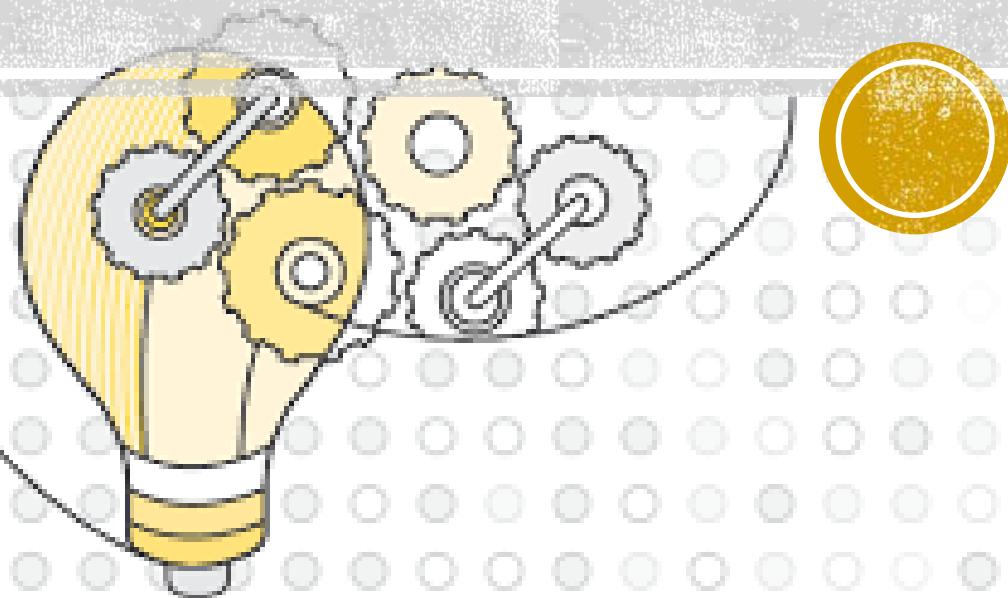


INTRODUCTION



- Class webpage:
http://people.sutd.edu.sg/~nengli_lim/teaching/
- Textbook:
Pattern Recognition and Machine Learning, Bishop 2006
- Grading:
 - HW: 50% ($5 \times 10\%$), Midterm: 20%, Final: 30%
 - Participation bonus/penalty: -2% to 2% of final score

ACKNOWLEDGEMENTS

- MIT 6.036 Introduction to Machine Learning
- SUTD 50.007 Machine Learning (Alex Binder)
- Stanford CS229 Machine Learning



WHAT IS MACHINE LEARNING?

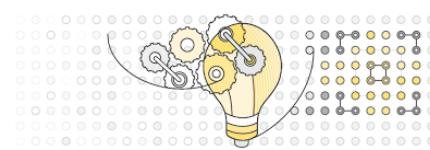


Hard-Coded



Trained

Giving computers the ability to learn
without being explicitly programmed
– Arthur Samuel (1959)



TYPES OF MACHINE LEARNING



Supervised Learning



TYPES OF MACHINE LEARNING



Unsupervised Learning



TYPES OF MACHINE LEARNING

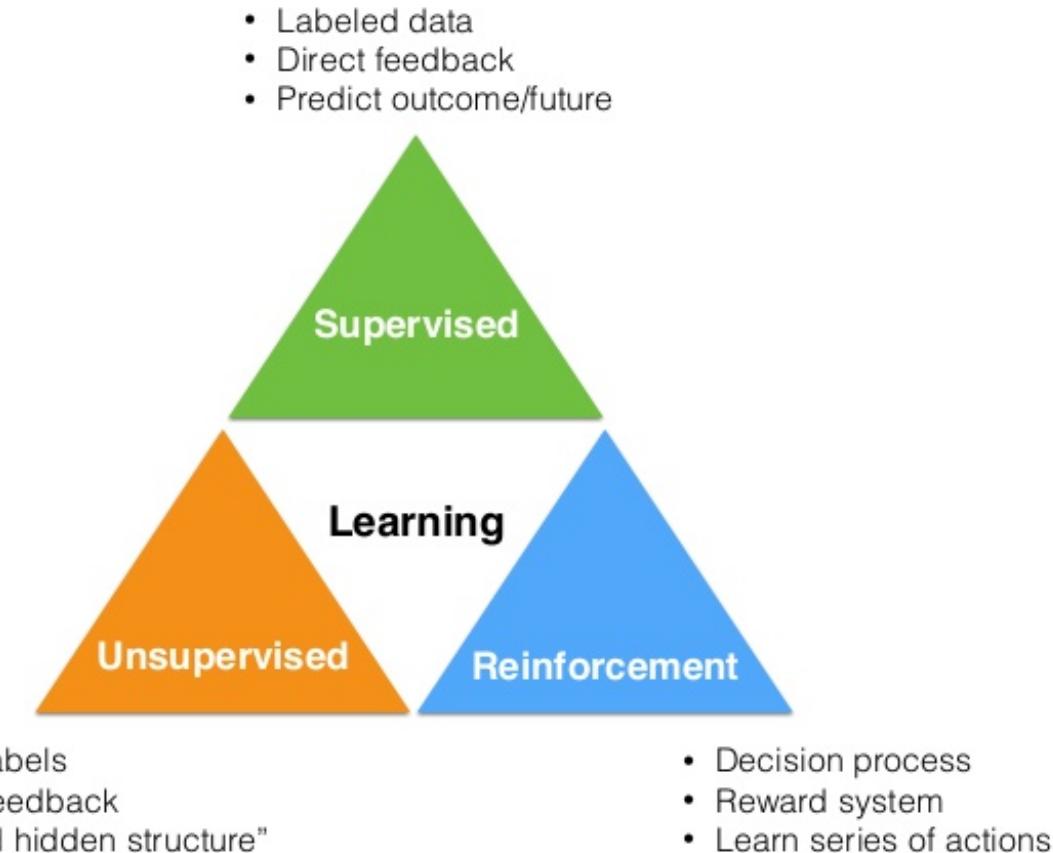


Playing is more
fun than watching!

Reinforcement Learning

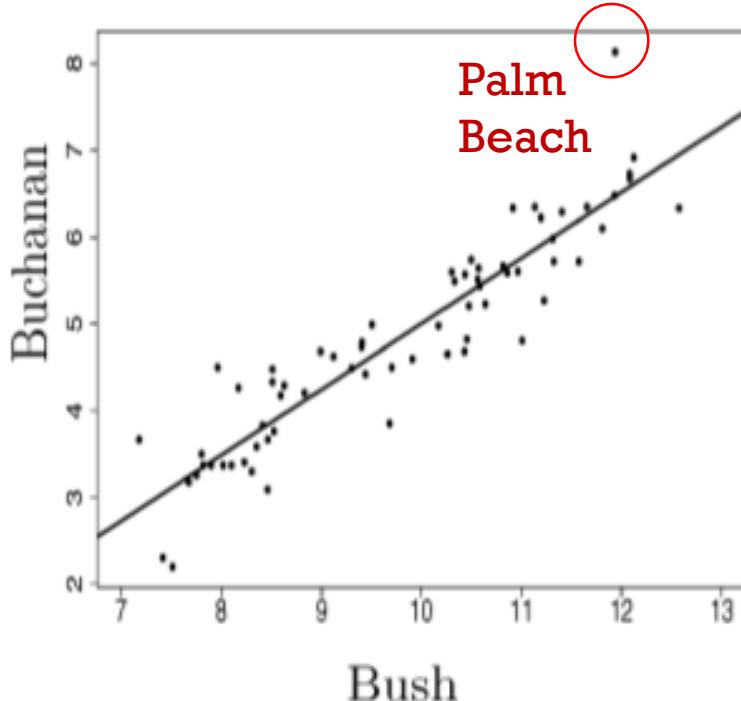


Machine Learning



SUPERVISED LEARNING

Regression (Linear)



Learning a function

$$y = f(x)$$

$$x \in \mathbb{R}$$

$$y \in \mathbb{R}$$

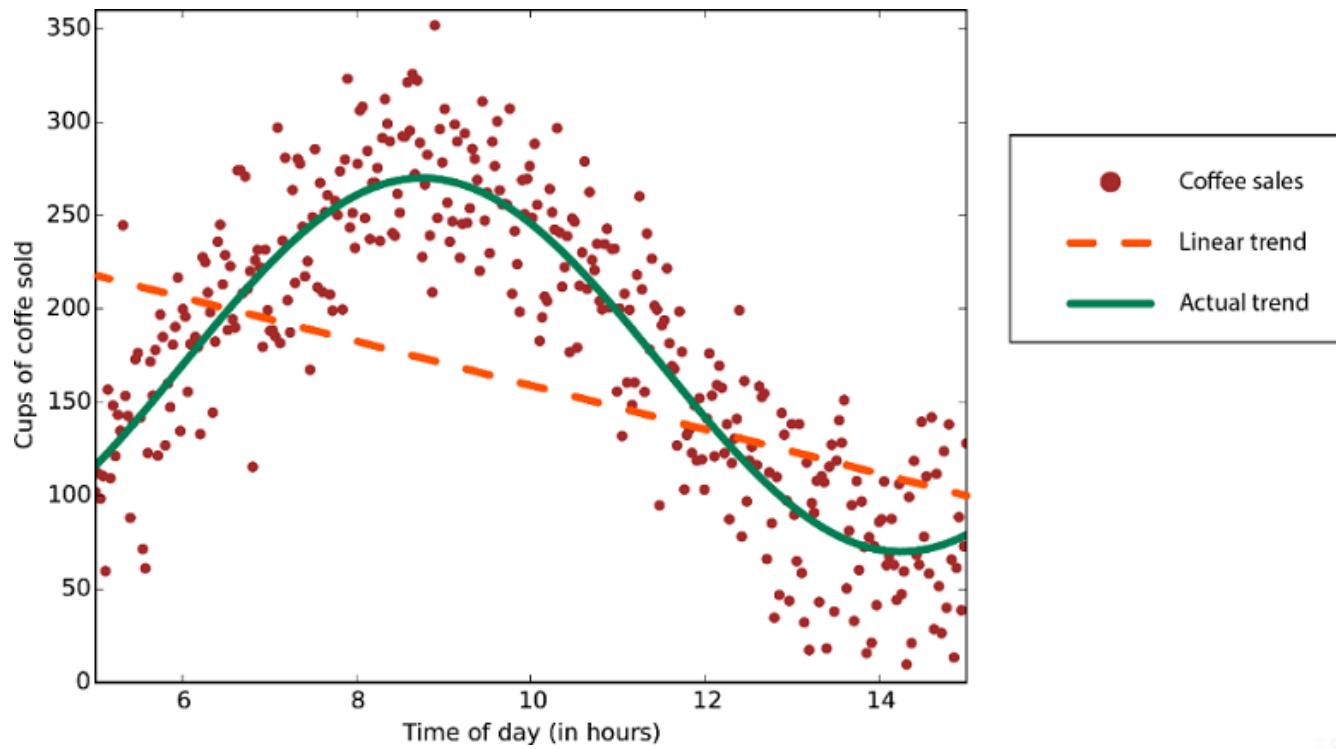
2000 USA Presidential Elections.

