CS4487 Machine Learning

Lecture 1: Introduction

- Course Information
  - Content
    - Lecture: Learning algorithm & Mathmetical Proof
    - Tutorial: Use learning algorithms on small examples
    - HW: four => derive and analyze machine learning algorithms with mathematical tools
  - Grade:
    - Coursework(40)
      - 10(tutorial)
      - 30(assignment)
    - midterm(30)
    - final(30)
  - Book:
    - Pattern Recognition & Machine Learning (C M B)
    - Convex Optimization (S B & L V)
    - The Matrix Cookbook (K B P)
    - Linear Algebra and Its Applications
    - Deep Learning
  - Syllabus:
    - Classification, regression, clustering, dimensionality reduction and deep learning
    - Model selection, model evaluation, regularization, design of experiments
  - Prerequisites:
    - Linear Algebra
    - Calculus
    - Probability and Statistics
    - Optimization
    - Python
  - Some problem
    - derivative of matrix with respect to another
    - maximum likelihood estimation(MLE) & maximum a posteriori estimation(MAP), and their relationship
    - difference between gradient descent and stochastic gradient descent
    - sigular value decomposition(SVD) and principal component analysis (PCA) and their relationship
- Introduction
  - type
    - By I/O
      - Supervised: Learning to predict (with a set of input and output)
      - Unsupervised: Learning to organize and represent

        - By Parameter
            - Non-parametric: storing the training data
            - Parametric: Using an algorithm to adapt the parameters in a mathematical or statistical model given training data
    - Sample => feature => training=>learned model
    - AI > ML (Data) > DL (multi-layer)
- Math
    - Vector Space (column vector)
    - Matrix
        - Square, diagonal, identity
        - Transpose, trace, inverse, **determinant**
        - **eigenvalue, eigenvectors**, orthogonality
    - Probability Concepts
        - Probability Distribution
        - Random Variable (X): Maps each element(get 2 balls) of the sample space to a value x(2) in X(called the range of the random variable)
        - Marginalization
        - Conditioning
        - Bayes Rules
        - Expectations
        - Classical Distributions (Bernoulli, Multinomial, Gaussian)

Lecture 2: KNN and Naive Bayes

Classification

- Definition:

Definition: The Classification Task

Given a feature vector $\mathbf{x} \in \mathbb{R}^n$ that describes an object that belongs to one of $|\mathcal{Y}|$ classes from the set $\mathcal{Y}$, predict which class the object belongs to

- Classifier Learning

Definition: Classifier Learning

Given a data set of example pairs $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)}), i = 1, \ldots, m\}$ where $\mathbf{x}^{(i)} \in \mathbb{R}^n$ is a feature vector and $y^{(i)} \in \mathcal{Y}$ is a class label, learn a function $f : \mathbb{R}^n \mapsto \mathcal{Y}$ that accurately predicts the class label y for any feature vector $\mathbf{x}$

- Error and Accuracy

Definition: Classification Error Rate

Given a data set of example pairs $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)}), i = 1, \ldots, m\}$ and a function $f : \mathbb{R}^n \mapsto \mathcal{Y}$, the classification error rate of f on $\mathcal{D}$ is

$$\text{Err}(f, \mathcal{D}) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{I}[y^{(i)} \neq f(\mathbf{x}^{(i)})]$$

Definition: Classification Accuracy Rate

Given a data set of example pairs $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)}), i = 1, \ldots, m\}$ and a function $f : \mathbb{R}^n \mapsto \mathcal{Y}$, the classification accuracy rate of f on $\mathcal{D}$ is

$$\text{Acc}(f, \mathcal{D}) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{I}[y^{(i)} = f(\mathbf{x}^{(i)})]$$

KNN

- Definition

The KNN classifier is a non-parametric classifier that simply stores the training data $\mathcal{D}$ and classifies each new instance $\mathbf{x}$ using a majority vote over its set of k nearest neighbors $\mathcal{N}_k(\mathbf{x})$ computed using any distance function $d : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$

KNN Classification Function

$$f_{\text{KNN}}(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{i \in \mathcal{N}_k(\mathbf{x})} \mathbb{I}[y^{(i)} = y]$$

Use of KNN requires choosing the distance function d and the number of neighbors k

- Notice that $argmax$ will traverse all the element in a set
- Work with any distance function d satisefying non-negativity$d(x, x') > 0$ and identity of indiscernibles $d(x, x) = 0$
- Alternatively, KNN can work with any similarity function ss atisfying non-negativity $s(x, x') \geq 0$ that attains it's maximum on indiscernibles $s(x, x) = max_{x'} s(x, x')$
- Distance Metrics

Definition: Minkowski Distance (ℓ_p-norms)

Given two data vectors $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^n$, the Minkowski Distance with parameter p (the ℓ_p-norm) is a proper metric defined as follows:

$$d_p(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_p = \left(\sum_{j=1}^{n} |x_j - x'_j|^p \right)^{1/p}$$

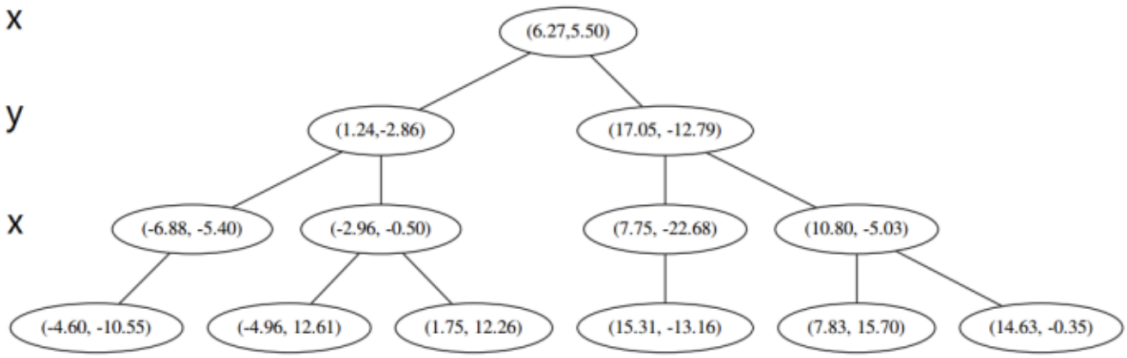
Special cases include Euclidean distance ($p = 2$), Manhattan distance ($p = 1$) and Chebyshev distance ($p = \infty$)

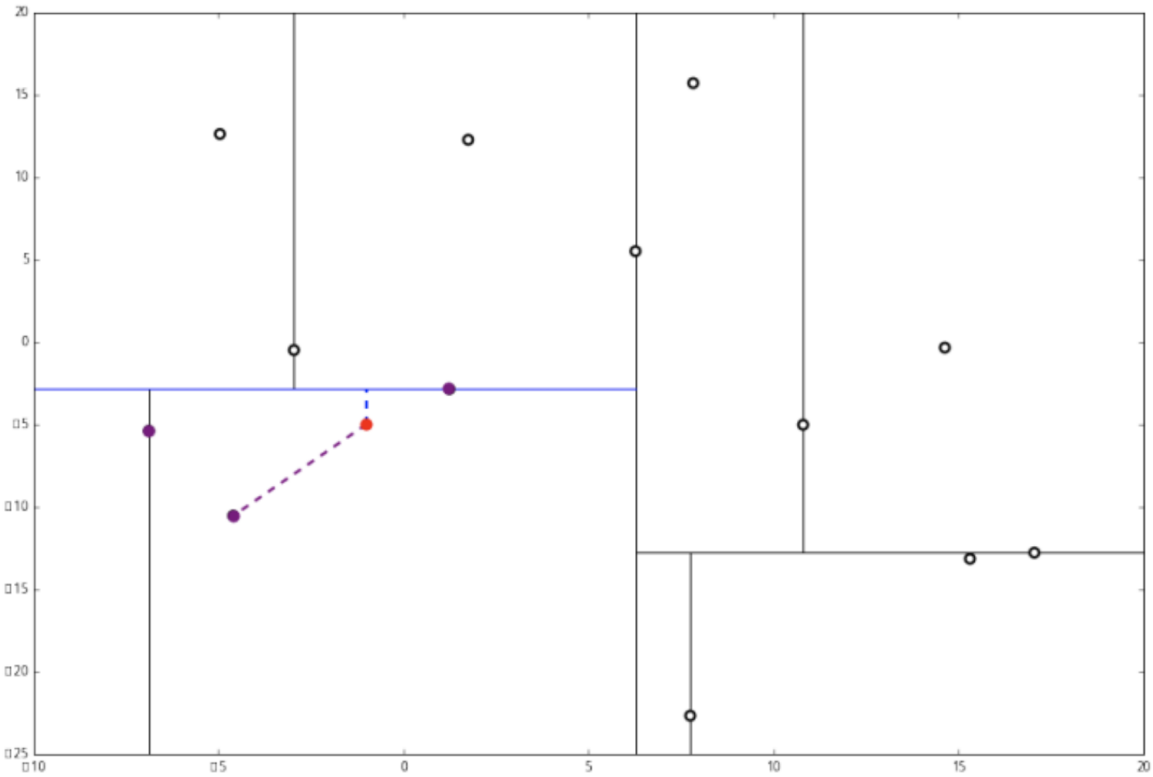
- Variant
  - If there are 4 very far point and 1 very close point => vote to very far
  - For each class y, traverse all the k closest points, add all weights that belongs to the class and divide by some of weight of all k closest points, where the weight of i_{th} point is defined as $exp(-\alpha d_i)$. Then, choose the class with the maximum value
    - Instead of giving all of the k neighbors equal weight in the majority vote, a distance-weighted majority can be used:

$$f_{\text{KNN}}(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \frac{\sum_{i \in \mathcal{N}_k(\mathbf{x})} w_i \mathbb{I}[y^{(i)} = y]}{\sum_{i \in \mathcal{N}_k(\mathbf{x})} w_i},$$
$$w_i = \exp(-\alpha d_i)$$

    - Instead of a brute force nearest neighbor search, data structures like k-d trees can be constructed over the training data that support nearest neighbor search with lower computational complexity

- **K-d tree** to reduce the complexity: Make use of the division line
  - Point to search: (-1, -5)





- Trade off
  - Low bias: Converges to the correct decision surface as data goes to infinity
  - High variance: Lots of variability in the decision surface when amount of data is low
  - Curse of dimensionality: Everything is far from everything else in high dimensions
  - Space and time complexity: Need to store all training data. Take time at run time to compute

Bayes Optimal Classifiers

- Probabilistic Classification
  - Ture probability of seeing a data case that belongs to $p(Y = y) = \phi_y$
  - Ture probability density of seeing a data vector $x \in \mathbb{R}^n$ that belongs to class y is $p(X = x|Y = y) = \varphi_y(x)$
  - To calculate with vector x the probability of any one of y :

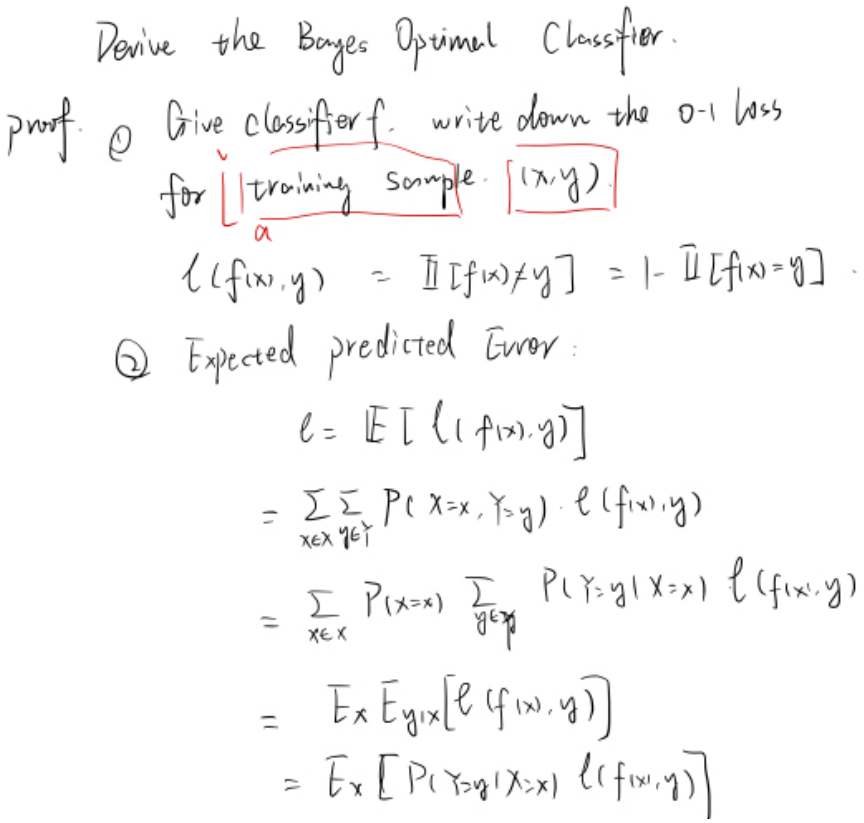
$$p(Y = y|X = x) = \frac{p(X = x|Y = y)p(Y = y)}{\sum_{\tilde{y}\in Y} p(X = x|Y = \tilde{y})p(Y = \tilde{y})} = \frac{\varphi_y(x)\phi_y}{\sum_{\tilde{y}\in Y} \varphi_{\tilde{y}}(x)\phi_{\tilde{y}}}$$

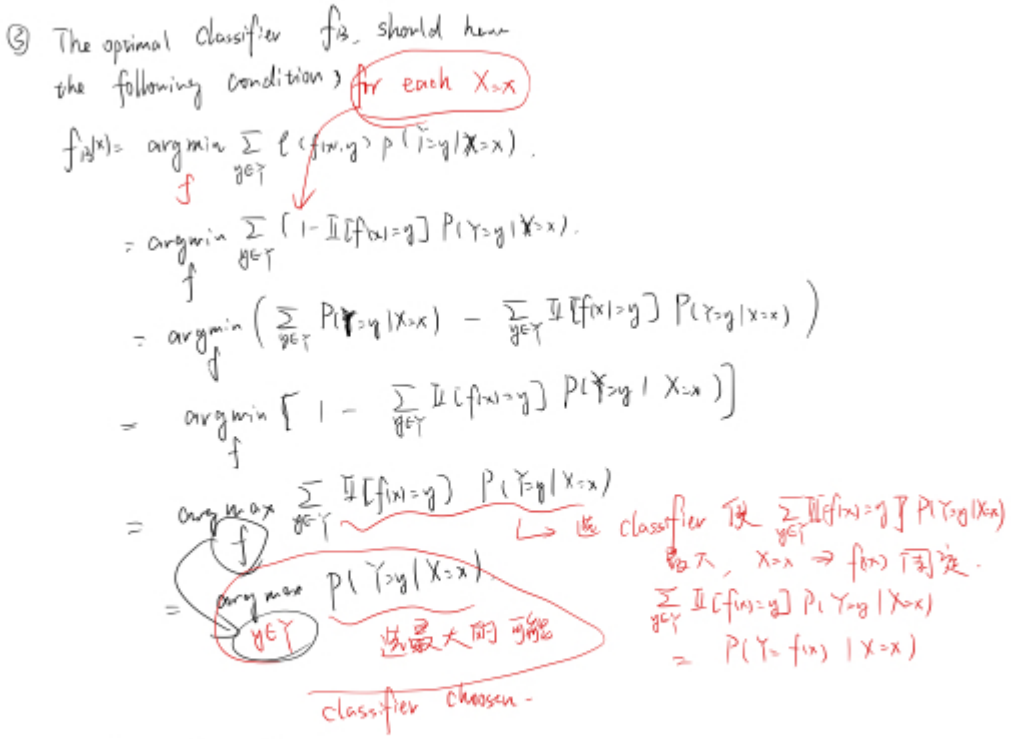
- Bayes Optimal Classifiers

$$f_B(x) = argmax_{y \in Y}\ p(Y = y|X = x) = argmax_{y \in Y}\ \varphi_y(x)\phi_y$$

$$MIN\ Error = 1 - E_x[max_{y \in Y} p(Y = y|X = x)]$$

  - $p(X = x)$ can be treated as a constant for specific input vector
  - E_x represents expectation with the condition $X = x$
  - Proof:





The classifier:

$$f_B(x) = \arg\max_{y\in Y} P(Y=y \mid X=x)$$

$$= \arg\max_{y\in Y} \frac{P(X=x \mid Y=y)\cdot P(Y=y)}{P(X=x)} \quad P(X=x) \text{ fixed}$$

$$= \arg\max_{y\in Y} P(X=x \mid Y=y)\, P(Y=y)$$

$$= \arg\max_{y\in Y} \varphi_y(x)\, \pi_y.$$

$\varphi_y(x)$: difficult to estimate. (limited sample).

- Not useful in practice: difficult to estimate $\varphi_y(x)$ in high dimensions

Lecture03: Naive Bayes, Linear Discriminant Analysis and Logistic Regression

Naive Bayes

- Naive bayes classifier approximates the Bayes optimal classifier using a simple form for the functions $\varphi_y(x)$
- Assumes that all of the data dimensions are statistically independent given the value of the class variable

$$\varphi_y(x) = p(X = x|Y = y) = \Pi_{j=1}^n p(X_j = x_j|Y = y) = \Pi_{j=1}^n \varphi_{j|y}(x_j)$$

- The general form for the classification function:

$$f_{NB}(x) = argmax_{y \in Y}\ \phi_y\ \Pi_{j=1}^n \varphi_{j|y}(x_j)$$

- The functions $\varphi_{j|y}(x_j) = p(X_j = x_j|Y = y)$ are called *marginal class conditional distributions*
  - For real valued x_j, $p(X_j = x_j|Y = y)$ is typically modeled as a Gaussian density $\varphi_{j|y}(x_j) = N(x_j; \mu_{j|y}, \sigma^2_{j|y})$
  - For binary valued x_j ,$p(X_j = x_j|Y = y)$ is typically modeled as a Bernoulli distribution $\varphi_{j|y}(x_j) = \theta_{j|y}^{x_j}(1 - \theta_{j|y})^{(1-x_j)}$
  - For general categorical values x_j, $p(X_j = x_j|Y = y)$ is typically modeled as a categorical distribution $\varphi_{j|y}(x_j) = \Pi_{x \in X_j} \theta_{x,j|y}^{I[x_j=x]}$
- The problem to maximize is how to choose a correct ϕ_y and $\varphi_{j|y}(x_j)$ => using the *maximum likelihood* over $D = \{(x^{(i)}, y^{(i)}), i = 1, \ldots, m\}$
  - **posterior probability**: $P(Y = y|X = x)$, ... : likelihood $P(X = x|Y = y)$, Prior probability: $P(Y = y)$
  - maximum likelihood: 让取样获得的大前提($P(D)$)可能性最大, D 为数据集

    - Class probabilities: $\phi_y = \frac{1}{m}\sum_{i=1}^{m} \mathbb{I}[y^{(i)} = y]$
    - Gaussian: $\mu_{j|y} = \frac{\sum_{i=1}^m \mathbb{I}[y^{(i)}=y]x_j^{(i)}}{\sum_{i=1}^m \mathbb{I}[y^{(i)}=y]}, \qquad \sigma^2_{j|y} = \frac{\sum_{i=1}^m \mathbb{I}[y^{(i)}=y](x_j^{(i)}-\mu_{j|y})^2}{\sum_{i=1}^m \mathbb{I}[y^{(i)}=y]}$
    - Bernoulli: $\theta_{j|y} = \frac{\sum_{i=1}^m \mathbb{I}[y^{(i)}=y]x_j^{(i)}}{\sum_{i=1}^m \mathbb{I}[y^{(i)}=y]}$
    - Categorical: $\theta_{x,j|y} = \frac{\sum_{i=1}^m \mathbb{I}[y^{(i)}=y \cap x_j^{(i)}=x]}{\sum_{i=1}^m \mathbb{I}[y^{(i)}=y]}$

  - for Bernoulli distribution, x_j is 0 or 1
  - proof of the class probabilities ϕ_y and the categorical probability $\theta_{x,j|y}$ => see lecture note
- Geometric Interpretation: for normal distribution of x, and binary classification problem we have =>

$$
\begin{aligned}
f_{\mathrm{NB}}(\mathbf{x}) &= \arg\max_{y\in\mathcal{Y}} \phi_y \prod_{j=1}^{n} \varphi_{j|y}(x_j) \\
&= \arg\max_{y\in\mathcal{Y}} \log(\phi_y) + \sum_{j=1}^{n} \log(\varphi_{j|y}(x_j)) \\
&= \arg\max_{y\in\mathcal{Y}} \log(\phi_y) + \sum_{j=1}^{n} \left(-\frac{1}{2}\log(2\pi\sigma_{j|y}^2) - \frac{(x_j-\mu_{j|y})^2}{2\sigma_{j|y}^2} \right)
\end{aligned}
$$

- The decision boundary consists of the set of points $\mathbf{x}$ where:

$$
\begin{aligned}
&\log(\phi_0) + \sum_{j=1}^{n} \left(-\frac{1}{2}\log(2\pi\sigma_{j|0}^2) - \frac{1}{2\sigma_{j|0}^2}(x_j-\mu_{j|0})^2 \right) \\
&- \log(\phi_1) - \sum_{j=1}^{n} \left(-\frac{1}{2}\log(2\pi\sigma_{j|1}^2) - \frac{1}{2\sigma_{j|1}^2}(x_j-\mu_{j|1})^2 \right) = 0
\end{aligned}
$$

- It's easy to see that the decision boundary is a quadratic function of $\mathbf{x}$ with the form: $\sum_{j=1}^{n} \left(a_j x_j^2 + b_j x_j \right) + c = 0$
- In the multi-class case, the decision boundary is piece-wise quadratic

- Laplace Smoothing
  - The maximum likelihood :

$$
\because \theta_{x,j|y} = \frac{\sum_{i=1}^{m} \mathbb{I}[y^{(i)}=y \cap x_j^{(i)}=x]}{\sum_{i=1}^{m} \mathbb{I}[y^{(i)}=y]}
$$

  - If the training set is small, it's possible that $I[x_j^{(i)} = x] = 0, for\ any\ i \in [1,m], i\ is\ integer$
  - Cause the probability to be zero, which is overfitting
  - Laplace smoothing method:

$$\theta_{x,j|y} = \frac{\sum_{i=1}^{m} \mathbb{I}[y^{(i)} = y \cap x_j^{(i)} = x] + \alpha}{\sum_{i=1}^{m} \mathbb{I}[y^{(i)} = y] + \alpha|\mathcal{X}|},$$

    - When $\alpha = 1$ it's called Laplace smoothing
    - it can be shown that Laplace smoothing is the posterior mean with Dirichlet distribution as the conjugate prior for Categorical distribution
- Trade offs
    - Speed: learning and classification are fast
    - Storage: $O(n)$ parameters, large compression of the training data
    - Interpretability: good because $\varphi_y(x)$ is the class conditional averages
    - Accuracy: feature independence assumption and canocial forms for class-conditional marginals will rarely be correct for real-world problems, leading to lower accuracy
    - Data: Some care is needed in the estimation of parameters in the discrete case when data is scarce(small data set)

Linear Discriminant Analysis

- Linear Discriminant Analysis
    - A different approximation to the Bayes optimal classifier for real-valued data (instead of Maximum Likelihood)
    - Use dependent Normal Distribution in Naive Bayes, LDA assumes $\varphi_y(x) = N(x; \mu_y, \sum)$, where $\sum$ is covariance matrix

$$\varphi_y(\mathbf{x}) = \frac{1}{|(2\pi)^n \boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_y)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_y)\right)$$

        - The classification function is $f_{\text{LDA}}(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \varphi_y(\mathbf{x})\phi_y$

To get an intuition for what a multivariate Gaussian is, consider the simple case where $n = 2$, and where the covariance matrix Σ is diagonal, i.e.,

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} \qquad \Sigma = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}$$

In this case, the multivariate Gaussian density has the form,

$$\begin{aligned} p(x; \mu, \Sigma) &= \frac{1}{2\pi \begin{vmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{vmatrix}^{1/2}} \exp\left(-\frac{1}{2} \begin{bmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{bmatrix}^T \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}^{-1} \begin{bmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{bmatrix} \right) \\ &= \frac{1}{2\pi(\sigma_1^2 \cdot \sigma_2^2 - 0 \cdot 0)^{1/2}} \exp\left(-\frac{1}{2} \begin{bmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{bmatrix}^T \begin{bmatrix} \frac{1}{\sigma_1^2} & 0 \\ 0 & \frac{1}{\sigma_2^2} \end{bmatrix} \begin{bmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{bmatrix} \right), \end{aligned}$$

where we have relied on the explicit formula for the determinant of a 2×2 matrix[3], and the fact that the inverse of a diagonal matrix is simply found by taking the reciprocal of each diagonal entry. Continuing,

$$\begin{aligned} p(x; \mu, \Sigma) &= \frac{1}{2\pi\sigma_1\sigma_2} \exp\left(-\frac{1}{2} \begin{bmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{bmatrix}^T \begin{bmatrix} \frac{1}{\sigma_1^2}(x_1 - \mu_1) \\ \frac{1}{\sigma_2^2}(x_2 - \mu_2) \end{bmatrix} \right) \\ &= \frac{1}{2\pi\sigma_1\sigma_2} \exp\left(-\frac{1}{2\sigma_1^2}(x_1 - \mu_1)^2 - \frac{1}{2\sigma_2^2}(x_2 - \mu_2)^2 \right) \\ &= \frac{1}{\sqrt{2\pi}\sigma_1} \exp\left(-\frac{1}{2\sigma_1^2}(x_1 - \mu_1)^2 \right) \cdot \frac{1}{\sqrt{2\pi}\sigma_2} \exp\left(-\frac{1}{2\sigma_2^2}(x_2 - \mu_2)^2 \right). \end{aligned}$$

- By using maximum likelihood, which reduces to using sample estimates:

  - Class probabilities: $\phi_y = \frac{1}{m} \sum_{i=1}^{m} \mathbb{I}[y^{(i)} = y]$
  - Class means: $\boldsymbol{\mu}_y = \frac{\sum_{i=1}^{m} \mathbb{I}[y^{(i)}=y] \mathbf{x}^{(i)}}{\sum_{i=1}^{m} \mathbb{I}[y^{(i)}=y]}$
  - Shared covariance: $\boldsymbol{\Sigma} = \frac{1}{m} \sum_{i=1}^{m} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_{y^{(i)}})(\mathbf{x}^{(i)} - \boldsymbol{\mu}_{y^{(i)}})^T$

- similar to the one dimensional case, the decision boundary consists of points x where:

$$\begin{aligned} &\log(\phi_0) - \frac{1}{2}\log|(2\pi)^n \boldsymbol{\Sigma}| - \frac{1}{2}(\mathbf{x} - \mu_0)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \mu_0) \\ &- \log(\phi_1) + \frac{1}{2}\log|(2\pi)^n \boldsymbol{\Sigma}| + \frac{1}{2}(\mathbf{x} - \mu_1)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \mu_1) = 0 \end{aligned}$$

  - We can cancel a large number of terms because of the common covariance matrix and obtain the following result:

$$\log(\phi_0) - \log(\phi_1) - 0.5\mu_0^T \boldsymbol{\Sigma}^{-1}\mu_0 + 0.5\mu_1^T \boldsymbol{\Sigma}^{-1}\mu_1 + (\mu_0 - \mu_1)^T \boldsymbol{\Sigma}^{-1}\mathbf{x} = 0$$

- Trade-offs

- Speed: quadratic dependence on n dimension makes LDA slower than Naive Bayes
- Storage: $O(n^2)$ parameters, good when $n << m$
- Interpretability: Good interpretability
- Accuracy: Rarely be correct in real-world problems. However, the induced linear decision boundaries can often perform reasonably well
- Data: LDA will generally need more data than NB since it needs to estimate the *O*(*n*2) parameters in the pooled covariance matrix

Logistic Regression

- Generative vs. Discriminative Classifiers
  - Naive Bayes and LDA are said to be *generative* classifier because they explicityly model the joint distribution $p(x, y)$ of the data vectors x and the lablel y
  - To build a probabilistic classifier, all we really need to model is $p(y|x)$
  - classifiers based on directly estimating $p(y|x)$ are called *discriminative* classifiers because they ignore the distribution of x and focus only on the class labels y
- Logistic Regression
  - In binary case, it directly models the decision boundary using a linear function:

$$\log p(Y = 1|\mathbf{x}) - \log p(Y = 0|\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

$$p(Y = 1|\mathbf{x}) = \frac{1}{1 + \exp(-(\mathbf{w}^T\mathbf{x} + b))}$$

  - Now the classifunction is : $f_{LR} = argmax_{y \in Y} p(Y = y|x)$:
    - the standard logistic function, or the sigmoid function is defined as:

$$g(z) = \frac{1}{1 + e^{-z}}$$
$$g'(z) = -(\frac{1}{1 + e^{-z}})^2 * (-e^{-z})$$
$$= g(z)(1 - g(z))$$

- The learning logistic regression
  - logistic regression model parameters $\theta = \{w, b\}$ are selected to optimize the conditional log likelihood of the labels given the data set $D = \{(x^{(i)}, y^{(i)}), i = 1, \ldots, m\}$:

$$\boldsymbol{\theta}^\star = \arg\max_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}|\mathcal{D}) = \arg\max_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^{m} \log p(Y = y^{(i)}|\mathbf{X} = \mathbf{x}^{(i)})$$

$$= \arg\max_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log g(\boldsymbol{\theta}^T\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log \left(1 - g(\boldsymbol{\theta}^T\mathbf{x}^{(i)})\right)$$

    - cannot be maximized analytically. Learning the model parameters requires numerical optimization methods
- Multiclass Logistic Regression
    - Logistic regression extends to multiclass case:

$$p(Y = y|\mathbf{x}) = \frac{\exp(\mathbf{W}_y\mathbf{x} + b_y)}{\sum_{\tilde{y}\in\mathcal{Y}} \exp(\mathbf{W}_{\tilde{y}}\mathbf{x} + b_{\tilde{y}})}$$

    - why different from binary case?
- Geometry
    - **Explicityly designed** to have a **linear decision boundary in the binary case**
    - In the multiclass case, the decision boundary is piece-wise linear
    - Same representational capacity as LDA
- Trade-offs
    - Speed: Faster than NB and LDA in classification time, numberical optimization is slower than naive bayes
    - Storage: O(n) parameters
    - Interpretability: The “importance” of different feature variables x^j can be understood in terms of their weights w^j
    - Accuracy: Tends to be better in high dimensions with limited data compared to LDA. Much worse than KNN in low dimensions with **lots of data and non-linear decision boundaries**
- Probability view of logistic function(sigmoid function)

2.2 Probabilistic View on the Sigmoid Function

Let's write the general form of the sigmoid function

$$F(z;\mu,s)=\frac{1}{1+e^{-\frac{z-\mu}{s}}}, \tag{25}$$

where μ and $s>0$ are location and scale parameters, respectively.

For a function $F(z)$ to be a legitimate cumulative distribution function (CDF)[4], it must satisfy four properties:

- Non-decreasing;
- Right-continuous (*i.e.*, $\lim_{z\to c^+} F(z)=F(c)$);
- $\lim_{z\to-\infty} F(z)=0$;
- $\lim_{z\to+\infty} F(z)=1$.

It is not hard to see that the sigmoid function satisfies all of the properties, and the corresponding probability density function (PDF) is

$$f(z)=F'(z)=\frac{e^{-r}}{s\left(1+e^{-r}\right)^2} \quad \text{and} \quad r=\frac{z-\mu}{s}, \tag{26}$$

which is known as the logistic distribution[5].

Lecture 04: Gradient Descent and Stochastic Gradient Descent

Gradient Descent

- Gradient Descent
  - typical form of machine learning problem

$$\min_{\boldsymbol{\theta}} \ \ell(\boldsymbol{\theta};\mathcal{D})=\min_{\boldsymbol{\theta}} \ \frac{1}{m}\sum_{i=1}^{m}\ell(\boldsymbol{\theta};\mathbf{x}^{(i)},y^{(i)}),$$

where $\ell:\mathbb{R}^n\mapsto\mathbb{R}$ is differentiable but not necessarily convex, and $\mathcal{D}=\{(\mathbf{x}^{(i)},y^{(i)}),i=1,\ldots,m\}$ is a finite training set

  - for logistic regression, trying to minimize the negative log likelihood:

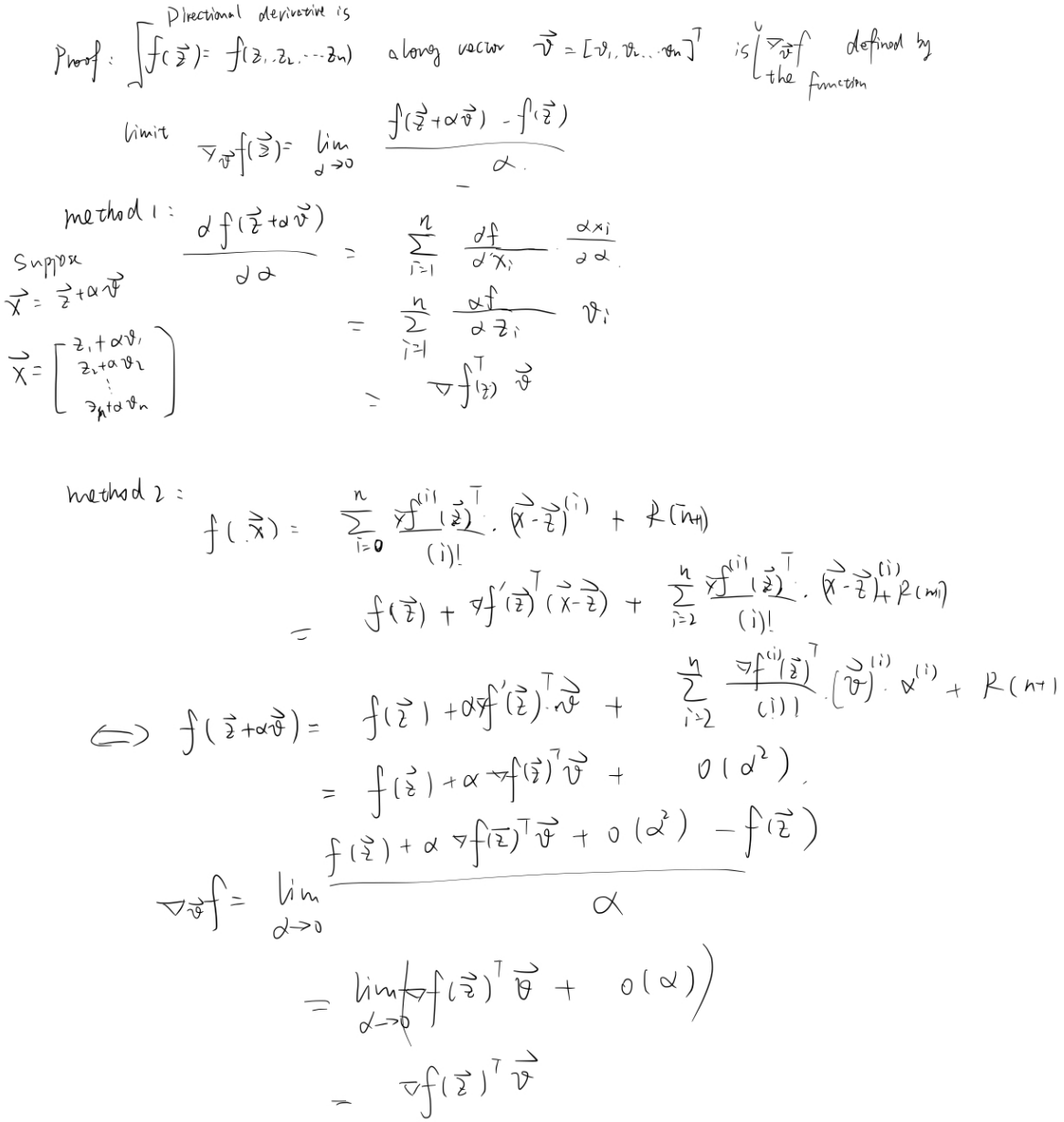
$$\min_{\boldsymbol{\theta}} \ -\frac{1}{m}\sum_{i=1}^{m}\log p(y^{(i)}|\mathbf{x}^{(i)};\boldsymbol{\theta})$$

- choose initial point $\theta^{(0)} \in \mathbb{R}^n$ and repeats

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \nabla \ell(\boldsymbol{\theta}^{(t)}; \mathcal{D}) = \boldsymbol{\theta}^{(t)} - \alpha \frac{1}{m} \sum_{i=1}^{m} \nabla \ell(\boldsymbol{\theta}^{(t)}; \mathbf{x}^{(i)}, y^{(i)})$$

How to stop? $\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}\| \leq \epsilon$ or $\|\nabla \ell(\boldsymbol{\theta}^{(t)}; \mathcal{D})\| \leq \epsilon$, where ϵ is a small positive constant, *e.g.*, 10^{-6}

- why gradient descent works



Reason. Pick any direction $\mathbf{d}$ with $\|\mathbf{d}\| = 1$. The rate of change at $\boldsymbol{\theta}$ is

$$\nabla \ell(\boldsymbol{\theta})^T \mathbf{d} \leq \|\nabla \ell(\boldsymbol{\theta})\| \|\mathbf{d}\| = \|\nabla \ell(\boldsymbol{\theta})\|.$$

If we set $\mathbf{d} = \frac{\nabla \ell(\boldsymbol{\theta})}{\|\nabla \ell(\boldsymbol{\theta})\|}$, then

$$\nabla \ell(\boldsymbol{\theta})^T \mathbf{d} = \|\nabla \ell(\boldsymbol{\theta})\|$$

- Moreover, a negative gradient step can decrease the loss function

  Reason. Let $\boldsymbol{\theta}^{(0)}$ be any initial point. Using the first order Taylor expansion for candidate point $\boldsymbol{\theta}^{(1)} = \boldsymbol{\theta}^{(0)} - \alpha \nabla \ell(\boldsymbol{\theta}^{(0)})$, we have

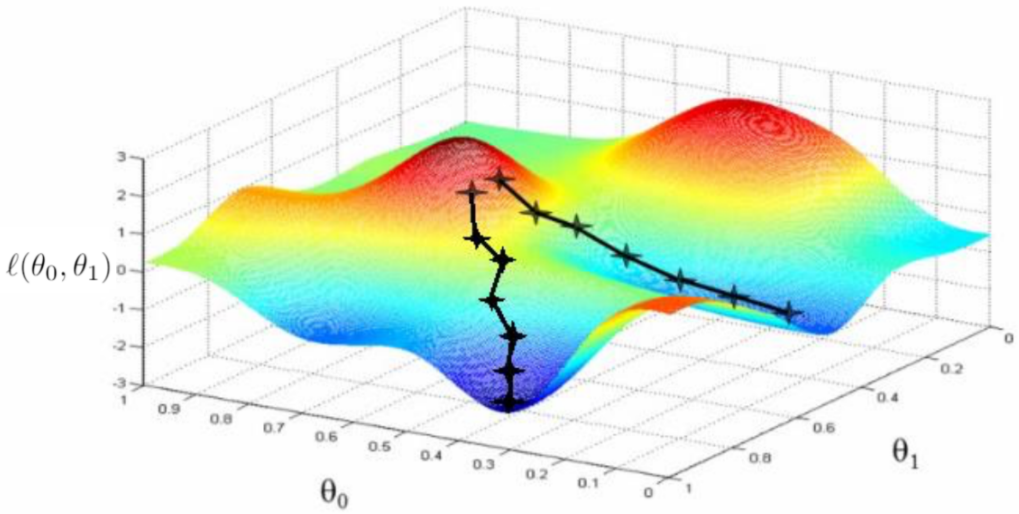
  $$\ell(\boldsymbol{\theta}^{(1)}) - \ell(\boldsymbol{\theta}^{(0)}) = -\alpha \|\nabla \ell(\boldsymbol{\theta}^{(0)})\|^2 + O(\alpha^2).$$

  Hence, if $\nabla \ell(\boldsymbol{\theta}^{(0)}) \neq 0$ and α is sufficiently small, we have

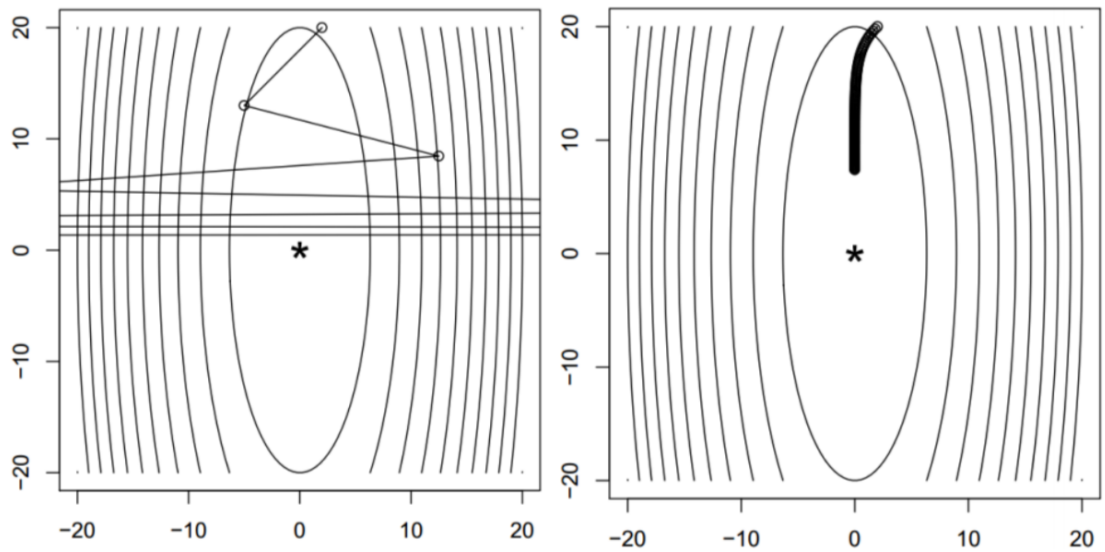
  $$\ell(\boldsymbol{\theta}^{(1)}) < \ell(\boldsymbol{\theta}^{(0)}).$$

  Therefore, for sufficiently small α, $\boldsymbol{\theta}^{(1)}$ is an improvement over $\boldsymbol{\theta}^{(0)}$

Initial point matters.



Learning rate matters.

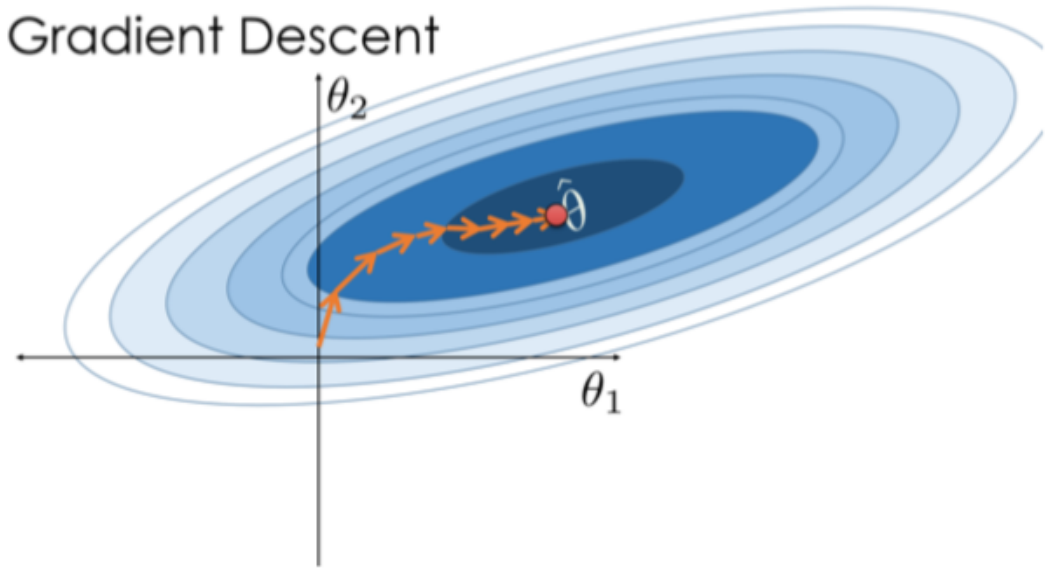


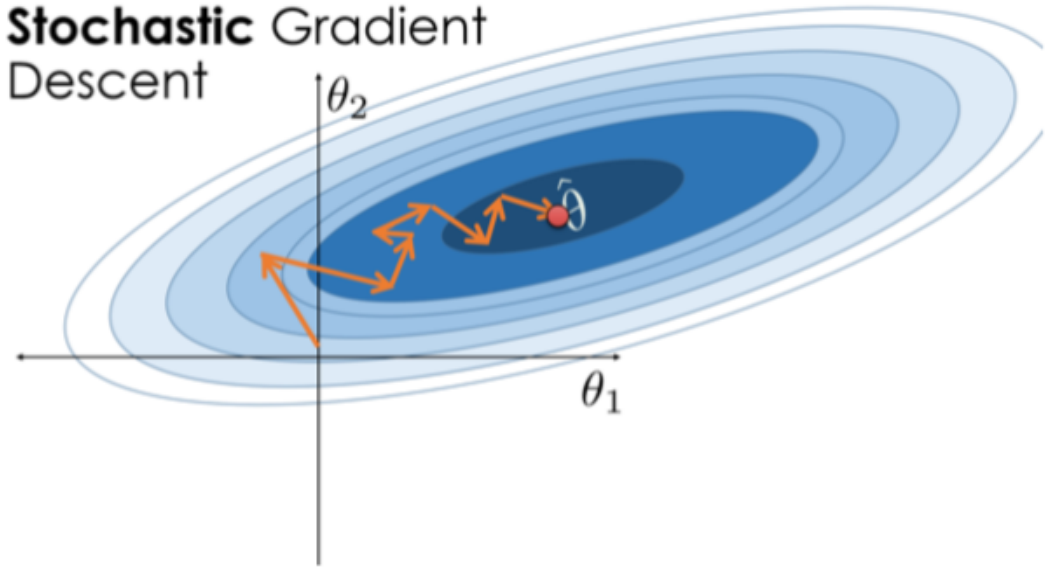
- Stochastic Gradient Descent
  - Compute loss for all the training data to get average takes $O(m)$ times
  - Can only choose one training data(i.e. one vecvtor) to (point)estimate the true gradient

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \nabla \ell(\boldsymbol{\theta}^{(t)}; \mathbf{x}^{(i)}, y^{(i)})$$

  - The stochastic gradient $\nabla \ell(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)})$ is the unbiased estimate of gradient $\nabla \ell(\boldsymbol{\theta}; \mathcal{D})$

$$\mathbb{E}_i[\nabla \ell(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)})] = \frac{1}{m} \sum_{i=1}^{m} \nabla \ell(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)}) = \nabla \ell(\boldsymbol{\theta}; \mathcal{D})$$


Gradient Descent
θ_2
$\hat{\theta}$
θ_1


Stochastic Gradient Descent
θ_2
$\hat{\theta}$
θ_1

Pros

- It is easy to fit into memory and computationally fast due to a single training sample being processed at each iteration
- The steps have oscillations, which may help the algorithm get out of bad local minima

Cons

- The steps are noisy. The objective does not always decrease for each step and it may take longer to achieve convergence
- It loses the advantage of vectorized operations as it deals with only a single example at a time

- Mini-Batch Gradient Descent
  - use a subset of samples => interval estimate => mean value theorem

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \nabla \ell(\boldsymbol{\theta}^{(t)}; \mathcal{B}) = \boldsymbol{\theta}^{(t)} - \alpha \frac{1}{|\mathcal{B}|} \sum_{(\mathbf{x},y)\in\mathcal{B}} \nabla \ell(\boldsymbol{\theta}^{(t)}; \mathbf{x}, y)$$

    - Mini-batch gradient descent and its variants are extremely popular and extensively used in machine learning

- Gradient Descent of Logistic Regression

3.2 Gradient Descent for Logistic Regression

Proof. Let's start by working with just one training example (x, y), and take derivatives to derive the stochastic gradient ascent rule:

$$\begin{aligned}\frac{\partial}{\partial\theta_j}\ell(\theta) &= \left(y\frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)}\right)\frac{\partial}{\partial\theta_j}g(\theta^T x)\\ &= \left(y\frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)}\right)g(\theta^T x)(1-g(\theta^T x))\frac{\partial}{\partial\theta_j}\theta^T x\\ &= \big(y(1-g(\theta^T x)) - (1-y)g(\theta^T x)\big)\,x_j\\ &= (y-g(\theta^T x))x_j. \end{aligned} \tag{30}$$

This therefore gives us the stochastic gradient ascent rule

$$\theta_j^{(t+1)} = \theta_j^{(t)} + \alpha(y^{(i)} - g((\theta^{(t)})^T x^{(i)}))x_j^{(i)}, \tag{31}$$

from which we can easily derive the gradient ascent rule

$$\theta_j^{(t+1)} = \theta_j^{(t)} + \alpha\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - g((\theta^{(t)})^T x^{(i)}))x_j^{(i)}. \tag{32}$$

Finally, the corresponding vectorized gradient ascent rule is

$$\theta^{(t+1)} = \theta^{(t)} + \alpha\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - g((\theta^{(t)})^T x^{(i)}))x^{(i)}. \tag{33}$$

□

Lecture05: Linear Regression

Regression

- Regression: Given a feature vector $x \in \mathbb{R}^n$, predict it's corresponding output value $y \in \mathbb{R}$
- Questions:
  - Does size of sample matters? => we prefere larger sample size
  - Does location of points matters? => Samples need be representative or informative
  - Given a set of m points, is curve fitting unique? => Estimating "continuous" model from "discrete" data is all ill-posed problem and can never be "certain"
  - How to choose the best model from infinite curves? => Have some criteria choose prefered model
- Example: House price, weather prediction, age estimation ...
- Regression learning problem:

  Definition: Regression Learning Problem

  Given a data set of example pairs $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)}), i = 1, \ldots, m\}$ where $\mathbf{x}^{(i)} \in \mathbb{R}^n$ is a feature vector and $y^{(i)} \in \mathbb{R}$ is the output, learn a function $f : \mathbb{R}^n \mapsto \mathbb{R}$ that accurately predicts y for any feature vector $\mathbf{x}$

- Error measure:

- MSE:

Definition: Mean Squared Error (MSE)

Given a data set of example pairs $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)}), i = 1, \cdots, m\}$ and a function $f : \mathbb{R}^n \mapsto \mathbb{R}$, the MSE of f on $\mathcal{D}$ is:

$$\text{MSE}(f, \mathcal{D}) = \frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - f(\mathbf{x}^{(i)}))^2$$

Related measures include:
Sum of Squared Errors: $\text{SSE}(f, \mathcal{D}) = m \cdot \text{MSE}(f, \mathcal{D})$
Risidual Sum of Squares: $\text{RSS}(f, \mathcal{D}) = m \cdot \text{MSE}(f, \mathcal{D})$
Root Mean Squared Error: $\text{RMSE}(f, \mathcal{D}) = \sqrt{\text{MSE}(f, \mathcal{D})}$

  - SSE and RSS are the same
- MAE:

Definition: Mean Absolute Error (MAE)

Given a data set of example pairs $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)}), i = 1, \ldots, m\}$ and a function $f : \mathbb{R}^n \mapsto \mathbb{R}$, the MAE of f on $\mathcal{D}$ is:

$$\text{MAE}(f, \mathcal{D}) = \frac{1}{m} \sum_{i=1}^{m} |y^{(i)} - f(\mathbf{x}^{(i)})|$$

Linear Regression

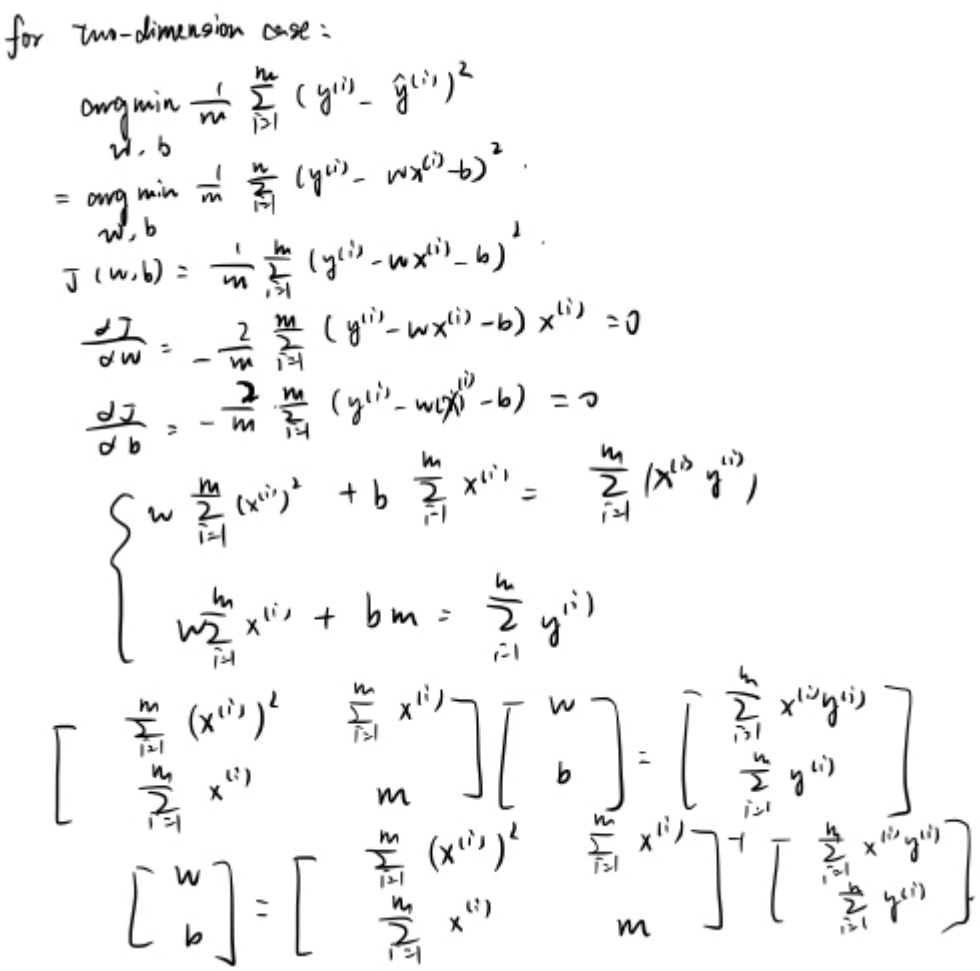
- A parametric regression method that assumes the relationship betweeen y and x is a linear function with parameters $w = [w_1, w_2, \ldots, w_n]^T$ and b

$$f_{\text{Lin}}(\mathbf{x}) = \left(\sum_{j=1}^{n} w_j x_j \right) + b = \mathbf{x}^T \mathbf{w} + b$$

- Ordinary Least Squares (OLS) Linear Regression
  - Minimize the MSE on the training data set:

$$\mathbf{w}^\star, b^\star = \arg\min_{\mathbf{w},b} \frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - (\mathbf{x}^{(i)})^T \mathbf{w} - b)^2$$

- Use derivative:



- General OLS Solution
  - Assume that $X \in \mathbb{R}^{m \times (n+1)}$ is a data matrix with one data case $x^{(i)} \in \mathbb{R}^{n+1}$ per row, where $x_0^{(i)} = 1$ and $y \in \mathbb{R}^m$ is a column vector containing the corresponding outputs. $w \in \mathbb{R}^{n+1}$ where $w_0 = b$
  - Use derivative:

for General OLS:

b is included in the vector $\vec{w}$ → $\vec{w}$ mapping all samples to a value.

$$L(\vec{w}) = \frac{1}{m}(\vec{y}-X\vec{w})^T(\vec{y}-X\vec{w})$$

$$\nabla_{\vec{w}} L(\vec{w}) = \frac{1}{m}\nabla_{\vec{w}}(\vec{y}-X\vec{w})^T(\vec{y}-X\vec{w})$$

$$= \frac{1}{m}\nabla_{\vec{w}}\left(y^Ty - \underset{1\times m}{y^T}\underset{m\times 1}{Xw} - \underset{1\times m}{(Xw)^T}\underset{m\times 1}{y} + (Xw)^TXw\right)$$

$$= \frac{1}{m}\nabla_w(y^Ty - 2y^TXw + w^TX^TXw)$$

$$= \frac{1}{m}\nabla_w(w^TX^TXw - 2y^TXw).$$

$$= \frac{1}{m}(2X^TXw - 2X^Ty) = \frac{2}{m}(X^TXw - X^Ty) = 0$$

$$w = (X^TX)^{-1}X^Ty.$$

Def: ① Since the matrix multiplication is actually linear transformation.

$$\frac{\partial A\vec{x}}{\partial \vec{x}} = \frac{\partial AX}{\partial X}.$$ (the change rates are the same for each vectors)

if $\vec{y}$ is m element vector, $\vec{x}$ is n-element vector.

$$\frac{\partial \vec{y}}{\partial \vec{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_n} & \frac{\partial y_2}{\partial x_n} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

shape: $n\times m$

Therefore: $\frac{\partial AX}{\partial X} = A^T$

② $\alpha = x^TAx$ $x(n\times 1)$ $A(n\times n)$

$$\frac{\partial \alpha}{\partial \vec{x}} = x^T(A+A^T)$$

proof: $\alpha = \sum_{j=1}^{n}\left(\sum_{i=1}^{n} a_{ij}x_i\right)x_j$ $\quad \frac{\partial \alpha}{\partial x_k} = \sum_{j=1}^{n} a_{kj}x_j + \sum_{i=1}^{n} a_{ik}x_i$

$$\frac{\partial \alpha}{\partial \vec{x}} = X^TA^T + X^TA = X^T(A^T+A) \rightarrow \begin{bmatrix} \frac{\partial \alpha}{\partial x_1} \\ \frac{\partial \alpha}{\partial x_2} \\ \vdots \\ \frac{\partial \alpha}{\partial x_m} \end{bmatrix}$$

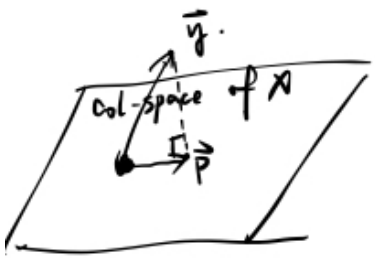
when A is symmetric.

$$\frac{\partial \alpha}{\partial \vec{x}} = 2x^TA$$

- Use projection:

for $Xw = y$, there is no w that fit X into y 100%. if the points in X is discrete and non-linear.

but projection $\vec{p}$ is the best answer. (closest Euclidean Distance). $\hat{y} = \vec{p}$.



$(\vec{y} - \vec{p}) \perp$ Col-space of X.

$\therefore Xw^* = \hat{y}$.

$\vec{y} - \vec{p} = (y - \hat{y}) \perp X$.

$X^T(y - Xw^*) = 0$.

