}{\sum_j \exp\left(a_j\right)} \quad \text{where } a_k = \mathbf{w}_k^\top \phi$$

- Likelihood function

$$p\left(\mathbf{T}|\mathbf{w}_1, \ldots, \mathbf{w}_K\right) = \prod_{n=1}^{N}\prod_{k=1}^{K} P\left(C_k|\mathbf{x}_n\right)^{t_{nk}} = \prod_{n=1}^{N}\prod_{k=1}^{K} y_{nk}^{t_{nk}}$$

  - $\mathbf{t}_n \in \{0,1\}^K$: 1-of-$K$ coding scheme
  - $y_{nk} = y_k\left(\phi_n\right)$, $\mathbf{T} = [t_{nk}]$: $N \times K$ matrix of target variables

- Taking negative logarithm $\rightarrow$ *cross-entropy error function*

$$E\left(\mathbf{w}_1, \ldots, \mathbf{w}_K\right) = -\ln p\left(\mathbf{T}|\mathbf{w}_1, \ldots, \mathbf{w}_K\right) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_{nk}$$

Gradient-based Optimization

Overview

$$\min_{\mathbf{x}} f(\mathbf{x})$$

1. Starting point $\mathbf{x}_0$
2. **Select a search direction $\mathbf{s}_i$ (★)**
3. Determine the step size η_i for movement along the search direction $\mathbf{s}_i$
4. Set a new point $\mathbf{x}_i = \mathbf{x}_{i-1} + \eta_i \mathbf{s}_i$
5. Check convergence: if not converged, go to step 2

Directional Derivative in Direction s

$$f(\mathbf{x}+\eta\mathbf{s}) = \nabla_{\mathbf{x}} f(\mathbf{x})^T \mathbf{s}$$

$$\left(\because \frac{df}{d\eta} = \sum \left(\frac{\partial f}{\partial x_i}\right)\left(\frac{dx_i}{d\eta}\right) = \nabla_{\mathbf{x}} f(\mathbf{x})^T \mathbf{s}\right)$$

- Change in the function for a small step in the direction $\mathbf{s}$

$$\Delta f(\mathbf{x}) \approx \frac{df}{d\eta}\Delta\lambda$$

  - Representing the expected change in the function for a small step in the direction $\mathbf{s}$
- When locating the minimum exactly

$$\left.\frac{df}{d\eta}\right|_{\eta=\eta^*} = \nabla_{\mathbf{x}} f(\mathbf{x})^T \mathbf{s} = 0$$

To minimize f, we would like to find the direction in which f decreases the fastest

$$\min_{\mathbf{s},\mathbf{s}^T\mathbf{s}=1} \mathbf{s}^T \nabla_{\mathbf{x}} f(\mathbf{x}) = \min_{\mathbf{s},\mathbf{s}^T\mathbf{s}=1} \|\mathbf{s}\|_2 \cdot \|\nabla_{\mathbf{x}} f(\mathbf{x})\|_2 \cos\theta$$

$$\Rightarrow \min_{\mathbf{s}} \cos\theta$$

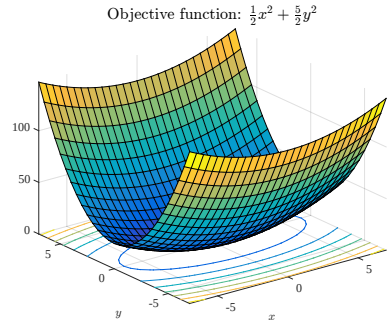
- minimum when $\mathbf{s}$ points in the opposite direction as the gradient
- *a.k.a.*, steepest descent or gradient descent

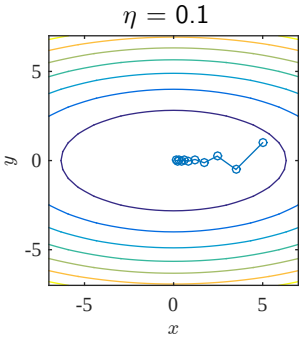
Steepest Gradient Descent

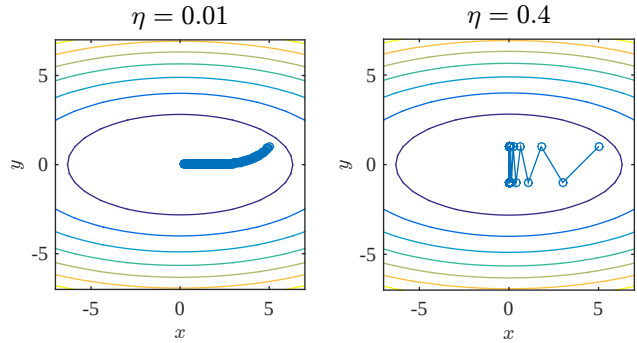
- One of the simplest unconstrained optimization methods
- Given an initial starting point, moves downhill until it can go no further.