Votes for Buchanan and Bush in cities of Florida on a log scale.



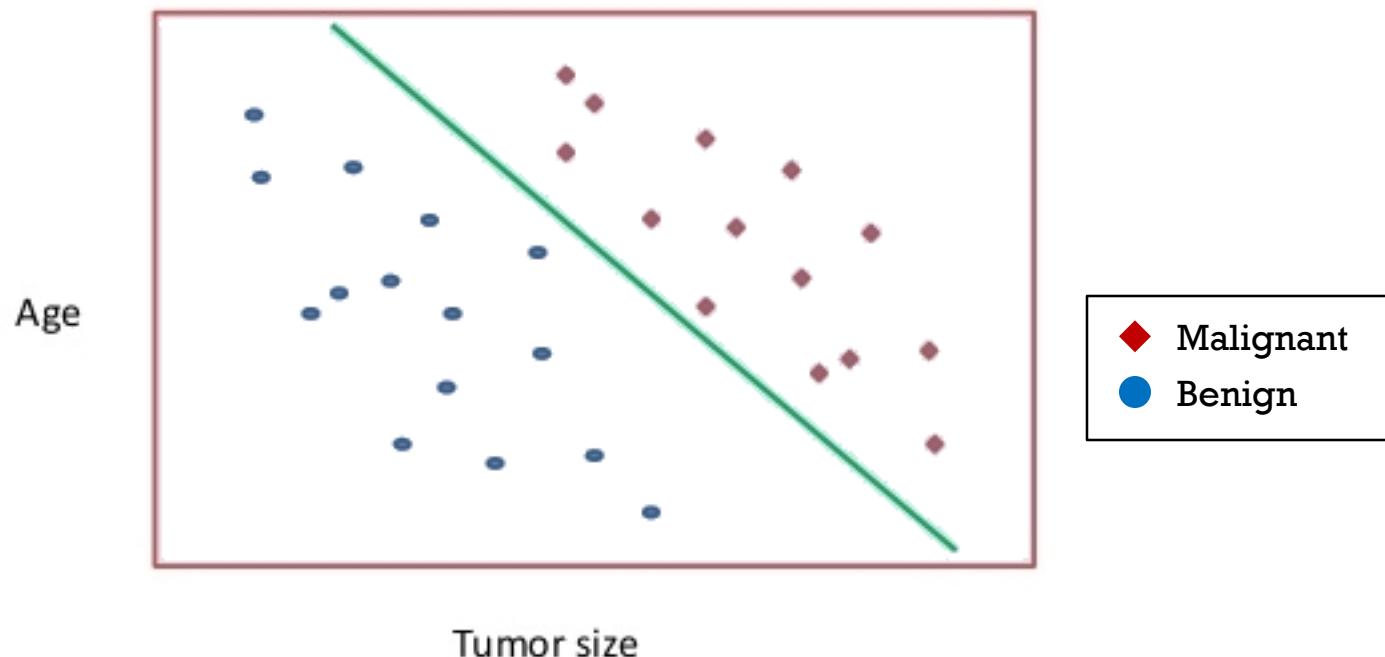
SUPERVISED LEARNING

Regression (Non-linear)



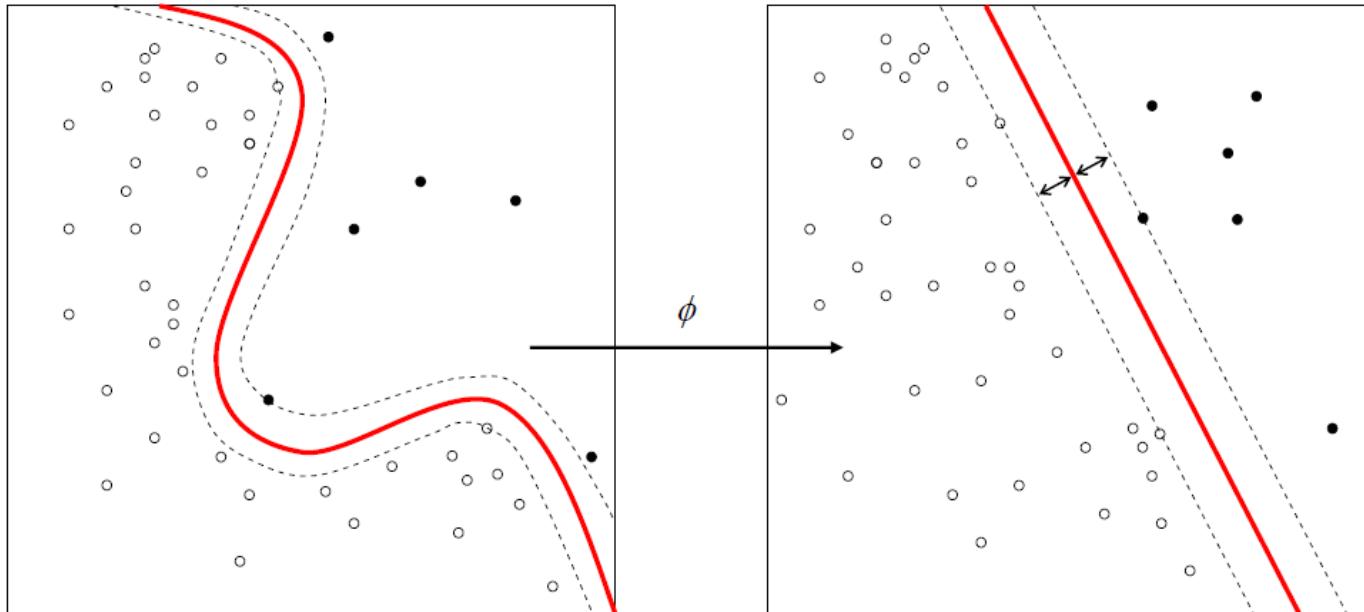
SUPERVISED LEARNING

Classification (Linear)



SUPERVISED LEARNING

Classification (Non-linear)

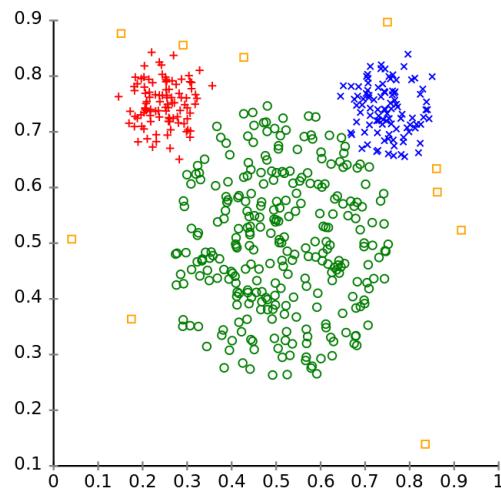


UNSUPERVISED LEARNING

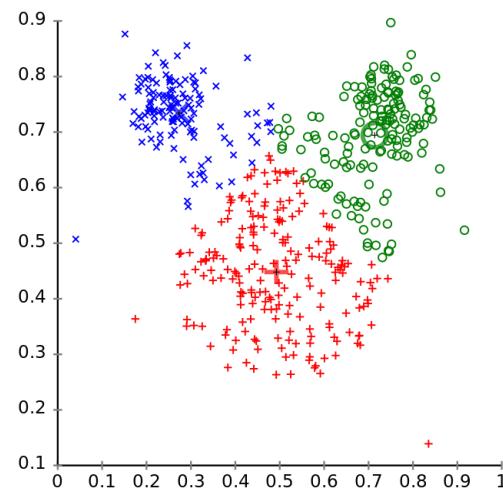
Clustering

Different cluster analysis results on "mouse" data set:

