智能感知与计算研究中心
Center for Research on Intelligent Perception and Computing

中国科学院自动化研究所
Institute of Automation
Chinese Academy of Sciences

2018

Lecture 2：Deep Learning Basics

王威

Center for Research on Intelligent Perception and Computing (CRIPAC)
National Laboratory of Pattern Recognition (NLPR)
Institute of Automation, Chinese Academy of Science (CASIA)

Outline

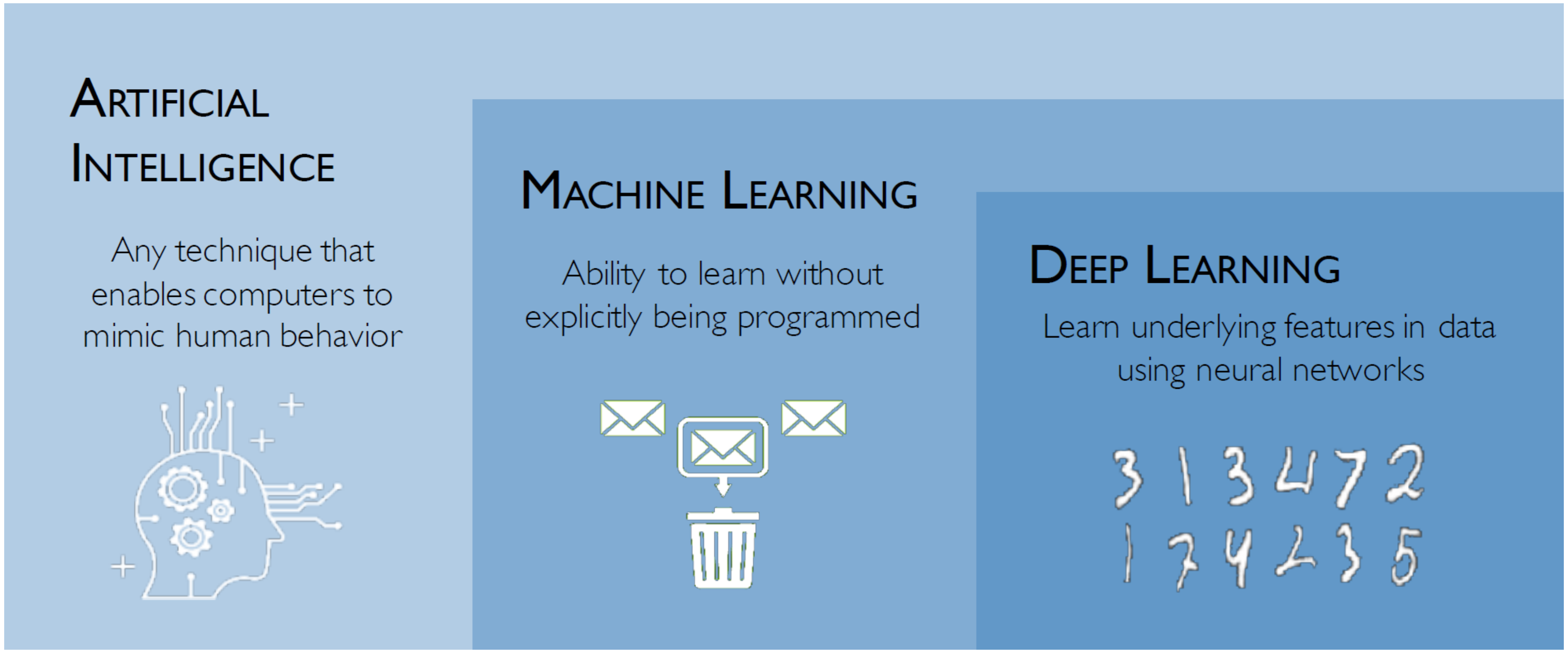
Review: Course Goals

- Give you an opportunity for you to explore an interesting multivariate analysis problem of your choice in the context of a real-world data set
- How deep learning works
- How to frame tasks into learning problems
- How to use toolkits to implement designed models
- When and why specific deep learning techniques work for specific problems

Review: Course Logistics

- Course: Theory (3 hours per week) + Course project
- The location of course room: 教1-101
- Final grade = 50% from the group project + 50% from the final exam
- The group project contains a technical report and project presentation
- The project presentation is decided by random sampling due to time limit
- Each team is composed of 5-10 members

Review: Deep Learning



ARTIFICIAL INTELLIGENCE

Any technique that enables computers to mimic human behavior

MACHINE LEARNING

Ability to learn without explicitly being programmed

DEEP LEARNING

Learn underlying features in data using neural networks

Review: Deep Neural Networks (DNN)

- **Originate from**:
  - 1962 – simple/complex cell, *Hubel and Wiesel*
  - 1970 – efficient error backpropagation, *Linnainmaa*
  - 1979 – deep neocognitron, convolution, *Fukushima*
  - 1987 – autoencoder, *Ballard*
  - 1989 – convolutional neural networks (CNN), *Lecun*
  - 1991 – deep recurrent neural networks (RNN), *Schmidhuber*
  - 1997 – long short-term memory (LSTM), *Schmidhuber*

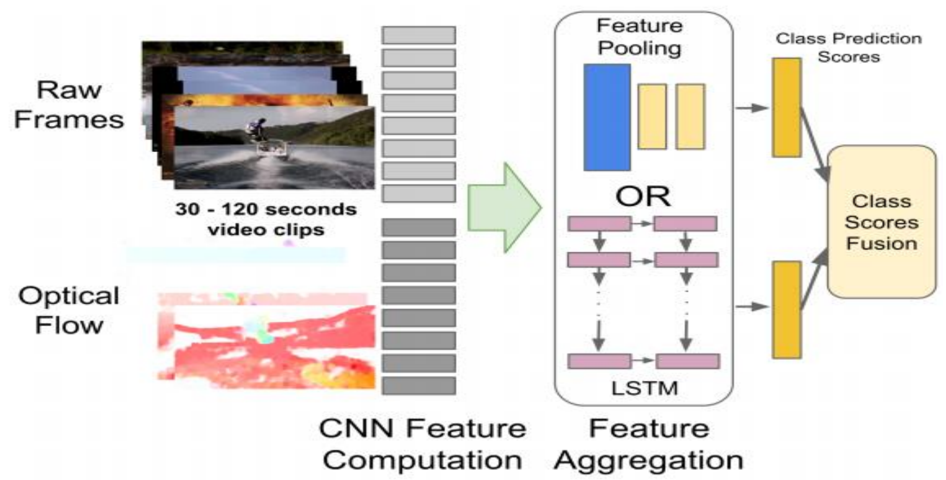
- **Two drawbacks:**

Large numbers of parameters → High computational cost

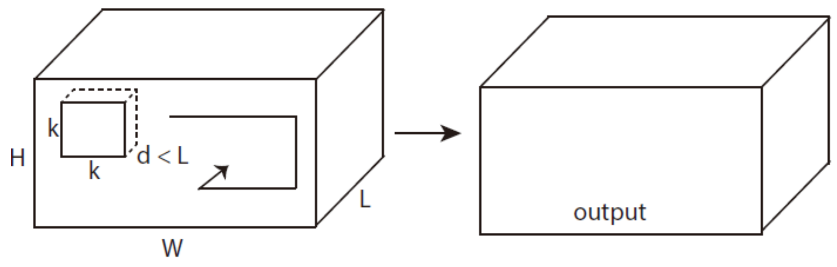
Small training set → Over-fitting problem

Review: Application for Video Classification

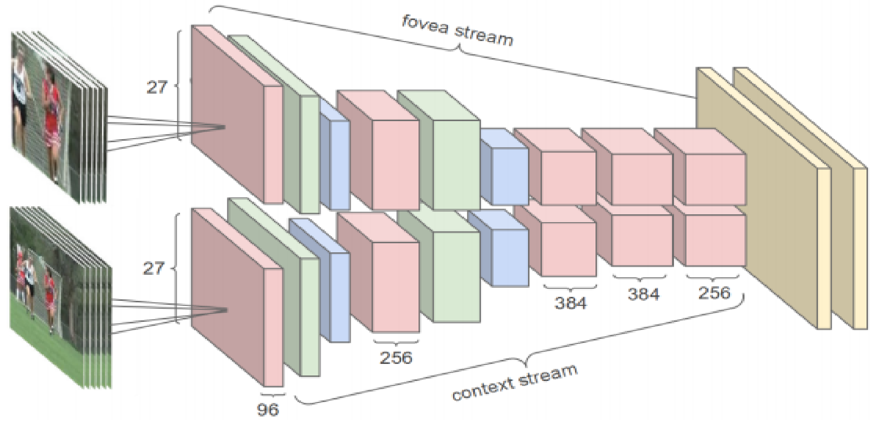
CNN: convolutional neural networks; RNN: recurrent neural networks



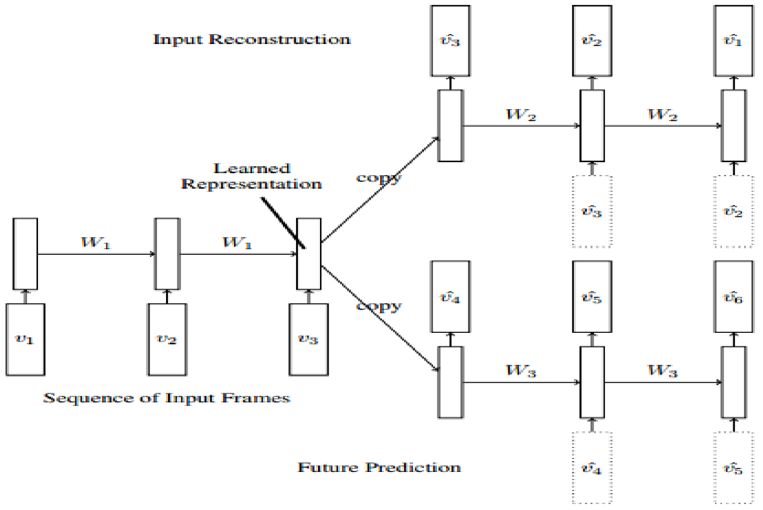
Long-range Videos Modelling, **CVPR2015**



3D Convolution, **ICCV2015**

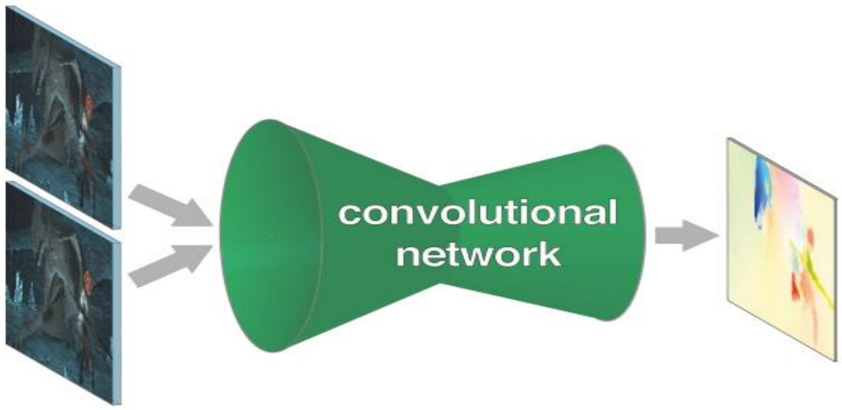


Multi-resolution CNN, **CVPR2014**

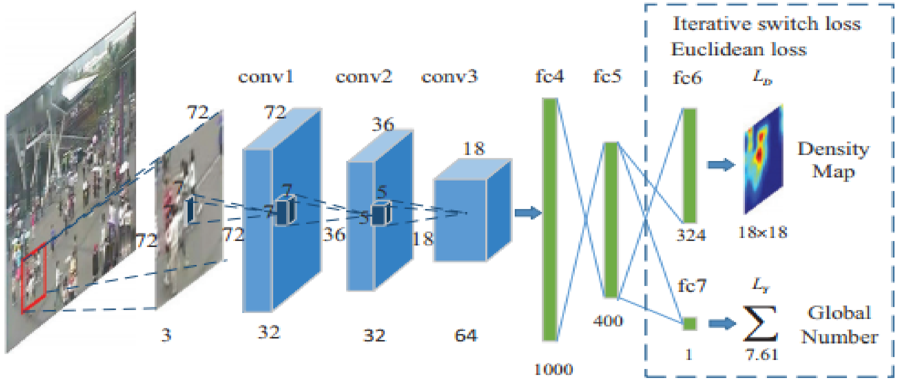


Autoencoder-RNN, **ICML2015**

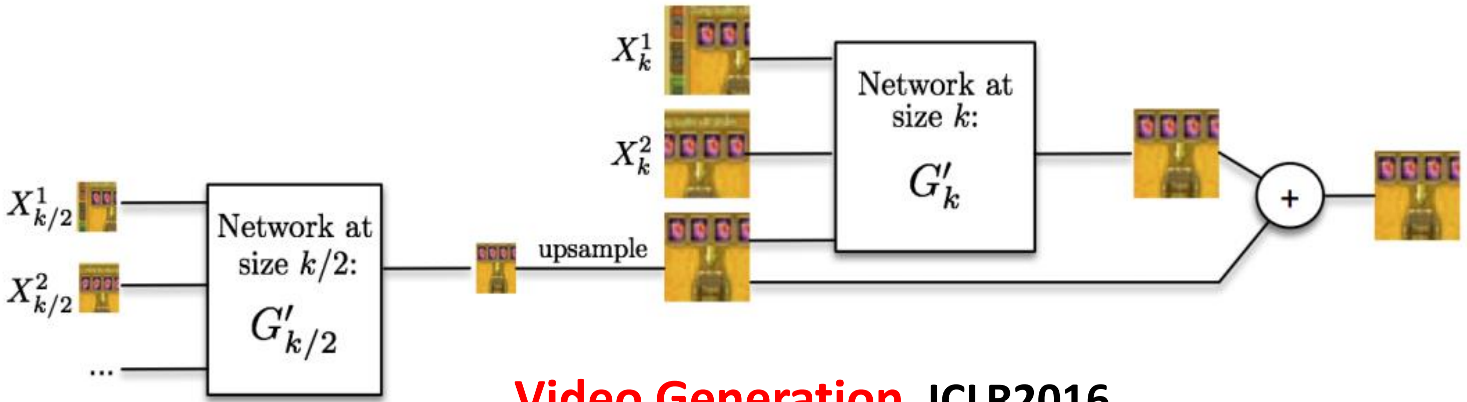
Review: Applications for Others



Optical Flow Prediction, ICCV2015



Cross-scene Crowd Counting, CVPR2015


X_k^1
X_k^2
$X_{k/2}^1$
$X_{k/2}^2$
...
Network at size $k/2$: $G'_{k/2}$
upsample
Network at size k: G'_k
+

Video Generation, ICLR2016

Achieve state-of-the-art results in various applications

Review: Future Directions

- Large scale deep learning
  - multiple GPUs
  - distributed system
- Unsupervised (semi-supervised) Learning
  - lack of supervised information in real applications
- Multimodal learning
  - videos are closely related to other data modalities, e.g., text, audio
- Brain-inspired deep models
  - conventional neural networks are inspired by human cognitive mechanism

Outline

1. Course Review
2. Linear Algebra
3. Probability and Information Theory
4. Numerical Computation for DL
5. Machine Learning Basics

About this part

- Not a comprehensive survey of all of linear algebra
- Focused on the subset most relevant to deep learning
- Larger subset: e.g., Linear Algebra by Georgi Shilov

Scalars

- A scalar is a single number
- Integers, real numbers, rational numbers, etc.
- We denote it with italic font:

$$a, n, x$$

Vectors

- A vector is a 1-D array of numbers:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

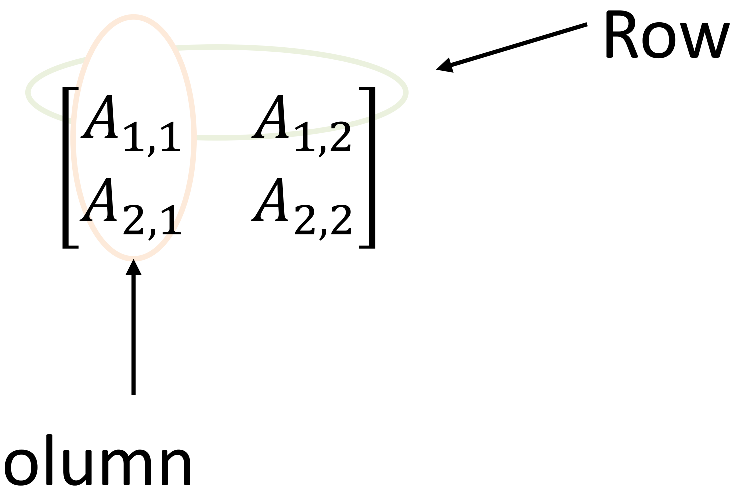
- Can be real, binary, integer, etc.

- Example notation for type and size:

$$\mathbb{R}^n$$

Matrices

- A matrix is a 2-D array of numbers:



$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}$$

- Example notation for type and shape:

$$A \in \mathbb{R}^{m \times n}$$

Tensors

- A tensor is an array of numbers, that may have
  - zero dimensions, and be a scalar
  - one dimension, and be a vector
  - two dimensions, and be a matrix
  - or more dimensions.