$X^T y = X^T X w^*$

$w^* = (X^T X)^{-1} X^T y$.

- Connection to Probabilistic Models
  - This same solution can be derived as the maximum conditional likelihood estimate for the parameters of a conditional Normal model
  - Assumption: The output (or the observation) $\mathbf{y}$ is from a deterministic function with additive Gaussian noise:

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \epsilon, \text{ where } p(\epsilon; \sigma^2) = \mathcal{N}(0, \sigma^2 \mathbf{I})$$

  - This implies that

$$p(\mathbf{y}|\mathbf{X}; \mathbf{w}, \sigma^2) = \frac{1}{\sqrt{(2\pi\sigma^2)^m}} \exp\left(-\frac{1}{2\sigma^2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})\right)$$

- Maximizing the log conditional likelihood, we have

$$\begin{aligned}
&\max_{\mathbf{w},\sigma^2} \log p(\mathbf{y}|\mathbf{X};\mathbf{w},\sigma^2) \\
=&\max_{\mathbf{w},\sigma^2} -\frac{m}{2}\log 2\pi - \frac{m}{2}\log\sigma^2 - \frac{1}{2\sigma^2}(\mathbf{y}-\mathbf{X}\mathbf{w})^T(\mathbf{y}-\mathbf{X}\mathbf{w}) \\
=&\min_{\mathbf{w},\sigma^2} \frac{m}{2}\log\sigma^2 + \frac{1}{2\sigma^2}(\mathbf{y}-\mathbf{X}\mathbf{w})^T(\mathbf{y}-\mathbf{X}\mathbf{w})
\end{aligned}$$

- We note that solving for $\mathbf{w}$ is independent of σ^2:

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} \frac{1}{2}(\mathbf{y}-\mathbf{X}\mathbf{w})^T(\mathbf{y}-\mathbf{X}\mathbf{w})$$

  - Proof:

3.5 A Probabilistic View of Linear Regression

The output y is from a deterministic function with additive Gaussian noise:

$$y = Xw + \epsilon, \text{ where } p(\epsilon;\sigma^2) = \mathcal{N}(\epsilon;0,\sigma^2 I). \tag{49}$$

Equivalently,

$$p(y|X;w,\sigma^2) = p(y-Xw;\sigma^2) = \mathcal{N}(y-Xw;0,\sigma^2 I) = \mathcal{N}(y;Xw,\sigma^2 I). \tag{50}$$

Recall the multivariate Gaussian distribution

$$\mathcal{N}(z;\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^m|\Sigma|}}\exp\left(-\frac{1}{2}(z-\mu)^T\Sigma^{-1}(z-\mu)\right), \tag{51}$$

and we have

$$\begin{aligned}
p(y|X;w,\sigma^2) &= \frac{1}{\sqrt{(2\pi)^m|\sigma^2 I|}}\exp\left(-\frac{1}{2}(y-Xw)^T(\sigma^2 I)^{-1}(y-Xw)\right) \\
&= \frac{1}{\sqrt{(2\pi\sigma^2)^m}}\exp\left(-\frac{1}{2\sigma^2}(y-Xw)^T(y-Xw)\right).
\end{aligned} \tag{52}$$

- Normal Equations

$$\mathbf{w}^\star = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

  - Pseudo-inverse

$$\mathbf{X}^{\dagger} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$$

    - For invertable X pseudo-inverse is just inverse
    - When $X^T X$ is non-invertable:
        - $m < n + 1$ => increase sample set and regularization
        - $m > n + 1$ => singular (remove redundant features)
            - Reason: $rank(X^T X) = rank(X)$
    - n to large => GD
    - m to large => SGD / mini-batch GD
- Strengths and Limitations of OLS
    - Need at least $m \geq n + 1$ data cases to learn a model with an n dimensional feature vector. Otherwise **inverse of** $X^T X$ is not defined
    - Very sensitive to noise and outliers due to MSE objective function/Normally distributed residuals assumption (prefer many medium error than one large error)
    - Sensitive to co-linear features ($x_i \approx a x_j + b$) . Otherwise inverse of $X^T X$ is not numerically stable
    - Computation is cubic in data dimension n

Lecture06: Regularized Linear Regression: Ridge, and Lasso & Overfitting, Regularization, and Cross Validation

Regularized Linear Regression

- Reason of regularization
    - Control the parameter space to avoid *overfitting*
    - Obtain numberically more stable solutions
    - Enforce the desired parameter space as a prior
    - What is the price that we pay? The regularization term will bias the least squares estimate (not exactly least square)
    - Probabilistix interpretation:

$$P(w \mid y, X; \sigma^2, \lambda) \propto P(y \mid w, X; \sigma^2)\, P(w; \lambda)$$

$$= \frac{P(w, y, X; \sigma^2, \lambda)}{P(y, X; \sigma^2)}$$

$$= \frac{P(y \mid w, X; \sigma^2, \lambda) \cdot P(w, \lambda)}{P(y, X; \sigma^2)}$$

$$\log P(w \mid y, X; \sigma^2, \lambda) \propto \log P(y \mid w, X, \sigma^2) + \log P(w; \lambda)$$

- General framework:

$$\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \frac{\lambda}{2}\sum_{j=1}^{n} |w_j|^q = \min_{\mathbf{w}} \frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \frac{\lambda}{2}\|\mathbf{w}\|_q^q$$

  - For ridge regression, $q = 2$. Closed-form solution exists
  - For lasso regression, $q = 1$. It is a quadratic programming problem with no closed-form solution in general. However, it can be solved efficiently using an algorithm called *least angle regression*. The advantage is that lasso simultaneously performs regularization and **encourages sparsity** (as a way of feature selection)
  - When $q < 1$, the problem is non-convex, but **encourages more sparsity**
- The relationship between the **weight matrix** of regularized version and linear regression version
  - assume that: $X^T X = I$ (each feature vector are unit orthogonal basis)
  - Linear regression: $min_w \frac{1}{2}||Xw - y||_2^2$

$$w_{LS} = (X^T X)^{-1} X^T y = X^T y$$

  - Ridge regression: $min_w \frac{1}{2}||Xw - y||_2^2 + \frac{\lambda}{2}||w||_2^2$

$$w_{RR} = (X^T X + \lambda I)^{-1} X^T y = X^T y/(1+\lambda) = W_{LS}/(1+\lambda)$$

  - Lasso regression: $min_w \frac{1}{2}||Xw - y||_2^2 + \lambda||w||_1$

$$w_{RR} = (X^T X + \lambda I)^{-1} X^T y = X^T y/(1+\lambda) = W_{LS}/(1+\lambda)$$

  - Proof:

Ridge

$$\min\left(\frac{1}{2}\|Xw-y\|_2^2+\frac{\lambda}{2}\|w\|_2^2\right)$$

$$=\min\left(\frac{1}{2}(Xw-y)^T(Xw-y)+\frac{\lambda}{2}w^Tw\right)\quad \text{b.c.}\ (X^TX=XX^T=I)$$

$$=\min\left(\frac{1}{2}(Xw-y)^TXX^T(Xw-y)+\frac{\lambda}{2}w^Tw\right)$$

$$=\min\left(\frac{1}{2}\left[X^T(Xw-y)\right]^T\left[X^T(Xw-y)\right]+\frac{\lambda}{2}w^Tw\right)$$

$$=\min\left[\frac{1}{2}\left(X^TXw-X^Ty\right)^T\left(X^TXw-X^Ty\right)+\frac{\lambda}{2}w^Tw\right]$$

$$=\min\left[\frac{1}{2}(w-X^Ty)^T(w-X^Ty)+\frac{\lambda}{2}w^Tw\right]$$

$$=\min\left(\|w-X^Ty\|_2^2+\lambda\|w\|_2^2\right)\quad \text{b.c. } X^Ty=w_{LR}$$

$$=\min\left(\|w-w_{LR}\|_2^2+\lambda\|w\|_2^2\right)$$

$$=\min\left(\|w\|_2^2(\lambda+1)-2w^Tw_{LR}+\|w_{LR}\|_2^2\right)$$

$$w=\left(\frac{1}{1+\lambda}\right)w_{LR}$$

Lasso

$$\min\left(\frac{1}{2}\|Xw-y\|_2^2+\lambda\|w\|_1\right)$$

$$=\min\left(\frac{1}{2}\|w\|_2^2-w^Tw_{LR}+\frac{1}{2}\|w_{LR}\|^2+\lambda\|w\|_1\right)$$

$$=\min\left(\frac{1}{2}\sum_{j=1}^{n}w_j^2-\sum_{j=1}^{n}w_j(w_{LS})_j+\lambda\sum_{j=1}^{n}|w_j|+\frac{1}{2}\|w_{LS}\|_2^2\right)$$

for each j: $\ell(w_j)=\frac{1}{2}w_j^2-w_j(w_{LS})_j+\lambda|w_j|+\frac{1}{2}(w_{LS})_j^2$ (常数项)

when $w_j(w_{LS})_j<0$

$-w_j(w_{LS})_j>0>-(-w_j)(w_{LS})_j$

Therefore, $-w_j$ is a better solution than w_j.

We only need to find the optimal under the constraint $w_j(w_{LS})_j\geq 0$

① when $(w_{LS})_j>0$ $\quad w_j\geq 0$

$$\ell(w_j)=\frac{1}{2}w_j^2+(\lambda-(w_{LS})_j)w_j$$

$$w_j^*=-\frac{\lambda-(w_{LS})_j}{2\times\frac{1}{2}}=-(\lambda-(w_{LS})_j)$$

$$=(w_{LS})_j-\lambda\quad (\text{if } (w_{LS})_j-\lambda\leq 0)$$

$$w_j^*=\max(0,(w_{LS})_j-\lambda)$$

if $(w_{LS})_j-\lambda<0$, we know that $w_j^*\geq 0$.

if $w_j^*>0$ $\quad(\lambda-(w_{LS})_j)>0$

$\ell(w_j)>0$

if $w_j^*=0$, $\ell(w_j)=0$

$w_j^*=0$ is the best

② when $(w_{LS})_j<0$ $\quad w_j\leq 0$

$$\ell(w_j)=\frac{1}{2}w_j^2-(\lambda+(w_{LS})_j)w_j$$

$$w_j^*=\frac{\lambda+(w_{LS})_j}{2\times\frac{1}{2}}=\lambda+(w_{LS})_j$$

$$(\text{when } (\lambda+(w_{LS})_j)\leq 0)$$

$$w_j^*=\min(\lambda+(w_{LS})_j,0)$$

if $\lambda+(w_{LS})_j>0$.

if $w_j^*<0$ $\Rightarrow$ $-(\lambda+(w_{LS})_j)w_j^*>0$

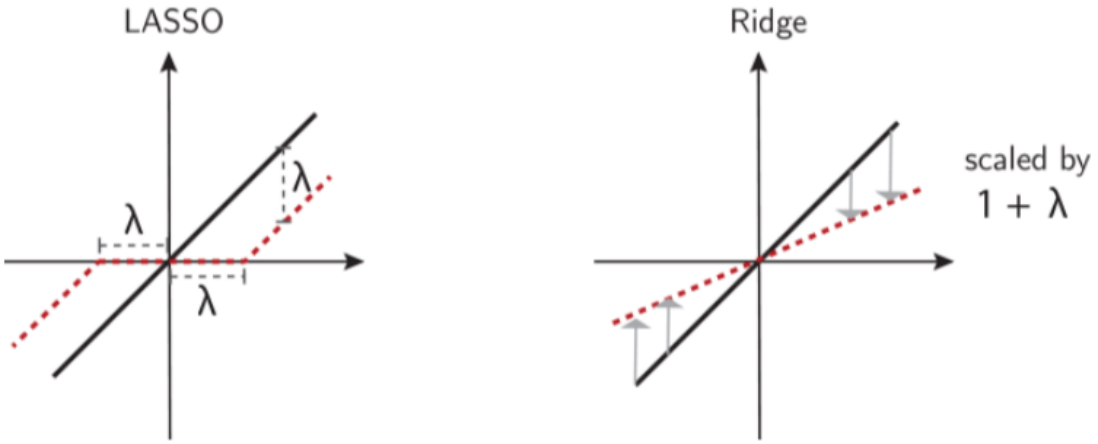
$\ell(w_j)>0$

$w_j^*=0$ $\Rightarrow$ $\ell(w_j)=0$

From ① & ②.

$$w_j^*=\text{sign}(w_{LS})\max(0,|w_{LS}|-\lambda)$$

- Geometrical meaning (with x-axis W_{LR})



- Lasso vs Ridge (2D case)

- Property of the minimum point

$$L_r = \sum_{i=1}^{n} (y - x_1^{(i)} w_1 - x_2^{(i)} w_2 - b) + \lambda(|w_1^*|^r + |w_2^*|^r) = C$$

(C_1 = the first sum, C_2 = the regularization term)

if the optimal of L_r is found to be C

and at that moment

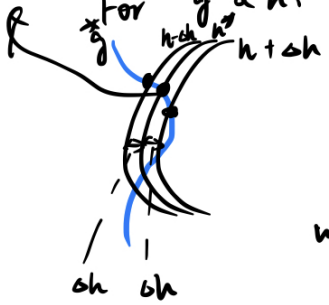
$$\begin{cases} \sum_{i=1}^{n} (y - x_1^{(i)} w_1^* - x_2^{(i)} w_2^* - b) = C_1 \\ \lambda(|w_1^*|^r + |w_2^*|^r) = C_2 \end{cases}$$

on w_1-w_2 plane, they will be two curves, which meet at the point (w_1^*, w_2^*).

(w_1^*, w_2^*) is the only common point

e.g. $f(w_1, w_2) = g(w_1, w_2) + h(w_1, w_2)$.

(w_1^*, w_2^*) for g & h. if they one not intersect at only 1 point



for minimum value.

when $dh \to 0$, $h^* + dh$ and $h^* - dh$ will have similar shape with h^*.

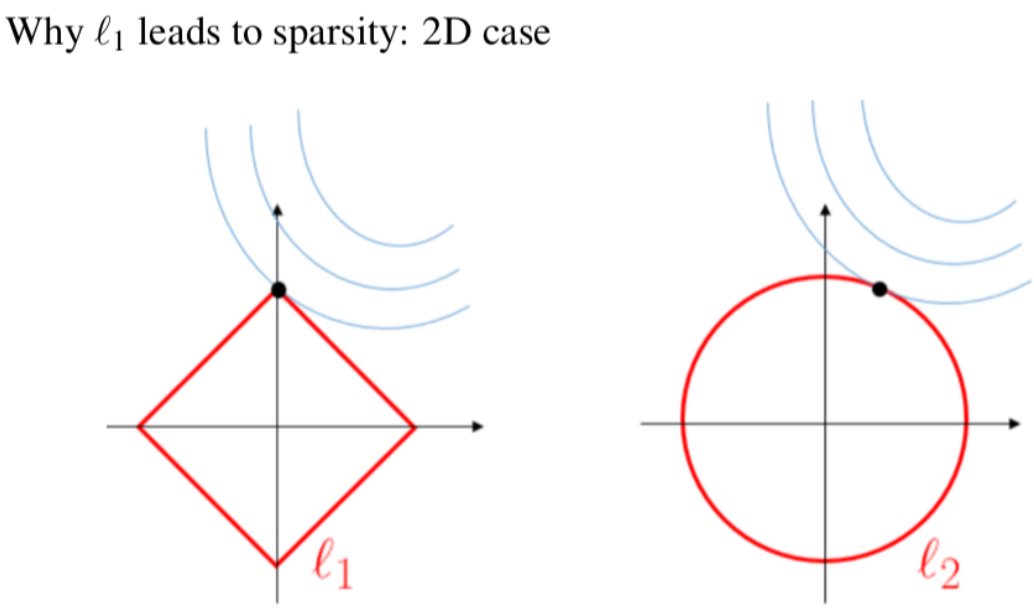
when $dh > 0$. $h^* - dh < h^*$

$dh < 0$ $h^* + dh < h^*$

$f = g^* + h^* - dh < f^*$ $f = g^* + h^* + dh < f^*$.

Therefore, there is only one intersection.

- Graph of Lasso regression and Ridge regression:



    - it's easier to have the intersection on the corner for l_1 case
    - one of w_j will be zero, which means the feature is dropped => sparsity
- Strengths and Limitations of Ridge and Lasso
    - Need at least $m \geq n+1$ data cases to learn a model with an $n+1$ dimensional feature vector (similar as linear regression)
    - Work when $X^T X$ is close to singular => select feature
    - MSE objective function is still sensitive to noice points
    - Do not solve **bias problem**
    - Computation for ridge is still cubic in data dimension n with additional cost to determine regularization parameters. Computation for lasso is similar

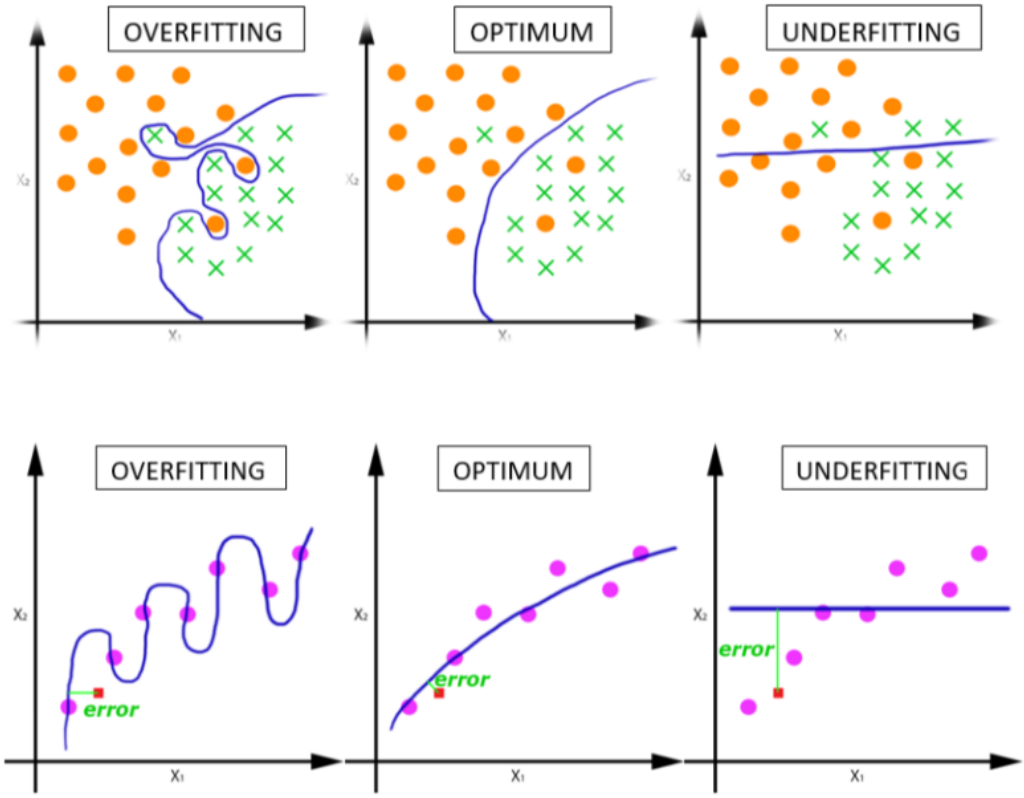
Review

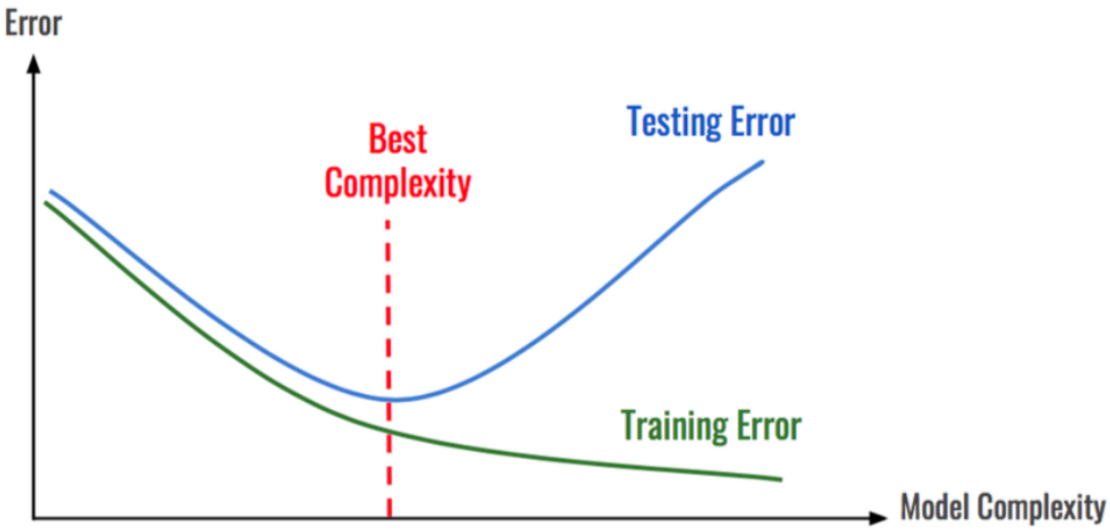
- We have learnt 4 classifiers in depth: KNN, Naive Bayes, LDA and Logistic Regression
    - KNN: Results sensitive to k
    - Logistic Regression: $p(y|x;w)$ sensitive to the magnitude of w

Capacity and Generalization

- Model capacity:
    - **Deterministic classifiers** that can represent **more complex decision boundaries** (in log form) are said to have **higher *capacity*** than classifiers that can only represent simpler boundaries
    - **Probabilistic classifiers** that can represent **more complex sets of conditional** $p(y|x)$ are said to have higher $capacity$ than probabilistic classifiers that can only represent simpler sets of conditionals
    - Rank of the leart classifiers: KNN > Naive Bayes, Logistic Regression $\approx$ LDA
- Reason of not always using the classifier with the highest possible capacity for every problem
    - This would minimize the error on the **training data**. However, what we really care about for prediction problems is *generalization*

- **Generalization**: The ability of a trained classifier to achieve an error rate on *future*, *unseen examples* (the generalization error rate) that is comparable to the training error rate
- **Capacity Control**: To achieve optimal generalization performance for a given traing set, we often need to control model capacity carefully
- **Overfitting vs underfitting**
    - **Overfitting**: The generalization error for a classifier is much worse than the training error. This usually results from choosing a classifier with too much capacity so that it models the noise in the training data
    - **Underfitting**: Occurs when the capacity of the classifier is too low to capture the actual structure in the training data, leading to both high training error and high generalization error
    - Examples(error on the down-left corner graph):

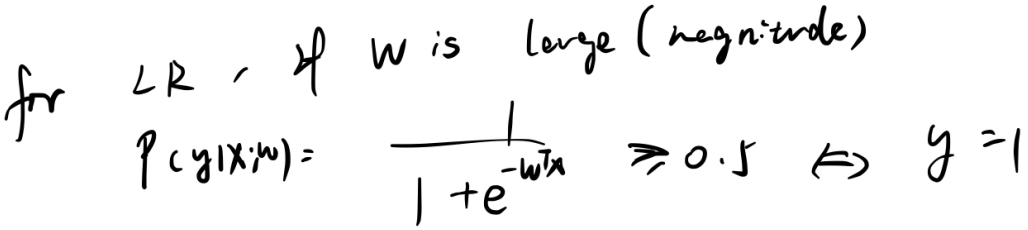


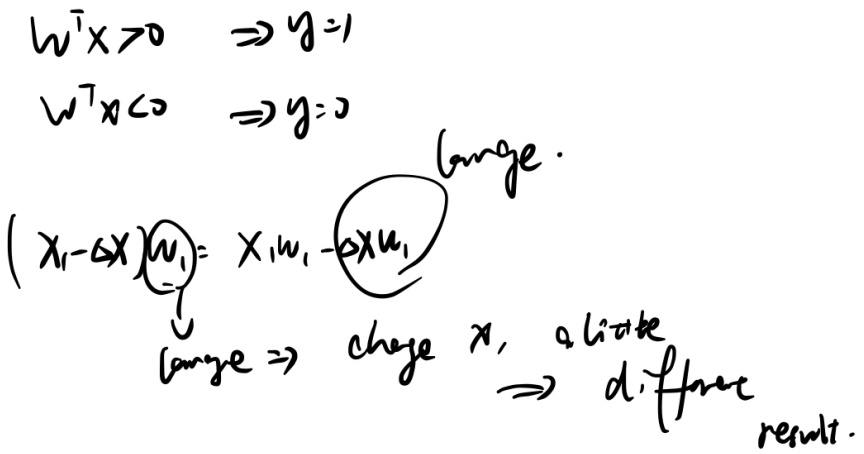


- Bias-Variance Trade-Off (both of them are negative things)
  - Bias: A classifier is said to have low *bias* if the true decision boundary or conditionals $p(y|x)$ can be **approximated closely by the model**
  - Variance: A classifier is said to have low *variance* if the decision boundary or conditionals $p(y|x)$ it constructs are stable with respect to small changes in the training data
  - Bias-Variance Dilemma: To achieve **low generalization error**, we need classifiers that are low-bias and low-variance, but this isn't always possible
  - Bias-Variance and Capacity: On **complex data**, models with **low capacity have low variance**, but **high bias**; while models with **high capacity** have **low bias**, but **high variance**

Hyperparameters

- Why need to control hyperparameters
  - In order to control the capacity of a classifier, it needs to have capacity control parameters
  - Because capacity control parameters can not be chosen based on training error, they are often called hyperparameters
  - For KNN: k, for logistic regression: $\alpha \; and \; \lambda$
- Capacity, Smoothness and Regularization





- **Capacity and Smoothness**: In the case of probabilistic classifiers like naive Bayes, LDA, and logistic regression, we can think of capacity in terms of the *smoothness* of $p(y|x)$
- **Regularization**: We can control the smoothness of $p(y|x)$ using a technique called *regularization* that penalizes parameters that result in overly complex $p(y|x)$ during learning
- **Laplace Smoothing**: In the case of Naive Bayes, we introduced Laplace smoothing for a categorical distribution

$$\theta_{x,j|y} = \frac{\sum_{i=1}^{m} \mathbb{I}[y^{(i)} = y \cap x_j^{(i)} = x] + \alpha}{\sum_{i=1}^{m} \mathbb{I}[y^{(i)} = y] + \alpha|\mathcal{X}|}$$

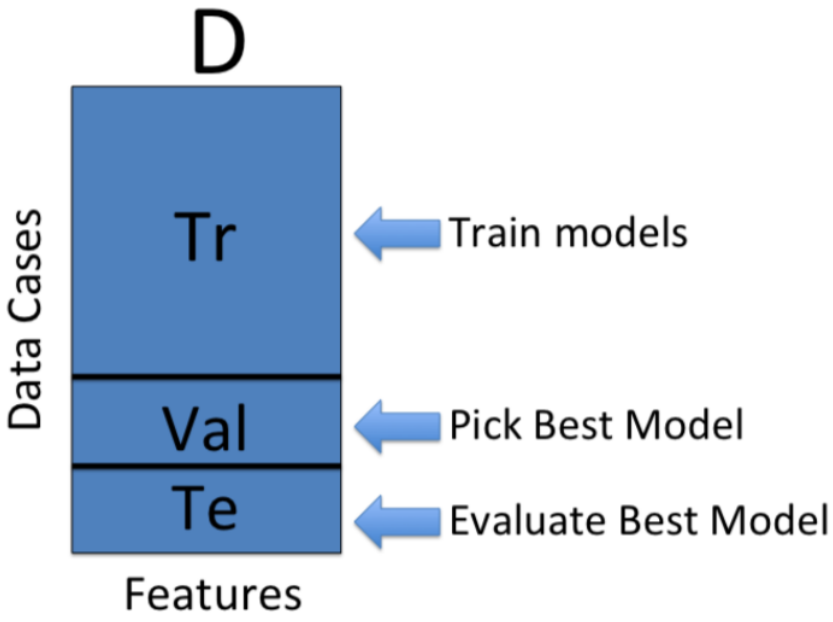
  - As the regularization hyperparameter α increases, our estimate of the distribution smooths out toward being uniform
- In the case of logistic regression, the smoothness of $p(y|x;\theta)$ is determined by the magnitude of θ
  - Similar in linear regression, we ca control the magnitude of θ by introducing a regularization term into the conditional log likelihood learning criteria that penalizes the l_p-norm of θ

$$\boldsymbol{\theta}^{\star} = \arg\min_{\boldsymbol{\theta}} -\sum_{i=1}^{m} \log p(y^{(i)}|\mathbf{x}^{(i)};\boldsymbol{\theta}) + \lambda||\boldsymbol{\theta}||_p^p$$

  - As the regularization hyperparameter λ increases, the learned $p(y|x;\theta)$ will smooth out

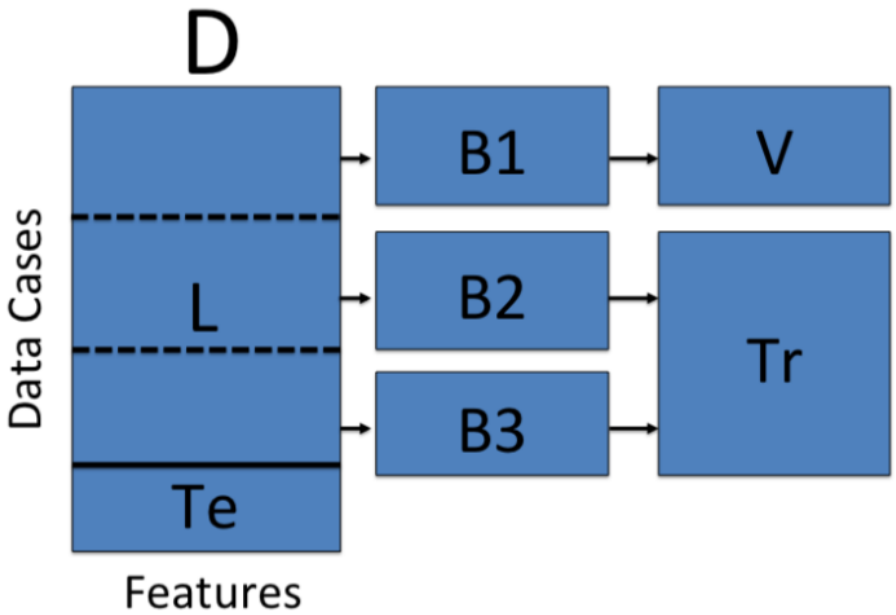
Model Selection and Evaluation

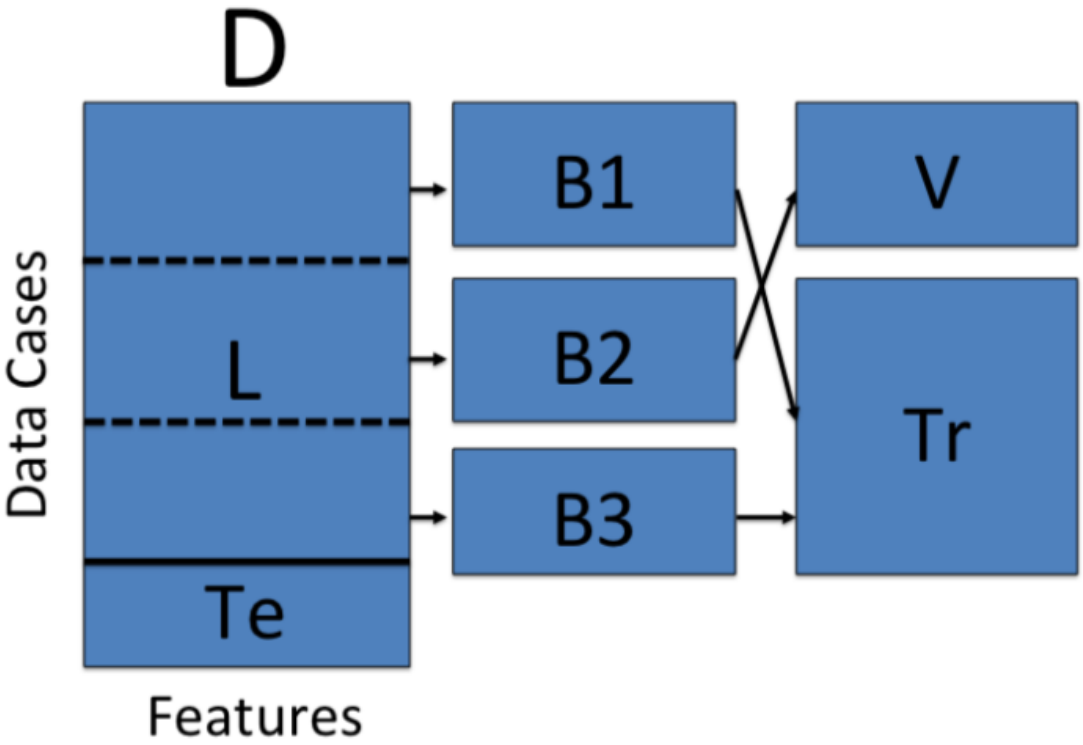
- Principles:
  - We've identified the parameters that **control the capacity** of our models, we need a way to choose optimal values for these parameters
  - In addition, we will want an estimate of the **generalization error** that the selected parameters achieve
  - To obtain valid results, we need to use appropriate methodology and construct learning experiments carefully
  - **Guiding Principle**: Data used to estimate generalization error(**testing set**) can not be used for any other purpose, $i.e.$, model training, hyperparameter selection, feature selection, etc. Otherwise, the results of the evaluation will ve $biased$
- Methods:
  - Recipe 1: Train, Validation, and Test
    - Given a data set D, we randomly partition the data cases into a **training set (Tr)**, a **validation set (Val)**, and a **test set (Te)**. Typical splits are 60/20/20, 90/5/5, etc.
    - Training
      - Models $\{f(i)\}$ are learned on Tr for each **one** choice of hyperparameters $\theta(i)$ (Choose weigth matrix in training set)
    - Validation
      - The validation error $Err_v^{(i)}$ of each model $f^{(i)}$ is evaluated on Val v
      - The hyperparameters $\theta^\star$ with the lowest validation error are selected and the classifier is re-trained using these hyperparameters on **Tr + Val**, yielding a final model $f^\star$ (Choose hyperparemeters in training & validation set)
    - Testing
      - Generalization performance is estimated by evaluating error/accuracy of $f^\star$ on the test data Te (Evaluate the final model)
    - Example: (Notice that the data cases needs to be randomly shuffled before partitioning D)

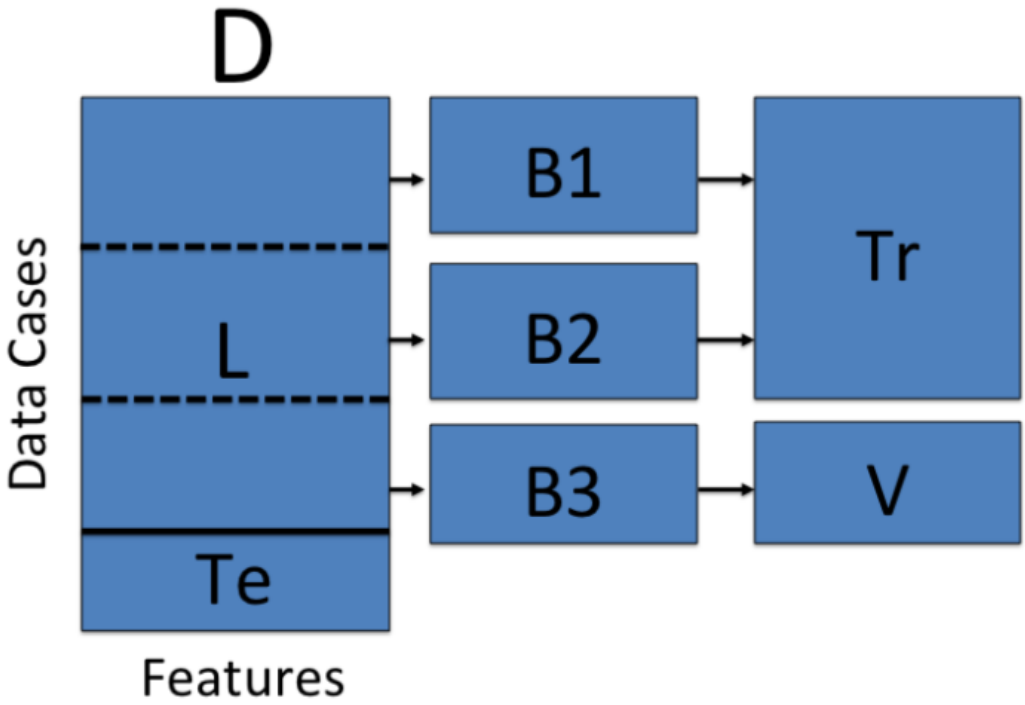


  - Recipe 2: Cross-Validation and Test

- Randomly partition D into a learning set L and a test set Te, typically 50/50, 80/20, etc.
- We next **randomly partition** L into a set of K blocks $B_1, \ldots, B_K$
- For each cross-validation fold $k = 1, \ldots, K$:
  - Let $Val = B_k$ and $Tr = L/B_k$ (the remaining $K-1$ blocks)
  - Learn $f^{(i,k)}$ on Tr for each choice of hyperparameters $\theta^{(i,k)}$
  - Compute $Err^{(i,k)}$ of $f^{(i,k)}$ on Val v
- Select hyperparameters $\theta^\star$ minimizing $\frac{1}{K}\sum_{k=1}^{K} Err_v^{(i,k)}$ and re-train model on L using these hyperparameters, yielding the final model $f^\star$
- Estimate generalization performance by evaluating error/accuracy of $f^\star$ on Te
- Example: Three Fold Cross-Validation and Test (Notice that the data cases needs to be randomly shuffled before partitioning D into L and Te)

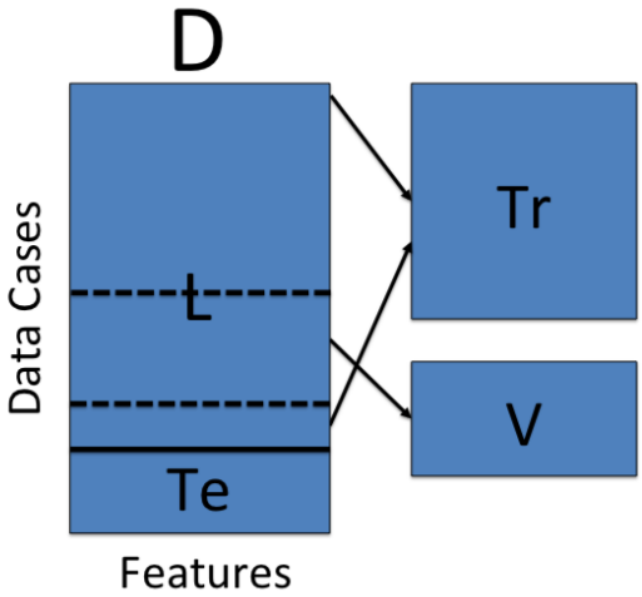






- Recipe 3: Random Resampling Validation and Test (Notice that the data cases needs to be randomly shuffled before partitioning D into L and Te)
  - Randomly partition the data cases into a learning set L and a test set Te, typically 50/50, 80/20, etc.
  - For sample $k = 1, \ldots, K$:
    - Randomly partition L into Tr and Val, again 50/50, 80/20, etc.

        - Learn $f^{(i,k)}$ on Tr for each choice of hyperparameters $\theta^{(i,k)}$
        - Compute $Err^{(i,k)}$ of $f^{(i,k)}$ on Val v
    - Select hyperparameters $\theta^\star$ minimizing $\frac{1}{K}\sum_{k=1}^{K} Err_v^{(i,k)}$ and re-train model on L using these hyperparameters, yielding the final model $f^\star$
    - Estimate generalization performance by evaluating error/accuracy of $f^\star$ on Te
    - Example:



- Trade-Offs:
    - In cases where the data has a benchmark split into a training set and a test set, we can use Recipes 1-3 by preserving the given test set and splitting the given training set into train and validation sets as needed
    - Choosing larger K in cross-validation will reduce bias. Choosing larger K in random re-sampling validation will reduce variance and bias. However, both increase computational costs. K = 3, 5, 10 are common choices for cross-validation. , $K = m$, also known as ***leave-one-out cross validation*** is also popular when feasible

Feature Selection

- Feature Selection: As a special and important case of model selection, feature selection selects a useful subset from all the features
- Reason of feature selection:
    - Some algorithms (e.g. normal equation) scale (computationally) poorly with increased feature dimension
    - Irrelevant features can confuse some algorithms
    - Redundant features may adversely affect regularization
    - Reduces data set and resulting model size
- Methods
    - Wrapper methods (keep the learning algorithm in the loop)
        - Requires repeated runs of the learning algorithm with different sets of features

        - Can be computationally expensive
    - Filter feature selection methods
        - Use some ranking criteria to rank features
        - Select the top ranking features
- Wrapper Methods
    - Forward search
        - Start with no features
        - Greedily include the most relevant feature
        - Stop when selected the desired number of features
        - Steps:
            - Let $F = \{\}$
            - While not selected desired number of features
            - For each unused feature f
                - Estimate model's error on feature set $F \cup f$
            - Add f with lowerst generalization error to F
    - Backward search (Maybe computational expensive)
        - Start with all the features
        - Greedily remove the least relevant feature
        - Stop when selected the desired number of features
        - Steps:
            - Let $F = \{all\ features\}$
            - While not reduced to desired number of features
            - For each unused feature $f \in F$
                - Estimate model's error on feature set $F \setminus f$ (using cross-validation)
            - Remove f with lowerst generalization error from F
    - Inclusion/Removal criteria uses cross-validation (*i.e.*, train model using the current subset of features and estimate the generalization error)
- Filter Feature Selection Methods (outdated => deep learning nowadays)
    - Uses heuristics but are much faster than wrapper methods
    - Correlation criterion: Rank features according to their correlation with the labels:

$$\mathrm{Corr}(\mathbf{x}_j, \mathbf{y}) = \frac{\sum_{i=1}^{m}(x_j^{(i)} - \mu_{\mathbf{x}_j})(y^{(i)} - \mu_{\mathbf{y}})}{\sqrt{\sum_{i=1}^{m}(x_j^{(i)} - \mu_{\mathbf{x}_j})^2}\sqrt{\sum_{i=1}^{m}(y^{(i)} - \mu_{\mathbf{y}})^2}}$$

    - Mutual information criterion:

$$\mathrm{MI}(x_j; y) = \sum_{x_j \in \mathcal{X}_j} \sum_{y \in \mathcal{Y}} p(x_j, y) \log \frac{p(x_j, y)}{p(x_j)p(y)},$$

    - where the probabilities$p(x_j, y), p(x_j)$ and $p(y)$ can be estimated according to their empirical distributions on the training set
- More on Mutual Information
    - Mutual information is a fundamental concept in *information theory*, measuring the mutual dependence between two variables
    - It quantifies the "amount of information" obtained about one random variable through observing the other random variable
    - As a function of $p(x, y) = p(x)p(y|x)$, $I(x; y)$ is concave in $p(x)$ for fixed $p(y|x)$, and is convex in $p(y|x)$ for fixed $p(x)$
    - Maximizing $I(x; y)$ w.r.t. $p(x)$ leads to *noisy-channel coding theorem*, establishing the maximum communication rate through a noisy channel
    - Minimizing $I(x; y)$ w.r.t. $p(y|x)$ leads to *rate-distortion theorem*, laying the theoretical foundations for lossy data compression

---

Lecture07 Support Vector Machine

Target: Increase the capacity of linear classifiers so they can produce non-linear classification boundaries

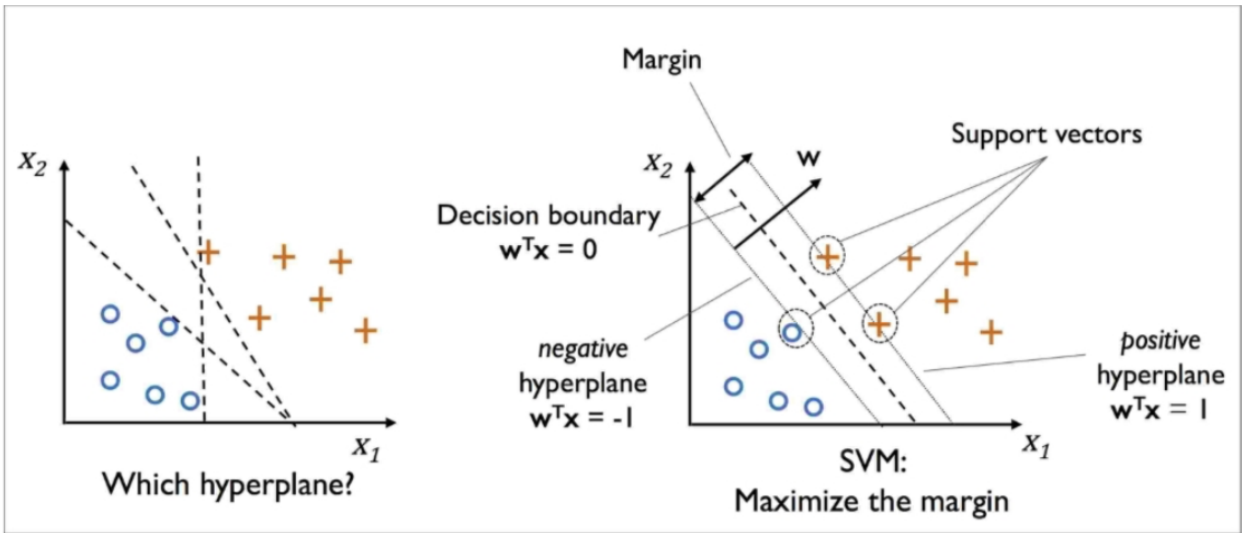
---

Support Vector Machine (SVM)

- Decision Boundary
    - A binary SVM is a discriminative classifier that takes labels in the set $\{-1, 1\}$
    - The decision function has the form:

$$f_{SVM}(x) = sign(\mathbf{w}^T \mathbf{x} + b)$$
$$\mathbf{w}^T \mathbf{x} + b = 0 \ (on\ the\ decision\ boundary)$$

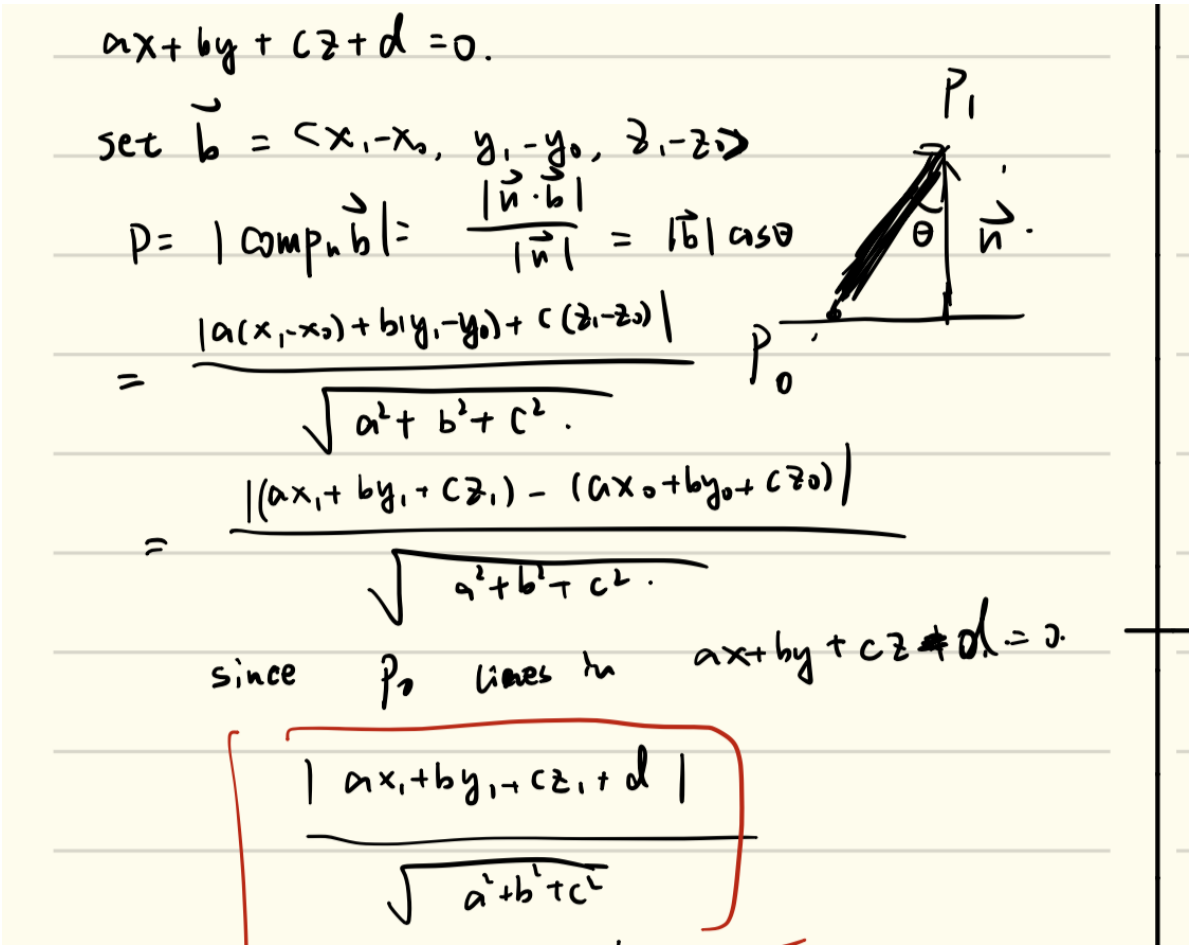
    - It's easy to show that the decision boundary for logistic regression can be written in exactly the same way
    - Question: If logistic refgression and SVMs have the same form for their decision boundaries, how do they differ
        - Logistic regression minimize the log probability of miss-classification
        - Logistic regression maximize the probability on the correct label (i.e. if the probability is more than 0.5, it can be correctly classified. But it tends to increase the probability during training)
- Maximum Margin Principle (**notice the axis**)



    - Define the *margin* of a linear classifier as the width that the **boundary could be increased before hitting a data point**
    - The simplest kind of SVM is a linear classifier with the maximum margin
    - Reason of MAX margin:
        - If we've made a small error in the location of the decision boundary, this gives us least chance of causing a misclassification
        - There's solid theory (using VC dimension) that provides indirect support for this proposition
- Compute the Margin
    - The distance between a point to a line:

$$d^{(i)} = \frac{|\mathbf{w}^T\mathbf{x} + b|}{||\mathbf{w}||_2} = \frac{y^{(i)}(\mathbf{w}^T\mathbf{x} + b)}{||\mathbf{w}||_2}$$

    - Proof:



- The $y^{(i)}$ can be added because the value can be only found in the set $\{-1, 1\}$
- The margin of $\mathbf{w}^T x + b = 0$ with respect to a training set $D = \{(\mathbf{x}^{(i)}, y^{(i)}), i = 1, \ldots, m\}$, where $||\mathbf{w}||_2 = \sqrt{\sum_{j=1}^{n} w_j^2}$

$$d(margin) = min\ d^{(i)} = min_{(\mathbf{x},y)\in D} \frac{y^{(i)}(\mathbf{w}^T\mathbf{x} + b)}{||\mathbf{w}||_2}$$

- Maximize the Margin
  - The maximum margin solution is found by solving

$$max_{\mathbf{w},b}(\frac{1}{||\mathbf{w}||_2} min_{(\mathbf{x},y)\in D}\ y^{(i)}(\mathbf{w}^T\mathbf{x} + b))$$

  - Note that if we make the rescaling $\mathbf{w} \rightarrow \gamma\mathbf{w}$ and $b \rightarrow \gamma b$, then the objective is unchanged
    - We can use this freedom to set: $min_{(\mathbf{x},y)\in D}\ y(\mathbf{w}^T\mathbf{x} + b) = 1$

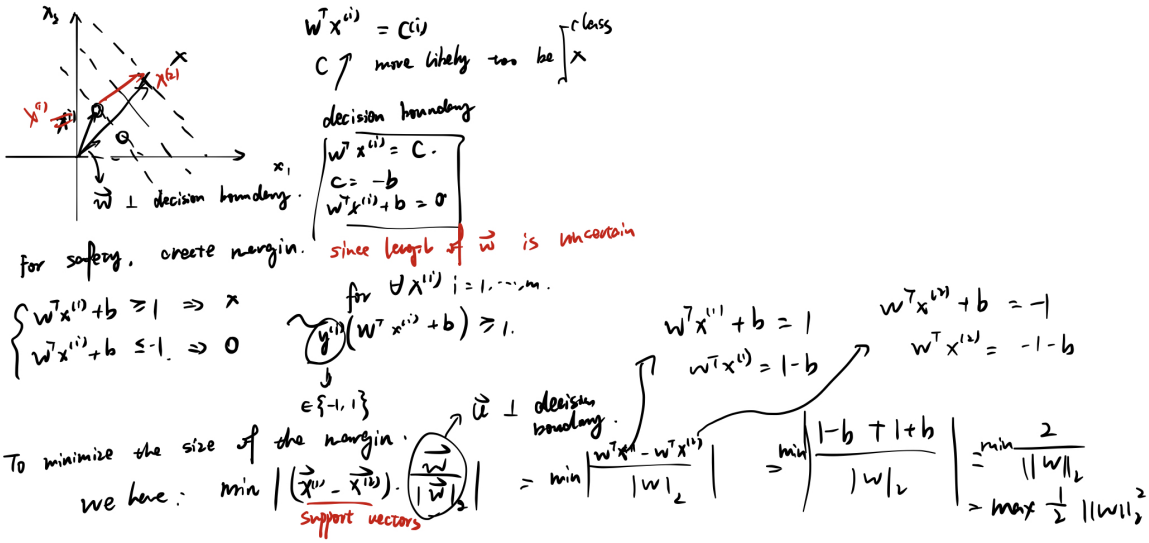
$$Ax + By + C = 0 <=> 2Ax + 2By + 2C = 0$$

    - Or equivalently: $y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1, \forall i = 1, \ldots, m$
- Primal Form, **Hard-Margin** of SVM

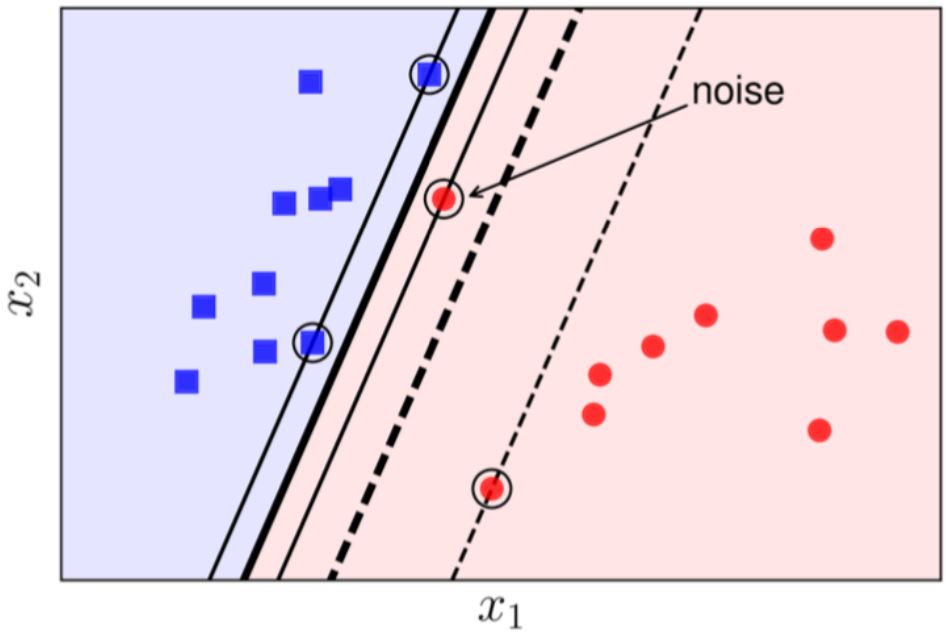
$$max_{\mathbf{w},b}(\frac{1}{||\mathbf{w}||_2}) <=> min_{\mathbf{w},b}\ \frac{1}{2}||\mathbf{w}||_2^2 (for\ convenience)$$
$$subject\ to\ y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1, \forall i = 1, \ldots, m$$

  - Quadratic programming problem with a quadratic objective function and linear inequality contstraint
  - Well studied problem in operations research

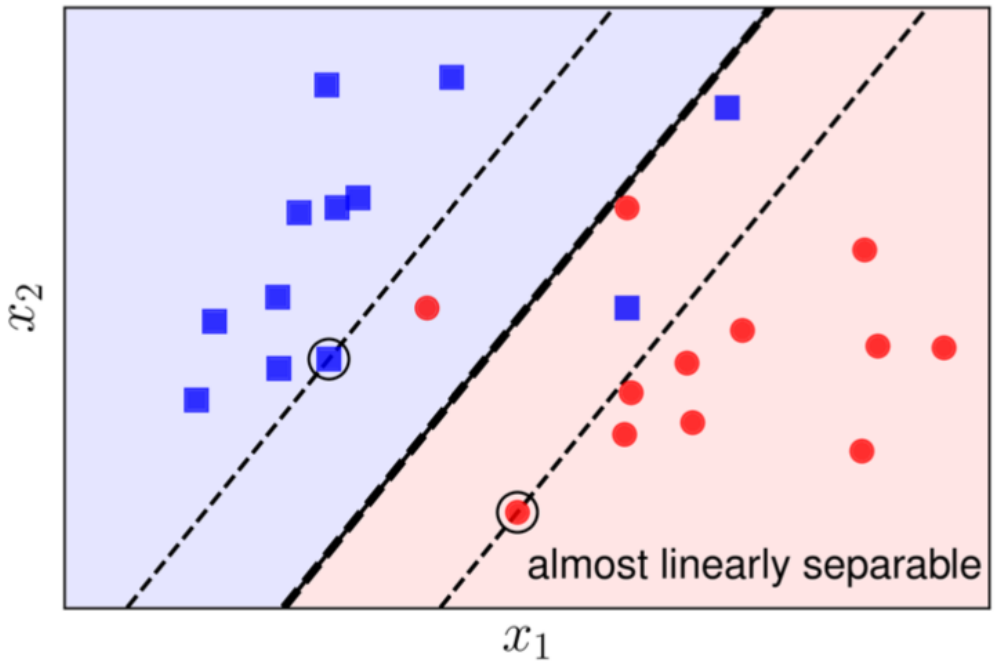
- Another ponit of view: (min and max in the last line should be reversed)



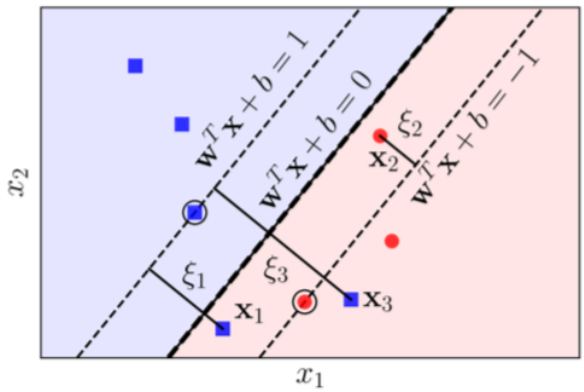
- Primal From, Soft-Margin
  - Hard margin is not enough:
    - When there is a noise point to narrow down the margin



    - when the data is not linear-discriminatable: **notice that in this time, margin is nearly zero**



- We need to add some level of tolerances to the model:



$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{m}\xi_i$$

$$\text{subject to} \quad y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 - \xi_i, \;\; i = 1, \ldots, m$$

$$\xi_i \geq 0, \;\; i = 1, \ldots, m$$

- In other word, for normal case, $y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1$, the penalty of misclassification is zero, while on the other hand, there is a penalty with the value $1 - y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b)$
- The level of tolerance can be found on the second equation:
  - Originally, $y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1$ means that, all pairs of the points should obey this rule, therefore, **the "distance" between two boundaries generated by the support vector should be at least 2**

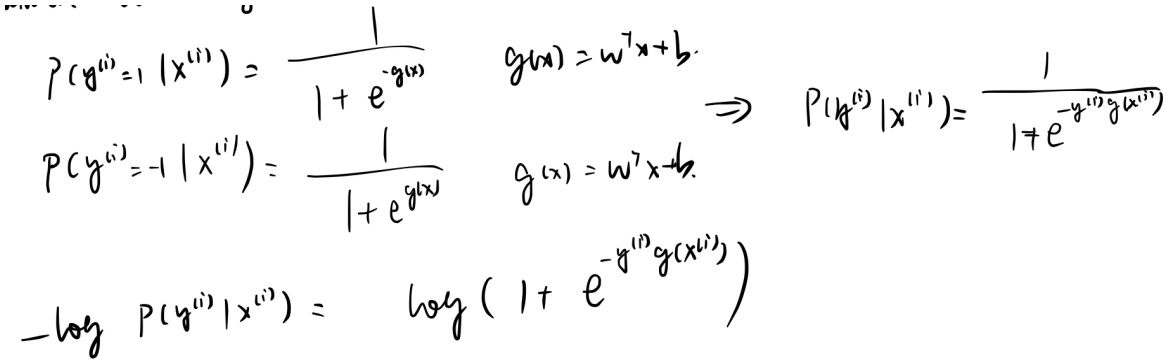
        - Now, $y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b) \geq 1-\xi_i, \xi_i > 0$ means that, for any one of the points, there is a **tailored** ξ_i to make sure that the point always lies in the safe constraint. Once $y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b) < 1$, the difference between $1\ and\ y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b)$ will be counted as penalty.
        - This approach makes sure that all points are not forced to have a closest support vector on one or two sides of the decision boundary, but the mistake caused by it will be countered as part of the loss function
- Hinge Loss
    - In the case of primal-form SVM with soft-margin, it is equivalent to minimizing the function:

$$C\sum_{i=1}^{m} max(0, 1-y^{(i)}.\, g(\mathbf{x}^{(i)})) + \frac{1}{2}||\mathbf{w}||_2^2$$

        - where $g(x) = \mathbf{w}^T\mathbf{x}+b$ and C is the hyperparameter that controls the amount of regularization
    - The function $l_h(y, g(\mathbf{x})) = max(0, 1-y.\, g(\mathbf{x}))$ is called the *hinge loss*
- Logistic Loss
    - In the case of logistic regression with l_2 regularization, we select the model parameters by minimizing the function

$$C\sum_{i=1}^{m} -log\, p(y^{(i)}|\mathbf{x}^{(i)}) + \frac{1}{2}||\mathbf{w}||_2^2$$

    - Under the assumption that the labels take the values $\{-1, 1\}$:

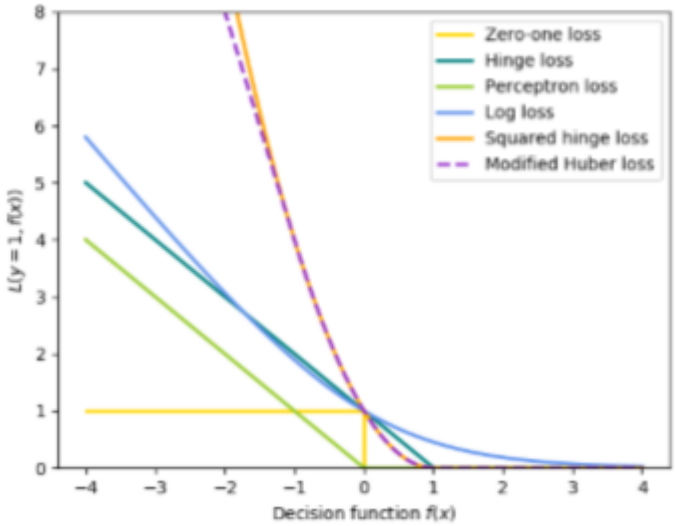


$$C\sum_{i=1}^{m} log\,(1+exp(-y^{(i)}.\, g(\mathbf{x}^{(i)}))) + \frac{1}{2}||\mathbf{w}||_2^2$$

        - Where the function $l_l(y, g(\mathbf{x})) = log(1+exp(-y^{(i)}.\, g(\mathbf{x}^{(i)})))$ is called the *logistic loss function*
- Zero-One Loss
    - Both the logistic loss and the hinge loss are convex upper bounds on the zero-one loss:

$$l_{01}(y, g(\mathbf{x})) = \mathbb{I}[y \neq sign(g(\mathbf{x}))]$$

    - Average of $01-loss$ is exactly the classification error rate
    - This is the loss function we'd like to minimize, but this generally isn't computationally feasible, thus the need for surrogate loss functions
- Comparison (x-axis is the decision function)



Lagrange Duality

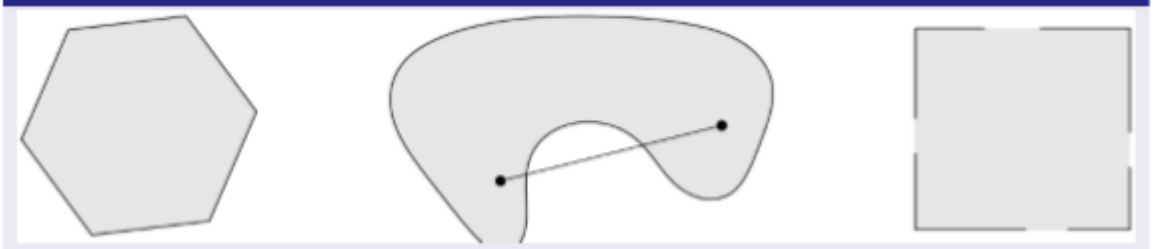
- Lagrange Duality: In mathematical optimization theory, **duality** or the **duality principle** is the principle that optimization problems may be viewed from either of two perspectives, the **primal problem** or the **dual problem**. The solution to the dual problem provides a lower bound to the solution of the primal (minimization) problem. However in general the optimal values of the primal and dual problems need not be equal. Their difference is called the duality gap. For convex optimization problems, the duality gap is zero under a constraint qualification condition.
- Convex set

Definition: Convex Set

A convex set contains line segment between any two points in the set

$$\mathbf{w}, \tilde{\mathbf{w}} \in W,\ 0 \leq \theta \leq 1 \implies \theta\mathbf{w} + (1-\theta)\tilde{\mathbf{w}} \in W$$

Example



- Convex function

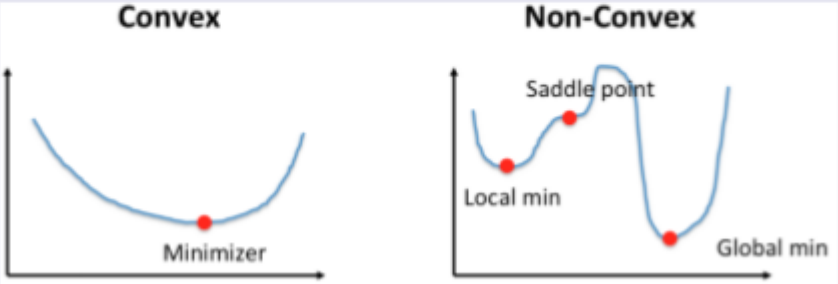
Definition: Convex Function

$f : \mathbb{R}^n \mapsto \mathbb{R}$ is convex if the domain of f, $\text{dom}(f)$, is a convex set and

$$f(\theta \mathbf{w} + (1-\theta)\tilde{\mathbf{w}}) \leq \theta f(\mathbf{w}) + (1-\theta) f(\tilde{\mathbf{w}})$$

for all $\mathbf{w}, \tilde{\mathbf{w}} \in \text{dom}(f)$, $0 \leq \theta \leq 1$.