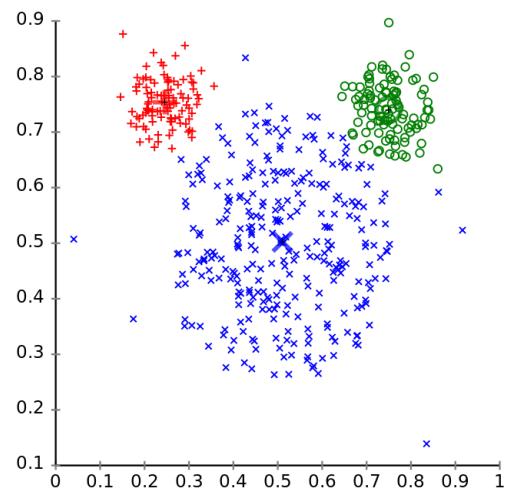
Original Data



k-Means Clustering

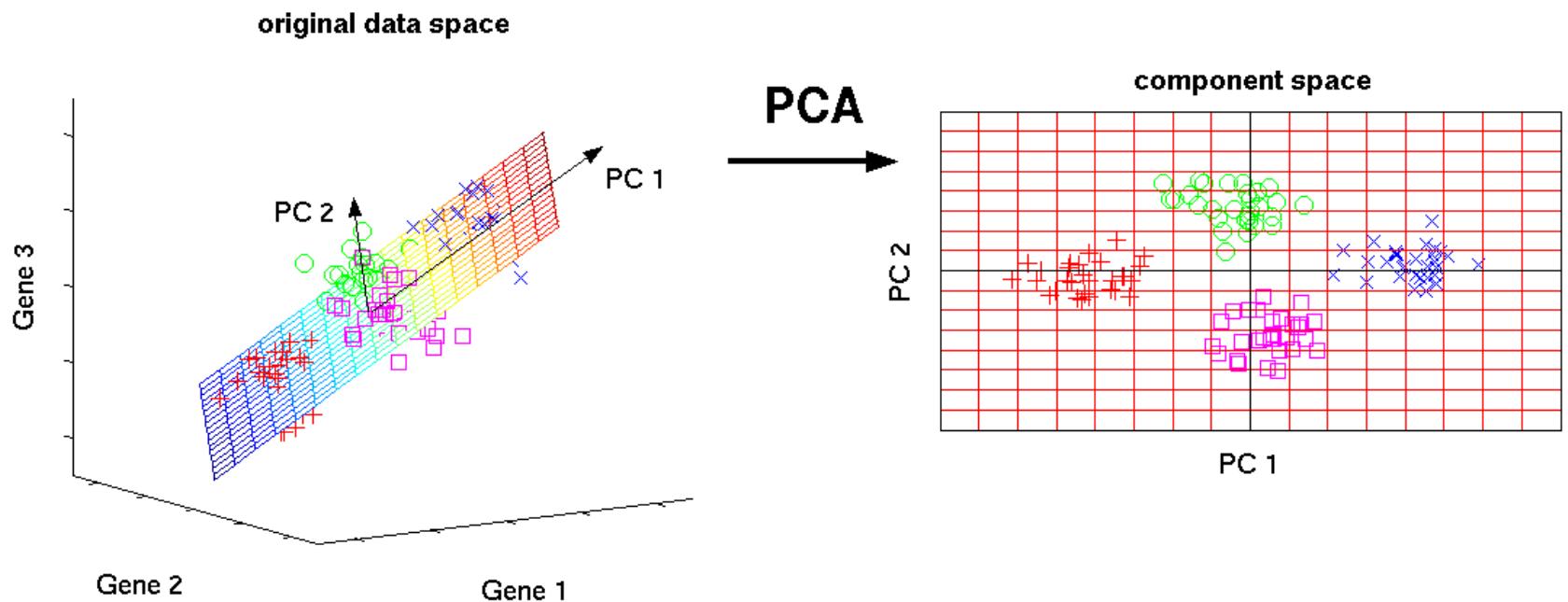


EM Clustering



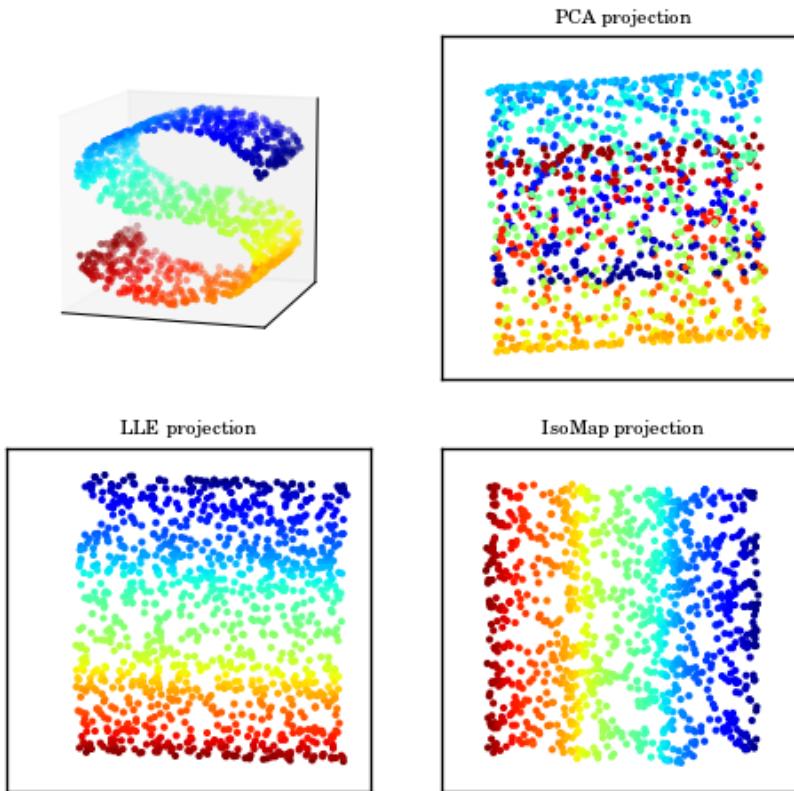
UNSUPERVISED LEARNING

Dimensionality Reduction: Subspace Learning

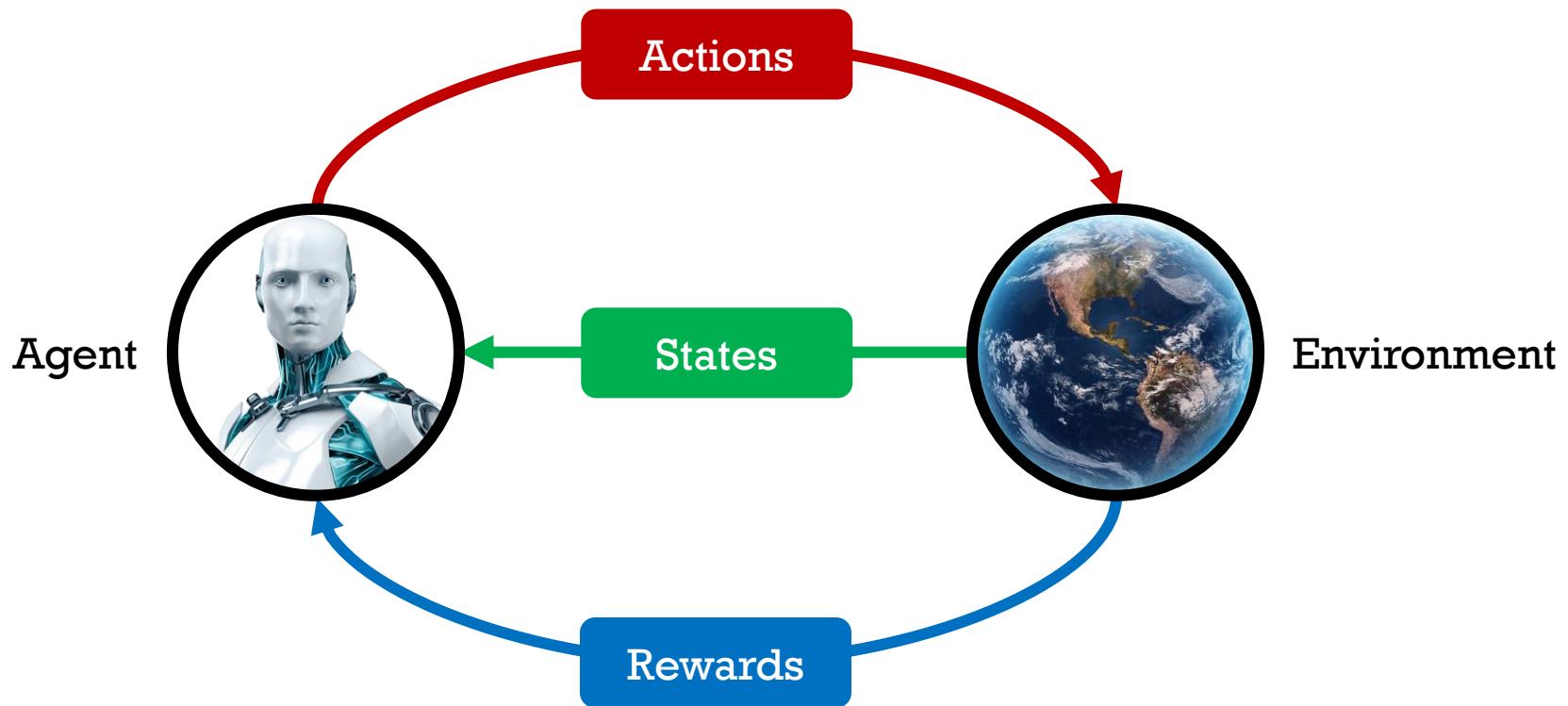


UNSUPERVISED LEARNING

Dimensionality Reduction: Manifold Learning



REINFORCEMENT LEARNING



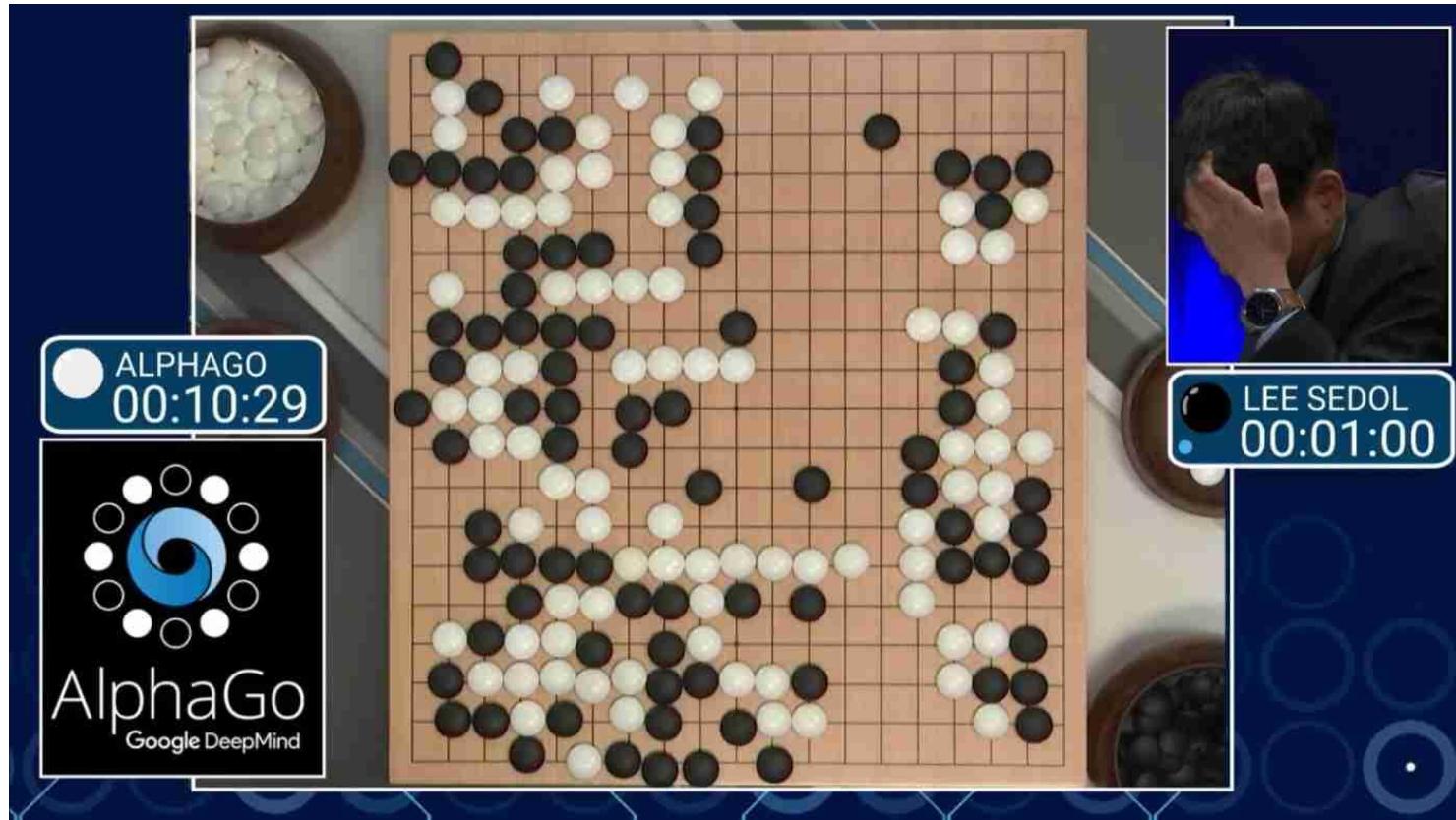
ATARI GAMES



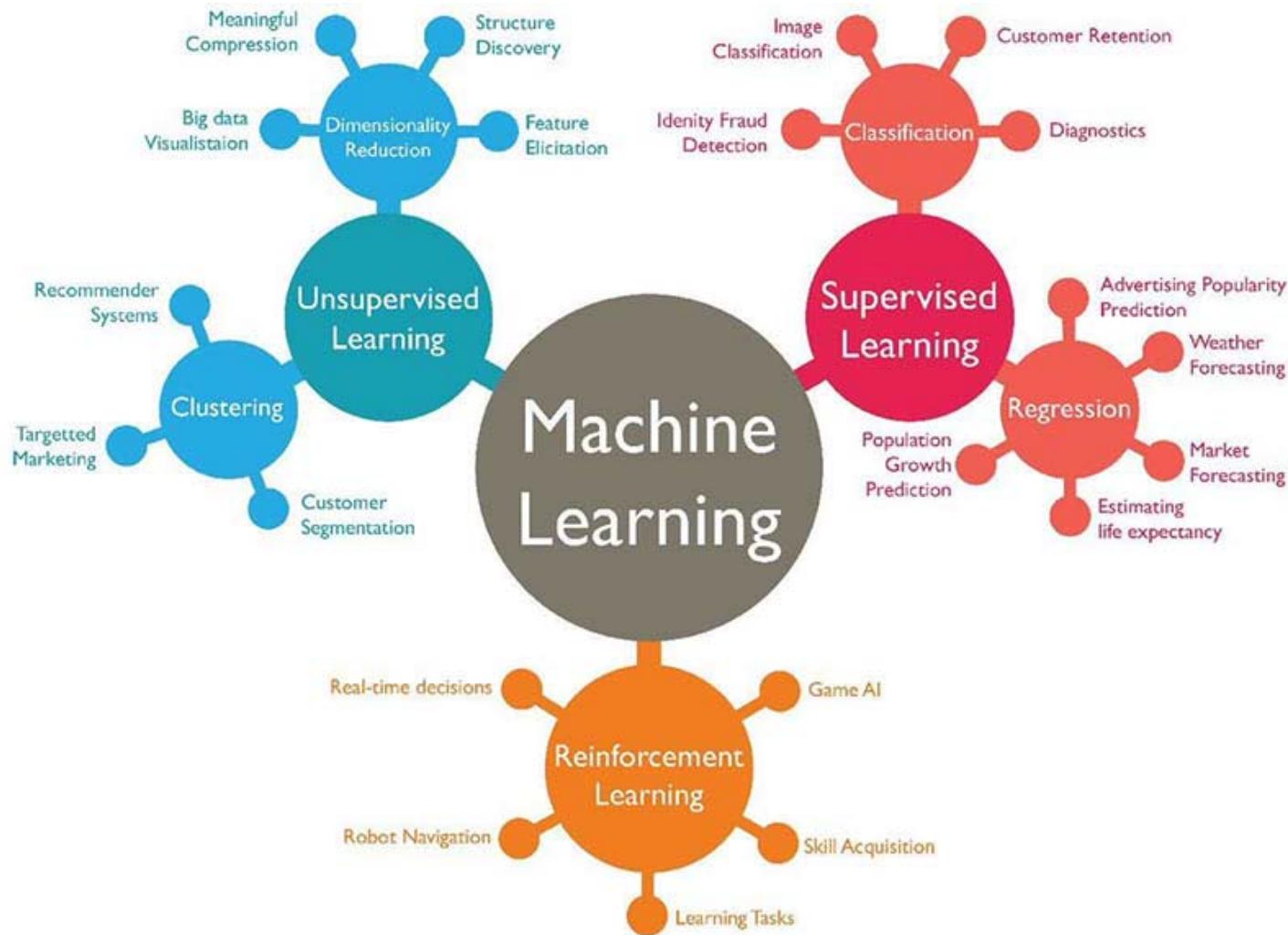
Google DEEPMIND



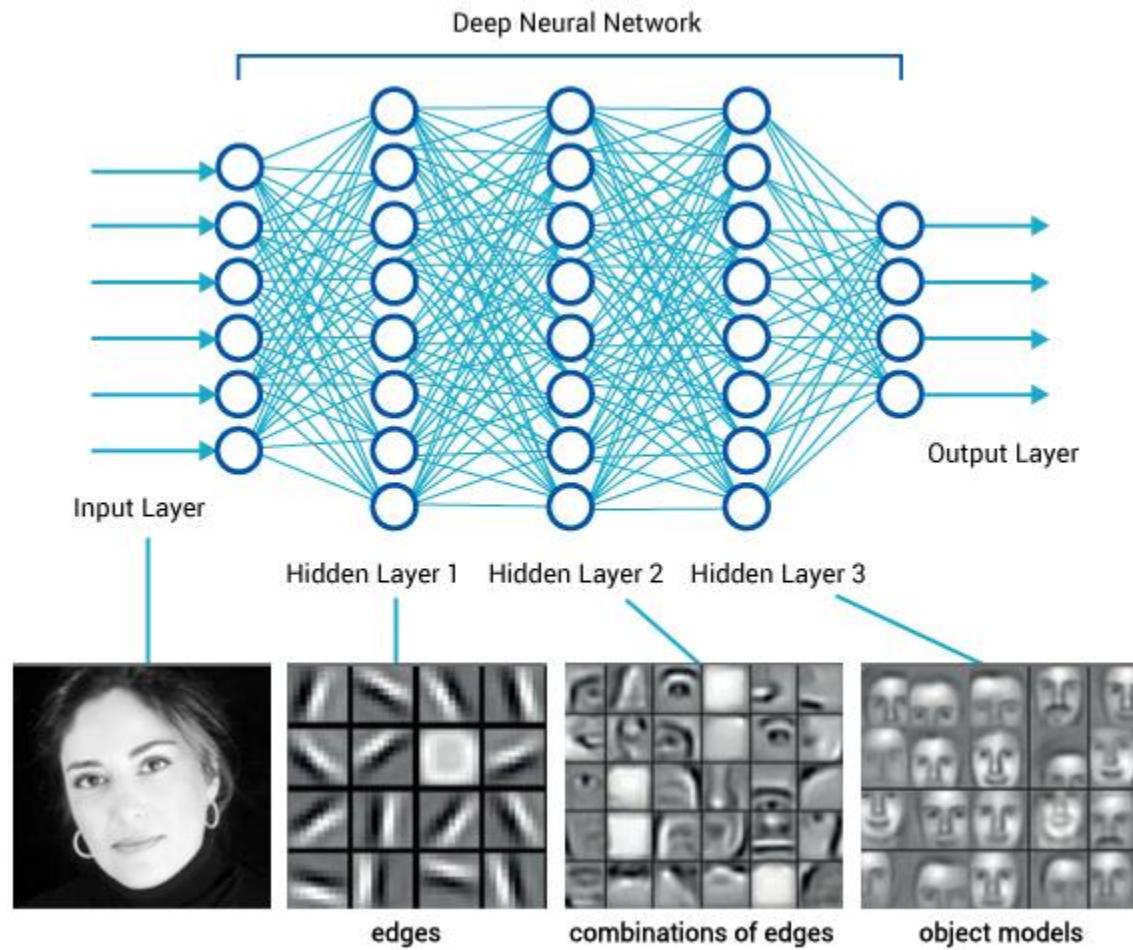
ALPHAGO



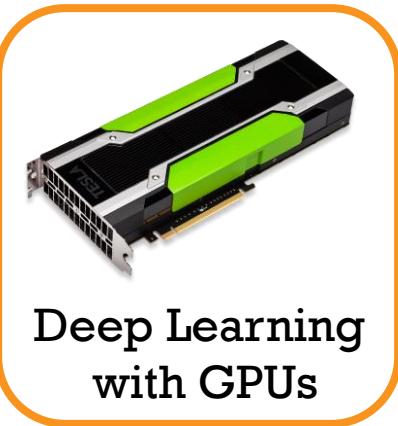
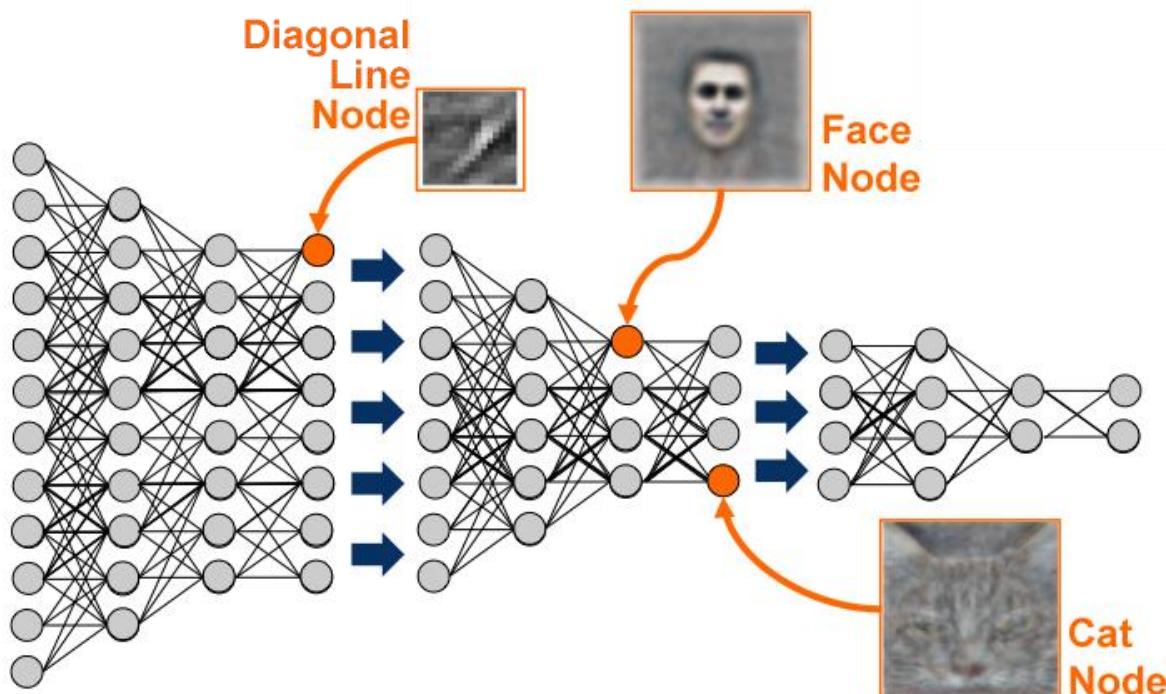
Machine Learning



DEEP LEARNING



GOOGLE CAT VIDEOS



5 min break

Linear algebra review

Let u and v be vectors in \mathbb{R}^d . The inner-product is defined as

$$\langle u, v \rangle = \left\langle \begin{bmatrix} u_1 \\ \vdots \\ u_d \end{bmatrix}, \begin{bmatrix} v_1 \\ \vdots \\ v_d \end{bmatrix} \right\rangle = \sum_{i=1}^d u_i v_d = [u_1, \dots, u_d] \begin{bmatrix} v_1 \\ \vdots \\ v_d \end{bmatrix} = u^T v$$

Recall the properties of the inner product:

- (i) $\langle u, v \rangle = \langle v, u \rangle$ (symmetry)
- (ii) $\langle au_1 + bu_2, v \rangle = a \langle u_1, v \rangle + b \langle u_2, v \rangle$ (linearity)

Now let $A = \{a_{ij}\}$ be a $d \times d$ matrix. We have