Matrix Transpose

$$(A^T)_{i,j} = A_{j,i}$$

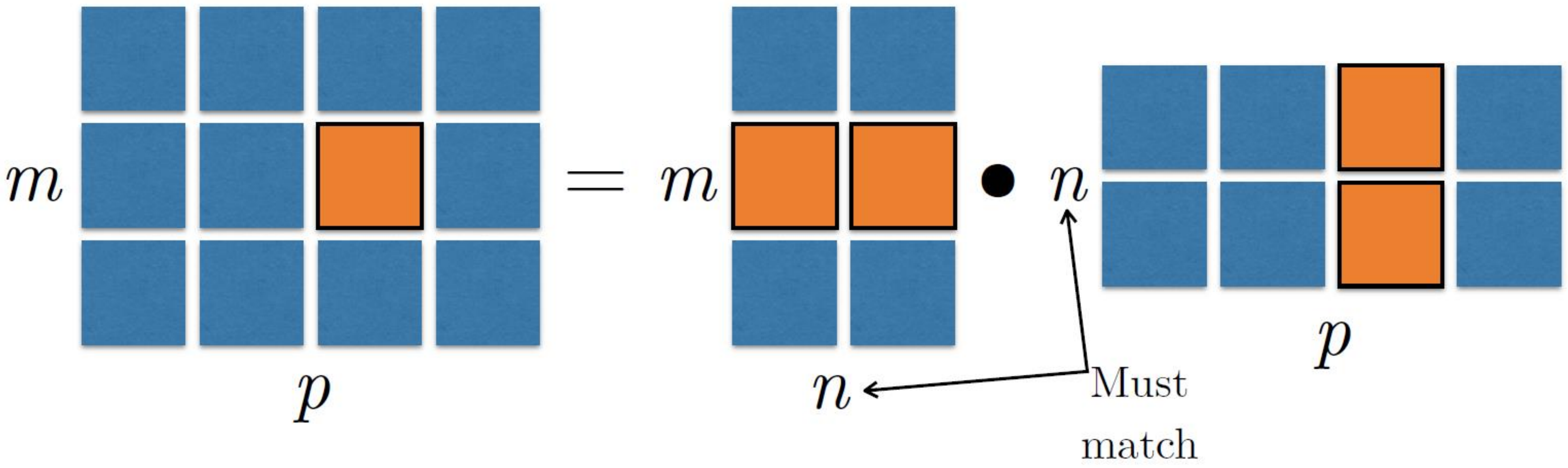
$$A = \left[\begin{tikzcd} A_{1,1} \arrow[dr, dash, dashed] & A_{1,2} \\ A_{2,1} \arrow[ur, bend left=90] & A_{2,2} \\ A_{3,1} & A_{3,2} \end{tikzcd}\right] \Rightarrow A^T = \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \end{bmatrix}$$

The transpose of the matrix can be thought of as a mirror image across the main diagonal.

$$(AB)^T = B^T A^T .$$

Matrix (Dot) Product

- $C = AB$
- $C_{i,j} = \sum_k A_{i,k} B_{k,j}$



Identity Matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Example identity matrix: This is I_3.

$$\forall x \in \mathbb{R}^n, I_n x = x$$

Systems of Equations

$$Ax = b$$

expands to

$$A_{1,:}x = b_1$$

$$A_{2,:}x = b_2$$

$$\dots$$

$$A_{m,:}x = b_m$$

Solving Systems of Equations

- A linear system of equations can have:
  - No solution
  - Many solutions
  - Exactly one solution: this means multiplication by the matrix is an invertible function

Matrix Inversion

- Matrix inverse:
  $$A^{-1}A = I_n$$

- Solving a system using an inverse:
  $$Ax = b$$
  $$A^{-1}Ax = A^{-1}b$$
  $$I_n x = A^{-1}b$$

- Numerically unstable, but useful for abstract analysis

Invertibility

- Matrix can’t be inverted if…

  - More rows than columns

  - More columns than rows

  - Redundant rows/columns (“linearly dependent”, “low rank”)

Norms

- Functions that measure how “large” a vector is
- Similar to a distance between zero and the point represented by the vector
  - $f(x) = 0 \Rightarrow x = 0$
  - $f(x + y) \leq f(x) + f(y)$ (the triangle inequality)
  - $\forall \alpha \in \mathbb{R}, f(\alpha x) = |\alpha| f(x)$

Norms

- L^p norm

$$\|x\|_p = \left(\sum_i |x_i|^p\right)^{\frac{1}{p}}$$

- Most popular norm: L2 norm, p=2
- L1 norm, p=1: $\|x\|_1 = \sum_i |x_i|$.
- Max norm, infinite p: $\|x\|_\infty = \max_i |x_i|$.

Special Matrices and Vectors

- Unit vector:

$$\|x\|_2 = 1$$

- Symmetric Matrix:

$$A = A^T$$

- Orthogonal matrix:

$$A^T A = AA^T = I$$
$$A^{-1} = A^T$$

Eigendecomposition

- Eigenvector and eigenvalue:

$$Av = \lambda v$$

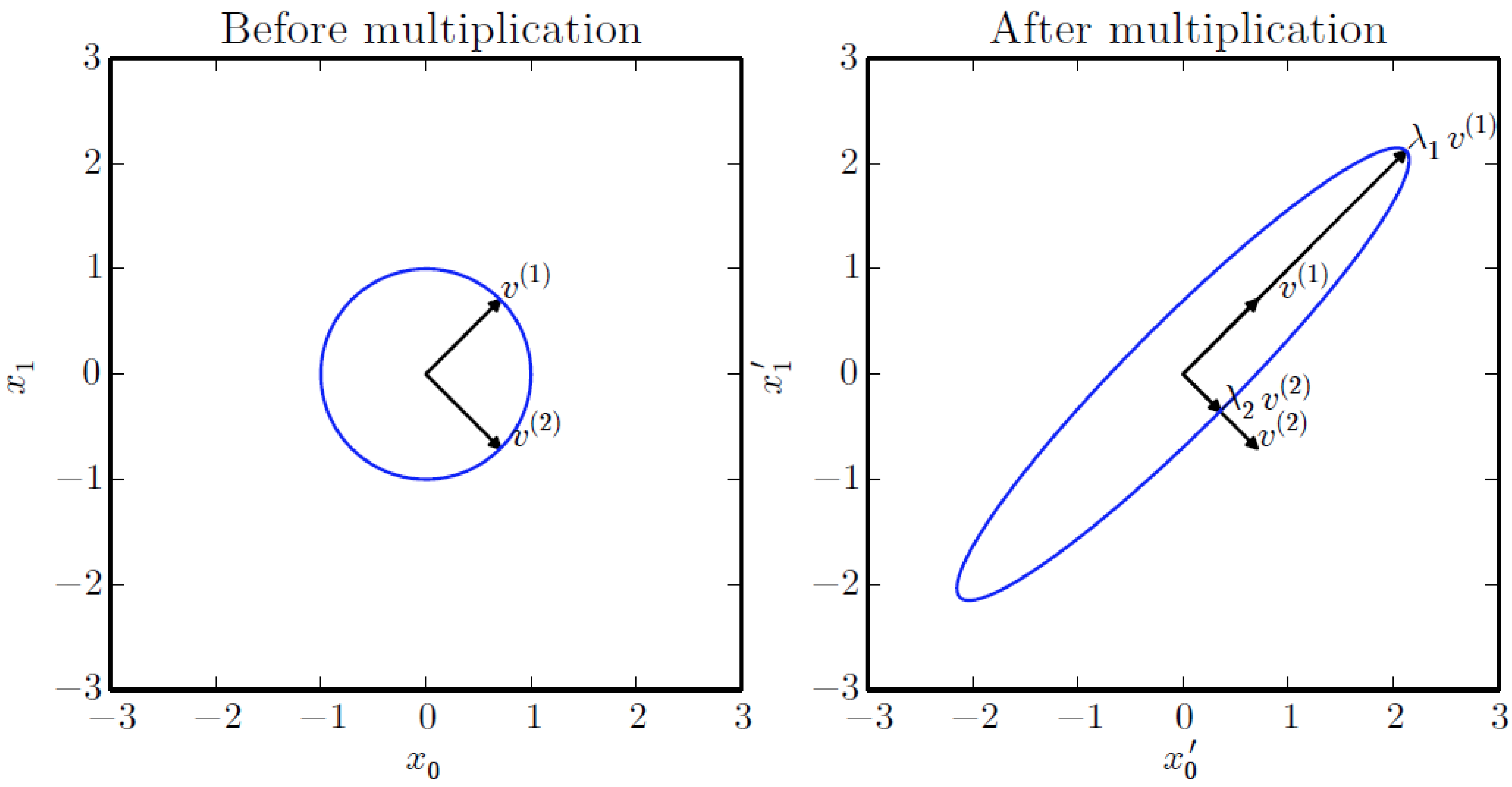
- Eigendecomposition of a diagonalizable matrix:

$$A = V diag(\lambda) V^{-1}$$

- Every real symmetric matrix has a real, orthogonal eigendecomposition:

$$A = Q \Lambda Q^T$$

Effect of Eigenvalues



Singular Value Decomposition

- Similar to eigendecomposition

- More general; matrix need not be square

$$A = UDV^T$$

Moore-Penrose Pseudoinverse

$$x = A^{+}y$$

- If the equation has:
  - Exactly one solution: this is the same as the inverse.
  - No solution: this gives us the solution with the smallest error $\|Ax - y\|_2$.
  - Many solutions: this gives us the solution with the smallest norm of x.

Trace

$$Tr(A) = \sum_i A_{i,i}$$

$$Tr(ABC) = Tr(CAB) = Tr(BCA)$$

Learning linear algebra

- Do a lot of practice problems
- Start out with lots of summation signs and indexing into individual entries
- Eventually you will be able to mostly use matrix and vector product notation quickly and easily

Outline

Probability Mass Function

- The domain of P must be the set of all possible states of x.
- $\forall x \in$ x, $0 \leq P(x) \leq 1$. An impossible event has probability 0 and no state can be less probable than that. Likewise, an event that is guaranteed to happen has probability 1, and no state can have a greater chance of occurring.
- $\sum_{x \in X} P(x) = 1$. We refer to this property as being **normalized**. Without this property, we could obtain probabilities greater than one by computing the probability of one of many events occurring.

Example: uniform distribution: $P(x = x_i) = \frac{1}{k}$

Probability Density Function

- The domain of p must be the set of all possible states
- $\forall x \ \in$ x, $0 \leq P(x)$. Note that we do not require $P(x) \leq 1$.
- $\int p(x)dx = 1$.

Example: uniform distribution: $u(x{:}\, a, b) = \frac{1}{b-a}$

Computing Marginal Probability with Sum Rule

$$\forall x \in \mathrm{x}, P(\mathrm{x} = x) = \sum_y P(\mathrm{x} = x, \mathrm{y} = y).$$

$$p(x) = \int p(x, y) dy.$$

Conditional Probability

$$P(\mathrm{y} = y | \mathrm{x} = x) = \frac{P(\mathrm{y} = y, \mathrm{x} = x)}{P(\mathrm{x} = x)}$$

Chain Rule of Probability

$$P\left(x^{(1)}, \ldots, x^{(n)}\right) = P\left(x^{(1)}\right) \prod_{i=2}^{n} P\left(x^{(i)} \middle| x^{(1)}, \ldots, x^{(i-1)}\right)$$

Independence

$$\forall x \in \mathrm{x}, y \in \mathrm{y}, p(\mathrm{x} = x, \mathrm{y} = y) = p(\mathrm{x} = x)p(\mathrm{y} = y)$$

Conditional Independence

$$\forall x \in \mathrm{x}, y \in \mathrm{y}, z \in \mathrm{z}, p(\mathrm{x} = x, \mathrm{y} = y \,|\mathrm{z} = z)$$
$$= p(\mathrm{x} = x|\mathrm{z} = z)p(\mathrm{y} = y|\mathrm{z} = z)$$

Expectation

$$E_{X\sim P}[f(x)] = \sum_x P(x)f(x),$$

$$E_{X\sim P}[f(x)] = \int p(x)f(x)dx,$$

linearity of expectations:

$$E_X[\alpha f(x) + \beta g(x)] = \alpha E_X[f(x)] + \beta E_X[g(x)]$$

Variance and Covariance

$$Var\left(f(x)\right) = E[(f(X) - E[f(x)])^2]$$

$$Cov\left(f(x), g(y)\right)$$
$$= E[(f(x) - E[f(x)])(g(y) - E[g(y)])]$$

Convariance matrix:

$$Cov(\mathrm{x})_{i,j} = Cov(\mathrm{x}_i, \mathrm{x}_j)$$

Bernoulli Distribution

$$P(\mathrm{x} = 1) = \emptyset$$

$$P(\mathrm{x} = 0) = 1 - \emptyset$$

$$P(\mathrm{x} = x) = \emptyset^{x}(1 - \emptyset)^{1-x}$$

$$E_{\mathrm{x}}[\mathrm{x}] = \emptyset$$

$$\mathrm{Var}_{\mathrm{x}}(\mathrm{x}) = \emptyset(1 - \emptyset)$$

Gaussian Distribution

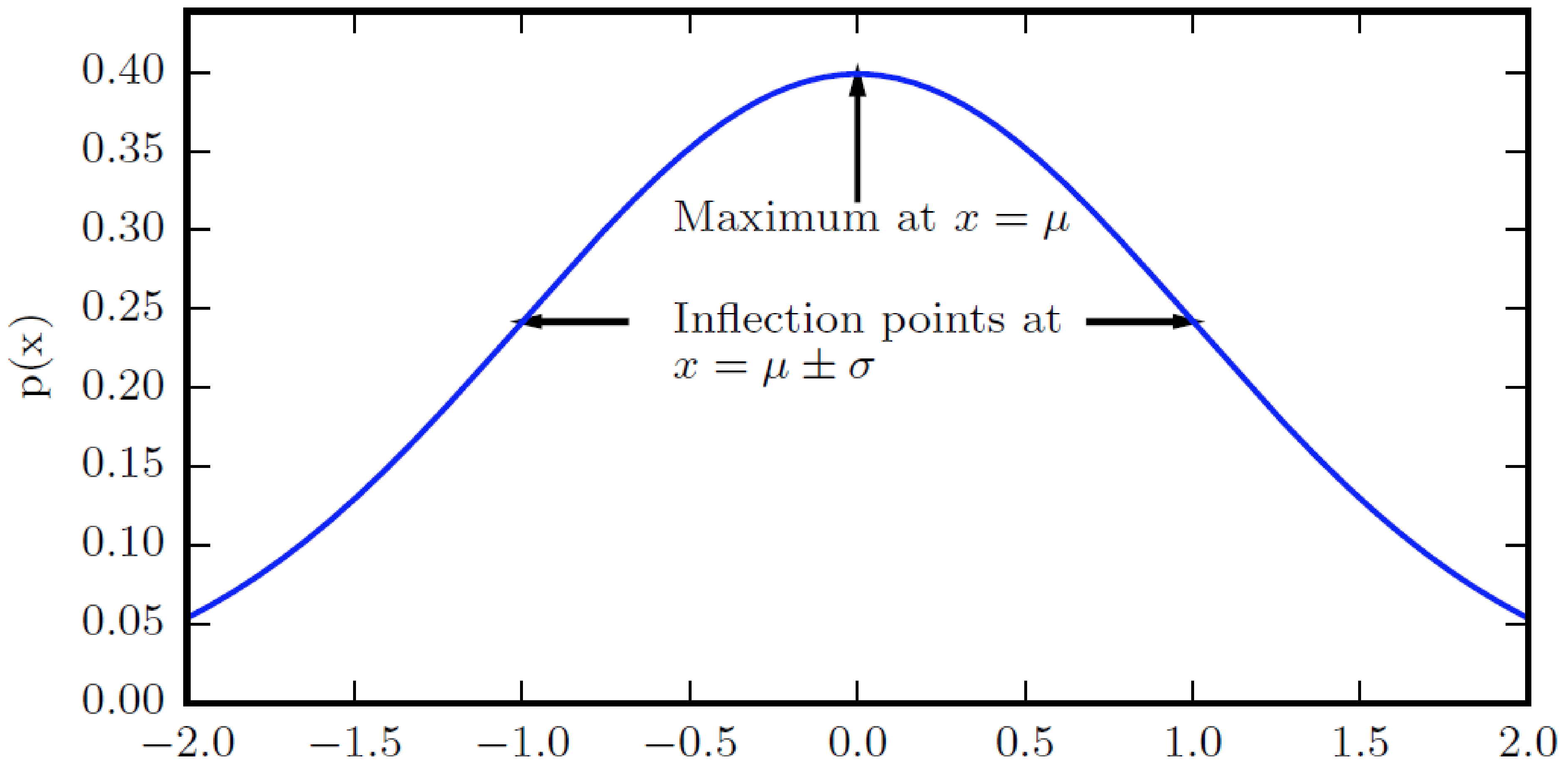
Parametrized by variance:

$$N(x; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)$$

Parametrized by precision:

$$N(x; \mu, \beta^{-1}) = \sqrt{\frac{\beta}{2\pi}} \exp\left(-\frac{1}{2}\beta(x-\mu)^2\right)$$

Gaussian Distribution



Multivariate Gaussian

Parametrized by covariance matrix:

$$N(x; \mu, \sum) = \sqrt{\frac{1}{(2\pi)^n \mathrm{det}(\sum)}} \exp\left(-\frac{1}{2}(x-\mu)^T \sum^{-1}(x-\mu)\right)$$