Example



- Standard form of optimization (minimize for instance) problems

$$\begin{aligned} & min_{\mathbf{w}} f_0(\mathbf{w}) \\ subject\ to\ & f_i(\mathbf{w}) \leq 0, i = 1, \ldots, r \\ & h_i(\mathbf{w}) = 0, i = 1, \ldots, s \end{aligned}$$

  - $\mathbf{w} \in \mathbb{R}^n$ is the optimization variable
  - $f_0 : \mathbb{R}^n \mapsto \mathbb{R}$ is the objective or cost function
  - $f_i : \mathbb{R}^n \mapsto \mathbb{R}, i = 1, \ldots, r$ are the inequality constraint function
  - $h_i : \mathbb{R}^n \mapsto \mathbb{R}, i = 1, \ldots, s$ is the equality constraint function
  - If f_0 and a set of f_i are convex functions, and h_i's are affine functions, it becomes a **convex optimization problem**
    - Affine function: $f(\mathbf{x}) = A\mathbf{x} + \mathbf{b}, A \in \mathbb{R}^{s \times n}, \mathbf{x} \in \mathbb{R}^n, \mathbf{b} \in \mathbb{R}^s$
- Lagrangian: $L : \mathbb{R}^n \times \mathbb{R}^r \times \mathbb{R}^s \mapsto \mathbb{R}$, with $dom(L) : W \times \mathbb{R}^r \times \mathbb{R}^s$ (fixed weight matrix)

$$L(\mathbf{w}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f_0(\mathbf{w}) + \sum_{i=1}^{r} \lambda_i f_i(\mathbf{w}) + \sum_{i=1}^{s} \nu_i h_i(\mathbf{w})$$

  - It's a weighted sum of objective and constraint function
  - λ_i is Lagrange multiplier associated with $f_i(x) \leq 0$
  - ν_i is Lagrange multiplier associated with $h_i(x) = 0$
- Lagrange dual function: $g : \mathbb{R}^r \times \mathbb{R}^s \mapsto \mathbb{R}$, (inf means min)

$$g(\boldsymbol{\lambda}, \boldsymbol{\nu}) = inf_{\mathbf{w} \in \mathbf{W}}(L(\mathbf{w}, \boldsymbol{\lambda}, \boldsymbol{\nu})) = inf_{\mathbf{w} \in \mathbf{W}}(f_0(\mathbf{w}) + \sum_{i=1}^{r} \lambda_i f_i(\mathbf{w}) + \sum_{i=1}^{s} \nu_i h_i(\mathbf{w}))$$

  - g is **concave** and can be $-\infty$ for some $\boldsymbol{\lambda}$ and $\boldsymbol{\nu}$
  - According to the constraints: $f_i(x) \leq 0,\ h_i(x) = 0, \lambda_i > 0$, we have:

$$f_0(\tilde{\mathbf{w}}) \geq L(\tilde{\mathbf{w}}, \boldsymbol{\lambda}, \boldsymbol{\nu}) \geq inf_{\mathbf{w}}(L(\tilde{\mathbf{w}}, \boldsymbol{\lambda}, \boldsymbol{\nu})) = g(\boldsymbol{\lambda}, \boldsymbol{\nu})$$

- Definition of p^* (the value we want):

$$p^* = inf\{f_0(\mathbf{w})|f_i(\mathbf{w}) \leq 0, i = 1, \dots r, h_i(\mathbf{w}) = 0, i = 1, \dots, s\}$$

- Then we have the following equation

$$d^* = max_{\lambda \geq 0, \nu}\ g(\lambda, \nu) \leq min\ f_0(\mathbf{w}\tilde{}) = p^*$$

- Lagrange dual problem (transformed by the Lagrange dual function)

$$max_{\lambda, \nu}\ g(\lambda, \nu)$$
$$subject\ to\ \lambda \geq 0$$

  - It is a convex optimization problem, with optimal value d^*
  - d^* is the best lower bound on p^*
  - **Weak duality**: $d^* \leq p^*$, which always holds for convex and nonconvex problems
  - **Strong duality**: $d^* = p^*$ does not hold in general, but (**usually**) holds for convex problems, e.g., SVM

Optimization of SVM: From Primal to Dual

- Primal Form:
  - Notice that for constraints f_i, there is a set of constraints

$$min_{\mathbf{w}, b}\ \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{i=1}^{m}\xi_i$$
$$subject\ to\ y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 - \xi_i, i = 1, \dots, m$$
$$\xi_i \geq 0, i = 1, \dots, m$$

- Dual Form (Weak Duality):

$$L(\mathbf{w}, \mathbf{b}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i[y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) - 1 + \xi_i] - \sum_{i=1}^{m}\beta_i\xi_i$$
$$= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{m}\alpha_i y^{(i)}\mathbf{w}^T\mathbf{x}^{(i)} - \sum_{i=1}^{m}\alpha_i y^{(i)}b + \sum_{i=1}^{m}(C - \alpha_i - \beta_i)\xi_i + \sum_{i=1}^{m}\alpha_i$$

  - Notice that α and β are two vectors contains the Lagrange Multiplier λ , there is no affine constraint
  - We minimize the **Lagrangian** (definition of g) $w.r.t$ the primal variables $\mathbf{w}, \mathbf{b}, \boldsymbol{\xi}$

$$\nabla_{\mathbf{w}}L = \mathbf{w} - \sum_{i=1}^{m}\alpha_i y^{(i)}\mathbf{x}^{(i)} = 0\ (1)$$
$$\nabla_{\mathbf{b}}L = -\sum_{i=1}^{m}\alpha_i y^{(i)} = 0\ (2)$$
$$\nabla_{\boldsymbol{\xi}_i}L = C - \alpha_i - \beta_i = 0\ (3)$$
$$i = 1, \dots, m$$

  - Plugin the gradient equality to the Lagrangian to get the **Lagrange dual function**:

$$g(\boldsymbol{\alpha},\boldsymbol{\beta}) = inf_{\mathbf{w},\mathbf{b},\boldsymbol{\xi}}(L(\mathbf{w},\mathbf{b},\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\beta})) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{m}\alpha_i y^{(i)}\mathbf{w}^T\mathbf{x}^{(i)}$$

$$- \sum_{i=1}^{m}\alpha_i y^{(i)} b (= 0 \; according \; to \; (2))$$

$$+ \sum_{i=1}^{m}(C-\alpha_i-\beta_i)\xi_i (= 0 \; according \; to \; (3)) + \sum_{i=1}^{m}\alpha_i$$

$$= \sum_{i=1}^{m}\alpha_i + \frac{1}{2}(\sum_{i=1}^{m}\alpha_i y^{(i)}\mathbf{x}^{(i)})^T(\sum_{i=1}^{m}\alpha_i y^{(i)}\mathbf{x}^{(i)}) - \sum_{i=1}^{m}\alpha_i y^{(i)}(\sum_{i=1}^{m}\alpha_i y^{(i)}\mathbf{x}^{(i)})^T\mathbf{x}^{(i)}$$

$$(according \; to \; (1))$$

$$= \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j(\mathbf{x}^{(i)})^T\mathbf{x}^{(j)}$$

  - The property: $(a+b)^T(c+d) = (a^T+b^T)(c+d) = a^Tc + a^Td + b^Tc + b^Td$ is used in the last step

- Finally, we get the **dual form** of the primary form of SVM optimization:

$$max_\alpha \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j(\mathbf{x}^{(i)})^T\mathbf{x}^{(j)}$$

$$subject \; to \; \nabla_{\mathbf{b}}L = -\sum_{i=1}^{m}\alpha_i y^{(i)} = 0 \; (2)$$

$$0 \le \alpha_i \le C (\alpha_i \ge 0, \beta_i \ge 0, \beta_i = C - \alpha_i \; (3))$$

- Once the dual optimization is solved, we can find the $\boldsymbol{\alpha}^*$, the optimal $\mathbf{w}^*$ can be computed by:

$$\mathbf{w}^* = \sum_{i=1}^{m}\alpha_i^* y^{(i)}\mathbf{x}^{(i)}$$

- How to compute the optimal bias b^*
  - For an item i:

$$y^{(i)}((\mathbf{w}^*)^T\mathbf{x}^{(i)} + b_i^*) = 1$$

$$(y^{(i)})^2((\mathbf{w}^*)^T\mathbf{x}^{(i)} + b_i^*) = y^{(i)}, y \in \{-1, 1\}$$

$$(\mathbf{w}^*)^T\mathbf{x}^{(i)} + b_i^* = y^{(i)}$$

$$b_i^* = y^{(i)} - (\mathbf{w}^*)^T\mathbf{x}^{(i)}$$

- For the whole training set D: $$ b^*={1 \over {|D|}} \sum_{(\textbf x,y)\in D}(y-(\textbf w^*)^T \textbf x) $$
- Strong Duality Form:
  - Complementary slackness: Assume the strong duality holds, and optimal variables are: $\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*, \mathbf{w}^*, \mathbf{b}^*, \boldsymbol{\xi}^*$, we have:

$$p^* = f_0(\mathbf{w}^*) = inf_{\mathbf{w}}(f_0(\mathbf{w}) + \sum_{i=1}^{r}\lambda_i^* f_i(\mathbf{w}) + (\sum_{i=1}^{s}\nu_i^* h_i(\mathbf{w}) = 0))$$

$$\le f_0(\mathbf{w}^*) + \sum_{i=1}^{r}\lambda_i^* f_i(\mathbf{w}^*) + (\sum_{i=1}^{s}\nu_i^* h_i(\mathbf{w}^*) = 0)$$

    - w^* minimize both $f_0(\mathbf{w})$ and $L(\mathbf{w},\lambda^*,\nu^*)$ with Lagrange multiplier λ^* and ν^*
    - $\sum_{i=1}^{r}\lambda_i^* f_i(\mathbf{w}^*) = 0$, notice that $\lambda_i^* f_i(\mathbf{w}^*) \le 0$ for each $i = 1, \ldots, m$(know as **complementary slackness**), which implies that $\lambda_i^* f_i(\mathbf{w}^*) = 0$.Therefore, there are only 2 situations:

- $\lambda_i > 0,\ f_i(\mathbf{w}^*) = 0; i = 1, \ldots, m$
- $\lambda_i = 0,\ f_i(\mathbf{w}^*) < 0; i = 1, \ldots, m$

- In the context of SVM with soft margin, we have the following complementary slackness (assume $C > 0$):

$$\alpha_i^\star \left[y^{(i)} \left((x^{(i)})^T w^\star + b^\star \right) - 1 + \xi_i^\star \right] = 0,\ i = 1, \ldots, m,$$
$$\beta_i^\star \xi_i^\star = 0,\ i = 1, \ldots, m.$$

Combining with

- $y^{(i)}((w^\star)^T x^{(i)} + b^\star) \geq 1 - \xi_i^\star$,
- $\xi_i^\star \geq 0$,
- $C - \alpha_i^\star - \beta_i^\star = 0$,
- $\alpha_i^\star \geq 0$,
- $\beta_i^\star \geq 0$,

we have

- if $\alpha^\star = 0$

$$\begin{cases} \alpha_i^\star = 0 \\ C - \alpha_i^\star - \beta_i^\star = 0 \end{cases} \implies \beta_i^\star = C, \tag{88}$$

$$\begin{cases} \beta_i^\star = C \\ \beta_i^\star \xi_i^\star = 0 \end{cases} \implies \xi_i^\star = 0, \tag{89}$$

$$\begin{cases} \xi_i^\star = 0 \\ y^{(i)}((w^\star)^T x^{(i)} + b^\star) \geq 1 - \xi_i^\star \end{cases} \implies y^{(i)}((w^\star)^T x^{(i)} + b^\star) \geq 1; \tag{90}$$

- if $\alpha^\star = C$

$$\begin{cases} \alpha_i^\star = C \\ \alpha_i^\star \left[y^{(i)} \left((x^{(i)})^T w^\star + b^\star \right) - 1 + \xi_i^\star \right] = 0 \end{cases} \implies y^{(i)} \left((x^{(i)})^T w^\star + b^\star \right) - 1 + \xi_i^\star = 0, \tag{91}$$

$$\begin{cases} y^{(i)} \left((x^{(i)})^T w^\star + b^\star \right) - 1 + \xi_i^\star = 0 \\ \xi_i^\star \geq 0 \end{cases} \implies y^{(i)} \left((x^{(i)})^T w^\star + b^\star \right) \leq 1; \tag{92}$$

- if $0 < \alpha^\star < C$

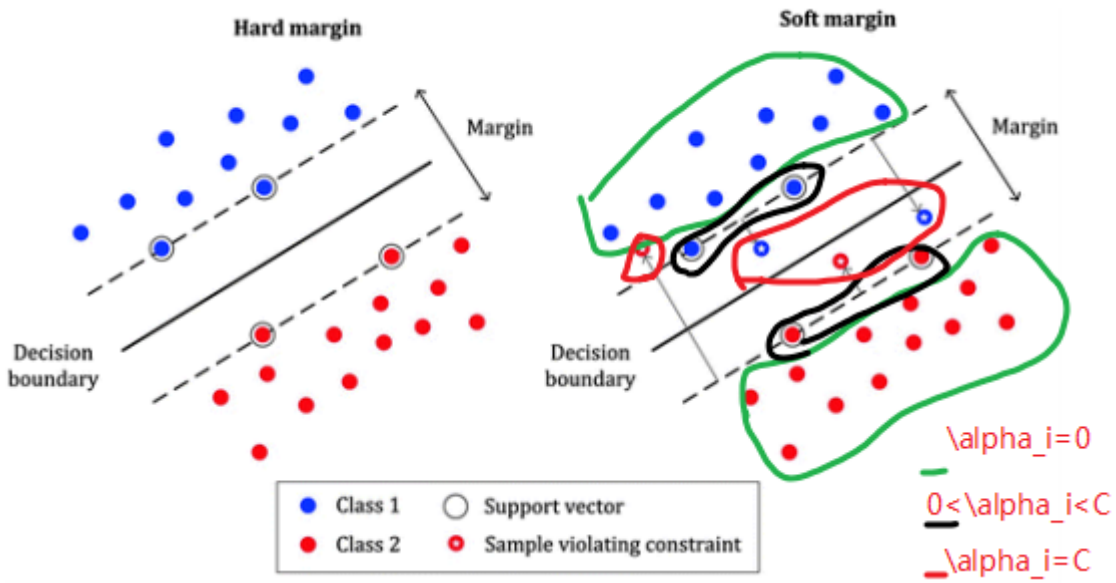
$$\begin{cases} 0 < \alpha^\star < C \\ C - \alpha_i^\star - \beta_i^\star = 0 \end{cases} \implies \beta_i^\star > 0, \tag{93}$$

$$\begin{cases} 0 < \alpha^\star < C \\ \alpha_i^\star \left[y^{(i)} \left((x^{(i)})^T w^\star + b^\star \right) - 1 + \xi_i^\star \right] = 0 \end{cases} \implies y^{(i)} \left((x^{(i)})^T w^\star + b^\star \right) - 1 + \xi_i^\star = 0, \tag{94}$$

$$\begin{cases} \beta_i^\star > 0 \\ \beta_i^\star \xi_i^\star = 0 \end{cases} \implies \xi_i^\star = 0, \tag{95}$$

$$\begin{cases} \xi_i^\star = 0 \\ y^{(i)}((w^\star)^T x^{(i)} + b^\star) - 1 + \xi_i^\star = 0 \end{cases} \implies y^{(i)}((w^\star)^T x^{(i)} + b^\star) = 1. \tag{96}$$

- Support Vectors:



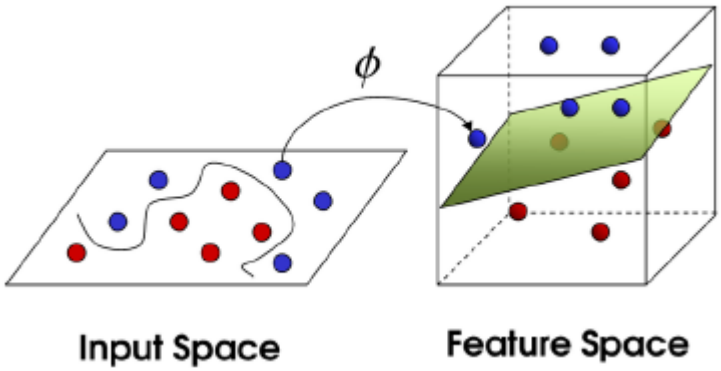
- Why Bother SVM Dual
  - How would one classify a new point x with primal SVM solved
    - We need to explicitly compute the scalar product $\mathbf{w}^T x + b$
  - How would one classify a new point x with dual SVM solved
    - Very efficient as we only compute a few "support vectors" whose α_i's are nonzero (for zero α_i, the constraints are not used)
    - In this situation, weight matrix can be considered as a linear combination of training samples

$$\mathbf{w}^T x + b = (\sum_{i=1}^{m} \alpha_i y^{(i)} \mathbf{x}^{(i)})^T \mathbf{x} + b$$
$$= \sum_{i=1}^{m} \alpha_i y^{(i)} (\mathbf{x}^{(i)})^T \mathbf{x} + b$$

- More importantly, the dual form lends itself easily to the *kernel trick*

Kernel Methods

- The drawback of Linear Models
  - Decision boundaries are by definition linear in the input feature space
  - High bias on complex data
  - Need to **relax the constraint of linear decision boundaries** while **retaining the nice properties of linear classifiers** => increase the dimension of the feature space



- Basis Function Expansion
  - Simple method to increase dimension: Apply a set (with cardinality l) of function $\{\phi_1, \ldots, \phi_l\}$ to the raw feature vector $\mathbf{x}$ to map it in to a new feature space:

$$\phi(\mathbf{x}) = [\phi_1(\mathbf{x}), \ldots, \phi_l(\mathbf{x})]^T$$

  - This is called a *basis function expansion* since $l > n$ in general.
  - This requires that we know the functions $\phi_1, \ldots, \phi_l$ in advance
  - We then define a linear classifier (SVM or logistic regression) in this new feature space:

$$\mathbf{w}^T \phi(\mathbf{x}) + b$$

- Basis Function Expansion Examples
  - **Degree 2 Polynomial Basis**: We include all single features x_j, their squares x_j^2, and all products of two distinct features $x_i x_j$
  - **Degree d Polynomial Basis**: We include all single features x_j, and all unique products of between 2 and d features
  - The problem is that the space complexity of representing the expanded set of features is essentially $O(n^d)$ => Need to be simplify
- Representer Theorem
  - One of the interesting properties of SVMs is that the optimal weight vectors can always be expressed as a weighted linear combination of the data vectors:

$$\mathbf{w} = \sum_{i=1}^{m} \gamma_i x^{(i)}$$

    - This result is called the *representer theorem*
- Dependence on Inner products
    - Plugging this result back in to the SVM objective, we find that the objective only depends on the data through inner products
    - For normalized input vector $\mathbf{x}, \mathbf{z}$, $\mathbf{x}^T\mathbf{z} = ||\mathbf{x}||.||\mathbf{z}|| \; cos\theta = cos\theta$, which represents the similarity between the two vectors

$$min_{\gamma,b,\xi} \frac{1}{2} \sum_{j=1}^{m} \sum_{k=1}^{m} \gamma_j \gamma_k \phi(\mathbf{x}^{(j)})^T \phi(\mathbf{x}^{(k)}) + C \sum_{i=1}^{m} \xi_i$$
$$subject\ to\ y^{(i)}((\sum_{j=1}^{m} \gamma_j \phi(\mathbf{x}^{(j)}))^T \phi(\mathbf{x}^{(i)} + b) \geq 1 - \xi_i,$$
$$\xi_i \geq 0, i = 1, \ldots, m$$

- The Kernel Trick
    - Amazingly, for many useful basis function expansions $\phi(x)$, it is possible to find a function $K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z})$ that can **compute the inner product under the basis expansion without ever explicitly performing the basis function expansion**, where $\mathbf{x}, \mathbf{z} \in \mathbb{R}^n$ are two input feature vectors
    - Such functions are called *kernel functions* and this is known as the “Kernel Trick”
    - In other words, we can also directly learn the parameters γ_i‘s using the kernel trick, without constructing the basis expansion (since the expression $\mathbf{w}^T \phi(\mathbf{x}) + b$ also needs inner product)
    - Interestingly, there **exist kernels for which the basis function expansion implied by the kernel is infinite dimensional**
- Examples of Kernel Functions
    - **Degree d Polynomial Kernel**: $K_p(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z} + c)^d$, which corresponds to a feature mapping to an $\binom{d+n}{d}$ feature space. Where $\mathbf{x}, \mathbf{z} \in \mathbb{R}^n$ are two input feature vectors, and $c \geq 0$ is a constant. If $c = 0$, $K(\mathbf{x}, \mathbf{z})$ reduces to the homogeneous(同質) polynomial kernel
        - proof:
            - Let $k_0, k_1, \ldots, k_n$ denote non-negative integers, such that $\sum_{j=0}^{n} k_j = d$. The multinomial expansion(多項式展開) of the inhomogeneous kernel is then given as

$$K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z} + c)^d = (c + \sum_{j=1}^{n} x_j z_j)^d = (c + x_1 z_1 + \ldots + x_n z_n)^d$$
$$= \sum_{k_0 + .. + k_n = d} \binom{d}{k_0, \ldots, k_n} c^{k_0} (x_1 z_1)^{k_1} \ldots (x_n z_n)^{k_n}$$
$$, (where \binom{d}{k_0, \ldots, k_n} \; means\ the\ joint\ selection\ number)$$
$$= \sum_{k_0 + .. + k_n = d} \binom{d}{k_0, \ldots, k_n} c^{k_0} (x_1^{k_1} \ldots x_n^{k_n})(z_1^{k_1} \ldots z_n^{k_n})$$
$$= \sum_{k_0 + .. + k_n = d} (\sqrt{\binom{d}{k_0, \ldots, k_n} c^{k_0}} \prod_{j=1}^{n} x_j^{k_j})(\sqrt{\binom{d}{k_0, \ldots, k_n} c^{k_0}} \prod_{j=1}^{n} z_j^{k_j})$$

            - Then, the mapping $\phi : \mathbb{R}^n \mapsto \mathbb{R}^l$ is given as the vector:

$$\phi(\mathbf{x}) = [\ldots, \sqrt{\binom{d}{k_0, \ldots, k_n} c^{k_0}} \prod_{j=1}^{n} x_j^{k_j}, \ldots]^T,$$

        - where $(k_0, \dots k_n)$ ranges over all the possible assignments, such that $k_j \geq 0$ for $0 \leq j \leq n$ and $\sum_{j=0}^{n} k_j = d$
        - i.e. partition d items to $n+1$(one bias) buckets => select the separator => $\binom{d+n}{n} = \binom{d+n}{d}$
    - **Gaussain/RBF Kernel:** $K_G(\mathbf{x}, \mathbf{z}) = exp(-\gamma||\mathbf{x} - \mathbf{z}||_2^2)$, which corresponds to an infinite dimensional feature space
        - proof:
            - First, use Taylor expansion of $f(x) = e^x$ at the point $(0, 1)$:

$$f(\theta) = e^{\theta} = \sum_{d=0}^{\infty} \frac{(\theta - 0)^d}{d!} f^d(0) = \sum_{d=0}^{\infty} \frac{\theta^d}{d!}$$
$$= 1 + \theta + \frac{1}{2!}\theta^2 + \frac{1}{3!}\theta^3 + \dots$$

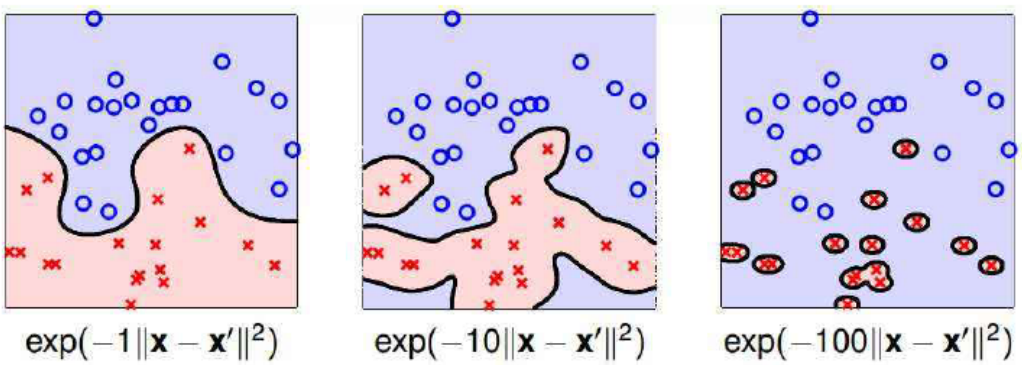
            - Further, noting that $||\mathbf{x} - \mathbf{z}||_2^2 = ||\mathbf{x}||_2^2 + ||\mathbf{z}||_2^2 - 2\mathbf{x}^T\mathbf{z}$, we can rewrite the Gaussian kernel as follows:

$$K(\mathbf{x}, \mathbf{z}) = exp\{-\gamma||\mathbf{x} - \mathbf{z}||_2^2\}$$
$$= exp\{-\gamma||\mathbf{x}||_2^2\} exp\{-\gamma||\mathbf{z}||_2^2\} exp\{2\gamma\mathbf{x}^T\mathbf{z}\}$$
$$= exp\{-\gamma||\mathbf{x}||_2^2\} exp\{-\gamma||\mathbf{z}||_2^2\} \sum_{d=0}^{\infty} \frac{(2\gamma)^d}{d!} (x^T\mathbf{z})^d$$
$$= exp\{-\gamma||\mathbf{x}||_2^2\} exp\{-\gamma||\mathbf{z}||_2^2\} \sum_{d=0}^{\infty} \frac{(2\gamma)^d}{d!} \Big(\sum_{k_1 + \dots + k_n = d} \binom{d}{k_0, \dots, k_n} c^{k_0} \prod_{j=1}^{n} (x_j z_j)^{k_j} \Big)$$
$$= \sum_{d=0}^{\infty} \sum_{k_1 + \dots + k_n = d} (\omega \times exp\{-\gamma||\mathbf{x}||_2^2\} \prod_{j=1}^{n} (x_j)^{k_j}) (\omega \times exp\{-\gamma||\mathbf{z}||_2^2\} \prod_{j=1}^{n} (z_j)^{k_j})$$
$$where\ \omega = \sqrt{\binom{d}{k_0, \dots, k_n} \frac{(2\gamma)^d}{d!}}$$

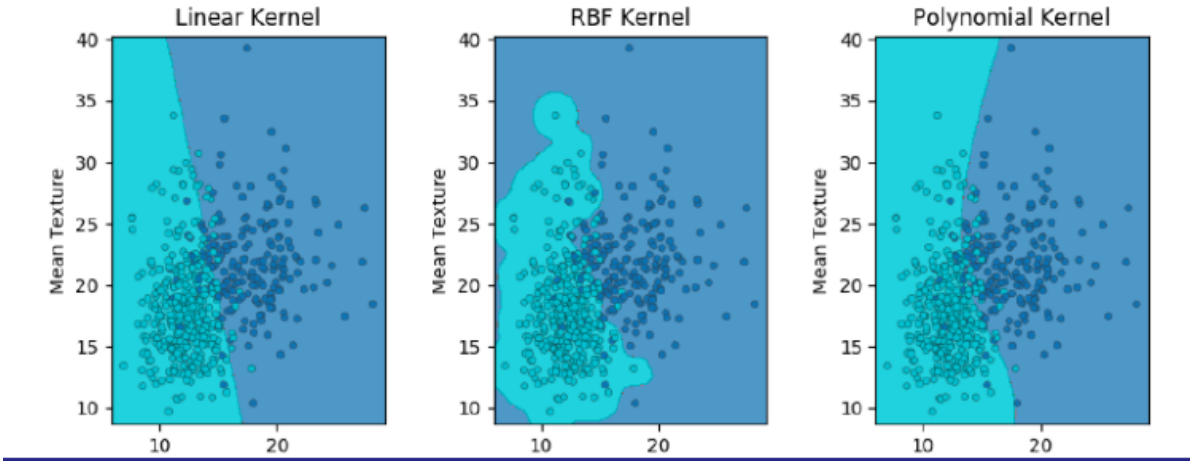
            - Where the mapping $\phi : \mathbb{R}^n \mapsto \mathbb{R}^{\infty}$ is given as the vector

$$\phi(\mathbf{x}) = [\dots, \sqrt{\binom{d}{k_0, \dots, k_n} \frac{(2\gamma)^d}{d!}} exp\{-\gamma||\mathbf{x}||_2^2\} \prod_{j=1}^{n} (x_j)^{k_j}, \dots]^T$$

            - Since d goes to ∞ and each d have $\binom{d+n}{d}$ combinations , the ϕ maps the input space into an infinite dimensional feature space, we obviously cannot compute $\phi(x)$, yet computing the Gaussian kernel $K(\mathbf{x}, \mathbf{z})$ is straightforward and efficient
            - γ also matters since it controls how the higher level Taylor expansions expressed



- The graphs:



- Kernel Trick for other ML methods
  - Many learning methods depend on computing inner products between features, e.g., linear regression
  - For those methods, we can use a kernel function in the place of the inner products, i.e., "kernelizing" the methods, thus, introducing nonlinear features/basis
  - Instead of first transforming the original features into the new feature space and then computing the inner product, we can compute the inner product in the new feature space directly through the kernel function
- Kernel Ridge Regression
  - Recall ridge regression: $min_w \frac{1}{2}||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 + \frac{\lambda}{2}||\mathbf{w}||_2^2$

$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_{n+1})^{-1}\mathbf{X}^T\mathbf{y} = \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I}_m)^{-1}\mathbf{y}$$

$$(P^{-1} + B^T R^{-1} B)^{-1} B^T R^{-1}$$
$$= (P + B^{-1} R (B^T)^{-1}) B^T R^{-1}$$
$$= P B^T R^{-1} + B^{-1}$$
$$= P B^T R^{-1} + P B^T (B^T)^{-1} P^{-1} B^{-1}$$
$$= P B^T (R^{-1} + (B^T)^{-1} P^{-1} B^{-1})$$
$$= P B^T (R + B P B^T)^{-1}$$

For $\underbrace{(X^TX + \lambda I_{n+1})^{-1} X^T}_{U} y$

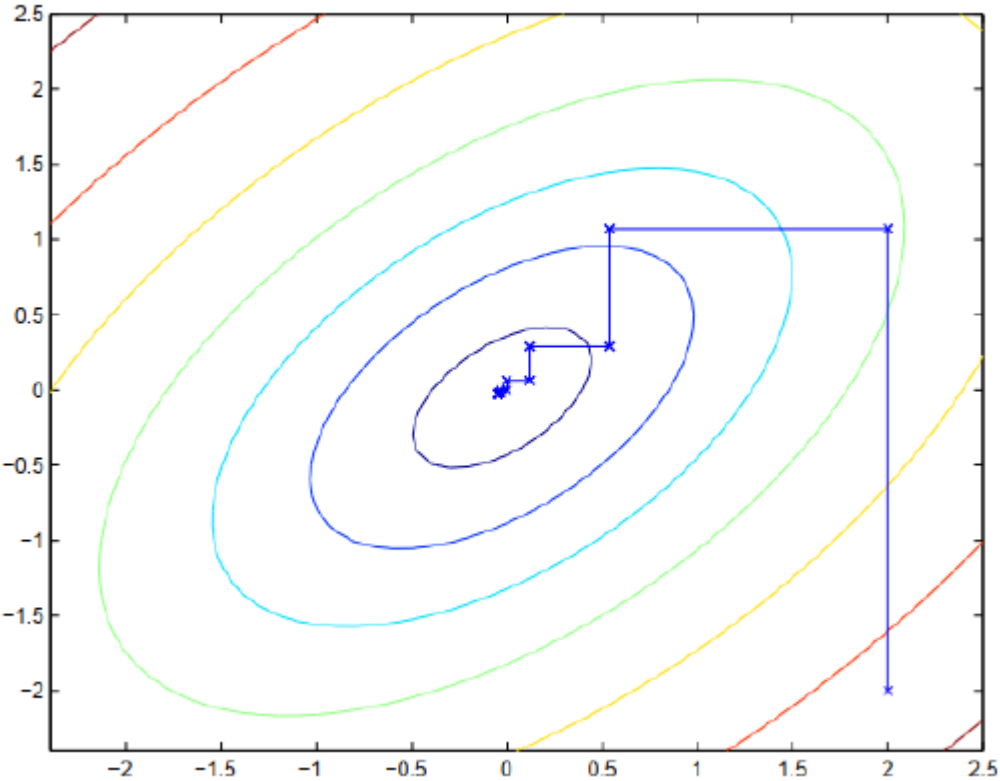
$$(X^TX + \lambda I_{n+1})^{-1} X^T$$
$$(P^{-1} = \lambda I_{n+1},\ B = X,\ R = I_m)$$
$$= (\lambda I_{n+1})^{-1} X^T (I_m + X(\lambda I_{n+1})^{-1})$$
$$= \frac{1}{\lambda} X^T (I_m + \frac{1}{\lambda} XX^T)^{-1}$$
$$= X^T (\lambda I_m + XX^T)^{-1}$$
$$(X^TX + \lambda I_{n+1})^{-1} X^T y$$
$$= X^T (\lambda I_m + XX^T)^{-1} y$$

  - This equation can be rewritten as: $\mathbf{w}^* = \sum_{i=1}^m \alpha_i \mathbf{x}^{(i)}$ with $\boldsymbol{\alpha} = (\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I}_m)^{-1}\mathbf{y}$. This indicates that the solution $\mathbf{w}^*$ must lie in the span of the data cases, which makes intuitive sense because the algorithm is linear in data space
  - For any new point $\mathbf{x}$, we make prediction by: $y = \mathbf{w}^T\mathbf{x} = \mathbf{y}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I}_m)^{-1}\mathbf{X}\mathbf{x}$, where $(\mathbf{X}\mathbf{X}^T)^T = \mathbf{X}\mathbf{X}^T$
  - Kernelizing the method, we have $y = \mathbf{w}^T\mathbf{x} = \mathbf{y}^T(\mathbf{K} + \lambda\mathbf{I}_m)^{-1}\kappa(\mathbf{x})$, where $K_{ij} = \mathrm{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi(\mathbf{x}^{(i)})^T\phi(\mathbf{x}^{(j)})$, $\kappa_i(\mathbf{x}) = \mathrm{K}(\mathbf{x}^{(i)}, \mathbf{x})$ and $\phi(\mathbf{x}) : \mathbb{R}^n \mapsto \mathbb{R}^l$ is a feature mapping

    - No extra computational burden is incurred, but it now becomes a (much more powerful) nonlinear regression model
- Trade-Offs
    - Linear SVM and logistic regression are both discriminative linear classifiers with identical capacity and space complexity. They have very similar convex loss functions and identical regularizers. They are fast and space efficient, but can have high bias
    - Basis expansion requires more space ($O(ml)$ (l functions) for data and $O(l)$ for parameters), but yields non linear classifiers that have lower bias
    - Kernel SVM requires $O(m^2)$(a function with two parameters, takes all the combination among two data vectors) space for storing all the kernel values during training and have $O(m)$ parameters. This can still be much lower than $O(ml)$ for large sets of basis functions. Kernels also yield non-linear classifiers that have lower bias (How does the complexity calculated)
    - Kernel SVM often has at least two **hyperparameters** (C and a kernel hyperparameter). These need to be set jointly, which may be computationally expensive
    - Gaussian kernel SVM has infinite capacity, closely related to KNN
    - Unlike logistic loss, hinge loss is not differentiable, so SVMs require more advanced optimization methods
    - It is somewhat more difficult to form a multi-class SVM than a multi-class logistic regression model, and many implementations use “hacks” like one-vs-all

Sequential Minimal Optimization

- Solving SVM: Coordinate Descent
    - Another method of optimization (compared with GD)
    - Consider the unconstrained optimization problem: $min(\theta_1, \theta_2, \dots, \theta_n)$
        - The coordinate descent method first selects an initial point $\theta^{(0)} \in \mathbb{R}^n$ and repeats:
            - Choose an index i from 1 to n
            - $\theta_i^{(t+1)} = argmin_{\hat{\theta}_i} l(\theta_1^{(t)}, \dots, \theta_{i-1}^{(t)}, \theta_i^{(t)}, \theta_{i+1}^{(t)}, \dots, \theta_n^{(t)})$
    - At each iteration, the algorithm determines a coordinate, and minimizes over the corresponding direction while fixing all other coordinates
    - Coordinate descent is applicable in both differentiable and derivative-free contexts



- Solving SVM: Sequential Minimal Optimization (SMO)
  - SMO is an algorithm for solving the quadratic programming problem that arises during the training of SVMs, invented by John C. Platt in 1998
  - Requirements:
    - Sequential:
      - Not parallel
      - Optimize a set of two Lagrange multiplier
    - Minimal : Optimize the smallest possible sub-problem at each step
    - Optimization: Satisfy the constraint for the **chosen pair** of Lagrange multipliers
  - Recall the dual form of kernelized SVM with soft margin:

$$max_\alpha \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \mathrm{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

$$s.t. \sum_{i=1}^{m} \alpha_i y^{(i)} = 0,$$

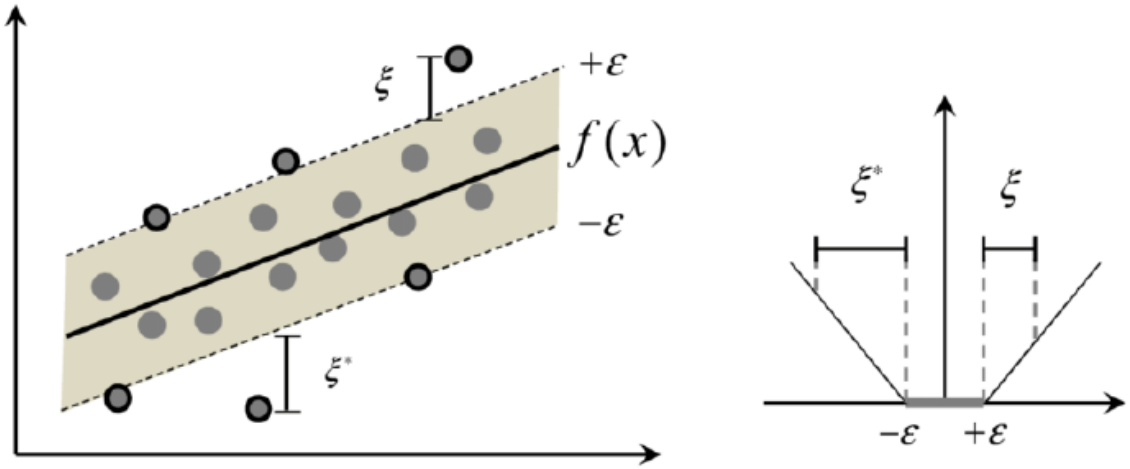
$$0 \leq \alpha_i \leq C, i = 1, \ldots, m$$

    - SMO is derived by decomposing the problem to its extreme and optimizing **a minimal subset of just two data points** at each iteration
      - The sub-problem for two data points admits an analytical solution, eliminating the need to use an iterative quadratic programming optimizer as part of the algorithm
      - To satisfy the second constraint, we need to use two data points, if use one data point, the second constraint will be broken since one of α_i will be changed while others are not => when one α is changed, the other one will be changed to fit the constraint

- At each step SMO chooses two elements α_i and α_j to jointly optimize, finds the optimal values for those two parameters given that all the others are fixed, and updates the $\boldsymbol{\alpha}$ vector accordingly
- The choice of the two points is determined by a **heuristic**, while the optimization of the two multipliers is performed analytically
- Despite needing more iterations to converge, each iteration uses so few operations that the algorithm exhibits an overall speed-up of some orders of magnitude

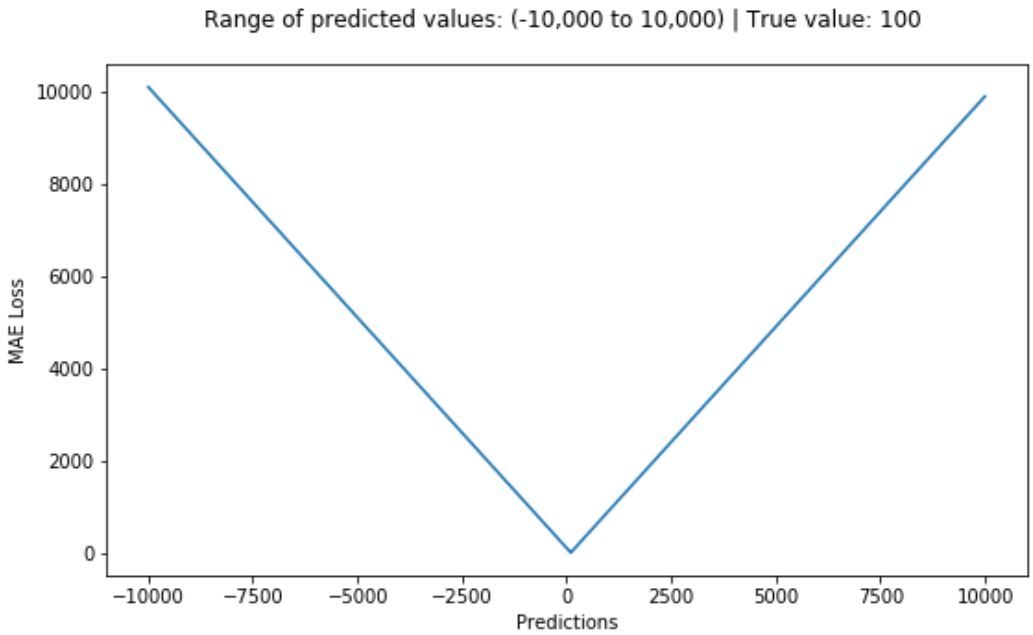
Support Vector Regression (SVR)

- Support vector regression (SVR) is the generalization of SVM to the case of regression
- Use a different objective function. In the following case, the *epsilon insensitive loss*



  - Where MAE is like this:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - y_i^p|}{n}$$



Plot of MAE Loss (Y-axis) vs. Predictions (X-axis)

- Primal Form:

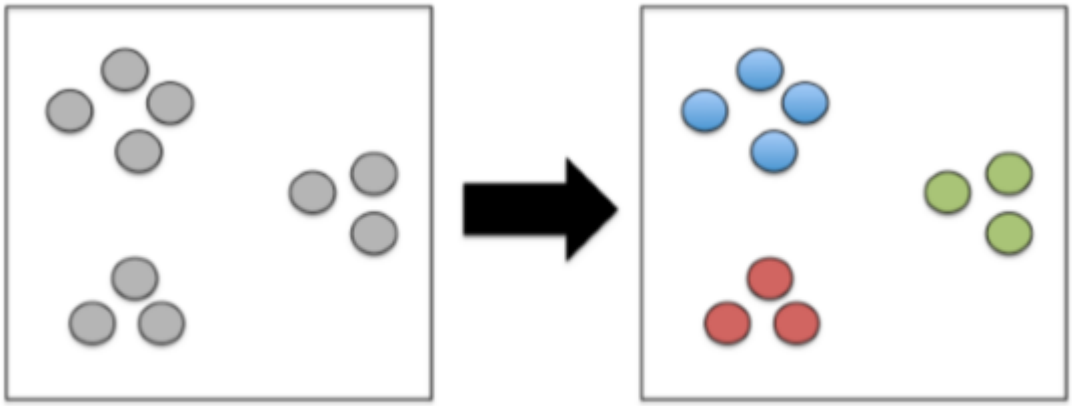
$$min_{\mathbf{w},b,\xi,\xi^*} \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i^*)$$
$$subject\ to\ y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)} - b \leq \epsilon + \xi_i, i = 1, \ldots, m$$
$$\mathbf{w}^T\mathbf{x}^{(i)} + b - y^{(i)} \leq \epsilon + \xi_i^*, i = 1, \ldots, m$$
$$\xi_i, \xi_i^* \geq 0, i = 1, \ldots, m$$

  - Where ξ_i and ξ_i^* represent different level of tolerance of different side of the decision boundary for the data point i
  - It can be generalized to non-linear models using the same representer theorem as in SVM (i.e. the kernel trick)
  - It is a quadratic programming problem that can be efficiently solved by SMO
- Trade-Offs
  - SVR is more robust to outliers than OLS (which minimizing the MSE) due to the loss being linear in the tails instead of quadratic. This is related to a long line of work on robust regression
  - Kernel SVR has low bias and good capacity control,but use of cross validation to select regularization hyperparameters is critical
  - The kernel matrix computation is quadratic in the data dimension, and the model has a support vector property

Lecture 08 K-Means Clustering and Gaussian Mixture Models

Introduction

- The Classification Task: Given a feature vector $\mathbf{x} \in \mathbb{R}^n$ that describes an object that belongs to one of $|Y|$ classes from the set Y, predict which class the object belongs to
- The Clustering Task: Give a collection of data cases $\mathbf{x}^{(i)} \in \mathbb{R}^n$, partition the data cases into groups such that the data cases within each partition are more similar to each other than they are to data cases in other partitions



Exhaustive Clustering

- Define a Clustering
  - Suppose we have m data cases $D = \{\mathbf{x}^{(i)}\}_{i=1,\ldots,m}$
  - A clustering of the m cases into K clusters is a partitioning of D into K mutually disjoint subsets $C = C_1, \ldots, C_k$ such that $C_1 \cup \ldots \cup C_K = D$
  - Suppose we have a function $f(C)$ that takes a partitioning C of the data set D and returns a score with lower scores indicating better clustering
  - The optimal clustering according to f is simply given by: $argmin_C f(C)$
  - The complexity of exhaustive clustering depends on the total number of partitions
- Number of Clusterings
  - The total number of clusterings of a data set with m elements is the Bell number B_m, where $B_0 = B_1 = 1$ and $B_{m+1} = \sum_{k=0}^{m} \binom{m}{k} B_k$
  - Wrong: The recursive relation means that, for the first cluster we choose $m - k$ elements, and calculate how many ways of combination for k elements left, and sum all the possibility of k's (It is wrong because for a partition, the order of choice doesn't matter)
  - True proof: Let $x^{(1)}$ to be a fixed to the first cluster, we can choose $m - k$ elements in the same cluster with $x^{(1)}$, then the order is fixed. Therefore, the equation is for $m + 1$ instead of m
  - The complexity of exhaustive clustering scales with B_m and is thus computationally totally intractable for general scoring functions

- We will need either approximation algorithms or scoring functions with special properties

K-Means Clustering

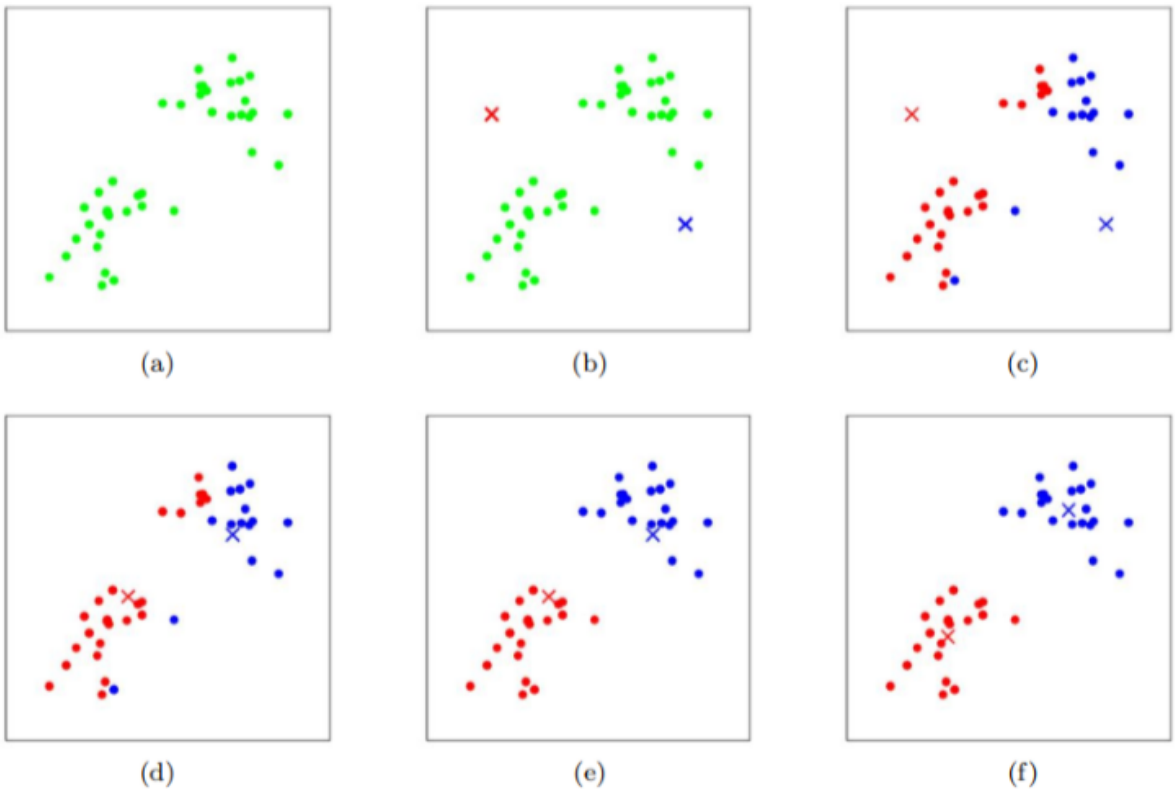
- The K-means algorithm is an iterative optimization algorithm for clustering that alternates between two steps
- The algorithm maintains a set of K cluster centroids or prototypes $\{\mu_k\}$ that represent the average (mean) feature vector of the data cases in each cluster
- Algorithm: Suppose we let $z^{(i)}$ indicate which cluster $\mathbf{x}^{(i)}$ belongs to and $\mu_k \in \mathbb{R}^n$ be the cluster centroid/prototype for cluster k. The two main steps of the algorithm can then be expressed as follows:
  - In the first step, the distance between each data case and each prototype is computed, and each data case is assigned to the nearest prototype

$$\boxed{1} \quad z^{(i)} = \arg\min_k ||\boldsymbol{\mu}_k - \mathbf{x}^{(i)}||_2^2 \text{ (the assignment step)}$$

  - In the second step, the prototypes are updated to the mean of the data cases assigned to them

$$\boxed{2} \quad \boldsymbol{\mu}_k = \frac{\sum_{i=1}^m \mathbb{I}[z^{(i)} = k]\mathbf{x}^{(i)}}{\sum_{i=1}^m \mathbb{I}[z^{(i)} = k]} \text{ (the update step)}$$