$$\begin{aligned}\langle u, Av \rangle &= \sum_{i=1}^d u_i (Av)_i \\&= \sum_{i=1}^d u_i \sum_{j=1}^d a_{ij} v_j \\&= \sum_{i=1}^d a_{ij} u_i \sum_{j=1}^d v_j \\&= \langle A^T u, v \rangle,\end{aligned}$$

where A^T is the transpose, or adjoint, of matrix A .

If $A^T = A^{-1}$, then A is an orthogonal matrix, i.e. $A^T = A^{-1}$ if and only if its columns are orthonormal vectors.

Example

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}^T = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}^{-1}$$

If $A^T = A$, then we say that A is self-adjoint, or symmetric.

Example

Hessian matrix of a (2-variable) function $f(x, y)$:

$$\nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

Covariance matrices

Definition

The covariance matrix of a random vector $[X_1, \dots, X_d]^T$ is a $d \times d$ matrix whose $(i, j)^{th}$ entry is the covariance $\text{cov}(X_i, X_j) = \mathbb{E}[(X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])]$.

Example

Covariance matrix of a mean-zero random vector $[X, Y]^T$; i.e. $\mathbb{E}[X] = 0 = \mathbb{E}[Y]$:

$$\begin{bmatrix} \mathbb{E}[X^2] & \mathbb{E}[XY] \\ \mathbb{E}[YX] & \mathbb{E}[Y^2] \end{bmatrix}$$

Gaussian probability density function

One dimension:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Multi-variate case ($x \in \mathbb{R}^d$):

$$p(x) = \frac{1}{\sqrt{\det A} (\sqrt{2\pi})^d} e^{-\frac{1}{2} \langle x - \mu, A^{-1}(x - \mu) \rangle},$$