Parametrized by precision matrix:

$$N(x; \mu, \beta^{-1}) = \sqrt{\frac{\mathrm{det}(\beta)}{(2\pi)^n}} \exp\left(-\frac{1}{2}(x-\mu)^T \beta (x-\mu)\right)$$

More Distributions

Exponential:

$$p(x;\lambda) = \lambda \mathbf{1}_{x\geq 0} exp(-\lambda x)$$

Laplace:

$$\text{Laplace}(x;\mu,\gamma) = \frac{1}{2\gamma}\exp(-\frac{|x-\mu|}{\gamma})$$

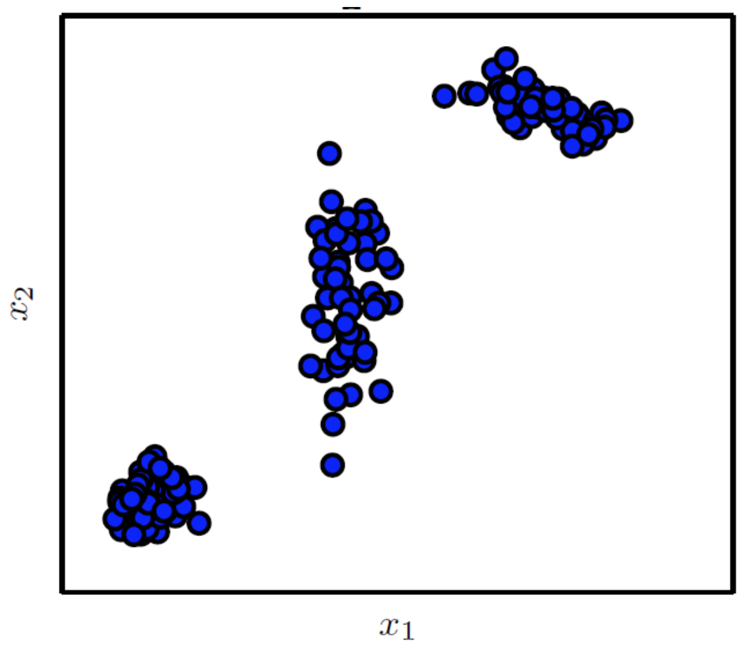
Dirac:

$$p(x) = \delta(x-\mu)$$

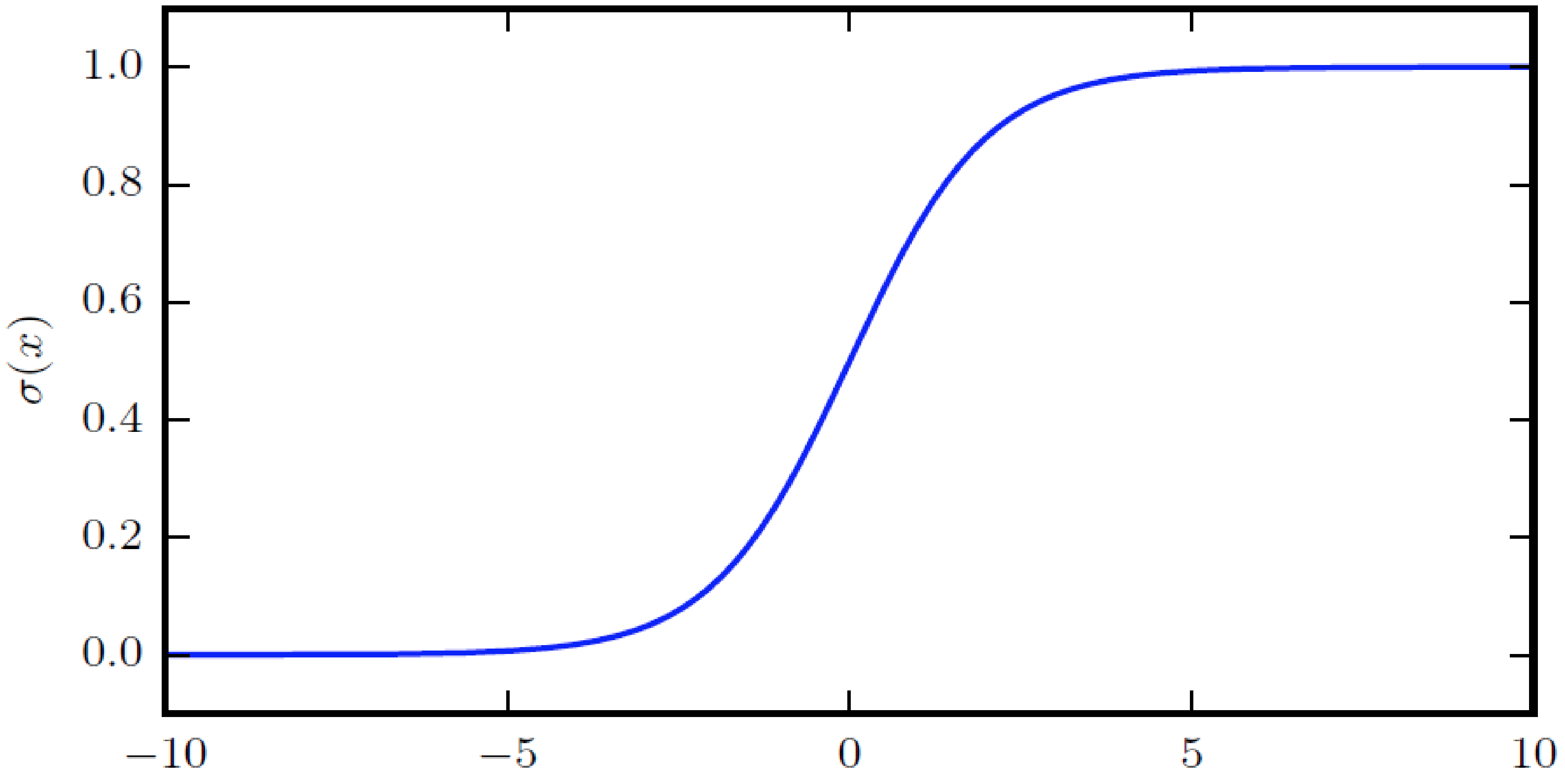
Mixture Distributions

$$P(x) = \sum_i P(c=i)P(x|c=i)$$

Gaussian mixture
with three
components



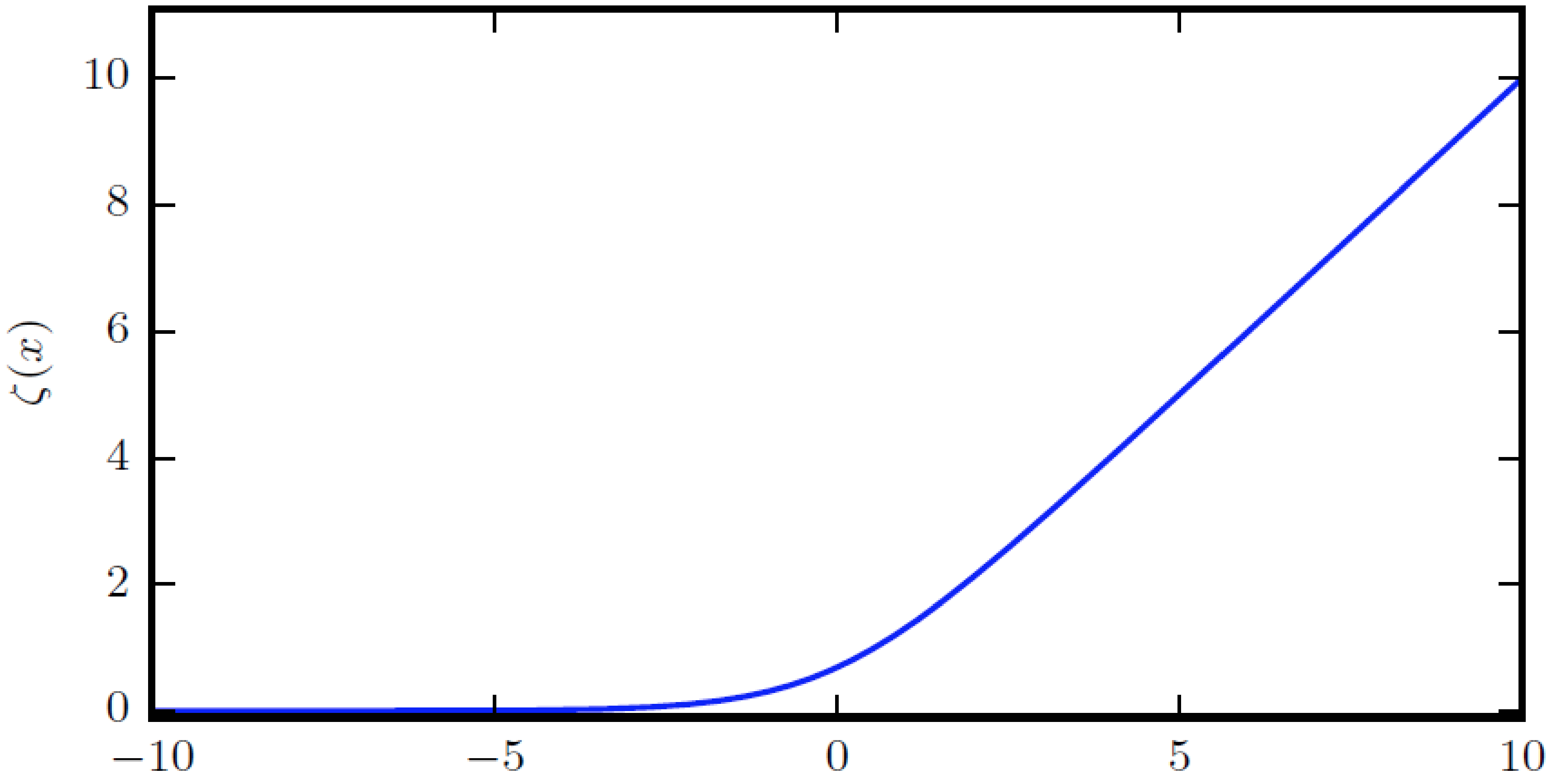
Logistic Sigmoid



The logistic sigmoid function

Commonly used to parametrize Bernoulli distributions

Softplus Function



The softplus function

Bayes' Rule

$$P(x|y) = \frac{P(x)P(y|x)}{P(y)}$$

Information Theory

Information:

$$I(x) = -\log P(x)$$

Entropy:

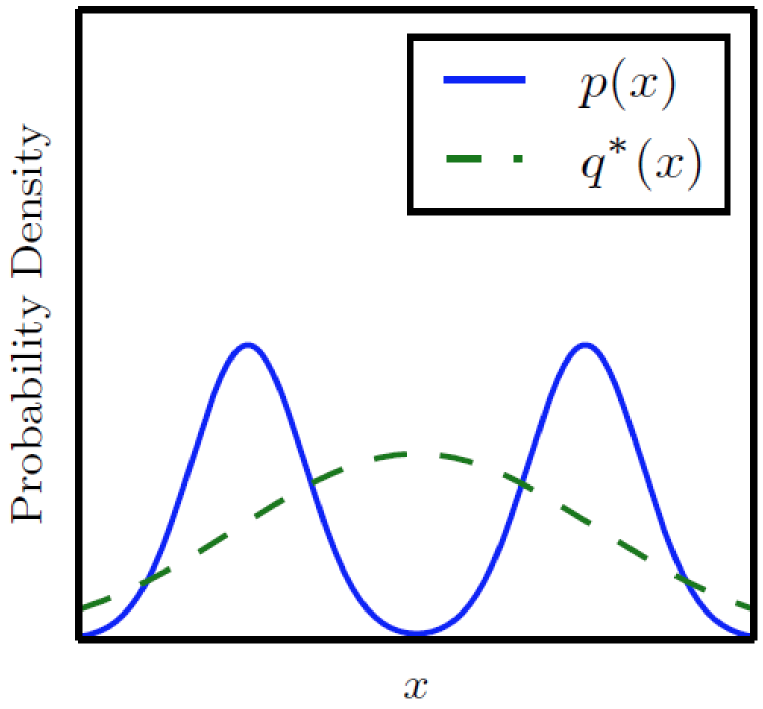
$$\mathrm{H}(x) = E_{X\sim P}[I(x)] = E_{X\sim P}[\log P(x)]$$

KL divergence:

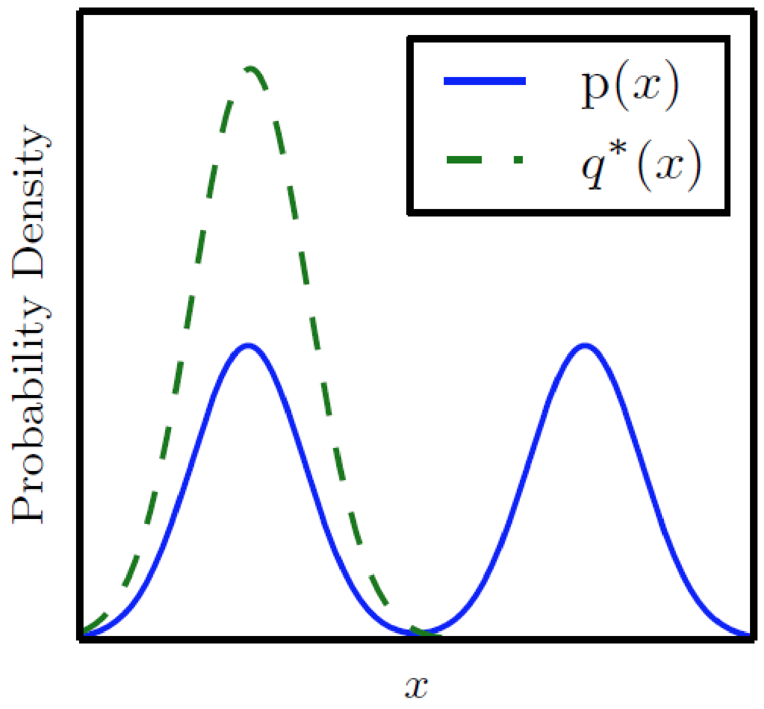
$$D_{KL}(P \parallel Q) = E_{X\sim P}\left[log\frac{P(x)}{Q(x)}\right]$$
$$= E_{X\sim P}[logP(x) - logQ(x)]$$

The KL Divergence is Asymmetric

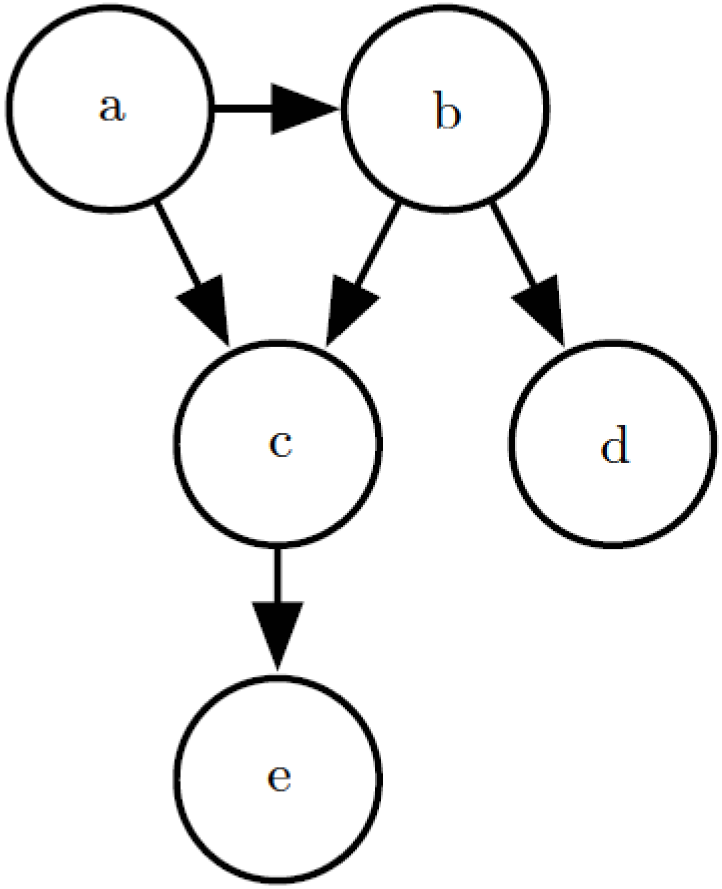
$q^{*} = argmin_q D_{KL}(p \parallel q)$