- Example:



- The K-Means Objective
  - The K-means algorithm attempts to minimize the sum of the within-cluster variation over all clusters (also called the within-cluster sum of squares):

$$\min_{\mathbf{z},\boldsymbol{\mu}} \ell(\mathbf{z}, \boldsymbol{\mu}) = \min_{\mathbf{z},\boldsymbol{\mu}} \sum_{i=1}^{m} \|\mathbf{x}^{(i)} - \boldsymbol{\mu}_{z^{(i)}}\|^2$$

- It can be shown that K-means is exactly coordinate descent on l. Specifically, the assignment step minimizes l with respect to z while holding μ fixed, and the update step minimizes l with respect to μ while holding z fixed. Thus, must monotonically decrease, and the value of must converge
- Note that l has many local optimal in general, each corresponding to a different clustering of the data. Finding the global optimum is not computationally tractable => **highly sensitive to initialization**

- Initialization
  - It is common to perform multiple random re-starts of the algorithm and take the clustering with the minimal total variation
  - Common initializations include setting the initial centers to be randomly selected data cases, setting the initial partition to a random partition, and selecting centers using a “furthest first”-style heuristic (more formally known as K-means++)
  - It often helps to initially to run with $Klog(K)$ clusters, then merge clusters to get down to K and run the algorithm from that initialization
- Issues
  - Only works with Euclidean distance. An alternative based on Manhattan distance is called the K-medians algorithm
  - Pre-processing like re-scaling/normalizing features can completely change the results
  - We need some way to determine the “right” number of clusters to focus on. We want to cluster on salient differences between data cases, not noise
  - The number of iterations to convergence is often small (like 20), but examples can be constructed that require an exponential number of steps to converge
  - Results in a hard assignment of data cases to clusters, which may be a problem if there are outliers

Expectation Maximization

- EM solves a maximum likelihood problem of the form:

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{m} log\, p(\mathbf{x}^{(i)}; \boldsymbol{\theta}) = \sum_{i=1}^{m} log \sum_{z^{(i)} \in Z} p(\mathbf{x}^{(i)}, z^{(i)}; \boldsymbol{\theta})$$

  - $\boldsymbol{\theta}$: Parameters of the probabilistic model we try to find
  - $\{\mathbf{x}^{(i)}\}_{i=1,\ldots,m}$: Observed training examples
  - $\{z^{(i)}\}_{i=1,\ldots,m}$: Unobserved latent variables
  - Z: Finite set of categorical values that $z^{(i)}$ can take on
- EM Derivation

$$\sum_{i=1}^{m} log \sum_{z^{(i)} \in Z} p(\mathbf{x}^{(i)}, z^{(i)}; \boldsymbol{\theta})$$

$$= \sum_{i=1}^{m} log \sum_{z^{(i)} \in Z} q(z^{(i)}) \frac{p(\mathbf{x}^{(i)}, z^{(i)}; \boldsymbol{\theta})}{q(z^{(i)})}, (where\ q\ is\ an\ arbitrary\ distribution\ on\ z\ )$$

$$= \sum_{i=1}^{m} log\, \mathbb{E}_{z^{(i)} \sim q}[\frac{p(\mathbf{x}^{(i)}, z^{(i)}; \boldsymbol{\theta})}{q(z^{(i)})}] \geq \sum_{i=1}^{m} E_{z^{(i)} \sim q}[log\, \frac{\mathrm{p}(\mathbf{x}^{(\mathrm{i})}, \mathrm{z}^{(\mathrm{i})}; \boldsymbol{\theta})}{\mathrm{q}(\mathrm{z}^{(\mathrm{i})})}]$$

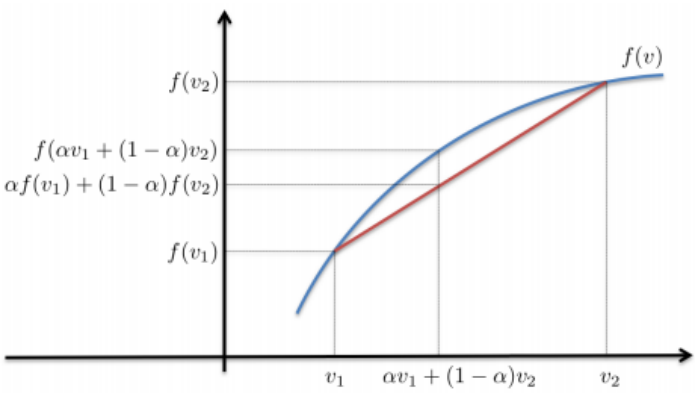
$$= \sum_{i=1}^{m} \sum_{z^{(i)} \in Z} q(z^{(i)}) log\, p(\mathbf{x}^{(i)}, z^{(i)}; \boldsymbol{\theta}) - \sum_{i=1}^{m} \sum_{z^{(i)} \in Z} q(z^{(i)}) log\, q(z^{(i)})$$

- Jensen's inequality:
  - Suppose $f : \mathbb{R} \mapsto \mathbb{R}$ is concave, then for all probability distributions p and all probability distributions p and all functions $g : \mathbb{R} \mapsto \mathbb{R}$, we have:

    $$f(\mathbb{E}_{v \sim p}[g(v)]) \geq E_{v \sim p}[f(\mathrm{g}(v))]$$

    with equality iff f **is an affine function or** $g(v)$ **is a constant**

Illustration:
$p(V = v_1) = \alpha,$
$p(V = v_2) = 1 - \alpha$



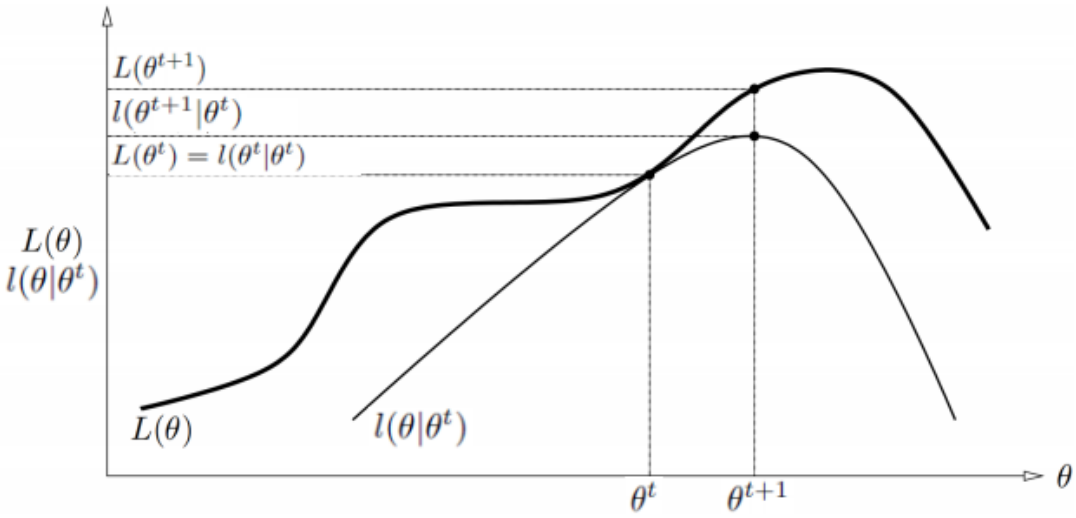
- Lower bound:

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{m} log \sum_{z^{(i)} \in Z} p(\mathbf{x}^{(i)}, z^{(i)}; \boldsymbol{\theta})$$

$$\geq \sum_{i=1}^{m} \sum_{z^{(i)} \in Z} q(z^{(i)}) log\, p(\mathbf{x}^{(i)}, z^{(i)}; \boldsymbol{\theta}) - \sum_{i=1}^{m} \sum_{z^{(i)} \in Z} q(z^{(i)}) log\, q(z^{(i)})$$

$$= l(\boldsymbol{\theta})$$

  - Since $f(x) = log(x)$, f is impossible to be an affine, the equality holds iff $g(q) = \frac{p(\mathbf{x}^{(i)}, z^{(i)}; \boldsymbol{\theta})}{q(z^{(i)})}$ is a constant, which can be achieved when $q(z^{(i)}) = p(z^{(i)} | \mathbf{x}^{(i)}; \boldsymbol{\theta}) \propto p(\mathbf{x}^{(i)}, z^{(i)}; \boldsymbol{\theta})$
- The EM algorithm repeatedly carries out the following two steps until convergence:
  - E-step: For each i, compute $q(z^{(i)}) = p(z^{(i)} | \mathbf{x}^{(i)}; \boldsymbol{\theta})$
    - We do not fill in the unobserved $z^{(i)}$ with hard values, but find a posterior distribution $q(z^{(i)})$, given $x^{(i)}$ and θ
  - M-step: $\boldsymbol{\theta} = arg\, max_{\boldsymbol{\theta}} l(\boldsymbol{\theta}) = arg\, max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \sum_{z^{(i)} \in Z} q(z^{(i)}) log\, p(\mathbf{x}^{(i)}, z^{(i)}; \boldsymbol{\theta})$ (the second part can be ignored)
    - M-step objective is upper bounded by true objective, and is equal to true objective at current parameter estimate. M-step optimization can be done efficiently in most cases
- A example:



    - $L = l$ at θ_t because of the E-step
- EM Convergence
    - Assuming $\boldsymbol{\theta}^{(t)}$ and $\boldsymbol{\theta}^{(t+1)}$ are the parameters from two successive iterations of EM, we have

$$
\begin{aligned}
& L(\boldsymbol{\theta}^{(t+1)}) \\
= l(\boldsymbol{\theta}^{(t)}) &= \sum_{i=1}^{m} \sum_{z^{(i)} \in Z} q(z^{(i)}; \boldsymbol{\theta}^{(t)}) log \frac{p(\mathbf{x}^{(i)}, z^{(i)}; \boldsymbol{\theta}^{(t)})}{q(z^{(i)}; \boldsymbol{\theta}^{(t)})} \\
&\leq \sum_{i=1}^{m} \sum_{z^{(i)} \in Z} q(z^{(i)}; \boldsymbol{\theta}^{(t)}) log \frac{p(\mathbf{x}^{(i)}, z^{(i)}; \boldsymbol{\theta}^{(t+1)})}{q(z^{(i)}; \boldsymbol{\theta}^{(t)})} \\
& \leq l(\boldsymbol{\theta}^{(t+1)}) \\
&= \sum_{i=1}^{m} \sum_{z^{(i)} \in Z} q(z^{(i)}; \boldsymbol{\theta}^{(t+1)}) log \frac{p(\mathbf{x}^{(i)}, z^{(i)}; \boldsymbol{\theta}^{(t+1)})}{q(z^{(i)}; \boldsymbol{\theta}^{(t+1)})} \\
& = L(\boldsymbol{\theta}^{(t+1)})
\end{aligned}
$$

    - Hence, EM causes the likelihood to increase monotonically, and if bounded above, converges to some L^*
- Remark: If we define:

$$
J(q, \boldsymbol{\theta}) = \sum_{i=1}^{m} \sum_{z^{(i)} \in \mathcal{Z}} q(z^{(i)}) \log \frac{p(\mathbf{x}^{(i)}, z^{(i)}; \boldsymbol{\theta})}{q(z^{(i)})},
$$

EM can also be viewed coordinate ascent on J, in which the E-step maximizes J w.r.t. q, and the M-step maximizes J with respect to $\boldsymbol{\theta}$

    - Proof:
        - For the objective function:

$$J(q,\theta) = \sum_{i=1}^{m} \sum_{z^{(i)} \in Z} q(z^{(i)}) log \frac{p(x^{(i)}, z^{(i)}; \theta)}{q(z^{(i)})}$$
$$where \sum_{z^{(i)} \in Z} q(z^{(i)}) = 1$$

- We what to maximize J, which is a **concave function**, we reverse the Lagrange dual form:
  - Lagrangian is :

$$L(q,\theta) = J(q,\theta) + \lambda(1 - \sum_{z^{(i)} \in Z} q(z^{(i)}))$$
$$= \sum_{i=1}^{m} \sum_{z^{(i)} \in Z} q(z^{(i)}) log \frac{p(x^{(i)}, z^{(i)}; \theta)}{q(z^{(i)})} + \lambda(1 - \sum_{z^{(i)} \in Z} q(z^{(i)}))$$

  - Take the partial derivative of L w.r.t $q(z^{(i)})$ and setting it to zero, we have:

$$log \frac{p(x^{(i)}, z^{(i)}; \theta)}{q(z^{(i)})} + q(z^{(i)}) \frac{q(z^{(i)})}{p(x^{(i)}, z^{(i)}; \theta)} (- \frac{p(x^{(i)}, z^{(i)}; \theta)}{q(z^{(i)})^2}) - \lambda = 0$$
$$log \frac{p(x^{(i)}, z^{(i)}; \theta)}{q(z^{(i)})} = 1 + \lambda$$
$$g(z^{(i)}) = \frac{p(x^{(i)}, z^{(i)}; \theta)}{q(z^{(i)})} = e^{1+\lambda} (a\ constant)$$

  - Therefore, when we maximize $J\ w.r.t\ q$, we fullfill the condition that g is a constant, which make sure once we use $q(z^{(i)}; \theta^{(t+1)})$, we maximize the function J which is an equivalent to maximizing $l(q,\theta)$. At the same time l is equal to L, which means we are maximizing L

$$L(\theta^{(t+1)}, q(z^i, \theta^{(t+1)})) > L(\theta^{(t+1)}, q(z^i, \theta^{(t)}))$$
$$> l(\theta^{(t+1)}, q(z^i, \theta^{(t)})) > l(\theta^{(t)}, q(z^i, \theta^{(t)})) = L(\theta^{(t)}, q(z^i, \theta^{(t)}))$$

  - Continue calculating the $q(z^{(i)})$

$$\sum_{z^{(i)} \in Z} q(z^{(i)}) = \sum_{z^{(i)} \in Z} \frac{p(x^{(i)}, z^{(i)}; \theta)}{e^{1+\lambda}} = \frac{p(x^{(i)}; \theta)}{e^{1+\lambda}} = 1$$
$$q(z^{(i)}) = \frac{p(x^{(i)}, z^{(i)}; \theta)}{e^{1+\lambda}} = p(z^{(i)} | x^{(i)}; \theta)$$

  - Which is exactly the M-step

Gaussian Mixture Models

- Mixture Models
  - A mixture model is a probabilistic clustering model that is the unsupervised analogue(模拟，近似) of the Bayes optimal classifier where the **unknown assignment of data cases** to clusters take the place of the known class labels
  - We let $\mathbf{x}^{(i)}$ be a data case and $z^{(i)} \in \{1, \ldots, K\}$ be the index of the cluster $\mathbf{x}^{(i)}$ belongs to. $z^{(i)}$ is often called the mixture indicator variable or the latent class (i.e. ubobserved)
  - Each cluster k specifies it's own distribution over the feature vectors $p(\mathbf{x}^{(i)} | z^{(i)} = k)$
  - We also have a discrete distribution $p(z^{(i)} = k) = \theta_k$, which describes the prior probability that $\mathbf{x}^{(i)}$ belongs to cluster k
- Data Distribution
  - The joint distribution of the feature($\mathbf{x}^{(i)}$) and the mixture indicator variable($z^{(i)}$) is :

$$p(\mathbf{x}^{(i)}, z^{(i)} = k) = p(\mathbf{x}^{(i)}|z^{(i)} = k)p(z^{(i)} = k)$$

- In clustering, we don't know that the right value of the mixture indicator variable is a priori, but we can marginalize it away to obtain a probability distribution on the feature vector only

$$p(\mathbf{x}^{(i)}) = \sum_{k=1}^{K} p(\mathbf{x}^{(i)}|z^{(i)} = k)p(z^{(i)} = k)$$

- Mixture Component Distributions: Common model choices for $p(\mathbf{x}|z = k)$ are:
  - Categorical: $\Pi_{j=1}^{n}\Pi_{x\in X_j}\theta_{x,j|k}^{\mathbb{I}[x_j=x]}$, where X_j contains all possibilities of j_{th} feature of $\mathbf{x}$
  - Independent Gaussian: $\Pi_{j=1}^{n} N(x_j; \mu_{j|k}, \sigma_{j|k}^2)$
  - Multivariate Gaussian: $N(\mathbf{x}, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$
- Learning:
  - Given data set $D = \{\mathbf{x}^{(i)}\}_{i=1,\ldots,m}$, we can learn the mixture model parameters by maximizing the log probability of the data give the parameters:

$$l = \sum_{i=1}^{m} log(\sum_{k=1}^{K} p(\mathbf{x}^{(i)}|z^{(i)} = k)p(z^{(i)} = k))$$

  - While we can do this directly using gradient-based optimization, it's often faster to use a special algorithm called *Expectation Maximization*
- Expectation Maximization for Gaussian Mixture Models
  - E-Step: In the first step of the algorithm, we compute the posterior probability that each data case belongs to each cluster using the Bayes rule:

$$w_k^{(i)} = p(z^{(i)} = k|\mathbf{x}^{(i)}) = \frac{\theta_k N(\mathbf{x}^{(i)}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k'=1}^{K} \theta_{k'} N(\mathbf{x}^{(i)}; \boldsymbol{\mu}_{k'}, \boldsymbol{\Sigma}_{k'})}$$

  - M-Step: In the second step, we update the parameters using the weighted averages:

$$\theta_k = \frac{\sum_{i=1}^{m} w_k^{(i)}}{m}, \qquad \boldsymbol{\mu}_k = \frac{\sum_{i=1}^{m} w_k^{(i)} \mathbf{x}^{(i)}}{\sum_{i=1}^{m} w_k^{(i)}}$$

$$\boldsymbol{\Sigma}_k = \frac{\sum_{i=1}^{m} w_k^{(i)} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_k)(\mathbf{x}^{(i)} - \boldsymbol{\mu}_k)^T}{\sum_{i=1}^{m} w_k^{(i)}}$$

- A Special Case
  - Suppose we fix $\theta_k = \frac{1}{K}$ and $\sum_k = I$. In this case we have:

$$N(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{\sqrt{|(2\pi)^n I|}} exp(-\frac{1}{2}||\mathbf{x}^{(i)} - \boldsymbol{\mu}_k||_2^2)$$

  - and we obtain the following special case of the EM algorithm for multivariate Gaussians:

multivariate Gaussians.

$$w_k^{(i)} = \frac{\exp\left(-\frac{1}{2}||\mathbf{x}^{(i)} - \boldsymbol{\mu}_k||_2^2\right)}{\sum_{k'=1}^{K} \exp\left(-\frac{1}{2}|\mathbf{x}^{(i)} - \boldsymbol{\mu}_{k'}||_2^2\right)},$$

$$\boldsymbol{\mu}_k = \frac{\sum_{i=1}^{m} w_k^{(i)} \mathbf{x}^{(i)}}{\sum_{i=1}^{m} w_k^{(i)}}$$

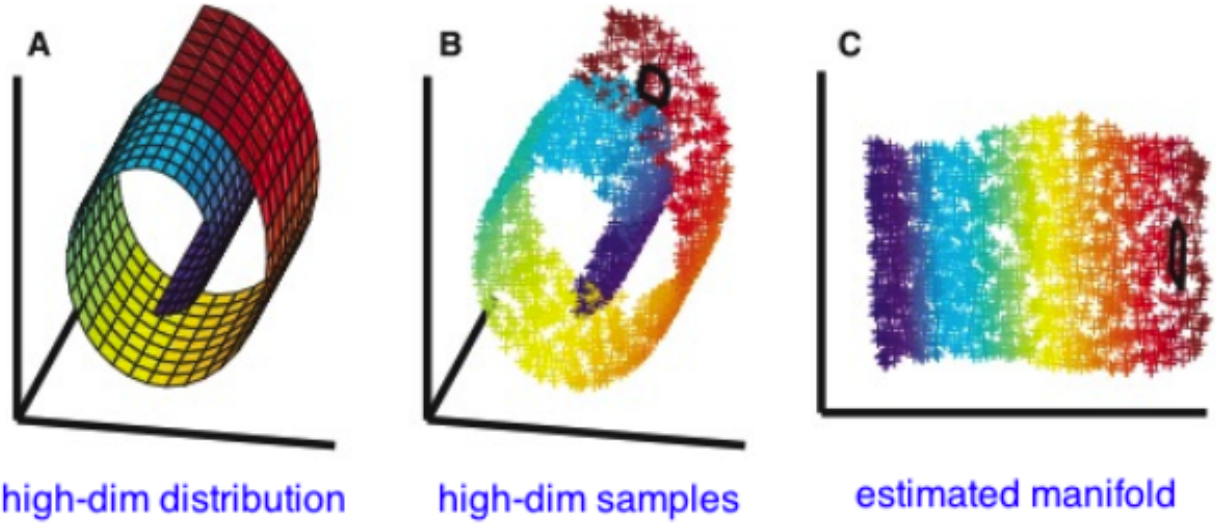
This is often referred to as soft K-means.

- Trade-Offs
  - We can see that the original *K*-means algorithm performs **hard assignments during clustering**, and implicitly assumes all clusters will have an equal number of points assigned as well as a unit covariance matrix
  - EM for Gaussian mixture models relaxes all of these assumptions. The objective still has multiple local optima, but **EM also produces a guaranteed non-decreasing sequence of objective function values**
  - EM can also be used with any component densities/distributions to customize the model to a given data set
  - As with *K*-means, initialization is important, but the same heuristics can be applied

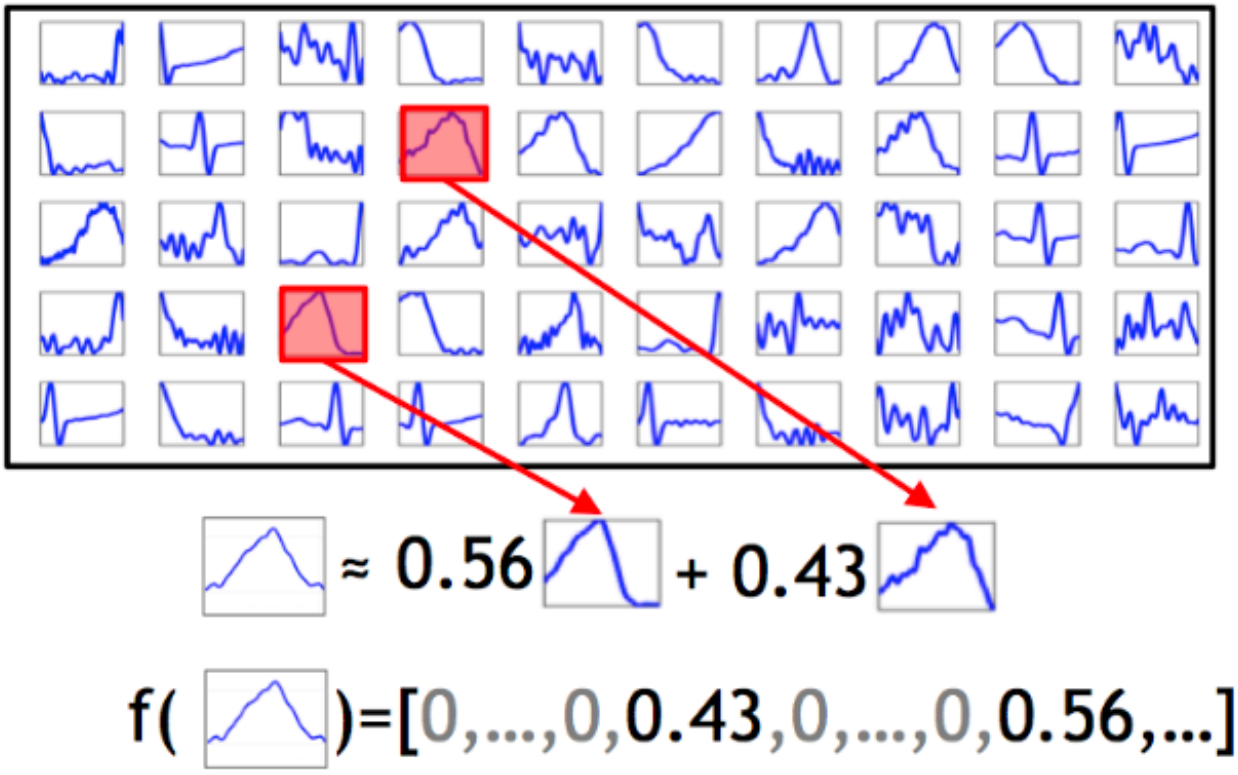
Lecture 09 Linear Dimensionality Reduction

Dimensionality Reduction

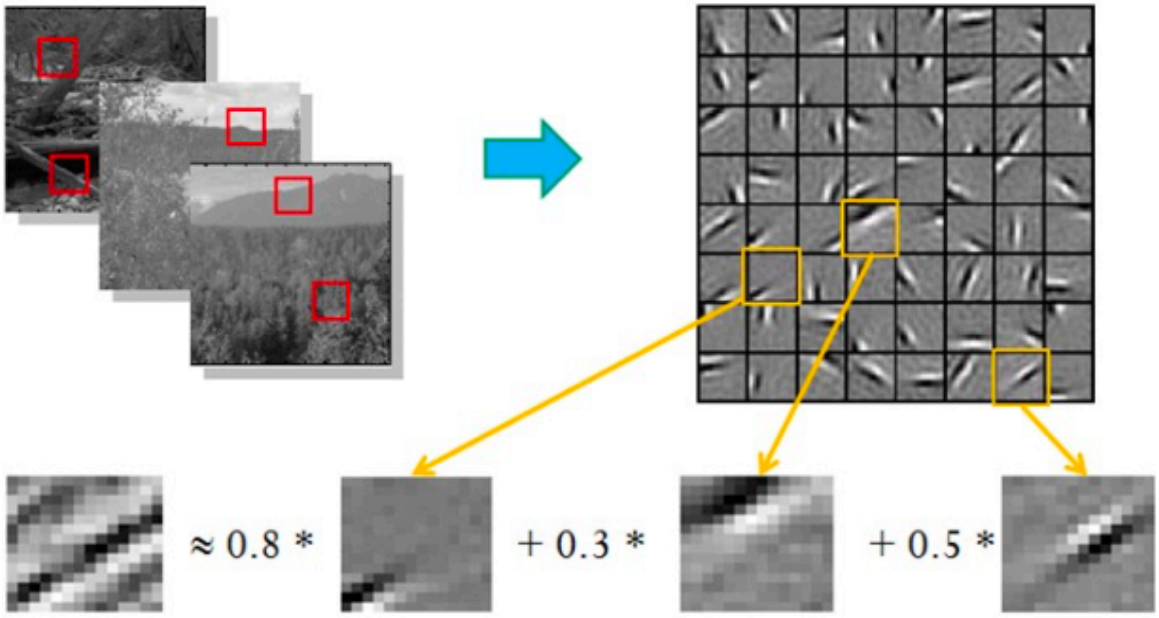
- Definition of dimensionality reduction task:
  - Given a collection of feature vectors $\{\mathbf{x}^{(i)} \in \mathbb{R}^n\}$, map the feature vectors into a lower dimensional space $\{\mathbf{z}^{(i)} \in \mathbb{R}^l\}$, where $l < n$, while preserving certain "properties" of the data



- Example
  - Representing Time Series



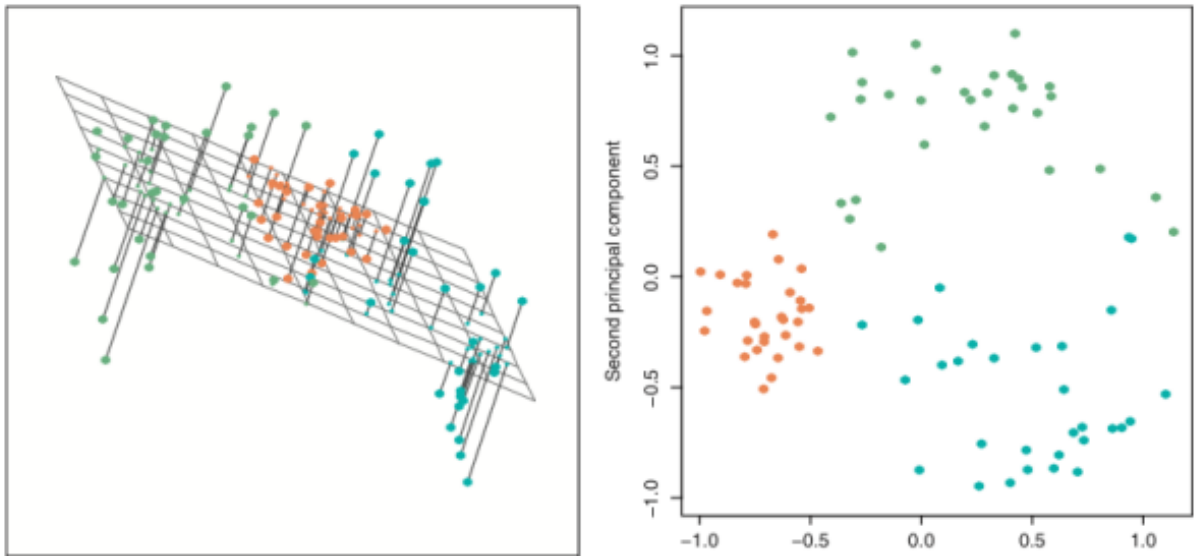
  - Representing Natural Image Patches



- Application
  - Can be used as a **pre-processing step** to enable more accurate classification and regression on manifold data
  - Very low dimensional embeddings (i.e., $l = 2, 3$) can be used to **visualize complex manifold-structured data** as in the previous example
  - Can be used to **"de-noise"** data by **projecting to lower-dimensional space** and then **projecting back to the feature space**
- Dimensionality Reduction vs. Feature Selection
  - The goal of feature selection is to **remove features that are not informative** with respect to the class label. This obviously reduces the dimensionality of the feature space
  - Dimensionality reduction can be used to find a **meaningful lower dimensional feature space** for manifold-structured data even when there is information in each of the feature dimensions so that none can be discarded (Linear independent)
  - Another important property of dimensionality reduction is that it is **unsupervised**. It will attempt to **preserve(保留) structure in the data that could be useful for a range of supervised problems**
  - Unlike feature selection, which is a supervised task, dimensionality reduction can sometimes **discard structure useful to a particular supervised task** thereby increasing error
- Dimensionality Reduction vs. Data Compression
  - While dimensionality reduction can be seen as a simplistic form of data compression, it is not equivalent to it, as the goal of data compression is to reduce the ***entropy*** of the representation **not only** the dimensionality
  - For example, in lossless data compression, *arithmetic coding* encodes the entire data into a single number, an arbitrary-precision fraction q where $0.0 \leq q < 1.0$. In some science fiction books, this is paraphrased as a pinpoint representing all information of the whole universe

Linear Dimensionality Reduction

- Linear Dimensionality Reduction:
  - The simplest dimensionality reduction methods assumes that the observed high dimensional data vectors $\mathbf{x}^{(i)} \in \mathbb{R}^n$ lie on an l-dimensional linear subspace with $\mathbb{R}^n$ (Project the data vectors to a lower dimensional space)



- Assumption: Firstly, assume all the data points are already on a lower dimensional space
  - Mathematically, the linear subspace assumption can be written as follows:

$$\mathbf{x}^{(i)} = \sum_{k=1}^{l} z_k^{(i)} \mathbf{b}_k \tag{1}$$

  - Where $\mathbf{b}_k = [b_{k_1}, b_{k_2}, \ldots, b_{k_n}]^T$ for b_{k_i} $where\ i = 1, \ldots, l$ are a set of basis vectors describing an l-dimensional linear sub-space of $\mathbb{R}^n$ and $z_k^{(i)}$'s are real-valued weights on those basis vectors (reduced data vectors)
- Connection to linear regression
  - This expression is exactly linear regression where $x_j^{(i)}$ is the target, $z_k^{(i)}$"s are the weights, and b_{kj} for each k are the features

$$x_j^{(i)} = \sum_{k=1}^{l} z_k^{(i)} b_{kj} \tag{2}$$

  - This observation is also true if we swap the roles of the weights and the features
  - However, unlike linear regression, we only know what corresponds to the targets. We must learn both the features and the weights
- Matrix Form
  - If we let $\mathbf{X}$ be the data matrix $X_{ij} = x_j^{(i)}$, $\mathbf{Z}$ be a matrix where $Z_{ik} = z_k^{(i)}$, and $\mathbf{B}$ be a matrix where $B_{kj} = b_{kj}$, we can express $\mathbf{X}$ as follows:

$$\mathbf{X} = \mathbf{Z} \times \mathbf{B} \tag{3}$$

- $\mathbf{Z} \in \mathbb{R}^{m \times l}$ is often referred to as the **factor loading matrix** while $\mathbf{B} \in \mathbb{R}^{l \times n}$ are refered to as the **latent factors**
- Most real world data will be subject to noise. If we assume that $\epsilon \in \mathbb{R}^{m \times n}$ is a matrix of noise values from some probability distribution, we have:

$$\mathbf{X} = \mathbf{Z} \times \mathbf{B} + \epsilon \tag{4}$$

- Learning
  - The learning problem for linear dimensionality reduction is to estimate values for both Z and B given only the noisy observations X
  - One possible learning criteria is to minimize the sum of squared errors when reconstructing $\mathbf{X}$ from $\mathbf{Z}$ and $\mathbf{B}$. This leads to:

$$argmin_{\mathbf{Z},\mathbf{B}} ||\mathbf{X} - \mathbf{ZB}||_F^2 \tag{5}$$

  - where $||\mathbf{A}||_F$ is the Frobenius norm of matrix $\mathbf{A}$ ((the square root of the sum of the squares of all matrix entries => Least Square)
- Some properties of **trace** in a matrix and the Frobenius norm:

We first recall the definition of the *trace* of a square matrix $A \in \mathbb{R}^{n \times n}$

$$\mathrm{tr}A = \sum_{i=1}^{n} A_{jj} = A_{11} + A_{22} + \cdots + A_{nn}. \tag{135}$$

The trace is a linear mapping. That is,

$$\mathrm{tr}(A + B) = \mathrm{tr}A + \mathrm{tr}B, \tag{136}$$

and

$$\mathrm{tr}cA = c\mathrm{tr}A. \tag{137}$$

We note a trivial but useful property of the trace

$$\mathrm{tr}A = \mathrm{tr}A^T = \sum_{j=1}^{n} A_{jj}. \tag{138}$$

This result implies that for $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{m\times n}$

$$\mathrm{tr}AB^T = \mathrm{tr}(AB^T)^T = \mathrm{tr}BA^T \tag{139}$$

and

$$\mathrm{tr}A^TB = \mathrm{tr}(A^TB)^T = \mathrm{tr}B^TA. \tag{140}$$

We prove another important property of the trace: for $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times m}$, $\mathrm{tr}AB = \mathrm{tr}BA$.

Proof.

$$\mathrm{tr}AB = \sum_{i=1}^{m}(AB)_{ii} = \sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}b_{ji} = \sum_{j=1}^{n}\sum_{i=1}^{m} b_{ji}a_{ij} = \sum_{j=1}^{n}(BA)_{jj} = \mathrm{tr}BA. \tag{141}$$

□

More generally, the trace is invariant under cyclic permutations, which can be easily proved using Eq. (141). For example, for $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times l}$, and $C \in \mathbb{R}^{l\times m}$, we have

$$\mathrm{tr}ABC = \mathrm{tr}A(BC) = \mathrm{tr}BCA = \mathrm{tr}B(CA) = \mathrm{tr}CAB. \tag{142}$$

We now prove that for $A \in \mathbb{R}^{m\times n}$

$$\|A\|_F^2 = \mathrm{tr}AA^T = \mathrm{tr}A^TA. \tag{143}$$

Proof.

$$\|A\|_F^2 = \sum_{i=1}^{m}\sum_{j=1}^{n} A_{ij}^2 = \sum_{i=1}^{m}\left(\sum_{j=1}^{n} A_{ij}A_{ij}\right) = \sum_{i=1}^{m}\left(\sum_{j=1}^{n} A_{ij}A_{ji}^T\right) \tag{144}$$

$$= \sum_{i=1}^{m}(AA^T)_{ii} = \mathrm{tr}AA^T. \tag{145}$$

In Eq. (144), we note the fact that the entry in the ith row and jth column of A is equal to the entry in the jth row and ith column of A^T. □

Using Eq. (143), we are able to decompose the loss function in linear dimensionality reduction as

$$\begin{aligned}
\ell(Z,B) = \|X - ZB\|_F^2 &= \mathrm{tr}(X - ZB)(X - ZB)^T \\
&= \mathrm{tr}(XX^T - X(ZB)^T - ZBX^T + ZB(ZB)^T) \\
&= \mathrm{tr}(XX^T - 2ZBX^T + ZBB^TZ^T) \qquad (146) \\
&= \mathrm{tr}XX^T - 2\mathrm{tr}ZBX^T + \mathrm{tr}ZBB^TZ^T. \qquad (147)
\end{aligned}$$

In Eq. (146), we make use of the trace identity $\mathrm{tr}AB^T = \mathrm{tr}BA^T$. To derive the closed-form update rules of alternating least squares in solving the linear dimensionality reduction problem, we first take the derivative of ℓ w.r.t. Z

$$\begin{aligned}
\frac{\partial \ell(Z,B)}{\partial Z} &= -2\frac{\partial}{\partial Z}\mathrm{tr}Z(BX^T) + \frac{\partial}{\partial Z}\mathrm{tr}Z(BB^T)Z^T \\
&= -2XB^T + ZBB^T + Z(BB^T)^T \\
&= -2XB^T + 2ZBB^T, \qquad (148)
\end{aligned}$$

where we make use of two derivatives of traces[22]

$$\frac{\partial}{\partial X}\mathrm{tr}XA = A^T \qquad (149)$$

and

$$\frac{\partial}{\partial X}\mathrm{tr}XAX^T = XA + XA^T. \qquad (150)$$

Setting the partial derivative to zero, we have

$$\begin{aligned}
ZBB^T &= XB^T \qquad (151) \\
Z &= XB^T(BB^T)^{-1} \qquad (152) \\
Z^T &= (XB^T(BB^T)^{-1})^T \qquad (153) \\
&= (BB^T)^{-1}BX^T. \qquad (154)
\end{aligned}$$

In Eq. (154), we exploit the fact that the inverse of the transpose of an invertible matrix is equal to the transpose of its inverse, *i.e.*, $(A^T)^{-1} = (A^{-1})^T$. Similarly, taking the derivative of ℓ w.r.t. B, we have

$$\begin{aligned}
\frac{\partial \ell(Z,B)}{\partial B} &= -2\frac{\partial}{\partial B}\mathrm{tr}ZBX^T + \frac{\partial}{\partial B}\mathrm{tr}ZBB^TZ^T \\
&= -2\frac{\partial}{\partial B}\mathrm{tr}B(X^TZ) + \frac{\partial}{\partial B}\mathrm{tr}(Z^TZ)BB^T \\
&= -2Z^TX + Z^TZB + (Z^TZ)^TB \\
&= -2Z^TX + 2Z^TZB, \qquad (155)
\end{aligned}$$

where we use the derivatives of trace[23]

$$\frac{\partial}{\partial X}\mathrm{tr}AXX^T = AX + A^TX. \tag{156}$$

Setting the partial derivative to zero, we have

$$\begin{aligned} Z^TZB &= Z^TX \\ B &= (Z^TZ)^{-1}Z^TX. \end{aligned} \tag{157}$$

- Learning Algorithm
  - Not surprisingly, we can obtain a solution to this learning problem by leveraging the OLS solution to linear regression. The algorithm is often referred to as Alternating Least Squares (ALS)
  - Starting from a random initialization, ALS iterates between **assuming Z are known features and optimizing B as the unknown weights**, and **assuming that B are the known features and optimizing Z as the unknown weights** (is it coordinate descent ?)

$$\mathbf{B} \leftarrow (\mathbf{Z}^T\mathbf{Z})^{-1}\mathbf{Z}^T\mathbf{X},$$

$$\mathbf{Z}^T \leftarrow (\mathbf{B}\mathbf{B}^T)^{-1}\mathbf{B}\mathbf{X}^T$$

- **Lack of Uniqueness** of Optimal Parameters
  - Suppose we run the ALS algorithm to convergence and obtain estimates for $\mathbf{Z}^*$ and $\mathbf{B}^*$ s.t.:

$$l^* = ||\mathbf{X} - \mathbf{Z}^*\mathbf{B}^*||_F^2 \tag{6}$$

  - Note that if we let $\mathbf{R} \in \mathbb{R}^{l\times l}$ be an arbitrary $l \times l$ invertible matrix, then we obtain exactly the same value $l*$ of the objective function for the alternate parameters $Z^{'} = Z^*R$ and $B^{'} = R^{-1}B^*$

$$\begin{aligned} l^* &= ||\mathbf{X} - \mathbf{Z}^*(\mathbf{I})\mathbf{B}^*||_F^2 \\ &= ||\mathbf{X} - \mathbf{Z}^*(\mathbf{R}\mathbf{R}^{-1})\mathbf{B}^*||_F^2 \\ &= ||\mathbf{X} - \mathbf{Z}^{'}\mathbf{B}^{'}||_F^2 \end{aligned} \tag{7}$$

Singular Value Decomposition

- SVD: (resource)
  - Let $\mathbf{X}$ be an $m \times n$ matrix, with $m \geq n$. It can be factorized

$$\mathbf{X} = \mathbf{U}\begin{pmatrix}\mathbf{\Sigma}\\ 0\end{pmatrix}\mathbf{V}^T \tag{8}$$

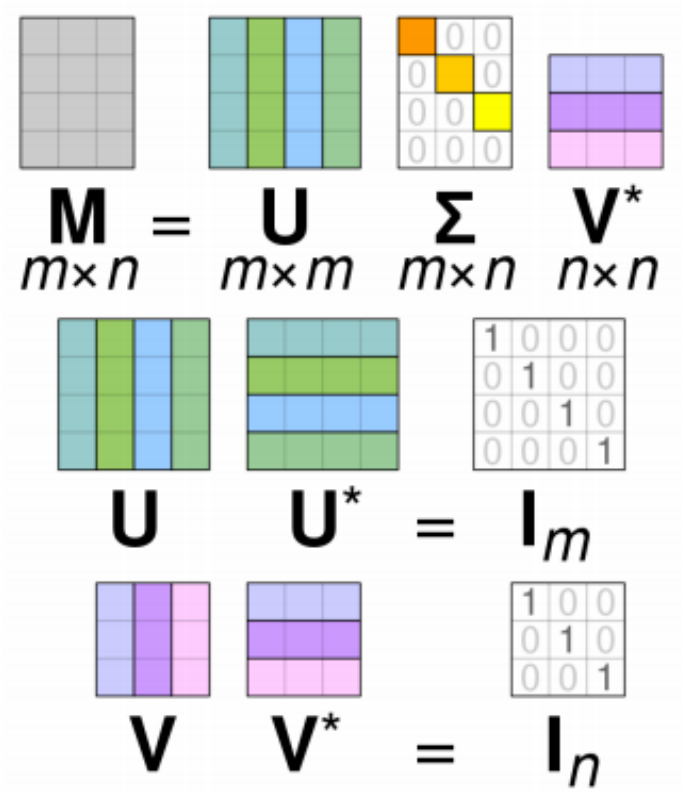
  - where $\mathbf{U} \in \mathbb{R}^{m\times m}$ and $\mathbf{V} \in \mathbb{R}^{n\times n}$ are orthogonal, i.e.,

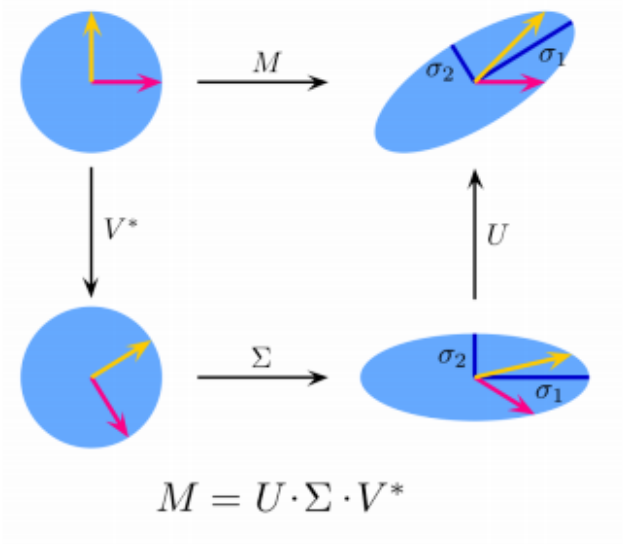
$$\begin{aligned} \mathbf{U}^T\mathbf{U} &= \mathbf{U}\mathbf{U}^T = \mathbf{I}_m \\ \mathbf{V}^T\mathbf{V} &= \mathbf{V}\mathbf{V}^T = \mathbf{I}_n \end{aligned} \tag{9}$$

  - and $\mathbf{\Sigma} \in \mathbb{R}^{n\times n}$ is diagonal (This matrix rescale the vectors in V)

$$\mathbf{\Sigma} = diag(\sigma_1, \sigma_2, \ldots, \sigma_n), \sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0 \tag{10}$$

- The quantities σ_i's are called *singular(eigen) values* of $\mathbf{X}$ and the columns of $\mathbf{U}$ and $\mathbf{V}$ are called the left and right *singular(eigen) vectors* of $\mathbf{X}$, respectively





- Upper left: The unit disc with the two canonical unit vectors
- Upper right: transformed with $\mathbf{M}$
- Lower left: The action of $\mathbf{V}^T$. This is just a rotation
- Lower right: The action of $\mathbf{\Sigma V}^T$. $\mathbf{\Sigma}$ scales in vertically and horizontally

- Properties of each matrix
  - Matrix $\mathbf{U}$

    - Columns of $\mathbf{U}$ consist of eigenvectors of $\mathbf{X}\mathbf{X}^T$

$$\begin{aligned}\mathbf{X}\mathbf{X}^T &= (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T \\ &= (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)(\mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T) \\ &= \mathbf{U}\mathbf{\Sigma}\mathbf{\Sigma}^T\mathbf{U}^T = \mathbf{U}(\mathbf{\Sigma})^2\mathbf{U}^T\end{aligned} \tag{11}$$

    - Orthogonal and has unit norm, $i.e.$, $\mathbf{U}\mathbf{U}^T = \mathbf{U}^T\mathbf{U} = \mathbf{I}$
  - Matrix $\mathbf{\Sigma}$
    - Diagonal
    - Square roots of the eigenvalues of both $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}\mathbf{X}^T$
    - Shows importance of each eigenvector
  - Matrix $\mathbf{V}$
    - Columns of $\mathbf{V}$ consist of eigenvectors of $\mathbf{X}^T\mathbf{X}$

$$\begin{aligned}\mathbf{X}^T\mathbf{X} &= (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T) \\ &= (\mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T)(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T) \\ &= \mathbf{V}\mathbf{\Sigma}^T\mathbf{\Sigma}\mathbf{V}^T = \mathbf{V}(\mathbf{\Sigma})^2\mathbf{V}^T\end{aligned} \tag{12}$$

    - Orthogonal and has unit form. $i.e.$, $\mathbf{V}\mathbf{V}^T = \mathbf{V}^T\mathbf{V} = \mathbf{I}$
- Reduced-Form Singular Value Decomposition (SVD)
  - If only $l < min\{m, n\}$ singular values are non-zeros, the SVD of $\mathbf{X} \in \mathbb{R}^{m\times n}$ can be represented in *reduced form* as follows:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}_l\mathbf{V}^T = \sum_{j=1}^{l}\sigma_j\mathbf{u}_j\mathbf{v}_j^T \tag{13}$$

  - where $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_l]$ and $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_l]$, satisfying

$$\mathbf{U}^T\mathbf{U} = \mathbf{I}_l,\ \mathbf{V}^T\mathbf{V} = \mathbf{I}_l \tag{14}$$

  - and $\mathbf{\Sigma}_l = diag(\sigma_1, \sigma_2, \ldots, \sigma_l), \sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_l \geq 0$
  - $\mathbf{u}_j\mathbf{v}_j^T \in \mathbb{R}^{m\times n}$ is the product of a column vector $\mathbf{u}_j$ and the transpose of a column vector $\mathbf{v}_j$. It has rank 1. That is, $\mathbf{X}$ is a weighted summation of l rank-1 matrices in $\mathbb{R}^{m\times n}$
    - $\mathbf{\Sigma}_l$ stores a combination of l rows in $\mathbf{V}$ (it **scales** each vectors in $\mathbf{V}$)
    - When $m > n$, it works like:
      - $\mathbf{U}$ is a matrice in $\mathbb{R}^{m\times m}$, and the l column vectors $\mathbf{u}^{(j)}$ corresponds to the σ_i are moved to the corresponding places
      - $\mathbf{V}$ is a matrice in $\mathbb{R}^{n\times n}$, and the l row vectors $\mathbf{v}^{(i)}$ corresponds to the σ_i are moved to the corresponding places
      - $\mathbf{\Sigma}_l$ and $\mathbf{V}^{'}$ are matrices in $\mathbb{R}^{l\times l}$
      - $\begin{pmatrix}\mathbf{V}^{'} & 0\\ 0 & 0\end{pmatrix}$ and $\begin{pmatrix}\mathbf{\Sigma}_l & 0\\ 0 & 0\end{pmatrix}$ are matrices in $\mathbb{R}^{m\times n}$

$$\mathbf{U}\begin{pmatrix}(V^{'})^T & 0\\ 0 & 0\end{pmatrix} = \mathbf{U}\begin{pmatrix}\mathbf{\Sigma}_l & 0\\ 0 & 0\end{pmatrix}\mathbf{V}^T \tag{15}$$

    - When $l = n$

$$\mathbf{U}\begin{pmatrix}(V^{'})^T\\0\end{pmatrix}=\mathbf{U}\begin{pmatrix}\mathbf{\Sigma}\\0\end{pmatrix}\mathbf{V}^T \tag{16}$$

Eckart-Young-Mirsky Theorem

- Given an $m\times n$ matrix $\mathbf{X}$ of rank $r\leq min\{m,n\}$ (resource1 resource2)and its singular value decomposition $\mathbf{X}=\mathbf{U}\mathbf{\Sigma}_r\mathbf{V}^T$, with singular values $\sigma_1\geq\sigma_2\geq\ldots\geq\sigma_r$, then among all $m\times n$ matrices of lower rank $l<r$, the best approximation is $\mathbf{Y}^*=\mathbf{U}\mathbf{\Sigma}_l\mathbf{V}^T$, **where Σ_l is the diagonal matrix with singular values $\sigma_1,\sigma_2,\ldots,\sigma_l$ in the sense that**

$$||\mathbf{X}-\mathbf{Y}^*||_F^2=min\{||\mathbf{X}-\mathbf{Y}||_F^2;\mathbf{Y}\in\mathbb{R}^{m\times n},rank\mathbf{Y}\leq l\} \tag{17}$$

- This is not a convex optimization problem since the condition function $rank$ is **neither convex nor concave**:

$$\begin{gathered}X_1=\begin{bmatrix}1&0\\0&0\end{bmatrix},\ X_2=\begin{bmatrix}0&0\\0&1\end{bmatrix} \\ rank(X_1)=rank(X_2)=1 \\ Y=\theta X_1+(1-\theta)X_2=\begin{bmatrix}\theta&0\\0&(1-\theta)\end{bmatrix},where\ \theta\in[0,1] \\ rank(Y)=2,when\ \theta\in(0,1) \\ 2=rank(\theta X_1+(1-\theta)X_2)>\theta\,rank(X_1)+(1-\theta)\,rank(X_2)=1 \\ X_1=\begin{bmatrix}1&0\\0&1\end{bmatrix},X_2=\begin{bmatrix}0&0\\0&-1\end{bmatrix} \\ rank(X_1)=2,\ rank(X_2)=1 \\ Y=\theta X_1+(1-\theta)X_2=\begin{bmatrix}\theta&0\\0&2\theta-1\end{bmatrix},where\ \theta\in[0,1] \\ rank(\mathbf{Y})=1,when\ \theta=\frac{1}{2} \\ 1=rank(\theta X_1+(1-\theta)X_2)<\theta\,rank(X_1)+(1-\theta)\,rank(X_2)=\frac{3}{2}\end{gathered} \tag{18}$$

- Proof: Since the Frobenius norm is preserved under unitary transformations, we have

$$\begin{gathered}||\mathbf{X}-\mathbf{Y}||_F^2=||\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T-\mathbf{Y}||_F^2=||\mathbf{\Sigma}-\mathbf{U}^T\mathbf{Y}\mathbf{V}||_F^2 \\ (for\ orthoganal\ matrix\ \mathbf{U}) \\ ||\mathbf{U}\mathbf{X}||_F^2=tr((\mathbf{U}\mathbf{X})^T(\mathbf{U}\mathbf{X}))=tr(\mathbf{X}^T\mathbf{U}^T\mathbf{U}\mathbf{X}) \\ =tr(\mathbf{X}^T\mathbf{X})=||\mathbf{X}||_F^2 \\ ||\mathbf{X}\mathbf{U}||_F^2=tr((\mathbf{X}\mathbf{U})^T(\mathbf{X}\mathbf{U}))=tr(\mathbf{U}^T\mathbf{X}^T\mathbf{X}\mathbf{U}) \\ =tr(\mathbf{X}^T\mathbf{X}\mathbf{U}\mathbf{U}^T)=tr(\mathbf{X}^T\mathbf{X})=||\mathbf{X}||_F^2 \\ \text{Denoting }\mathbf{Z}=\mathbf{U}^T\mathbf{Y}\mathbf{V},\text{an }m\times n\text{ matrix of rank }l,\text{a direct caculation gives} \\ ||\mathbf{\Sigma}-\mathbf{Z}||_F^2=\sum_{ij}|\Sigma_{ij}-Z_{ij}|^2=\sum_{i=1}^{r}|\sigma_i-Z_{ij}|^2+\sum_{i>r}|Z_{ii}|^2(=0)+\sum_{i\neq j}|Z_{ij}|^2\end{gathered} \tag{19}$$

  - If $\mathbf{Z}$ is not diagonal, set $Z_{ij}=0(i\neq j)$ will have a better result
  - if we don't use exactly the largest (mean value) $\sigma_1,\sigma_2,\ldots,\sigma_l$ to match the larger l σ in

$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r$, there will also be some better result
    - Therefore, $\mathbf{Z}^* = diag(\sigma_1, \sigma_2, \ldots, \sigma_l) = \mathbf{\Sigma}_l; \mathbf{Y}^* = \mathbf{U}\mathbf{\Sigma}_l\mathbf{V}^T$
- Trade-Offs
    - Minimum Frobenius norm linear dimensionality reduction is nice because it will find the globally optimal linear sub-space
    - The basic ALS(Alternating Least Square) algorithm scales as $O(l^3 + ml^2 + nl^2 + mnl)$ per pair of iterations. The full SVD algorithm scales as $O(min(mn^2, nm^2))$. Fast approximations exist for both problems
    - A significant limitation of linear dimensionality reduction is that it can fail to achieve a useful compact(dense) representation of the data if the underlying manifold is not actually linear (non-unique)
    - As with clustering, the rank l of the latent sub-space is a free parameter

Principal Component Analysis

- Eigenvectors
    - Assume $\mathbf{A} \in \mathbb{R}^{n \times n}, \mathbf{v} \in \mathbb{C}^{n \times 1}, and\ \lambda \in \mathbb{C}$
    - If $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ the $\mathbf{v}$ is a right eigenvector of $\mathbf{A}$ with eigenvalue λ
    - If $\mathbf{A}^T\mathbf{v} = \lambda\mathbf{v}(\mathbf{v}^T\mathbf{A} = \lambda\mathbf{v}^T)$ the $\mathbf{v}$ is a right eigenvector of $\mathbf{A}$ with eigenvalue λ
    - If $\mathbf{A}$ is symmetric so that $\mathbf{A} = \mathbf{A}^T$, then the left and right eigenvectors of $\mathbf{A}$ are the same with the same eigenvectors ($\mathbf{A}\mathbf{v} = \mathbf{A}^T\mathbf{v} = \lambda\mathbf{v}$)
    - Why there is complex eigenvalue => because for rotation matrix, only times a real number, the direction of each vector are not changed
- Eigendecomposition
    - Let $\mathbf{V} \in \mathbb{R}^{n \times n}$ be a matrix whose column $\mathbf{v}_i$ are n linearly independent eigenvectors of $\mathbf{A}$ with $\mathbf{\Lambda}$ the corresponding diagonal matrix of eigenvalues such that $\mathbf{\Lambda}_{ii} = \lambda_i$. Then:

$$\mathbf{A}\mathbf{V} = \sum_{j=0}^{n} \mathbf{A}\mathbf{v}^{(j)} = \sum_{j=0}^{n} \lambda_j \mathbf{v}^{(j)} = \mathbf{V}\mathbf{\Lambda} \tag{20}$$
$$\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$$
$$\mathbf{V}^{-1}\mathbf{A}\mathbf{V} = \mathbf{\Lambda}$$

    - Only *diagnalizable matrices* have **eigendecomposition** and eigenvalues (!!!eigenvalue is different from singular value)
    - If $\mathbf{A}$ is symmetric, we can choose n orthonormal eigenvectors so that $||\mathbf{v}_i||_2 = 1, \mathbf{v}_i^T\mathbf{v}_j = 0$ and n real eignevalues $\lambda_n \in \mathbb{R}$. As a result, we have:

$$\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^T = \mathbf{A}^T = \mathbf{V}\mathbf{\Lambda}^T\mathbf{U}^T = \mathbf{V}\mathbf{\Lambda}\mathbf{U}^T (m = n) \tag{21}$$
$$\mathbf{U} = \mathbf{V}$$
$$\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1} = \sum_{i=1}^{n} \lambda_i \mathbf{v}_i \mathbf{v}_i^T$$

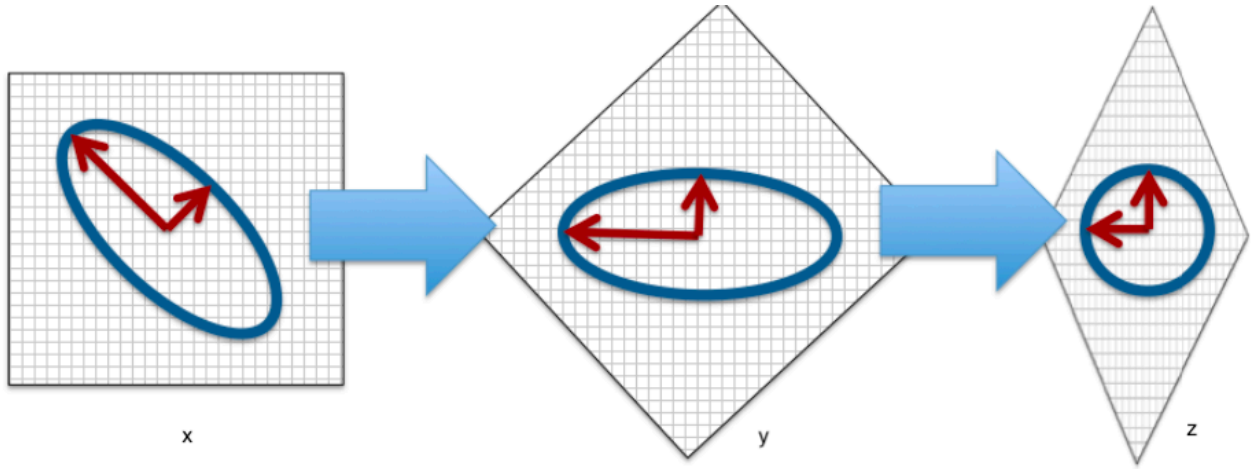
    - eigenvalues and singularvalues are the same for symmetric matrices
    - If symmetric it is diagnalizable, if diagnalizable it may not be symmetric
- Representation of a vector in the Eigen Basis