where A is the covariance matrix.

Theorem

Let A be a self-adjoint matrix. Then there exists an orthonormal eigenbasis (v_1, \dots, v_d) with eigenvalues $\lambda_1, \dots, \lambda_d$ such that

$$A = \begin{bmatrix} | & \cdots & | \\ v_1 & \ddots & v_d \\ | & \cdots & | \end{bmatrix} \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_d \end{bmatrix} \begin{bmatrix} | & \cdots & | \\ v_1 & \ddots & v_d \\ | & \cdots & | \end{bmatrix}^{-1}.$$

First-order approximation

$$f(x + h) \approx f(x) + \langle \nabla f(x), h \rangle$$

Example

In two dimensions,

$$f(x_1 + h_1, x_2 + h_2) \approx f(x_1, x_2) + \left\langle \begin{bmatrix} \frac{\partial f}{\partial e_1}(x_1, x_2) \\ \frac{\partial f}{\partial e_2}(x_1, x_2) \end{bmatrix}, \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} \right\rangle$$

Gradient ascent

What direction should h take to maximally increase f ?

Theorem (Cauchy-Schwarz inequality)

$|\langle u, v \rangle| \leq \|u\| \|v\|$, and both sides are equal if and only if $v = \lambda u$ for some $\lambda \in \mathbb{R}$.

Note that if $v = \lambda u$, we have

$$\begin{aligned}\langle u, v \rangle &= \langle u, \lambda u \rangle \\ &= \lambda \langle u, u \rangle = \lambda \|u\|^2,\end{aligned}$$

so we get maximum increase if $h = \lambda \nabla f$ and $\lambda > 0$ (and similarly maximum decrease if $\lambda < 0$).