$q^{*} = argmin_q D_{KL}(q \parallel p)$



Directed Model



$$p(a, b, c, d, e) = p(a)p(b|a)p(c|a, b)p(d|b)p(e|c)$$

Undirected Model

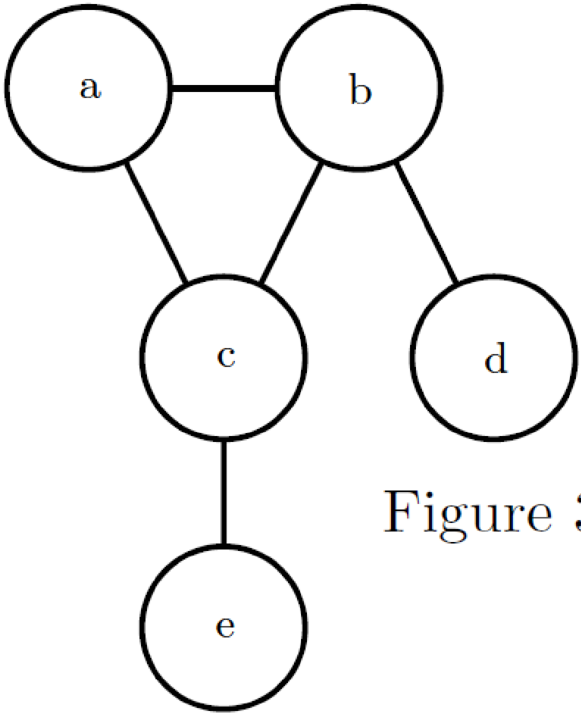


Figure 3.8

$$p(a,b,c,d,e) = \frac{1}{Z}\phi^{(1)}(a,b,c)\phi^{(2)}(b,d)\,\phi^{(3)}(c,e)$$

Outline

Numerical concerns for DL

- Algorithms are often specified in terms of real numbers; real numbers cannot be implemented in a finite computer
    - Does the algorithm still work when implemented with a finite number of bits?
- Do small changes in the input to a function cause large changes to an output?
    - Rounding errors, noise, measurement errors can cause large changes
    - Iterative search for best input is difficult

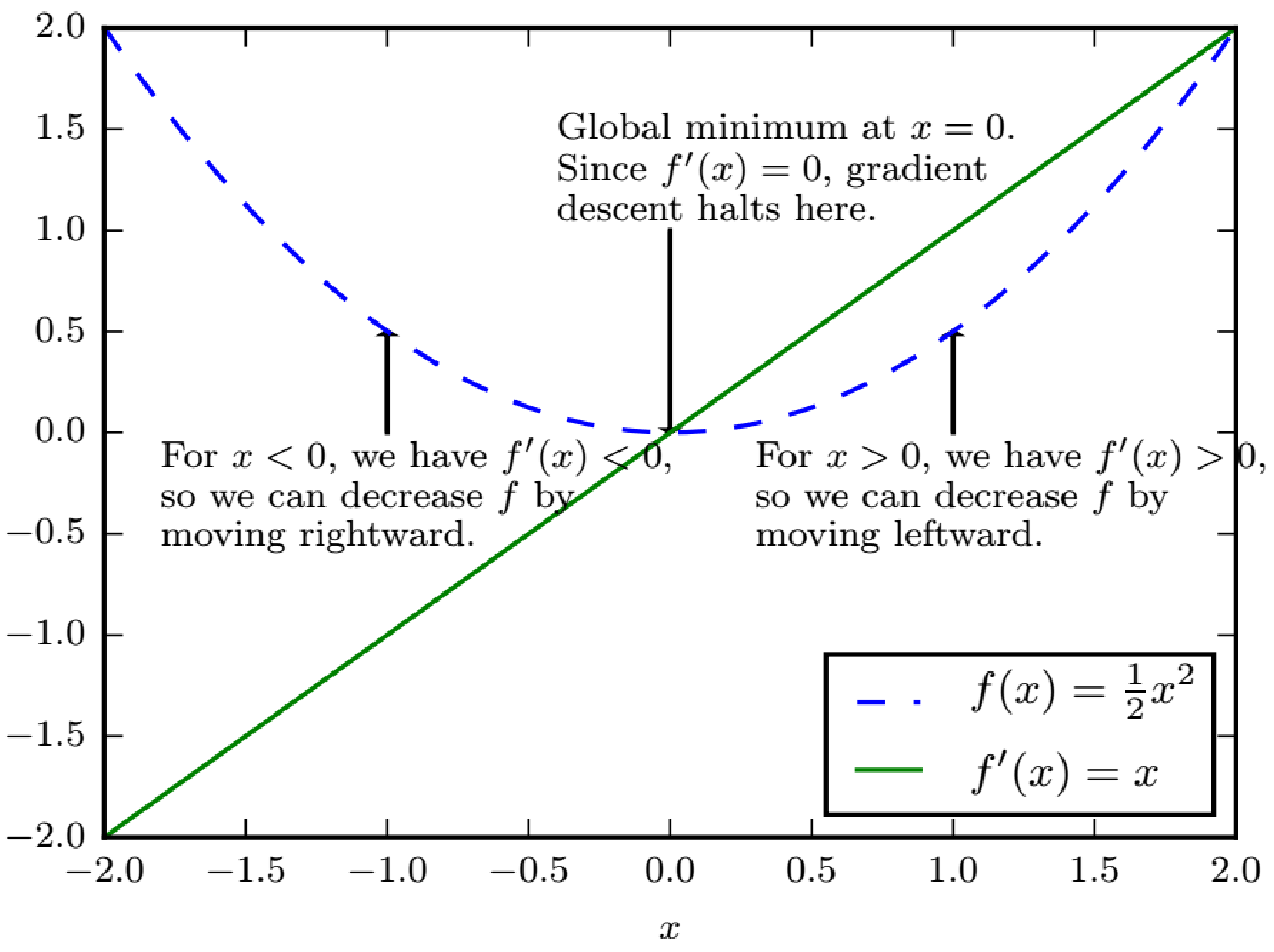
Roadmap

- Iterative Optimization
- Rounding error, underflow, overflow

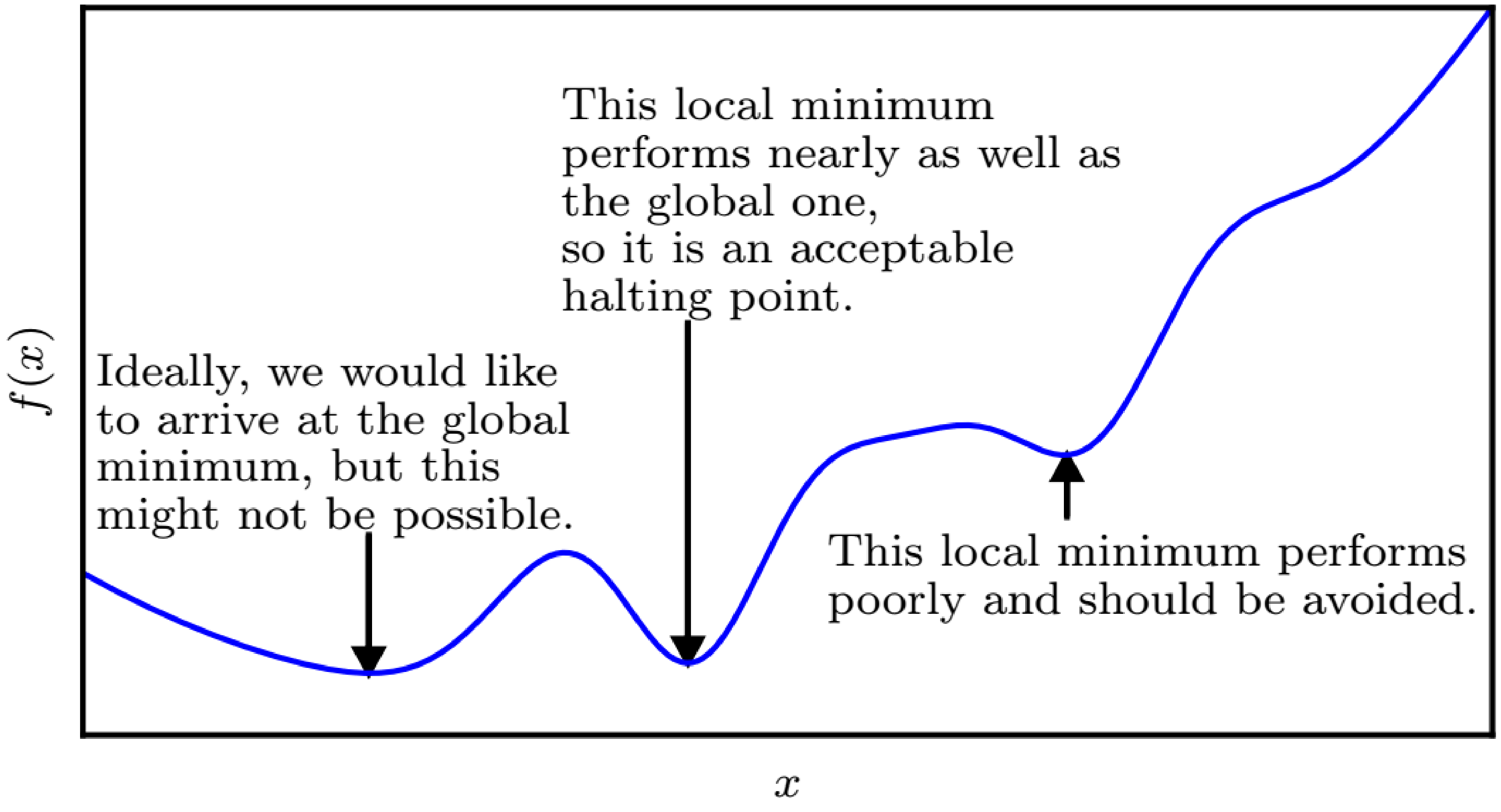
Iterative Optimization

- Gradient descent
- Curvature

Gradient Descent

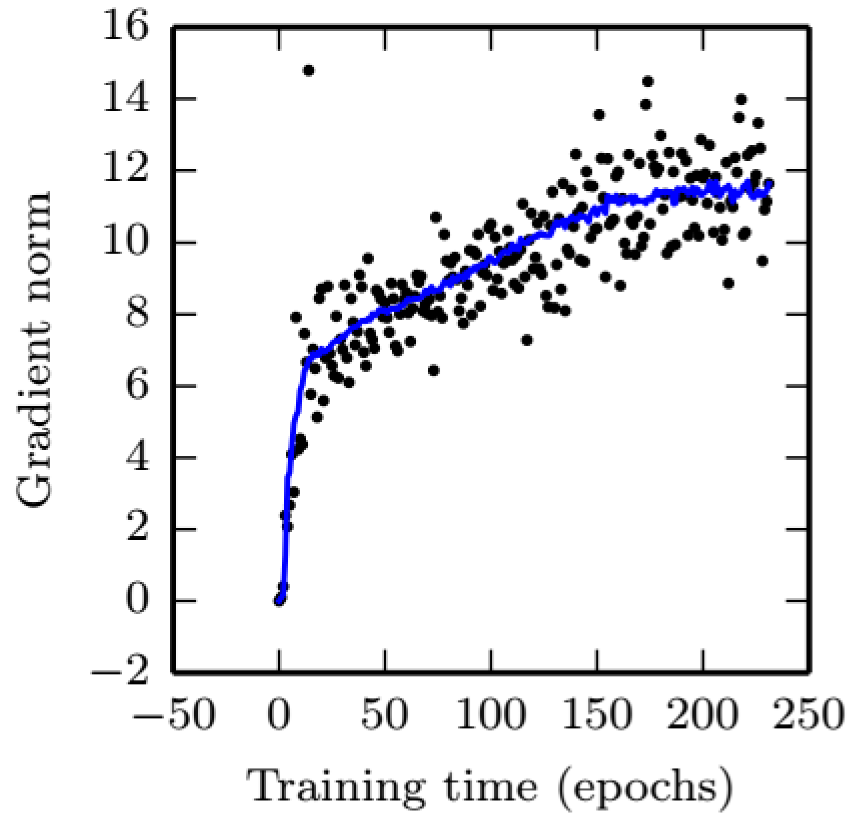


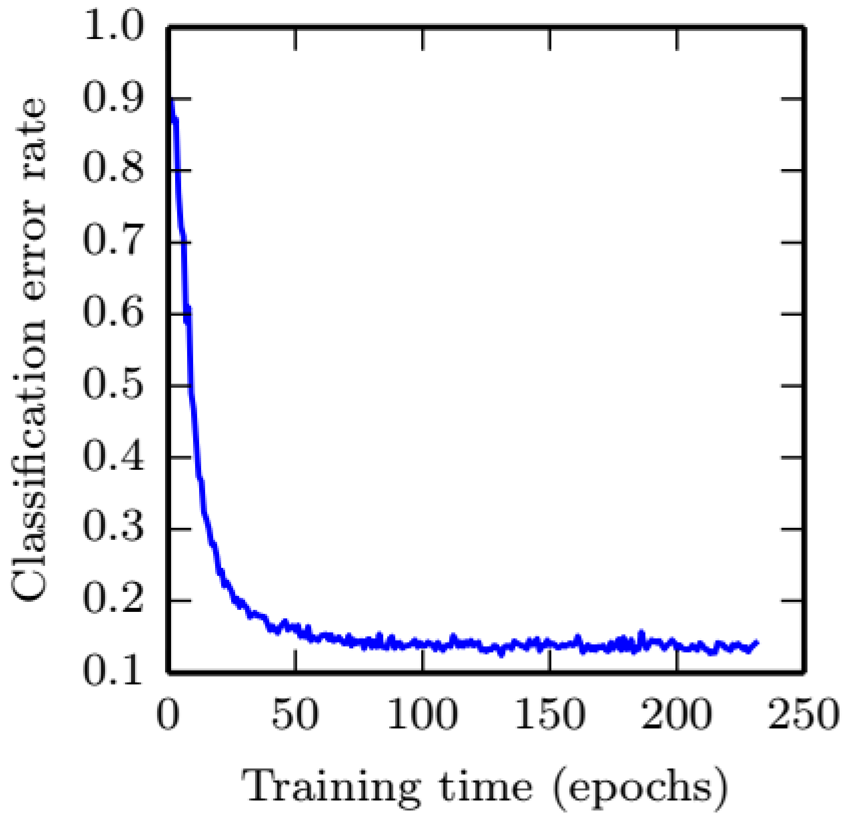
Approximate Optimization



Approximate Optimization

We usually don't even reach a local minimum.





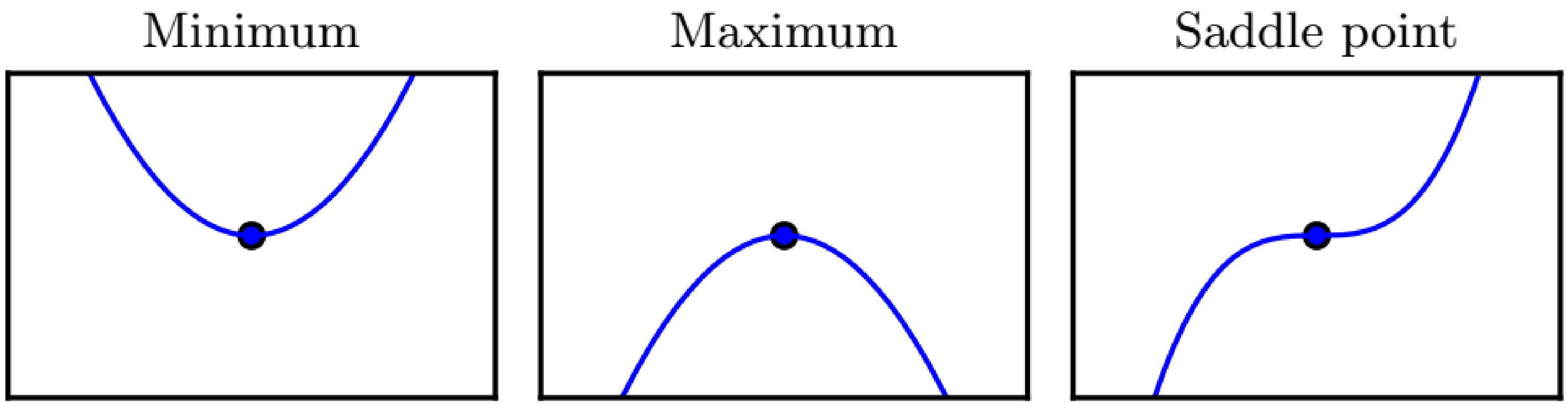
Optimization Way of Life in DL

- Pure math way of life:
  - Find literally the smallest value of *f(x)*
  - Or maybe: find some critical point of *f(x)* where the value is locally smallest
- Deep learning way of life:
  - Decrease the value of *f(x)* a lot