- Similarly, if $\mathbf{a}$ is an arbitrary vector, then we can also represent $\mathbf{a}$ using the basis provided by the eigenvectors $\mathbf{V}$ of a real symmetric matrix $\mathbf{A}(\mathbf{a}\mathbf{a}^T)$. We obtain:

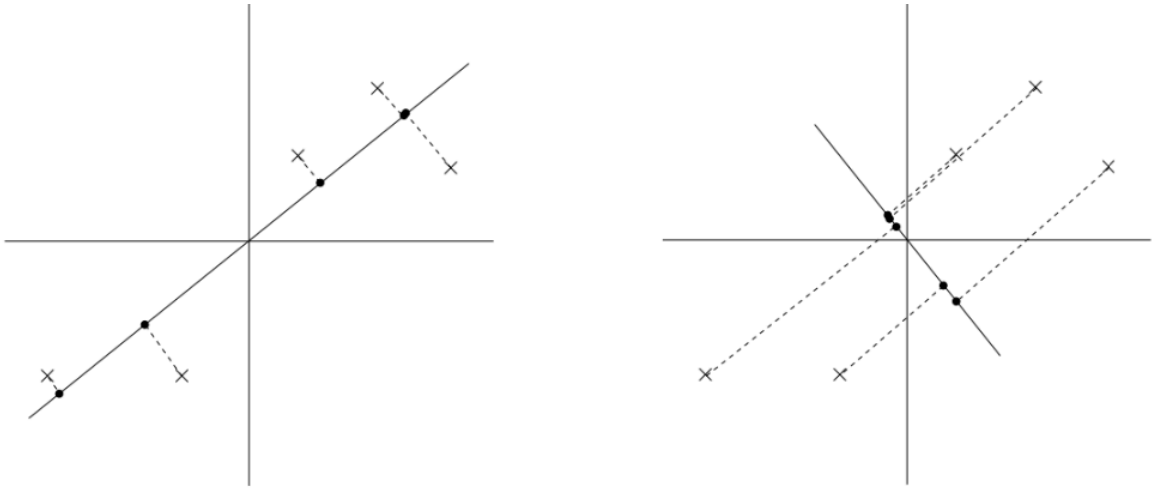
$$\mathbf{a} = \sum_{i=1}^{n} \alpha_i \mathbf{v}_i \tag{22}$$
$$\mathbf{a}^T \mathbf{v}_i = \sum_{j \neq i}^{n} \alpha_j \mathbf{v}_j^T \mathbf{v}_i + \alpha_i \mathbf{v}_i^T \mathbf{v}_i = \alpha_i$$

- Geometry: If A is a **real symmetric matrix** with *positive* eigenvalues, then the quadratic equation $\mathbf{x}^T \mathbf{A}\mathbf{x}$ = 1 defines an ellipsoid in an n-dimensional space, which provides a different way of thinking about these operations:

$$\begin{gathered} \mathbf{x}^T \mathbf{A}\mathbf{x} = 1, \mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T \\ \mathbf{x}^T \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T \mathbf{x} = 1, denotes\ \mathbf{y} = \mathbf{V}^T \mathbf{x} \\ \mathbf{y}^T \mathbf{\Lambda}\mathbf{y} = 1 => \mathbf{y}^T \mathbf{\Lambda}^{\frac{1}{2}} \mathbf{\Lambda}^{\frac{1}{2}} \mathbf{y} = 1, denotes\ \mathbf{z} = \mathbf{\Lambda}^{\frac{1}{2}} \mathbf{y} \\ \mathbf{z}^T \mathbf{z} = 1 \end{gathered} \tag{23}$$



- Principal Component Analysis (PCA) (resource)
  - Given a data matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$, the goal of PCA is to identify the directions of maximum variance contained in the data (first one is better):
    - Project all the points to the direction s.t. the result data are well seperated
    - Ignore the dimension where the data are similar (low variance), which means that this dimension is less useful



- Sample Variance in a Given Direction
  - Let $\mathbf{v} \in \mathbb{R}^n$ s.t. $||\mathbf{v}||_2 = \sqrt{\mathbf{v}^T\mathbf{v}} = 1$ (unit vector)
  - The sample estimate of the variance in the direction $\mathbf{v}$ is given by the expression:

$$\frac{1}{m}\sum_{i=1}^{m}(\mathbf{v}^T\mathbf{x}^{(i)} - \boldsymbol{\mu})^2, where\ \boldsymbol{\mu} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{v}^T\mathbf{x}^{(i)} \tag{24}$$

  - $\mathbf{v}^T\mathbf{x}^{(i)}$ represents the length of projection when project $\mathbf{x}^{(i)}$ on $\mathbf{v}$
- Pre-centering: Under the assumption that the data are pre-centered so that $\frac{1}{m}\sum_{i=1}^{m}\mathbf{v}^T\mathbf{x}^{(i)} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}^{(i)} = 0$, (向量和为零则每个方向上分量为零) this expression simplifieds to:

$$\frac{1}{m}\sum_{i=1}^{m}(\mathbf{v}^T\mathbf{x}^{(i)})^2 = \frac{1}{m}(\mathbf{X}\mathbf{v})^T(\mathbf{X}\mathbf{v}) = \frac{1}{m}\mathbf{v}^T\mathbf{X}^T\mathbf{X}\mathbf{v} \tag{25}$$

- The Direction of the Maximum Variance
  - Suppose we want to identify the direction $\mathbf{v}_1$ of maximum variance given the data matrix $\mathbf{X}$. We can formulate this optimization problem as follows (this is not a convex optimization => weak duality):

$$\begin{aligned} &max_v \frac{1}{m}\mathbf{v}^T\mathbf{X}^T\mathbf{X}\mathbf{v} \\ &subject\ to\ ||\mathbf{v}||_2^2 = 1 \end{aligned} \tag{26}$$

  - Letting $\boldsymbol{\Sigma} = \frac{1}{m}\mathbf{X}^T\mathbf{X}$ which is a symmectric matrix, we form the Lagrangian

$$L(\mathbf{v}, \lambda) = \mathbf{v}^T\boldsymbol{\Sigma}\mathbf{v} + \lambda(1 - ||\mathbf{v}||_2^2) \tag{27}$$

  - It can be proved that the eigenvalue of $\sum$ is non-negative by using SVD
  - Taking the derivative of $L\ w.r.t\ v$ and setting it to zero, we have:

$$\begin{aligned} \frac{\delta L(\mathbf{v}, \lambda)}{\delta \mathbf{v}} &= 2\boldsymbol{\Sigma}\mathbf{v} - 2\lambda\mathbf{v} = 0 \\ \boldsymbol{\Sigma}\mathbf{v} &= \lambda\mathbf{v} \end{aligned} \tag{28}$$

    - It means the optimal $\mathbf{v}$ appears when $\mathbf{v}$ is an eigenvector of $\boldsymbol{\Sigma}$

- As $\mathbf{v} \neq 0$, $\mathbf{v}$ must be an eigenvector of $\mathbf{\Sigma}$ with eigenvalue λ. Assuming $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ are the eigenvectors of $\mathbf{\Sigma}$, corresponding to eigenvalues $\sigma_1 \geq \ldots \geq \sigma_b \geq 0$, respectively, too choose the best $\mathbf{v}$ while fixing $\mathbf{X}$ we have (i.e. the largest mean value of λ_i where λ_i is the **singular values** of $\mathbf{X}$)

$$\begin{aligned} \mathbf{v}^* &= \mathbf{v}_1, \\ p^* &= \mathbf{v}_1^T \mathbf{\Sigma} \mathbf{v}_1 = \lambda = \sigma_1 \end{aligned} \tag{29}$$

- l largest Directions of Variance
  - Suppose instead of just the direction of maximum variance, we want the l largest directions of variance that are all mutually $orthogonal$
  - Finding the second-largest direction of variance corresponds to solving the problem:

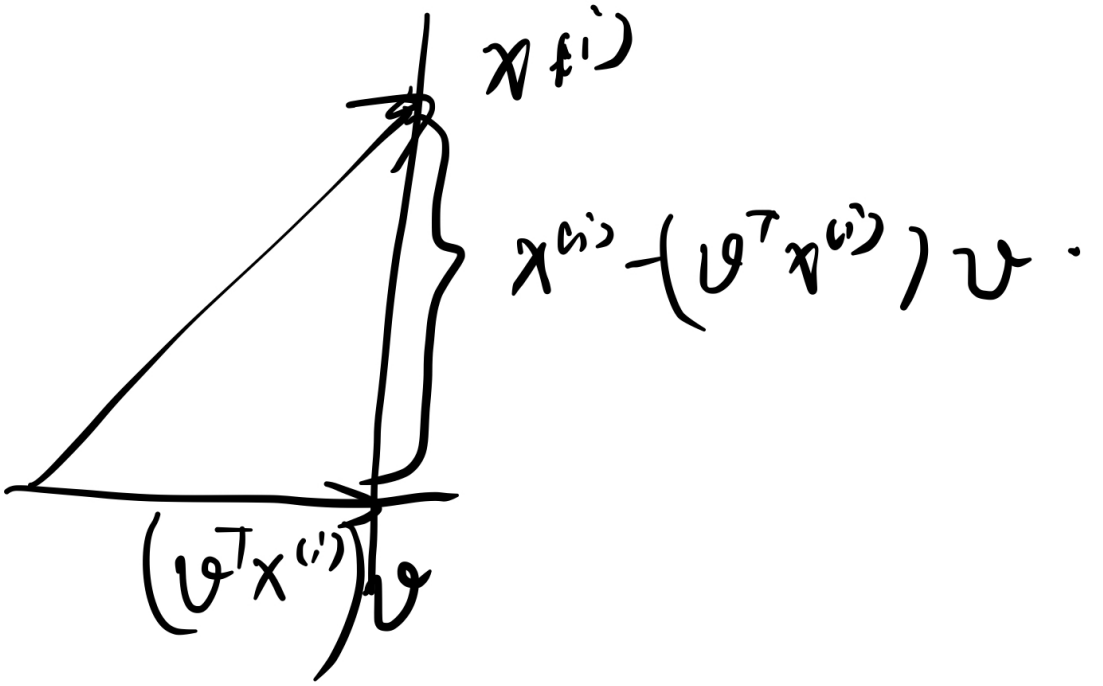
$$\begin{aligned} &max_{\mathbf{v}} \mathbf{v}^T \mathbf{\Sigma} \mathbf{v} \\ &subject\ to\ ||\mathbf{v}||_2^2 = 1 \\ &\mathbf{v}^T \mathbf{v}_1 = 0 \end{aligned} \tag{30}$$

  - We form the Lagrangian

$$\begin{aligned} L(\mathbf{v}, \lambda, v) &= \mathbf{v}^T \mathbf{\Sigma} \mathbf{v} + \lambda(1 - ||\mathbf{v}||_2^2) + v\mathbf{v}^T \mathbf{v}_1 \\ \frac{\delta L}{\delta \mathbf{v}} &= 2\mathbf{\Sigma}\mathbf{v} - 2\lambda\mathbf{v} + v\mathbf{v}_1 = 0 \\ 2\mathbf{v}_1^T \mathbf{\Sigma}\mathbf{v} - 2\lambda\mathbf{v}_1^T \mathbf{v} + v\mathbf{v}_1^T \mathbf{v}_1 &= 0 \\ 2(\mathbf{\Sigma}(\mathbf{\Sigma} = \mathbf{\Sigma}^T)\mathbf{v}_1)^T \mathbf{v} - 2\lambda\mathbf{v}_1^T \mathbf{v} + v\mathbf{v}_1^T \mathbf{v}_1 &= 0 \\ 2\sigma_1 \mathbf{v}_1^T \mathbf{v} - 0 = v &= 0 \end{aligned} \tag{31}$$

  - In general, the top l directions of variance $\mathbf{v}_1, \ldots, \mathbf{v}_l$ are given by the l eigenvectors corresponding to the l largest eigenvalues of $\frac{1}{m}\mathbf{X}^T\mathbf{X}$
- Minimize the Projection error:
  - PCA can also be derived by picking the principal vectors that minimize the approximation error arising from projecting the data onto the l-dimensional subspace spanned by them

$$min_{\mathbf{v}} \frac{1}{m} \sum_{i=1}^{m} ||\mathbf{x}^{(i)} - (\mathbf{v}^T \mathbf{x}^{(i)})\mathbf{v}||_2^2 \tag{32}$$



- Proof: Without loss of generality, we assume $l = 1$. The approximation error, i.e., the mean squared error, between $\{x(i)\}$ and their projections are

$$\frac{1}{m}\sum_{i=1}^{m}||\mathbf{x}^{(i)} - (\mathbf{v}^T\mathbf{x}^{(i)})\mathbf{v}||_2^2 \tag{33}$$
$$= \frac{1}{m}\sum_{i=1}^{m}(||\mathbf{x}^{(i)}||_2^2 + (\mathbf{v}^T\mathbf{x}^{(i)})^2||\mathbf{v}||_2^2 - 2(\mathbf{v}^T\mathbf{x}^{(i)})\mathbf{v}^T\mathbf{x}^{(i)})$$
$$= \frac{1}{m}\sum_{i=1}^{m}(||\mathbf{x}^{(i)}||_2^2 - (\mathbf{v}^T\mathbf{x}^{(i)})^2)$$

- Since $\mathbf{x}_i$ is fixed for each i, minimizing the approximation error is equivalent to minimizing the variance of the projections:

$$min_{\mathbf{v}}\frac{1}{m}\sum_{i=1}^{m}(-(\mathbf{v}^T\mathbf{x}^{(i)})^2) \tag{34}$$
$$= max_{\mathbf{v}}\frac{1}{m}\sum_{i=1}^{m}((\mathbf{v}^T\mathbf{x}^{(i)})^2)$$
$$= max_{\mathbf{v}}\mathbf{v}^T\mathbf{\Sigma}\mathbf{v}$$

- Dimensionality Reductino with PCA
  - Data preprocessing: Compute $\boldsymbol{\mu} = \frac{1}{m}\sum_i \mathbf{x}^{(i)}$ and replace each $\mathbf{x}^{(i)}$ with $\mathbf{x}^{(i)} - \boldsymbol{\mu}$; compute $\sigma_j^2 = \frac{1}{m}\sum_i (x_j^{(i)})^2$ and replace each $\mathbf{x}_j^{(i)}$ with $\frac{\mathbf{x}_j^{(i)}}{\sigma_j}$ (normalize the data)
  - Given pre-processed data matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$, compute the sample covariance matrix $\Sigma = \frac{1}{m}X^TX$
  - Compute the l leading eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_l$ of $\mathbf{\Sigma}$ where $\mathbf{v}_i \in \mathbb{R}^n$
  - Stack the eigenvectors together into an $n \times l$ matrix $\mathbf{V}$ where each column i of $\mathbf{V}$ corresponds

to $\mathbf{v}_i$
    - Project $\mathbf{X}$ into the subspace spanned by the set of eigenvectors $\mathbf{Z} = \mathbf{XV}$
    - To reconstruct $\mathbf{X}$ given $\mathbf{Z}$ and $\mathbf{V}$, we use $\hat{\mathbf{X}} = \mathbf{ZV}^T$

Connection between PCA and SVD

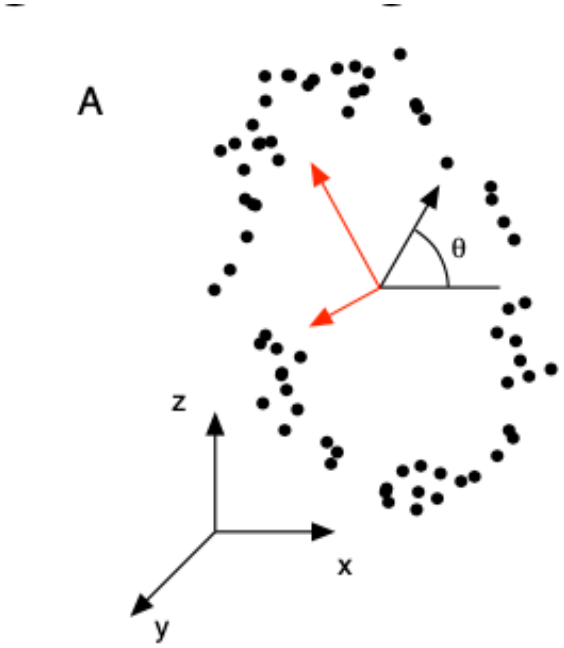
- We have saw that the minimum Frobenius norm linear dimensionality reduction problem could be solved using the rank-l SVD of X:

$$argmin_{\mathbf{U},\mathbf{S},\mathbf{V}} ||\mathbf{X} - \mathbf{USV}^T||_F^2 \tag{35}$$

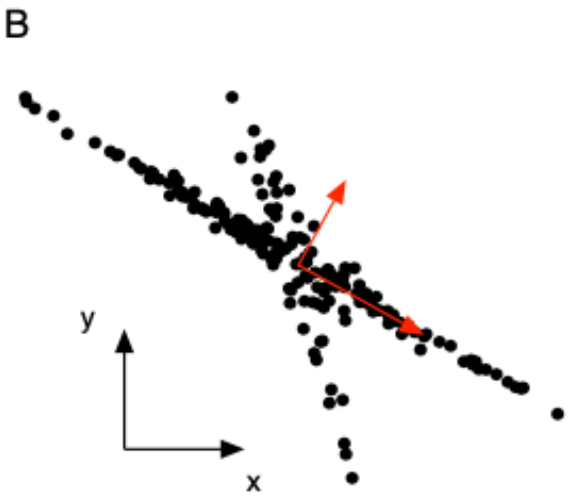
    - Where $\mathbf{U} \in \mathbb{R}^{m \times l}, \mathbf{S} \in \mathbb{R}^{l \times l}, \mathbf{V} \in \mathbb{R}^{n \times l}$. The matrix product $\mathbf{Z} = \mathbf{US}$ gives the optimal rank-l representation of $\mathbf{X}$ with respect to Frobenius norm minimization
    - Let $l = n$ then $\mathbf{X} = \mathbf{USV}^T$ and $\mathbf{X}^T\mathbf{X} = \mathbf{VSU}^T\mathbf{USV}^T = \mathbf{VS}^2\mathbf{V}^T$
    - This means
        - right singular vectors of $\mathbf{X}$ is the same as all the singular vectors of $\mathbf{X}^T\mathbf{X}$
        - eigenvalues of $\mathbf{X}^T\mathbf{X}$ are the diagnal elements of $\mathbf{S}$
        - The l largest singular values for SVD and l largest eigenvalues for PCA correspond to the same l basis vectors
- According to PCA, the projection operation is $\mathbf{Z} = \mathbf{XV}$, therefore

$$\mathbf{Z} = \mathbf{XV} = (\mathbf{USV}^T)\mathbf{V} = \mathbf{US} \tag{36}$$

    - Finally, note that if the decompositions are based only on the l leading principal vectors, which are identical under both PCA and SVD, the projection $\mathbf{Z} = \mathbf{XV}$ and $\mathbf{Z} = \mathbf{US}$ will still be identical
    - These manipulations show that PCA on $X^T X$ and SVD on X identify exactly the same sub space and result in exactly the same projection of the data into that sub-space
    - As a result, generic linear dimensionality reduction simultaneously minimizes the Frobenius norm of the reconstruction error of $\mathbf{X}$ and maximizes the retained variance in the learned sub-space
    - Both SVD and PCA provide the same description of generic linear dimensionality reduction: an orthogonal basis for exactly the same optimal linear subspace
- Issues:
    - The computational complexity of PCA is $O(n^2m + n^3)$ if the full eigendecomposition is obtained and then truncated, compared to $O(min(nm^2, mn^2))$ for SVD
    - The basic SVD and PCA algorithm are not suitable for large-scale data. Instead, randomized algorithms are often used
    - The value of l can sometimes be chosen based on looking for eigenvalue gaps in the eigenspectrum of the covariance matrix
- When does PCA fail
    - The primary motivation behind PCA is to decorrelate the dataset, i.e., remove second-order dependencies. If higher-order dependencies exist between the features in the data, PCA may be insufficient at revealing all structure in the data (for example, the data distributed on a circle)



- PCA requires that each component must be perpendicular to the previous ones, but clearly this requirement is overly stringent and the data might be arranged along non-orthogonal axies



Questions about Lagrange Multiplier

- When using weak duality, why it is ok to optimize L and f at the same time
- Why g is concave for lambda and v
- Why

Lecture 10: Kernel Principal Component Analysis

- Limitation of PCA is it can only find the optimal linear subspace that retain maximum variances

Basis Expansion

- One way to move beyond linear dimensionality reduction is to first apply a non-linear basis expansion to the data vectors, and then apply linear dimensionality reduction
- Any basis expansion can be applied (e.g. polynomials), just as in the case of classification and regression

- Given a data set $\mathbf{X} \in \mathbb{R}^{m \times n}$ and a basis expansion function $\phi : \mathbb{R}^n \mapsto \mathbb{R}^l$ for $l > n$, we obtain the following SVD-based algorithm:
  - Compute $\mathbf{U}, \mathbf{S}, \mathbf{V} = SVD(\phi(\mathbf{X}))$
  - Return $\mathbf{Z} = \mathbf{US}$

Kernel PCA

- Step of Kernel PCA
  - Given a data set $\mathbf{X} \in \mathbb{R}^{m \times n}$ and a basis expansion function $\phi : \mathbb{R}^n \mapsto \mathbb{R}^l$ for $l > n$, we obtain the following PCA-based algorithm (expand to l and then reduce to k):
    - Compute $\mathbf{\Sigma} = \frac{1}{m} \sum_i (\phi(\mathbf{x}^{(i)} - \boldsymbol{\mu})(\phi(\mathbf{x}^{(i)} - \boldsymbol{\mu})^T)$ where $\mu = \frac{1}{m} \sum_i \phi(\mathbf{x}^{(i)})$
    - Compute the k leading eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ of $\mathbf{\Sigma}$ where $\mathbf{v}_j \in \mathbb{R}^{l \times 1}$ for $j = 1, \ldots, k$
    - Stack the eigenvectors together to form $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k]$, where $\mathbf{V} \in \mathbb{R}^{l \times k}$
    - Project the matrix $\phi(\mathbf{X})$ into the rank-k subspace of maximum variance by computing the matrix product $\mathbf{Z} = \phi(\mathbf{X})\mathbf{V}$
- Kernel function
  - As in the classification case, it becomes very expensive to use an explicit basis function expansion that maps data into a high-dimensional space
  - In the basic SVD-based algorithm, there's no way to avoid this problem
  - In the PCA-based algorithm, we are able to take advantage of the kernel trick, which provides a much more efficient computation of feature-based inner products without explicitly computing the basis expansion:

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)}) \tag{37}$$

- Convert the inner product used in kernel PCA
  - Given $\phi : \mathbb{R}^n \mapsto \mathbb{R}^l$, we compute the covariance matrix in the new feature space

$$\mathbf{\Sigma} = \frac{1}{m} \sum_{j=1}^{m} \phi(\mathbf{x}^{(j)}) \phi(\mathbf{x}^{(j)})^T \tag{38}$$

  - Eigendecomposition of $\mathbf{\Sigma}$ is given by(since it is a symmetric matrix):

$$\mathbf{\Sigma}\mathbf{v}_k = \frac{1}{m} \sum_{j=1}^{m} \phi(\mathbf{x}^{(j)}) \phi(\mathbf{x}^{(j)})^T \mathbf{v}_k = \lambda_k \mathbf{v}_k, \forall k = 1, \ldots, l \tag{39}$$

  - It is not hard to see that $\mathbf{v}_k$ can be expressed as

$$\mathbf{v}_k = \frac{1}{m\lambda_k}\sum_{j=1}^{m}\phi(\mathbf{x}^{(j)})\phi(\mathbf{x}^{(j)})^T\mathbf{v}_k \tag{40}$$

$$= \sum_{j=1}^{m}\frac{1}{m\lambda_k}[\phi(\mathbf{x}^{(j)})^T\mathbf{v}_k]\phi(\mathbf{x}^{(j)})(scalar)$$

$$= \sum_{j=1}^{m} w_k^{(j)}\phi(\mathbf{x}^{(j)})\ where\ w_k^{(j)} = \frac{1}{m\lambda_k}[\phi(\mathbf{x}^{(j)})^T\mathbf{v}_k]$$

$$\phi(\mathbf{x}^{(i)})^T\mathbf{v}_k = \sum_{j=1}^{m} w_k^{(j)}\phi(\mathbf{x}^{(i)})^T\phi(\mathbf{x}^{(j)}) = m\lambda_k w_k^{(j)}$$

$$\sum_{j=1}^{m} w_k^{(j)}K_{ij} = m\lambda_k w_k^{(j)}$$

$$\mathbf{K}\mathbf{w}_k = m\lambda_k\mathbf{w}_k$$

- We can find that $\mathbf{w}_k$ is the kth eigenvector of $\mathbf{K}$ with the corresponding eigenvalue $m\lambda_k$
- As the eigenvalues of $\mathbf{K}$ are proportional to the eigenvalues $\lambda's$ of the covariance matrix $\mathbf{\Sigma}$ in the feature space, PCA on $\mathbf{\Sigma} \in \mathbb{R}^{l\times l}$ is equivalent to PCA on $\mathbf{K} \in \mathbb{R}^{m\times m}$
- For any new data point $\boldsymbol{x}$ with its high-dimensional feature representation $\phi(\mathbf{x})$, the kth PCA component can be obtained by projecting it on the k-th eigenvector $\mathbf{v}_k$ of $\mathbf{\Sigma}$

$$\phi(\mathbf{x})^T\mathbf{v}_k = \sum_{i=1}^{m} w_k^{(i)}\phi(\mathbf{x})^T\phi(\mathbf{x}^{(i)}) = \sum_{j=1}^{m} w_k^{(i)}K(\mathbf{x},\mathbf{x}_i) \tag{41}$$

- Complete steps of PCA: Given a data set $\mathbf{X} \in \mathbb{R}^{m\times n}$ and a kernel function K, kernel PCA can be computed as follows:
  - Compute $K_{ij} = \mathbf{x}_i, \mathbf{x}_j$ for all i, j
  - Compute $\mathbf{K}' = (\mathbf{I} - \mathbf{1}_m)\mathbf{K}(\mathbf{I} - \mathbf{1}_m)$ where $\mathbf{1}_m$ is an $m \times m$ matrix where every entry is $\frac{1}{m}$ (to zero centro the data)

$$\phi'(\mathbf{x}^{(i)}) = \phi(\mathbf{x}^{(i)}) - \frac{1}{m}\sum_{k=1}^{m}\phi(\mathbf{x}^{(k)}) \tag{42}$$

$$\mathbf{K}'_{ij} = \phi'(\mathbf{x}^{(i)})^T\phi'(\mathbf{x}^{(j)})$$

$$= [\phi(\mathbf{x}^{(i)}) - \frac{1}{m}\sum_{k=1}^{m}\phi(\mathbf{x}^{(k)})]^T[\phi(\mathbf{x}^{(j)}) - \frac{1}{m}\sum_{k=1}^{m}\phi(\mathbf{x}^{(k)})]$$

$$= \mathbf{K}_{ij} - \frac{1}{m}\sum_{k=1}^{m}\mathbf{K}_{ik} - \frac{1}{m}\sum_{k=1}^{m}\mathbf{K}_{kj} + \frac{1}{m^2}\sum_{k=1}^{m}\sum_{k'=1}^{m}\mathbf{K}_{kk'}$$

$$= \mathbf{K}_{ij} - \mathbf{K}_{i,*}\mathbf{1}_m - \mathbf{1}_m^T\mathbf{K}_{*,j} + \mathbf{1}_m^T\mathbf{K}\mathbf{1}_m$$

$$\mathbf{K} = \mathbf{K} - \mathbf{K}\mathbf{1}_{m\times m} - \mathbf{1}_{m\times m}\mathbf{K} + \mathbf{1}_{m\times m}^T\mathbf{K}\mathbf{1}_{m\times m}$$

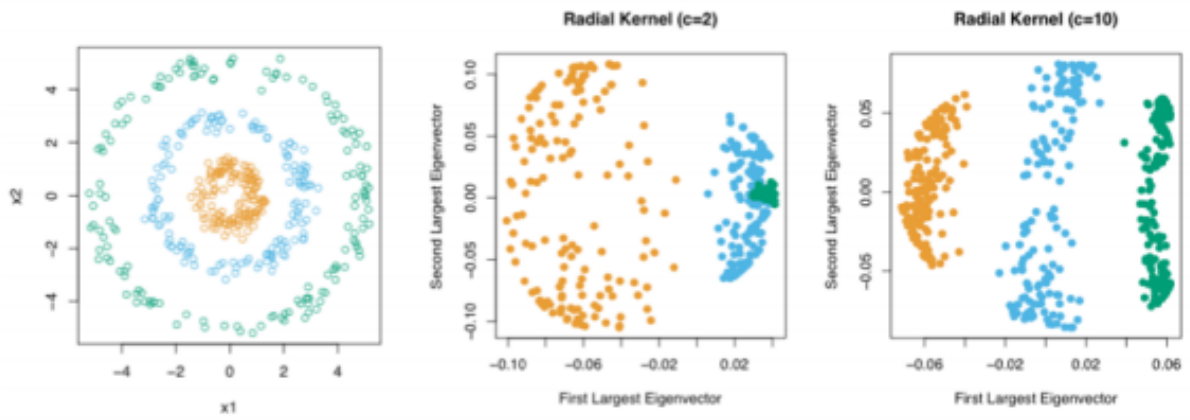
$$= (\mathbf{I} - \mathbf{1}_m)\mathbf{K}(\mathbf{I} - \mathbf{1}_m)$$

  - Compute the r leading eigenvectors $\mathbf{w}_1, \ldots, \mathbf{w}_r$ of $\mathbf{K}'$ along with their eigenvalue $m\lambda_1, \ldots, m\lambda_r$

- Compute the kth element of the projected data vector $\mathbf{z} \in \mathbb{R}^{r \times 1}$ using $z_k = \sum_{j=1}^{m} w_k^{(i)} K(\mathbf{x}, \mathbf{x}_i)$

- Example:

Consider the case of using the radial basis function kernel
$\mathcal{K}(\mathbf{x}, \mathbf{y}) = \exp(-\frac{1}{c}||\mathbf{x} - \mathbf{y}||_2^2)$



- Summary
  - Kernel PCA provides a non-linear dimensionality reduction method that can be used to extract directions of maximum variance without explicitly performing basis expansion
  - Kernel PCA can be thought of as looking for directions of variance in the space of similarities between data cases, and can extract components that correspond to fairly complex structures
  - When a good kernel is known a priori, kernel PCA can provide an effective pre-processing step for clustering methods as well as linear classification and regression methods
  - 0However, exact computation of kernel PCA can be expensive because the size of the matrix to be decomposed is $m \times m$

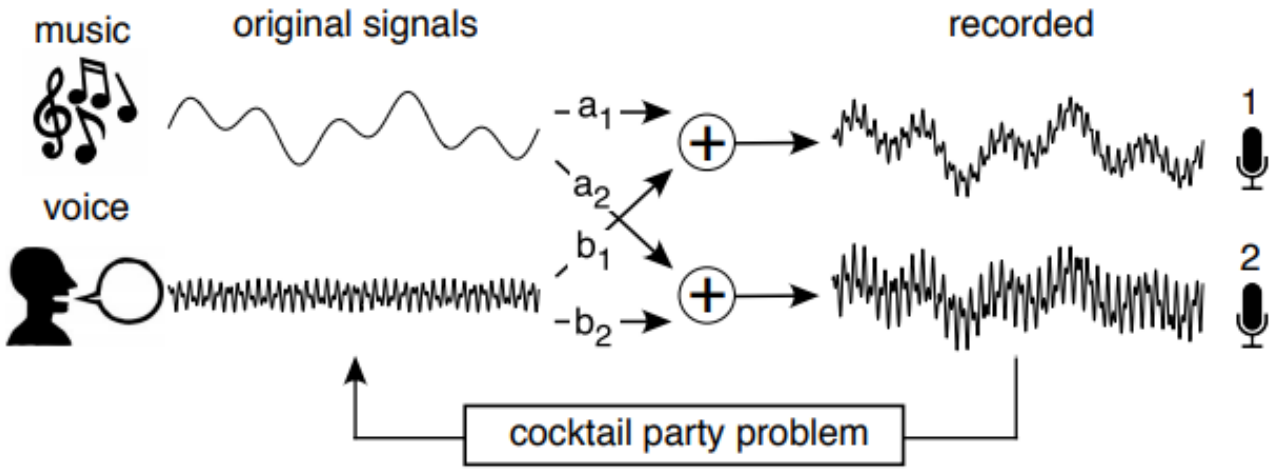
Lecture 11: Independent Component Analysis

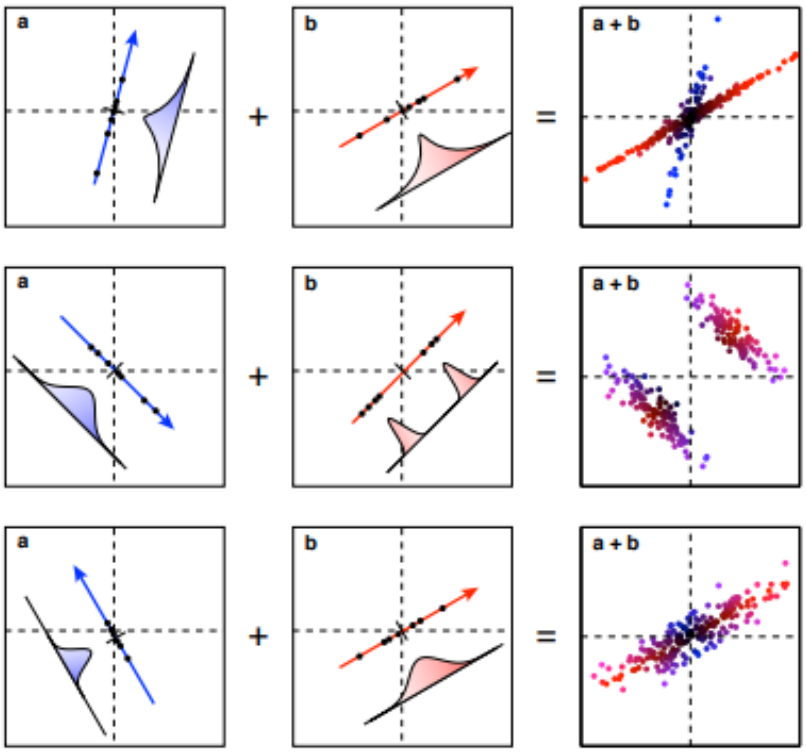
Independent Component Analysis Motivation

- Measurements in the real world are corrupted by random noise - but that is only one piece of the story. Often, measurements cannot be made in isolation, but reflect the combination of many distinct sources
- Linear mixing of independent sources is exactly what occurs then you listen to multiple audio sources at the same time
- Humans are able to de-mix multiple sources of audio (multiple people speaking) into distinct sources very accurately
- Independent component analysis (ICA) is designed to solve exactly this problem (called blind source separation) and can do so very reliably when the number of observed linearly mixed channels is equal to the number of sources
- Demo

ICA Illustration

- An example





- Where the x-axis and the y-axis represents the signal of music recorded for microphone 1 and 2 respectively (the loud of the sound times the corresponding weight for two microphones)
- Since the voice has positive correlation to the distance between the source and the microphone, the proportion of voices recorded by different microphone is a constant because of the fixed distance => if each data point represents the record of a specific time, the points for the same source are distributed on a line throw the origin with the slope equals to the proportion of distances between source and two different microphones
- We assume that the points for each microphone has a normal distribution on time (the signal is time independent)
- In this case, it is useless to find a maximum variance direction

ICA Formulation

- To formalize this problem, assume the data $\mathbf{s} \in \mathbb{R}^{n \times 1}$ is generated via n independent sources (at a specific time). What we observe is

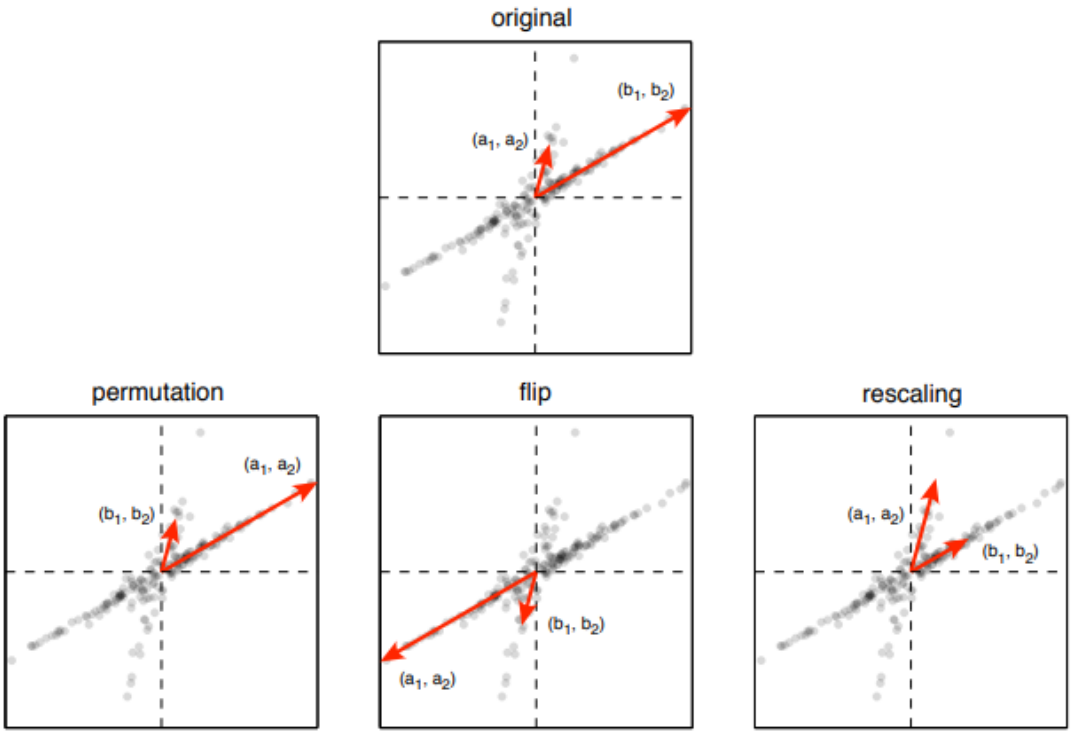
$$\mathbf{x} = \mathbf{A}\mathbf{s} \tag{1}$$

  - where $\mathbf{A} \in \mathbb{R}^{n \times n}$ is an unknown *mixing* matrix
  - **Each row of $\mathbf{A}$ represents a microphone**
- Repeated observations (get the sample different time) gives us a dataset $D : \{\mathbf{x}^{(i)}, i = 1, \dots, m\}$, and our goal is to find $\mathbf{A}$, so that given the observation $\mathbf{x}^{(i)}$, we can recover the sources $\mathbf{s}^{(i)}$ by computing $\mathbf{s}^{(i)} = \mathbf{W}\mathbf{x}^{(i)}$, where $\mathbf{W} = \mathbf{A}^{-1}$ is the *unmixing* matrix
- In our cocktail party problem, $s_j^{(i)}$ is the sound that speaker j was uttering at time i. Similarly, $\mathbf{x}_j^{(i)}$ is acoustic reading recorded by microphone j at time i
- Notice that here the i is the index of **each column**

ICA Ambiguities (歧義, 等价)

- To what degree can $\mathbf{W}^{-1} = \mathbf{A}$ be recovered
- Permutation ambiguity: The labels of each independent component can be arbitrarily permuted. That is, given only the $\mathbf{x}^{(i)}$'s, there will be no way to distinguish between $\mathbf{A}$ and $\mathbf{AP}$, where $\mathbf{P} \in \mathbb{R}^{n \times n}$ is a permutation matrix (the order of independent components can be changed)
- Sign ambiguity: Any independent component can be flipped across the origin. That is, $\mathbf{A}$and $-\mathbf{A}$ are treated identical in practical applications (the direction of independent components can be changed)
- Scale ambiguity: Any independent component can be rescaled with arbitrary length. For example, given $\alpha \neq 0$, we have (the length of independent components can be changed, and absorbed to the matrix $\mathbf{A}$)

$$\mathbf{x}^{(i)} = \mathbf{A}\mathbf{s}^{(i)} = \alpha\mathbf{A}(\frac{s^{(i)}}{\alpha}) = \hat{\mathbf{A}}\hat{\mathbf{s}}^{(i)} \tag{2}$$



A Strategy for Solving ICA

- Divide-and-conquer provides a strategy to solve this problem. Rather than trying to solve for $\mathbf{s}$ and $\mathbf{A}$ simultaneously, we focus on finding $\mathbf{A}$. Furthermore, rather than trying to solve for $\mathbf{A}$ all at once, we solve for $\mathbf{A}$ in a piece-meal fashion by cutting up $\mathbf{A}$ into simpler and more manageable parts
- SVD of $\mathbf{A}$ is given by $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ (unknown)
- We estimate $\mathbf{A}$ and its inverse $\mathbf{W}$ by recovering each piece of the decomposition individually

$$\mathbf{W} = \mathbf{A}^{-1} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T \tag{3}$$

- To compute $\mathbf{U}$ and $\mathbf{\Sigma}$, we first estimate the covariance of the observed mixed data in terms of the underlying sources (PCA)

$$
\begin{aligned}
\mathbf{C} &= \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}^{(i)}(\mathbf{x}^{(i)})^T = \frac{1}{m}\sum_{i=1}^{m}\mathbf{A}\mathbf{s}^{(i)}(\mathbf{A}\mathbf{s}^{(i)})^T \\
&= \frac{1}{m}\sum_{i=1}^{m}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{s}^{(i)}(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{s}^{(i)})^T \\
&= \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\left(\frac{1}{m}\sum_{i=1}^{m}\mathbf{s}^{(i)}(\mathbf{s}^{(i)})^T\right)\mathbf{V}\mathbf{\Sigma}\mathbf{U}^T \\
&= \mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^T,
\end{aligned}
$$

$$
\text{assuming } \tfrac{1}{m}\textstyle\sum_i \mathbf{x}^{(i)} = 0 \text{ and } \tfrac{1}{m}\textstyle\sum_i \mathbf{s}^{(i)}(\mathbf{s}^{(i)})^T = \mathbf{I}
$$

- The first assumption get the equation and the second one is from the three ambiguities

$$
\frac{1}{m}\sum_i \mathbf{s}^{(i)}\mathbf{s}^{(i)T} = E(\mathbf{s}^{(i)}\mathbf{s}^{(i)T}) = \tag{4}
$$

$$
\begin{bmatrix}
E(\mathbf{s}_1\mathbf{s}_1) & E(\mathbf{s}_1\mathbf{s}_2) & \dots & E(\mathbf{s}_1\mathbf{s}_n) \\
\dots & \dots & \dots & \dots \\
\dots & \dots & \dots & E(\mathbf{s}_n\mathbf{s}_n)
\end{bmatrix} = \mathbf{I} = \frac{1}{m}\mathbf{s}\mathbf{s}^T
$$

  - Each row is orthogonal to other rows
  - Because we assume $\mathbf{s}$ is normalized and different features(**speakers**) of $\mathbf{s}$ are independent
    - $E(s_i s_j) = 0$ when $i \neq j$

$$
\begin{aligned}
E(S_iS_j) &= \sum_{s_i\in S_i, s_j\in S_j} p(S_i = s_i, S_j = s_j)s_i s_j \qquad (5)\\
&= \sum_{s_i\in S_i, s_j\in S_j} p(S_i = s_i)p(S_j = s_j)s_i s_j \\
&= \sum_{s_i\in S_i} p(S_i = s_i)s_i \sum_{s_j\in S_j} p(S_j = s_j)s_j \\
&= E(S_i)E(S_j) = 0 * 0 = 0
\end{aligned}
$$

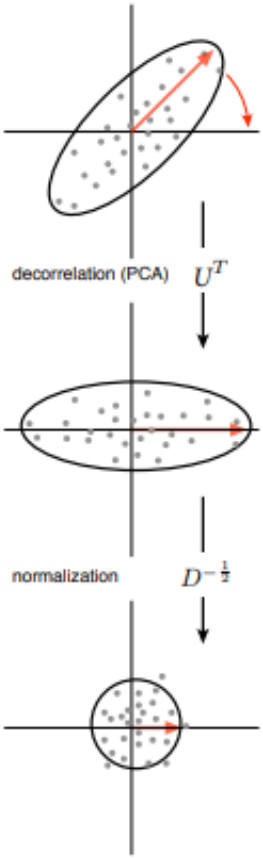
    - $E(s_i s_j) = 1$ when $i = j$

$$
\begin{aligned}
Var(S_i) &= \sum_{s_i \in S_i} (s_i - \mu)^2 p(s_i) \\
&= \sum_{s_i \in S_i} (s_i^2 + \mu^2 - 2 s_i \mu) p(s_i) \\
&= \sum_{s_i \in S_i} s_i^2 p(s_i) + \sum_{s_i \in S_i} \mu^2 p(s_i) - 2 \sum_{s_i \in S_i} s_i \mu \, p(s_i) \\
&= \sum_{s_i \in S_i} s_i^2 p(s_i) + \mu^2 \sum_{s_i \in S_i} p(s_i) - 2\mu \sum_{s_i \in S_i} s_i \, p(s_i) \\
&= \sum_{s_i \in S_i} s_i^2 p(s_i) + \mu^2 - 2\mu^2 \\
&= E(S_i^2) - E(S_i)^2 \\
Var(S_i) &= 1, E(S_i) = 0 \\
E(S_i^2) &= 1
\end{aligned} \tag{6}
$$

- Transform the problem
  - By out shrewd(精明) choice of assumption, the covariance of the data $\mathbf{C} = \mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^T$ is independent of sources $\mathbf{s}^{(i)}$'s as well as $\mathbf{V}$, which can be solved using eigendecomposition
  - Therefore, if our assumptions behind ICA are correct, then we have identified a partial solution to $\mathbf{A}$: $\mathbf{U}$ is a matrix of the **stacked eigenvectors of the covariance of the data ($\mathbf{x}$)** and $\mathbf{\Sigma}$ is a diagonal matrix with the square root of the associated eigenvalue in the diagonal
  - Interestingly, the eigenvectors of the covariance are essentially the principal components of the data in PCA
- Defining $\hat{\mathbf{x}}^{(i)} = (\mathbf{\Sigma}^{-1}\mathbf{U}^T)\mathbf{x}^{(i)} = \mathbf{V}^T\mathbf{s}^{(i)}$, we have:

$$
\hat{\mathbf{C}} = \frac{1}{m}\sum_{i=1}^{m} \hat{\mathbf{x}}^{(i)}(\hat{\mathbf{x}}^{(i)})^T = \frac{1}{m}\sum_{i=1}^{m} \mathbf{V}^T\mathbf{s}^{(i)}(\mathbf{V}^T\mathbf{s}^{(i)})^T = \mathbf{I} \tag{7}
$$

- where $\mathbf{x}^{(i)}$ is first projected on the principal components, $\mathbf{U}^T\mathbf{x}^{(i)}$, to remove linear correlations, and then scaled so that every direction has unit variance $\mathbf{\Sigma}^{-1}\mathbf{U}^T\mathbf{x}^{(i)}$. In signal processing, this is termed as *whitening*



- Plugging $\hat{\mathbf{x}}^{(i)}$ into $s^{(i)} = \mathbf{W}\mathbf{x}^{(i)}$, we simplify ICA down to solving $\mathbf{s}^{(i)} = \mathbf{V}\hat{\mathbf{x}}^{(i)}$, i.e. finding a rotation matrix $\mathbf{V}$
- We now need to **exploit the statistics of independence** to identify V. Whitening the data removed all second-order correlations, hence discerning the last rotation matrix V requires examining other measures of dependency
- Statistical independence is the strongest measure of dependency between random variables. In particular, if two random variables a and b are independent, then $p(a, b) = p(a)p(b)$ - the joint probability factorizes. In the context of ICA, we assume that all sources are statistically independent, thus $p(\mathbf{s}) = \Pi_j p(\mathbf{s}_j)$
- Mutual Information
  - We resort to *multi-information*, a generalization of mutual information from information theory, to **judge how close a distribution is to statistical independence for multiple variables**

$$I(\mathbf{s}) = \sum_{s \in S} p(\mathbf{s}) log \frac{p(\mathbf{s})}{\Pi_j p(s_j)} \tag{8}$$

    - It is a non-negative quantity that reaches a minimum of zero if and only if all variables are statistically independent. For example, if $p(\mathbf{s}) = \Pi_j p(\mathbf{s}_j)$, $I(\mathbf{s}) = 0$
  - Minimizing the multi-information is difficult in practice but can be simplified
  - The multi-information is a function of the entropy, which is defined as $H(\mathbf{s}) = -\sum_{s \in S} p(\mathbf{s}) log p(\mathbf{s})$, measuring the amount of uncertainty about a distribution $p(\mathbf{s})$
  - The multi-information is the difference between the sum of entropies of the marginal distributions and the entropy of the joint distribution
    - $|\mathbf{V}| = 1$, since $\mathbf{V}$ is orthogonal

$$I(\mathbf{s}) = \sum_j H(s_j) - H(\mathbf{s}) = \sum_j H((\mathbf{V}\hat{\mathbf{x}})_j) - H(\mathbf{V}\hat{\mathbf{x}}) \tag{9}$$
$$= \sum_j H((\mathbf{V}\hat{\mathbf{x}})_j) - (H(\hat{\mathbf{x}}) + log_2|\mathbf{V}|) = \sum_j H((\mathbf{V}\hat{\mathbf{x}})_j) - H(\hat{\mathbf{x}})$$

9.3 Entropy under Linear Transformations (Optional)

For a linear transformation $A \in \mathbb{R}^{n\times n}$ and a random variable $x \in \mathbb{R}^{n\times 1}$, we have

$$H[Ax] = H[x] + \log_2|A| \tag{173}$$

where $|A|$ is the absolute value of the determinant[25] of A, and is equal to the absolute value of the product of its eigenvalues.