Second-order approximation

$$f(x + h) \approx f(x) + \langle \nabla f(x), h \rangle + \frac{1}{2} \langle h, \nabla^2 f(x)h \rangle$$

Example

In two dimensions,

$$\begin{aligned} f(x_1 + h_1, x_2 + h_2) &\approx f(x_1, x_2) + \left\langle \begin{bmatrix} \frac{\partial f}{\partial e_1}(x_1, x_2) \\ \frac{\partial f}{\partial e_2}(x_1, x_2) \end{bmatrix}, \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} \right\rangle \\ &+ \frac{1}{2} \left\langle \begin{bmatrix} h_1 \\ h_2 \end{bmatrix}, \begin{bmatrix} \frac{\partial^2 f}{\partial e_1^2}(x_1, x_2) & \frac{\partial^2 f}{\partial e_1 \partial e_2}(x_1, x_2) \\ \frac{\partial^2 f}{\partial e_2 \partial e_1}(x_1, x_2) & \frac{\partial^2 f}{\partial e_2^2}(x_1, x_2) \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} \right\rangle \end{aligned}$$

Newton's method

Let $\tilde{f}(x) := f(x) + \langle \nabla f(x), h \rangle + \frac{1}{2} \langle h, \nabla^2 f(x) h \rangle$. To find the best h to maximize (minimize) \tilde{f} , we take the gradient with respect to h and set to 0:

$$\nabla_h \tilde{f}(x) = \nabla f(x) + \nabla^2 f(x) h = 0,$$

which implies that

$$h = (\nabla^2 f(x))^{-1} (-\nabla f(x)).$$

- In summary, we optimize at each time step by

$$x_{n+1} = x_n + \alpha h,$$

where $h = \pm \nabla f(x_n)$ in the case of gradient descent (ascent), and $h = (\nabla^2 f(x))^{-1}(-\nabla f(x))$ in Newton's method.

- Newton's method converges faster with fewer iterations, but each iteration is computationally more expensive as one has to invert the Hessian matrix.

Gentle introduction to tensors

We encountered the quadratic form

$$\langle x, Ax \rangle$$

several times already; now we will discuss its significance in terms of tensors.

Definition

A k -tensor $T : \underbrace{\mathbb{R}^d \times \cdots \times \mathbb{R}^d}_{k \text{ times}} \rightarrow \mathbb{R}$ is a scalar function which is linear in each component.

Example

Given a vector u , the function

$$x \mapsto \langle u, x \rangle$$

is a 1-tensor.

Example

Given a matrix A , the function

$$(x, y) \mapsto \langle x, Ay \rangle$$

is a 2-tensor.

Example

The determinant of a $d \times d$ matrix, as a function of the d columns (or rows), is a d -tensor.

Theorem

All 1-tensors $T(x)$ can be written as

$$\langle u_T, x \rangle$$

for some vector u_T , and all 2-tensors $S(x, y)$ can be expressed as

$$\langle x, A_S y \rangle$$

for some matrix A_S .

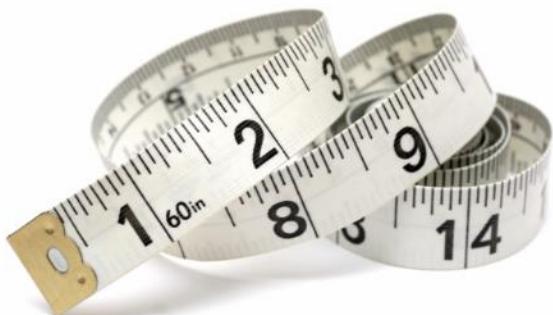
REGRESSION



MACHINE LEARNING



Task



Performance



Experience

Algorithms that improve their performance
at some task with experience

– Tom Mitchell (1998)



REGRESSION

Machine Learning

> Supervised Learning
> Regression

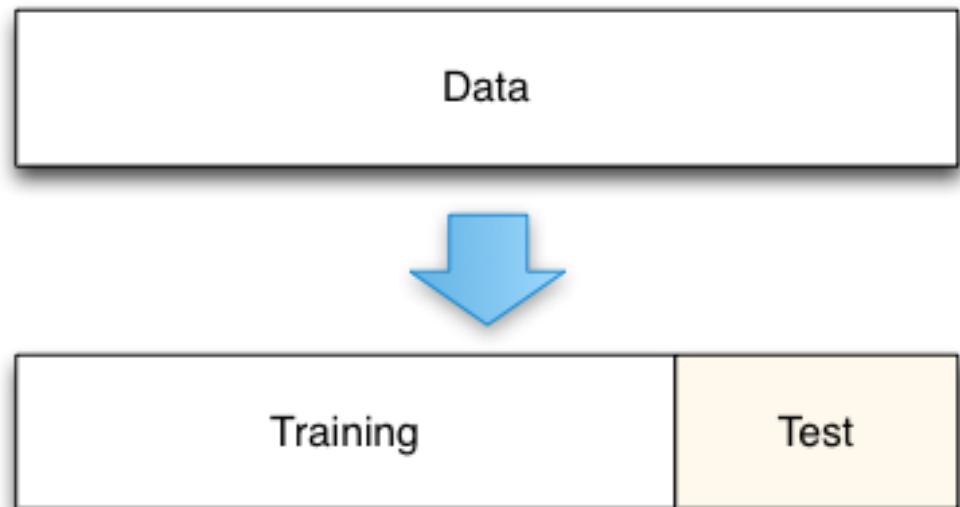
- **Task.** Find function $f: \mathbb{R}^d \rightarrow \mathbb{R}$ such that $y \approx f(x; \theta)$
- **Experience.** Training data $(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})$
- **Performance.** Prediction error $y - f(x; \theta)$ on test data



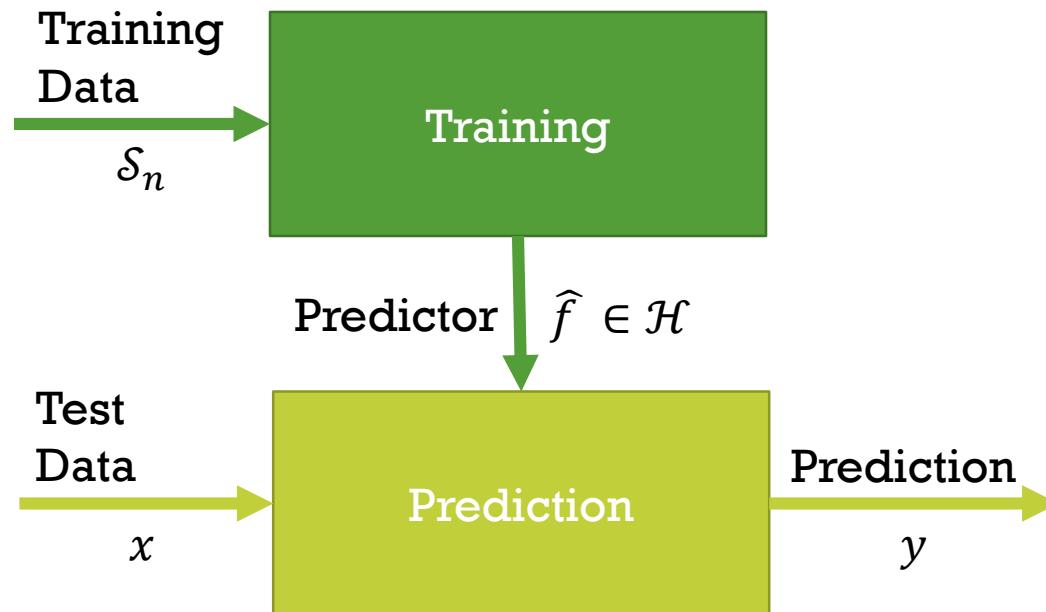
TRAINING DATA VS TEST DATA

Partition data into:

- Training data set \mathcal{S}_n
- Test data set \mathcal{S}_*



LEARNING AND PREDICTION



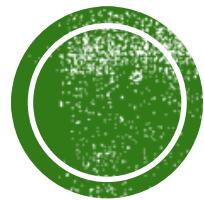
Assumption. Test data and training data are **identically distributed**.



GENERALIZATION

The goal of machine learning is to find a predictor $\hat{f} \in \mathcal{H}$ that **generalizes** well, i.e. that predicts well on test data \mathcal{S}_* .





MULTIVARIATE LINEAR REGRESSION



Given a training set $\{x^{(i)}\}_{i=1}^m$, where $x^{(i)} \in \mathbb{R}^n = (x_1^{(i)}, \dots, x_n^{(i)})$ we define the design matrix as

$$X = \begin{bmatrix} x_1^{(1)} & \dots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(m)} & \dots & x_n^{(m)} \end{bmatrix},$$

and denote the parameters $\theta \in \mathbb{R}^n$ and target values $y \in \mathbb{R}^m$ as

$$\theta = \begin{bmatrix} \theta^{(1)} \\ \vdots \\ \theta^{(n)} \end{bmatrix}, \quad y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}.$$

Then the least squares error

$$\mathcal{J}(\theta) = \frac{1}{2} \sum_{i=1}^m \left(\langle \theta, x^{(i)} \rangle - y^{(i)} \right)^2$$

can be written as $\frac{1}{2} \|X\theta - y\|^2 = \frac{1}{2} \langle X\theta - y, X\theta - y \rangle$, or alternatively as

$$\frac{1}{2} (X\theta - y)^T (X\theta - y).$$

Exact formula

- Recall from Systems World that finding the value of θ that minimizes the distance between $X\theta$ and y is the same as finding the projection of y onto the column space of X .
- This is the same as finding θ such that $X\theta - y$ is perpendicular to every column of X , i.e.

$$X^T(X\theta - y) = 0 \implies X^T X\theta = X^T y,$$

which gives $\theta = (X^T X)^{-1} X^T y$.

Gradient descent

- If the matrix X is large, inverting $X^T X$ will be computationally costly.
- Instead we can find the least squares error using gradient descent. First we compute the gradient of the loss function:

$$\begin{aligned}\frac{\partial \mathcal{J}(\theta)}{\partial \theta_j} &= \sum_{i=1}^m \left(\langle \theta, x^{(i)} \rangle - y^{(i)} \right) \frac{\partial \langle \theta, x^{(i)} \rangle}{\partial \theta_j} \\ &= \sum_{i=1}^m \left(\langle \theta, x^{(i)} \rangle - y^{(i)} \right) \frac{\partial}{\partial \theta_j} \sum_{k=1}^n x_k^{(i)} \theta_k \\ &= \sum_{i=1}^m \left(\langle \theta, x^{(i)} \rangle - y^{(i)} \right) x_j^{(i)}\end{aligned}$$

Gradient descent cont.