Iterative Optimization

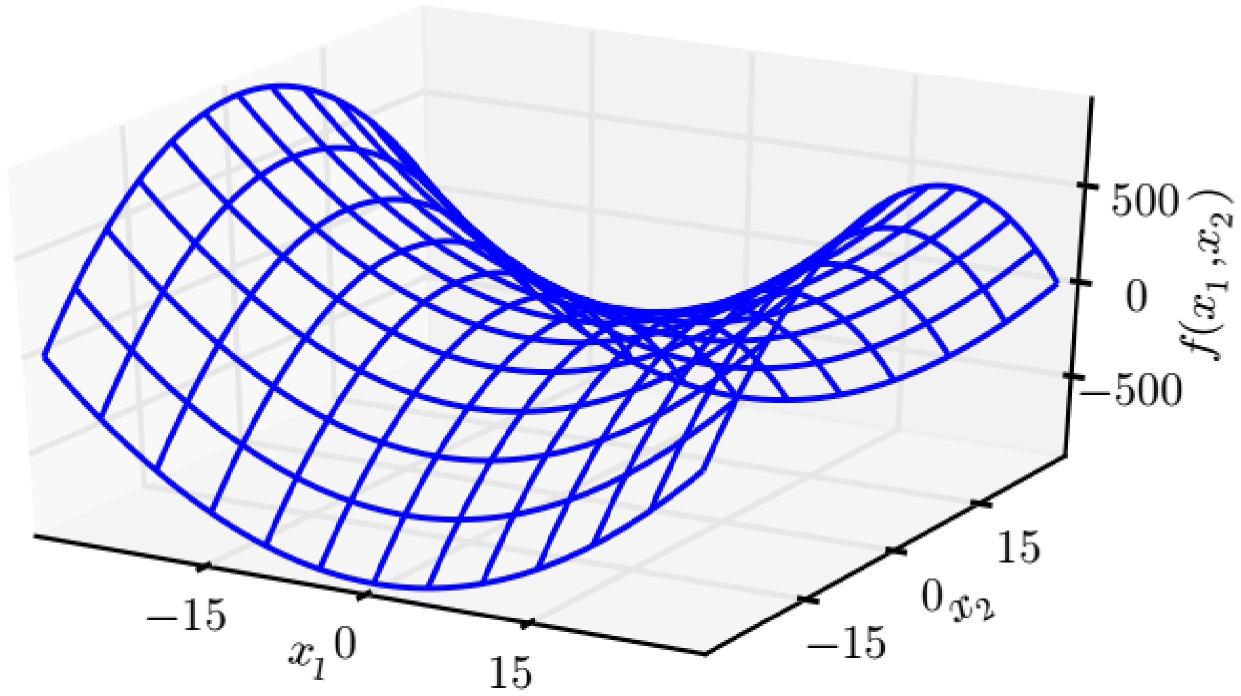
- Gradient descent
- Curvature

Critical Points



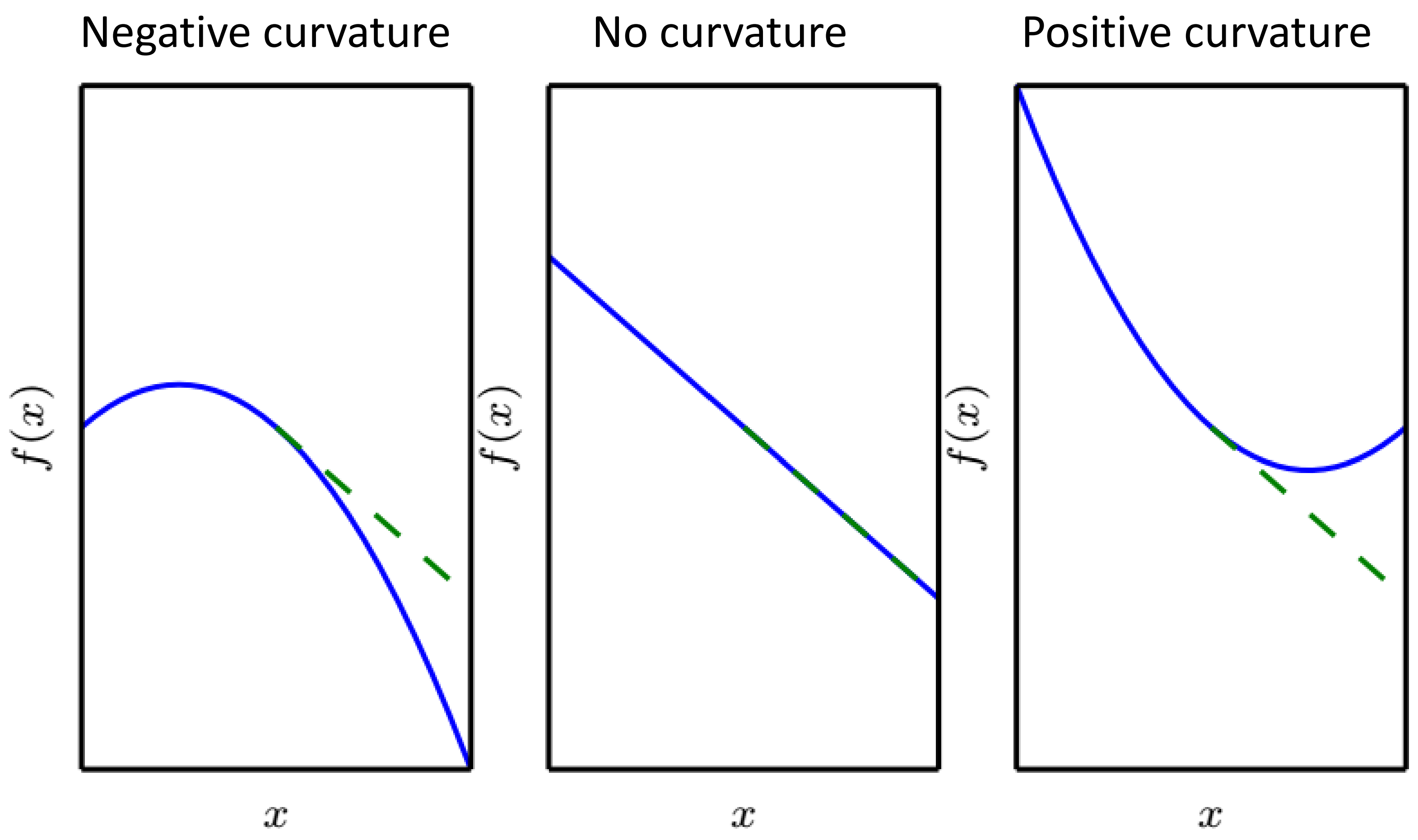
Saddle Points

A saddle point containing both positive and negative curvature.

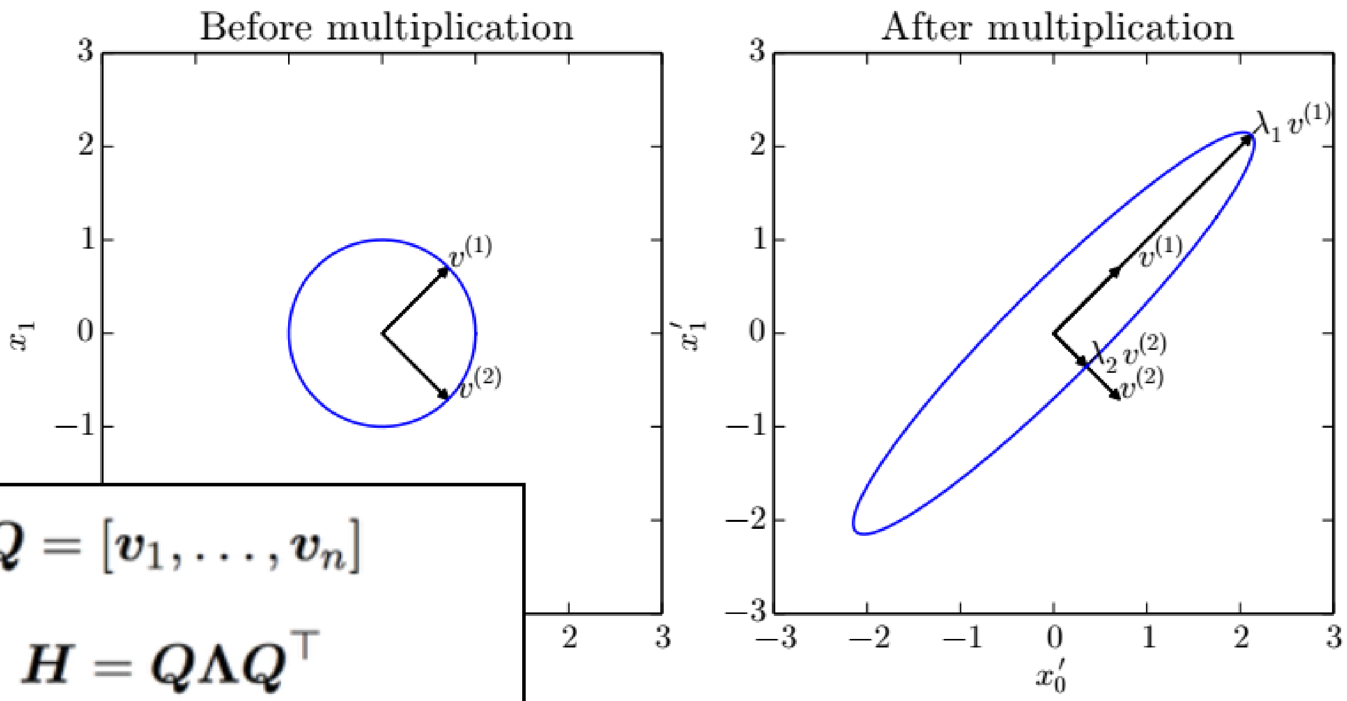


Saddle points attract Newton’s method

Curvature


Negative curvature
No curvature
Positive curvature
$f(x)$
$f(x)$
$f(x)$
x
x
x

Directional Second Derivatives



$$Q = [v_1, \dots, v_n]$$

$$H = Q\Lambda Q^\top$$

Second derivative in direction d:

$$d^\top H d = \sum_i \lambda_i \cos^2 \angle(v_i, d)$$

Predicting optimal step size

Using Taylor series

$$f(\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}) \approx f(\boldsymbol{x}^{(0)}) - \epsilon \boldsymbol{g}^\top \boldsymbol{g} + \frac{1}{2}\epsilon^2 \boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}.$$

$$\epsilon^* = \frac{\boldsymbol{g}^\top \boldsymbol{g}}{\boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}}.$$

Big gradients speed you up

Big eigenvalues slow you down if you align with their eigenvectors

Outline

1/ Course Review

2/ Linear Algebra

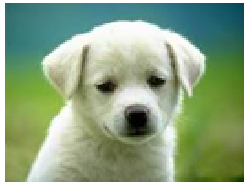
3/ Probability and Information Theory

4/ Numerical Computation for DL

5/ Machine Learning Basics

Machine Learning

$$\begin{tikzcd} x \arrow[r] & \boxed{f(x)} \arrow[r] & y \end{tikzcd} \left\{ \begin{array}{c} \text{Class label} \\ \text{(Classification)} \\ \\ \text{Real-valued vector} \\ \text{(Regression)} \end{array} \right.$$



$\xrightarrow{\text{Object recognition}}$ {dog, cat, horse, flower, …}

Low-resolution image

$\xrightarrow{\text{Super resolution}}$

High-resolution image

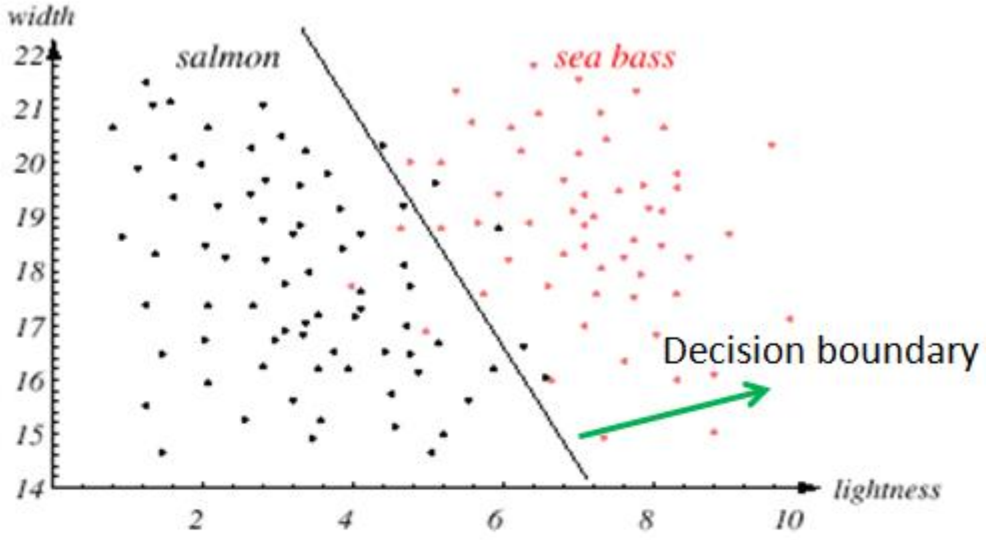
Classification

- f (x) predicts the category that x belongs to

$$f : \mathcal{R}^D \rightarrow \{1, \dots, K\}$$

- f (x) is decided by the decision boundary
- As an variant, f can also predict the probability distribution over classes given x, f (x) = P(y|x). The category is predicted as

$$y^* = \arg\max_k P(y = k|\mathbf{x})$$



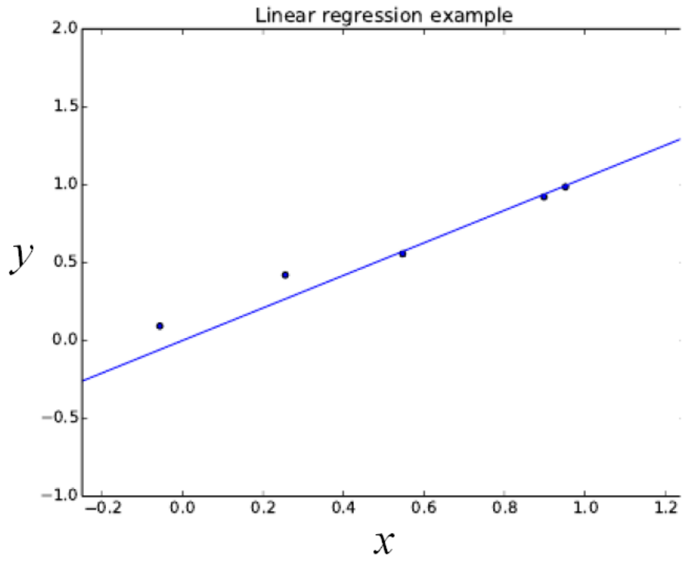
Regression

- Predict real-valued output

$$f : \mathcal{R}^D \rightarrow \mathcal{R}^M$$

- Example: linear regression

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



Training

- Training: estimate the parameters of f from $\{(\mathbf{x}_i^{\text{(train)}}, y_i^{\text{(train)}})\}$
  - Decision boundary, parameters of P(y|x), and w in linear regression
- Optimize an objective function on the training set. It is a performance measure on the training set and could be different from that on the test set.
  - Mean squared error (MSE) for linear regression

$$\text{MSE}_{\text{train}} = \frac{1}{N}\sum_i ||\mathbf{w}^t\mathbf{x}_i^{\text{(train)}} - y_i^{\text{(train)}}||_2^2$$

  - Cross entropy (CE) for classification

$$\text{CE}_{\text{train}} = \frac{1}{N}\sum_i \log P(y = y_i^{\text{(train)}}|\mathbf{x}_i^{\text{(train)}})$$

Why not use classification errors #{f($x_i^{(train)}$ $\neq y_i^{(train)}$)} ?

Optimization

- The choice of the objective function should be good for optimization
- Take linear regression as an example

$$\nabla_{\mathbf{w}} \text{MSE}_{\text{train}} = 0$$

$$\Rightarrow \nabla_{\mathbf{w}} ||\mathbf{X}^{(\text{train})}\mathbf{w} - \mathbf{y}^{(\text{train})}||_2^2 = 0$$

$$\mathbf{w} = (\mathbf{X}^{(\text{train})t}\mathbf{X}^{(\text{train})})^{-1}\mathbf{X}^{(\text{train})t}\mathbf{y}^{(\text{train})}$$

where $\mathbf{X}^{(\text{train})} = [\mathbf{x}_1^{(\text{train})}, \ldots, \mathbf{x}_N^{(\text{train})}]$ and $\mathbf{y}^{(\text{train})} = [y_1^{(\text{train})}, \ldots, y_N^{(\text{train})}]$.