$$\mathbf{x}^{\text{new}} = \mathbf{x} - \eta \nabla_{\mathbf{x}} f(\mathbf{x})$$

  - $-\nabla_{\mathbf{x}} f(\mathbf{x})$: search direction
  - η: stepsize, learning rate
    - How to set η: line search method, fixed value (*e.g.*, 10^{-2})


Objective function: $\frac{1}{2}x^2 + \frac{5}{2}y^2$
100
50
0
5
0
-5
y
-5
0
5
x


$\eta = 0.1$
5
0
-5
y
-5
0
5
x



- Small η: long convergence time
- Large η: oscillations and even divergence

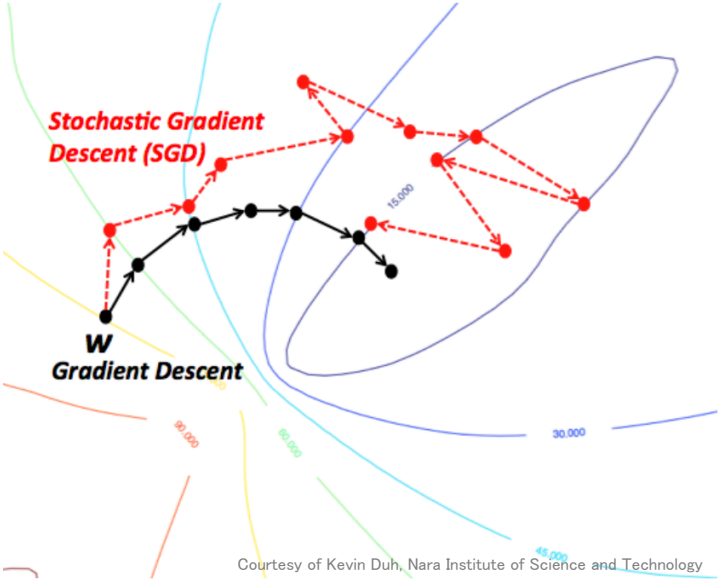
Batch vs. Stochastic (Online)

(Vanilla) Batch

1. Initialize $\mathbf{w}$
2. Compute $\nabla_{\mathbf{w}} f(D) = \sum_n \nabla_{\mathbf{w}} f(\mathbf{x}_n)$
3. Update $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} f(D)$
4. Repeat steps 2-3 until convergence

Stochastic Gradient Descent (SGD)

1. Initialize $\mathbf{w}$
2. Shuffle data $\mathcal{D}$
3. For each sample in $\{\mathbf{x}_n\}_{n=1}^N$
   - Compute $\nabla_{\mathbf{w}} f(\mathbf{x}_n)$
   - Update $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} f(\mathbf{x}_n)$
4. Repeat step 2 until convergence



Courtesy of Kevin Duh, Nara Institute of Science and Technology

- Batch learning goes in the direction of steepest descent
  - Slower to compute per iteration for a large dataset

- Stochastic can be considered as noisy descent
  - when stuck in a local optimum, a possibility of getting out of it
  - large η: update depends much on the recent instances (short memory)

Good tradeoff: *mini-batch stochastic gradient descent*
(a bunch of samples at a time)

1. Initialize $\mathbf{w}$
2. Shuffle data $\mathcal{D}$
3. For batch $\mathcal{B}$ in $\mathcal{D}$
   - Compute $\nabla_{\mathbf{w}} f(\mathcal{B}) = \sum_{\mathbf{x}_n \in \mathcal{B}} \nabla_{\mathbf{w}} f(\mathbf{x}_n)$
   - Update $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} f(\mathcal{B})$
4. Repeat step 2 until convergence

Pros

- Always goes downhill, *i.e.*, reducing the function value
- Guaranteed to converge to a local optimum when enough steps are taken
- Simple to implement

Cons

- Very slow convergence on elongated functions

Hessian Matrix

$$\mathbf{H}(\mathbf{x}) = \nabla^2 f(\mathbf{x}) = \nabla \left(\nabla f(\mathbf{x})^T \right)$$

$$H_{ij} = \frac{\partial^2}{\partial x_i \partial x_j} f(\mathbf{x})$$

- Anywhere that second derivatives are continuous, the differential operators are commutative

$$H_{ij} = \frac{\partial^2}{\partial x_i \partial x_j} f(\mathbf{x}) = \frac{\partial^2}{\partial x_j \partial x_i} f(\mathbf{x}) = H_{ji} \quad (\mathbf{H} : \text{symmetric})$$

Gives us information about the curvature of a function
and tells us how the gradient is changing

Take Home Messages

- Regularized linear/logistic regression
  - Basis function for non-linear modeling
- Steepest gradient descent
  - Batch / Mini-batch / Stochastic

Thank you for your attention!!!

(Q & A)

hisuk (AT) korea.ac.kr

http://www.ku-milab.org

Machine Intelligence Lab