- We then use it to perform the update rule

$$\theta_j(t+1) = \theta_j(t) - \alpha \sum_{i=1}^m \left(\langle \theta(t), x^{(i)} \rangle - y^{(i)} \right) x_j^{(i)}, \quad j = 1, \dots, n$$

until the change in the loss function is below a certain preset tolerance.

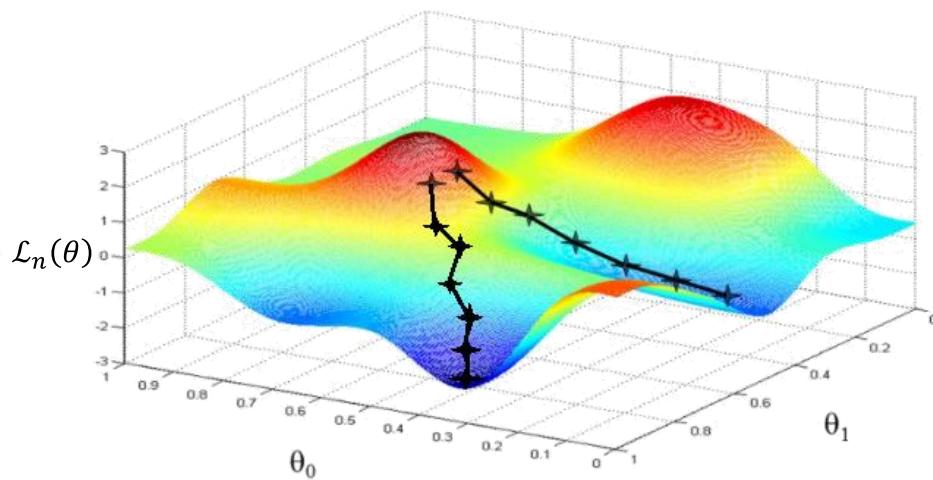
- In matrix notation the update rule can be written as

$$\theta(t+1) = \theta(t) - \alpha (X^T X \theta(t) - X^T y)$$

- This is also known as batch gradient descent, where all m training points are used at every step.
- For linear regression, the loss function is convex and has only one global minimum, so convergence is guaranteed unless the learning rate α is too large.

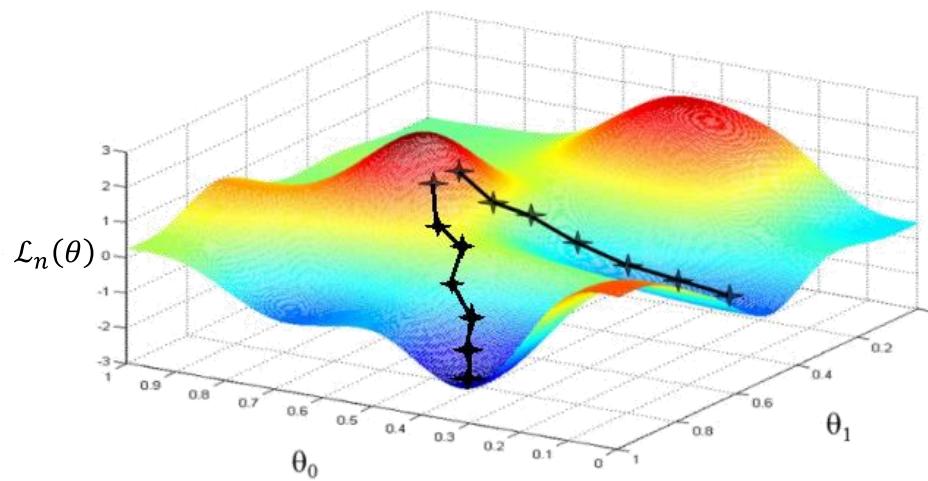
GRADIENT DESCENT

1. Initialize θ randomly.
2. Update $\theta \leftarrow \theta - \eta_k \nabla \mathcal{L}_n(\theta)$,
 η_k learning rate,
 k iteration number.
3. Repeat (2) until convergence.
(e.g. when improvement in $\mathcal{L}_n(\theta)$ is small enough)



LOCAL MINIMA

- Gradient descent leads us to a local minimum, which is not necessarily the global minimum. Different starting points may lead to different local minima.
- Typically, we perform gradient descent from several starting points, and run through all the local minima to find the parameter that has the smallest training loss.



STOCHASTIC GRADIENT DESCENT

training gradient = average of point gradients

$$\nabla \mathcal{L}_n(\theta; \mathcal{S}_n) = \frac{1}{n} \sum_{(x,y) \in \mathcal{S}_n} \nabla \mathcal{L}_1(\theta; x, y)$$

This average can take a long time to compute for large data sets.

Trick

Estimate the gradient by averaging over a smaller *minibatch* (subset of the training data).

$$\begin{aligned}\nabla \mathcal{L}_n(\theta; \mathcal{S}_n) &\approx \nabla \mathcal{L}_m(\theta; \mathcal{B}_m) \\ &= \frac{1}{m} \sum_{(x,y) \in \mathcal{B}_m} \nabla \mathcal{L}_1(\theta; x, y)\end{aligned}$$



Probabilistic interpretation

Assume for each i that

$$y^{(i)} = \langle \theta, x^{(i)} \rangle + \epsilon^{(i)},$$

where $\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$ and are independent from one another. This implies that $y^{(i)} \mid x^{(i)} \sim \mathcal{N}(\langle \theta, x^{(i)} \rangle, \sigma^2)$, i.e.

$$p_{\theta} (y^{(i)} \mid x^{(i)}) = \frac{1}{\sigma \sqrt{2\pi}} \exp \frac{-(y^{(i)} - \langle \theta, x^{(i)} \rangle)^2}{2\sigma^2}.$$

Given this setup, we want to find the optimal value of θ which maximizes the likelihood function

$$\mathcal{L}(\theta) = \prod_{i=1}^m p_\theta \left(y^{(i)} \mid x^{(i)} \right),$$

or equivalently the log-likelihood function (since log is a monotonic function)

$$\begin{aligned}\ell(\theta) &= \log \prod_{i=1}^m p_\theta \left(y^{(i)} \mid x^{(i)} \right) \\ &= \sum_{i=1}^m \log p_\theta \left(y^{(i)} \mid x^{(i)} \right) \\ &= m \log \frac{1}{\sigma \sqrt{2\pi}} - \frac{1}{2\sigma^2} \sum_{i=1}^m \left(y^{(i)} - \langle \theta, x^{(i)} \rangle \right)^2.\end{aligned}$$

Since m , π and σ are constants, we find that maximizing the likelihood is the same as minimizing the least squares error

$$\sum_{i=1}^m \left(y^{(i)} - \langle \theta, x^{(i)} \rangle \right)^2,$$

and that the choice of σ was inconsequential.

Bias-Variance tradeoff

- In general, assume that the targets are given by

$$y^{(i)} = f(x^{(i)}) + \epsilon^{(i)},$$

for some unknown function f of the inputs which represents the true model.

- We assume that the $\epsilon^{(i)}$'s have mean 0 and variance σ^2 , and are independent from one another.

Given a fixed dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$, and denoting $\hat{f}_{\mathcal{D}}$ as the learnt hypothesis function (which depends on \mathcal{D}), the mean-squared error is given by

$$\frac{1}{n} \sum_{x^{(i)} \in \mathcal{D}} \int_{\mathbb{R}} \left(\hat{f}_{\mathcal{D}}(x^{(i)}) - y^{(i)} \right)^2 p(\epsilon^{(i)}) d\epsilon^{(i)}.$$

This expression can be simplified by noting that

$$\begin{aligned} & \int_{\mathbb{R}} \left(\hat{f}_{\mathcal{D}}(x^{(i)}) - y^{(i)} \right)^2 p(\epsilon^{(i)}) d\epsilon^{(i)} \\ &= \int_{\mathbb{R}} \left(\hat{f}_{\mathcal{D}}(x^{(i)}) - f(x^{(i)}) - \epsilon^{(i)} \right)^2 p(\epsilon^{(i)}) d\epsilon^{(i)} \\ &= \left(\hat{f}_{\mathcal{D}}(x^{(i)}) - f(x^{(i)}) \right)^2 + \left(\hat{f}_{\mathcal{D}}(x^{(i)}) - f(x^{(i)}) \right) \int_{\mathbb{R}} \epsilon_i p(\epsilon^{(i)}) d\epsilon^{(i)} \\ &\quad + \int_{\mathbb{R}} \epsilon_i^2 p(\epsilon^{(i)}) d\epsilon^{(i)} \\ &= \left(\hat{f}_{\mathcal{D}}(x^{(i)}) - f(x^{(i)}) \right)^2 + \sigma^2, \end{aligned}$$

and hence

$$\frac{1}{n} \sum_{x^{(i)} \in \mathcal{D}} \int_{\mathbb{R}} \left(\hat{f}_{\mathcal{D}}(x^{(i)}) - y^{(i)} \right)^2 p(\epsilon^{(i)}) d\epsilon^{(i)} = \sigma^2 + \frac{1}{n} \sum_{x^{(i)} \in \mathcal{D}} \left(\hat{f}_{\mathcal{D}}(x^{(i)}) - f(x^{(i)}) \right)^2$$