Proof. Let $z = Ax$. The input and output probabilities are related by

$$p(z)dz = p(x)dx, \tag{174}$$

on an infinitesimal area and

$$p(z) = \frac{p(x)}{\left|\frac{dz}{dx}\right|} = \frac{p(x)}{|A|}. \tag{175}$$

The entropy of z can be computed by

$$\begin{aligned} H[z] =& -\int p(z)\log_2 p(z)dz \\ =& -\int \frac{p(x)}{|A|}\log_2\frac{p(x)}{|A|}|A|dx \\ =& -\int p(x)\log_2\frac{p(x)}{|A|}dx \\ =& -\int p(x)\log_2 p(x)dx + \int p(x)\log_2|A|dx \\ =& H[x] + \log_2|A|. \end{aligned} \tag{176}$$

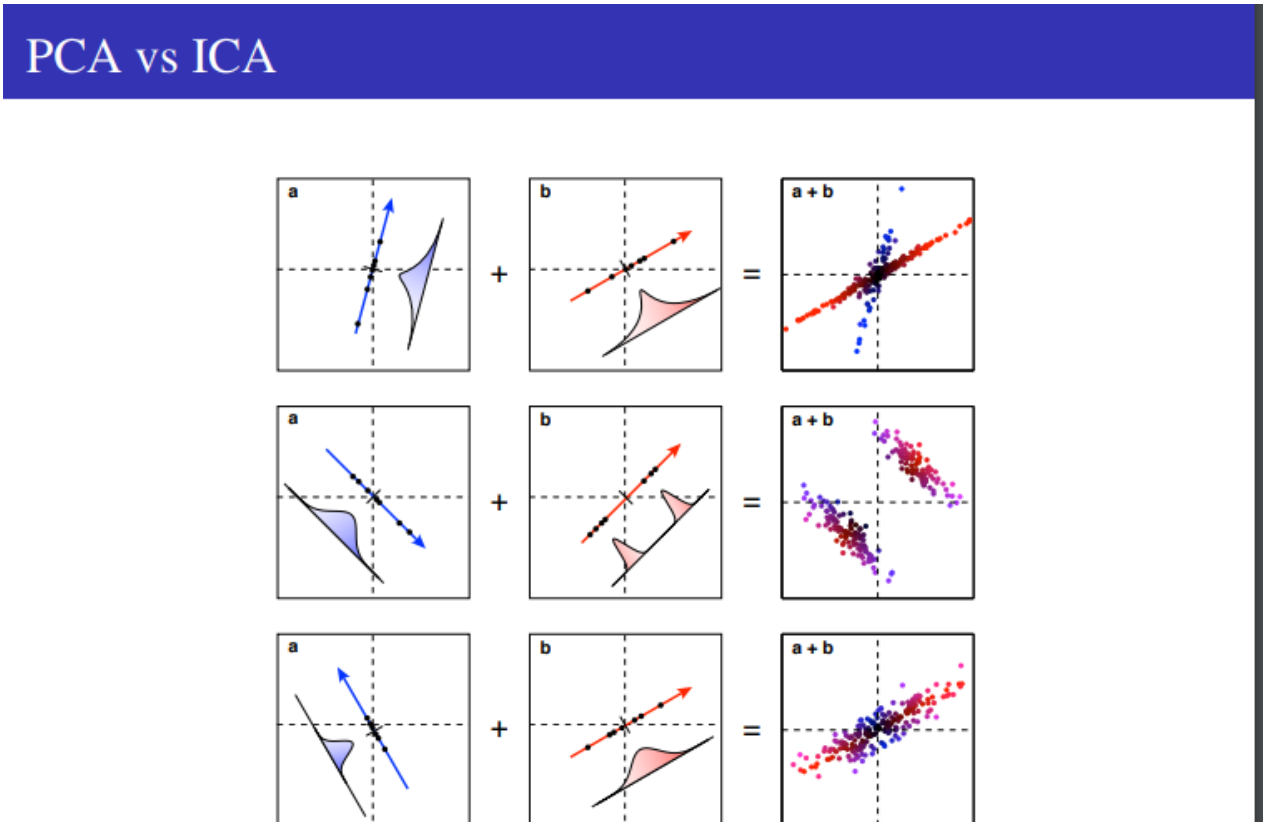
□

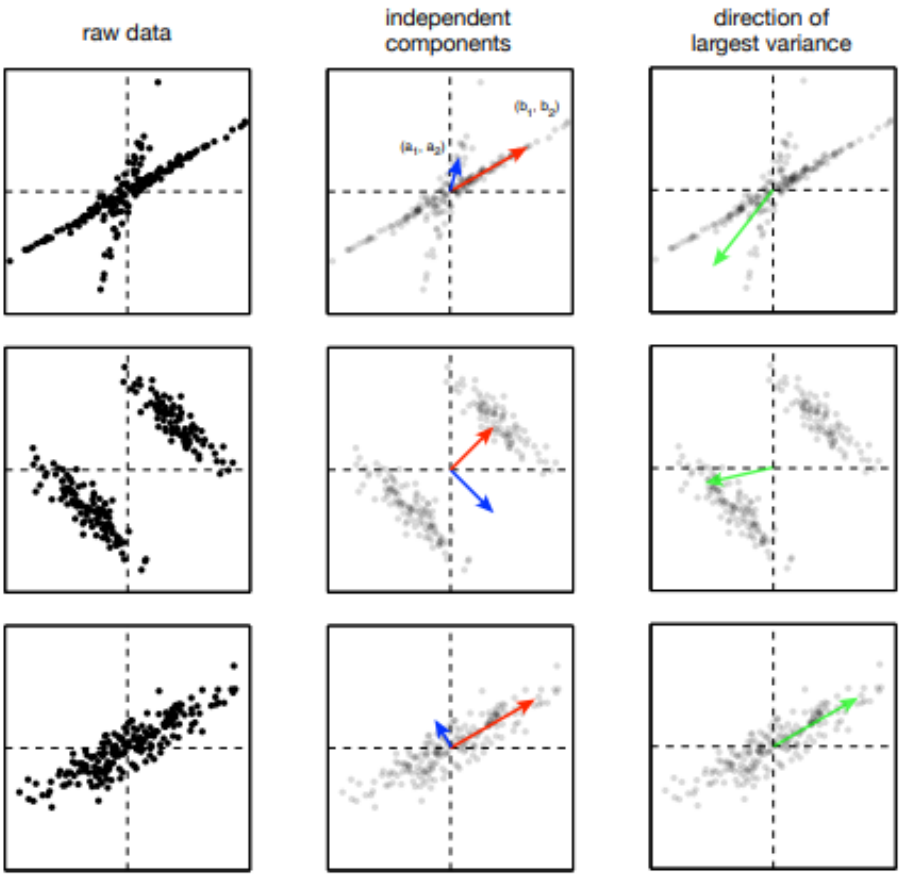
- Optimization
  - The optimization is simplified further by recognizing that the term $H(\hat{\mathbf{x}})$ is a constant, independent of V, and can be dropped:

$$\mathbf{V}^\star = \arg\min_{\mathbf{V}} \sum_j H((\mathbf{V}\hat{\mathbf{x}})_j)$$

  - Calculating the entropy from a finite data set is difficult and should be approached with abundant caution. **Many ICA variants focus on approximations to the above equation**
  - It is equivalent to finding the rotation that maximizes the expected log-likelihood of the observed data under the assumption that the data are statistically independent

    - Likewise, a solution to the equation finds the rotation that maximizes the “non-Gaussianity” of the transformed data
- The ICA Algorithm
    - Subtract off the mean of the data in each dimension
    - Whiten the data by calculating the **eigenvectors of the covariance of the data**
    - Identify final rotation matrix that optimizes statistical independence





- Discussion
  - By assumption ICA expects the data to arise from an **(invertible) linear transformation** applied to a **factorial distribution.** Hence, ICA expects and will work optimally on data arising from linearly transformed factorial distributions - applying ICA to any other type of data is a gamble with no guarantees
  - ICA is equivalent to PCA for Gaussian data because a whitening filter based on PCA already achieves a factorial distribution and no need exists for computing V
  - If higher-order correlations are small or insignificant, then PCA is sufficient to recover the independent sources of the data
  - Empirically measuring $I(s)$ provides a useful check on how independent the recovered sources are. In practice, $I(s)$ rarely achieves 0 either because of sampling issues or because of local minima in the optimization

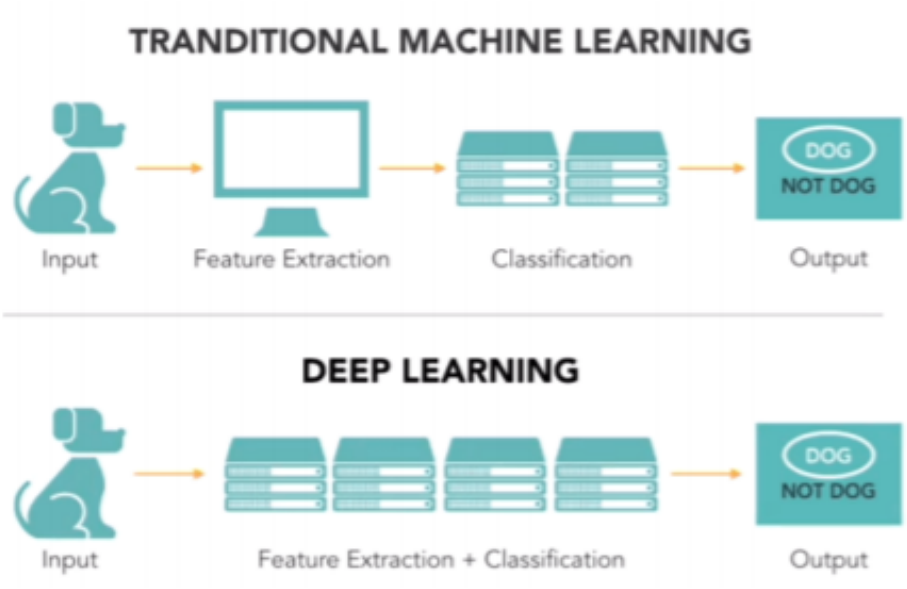
Questions

- What is the second order correlation
- Why PCA and ICA are equivalents for Gaussian data

Lecture 12: Neural Networks and Deep Learning

Overview

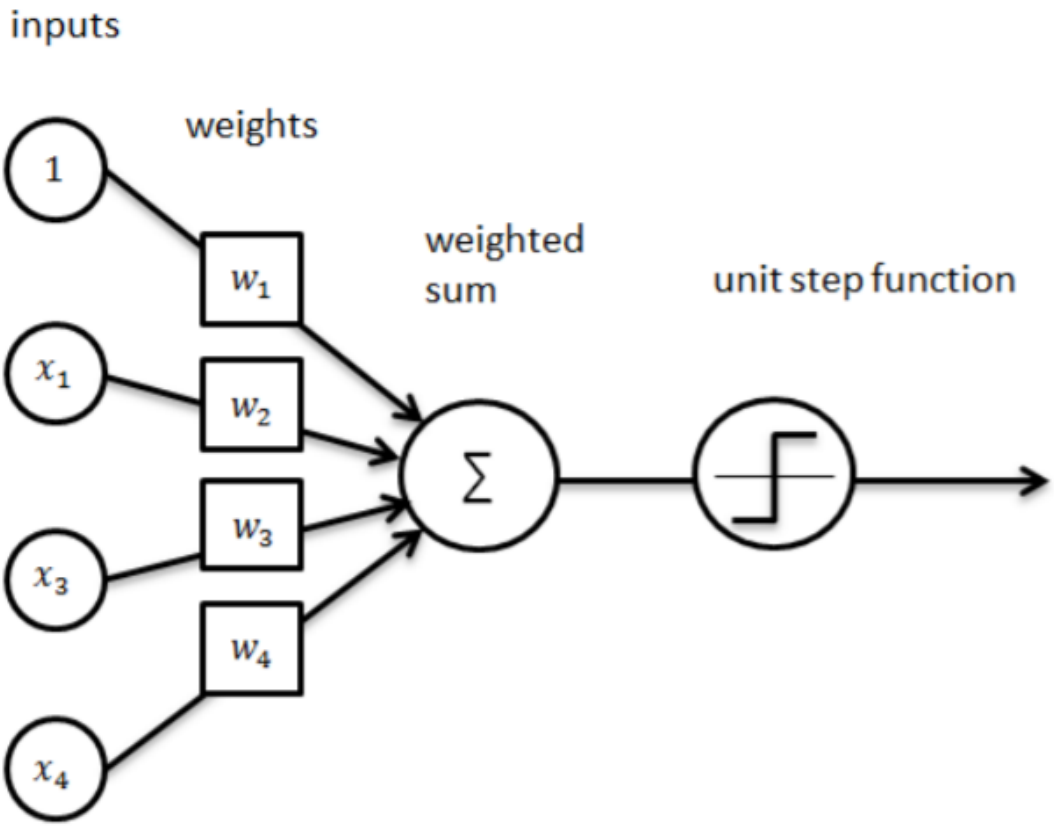
- Feature Learning



- Grand Challenge: Why/how deep learning works
  - Gap between engineering (or art) and science: Lack of theoretical understandings guarantees, and analytical tools
  - Training is computationally expensive and difficult, relying on many “magics”
    - Structure design
    - Network initialization
    - Hyper-parameter tuning
  - No principled way to incorporate(包括) domain expertise(专长)
  - No principled way to interpret the model behaviors

Neural Networks

- Mcculloch and Pitts Neuron (1943)



- The Perceptron (感受器) (for **binary** classification problems)
  - The Perceptron is a simple online algorithm for adapting the weights in a McCulloch/Pitts neuron. It was developed in the1950s by Rosenblatt at Cornell
    - Start with all-zero weight vector $\mathbf{w}^{(0)}$, and initialize t to 0
    - For all $(\mathbf{x}^{(i)}, y^{(i)}) \in D$, compute the activation $\mathbf{a}^{(i)} = (x^{(i)})^T \mathbf{w}^{(t)}$
    - If $\mathbf{y}^{(i)} \mathbf{a}^{(i)} < 0$, then $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + y^{(i)} \mathbf{x}^{(i)}$ and $t \leftarrow t + 1$

$$\begin{aligned} \mathbf{a}_{t+1}^{(i)} &= (x^{(i)})^T w^{(t+1)} \\ &= (x^{(i)})^T (w^{(t)} + y^{(i)} \mathbf{x}^{(i)}) \\ &= a_t^{(i)} + y^{(i)} (x^{(i)})^T x^{(i)} \end{aligned} \tag{1}$$

$$\hat{y} = \begin{cases} 1, & \text{if } w^T x \geq 0 \\ -1 & \text{if } w^T x < 0, \end{cases}$$

  - If $y^{(i)} = 1$ and $\hat{y}^{(i)} = -1$, the activation ($\hat{y}^{(i)}$) is initially negative and will be increased (see the equation)
  - If $y^{(i)} = -1$ and $\hat{y}^{(i)} = 1$, the activation is initially positive and will be decreased
- The Perceptron Theorem
  - Let the training set be $\mathbf{D} = \{(\mathbf{x}^{(i)}, y^{(i)}), i = 1, \ldots, m\}$. Assume that $||\mathbf{x}^{(i)}||_2 \leq \alpha$ for all i and further that there exists a unit-length vector $\mathbf{u}$ ($||\mathbf{u}||_2 = 1$) such that $y^{(i)} (\mathbf{u}^T \mathbf{x}^{(i)}) \geq \beta$ (weighted average of dimensions $\mathbf{x}^{(i)}$ times $y^{(i)}$) for all examples in D. Then the total number of mistakes that the perceptron algorithm makes on this sequence is at most $(\alpha/\beta)^2$

- Proof:
  - The perceptron updates its weights only for following case:

$$(x^{(i)})^T w^{(t)} y^{(i)} \leq 0 \tag{2}$$

- According the rule of perceptron learning

$$u^T w^{(t+1)} = u^T (w^{(t)} + y^{(i)} x^{(i)}) \geq u^T w^{(t)} + \beta \tag{3}$$

  - The weight matrix is initialized as 0 :

$$u^T w^{(t+1)} \geq u^T w^{(1)} + t\beta \geq t\beta \tag{4}$$

- Therefore:

$$\begin{aligned} ||w^{(t+1)}||_2^2 &= ||w^{(t)} + y^{(i)} x^{(i)}||_2^2 \\ &= ||w^{(t)}||_2^2 + ||y^{(i)} x^{(i)}||_2^2 + 2y^{(i)} (x^{(i)})^T w^{(t)} \\ &= ||w^{(t)}||_2^2 + ||x^{(i)}||_2^2 + 2y^{(i)} (x^{(i)})^T w^{(t)} \\ &\leq ||w^{(t)}||_2^2 + ||x^{(i)}||_2^2 \\ &\leq ||w^{(t)}||_2^2 + \alpha^2 \leq ||w^{(1)}||_2^2 + t\alpha^2 = t\alpha^2 \end{aligned} \tag{5}$$

- Therefore:

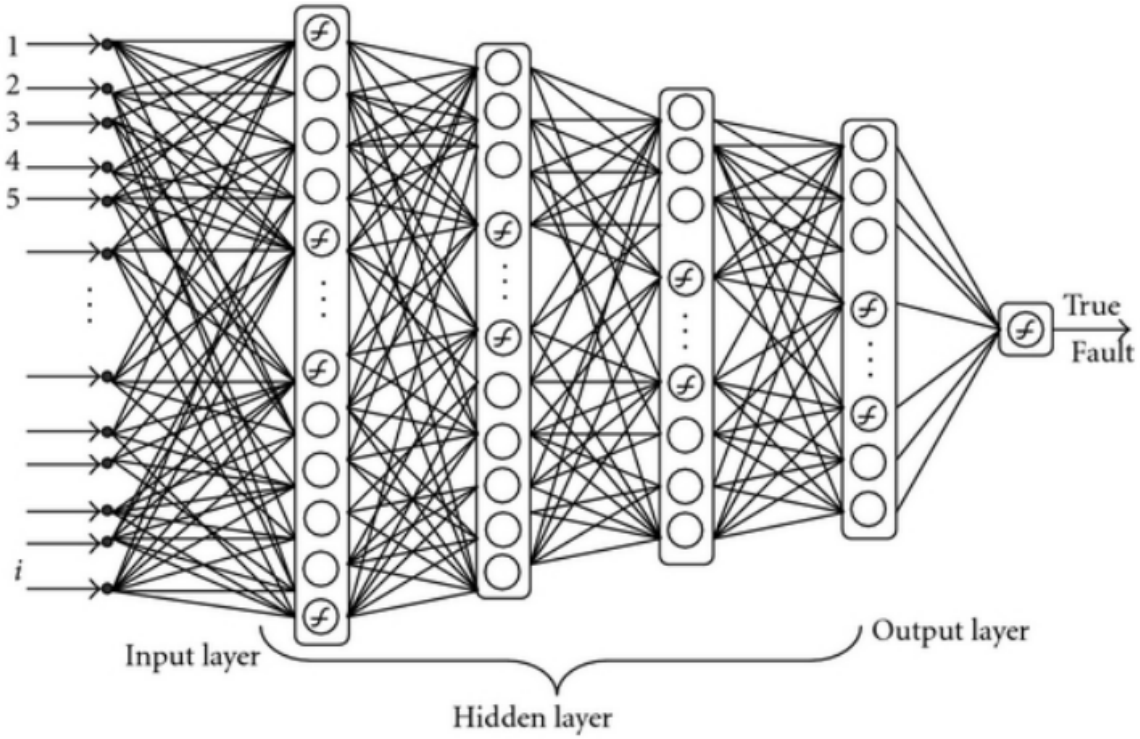
$$\begin{aligned} \sqrt{t}\alpha &= ||w^{(t+1)}||_2 \\ &= ||w^{(t+1)}||_2 ||u||_2 \geq ||w^{(t+1)}||_2 ||u||_2 cos\ \theta \\ &= u^T w^{(t+1)} \geq t\beta \end{aligned} \tag{6}$$

- Which implies that:

$$t \leq (\frac{\alpha}{\beta})^2 \tag{7}$$

- Hence, if the perceptron made a t-th mistake, t must be less than or equal to $(\frac{\alpha}{\beta})^2$. That is, the perceptron always converges as long as the data set D is linearly separable (i.e. there exists a unit-length vector u s.t. $y^{(i)} (\mathbf{u}^T \mathbf{x}^{(i)}) \geq \beta$ for $\forall\ x^{(i)} \in D$)
- Limitations of Single Layer Perceptrons
  - The representational power of the single-layer perceptron was not well understood at first in the AI community
  - In 1969, Minsky and Papert at MIT popularized a set of arguments showing that the single-layer perceptron could not learn certain classes of functions (including XOR)
  - They also showed that more complex functions could be represented using a multi-layer perceptron or MLP, but no one knew how to learn them from examples
  - This led to a shift away from mathematical/statistical models in AI toward logical/symbolic models

- Multi-Layer Perceptron



- The solution to MLP learning turned out to be:
  - Make the hidden layer non-linearities smooth (sigmoid/logistic) functions:

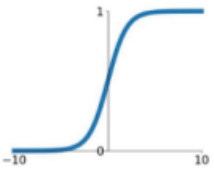
$$h_j^{(1)} = \frac{1}{1 + exp(-(\sum_k w_{jk}^{(1)} x_k + b_j^{(1)}))} \tag{8}$$

$$h_j^{(t)} = \frac{1}{1 + exp(-(\sum_k w_{jk}^{(t)} h_j^{(t-1)} + b_j^{(t)}))}$$

- Make the output layer non-linearity a smooth function
- Use standard numerical optimization methods (gradient descent) to learn the parameters. The algorithm is known as back propagation and was popularized by Rumelhart, Hinton and Willianms in the 1980s
- Activation Functions

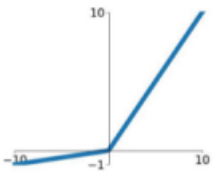
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



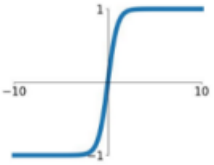
Leaky ReLU

$\max(0.1x, x)$



tanh

$\tanh(x)$

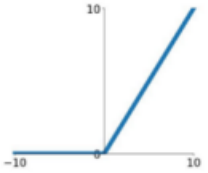


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

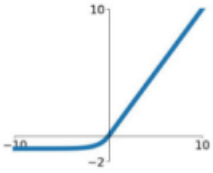
ReLU

$\max(0, x)$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



- Sigmoid or Logistic Activation Function
  - Sigmoid function translates the input ranged in $[-\infty; +\infty]$ to the range in $(0, 1)$
  - A more generalized sigmoid function that is used for **multiclass classification** called softmax function
  - Problems with sigmoid function
    - The $exp(.)$ function is computationally expensive
    - It has the problem of vanishing gradients
  - Tanh Activation Function
    - The tanh function is defined as

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{9}$$

    - It is bound to range $(-1, 1)$
    - The gradient is stronger (i.e., steeper (陡)) for tanh than sigmoid
    - Like sigmoid, tanh also has a vanishing gradient problem
    - In practice, optimization is easier for tanh hence in practice it is always preferred over sigmoid function
  - ReLU Activation Function
    - The ReLU (Rectified (矫正) Linear Units) function is defined as

$$ReLU(x) = \begin{cases} x, \ if \ x \geq 0 \\ 0, \ if \ x < 0 \end{cases} \tag{10}$$

    - ReLU is identity for positive values, and zero for negative values
    - Benefits of ReLU
      - Cheap to compute and easy to optimize
      - It converges faster
      - No vanishing gradient problem
      - Can output a true zero value, leading to representational sparsity
    - Problems with ReLU: If one neuron gets negative it is unlikely for it to recover (since the gradient is zero). This is called "**dying ReLU**" problem
  - Variants of ReLU
    - LReLU
      - Leaky ReLU is defined as:

$$LReLU(x) = \begin{cases} x, \ if \ x \geq 0 \\ ax, \ if \ x < 0 \end{cases} \tag{11}$$

- Leaky ReLU attpempts to fix the "dying ReLU" problem. Instead of the function being zero when $x < 0$, a leaky ReLU gives a small negative slope
- a is a parameter constrained to be postive. It can either be pre-determined or learned from data.
- ELU (Exponential Linear Unit)
  - The ELU is defined as: $ELU(x) = \begin{cases} x, \ if \ x \geq 0 \\ a(e^x - 1), \ if \ x < 0 \end{cases}$

        - It follows the same rule for $x \geq 0$ as ReLU, and increases exponentially for $x < 0$.
        - ELU tries to make the mean activations closer to zero which speeds up training (by adjusting a)
        - Empirically, ELU leads to higher classification results
    - Maxout Activation
        - The maxout activation function is defined as

$$Maxout(\mathbf{x}; \mathbf{w}_1, \mathbf{w}_2) = max\{\mathbf{w}_1^T\mathbf{x}, \mathbf{w}_2^T\mathbf{x}\} \tag{12}$$

        - It is piecewise linear that returns the maximum of the inputs
        - The maxout activation is a generalization of ReLU and leaky ReLU. It is a learnable activation function
        - The maxout neuron, therefore, enjoys all the benefits of ReLU (linear regime of operation, no saturation) and does not have its drawbacks (dying ReLU)
        - However, it doubles the total number of parameters for each neuron and hence, a higher total number of parameters need to be trained
    - Universal Approximation Theorem
        - It can be shown that a sigmoid network with one hidden layer (of infinite nodes) is a universal function approximator
            - In terms of classification, this means neural networks with one hidden layer (of unbounded size) can represent any decision boundary and thus have infinite capacity
            - It was also shown that deep networks can be more efficient at representing certain types of functions than shallow networks
            - It is not the specific choice of the activation function, but rather the multi-layer feedforward architecture itself which gives neural networks the potential of being universal approximators
            - It does not touch upon the algorithmic learnability of those parameters
        - More Failures
            - Through the 1990s, it became clear that while these models could represent arbitrarily complex functions and that deep networks should work better than shallow networks, no one could effectively train deep networks
            - This led to a shift away from neural networks towards statistical models and SVMs through the late 1990s and 2000s
            - Only a few groups continued to work on neural networks during this time period (Schmidhuber, Hinton, LeCun, Bengio, Ng)
            - Highly recommended reading: “Deep Learning: Our Miraculous Year 1990-1991” by Jürgen Schmidhuber

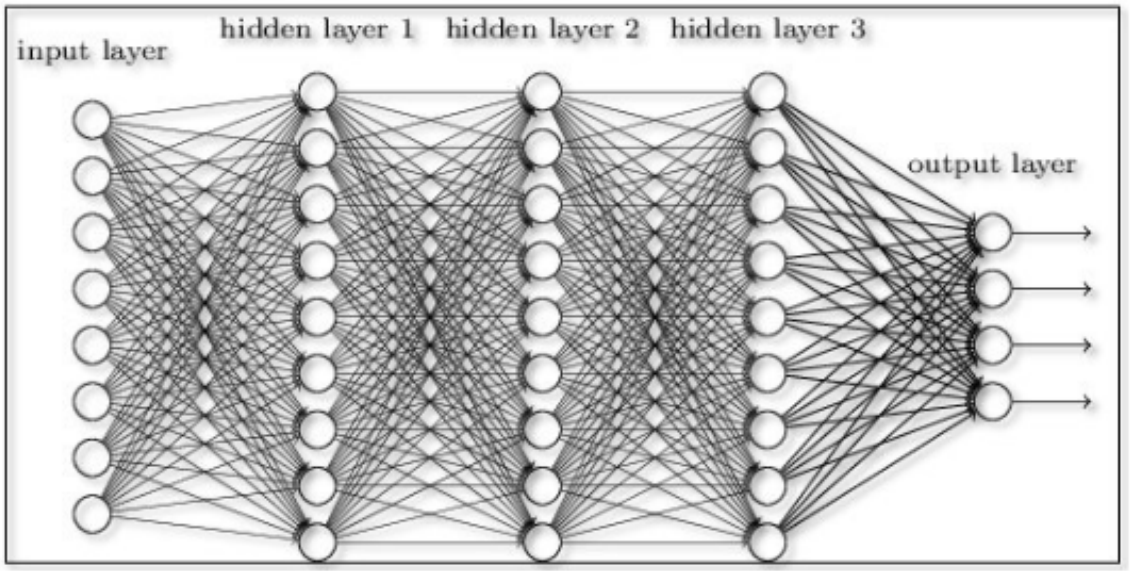
Deep Learning

- Solution to the deep learning problem (as of right now) appears to be:
    - Have access to lots of labeled data (i.e., millions of examples)
    - Make the non-linearities non-smooth again (rectified linear units, ReLU):

$$h_j^{(1)} = max(0, \sum_k w_{jk}^{(1)} x_k + b_j^{(1)}) \tag{13}$$
$$h_j^{(t)} = max(0, \sum_k w_{jk}^{(t)} h_k^{(t-1)} + b_j^{(t)})$$

  - Use improved optimization methods to learn the parameters
  - Use new regularization schemes to constrain the large number of model parameters in the network
  - Use GPUs for parallel computing with a massive 20-30 times speedup. Model training takes less than ten days for large vision problems instead of one year

- Main Idea
  - Learn a *feature hierarchy* all the way from raw data (i.e. image pixels for vision problems) to classifier
  - Each layer extracts features from the output of the previous layer
  - Train all layers jointly



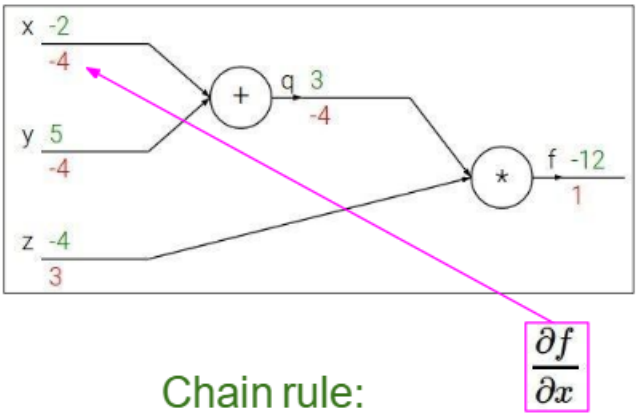
Backpropagation

- Backpropagation is a optimizing method to train deep neural network
- An example (non-vectorized)

Backpropagation: a simple example

$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Chain rule:

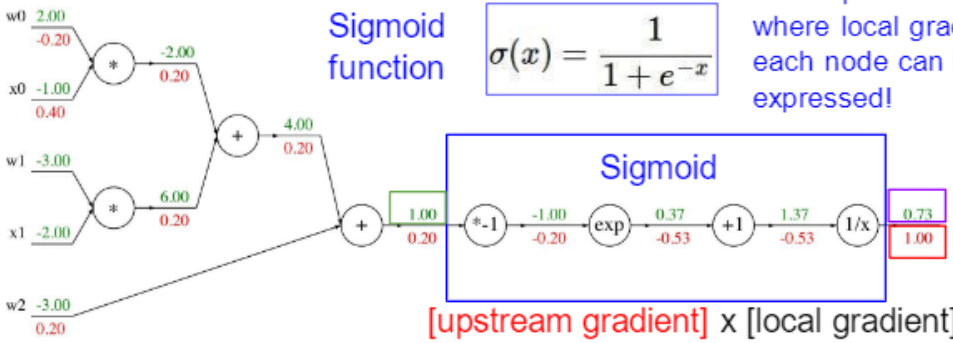
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

Upstream gradient — Local gradient

- Another example:

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!



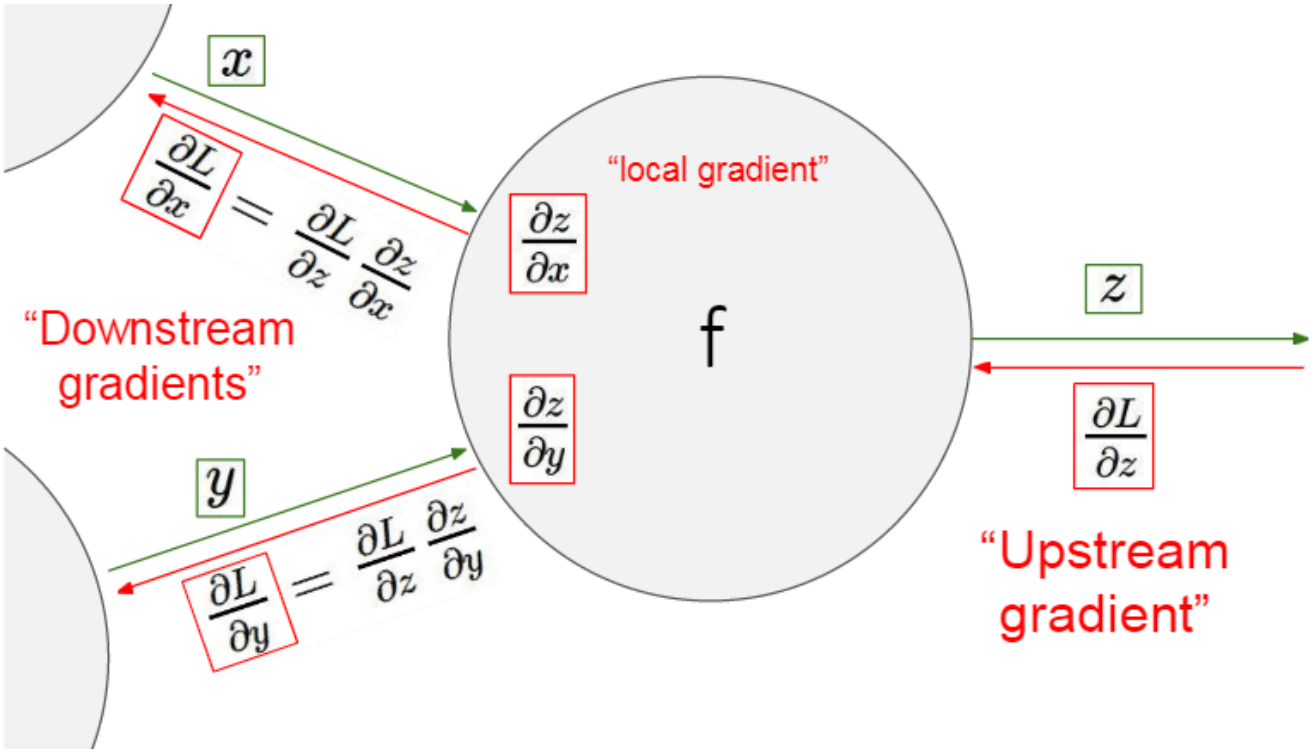
Sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$

[upstream gradient] x [local gradient]

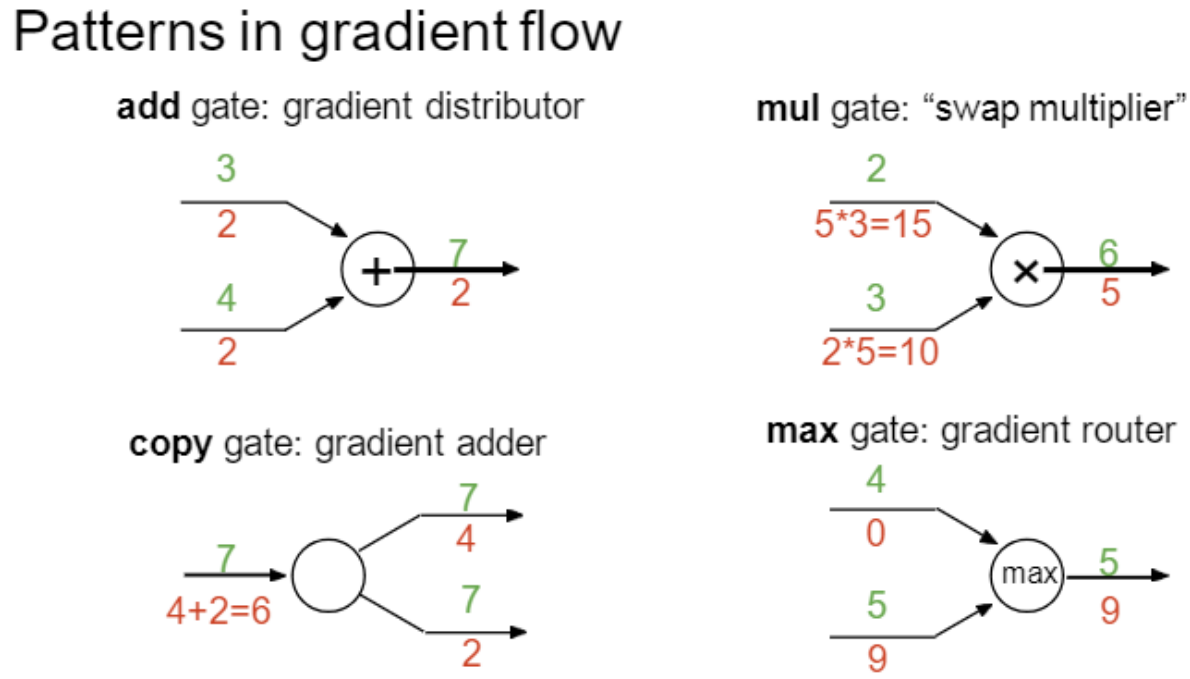
[1.00] x [(1 - 0.73) (0.73)] = 0.2

Sigmoid local gradient: $\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$

- General Case

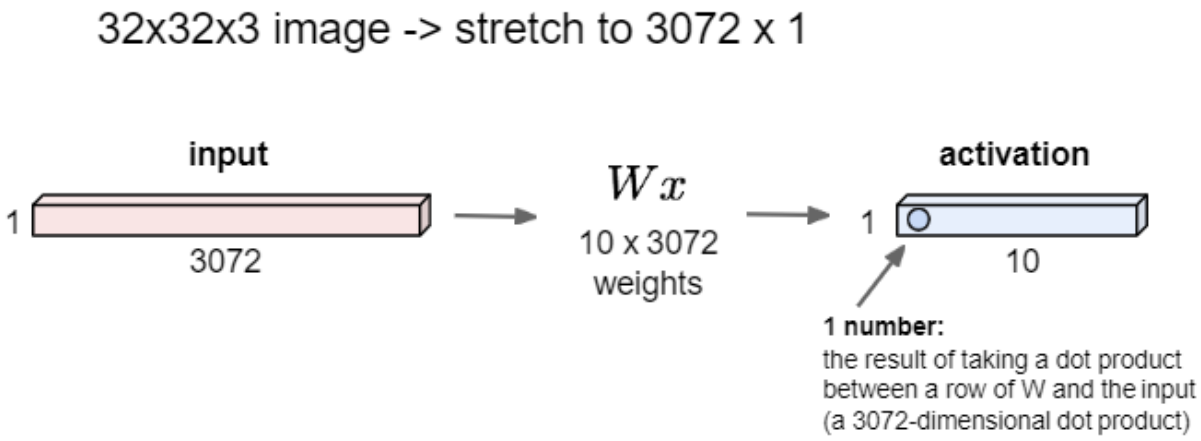


- Patterns

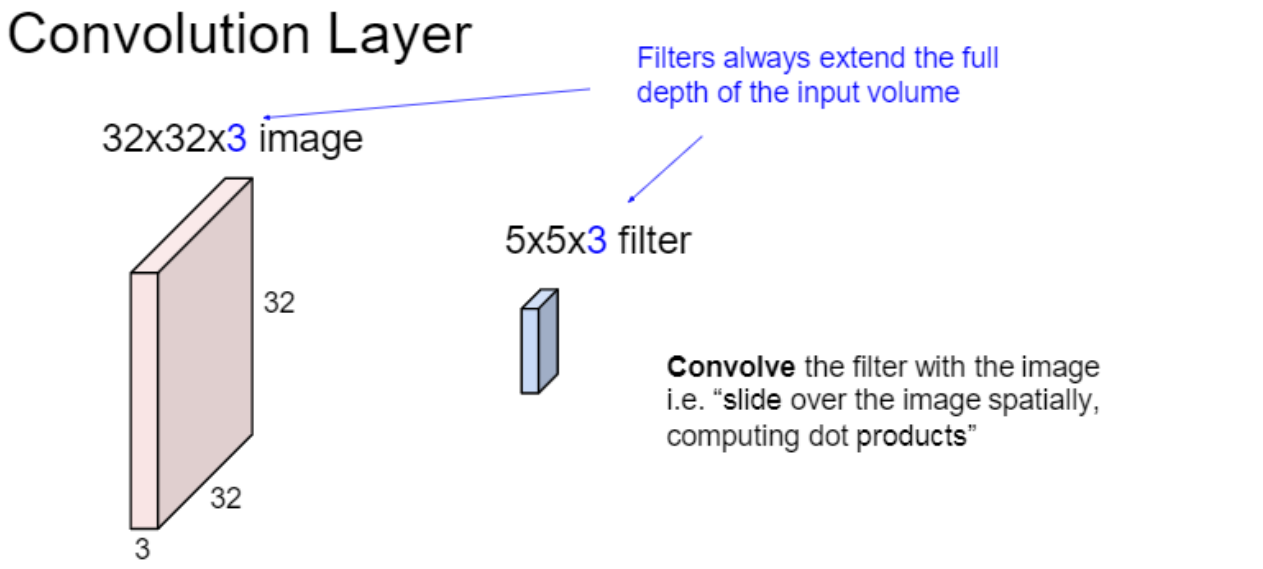


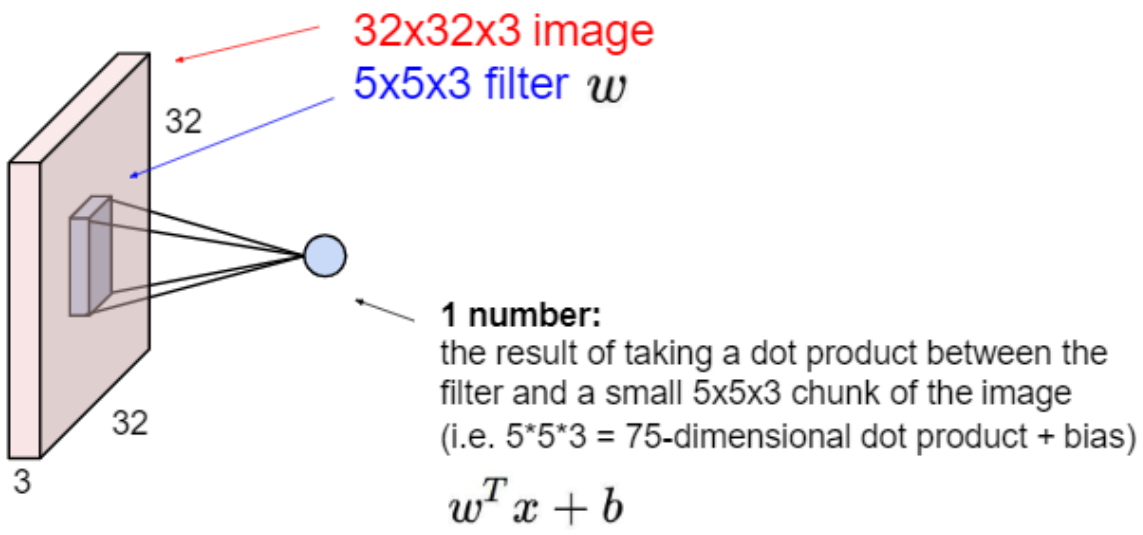
Convolutional Networks

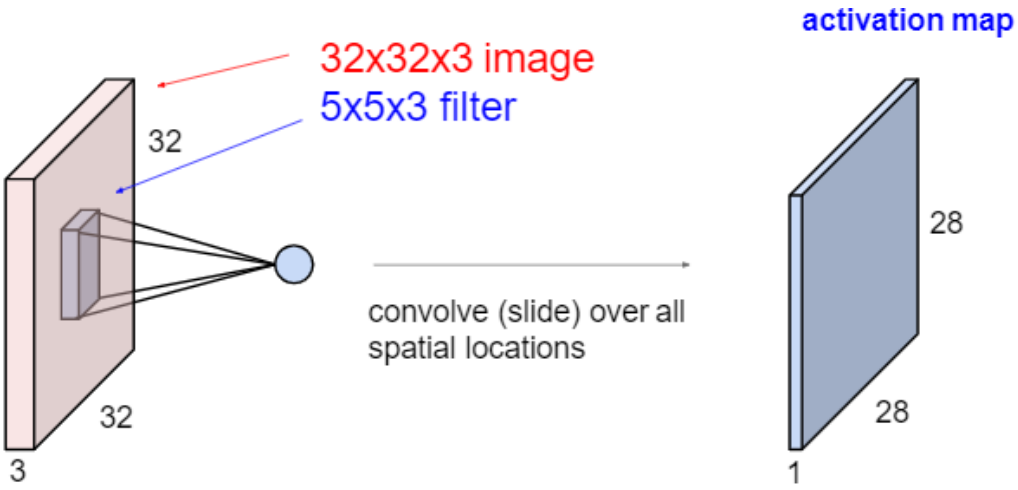
- Fully Connected Layer



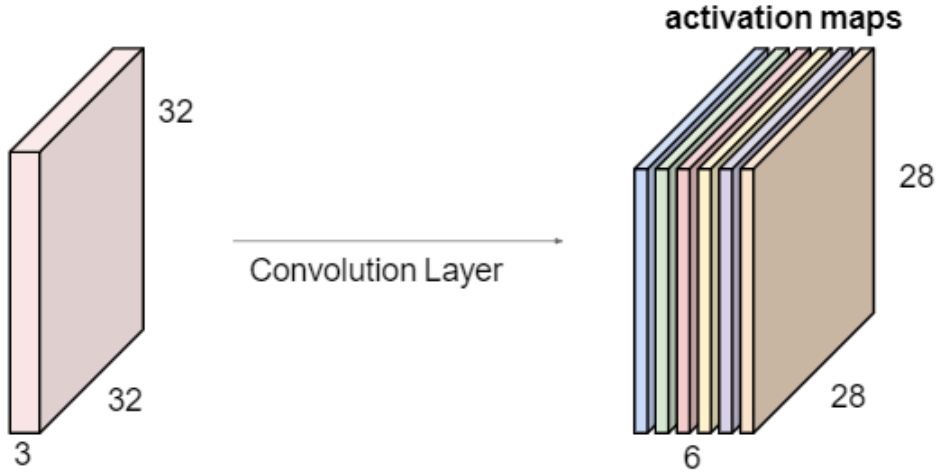
- Convolution Layer





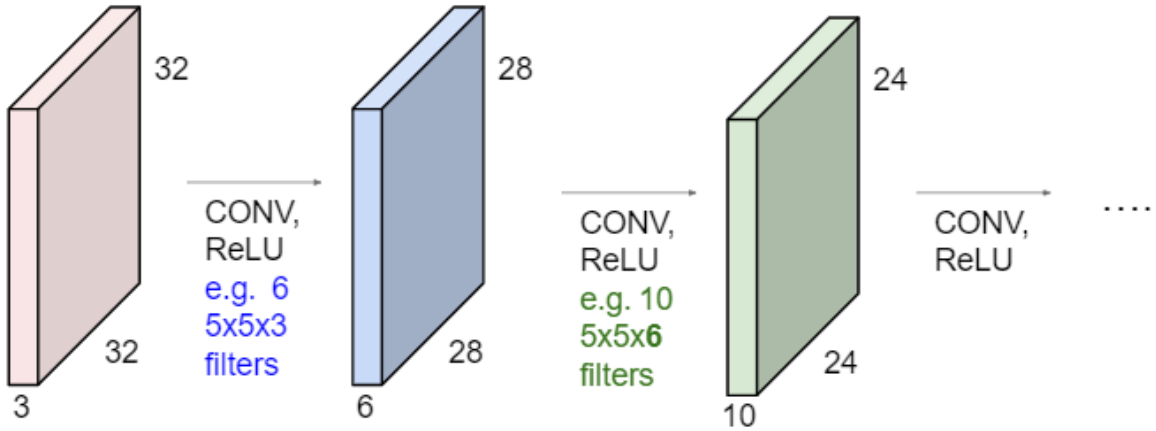


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

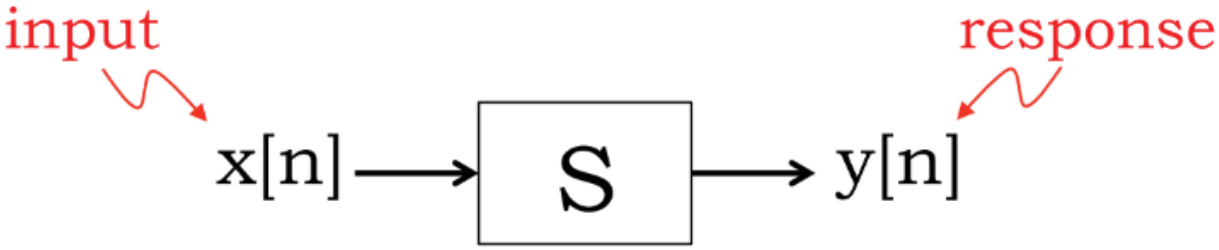


We stack these up to get a "new image" of size 28x28x6!

- ConvNet is a sequence of Convolution Layers, interspersed with activation function

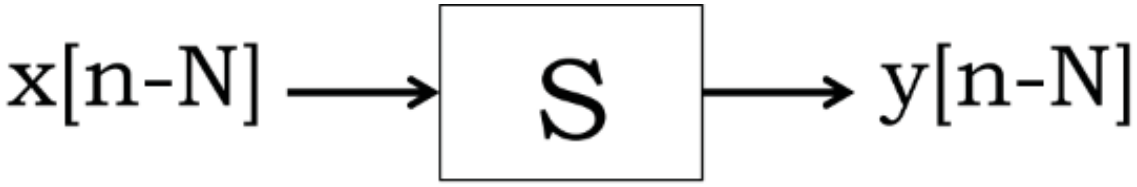


- Origin of convolution idea
  - Convolution arises from the field of signal processing, which describes the output (in terms of the input) of an important class of operations (or systems) known as linear time-invariant(LTI)



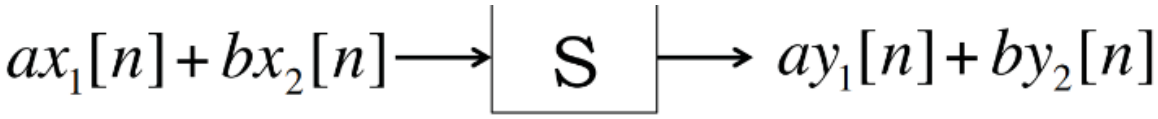
- A discrete-time signal such as $x[n]$ or $y[n]$ is described by an infinite sequence of values, $i.e.$, the time index n takes values in $-\infty$ to ∞. The above picture is a snapshot at a particular time n
- The sequence of output values $y[.]$ is the response of system S to the input sequence $x[.]$

- Time Invariant Systems
  - Let $y[n]$ be the response of S to input $x[n]$
  - If for all possible sequences $x[n]$ and integers N



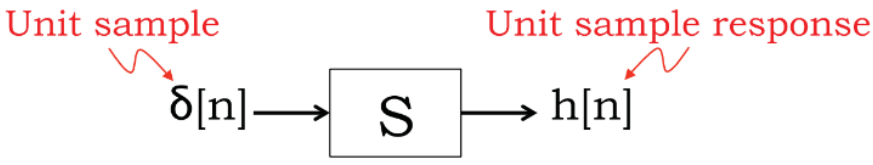
  - Then system S is said to be time invariant (TI). A time shift in the input sequence to S results in an identical time shift of the output sequence

- Linear Systems
  - Let $y_1[n]$ be the response of S to an arbitrary input $x_1[n]$ and $y_2[n]$ be the response to an arbitrary $x_2[n]$
  - If, for arbitrary scalar coefficients a and b, we have



  - Then system S is said to be linear. If the input is the weighted sum of several signals, the response is the superposition ($i.e.$, the same weighted sum) of the response to those signals
  - One key consequence: If the input is identically 0 for a linear system, the output must also be identically 0

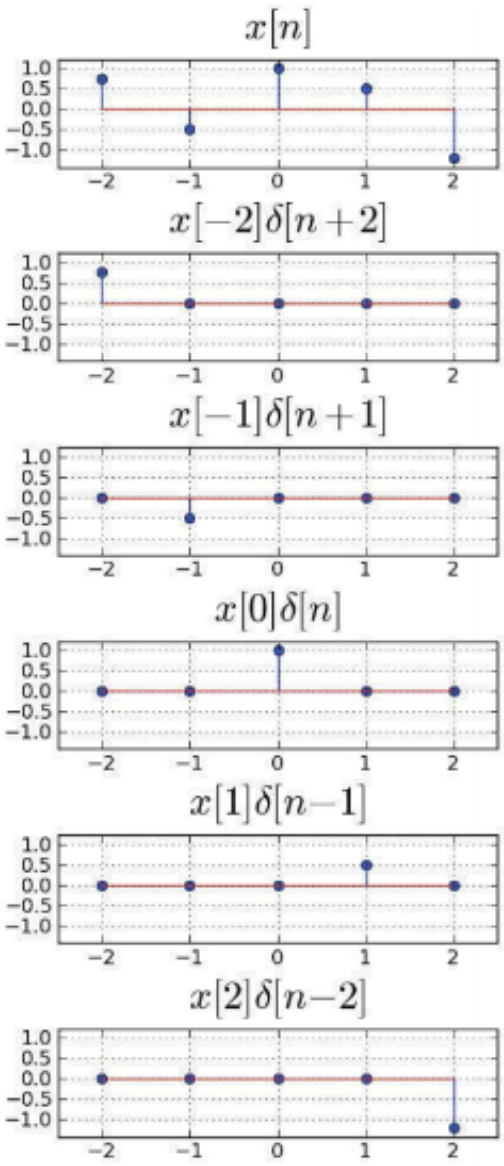
- Unit Sample Responses



$$\delta[n] = \begin{cases} 1, & \text{if } n = 0 \\ 0, & \text{otherwise} \end{cases} \qquad \text{and} \qquad \delta[n-N] = \begin{cases} 1, & \text{if } n = N \\ 0, & \text{otherwise.} \end{cases}$$

- The unit sample response of a system S is the response of the system to the unit sample input. We will always denote the unit sample response as $h[n]$.

- Unit Sample Decomposition
  - A discrete-time signal can be decomposed into a sum of time-shifted, scaled unit samples
  - Example:

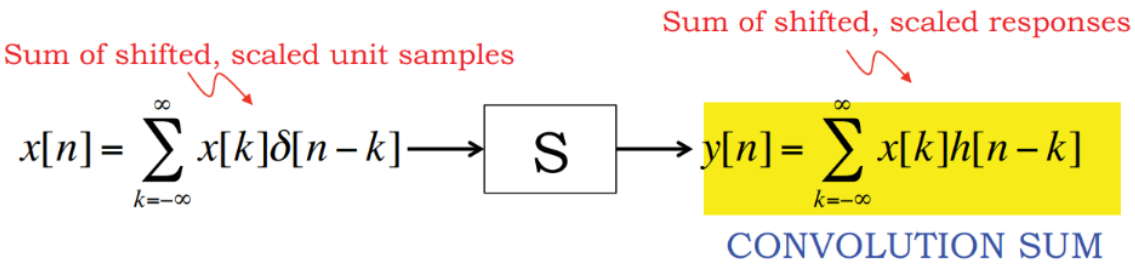


$$x[n] = x[-2]\delta[n+2] + x[-1]\delta[x+1] + x[0]\delta[n] + x[1]\delta[n-1] + x[2]\delta[n-2] \tag{14}$$

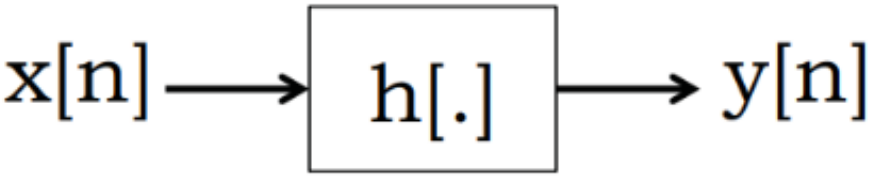
  - In general, for any particular index, only one term of this sum is non-zero

$$x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n-k] \tag{15}$$

- Modeling LTI Systems
  - If system S is both linear and time-invariant, then we can use the unit sample response to predict the response to any input waveform $x[n]$



- Indeed, the unit sample response $h[n]$ completely characterizes the LTI system S, so you often see



- Discrete Convolution: Discrete Convolution typically contains the following steps:
  1. List the index 'k' covering a sufficient range
  2. List the input $x[k]$
  3. Obtain the reversed sequence $h[-k]$, and align the rightmost element of $h[n-k]$ to the leftmost element of $x[k]$
  4. Cross-multiply and sum ($i.e.$, dot product) the nonzero overlap terms to produce $y[n]$
  5. Slide $h[n-k]$ to the right by one position
  6. Repeat Steps 4 and 5; stop if all the output values are zero or if required
- Example

Example: Find the convolution of the two sequences x[n] and h[n] given by,

x[k] = [3 1 2] h[k] = [3 2 1]

k:	-2	-1	0	1	2	3	4	5
x[k]:			3	1	2			
h[-k]:	1	2	3					
h[1-k]:		1	2	3				
h[2-k]:			1	2	3			
h[3-k]:				1	2	3		
h[4-k]:					1	2	3	
h[5-k]:						1	2	3

Hint: The value of k starts from (**– length of h + 1**) and continues till (**length of h + length of x – 1**)

Here ***k*** starts from **-3 + 1 = -2** and continues till **3 + 3 – 1 = 5**

y[0] = 3 x 3 = 9

y[1] = 3 x 2 + 3 x 1 = 9

y[2] = 3 x 1 + 1 x 2 + 2 x 3= 11

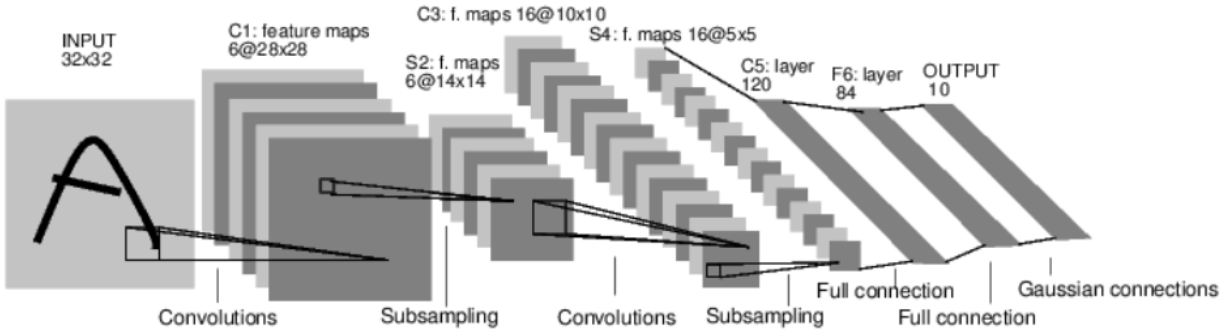
y[3] = 1 x 1 + 2 x 2 = 5

y[4] = 2 x 1 = 2

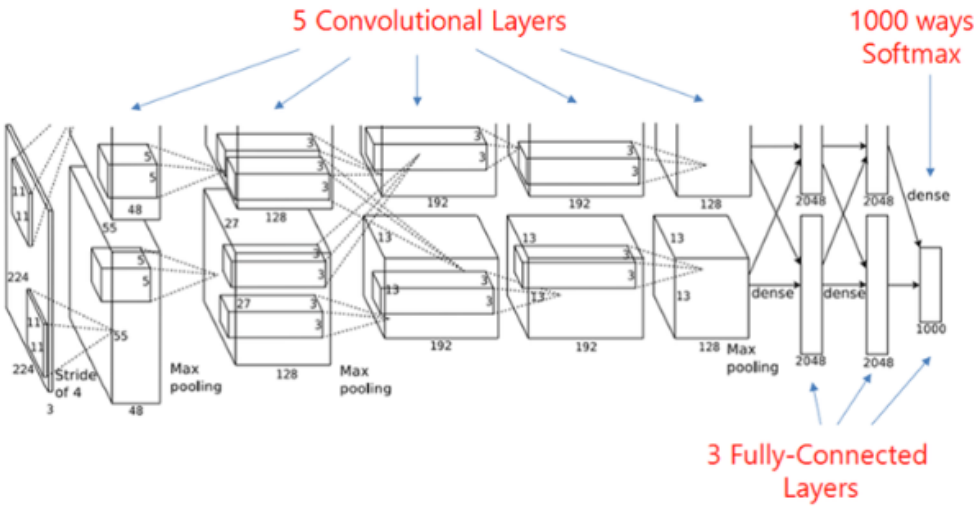
y[5] = 0 (no overlap)

$$y[n] = \{9 \quad 9 \quad 11 \quad 5 \quad 2 \quad 0\}$$

- Representative CNNs
  - LeNet-5



    - Average pooling (sub-sampling)
    - Sigmoid or tanh nonlinearity
    - Fully connected layers at the end
    - Trained on MNIST digit dataset with 60K training examples
    - Ref: Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,”Proc. IEEE, 1998.
  - AlexNet, 2012

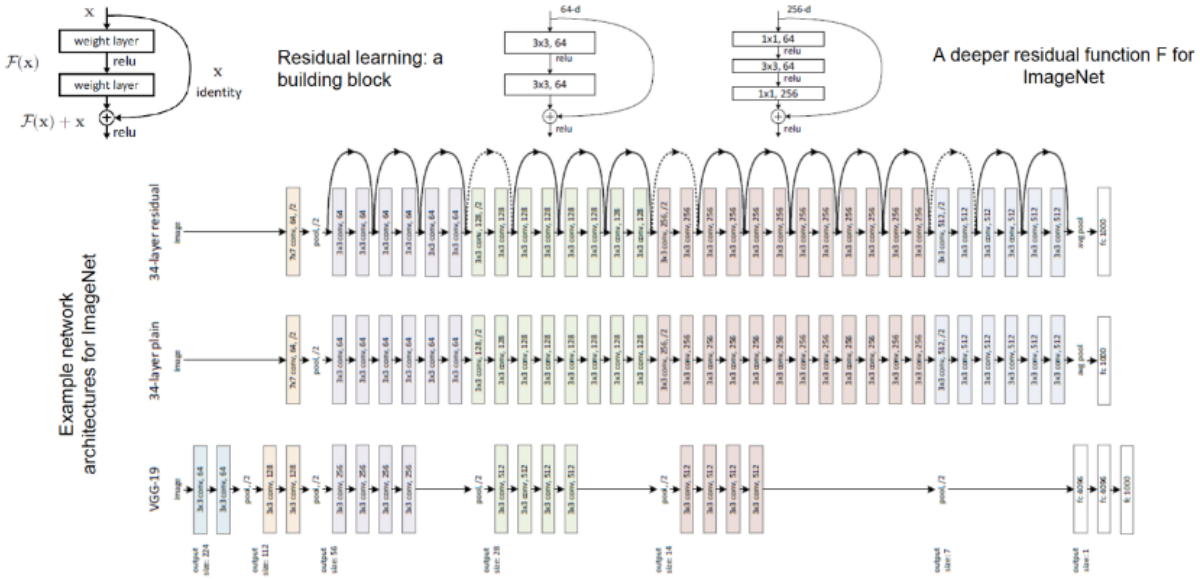


    - The **first** winner deep model in computer vision
    - 5 convolutional layers + 3 fully-connected layers + softmax classifier
    - Key technical features: ReLU, dropout, data augmentation
    - Ref: A. Krizhevsky, I. Sutskever, and G. Hinton, “Imagenet classification with deep convolutional neural networks.” inNIPS, 2012.
  - VGG-Net, 2014

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

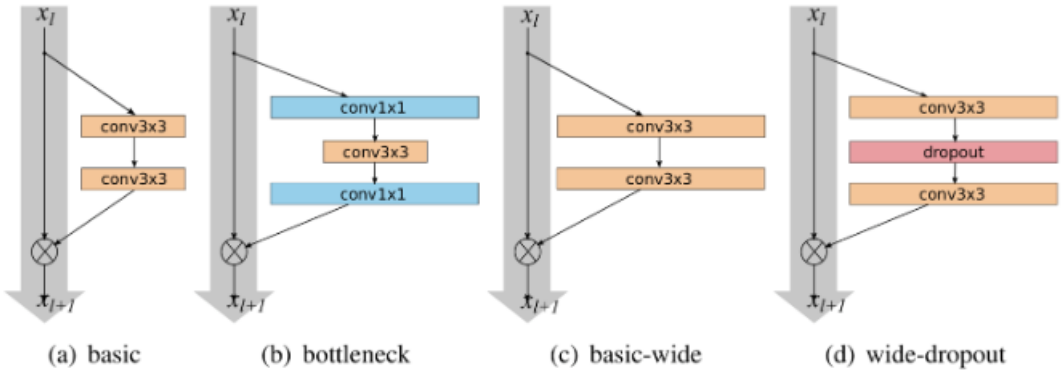
- Key technical features: Increased depth (up to 19) and smaller filter size (3)
- Configurations D and E are widely used for various tasks, called VGG-16 and VGG-19
- Ref: K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” inICLR, 2015.

- Deep Residual Network (ResNet), 2015



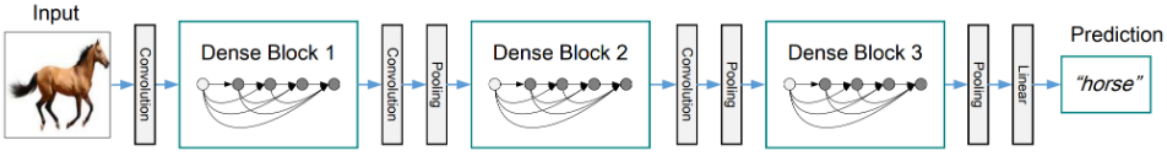
- Key technical features: Skip connections for residual mapping up to > 1000 layers
- Ref: K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” inCVPR, 2016.