Generalization

- We care more about the performance of the model on new, previously unseen examples
- The training examples usually cannot cover all the possible input configurations, so the learner has to generalize from the training examples to new cases
- Generalization error: the expected error over **ALL** examples
- To obtain theoretical guarantees about generalization of a machine learning algorithm, we assume all the samples are drawn from a distribution p(x,y), and calculate generalization error (GE) of a prediction function f by taking expectation over p(x, y)

$$GE_f = \int_{\mathbf{x},y} p(\mathbf{x}, y)\mathbf{Error}(f(\mathbf{x}), y)$$

Generalization

- However, in practice, p(x,y) is unknow. We assess the generalization performance with a test set $\{\mathbf{x}_i^{(\text{test})}, y_i^{(\text{test})}\}$

$$\text{Performance}_{\text{test}} = \frac{1}{M} \sum_{i=1}^{M} \text{Error}(f(\mathbf{x}_i^{(\text{test})}), y_i^{(\text{test})})$$

- We hope that both test examples and training examples are drawn from p(x, y) of interest, although it is unknown

Capacity

- The ability of the learner (or called model) to discover a function taken from a family of functions. Examples:
  - Linear predictor
    $$y = wx + b$$
  - Quadratic predictor
    $$y = w_2 x^2 + w_1 x + b$$
  - Degree-10 polynomial predictor
    $$y = b + \sum_{i=1}^{10} w_i x^i$$
  - The latter family is richer, allowing to capture more complex functions
- Capacity can be measured by the number of training examples $\{\mathbf{x}_i^{(\text{train})}, y_i^{(\text{train})}\}$ that the learner **could always fit**, no matter how to change the values $\mathbf{x}_i^{(\text{train})}$ and $y_i^{(\text{train})}$

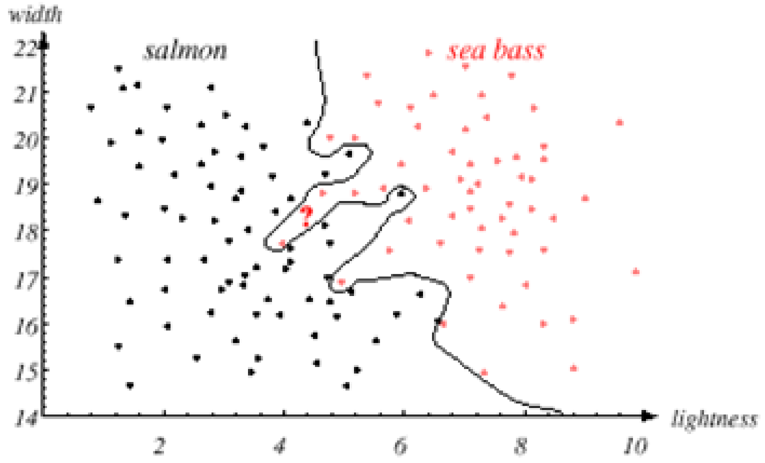
Underfitting

- The learner cannot find a solution that fits training examples well
  - For example, use linear regression to fit training examples $\{\mathbf{x}_i^{(\text{train})}, y_i^{(\text{train})}\}$ where $y_i^{(\text{train})}$ is an quadratic function of $\mathbf{x}_i^{(\text{train})}$
- Underfitting means the learner cannot capture some important aspects of the data
- Reasons for underfitting happening
  - Model is not rich enough
  - Difficult to find the global optimum of the objective function on the training set or easy to get stuck at local minimum
  - Limitation on the computation resources (not enough training iterations of an iterative optimization procedure)
- Underfitting commonly happens in deep learning with large scale training data and could be even a more serious problem than overfitting in some cases

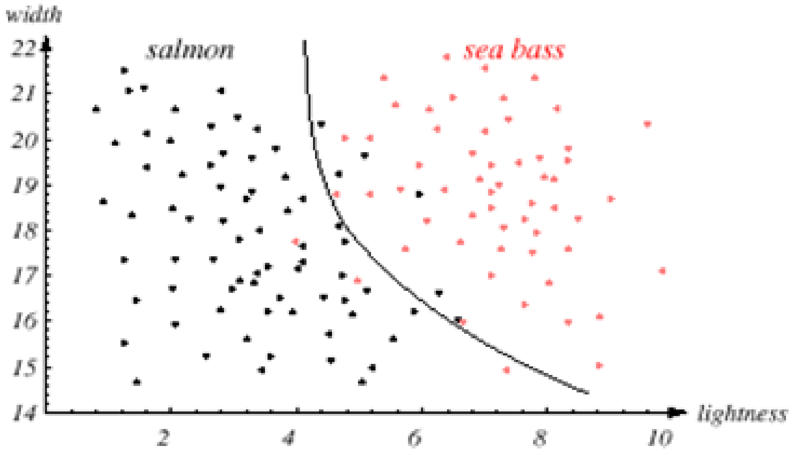
Overfitting

- The learner fits the training data well, but loses the ability to generalize well, i.e. it has small training error but larger generalization error
- A learner with large capacity tends to overfit
  - The family of functions is too large (compared with the size of the training data) and it contains many functions which all fit the training data well.
  - Without sufficient data, the learner cannot distinguish which one is most appropriate and would make an arbitrary choice among these apparently good solutions
  - A separate validation set helps to choose a more appropriate one
  - In most cases, data is contaminated by noise. The learner with large capacity tends to describe random errors or noise instead of the underlying models of data (classes)

Overfitting



Overly complex models lead to complicated decision boundaries. It leads to perfect classification on the training examples, but would lead to poor performance on new examples.



The decision boundary might represent the optimal tradeoff between performance on the training set and simplicity of classifier, therefore giving highest accuracy on new examples.

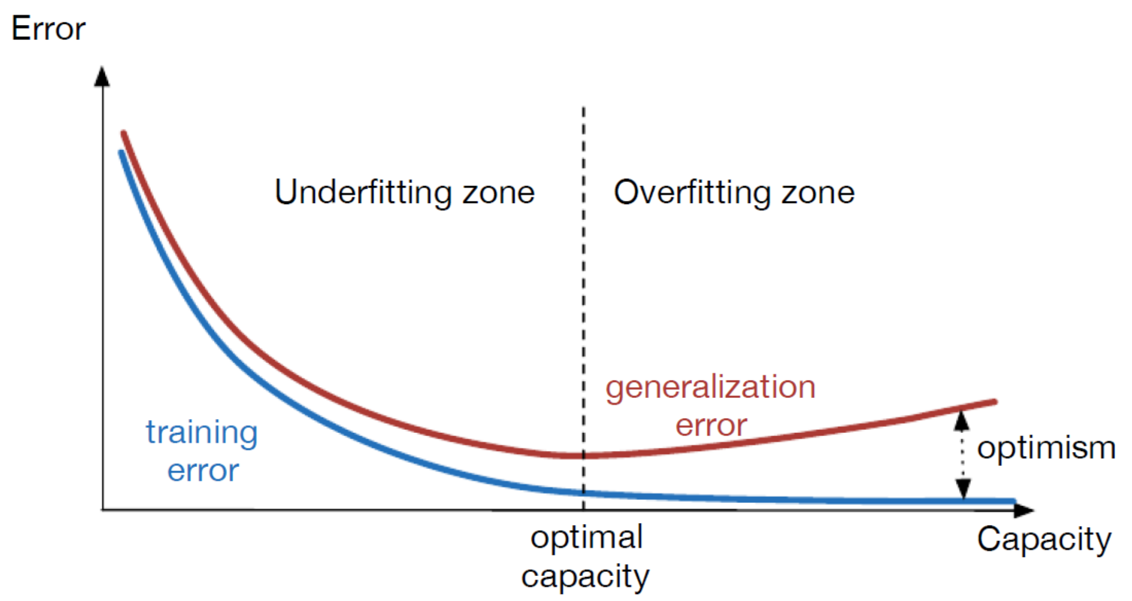
Occam’s Razor

- **The fundamental element of machine learning is the trade-off between capacity and generalization**
- Occam’s Razor states that among competing functions that could explains the training data, one should choose the “simpler” one. Simplicity is the opposite of capacity.
- Occam’s Razor suggests us pick the family of functions just enough large enough to leave only one choice that fits well the data.

Optimal capacity

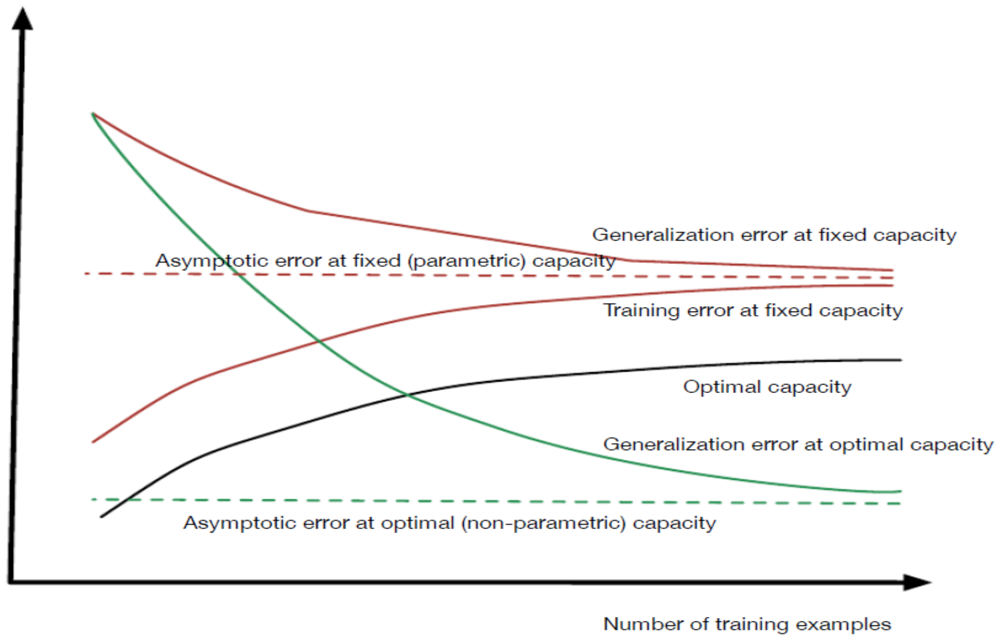
- Difference between training error and generalization error increases with the capacity of the learner
- Generalization error is a U-shaped function of capacity
- Optimal capacity capacity is associated with the transition from underfitting to overfitting
    - One can use a validation set to monitor generalization error empirically
- Optimal capacity should increase with the number of training examples

Optimal capacity



Typical relationship between capacity and both training and generalization (or test) error. As capacity increases, training error can be reduced, but the optimism (difference between training and generalization error) increases. At some point, the increase in optimism is larger than the decrease in training error (typically when the training error is low and cannot go much lower), and we enter the overfitting regime, where capacity is too large, above the optimal capacity. Before reaching optimal capacity, we are in the underfitting regime.

Optimal capacity



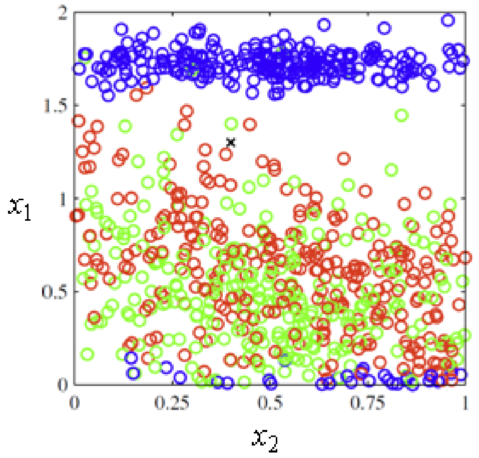
As the number of training examples increases, optimal capacity (bold black) increases (we can afford a bigger and more flexible model), and the associated generalization error (green bold) would decrease, eventually reaching the (non-parametric) asymptotic error (green dashed line). If capacity was fixed (parametric setting), increasing the number of training examples would also decrease generalization error (top red curve), but not as fast, and training error would slowly increase (bottom red curve), so that both would meet at an asymptotic value (dashed red line) corresponding to the best achievable solution in some class of learned functions.

How to reduce capacity?

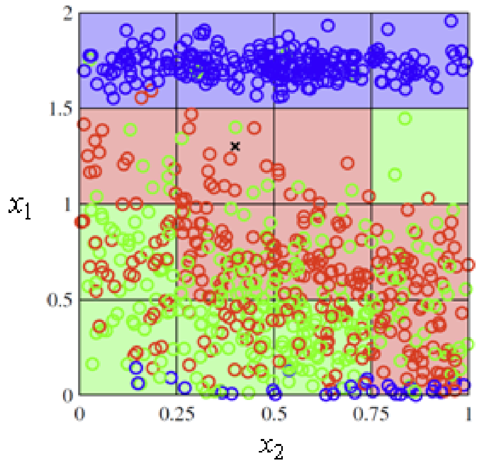
- Reduce the number of features
- Reduce the number of independent parameters
- Reduce the network size of deep models
- Reduce the number of training iterations
- Add regularization to the learner
- . . .

Curse of dimensionality

- Why do we need to reduce the dimensionality of the feature space?



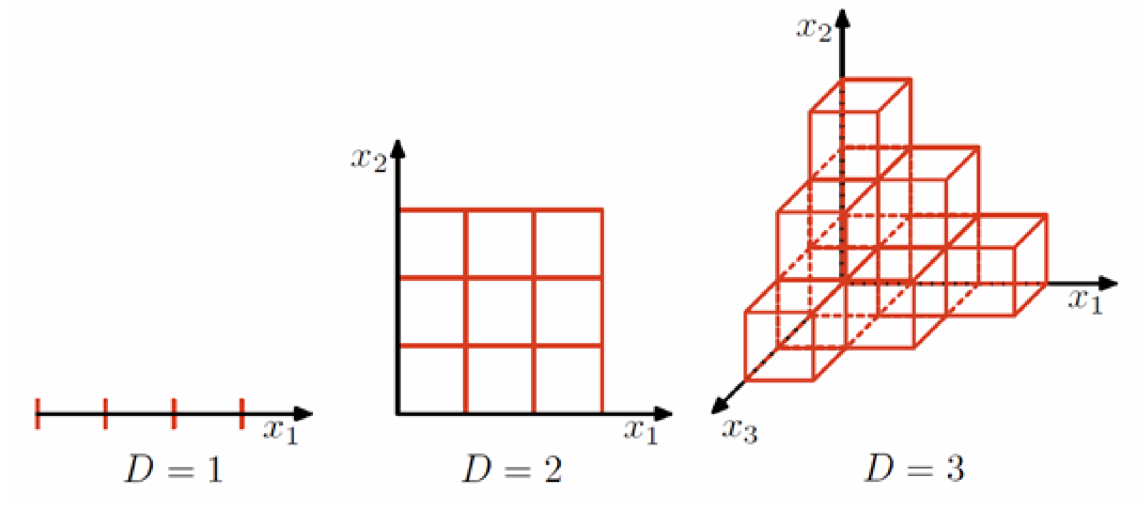
Scatter plot of the training data of three classes. Two features are used. The goal is to classify the new testing point denoted by '×'.



The feature space is uniformly divided into cells. A cell is labeled as a class, if the majority of training examples in that cell are from that class. The testing point is classified according to the label of the cell where it falls in.

Curse of dimensionality

- The more training samples in each cell, the more robust the classifier
- The number of cells grows exponentially with the dimensionality of the feature space. If each dimension is divided into three intervals, the number of cells is $N = 3^D$
- Some cells are empty when the number of cells is very large!



Regularization

- Equivalent to imposing a preference over the set of functions that a learner can obtain as a solution
- In Bayesian learning, it is reflected as a prior probability distribution over the space of functions (or equivalently their parameters) that the learn can assess
- Regularization prevents overfitting by adding penalty for complexity
- Training a classifier/regressor is to minimize
  Prediction error on the training set + regularization
- Examples
  - The objective function for linear regression becomes

$$\text{MSE}_{\text{train}} + \text{regularization} = \frac{1}{N}\sum_i (\mathbf{w}^t \mathbf{x}_i^{(\text{train})} - y_i^{(\text{train})})^2 + \lambda ||\mathbf{w}||_2^2$$

  - Multi-task learning, transfer learning, dropout, sparsity, pre-training

Function estimation

- We are interested in predicting y from input x and assume there exists a function that describes the relationship between y and x,
  e.g $y = f(\mathbf{x}) + \epsilon$ where $\epsilon$is random noise following certain distribution.
- Prediction function f can be parametrized by a parameter vectorθ.
- Estimating$\widehat{f_n}$ from a training set $\mathcal{D}_n = \{(\mathbf{x}_1^{(\text{train})}, y_1^{(\text{train})}), \ldots, (\mathbf{x}_n^{(\text{train})}, y_n^{(\text{train})})\}$ is
- equivalent to estimating $\widehat{\theta_n}$ from D_n.
- Since D_n is randomly generated from a underlying distribution, both $\hat{\theta}$and $\hat{f}$ are random variables (or vectors, or functions) distributed according to some probability distributions.
- The quality of estimation can be measured by bias and variance
- compared with the “true” parameter vector θ or function $\hat{f}$.
- With a better design of the parametric form of the function, the learner could achieve low generalization error even with small capacity
- This design process typical involves domain knowledge

Bias

$$\text{bias}(\hat{\theta}) = E(\hat{\theta}) - \theta$$

where expectation is over all the train sets of size n sampled from the underlying distribution

- An estimator is called unbiased if $E(\hat{\theta}) = \theta$
- Example: Gaussian distribution. $p(\mathbf{x}_i; \theta) = \mathcal{N}(\theta, \Sigma)$ and the estimator is $\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i^{(\text{train})}$

$$E(\hat{\theta}) = E\left[\frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i^{(\text{train})}\right] = \frac{1}{n}\sum_{i=1}^{n} E\left[\mathbf{x}_i^{(\text{train})}\right] = \frac{1}{n}\sum_{i=1}^{n} \theta = \theta$$

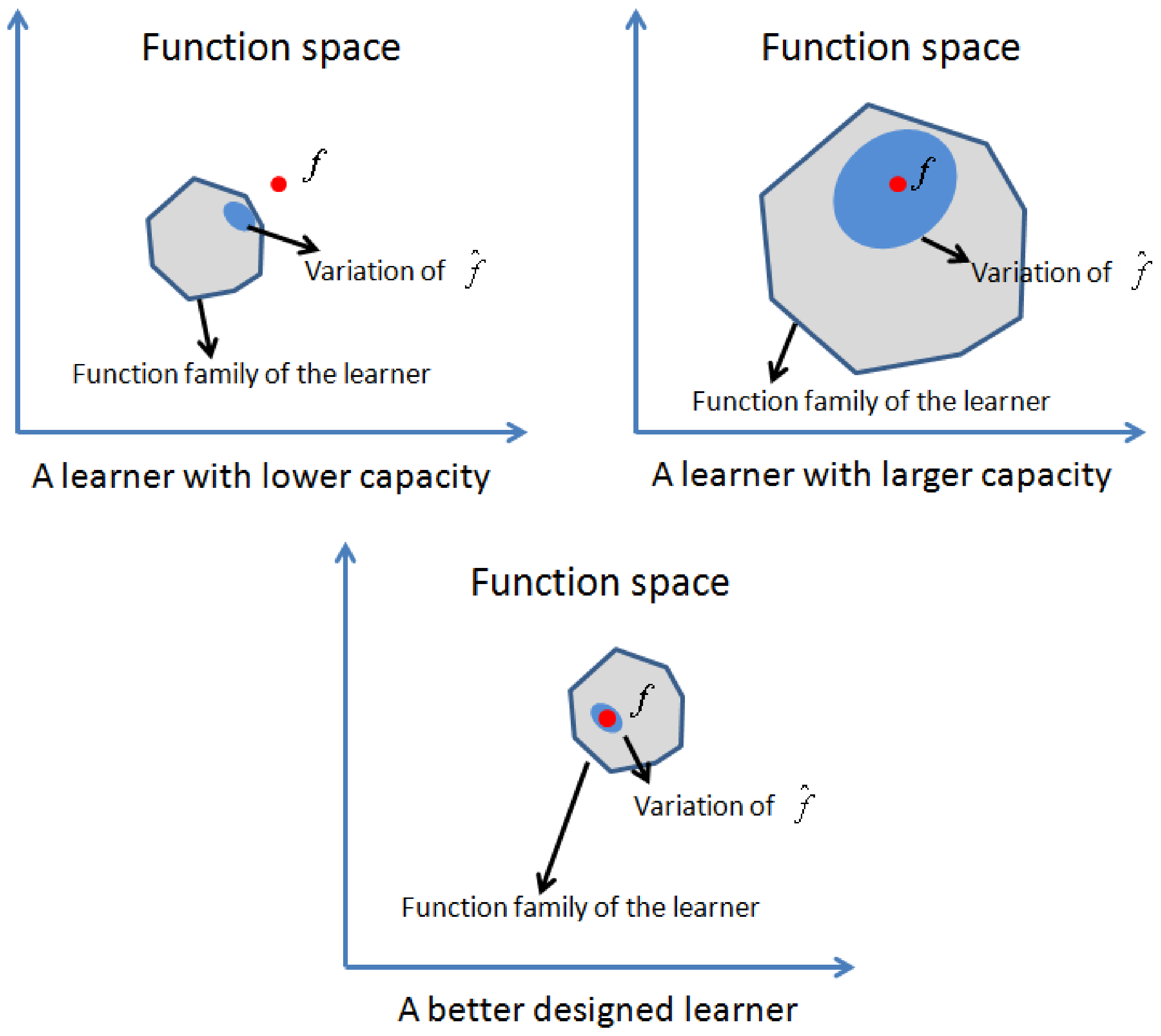
Variance

$$\mathrm{Var}[\hat{\theta}] = E[(\hat{\theta} - E[\hat{\theta}])^2] = E[\hat{\theta}^2] - E[\hat{\theta}]^2$$

- Variance typically decreases as the size of the train set increases
- Both bias and variance are the sources of estimation errors

$$\mathrm{MSE} = E[(\hat{\theta} - \theta)^2] = \mathrm{Bias}(\hat{\theta})^2 + \mathrm{Var}[\hat{\theta}]$$

- Increasing the capacity of a learner may also increase variance, although it has better chance to cover the true function

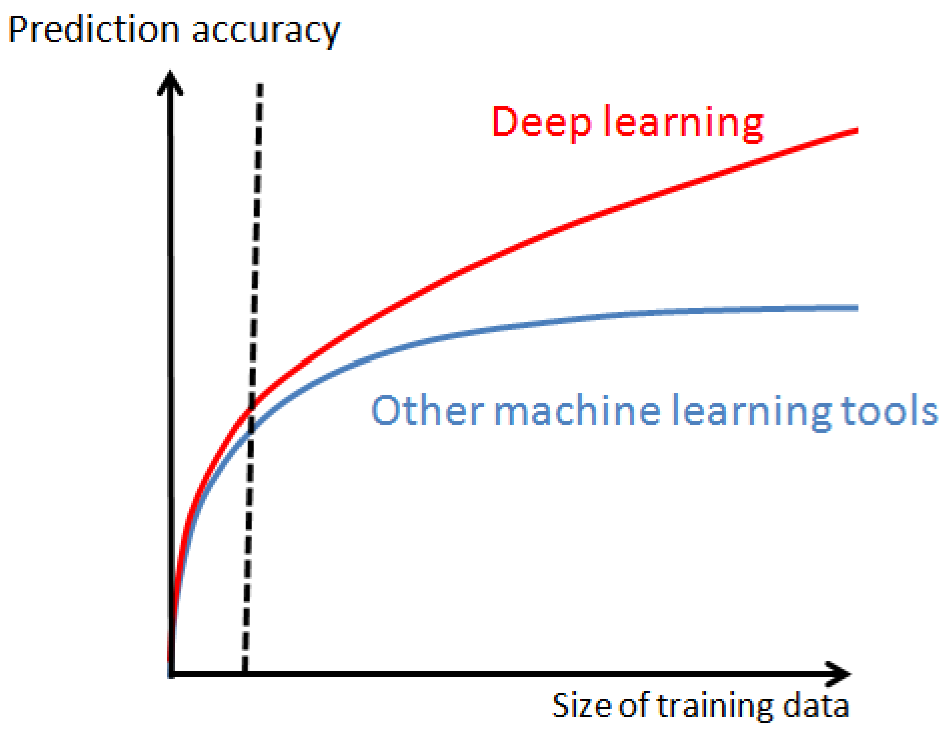

Function space
f
Variation of $\hat{f}$
Function family of the learner
A learner with lower capacity
Function space
f
Variation of $\hat{f}$
Function family of the learner
A learner with larger capacity
Function space
f
Variation of $\hat{f}$
Function family of the learner
A better designed learner

Issues to be concerned in machine learning

- Effective optimization methods and models to address the underfitting problem
- How to balance the trade-off between capacity and generalization?
- How to effectively reduce capacity (which means also reducing estimation variance) without increasing the bias much?
- For machine learning with big training data, how to effectively increase capacity to cover or get closer to the true function to be estimated?

Open discussion

- Why does deep learning have different behavior than other machine learning methods for large scale training?



Discriminative model

- Directly model P(y|x) and decision boundaries
- Learn the discriminative functions g_k (x)

$$y = \arg\max_k g_k(\mathbf{x})$$

- In the linear case, $g_k(\mathbf{x}) = \mathbf{w}_k^t \mathbf{x}$
- P(y|x) can be estimated from the linear discriminant functions

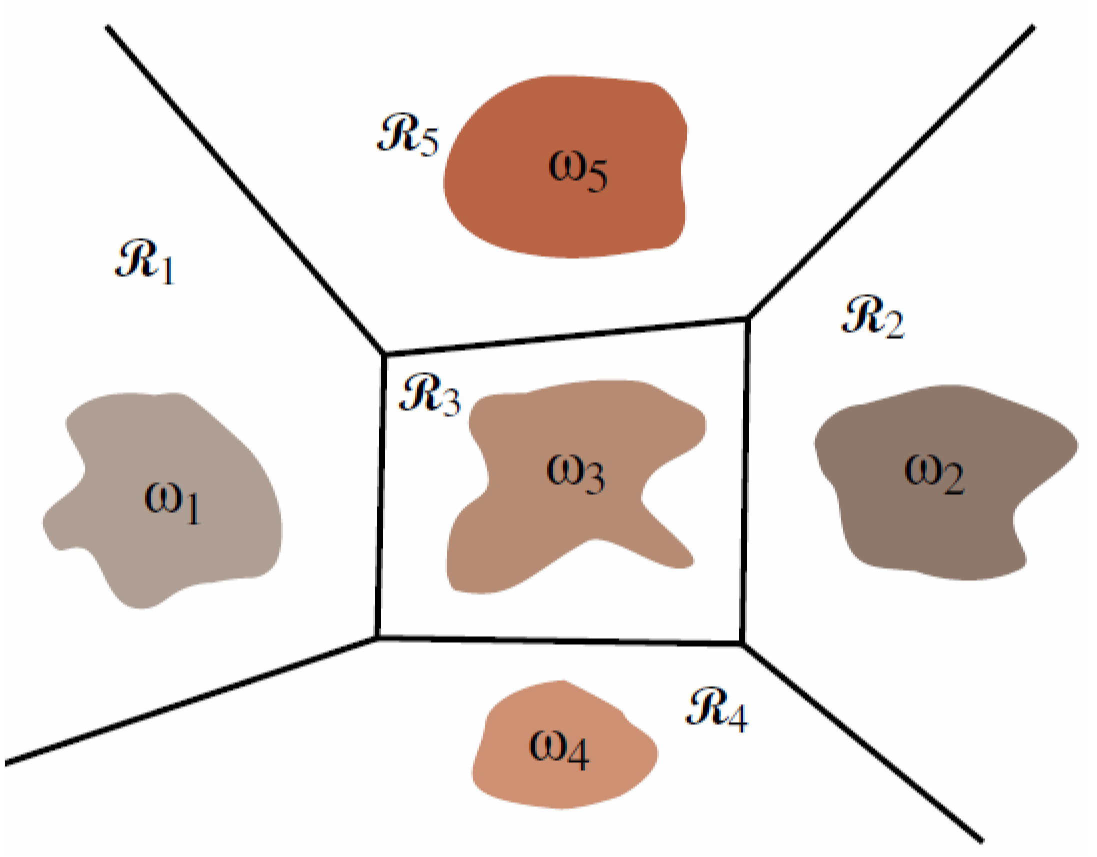
$$P(y = j|\mathbf{x}) = \frac{e^{\mathbf{w}_j^t \mathbf{x}}}{\sum_{k=1}^{K} e^{\mathbf{w}_k^t \mathbf{x}}}$$

It is also called softmax function

- Examples: SVM, boosting, K-nearest-neighbor

Discriminative model

- It is easier for discriminative models to fit data



Discriminative model

- Parameter $\theta = \{\mathbf{w}_k\}$ can be estimated from maximizing the data likelihood

$$\hat{\theta} = \arg\max_{\theta} P(\mathcal{D}_n|\theta) = \arg\max_{\theta} \prod_{i=1}^{n} P(y_n^{(\text{train})}|\mathbf{x}_n^{(train)}, \theta)$$

- Maximum a posteriori (MAP) estimation

$$\theta = \arg\max_{\theta} p(\theta|\mathcal{D}_n) = \arg\max_{\theta} \log P(\mathcal{D}_n|\theta) + \log p(\theta)$$

- According to the Bayes' rule, i.e., $p(\theta|\mathcal{D}_n) = P(\mathcal{D}_n|\theta)p(\theta)/P(\mathcal{D}_n)$,

$$\theta = \arg\max_{\theta} \log P(\mathcal{D}_n|\theta) + \log p(\theta)$$

$$\theta = \arg\max_{\theta} \sum_{i=1}^{n} \log P(y_n^{(\text{train})}|\mathbf{x}_n^{(\text{train})}, \theta) + \log p(\theta)$$

- prior p(θ) corresponds to a regularizer, e.g.

$$p(\theta) = e^{-\lambda||\theta||^2}$$

Generative model

- Estimate the underlying class conditional probability densities p(x|y = k) and then construct the classifier using the Bayesian decision theory

$$P(y = k|\mathbf{x}) = \frac{p(\mathbf{x}|y = k)P(y = k)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|y = k)P(y = k)}{\sum_{k'=1}^{K} p(\mathbf{x}|y = k')P(y = k')}$$

- p(x|y) and P(y) are parameterized by θ
- Prior P(y) can be used to model the dependency among predictions, such as the segmentation labels of pixels or predictions of speech sequences.
- It is more difficult to model class conditional probability densities. However, it also adds stronger regularization to model fitting, since the learned model not only needs to predict class labels but also generate the input data.
- It is easier to add domain knowledge when desgining the models of p(x|y)

Supervised and unsupervised learning

- Supervised learning: the goal is to use input-label pairs, (x; y) to learn a function f that predicts a label (or a distribution over labels) given the input, $\hat{y} = f(\mathbf{x})$

- Unsupervised learning: no label or other target is provided. The data consists of a set of examples x and the objective is to learn about the statistical structure of x itself.

- Weakly supervised learning: the training data contains (x; y) pairs as in supervised learning, but the labels y are either unreliably present (i.e. with missing values) or noisy (i.e. where the label given is not the true label)

Unsupervised learning

- Find the “best” representation of data that reserves as much information about x as possible while being “simpler” than x. Taking linear case as an example

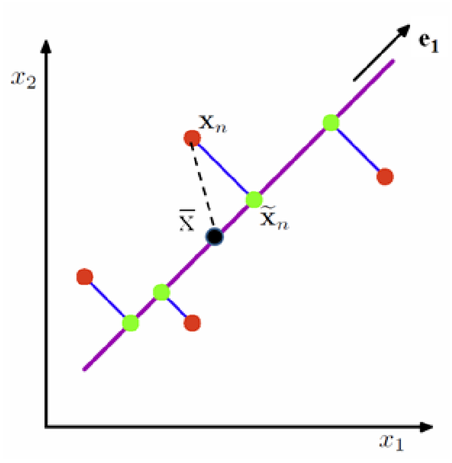
$$\tilde{\mathbf{x}} = a_0 + \sum_{i=1}^{d'} a_i \mathbf{e}_i$$

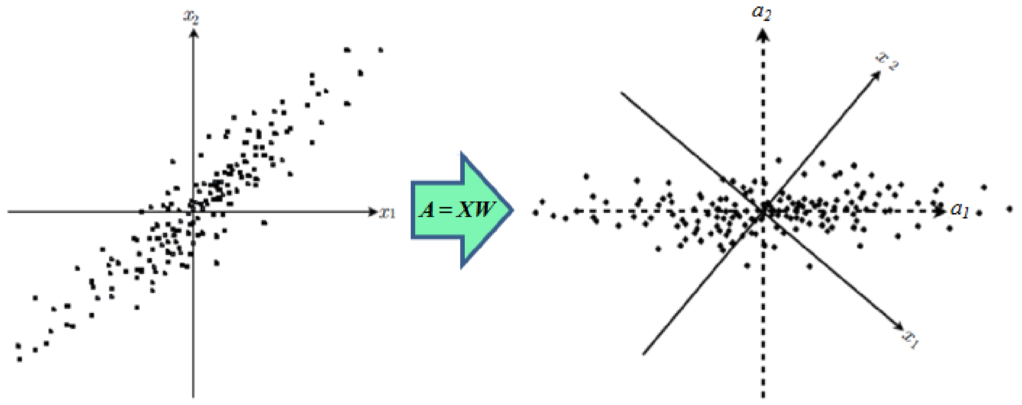
  - Lower dimensional representation: $d' < d$
  - Sparse representation: the number of non-zero a_i is small
  - Independent representation: disentangle the sources of variations underlying the data distributions such that the dimensions of the representation are statistically independent, i.e. a_i and a_j are statistically independent

- Deep learning is to learn data representation, but in a nonlinear and hierarchical way

Principal Component Analysis (PCA)

- There are *n* d-dimensional samples $\mathbf{x}_1$, … , $\mathbf{x}_n$.
- PCA seeks a principal subspace spanned by d’ (d’ < d) orthonormal vectors $\mathbf{e}_1$, …, $\mathbf{e}_{d'}$, such that
    - the projected samples ($\tilde{\mathbf{x}}_k = a_0 + \sum_{i=1}^{d'} a_{ki}\mathbf{e}_i$) onto this subspace has maximum variance; or equivalently
    - the mean squared distance between the samples and their projections are minimized.
    - The projections $\{a_{ki}\}$ are uncorrelated (i.e. independent if data distribution is assumed as Gaussian distribution)





Formulation of PCA

$$\arg\max_{\{\mathbf{e}_i\}} \sum_{k=1}^{n} \|\tilde{\mathbf{x}}_k - \bar{\mathbf{x}}\|^2$$

or

$$\arg\min_{\{\mathbf{e}_i\}} \sum_{k=1}^{n} \|\mathbf{x}_k - \tilde{\mathbf{x}}_k\|^2$$

$(\|\mathbf{x}_k - \bar{\mathbf{x}}\|^2 = \|\mathbf{x}_k - \tilde{\mathbf{x}}_k\|^2 + \|\tilde{\mathbf{x}}_k - \bar{\mathbf{x}}\|^2)$

Zero- and One-Dimensional Representations by PCA

- zero-dimensional representation: use a single vector $\mathbf{x}_0$ to represent all the samples and minimize the squared-error function

$$J_0(\mathbf{x}_0) = \sum_{k=1}^{n} \|\mathbf{x}_0 - \mathbf{x}_k\|^2$$

The solution is $\mathbf{x}_0 = \bar{\mathbf{x}} = \frac{1}{n}\sum_{k=1}^{n} \mathbf{x}_k$

- One-dimensional representation: project the data to a line running through the sample mean, $\tilde{\mathbf{x}}_k = \bar{\mathbf{x}} + a_{k1}\mathbf{e}_1$ and minimize the squared-error criterion function

$$J_1(a_{11}, \ldots, a_{n1}, \mathbf{e}_1) = \sum_{k=1}^{n} \|\tilde{\mathbf{x}}_k - \mathbf{x}_k\|^2$$

Find the Principal Components a_{k1}

$$\begin{aligned} & J_1(a_{11}, \ldots, a_{n1}, \mathbf{e}_1) \\ = & \sum_{k=1}^{n} \|(\bar{\mathbf{x}} + a_{k1}\mathbf{e}_1) - \mathbf{x}_k\|^2 = \sum_{k=1}^{n} \|a_{k1}\mathbf{e}_1 - (\mathbf{x}_k - \bar{\mathbf{x}})\|^2 \\ = & \sum_{k=1}^{n} a_{k1}^2 \|\mathbf{e}_1\|^2 - 2\sum_{k=1}^{n} a_{k1}\mathbf{e}^t(\mathbf{x}_k - \bar{\mathbf{x}}) + \sum_{k=1}^{n} \|\mathbf{x}_k - \bar{\mathbf{x}}\|^2 \end{aligned}$$

Since $\mathbf{e}_1$ is a unit vector, $\|\mathbf{e}_1\| = 1$. To minimize J_1, set $\frac{\partial J_1}{\partial a_{k1}} = 0$ and we have

$$a_{k1} = \mathbf{e}_1^t(\mathbf{x}_k - \bar{\mathbf{x}})$$

- We obtain a least-squares solution by projecting the vector $\mathbf{x}_k$ onto the line in the direction of $\mathbf{e}_1$ passing through the mean.