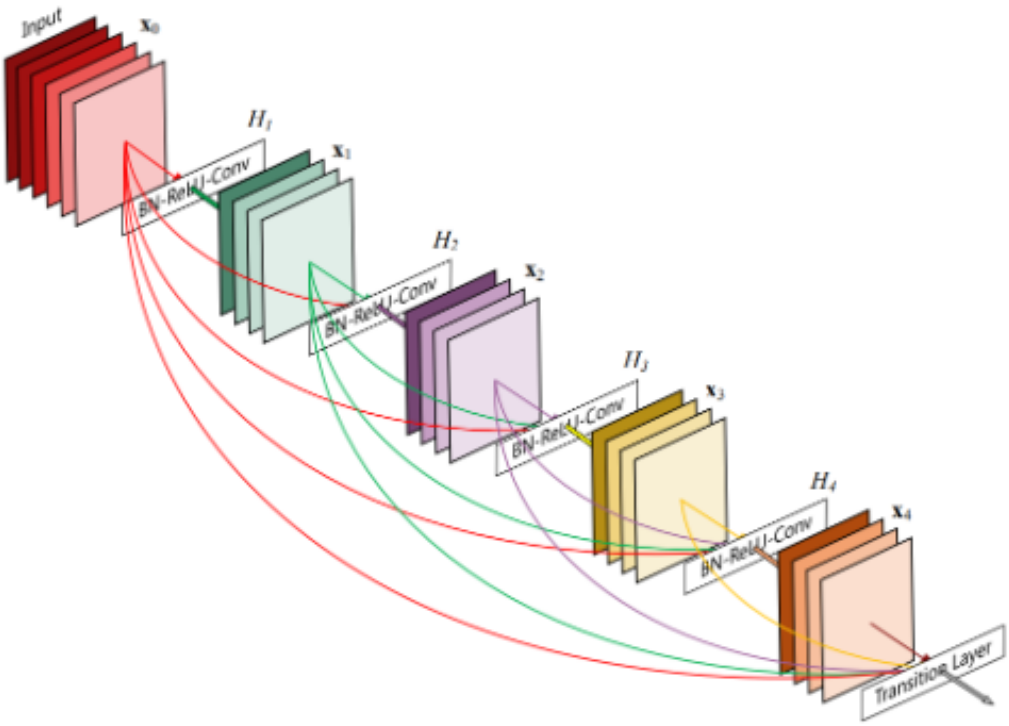
- Wide ResNet, 2016



- Widening of ResNet blocks (if done properly) provides a more effective way of improving performance of residual networks compared to increasing their depth
- A wide 16-layer deep network has the same accuracy as a 1000-layer thin deep network and a comparable number of parameters, although being several times faster to train
- Ref: S. Zagoruyko and N. Komodakis, “Wide residual networks,” inBMVC, 2016.

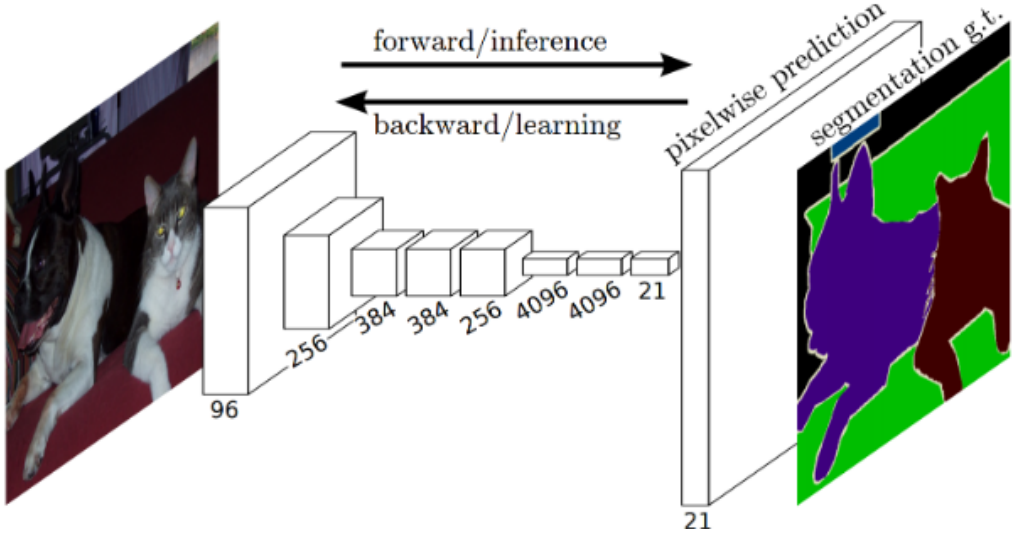
- Densely Connected Networks (DenseNet), 2017





- Key technical features: Finer combination of multi-scale features
- Ref: G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” inCVPR, 2017.

- More Art of Convolutions:
  - Fully Convolutional Networks (FCN), 2014

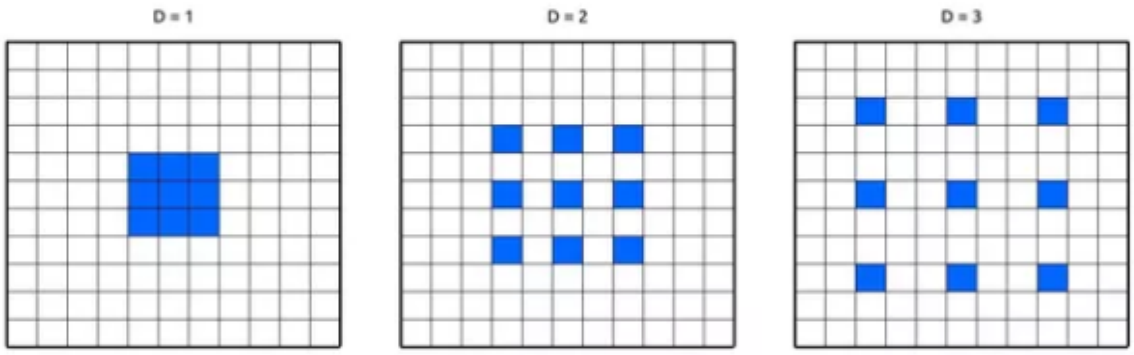


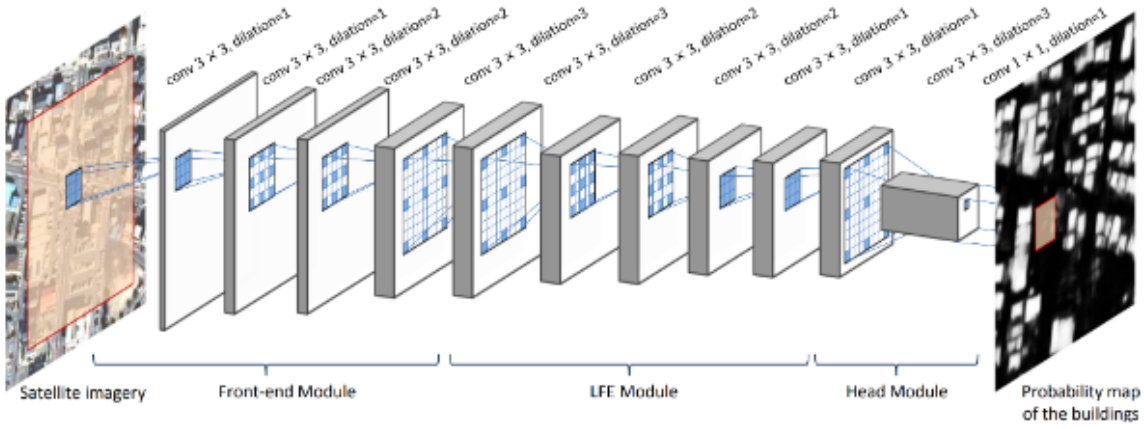
- Key technical features: No fully-connected layer => No fixed requirement on input size
- Widely adopted in pixel-to-pixel prediction tasks, $e.g.$, image segmentation
- Ref: J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” inCVPR, 2015.

  - U-Net, 2015
    - The architecture consists of a contracting path to capture context
    - and a symmetric expanding path to enable precise localization

- Very popular backbone for dense prediction (image segmentation, restoration, etc.)
- Ref: O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional networks for biomedical image segmentation,” inMICAI, 2015.

- Dilated Convolution, 2015





- Aggregates multi-scale contextual information without losing resolution or analyzing rescaled images
- Has layer receptive fields with the same computation and memory cost
- Improves the performance on dense prediction tasks,e.g.,semantic segmentation
- Ref: F. Yu, and K. Vladlen, “Multi-scale context aggregation by dilated convolutions,” in ICLR, 2016.

Lecture 13: Deep Learning -- Optimization

Optimization and Implementation

- Gradient Descent (GD)

Algorithm 1 Batch Gradient Descent at Iteration t

Require: Learning Rate $\alpha^{(t)}$
Require: Initial Parameter $\boldsymbol{\theta}^{(0)}$
1: **while** stopping criteria not met **do**
2: Compute gradient estimate over m examples:
3: $\hat{\mathbf{g}} = \frac{1}{m} \nabla_{\boldsymbol{\theta}^{(t)}} \sum_{i=1}^{m} \ell(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(t)}), y^{(i)})$
4: Apply update: $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha^{(t)} \hat{\mathbf{g}}$
5: $t \leftarrow t + 1$
6: **end while**

- Pros: Gradient estimates are stable
- Cons: Need to compute gradients over the entire training set for one update

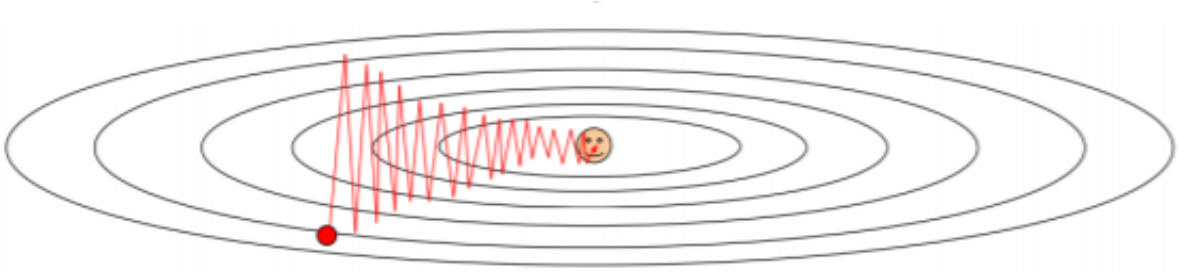
- Stochastic Gradient Descent (SGD)

Algorithm 2 Stochastic Gradient Descent at Iteration t

Require: Learning Rate $\alpha^{(t)}$
Require: Initial Parameter $\boldsymbol{\theta}^{(0)}$
1: **while** stopping criteria not met **do**
2: Sample example $(\mathbf{x}^{(i)}, y^{(i)})$ from the training set
3: Compute gradient estimate:
4: $\hat{\mathbf{g}} = \nabla_{\boldsymbol{\theta}^{(t)}} \ell(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(t)}), y^{(i)})$
5: Apply update: $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha^{(t)} \hat{\mathbf{g}}$
6: $t \leftarrow t + 1$
7: **end while**

- Pros: Computation time per update does not depend on m, allowing convergence on extremely large datasets
- Cons: Gradient estimates can be noisy. Use mini batches

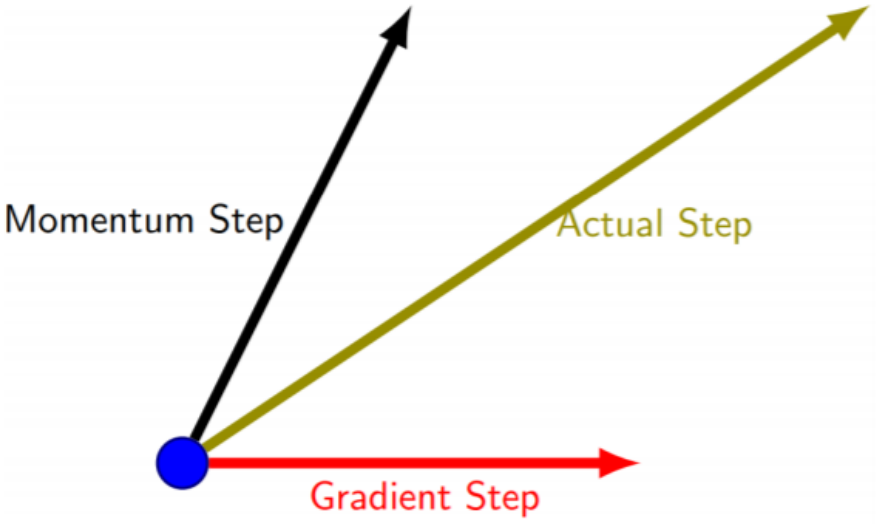
- Problems with GD and SGD
  - What if loss changes quickly in one direction and slowly in another ($i.e.$, error surface has high curvature)



- What does gradient descent do: Very slow progress along shallow dimension, jittering along steep direction

- Momentum
  - How do we try and solve this problem
  - Introduce a new variable $\mathbf{v}$, the velocity
  - We think of $\mathbf{v}$ as the direction and speed by which the parameters move as the learning dynamics progress
  - The velocity is an exponentially decaying moving average of the negative gradients

$$\mathbf{v}^{t+1} = \rho\mathbf{v}^{t} - \alpha^{(t)}\nabla_{\boldsymbol{\theta}^{(t)}} l(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(t)}), y^{(i)}) \tag{16}$$
$$\alpha \in [0, 1)$$
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \mathbf{v}^{(t+1)}$$



  - The velocity accumulates the previous gradients
  - The role of ρ
    - If ρ is larger than $\alpha^{(t)}$, the current update is more affected by the previous gradients
    - Usually values for ρ are set high ≈ 0.9
  - Performance

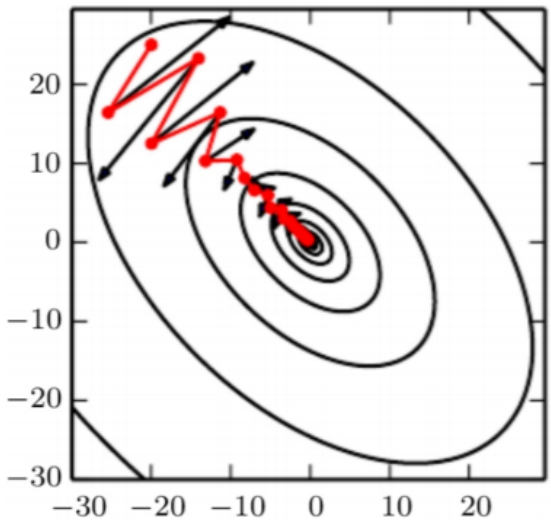


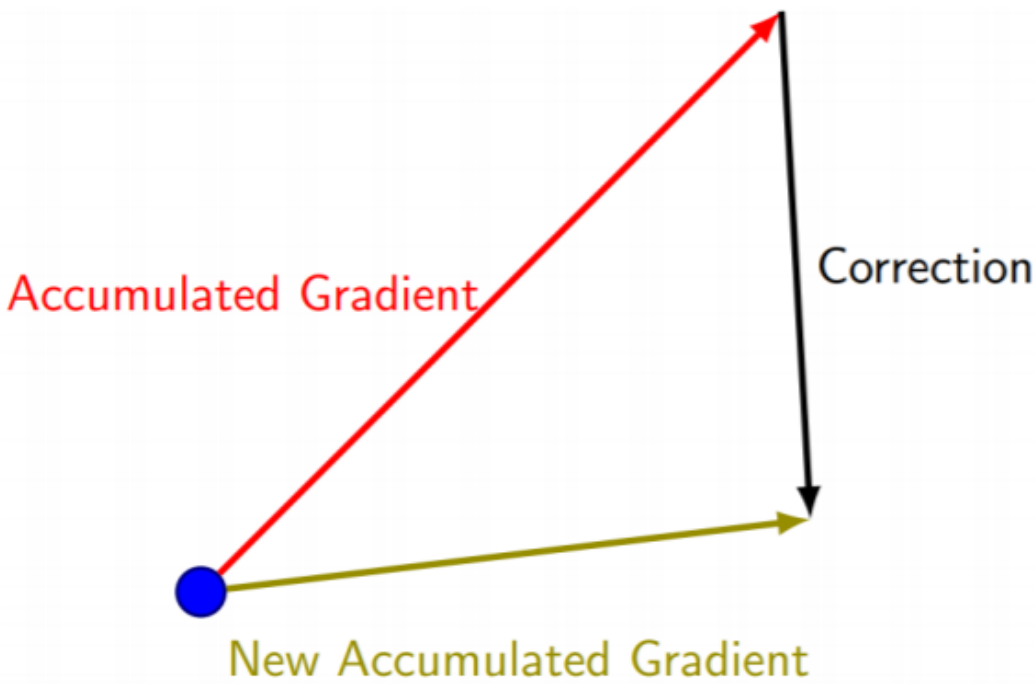
Illustration of how momentum traverses such an error surface better compared to SGD

- Pseudo code

Algorithm 3 SGD with Momentum at Iteration t

Require: Learning Rate $\alpha^{(t)}$
Require: Momentum Parameter ρ
Require: Initial Parameter $\boldsymbol{\theta}^{(0)}$
Require: Initial Velocity $\mathbf{v}^{(0)}$
while stopping criteria not met **do**
  Sample example $(\mathbf{x}^{(i)}, y^{(i)})$ from the training set
  Compute gradient estimate: $\hat{\mathbf{g}} = \nabla_{\boldsymbol{\theta}^{(t)}} \ell(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(t)}), y^{(i)})$
  Compute the velocity update: $\mathbf{v}^{(t+1)} = \rho \mathbf{v}^{(t)} - \alpha^{(t)} \hat{\mathbf{g}}$
  Apply update: $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \mathbf{v}^{(t+1)}$
  $t \leftarrow t + 1$
end while

- Nesterov Momentum
  - Another approach:
    - First take a step in the direction of the accumulated gradient
    - Then calculate the gradient and make a correction



- Equation

$$\begin{aligned} \hat{\boldsymbol{\theta}} &= \boldsymbol{\theta}^{(t)} + \rho \mathbf{v}^{(t)} \\ \mathbf{v}^{(t+1)} &= \rho \mathbf{v}^{t} - \alpha^{(t)} \nabla_{\boldsymbol{\theta}^{(t)}} l(f(\mathbf{x}^{(i)}; \hat{\boldsymbol{\theta}}), y^{(i)}) \\ \boldsymbol{\theta}^{(t+1)} &= \boldsymbol{\theta}^{(t)} + \mathbf{v}^{(t+1)} \end{aligned} \tag{17}$$

- Pseudo code

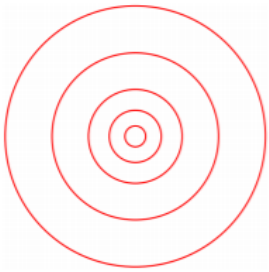
Algorithm 4 SGD with Nesterov Momentum at Iteration t

Require: Learning Rate $\alpha^{(t)}$
Require: Momentum Parameter ρ
Require: Initial Parameter $\boldsymbol{\theta}^{(0)}$
Require: Initial Velocity $\mathbf{v}^{(0)}$

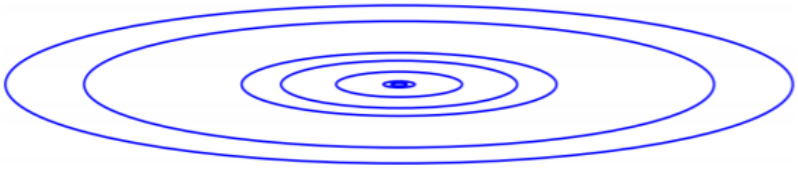
while stopping criteria not met **do**
  Sample example $(\mathbf{x}^{(i)}, y^{(i)})$ from the training set
  Update parameters: $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta}^{(t)} + \rho \mathbf{v}^{(t)}$
  Compute gradient estimate: $\hat{\mathbf{g}} = \nabla_{\tilde{\boldsymbol{\theta}}} \ell(f(\mathbf{x}^{(i)}; \tilde{\boldsymbol{\theta}}), y^{(i)})$
  Compute the velocity update: $\mathbf{v}^{(t+1)} = \rho \mathbf{v}^{(t)} - \alpha^{(t)} \hat{\mathbf{g}}$
  Apply update: $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \mathbf{v}^{(t+1)}$
  $t \leftarrow t + 1$
end while

- Adaptive Learning Rate Methods
  - Till now we assign the same learning rate to all parameters θ_j's

- If θ_j's vary in importance and frequency, why is this a good idea?
- It's probably not



Nice (all features are equally important)



Harder!

- AdaGrad

Algorithm 5 AdaGrad

Require: Global Learning Rate α
Require: Initial Parameter $\boldsymbol{\theta}^{(0)}$
Initialize $\mathbf{r}^{(0)} = 0$
while stopping criteria not met **do**
  Sample example $(\mathbf{x}^{(i)}, y^{(i)})$ from the training set
  Compute gradient estimate: $\hat{\mathbf{g}} = \nabla_{\boldsymbol{\theta}^{(t)}} \ell(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(t)}), y^{(i)})$
  Accumulate: $\mathbf{r}^{(t+1)} = \mathbf{r}^{(t)} + \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
  Compute update: $\Delta\boldsymbol{\theta} = -\frac{\alpha}{\delta + \sqrt{\mathbf{r}^{(t+1)}}} \odot \hat{\mathbf{g}}$
  Apply update: $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \Delta\boldsymbol{\theta}$
  $t \leftarrow t + 1$
end while

  - δ is used to make sure not dividing zero
  - Idea: Scale the gradient of a model parameter by square root of sum of squares of all its historical values
  - Progress along "steep" directions (with large partial derivatives) is damped
  - Progress along "flat" directions (with small partial derivatives) is accelerated
  - What happens to the step size over long time? Decays to zero
- RMSProp: "Leaky AdaGrad"
  - AdaGrad is good when the objective is convex
  - AdaGrad might shrink the learning rate too aggressively, we want to keep the history in mind
  - We can adapt it to perform better in non-convex settings by accumulating an exponentially decaying average of gradient

- This is an idea that we use again and again in deep learning
- Has about 2,500 citations on Google Scholar (As of Aug. 2019), but was proposed in a slide Geoffrey Hinton's Cousera course

Algorithm 6 RMSProp

Require: Global Learning Rate α
Require: Decay Parameter ρ
Require: Initial Parameter $\boldsymbol{\theta}^{(0)}$
Initialize $\mathbf{r}^{(0)} = 0$
while stopping criteria not met **do**
  Sample example $(\mathbf{x}^{(i)}, y^{(i)})$ from the training set
  Compute gradient estimate: $\hat{\mathbf{g}} = \nabla_{\boldsymbol{\theta}^{(t)}} \ell(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(t)}), y^{(i)})$
  Accumulate: $\mathbf{r}^{(t+1)} = \rho \mathbf{r}^{(t)} + (1 - \rho)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
  Compute update: $\Delta\boldsymbol{\theta} = -\frac{\alpha}{\delta + \sqrt{\mathbf{r}^{(t+1)}}} \odot \hat{\mathbf{g}}$
  Apply update: $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \Delta\boldsymbol{\theta}$
  $t \leftarrow t + 1$
end while

- Adam: ADAptive Moments
  - Adam is currently THE default optimization algorithm for training deep neural networks
  - Introduced by Kingma and Ba in 2015 and quickly accumulated more than 25,000 Google Scholar Citations as of Aug. 2019
  - Adam is like RMSProp with Momentum but with bias correction terms for the first and second moments

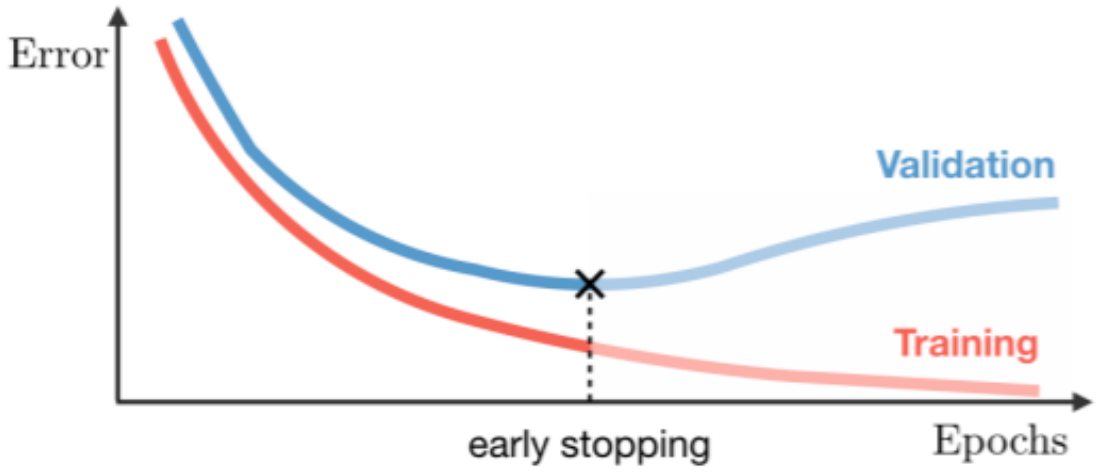
Algorithm 7 Adam

Require: Global Learning Rate: α (set to 0.0001), Decay Rates: ρ_1 (set to 0.9), ρ_2 (set to 0.9), and Initial Parameter: $\boldsymbol{\theta}^{(0)}$

Initialize moment variables $\mathbf{s}^{(0)} = 0$ and $\mathbf{r}^{(0)} = 0$
while stopping criteria not met **do**
  Sample example $(\mathbf{x}^{(i)}, y^{(i)})$ from the training set
  Compute gradient estimate: $\hat{\mathbf{g}} = \nabla_{\boldsymbol{\theta}^{(t)}} \ell(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(t)}), y^{(i)})$
  Update: $\mathbf{s}^{(t+1)} = \rho_1 \mathbf{s}^{(t)} + (1 - \rho_1)\hat{\mathbf{g}}$
  Update: $\mathbf{r}^{(t+1)} = \rho_2 \mathbf{r}^{(t)} + (1 - \rho_2)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
  Correct biases: $\hat{\mathbf{s}} = \frac{\mathbf{s}^{(t+1)}}{1-\rho_1^{t+1}}$ and $\hat{\mathbf{r}} = \frac{\mathbf{r}^{(t+1)}}{1-\rho_2^{t+1}}$
  Compute and apply update: $\Delta\boldsymbol{\theta} = -\alpha \frac{\hat{\mathbf{s}}}{\delta + \sqrt{\hat{\mathbf{r}}}}$, $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \Delta\boldsymbol{\theta}$
  $t \leftarrow t + 1$
end while

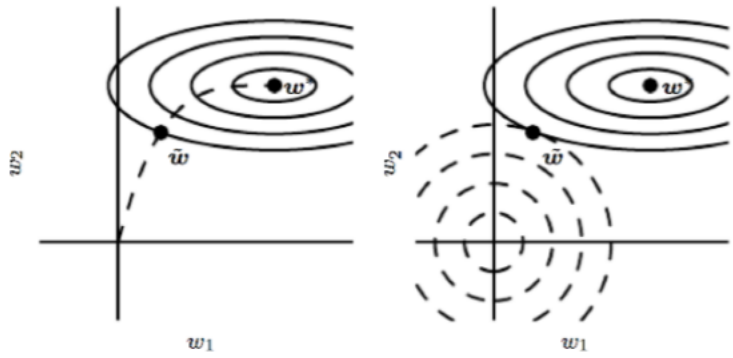
Regularizations

- Data Augmentation
  - Adding noise to the input: a special kind of augmentation
  - Be careful about the transformation applied => label preserving
    - Example: classifying ‘b’ and ‘d’; ‘6’ and ‘9’
- Early Stopping



  - Idea: Don’t train the network to t small training error
  - Recall overfitting: With a larger hypothesis class, it is easier to find a hypothesis that fits training samples
  - Prevent overfitting: Use validation error to decide when to stop

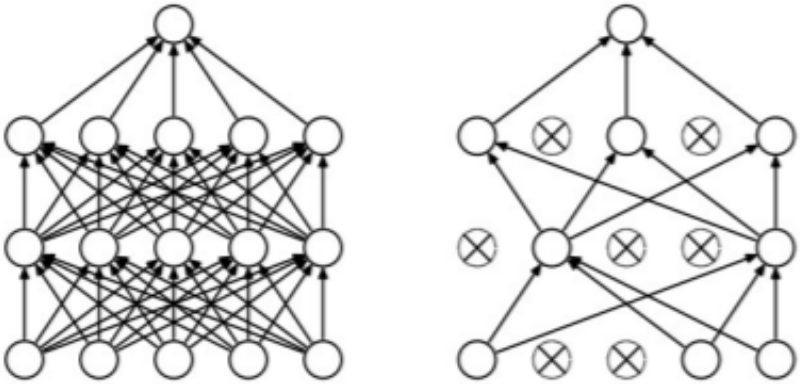
Early Stopping as Regularization



Qualitatively, the number of degrees of freedom in the network starts out small and grows during the training process, corresponding to a steady increase in the effective complexity of the model. Early stopping then represents a way of limiting the effective network complexity

- In practice, when training, also output validation error
- Every time validation error improves, store a copy of the network weights
- When validation error plateaus for some time, stop
- Return the copy of the weights stored
- The number of training steps (i.e., iterations) is also a hyperparameter

- Dropout



(a) Standard Neural Net (b) After applying dropout.

- In each update step, randomly sample a different binary mask to all the input and hidden units
- Multiple the mask with the units and do the update as usual
- Typical dropout probability: 0.8 for input and 0.5 for hidden units
- Very useful for fully connected layers, less for convolutional layers
- How can this possibly be a good idea?

        - Forces the network to have a redundant representation; prevents co-adaptation of features
        - Trains a large ensemble of models (that share parameters); Each binary mask corresponds to one model
    - How to vote to make a final prediction
        - Mask sampling: Randomly sample some (typically, 10 ~20) masks
        - For each mask, apply it to the trained model to get a prediction
        - Average the prediction
        - Ref: N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," JMLR, 2014.
- Batch Normalization
    - Idea: Adjusts activations to lie within a desired operating range, while maintaining their relative values
    - Given a mini-batch $\mathrm{B} = \{(\mathbf{x}^{(i)}, y^{(i)}), i = 1, \ldots, m\}$
    - Compute per-channel ($i.e.$ feature) mean: $\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$
    - Compute per-channel variance: $\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2$
    - Normalize $\mathbf{x}^{(i)}$ across channel: $\hat{x}_j^{(i)} = \frac{x^{(i)} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$
    - Issue: What if zero-mean, unit variance is too hard of a constraint
    - Incorporate learnable scale and shift parameters γ and β

$$z_j^{(i)} = \gamma_j \hat{x}_j^{(i)} + \beta_j \tag{18}$$

    - Test time
        - Mean and variance estimates depend on mini-batch; can't do this at test-time
        - Solution: Keep exponentially decaying moving average of values seen during training and use them for testing
        - During testing, batch normalization becomes a linear operator; can be fused with the previous convolutional layer
        - Learning $\gamma_j = \sigma_j$ and $\beta_j = \mu_j$ will recover the identity function
        - Usually inserted after convolutional layers and before nonlinearity
        - Behaves differently during training and testing: this is a very common source of bugs

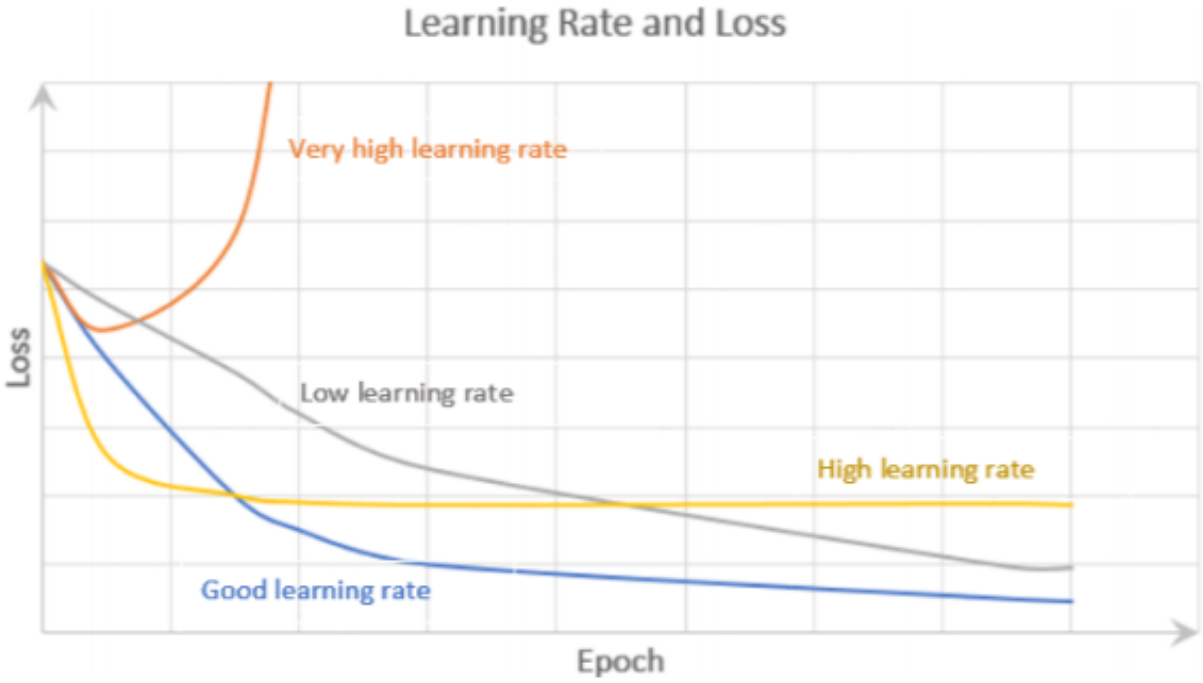
Practical Tricks

- Data pre-processing
    - Simplest: Zero-center the data, and then normalize them
        - Makes sense only when data attributes have approximately equal importance but different scales
    - PCA whitening: The data is first centered, and then projected into the eigenbasis, followed by dividing every dimension by the corresponding eigenvalue
    - Never forget to shuffle your data each epoch for SGD input
- Weight Initialization
    - All zero initialization: Terribly Wrong!

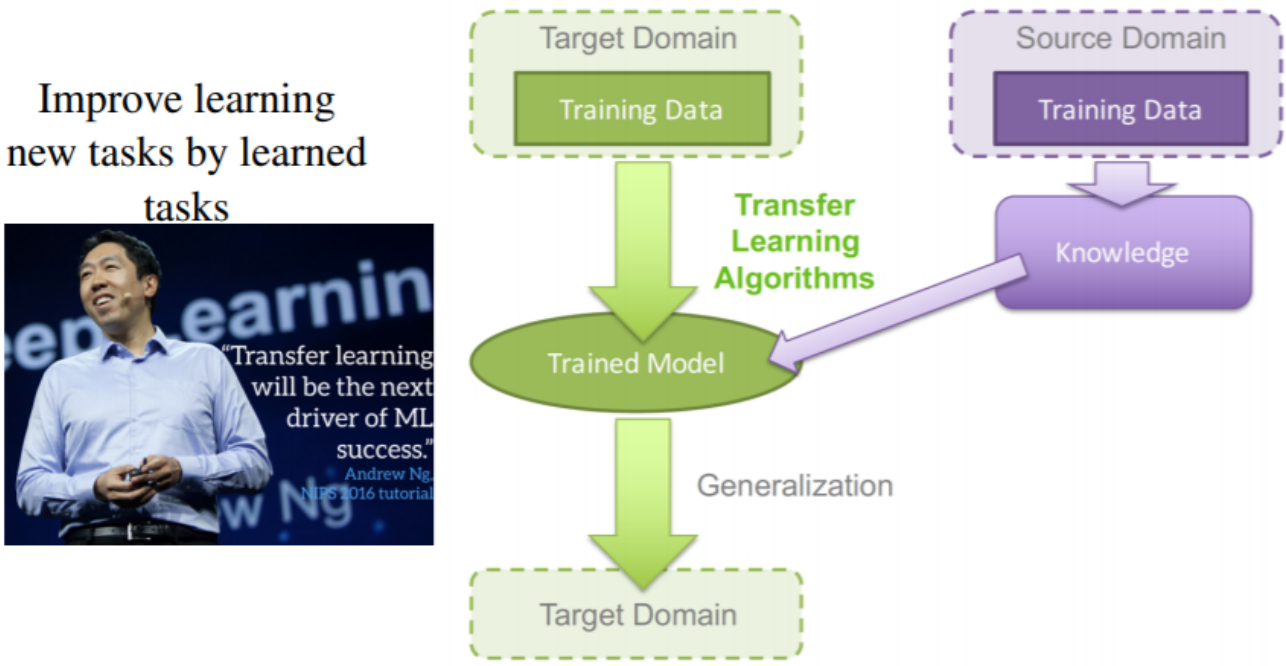
        - If every neuron in the network computes the same output, then all of them will also compute the same gradients during backpropagation and undergo the exact same parameter updates
    - Small random initialization
        - Works okay for small networks, but not for deeper networks
    - Current recommendation for initializing CNNs with ReLU:

```
w = np.random.randn(n) *sqrt(2.0/n)
```

      'randn': Gaussian; 'n': the number of inputs for the current layer
    - Proper initialization is an active area of research
- Learning Rate Decay



    - SGD, Adagrad, RMSProp, Adam all have learning rate as a hyperparameter. Which one of these learning rates is best to use?
    - Answer: All of them! Start with large learning rates and decay over time
    - Reduce learning rate at a few fixed points. E.g. for ResNets, multiply α by 0.1 after epochs 30, 60, and 90.
    - Cosine: $\alpha_t = \frac{1}{2}\alpha_0(1 + cos(t\pi/T))$
    - Linear: $\alpha_t = \alpha_0(1 - t/T)$
- Transfer Learning



- Deep features are fairly transferable, and open-source pre-trained models are now everywhere.

	very similar dataset	very different dataset
very little data	Use linear classifier on top layer	You're in trouble... Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a large number of layers

Lecture 14: Deep Learning -- Applications in Low-Level Computer Vision

Low Level Vision

- Recover the unseen pixels -- image inpainting (修补)

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{e} \tag{19}$$

  - $\mathbf{x}$: Signal to be recovered
  - $\mathbf{e}$: Measurement errors
  - $\mathbf{A}$: Sensing matrix
  - $\mathbf{y}$: Measurements

$$(\mathrm{P0}) \quad \min_{x} \overbrace{\|y - Ax\|_2^2}^{\textbf{Fidelity}}.$$

  - What kind of prior knowledge we can have for this ill-condition task
- Deep Learning for Image Restoration and Enhancement
  - Current trend: end-to-end + image-level
  - Future promise: end-to-end + domain-expertise + self-similarity
  - Three examples
    - Image denoising
    - Image deblurring
    - Image super-resolution
  - More "wild" image restoration examples: compression artifact removal, haze/rain/snow removal, underwater/low-light enhancement etc.

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{e}$$

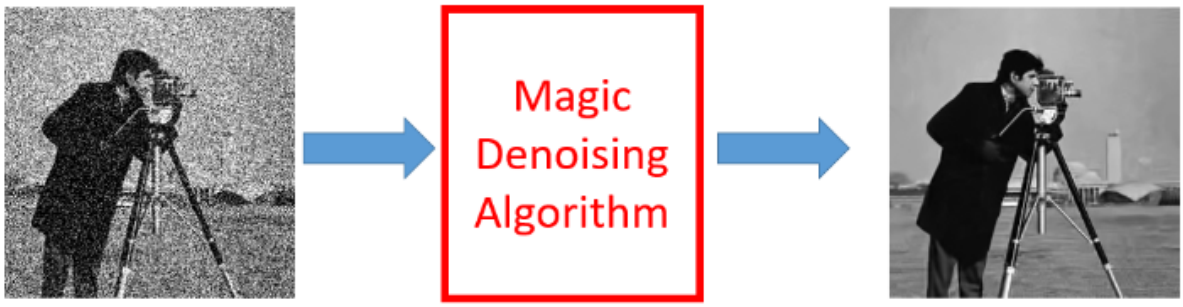
$$(\mathrm{P0}) \quad \min_{x} \overbrace{\|y - Ax\|_2^2}^{\textbf{Fidelity}} + \overbrace{\lambda\Phi(x)}^{\textbf{Signal Regularizer}}$$

- $\mathbf{x}$: Signal to be recovered
- $\mathbf{e}$: Measurement errors
- $\mathbf{A}$: Sensing matrix
- $\mathbf{y}$: Measurements
- $\Phi(\mathbf{x})$: Signal regularizer
  - Sparse coding - dictionary learning
  - Collaborative filtering - BM3D
  - Deep learning - convolutional neural networks

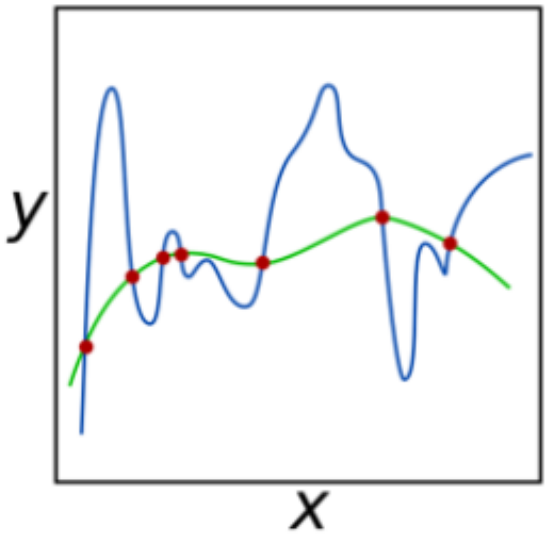
Image Denoising

- Image Denoising
  - Simplest low-level vision problem
  - Noisy measurement: $\mathbf{y} = \mathbf{x} + \mathbf{e}$

  - Estimate the clean image: $\hat{\mathbf{x}} = f(\mathbf{y})$



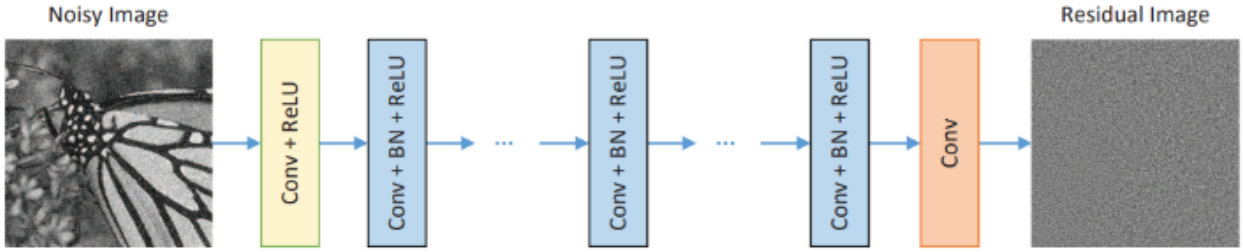
- Conventional(常规) Methods
  - Collaborative (协同) Filtering
    - Non-local means, BM3D, etc.
    - Refs:
      - A. Buades, B. Coll and J. M. Morel, “A non-local algorithm for image denoising,” in CVPR, 2005.
      - K. Dabov, A. Foi, V, Katkovnik, and K. Egiazarian, “Image denoising by sparse 3-D transform-domain collaborative filtering,” IEEE TIP, 2007.
  - Piece-wise smoothing
    - Total variation, Tikhonov regularization, etc.



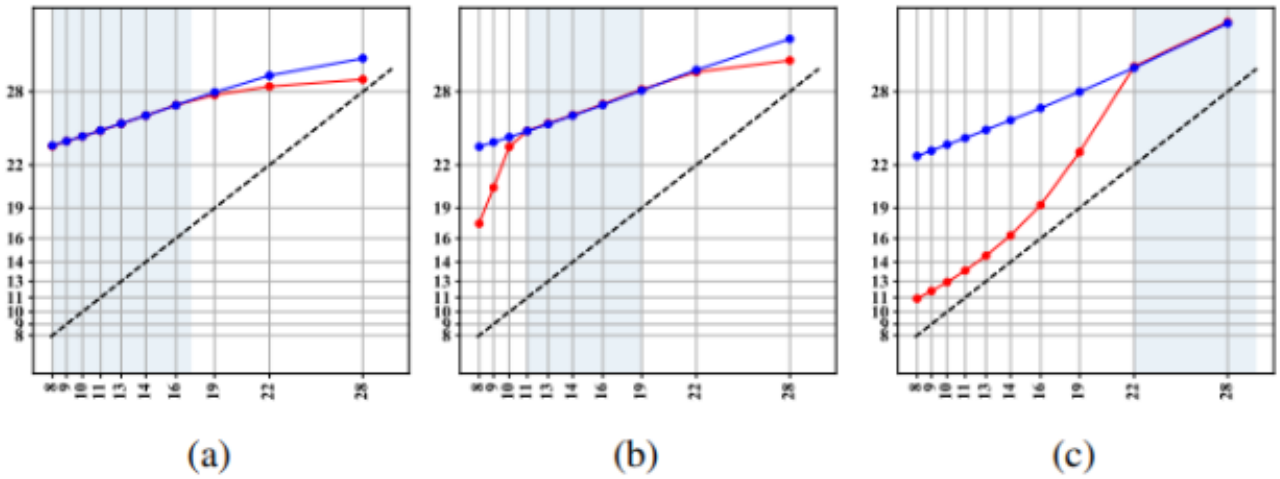
    - Sparsity
        - Discrete cosine transform (DCT), Wavelets, etc.
        - Dictionary learning: OMP, Lasso, K-SVD, etc
        - Refs:
            - J. Portilla, V. Strela, M. J. Wainwright, and E. P. Simoncelli, “Image denoising using scale mixtures of Gaussians in the wavelet domain,”IEEE TIP, 2003.
            - M. Aharon, M. Elad, and A. Bruckstein, “K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation,”IEEE TSP, 2006.
- Image Denoising by Deep Learning
    - Natural idea: train a (convolutional) denoising autoencoder, that regresses clean images from noisy ones
    - It is not easy for deep networks to outperform classical methods such as BM3D
        - BM3D is shown to be better at dealing with self-repeating regular structures
    - How to outperform(在...方面优于...) BM3D using a deep network denoiser?
        - The model capacity is chosen large enough, i.e., enough hidden layers with sufficiently many hidden units
        - The patch (grid) size is chosen large enough, i.e., a patch contains enough information to fit a complicated denoising function that covers the long tail
        - The chosen training set is large enough
- Conventional vs. Deep Learning Methods

	Convenrional	**Deep Learning**
Model	Shallow model (Equivalently one free layer)	Deep model (Multiple free layers)
Supervise	Unsupervised (1. No training corpus (语料库) needed 2. Data efficient)	Supervised (1. Training corpus needed 2. Data inefficient)
Inverse problem	1. Assumption & understanding of the data 2. Regularizers & structures of the model 3. Flexible	1. Little assumption 2. Almost free model 3. Few work until recently

    - Recent work
- Residual Learning (learn the noisy image)



    - MSE is the perfect loss in this setting
    - Much easier to optimize
    - Ref: K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a Gaussian denoiser: Residual learning of deep CNN for imagedenoising,"IEEE TIP, 2017.
- DnCNN with all bias terms removed



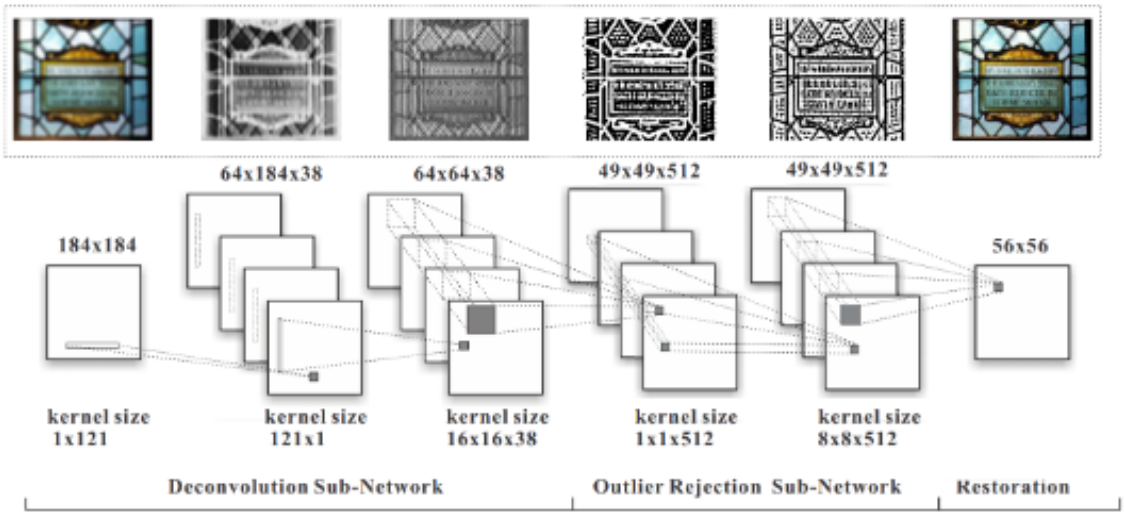
    - Interpretable (可解释的)
    - Robust (强大的)
    - Ref: S. Mohan, Z. Kadkhodaie, E. P. Simoncelli, and C. Fernandez-Granda, "Robust and interpretable blind image denoising viabias-free convolutional neural networks,"arXiv.org e-prints, Technical Report 1906.05478, 2019.
- Why are people interested of denoising
    - Simplest inverse problem with $\mathbf{A} = \mathbf{I}$
    - Ideal test bed to verify the effectiveness of the proposed regularizer

Image Deblurring

- Image Deblurring
    - Blurred Measurement: $\mathbf{y} = \mathbf{M} * \mathbf{x}$

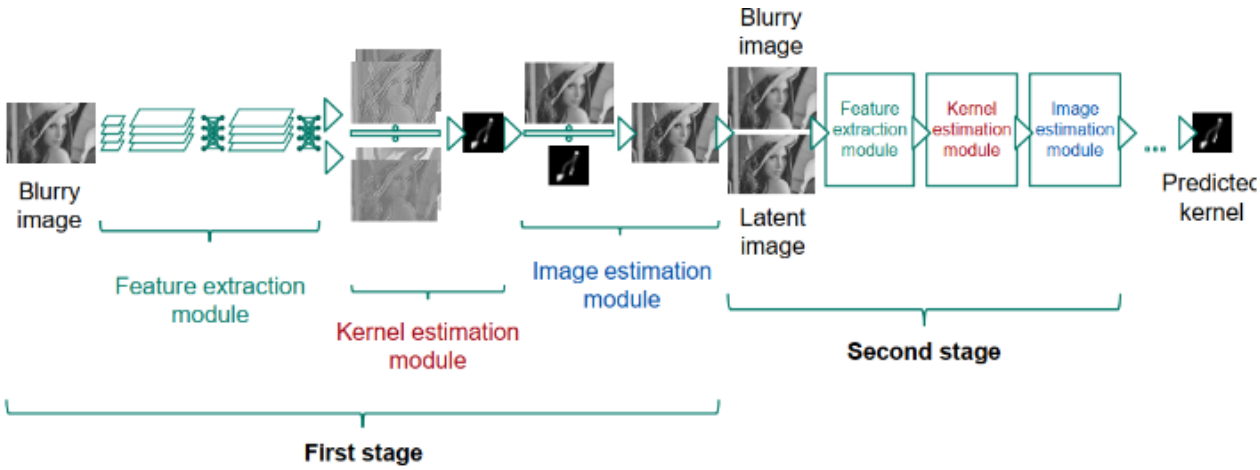
    - Where $\mathbf{M}$ is the convolution kernel
  - Estimate the stable image: $\hat{\mathbf{x}} = f(\mathbf{y})$

  - Non-blind image debluring
    - Suppose the blurring kernel $\mathbf{M}$ is known
    - $\hat{\mathbf{x}} = f(\mathbf{y})$
    - All training data need to have consistent $\mathbf{M}$ as the test data
    - Also difficult because of the compression loss
  - Blind image deblurring - a much more challenging problem
    - Estimate both the image, and the blurring kernel
    - $\{\hat{\mathbf{x}}, \hat{\mathbf{M}}\} = f(\mathbf{y})$
- Image Deblurring by Deep Learning
  - Key technical features:
    - Treat deblurring as a deconvolution task, which can be approximated by a convolution network with larger filter sizes
    - Concatenation of deconvolution CNN module with another denoising CNN module to suppress artifacts and reject outliers (压制伪影并拒绝异常值)



  - Ref: L. Xu, J. Ren, C. Liu, and J. Jia, “Deep convolutional neural network for image deconvolution,” inNIPS, 2014.

- Blind Image Deblurring



  - Key technical features:
    - Iteratively estimate both the blurring kernel and the underlying image
    - Separate networks to capture the image property and the kernel information, respectively
  - Ref: C. J. Schuler, M. Hirsch, S. Harmeling, and B. Scholköpf, “Learning to deblur,”IEEE TPAMI, 2016.

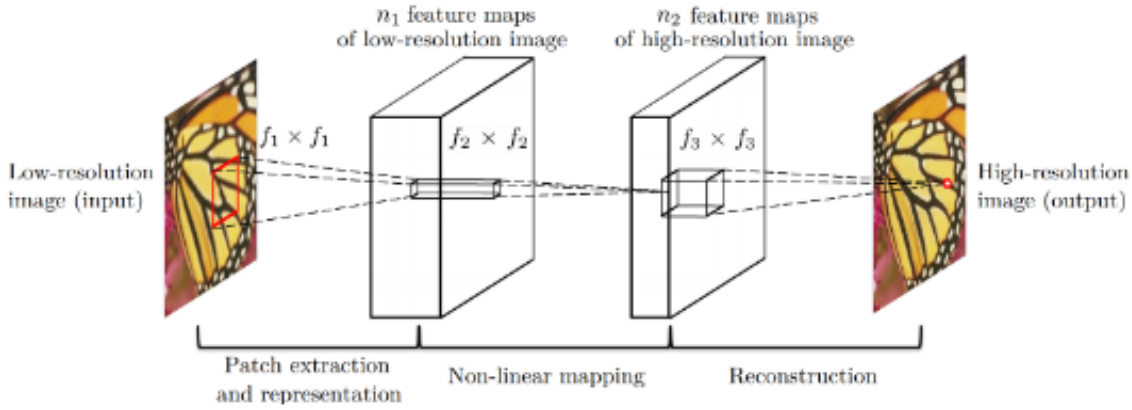
Image Super-Resolution

- Image Super-Resolution
  - Low-resolution measurement: $\mathbf{y} = \mathbf{D} \odot (\mathbf{M} * \mathbf{x})$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \circ \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}\,b_{11} & a_{12}\,b_{12} & a_{13}\,b_{13} \\ a_{21}\,b_{21} & a_{22}\,b_{22} & a_{23}\,b_{23} \\ a_{31}\,b_{31} & a_{32}\,b_{32} & a_{33}\,b_{33} \end{bmatrix}.$$

- Estimate the high-resolution image: $\hat{\mathbf{x}} = f(\mathbf{y})$

- Image Super-Resolution by Deep Learning



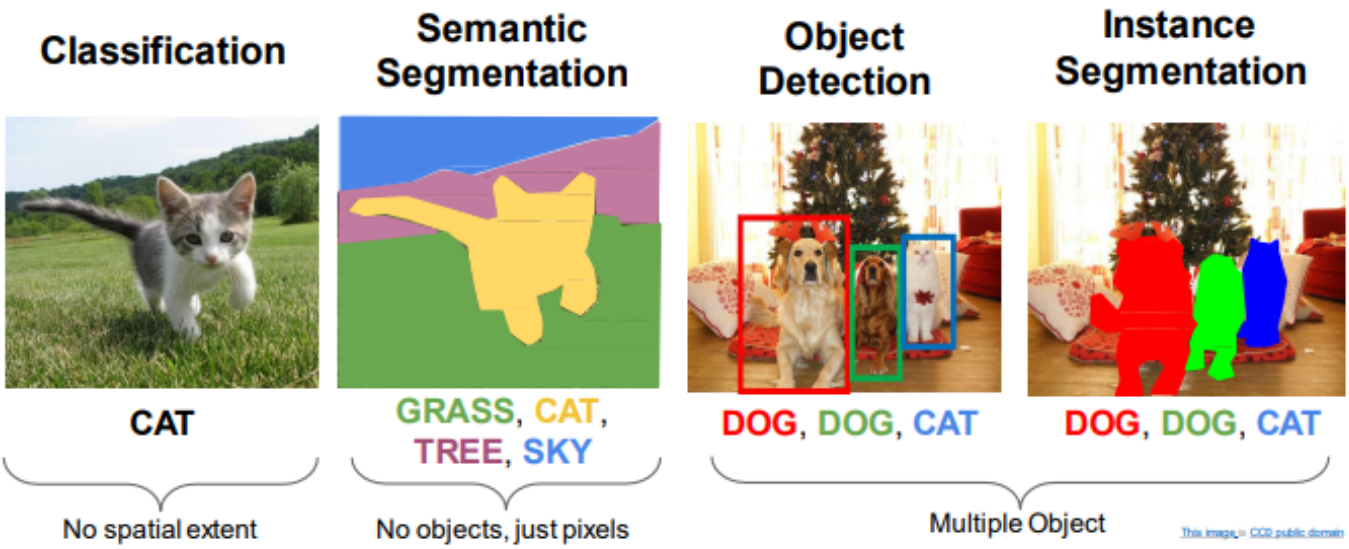
  - Key technical features:
    - Learn an end-to-end mapping from low to high-resolution images as a deep CNN
    - Closely mimic the traditional SR pipeline: LR feature extraction => coupled LR-HR feature space mapping => HR image reconstruction
  - Ref: C. Dong, C. C. Loy, K. He, and X. Tang, “Image super-resolution using deep convolutional networks,”IEEE TPAMI, 2015.

New Trends

- New topic: dehazing, deraining, low-light enhancement, etc.
- New goal: Human perception v.s. machine perception (知觉)
- New setting: From supervised to unsupervised training (no“GT”) or relying on “synthetic pairs”
- New domain: Medical images, infrared images, remote sensing images, etc.
- New concern: “All-in-one” adaptivity, efficient implementation, etc.