Find the Optimal Projection Direction $\mathbf{e}_1$

$$\begin{aligned}
J_1(\mathbf{e}_1) &= \sum_{k=1}^{n} a_{k1}^2 - 2\sum_{k=1}^{n} a_{k1}^2 + \sum_{k=1}^{n} \|\mathbf{x}_k - \bar{\mathbf{x}}\|^2 \\
&= -\sum_{k=1}^{n} \left[\mathbf{e}_1^t(\mathbf{x}_k - \bar{\mathbf{x}})\right]^2 + \sum_{k=1}^{n} \|\mathbf{x}_k - \bar{\mathbf{x}}\|^2 \\
&= -\sum_{k=1}^{n} \mathbf{e}_1^t(\mathbf{x}_k - \bar{\mathbf{x}})(\mathbf{x}_k - \bar{\mathbf{x}})^t\mathbf{e}_1 + \sum_{k=1}^{n} \|\mathbf{x}_k - \bar{\mathbf{x}}\|^2 \\
&= -\mathbf{e}_1^t\mathbf{S}\mathbf{e}_1 + \sum_{k=1}^{n} \|\mathbf{x}_k - \bar{\mathbf{x}}\|^2
\end{aligned}$$

Find the Optimal Projection Direction $\mathbf{e}_1$

- $\mathbf{S} = \sum_{k=1}^{n} (\mathbf{x}_k - \bar{\mathbf{x}})(\mathbf{x}_k - \bar{\mathbf{x}})^t$ is the scatter matrix
- $\mathbf{e}_1^t \mathbf{S} \mathbf{e}_1 = \sum_{k=1}^{n} a_{k1}^2 \; (\sum_{k=1}^{n} a_{k1} = 0)$ is the variance of the projected data
- The vector $\mathbf{e}_1$ that minimizes J_1 also maximizes $\mathbf{e}_1^t \mathbf{S} \mathbf{e}_1$, subject to the constraint that $\|\mathbf{e}_1\| = 1$

Lagrange Optimization

- Seek the position $\mathbf{x}_0$ of an extremum of a scalar-valued function f ($\mathbf{x}$) subject to the constrain that g($\mathbf{x}$) = 0

- First from the Lagrangian function

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$$

where λ is a scalar called the Lagrange *undetermined multiplier*.

- Convert into an unconstrained problem by taking the derivative,

$$\frac{\partial L(\mathbf{x}, \lambda)}{\partial \mathbf{x}} = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} + \lambda \frac{\partial g(\mathbf{x})}{\partial \mathbf{x}} = 0$$

- Solve **x** and λ considering g($\mathbf{x}$) = 0

Find the Optimal Projection Direction $\mathbf{e}_1$

- Use the method of Lagrange multipliers to maximize the $\mathbf{e}_1^t \mathbf{S} \mathbf{e}_1$ subject to the constraint that $\|\mathbf{e}_1\| = 1$,

$$u = \mathbf{e}_1^t \mathbf{S} \mathbf{e}_1 - \lambda(\mathbf{e}_1^t \mathbf{e}_1 - 1),$$

$$\frac{\partial u}{\partial \mathbf{e}_1} = 2\mathbf{S}\mathbf{e}_1 - 2\lambda \mathbf{e}_1 = 0.$$

- Setting the gradient vector equal to zero, we see that $\mathbf{e}_1$ must be an eigenvector of the scatter matrix

$$\mathbf{S}\mathbf{e}_1 = \lambda \mathbf{e}_1$$

- Since $\mathbf{e}_1^t \mathbf{S} \mathbf{e}_1 = \lambda \mathbf{e}_1^t \mathbf{e}_1 = \lambda$, to maximize $\mathbf{e}_1^t \mathbf{S} \mathbf{e}_1$, we select the eigenvector with the largest eigenvalue

d'-Dimensional Representation by PCA

- d'-dimensional representation: $\tilde{\mathbf{x}}_k = \bar{\mathbf{x}} + \sum_{i=1}^{d'} a_{ki}\mathbf{e}_i$

- Mean-squared criterion function:

$$J_{d'} = \sum_{k=1}^{n} \left\| \left(\bar{\mathbf{x}} + \sum_{i=1}^{d'} a_{ki}\mathbf{e}_i \right) - \mathbf{x}_k \right\|^2$$

- Define additional principal components in an incremental fashion by choosing each new direction minimizing *J* amongst all possible directions orthogonal to those already considered

- To minimize $J_{d'}$, $\mathbf{e}_1$, …, $\mathbf{e}_{d'}$ are the d' eigenvectors of the scatter matrix with the largest eigenvalues. a_{ki} are the principal components of samples.

d'-Dimensional Representation by PCA

- Since the scatter matrix is real and symmetric, its eigenvectors are orthogonal and its eigenvalues are nonnegative.

- The squared error:

$$J_{d'} = -\sum_{i=1}^{d'} \lambda_i + \sum_{k=1}^{n} \|\mathbf{x}_k - \bar{\mathbf{x}}\|^2$$

$$J_d = 0 \Rightarrow \sum_{k=1}^{n} \|\mathbf{x}_k - \bar{\mathbf{x}}\|^2 = \sum_{i=1}^{d} \lambda_i$$

$$J_{d'} = \sum_{i=d'+1}^{d} \lambda_i$$

Variance of Data Captured by the PCA Subspace

- The variance of data projected onto the first d’ eigenvectors is $\sum_{i=1}^{d'} \lambda_i$

- Measure how much variance has been captured by the first d’ eigenvectors:

$$\frac{\sum_{i=1}^{d'} \lambda_i}{\sum_{j=1}^{d} \lambda_j}$$

Covariance Matrix of Principal Components

- The correlation between projections on $\mathbf{e}_i$ and $\mathbf{e}_j$ ($i \neq j$) is

$$\sum_{k=1}^{n} a_{ki} a_{kj} = \sum_{k=1}^{n} \mathbf{e}_i^t (\mathbf{x}_k - \bar{\mathbf{x}})(\mathbf{x}_k - \bar{\mathbf{x}})^t \mathbf{e}_j = \mathbf{e}_i^t \mathbf{S} \mathbf{e}_j = \lambda_i \mathbf{e}_i^t \mathbf{e}_j = 0$$

- The covariance matrix of samples in the PCA subspace is diag $[\lambda_1, \ldots, \lambda_{d'}]$

Summary of PCA

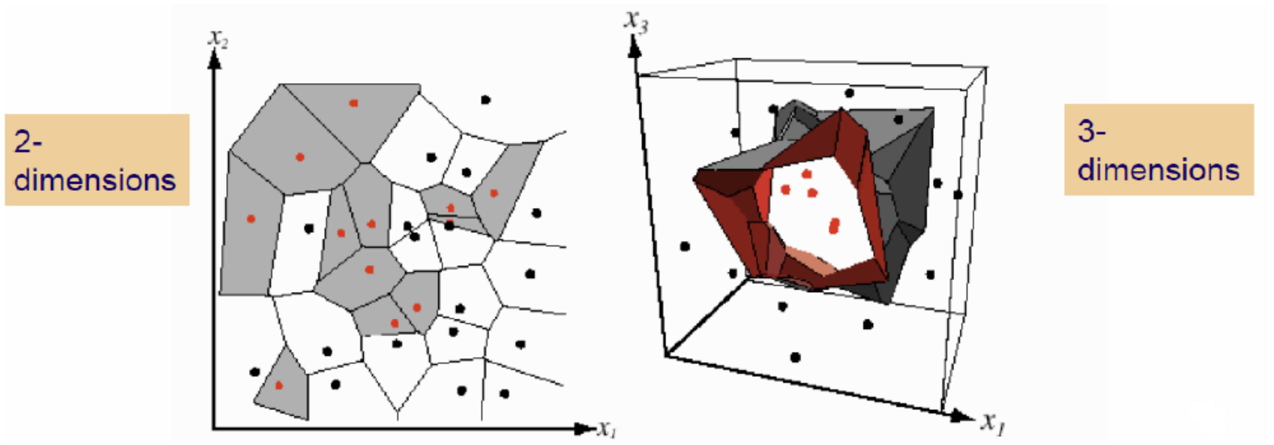
- The principal subspace is spanned by d' orthonormal vectors **e**$_1$, …, **e**$_{d'}$ which are computed as the d' eigenvectors of the scatter matrix **S** with the largest eigenvalues $\lambda_1, \ldots, \lambda_{d'}$.
- The principal components a_{ki} of the samples are computed as $a_{ki} = \mathbf{e}_i^t(\mathbf{x}_k - \bar{\mathbf{x}})$
- The variance of the projected samples onto this principal subspace is $\sum_{i=1}^{d'} = \lambda_i$
- The mean squared distance between the samples and their projections are $\sum_{i=d'+1}^{d} = \lambda_i$
- PCA disentangles the factors of variation underlying the data, assuming such variation is a Gaussian distribution
- We are interested in learning representations that disentangle more complicated forms of feature dependencies

Smoothness Prior

- Shallow models assume smoothness prior on the prediction function to be learned, i.e.

$$f^*(\mathbf{x}) \approx f^*(\mathbf{x} + \epsilon)$$

  where ϵ is a small change.
- K-nearest neighbor predictors assume piecewise constant
  - For classification and K = 1, f (**x**) is the output class associated with the nearest neighbor of **x** in the training set
  - For regression, f (**x**) is the average of the outputs associated with the K nearest neighbors of **x**
  - The number of distinguishable regions cannot be more than the number of training examples



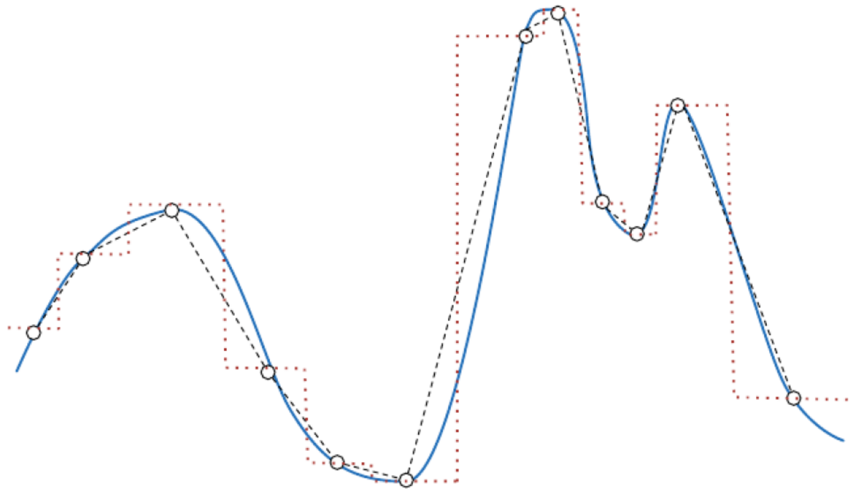
Interpolation with Kernel

$$f(\mathbf{x}) = b + \sum_{i=1}^{n} \alpha_i K(\mathbf{x}, \mathbf{x}_i)$$

K is a kernel function, e.g., the Gaussian kernel

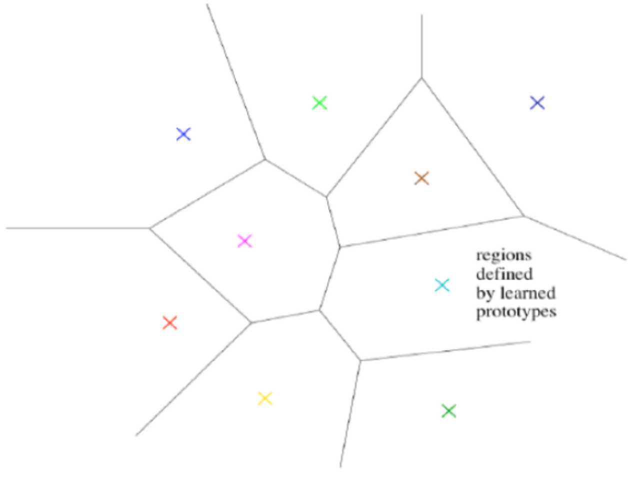
$$K(\mathbf{u}, \mathbf{v}) = N(\mathbf{u} - \mathbf{v}; 0, \sigma^2 I)$$

- b and α_i can be learned by SVM
- Treat each $\mathbf{x}_i$ is a template and the kernel function as a similarity function that matches a template and a test example



Local Representation

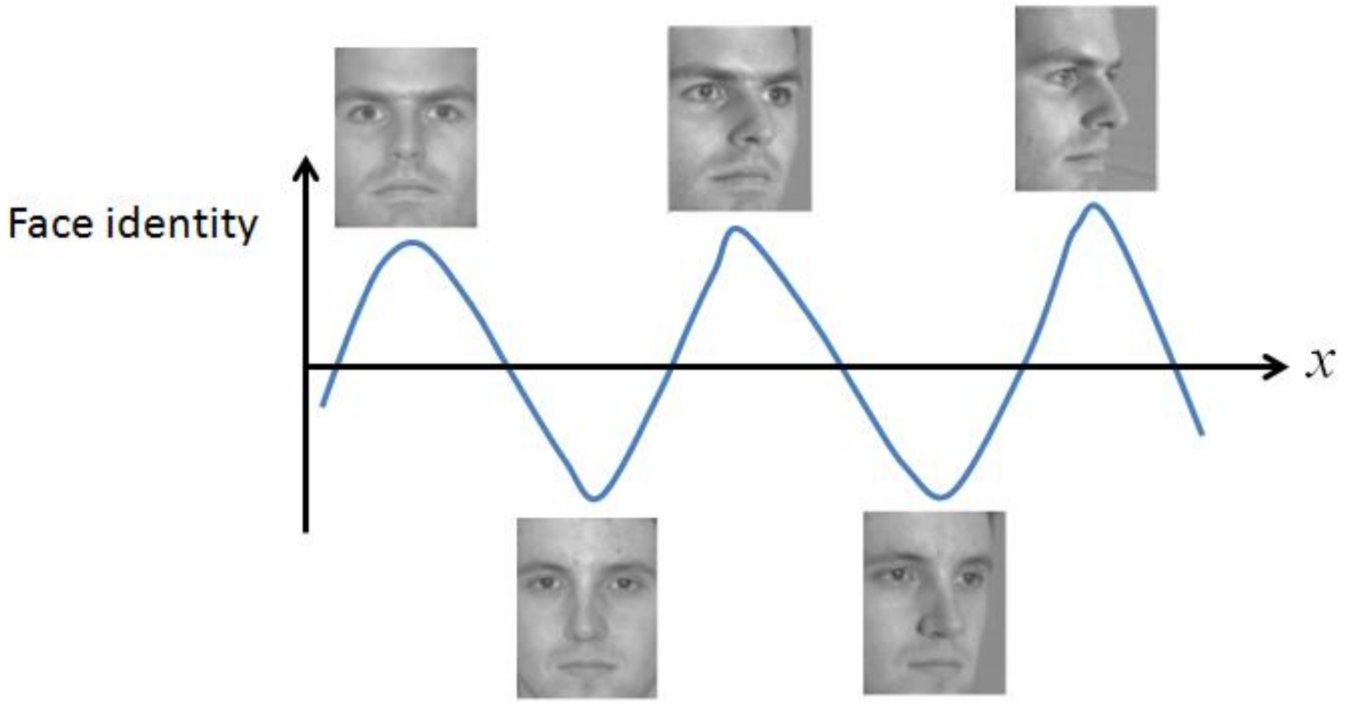
- One can think of the training samples as control knots which locally specify the shape of the prediction function
- The smoothness prior only allows the learner to generalize locally. If ($\mathbf{x}_i$, y_i) is a supervised training example and $\mathbf{x}_i$ is a near neighbor of $\mathbf{x}$, we expect that f ($\mathbf{x}$) $\approx y_i$. Better generalization can be obtained with more neighbors.
- To distinguish O(N) regions in the input space, shallow models require O(N) examples (and typically there are O(N) parameters associated with the O(N) regions).



LOCAL PARTITION

Local Representation

- If the function is complex, more regions and more training samples are required.
- The representation learned by deep models can be generalized non-locally



Next Lecture

- Deep Feedforward Network

Acknowledgement

Some of the materials in these slides are drawn inspiration from:

- Ian Goodfellow, MIT University, Deep Learning Slides
- Hung-yi Lee, National Taiwan University, Machine Learning and having it Deep and Structured course
- Xiaogang Wang, The Chinese University of Hong Kong, Deep Learning Course
- Fei-Fei Li, Standord University, CS231n Convolutional Neural Networks for Visual Recognition course

Questions?

Thank You !

WeChat Group for Deep Learning