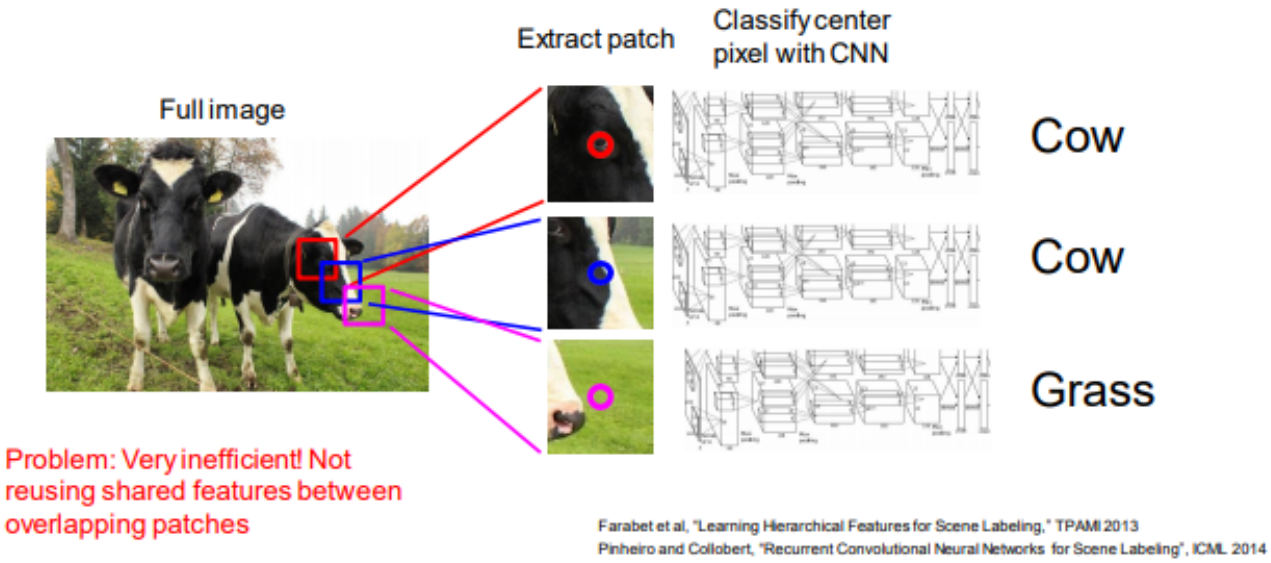
Lecture 15: Deep Learning -- Applications in High-Level Computer Vision

Tasks

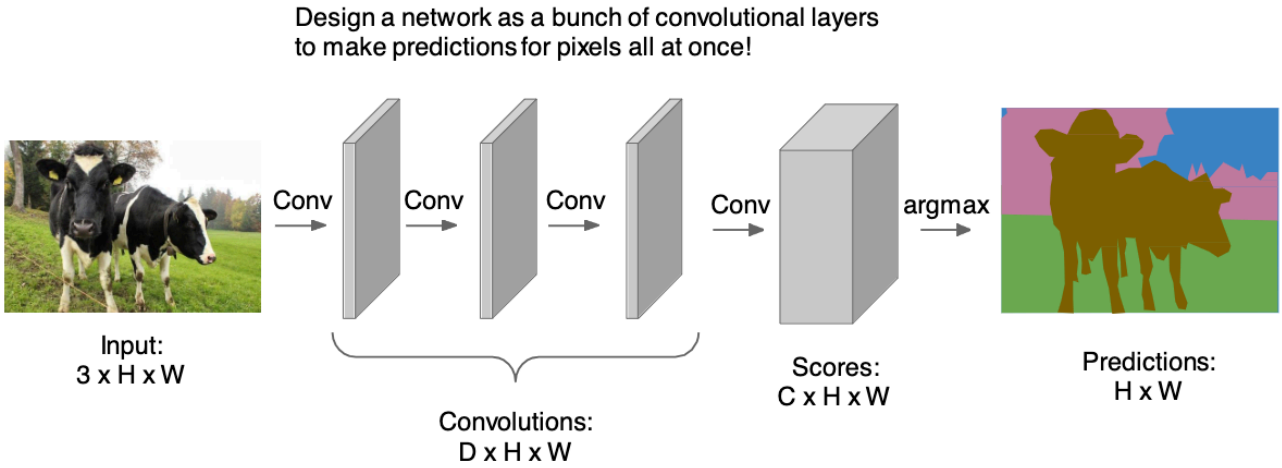


Semantic Segmentation

- Task:
  - Label each pixel in the image with a category label
  - Don't differentiate instances, only care about pixels
- Sliding Window

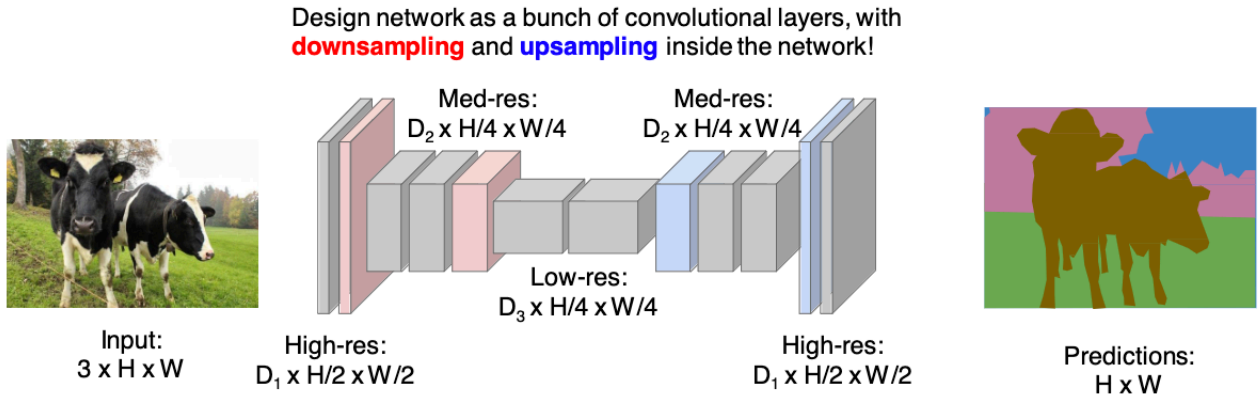


- Fully Convolution



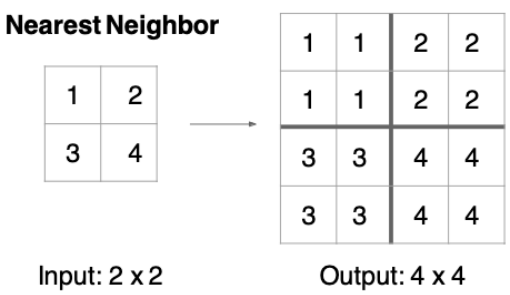
- Problem: Convolutins at orignal image resolution will be very expensive

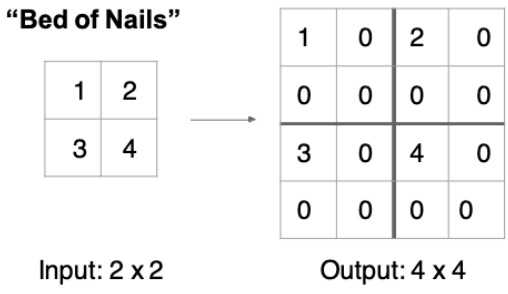
- Fully Convolution with downsampling and upsampling



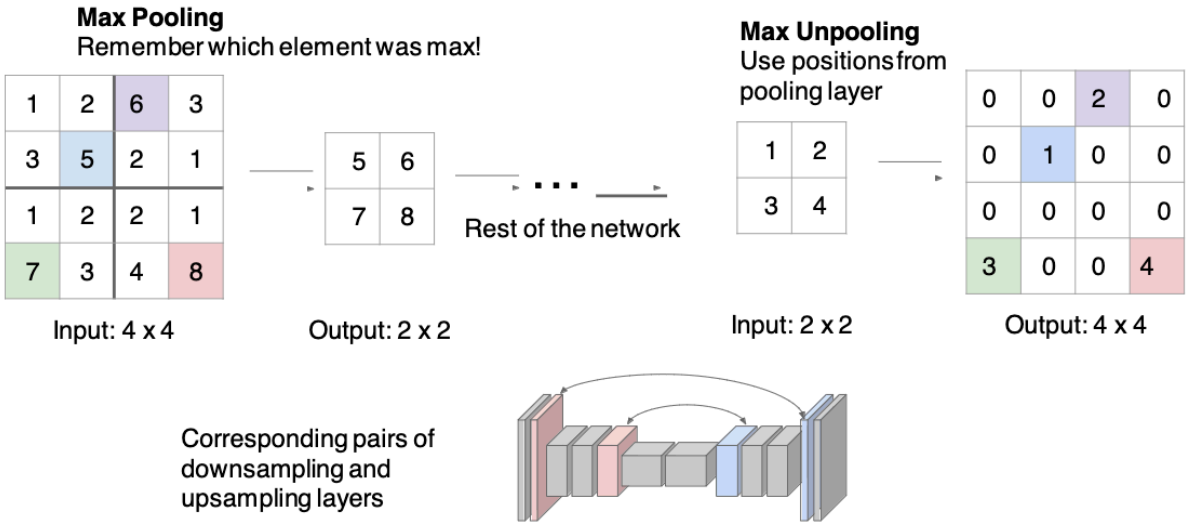
  - $CNN + CNN^{-1}$
  - Downsampling: Pooling, strided convolution
  - Upsampling: **Unpooling or striped transpose convolution**
  - Refs: Long, Shelhamer, and Darrell, “Fully Convolutional Networks for Semantic Segmentation”, CVPR2015Noh et al, “Learning Deconvolution Network for Semantic Segmentation”, ICCV2015

- Upsampling
  - Unpooling

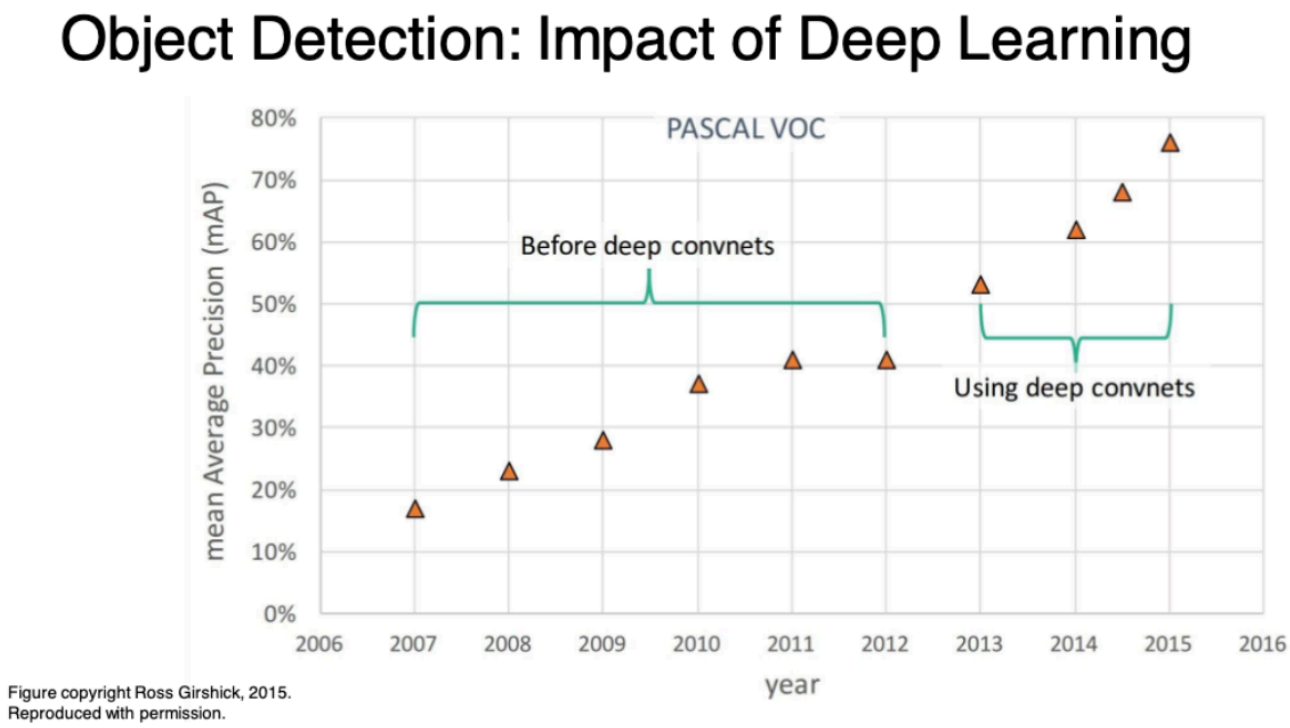




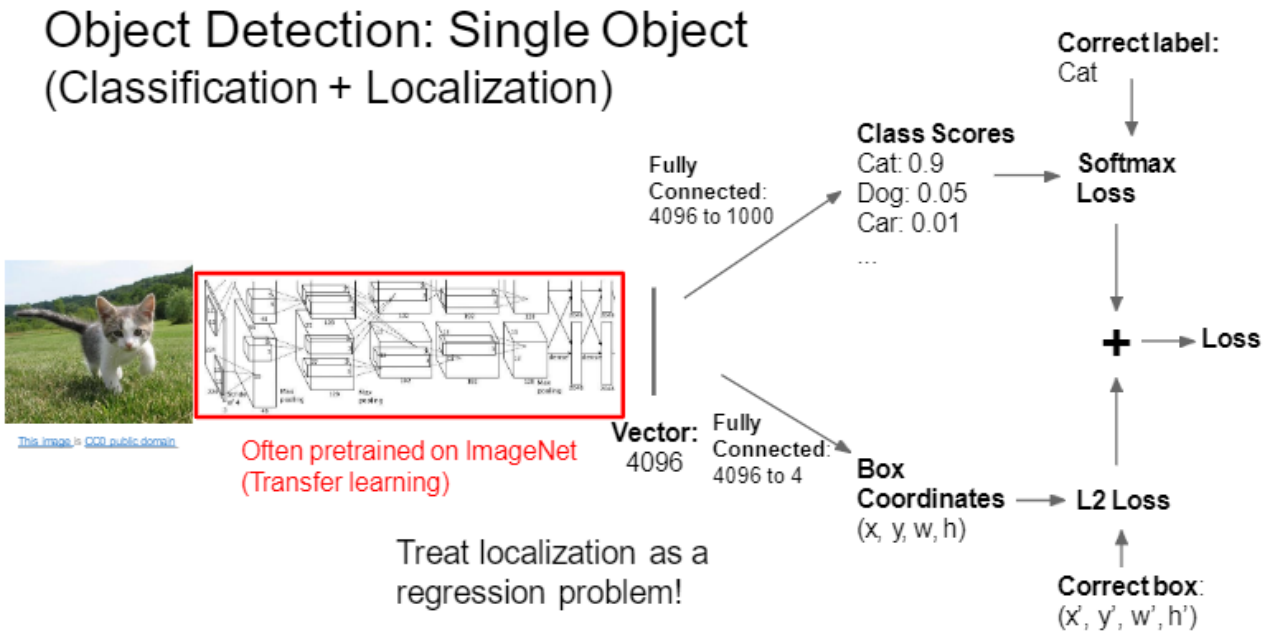
  - Max Unpooling
    - Remember the position of copying the value in the max pooling layer, and just copy back to the position and fill zero



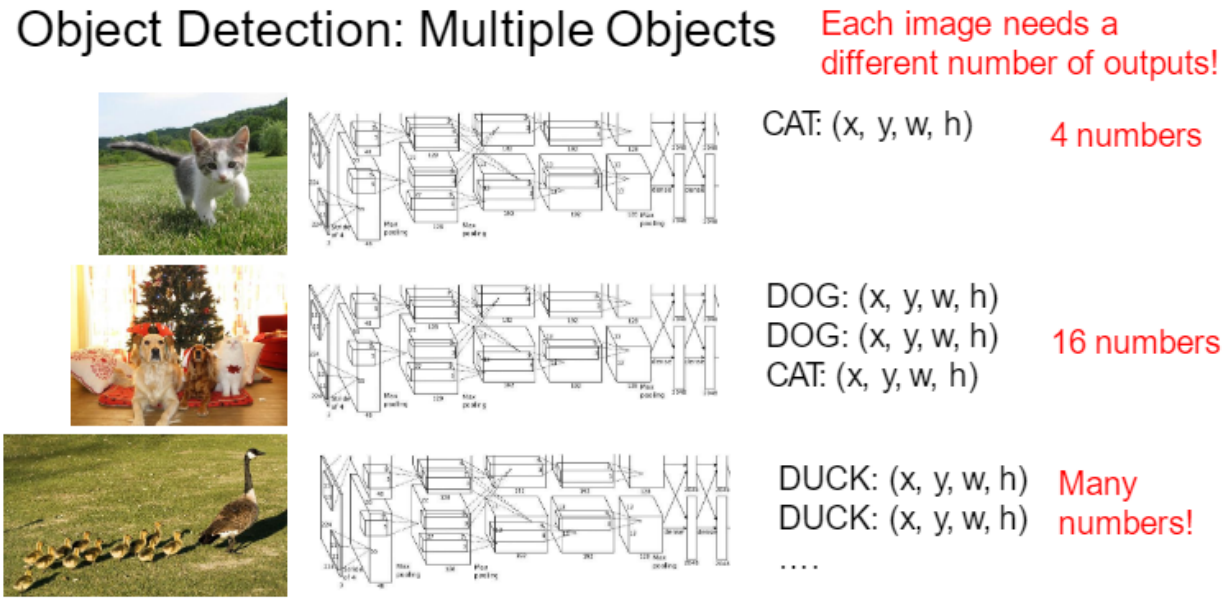
Object Detection



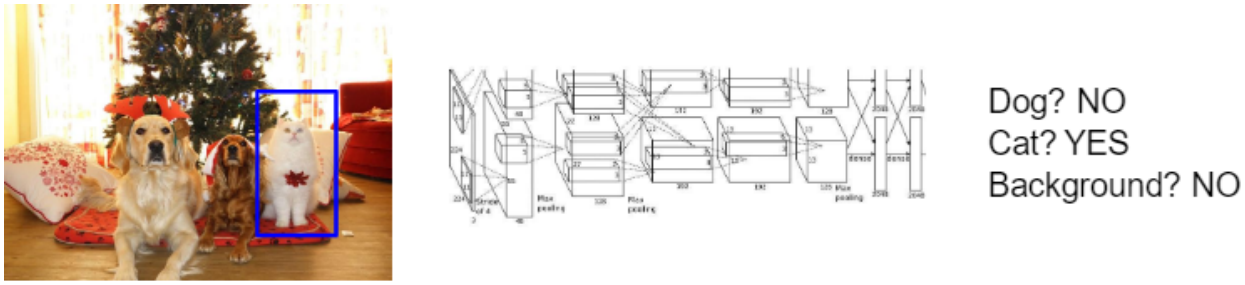
- Single Object Detection => Classification + Localization
  - Use multitask loss



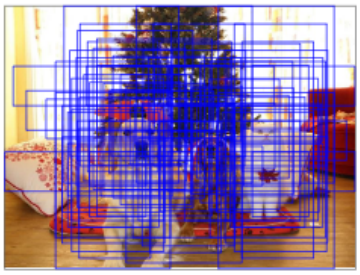
- Multiple Objects Detection => Expensive calculation

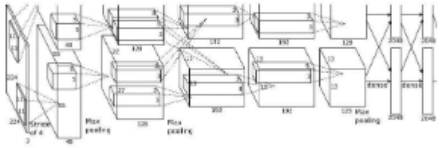


  - Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



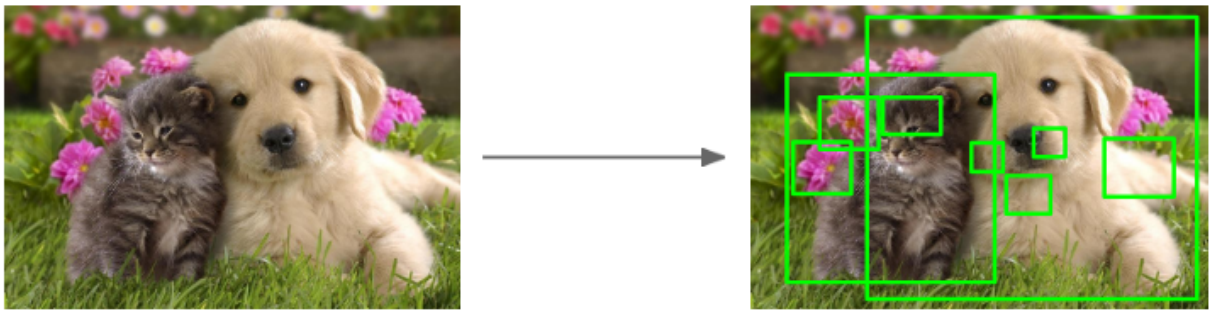
    - Problem: Need to apply CNN to huge number of locations, scales, and aspect ratios, very computationally expensive!



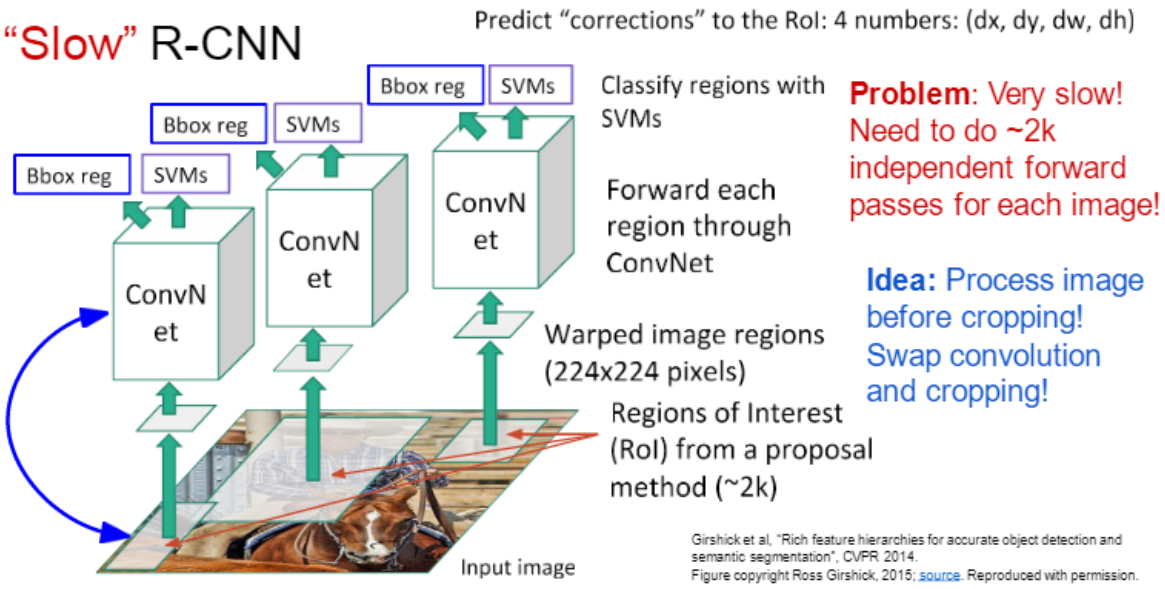


Dog? NO
Cat? YES
Background? NO

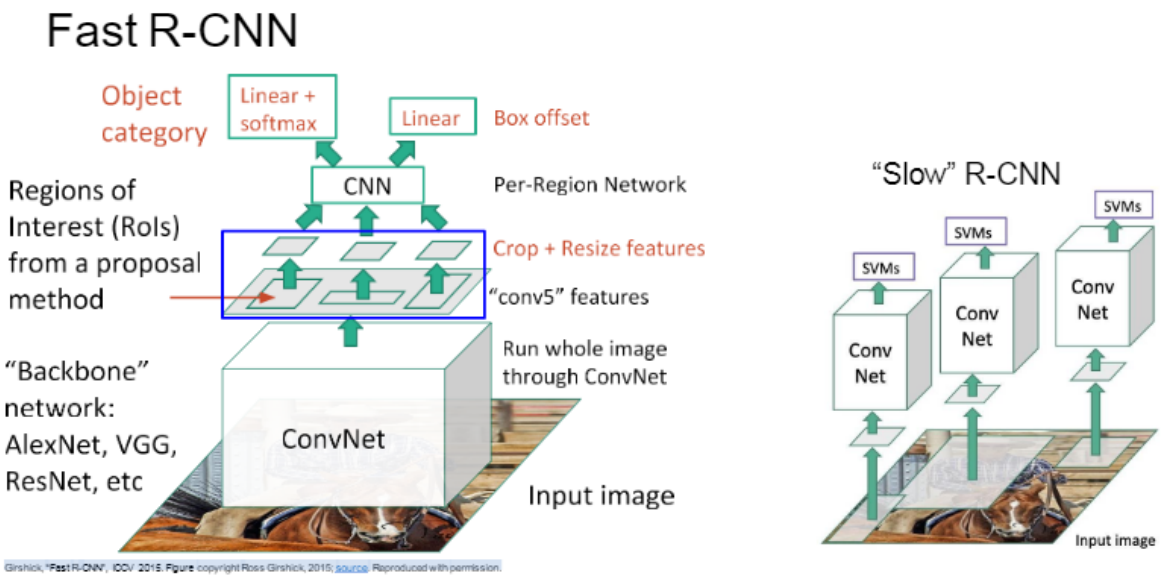
- Region Proposals: Selective Search



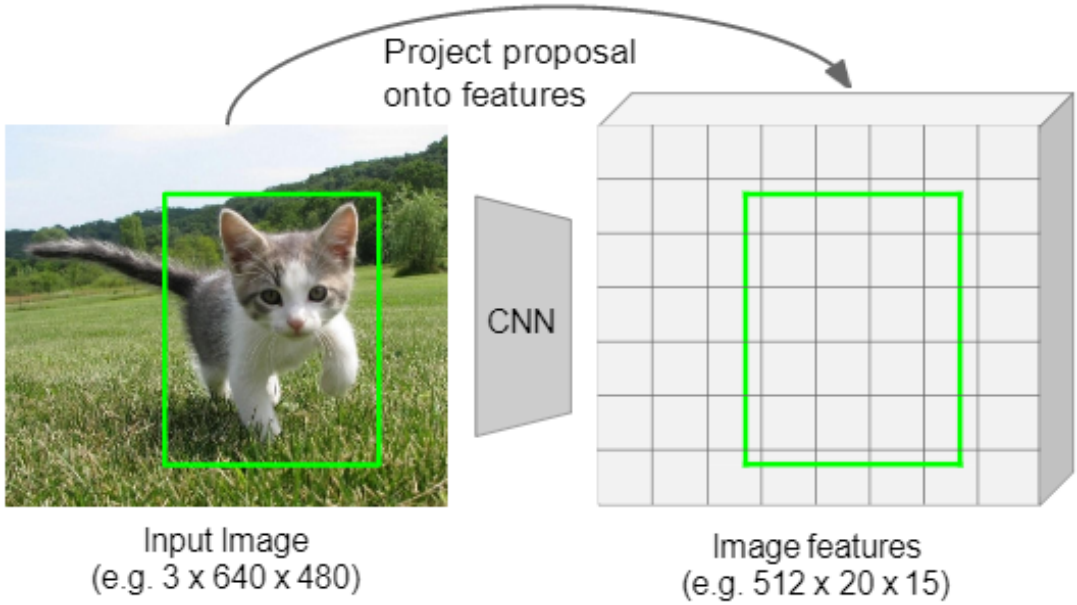
  - Find "blobby" that are likely to contain objects
  - Relatively fast to run; e.g. Selective Search gives 2000 region proposals in a few seconds on CPU
  - Ref: Alexe et al, “Measuring the objectness of image windows”, TPAMI 2012 Uijlings et al, “Selective Search for Object Recognition”, IJCV2013Cheng et al, “BING: Binarized normed gradients for objectness estimation at 300fps”, CVPR 2014 Zitnick and Dollar, “Edge boxes: Locating object proposals from edges”, ECCV2014
- R-CNN
  - "Slow" R-CNN



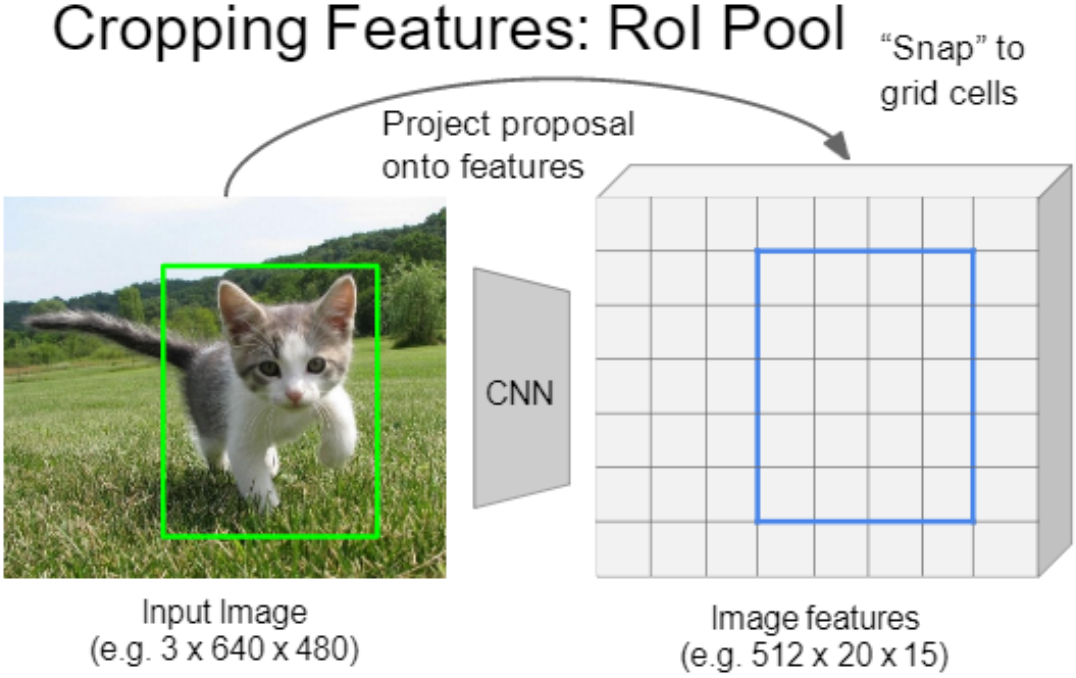
    - Ref: Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR2014.Figure copyright Ross Girshick, 2015; source. Reproduced with permission.
  - Fast R-CNN



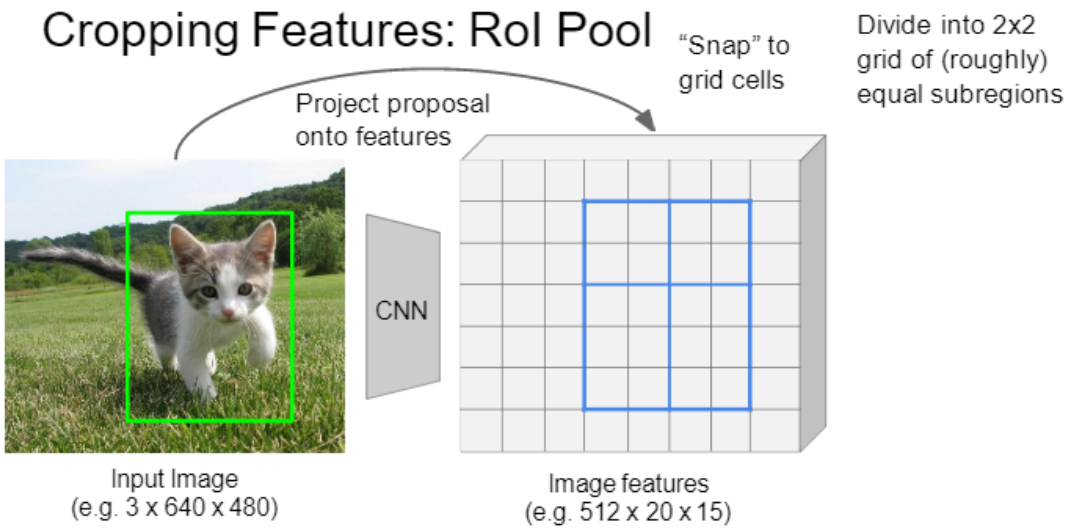
- Feature cropping methods
  - RoI Pool



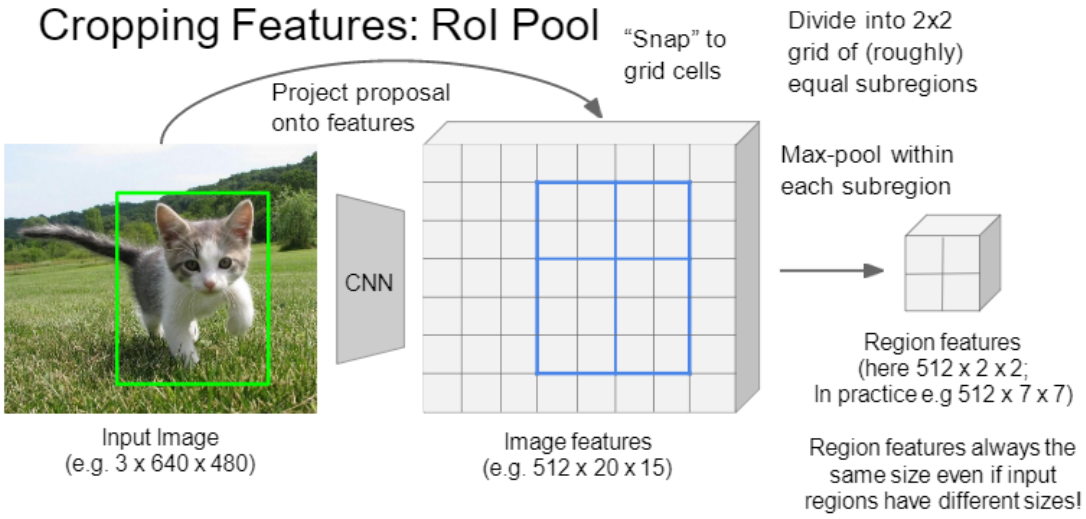
Cropping Features: RoI Pool



Cropping Features: RoI Pool

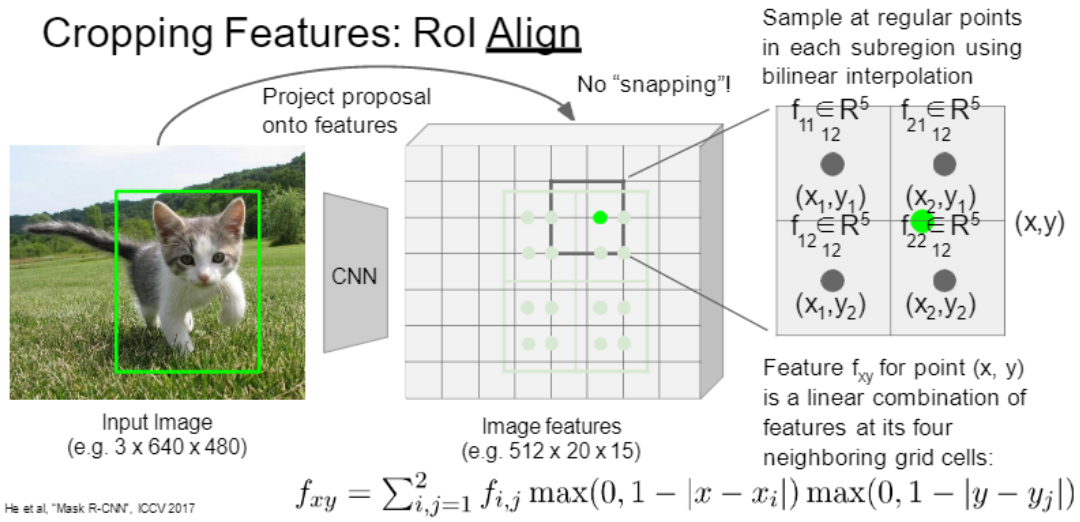


Cropping Features: RoI Pool

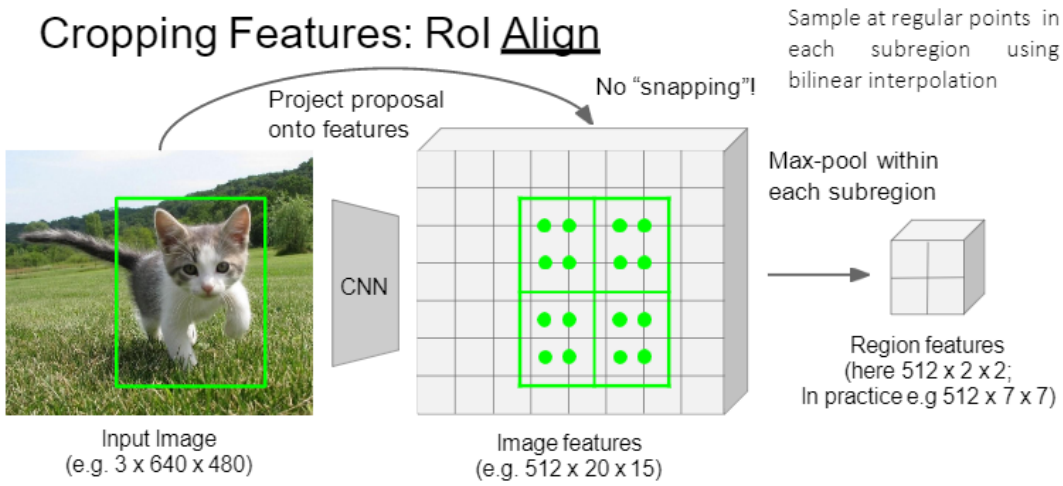


Girshick, "Fast R-CNN", ICCV 2015.

- Problem: Region features slightly misaligned (due to snapping)
- RoI Align

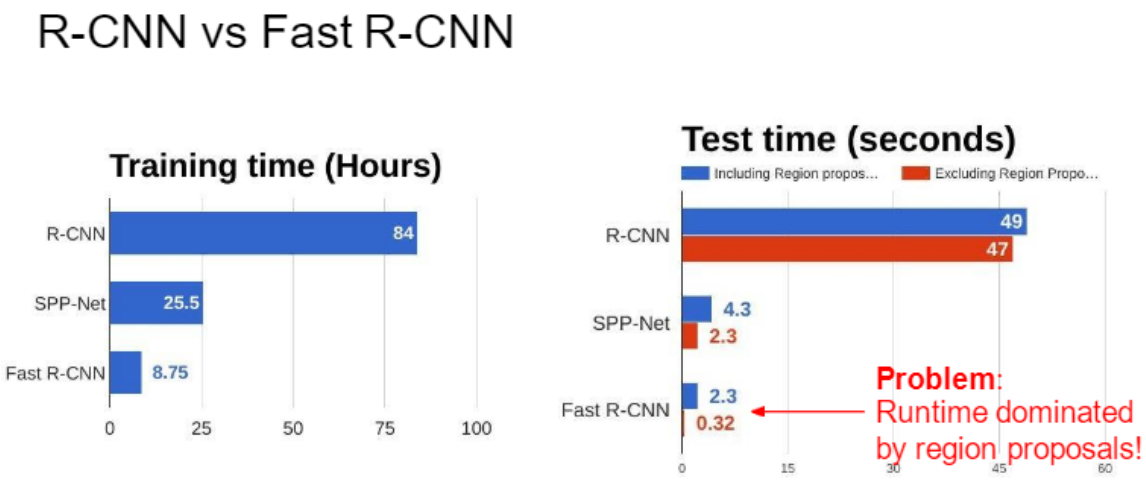


- If the two-axis-distance is less than 1, add the weighted score

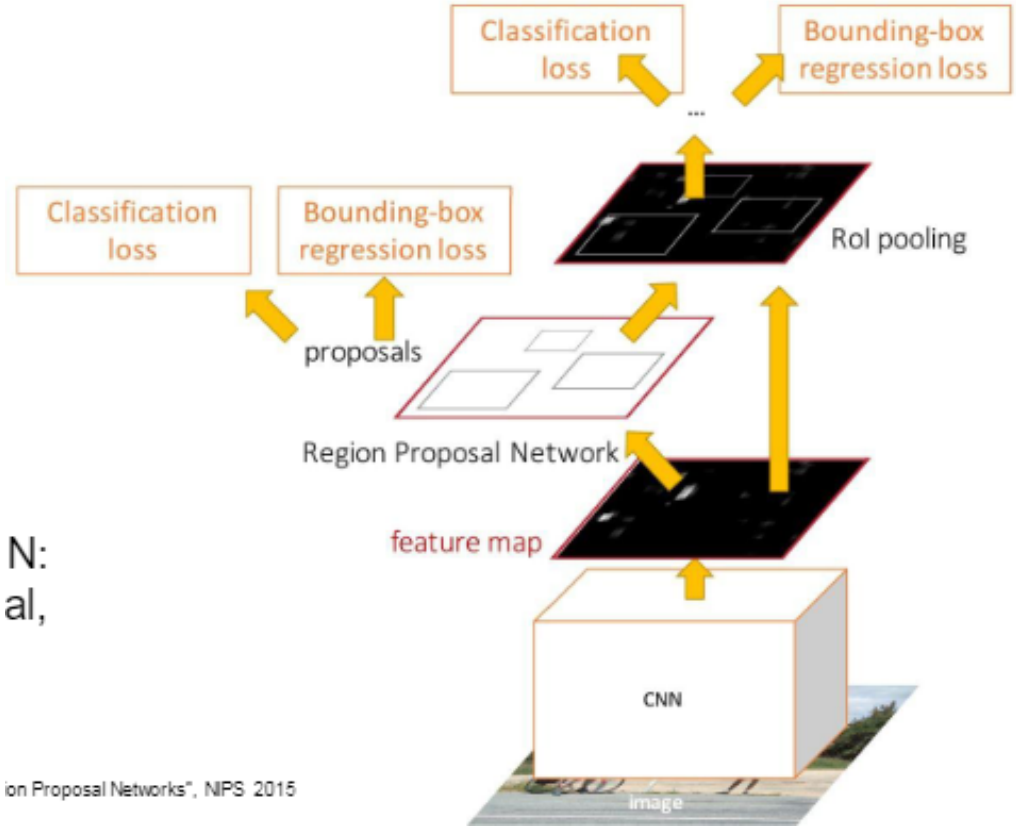


- Ref: Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced w ithpermission.

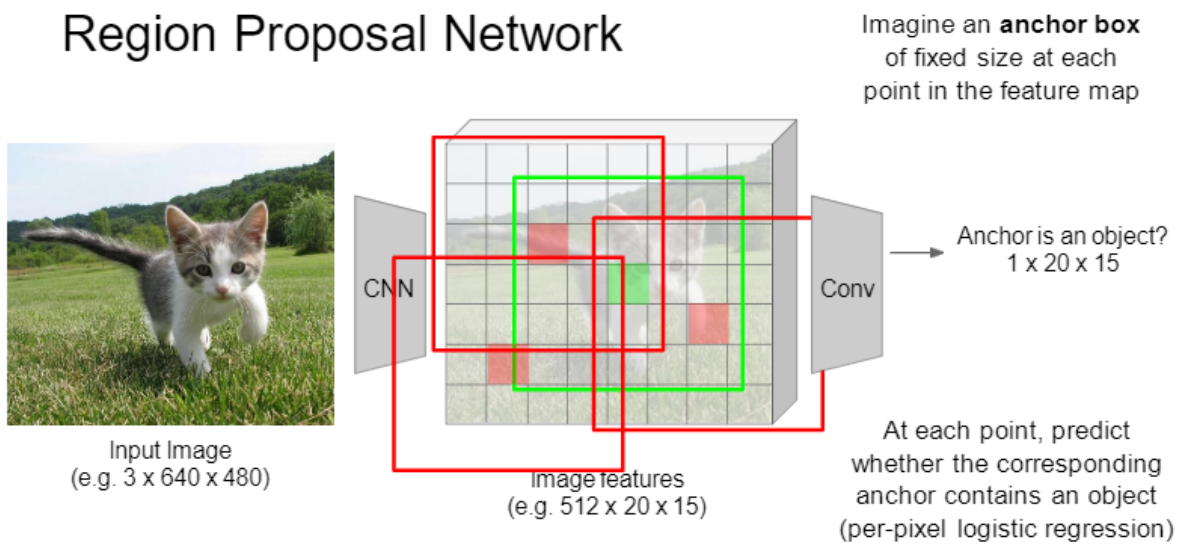
- R-CNN vs. Fast R-CNN

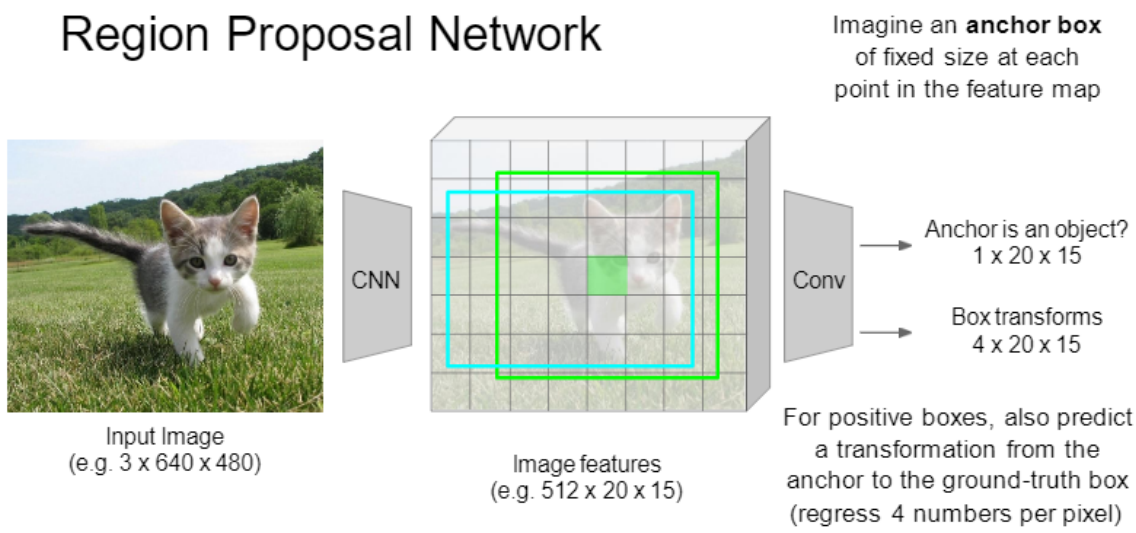


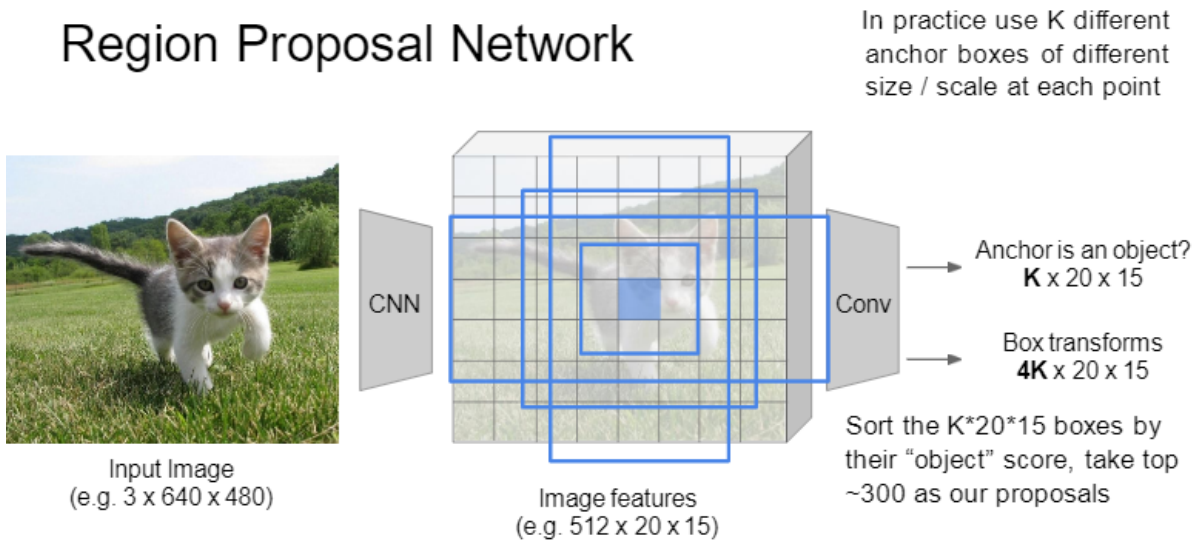
- Faster R-CNN
  - Idea: Make CNN do proposals
  - Method: Insert Region Proposal Network (RPN) to predict proposals from features. Otherwise same as Fast R-CNN: Crop features for each proposal, classify each one. Jointly train 4 losses:
    - RPN classify object / not object
    - RPN regress box coordinates
    - Final classification score (object classes)
    - Final box coordinates



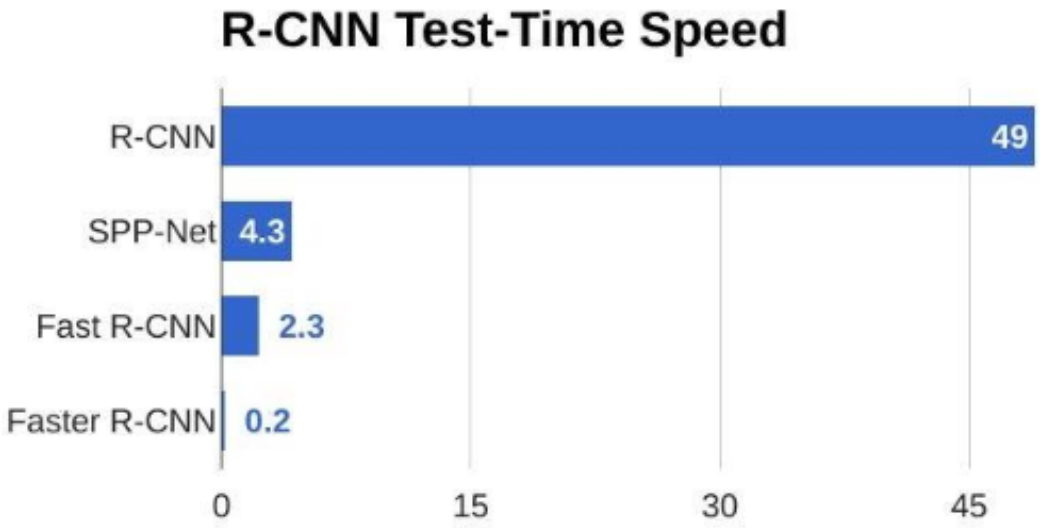
  - RPN





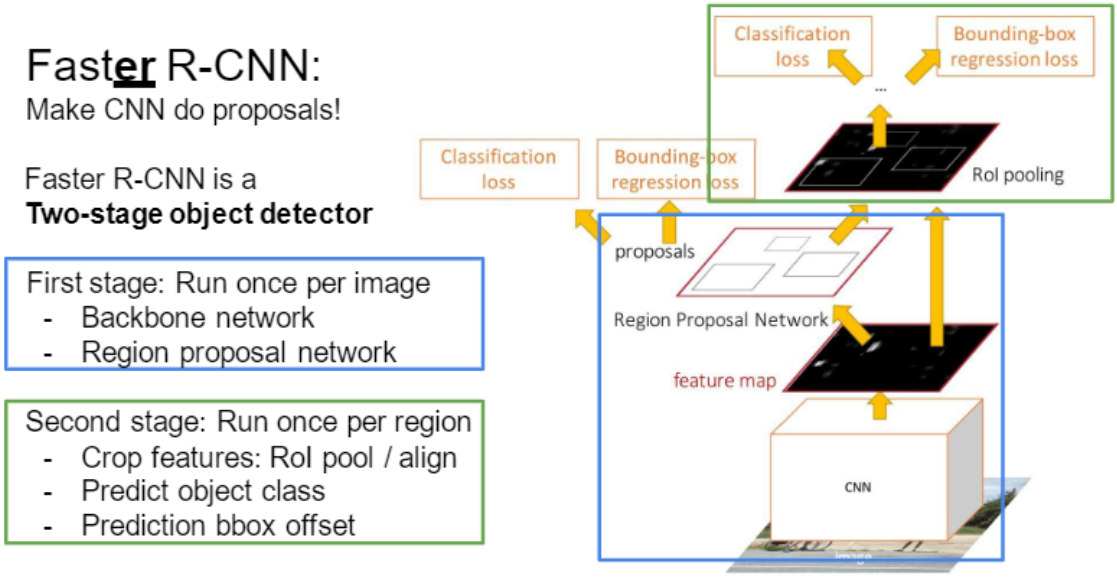


- Performance

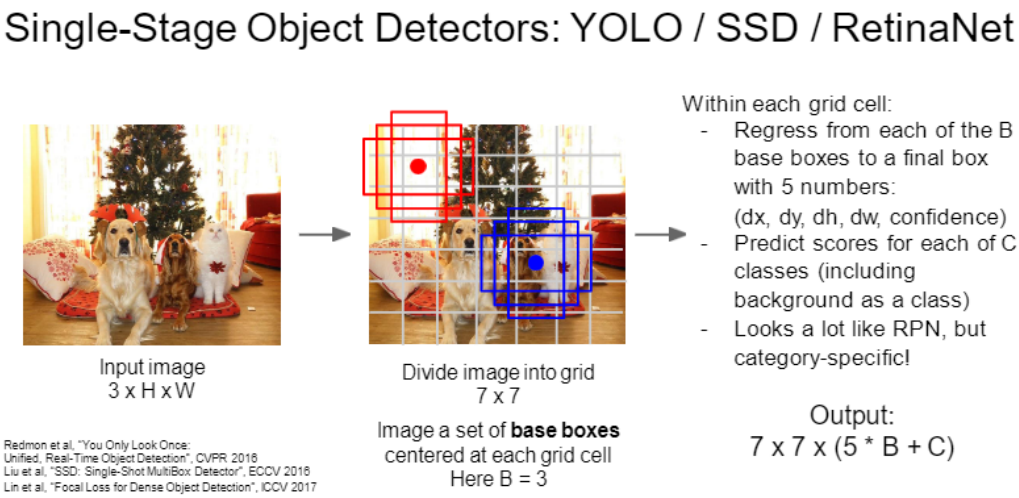


- Glossing over many details

- Ignore overlapping proposals with non-max suppression
- How to determine whether a proposal is positive or negative?
- How many positives/negatives to send to second stage?



- Do we really need second stage?



  - Ref: Redmon et al, “You Only Look Once: Unified, Real-Time Object Detection”, CVPR2016 Liu et al, “SSD: Single-Shot MultiBox Detector”, ECCV 2016 Lin et al, “Focal Loss for Dense Object Detection”, ICCV201
- How to parameterize bounding box regression?
- Ref: Ren et al, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, NIPS 2015 Figure copyright 2015, Ross Girshick; reproduced with permission

- Comparation

Object Detection: Lots of variables ...

Backbone Network
VGG16
ResNet-101
Inception V2
Inception V3
Inception ResNet
MobileNet

"Meta-Architecture"
Two-stage: Faster R-CNN
Single-stage: YOLO / SSD
Hybrid: R-FCN

Image Size
 # Region Proposals
...

Takeaways
Faster R-CNN is slower but more accurate

SSD is much faster but not as accurate

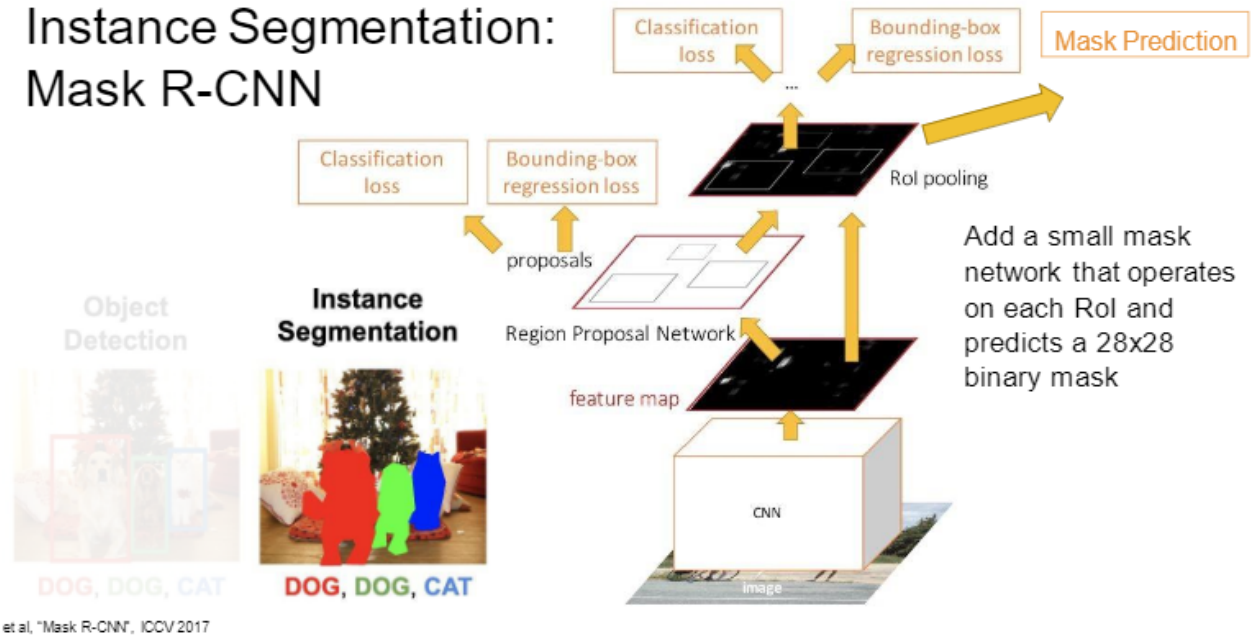
Bigger / Deeper backbones work better

Huang et al, "Speed/accuracy trade-offs for modern convolutional object detectors", CVPR 2017
Zou et al, "Object Detection in 20 Years: A Survey", arXiv 2019 (today!)

R-FCN: Dai et al, "R-FCN: Object Detection via Region-based Fully Convolutional Networks", NIPS 2016
Inception-V2: Ioffe and Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", ICML 2015
Inception V3: Szegedy et al, "Rethinking the Inception Architecture for Computer Vision", arXiv 2016
Inception ResNet: Szegedy et al, "Inception-V4, Inception-ResNet and the Impact of Residual Connections on Learning", arXiv 2016
MobileNet: Howard et al, "Efficient Convolutional Neural Networks for Mobile Vision Applications", arXiv 2017

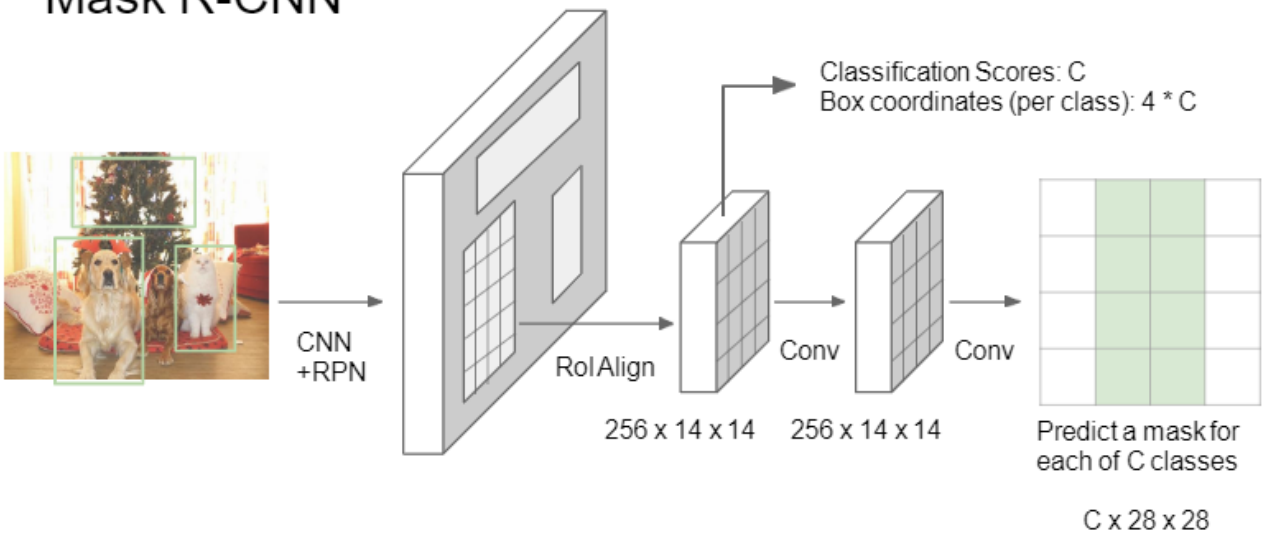
Instance Segmentation

- Mask R-CNN



He et al, "Mask R-CNN", ICCV 2017

Mask R-CNN



He et al, "Mask R-CNN", arXiv 2017