We would like to average this over all possible datasets to measure the performance of our learning algorithm. To model this, we can treat \mathcal{D} as a random variable drawn from some distribution $p(\mathcal{D})$, and denote the expected error as

$$\begin{aligned} & \int \sigma^2 + \frac{1}{n} \sum_{x^{(i)} \in \mathcal{D}} \left(\hat{f}_{\mathcal{D}}(x^{(i)}) - f(x^{(i)}) \right)^2 p(\mathcal{D}) d\mathcal{D} \\ &= \sigma^2 + \frac{1}{n} \sum_{x^{(i)} \in \mathcal{D}} \mathbb{E}_{\mathcal{D}} \left[\left(\hat{f}_{\mathcal{D}}(x^{(i)}) - f(x^{(i)}) \right)^2 \right]. \end{aligned}$$

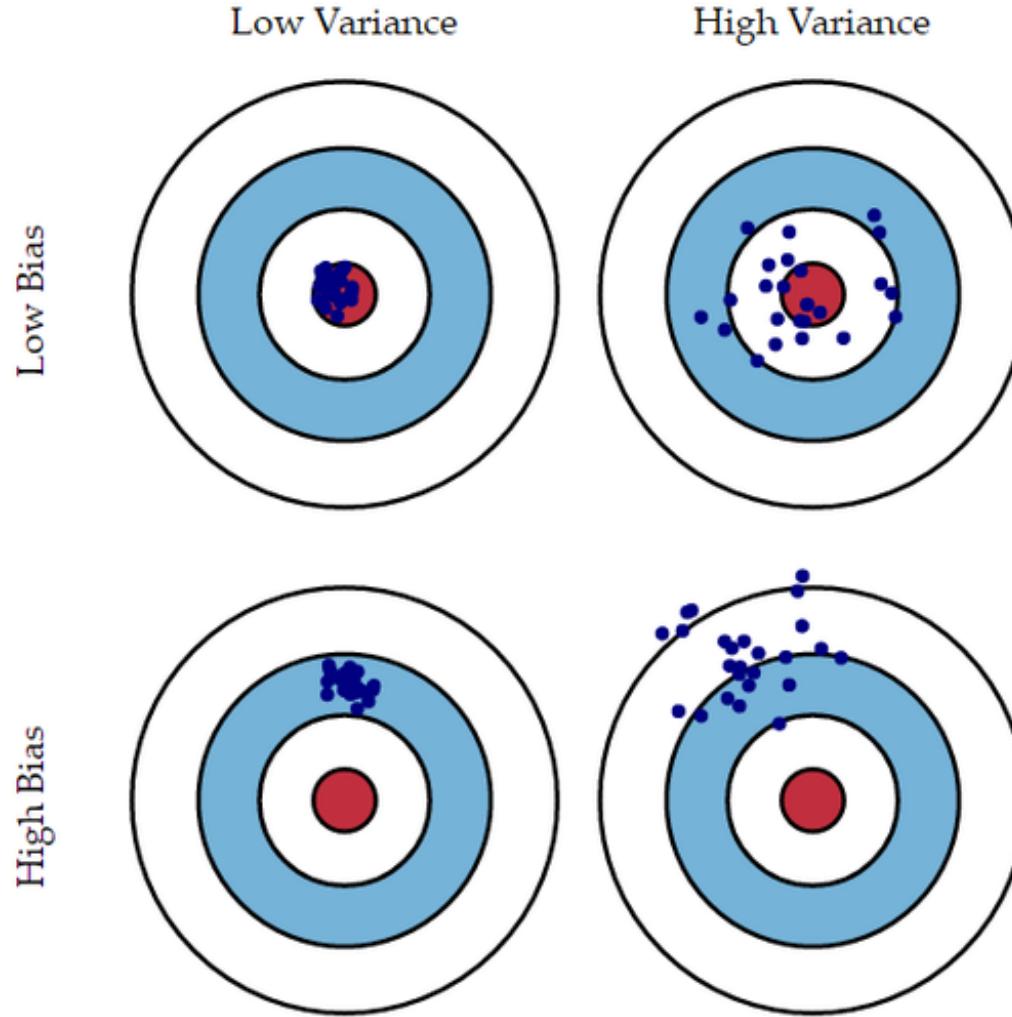
(we can bring the summation outside and sum over all $x^{(i)}$ by extending $\hat{f}_{\mathcal{D}}$ and f to be 0 if $x^{(i)}$ is not in \mathcal{D})

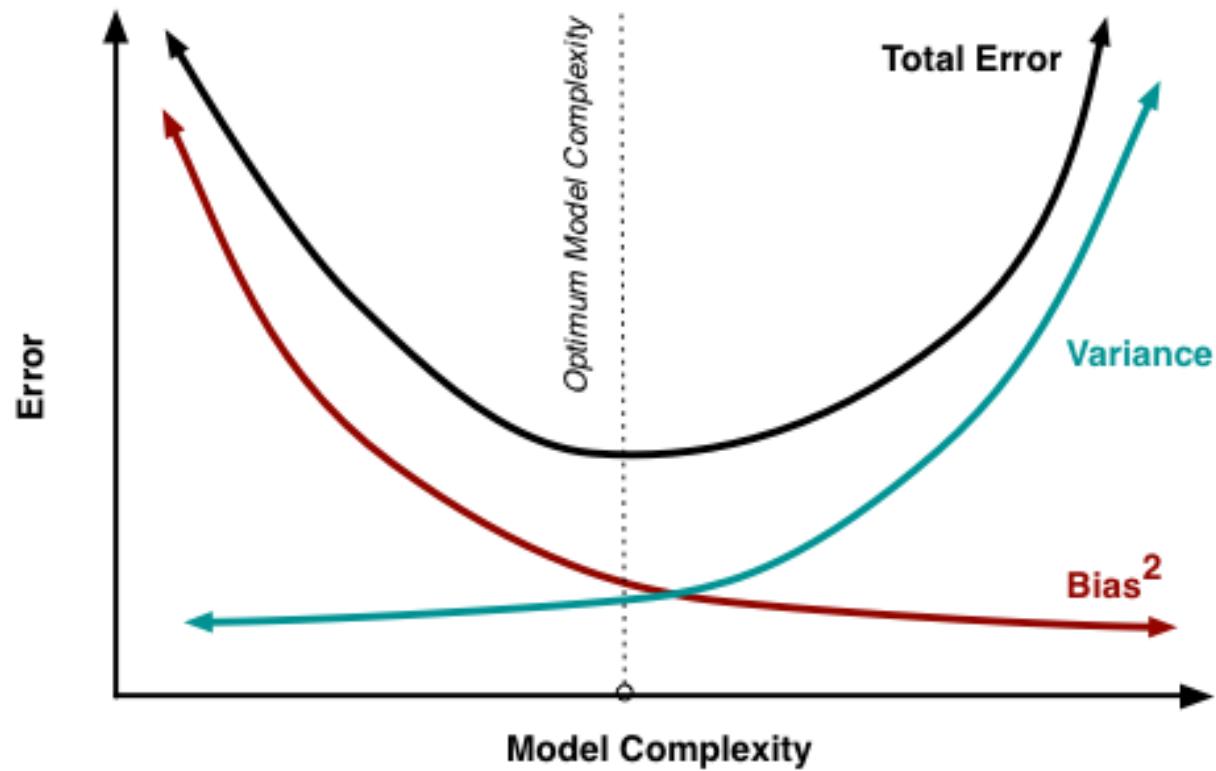
The expectation in the preceding line can be further decomposed as

$$\begin{aligned}\mathbb{E}_{\mathcal{D}} & \left[\left(\hat{f}_{\mathcal{D}}(x^{(i)}) - f(x^{(i)}) \right)^2 \right] \\&= \mathbb{E}_{\mathcal{D}} \left[\left(\hat{f}_{\mathcal{D}}(x^{(i)}) - \mathbb{E}_{\mathcal{D}} \left[\hat{f}_{\mathcal{D}}(x^{(i)}) \right] + \mathbb{E}_{\mathcal{D}} \left[\hat{f}_{\mathcal{D}}(x^{(i)}) \right] - f(x^{(i)}) \right)^2 \right] \\&= \mathbb{E}_{\mathcal{D}} \left[\left(\hat{f}_{\mathcal{D}}(x^{(i)}) - \mathbb{E}_{\mathcal{D}} \left[\hat{f}_{\mathcal{D}}(x^{(i)}) \right] \right)^2 \right] \\&\quad + 2 \left(\mathbb{E}_{\mathcal{D}} \left[\hat{f}_{\mathcal{D}}(x^{(i)}) \right] - f(x^{(i)}) \right) \mathbb{E}_{\mathcal{D}} \left[\hat{f}_{\mathcal{D}}(x^{(i)}) - \mathbb{E}_{\mathcal{D}} \left[\hat{f}_{\mathcal{D}}(x^{(i)}) \right] \right] \\&\quad + \mathbb{E}_{\mathcal{D}} \left[\left(\mathbb{E}_{\mathcal{D}} \left[\hat{f}_{\mathcal{D}}(x^{(i)}) \right] - f(x^{(i)}) \right)^2 \right] \\&= \mathbb{E}_{\mathcal{D}} \left[\left(\hat{f}_{\mathcal{D}}(x^{(i)}) - \mathbb{E}_{\mathcal{D}} \left[\hat{f}_{\mathcal{D}}(x^{(i)}) \right] \right)^2 \right] + \left(\mathbb{E}_{\mathcal{D}} \left[\hat{f}_{\mathcal{D}}(x^{(i)}) \right] - f(x^{(i)}) \right)^2.\end{aligned}$$

In conclusion, after summing over the $x^{(i)}$'s, the expected error can be written as

$$\underbrace{\left(\mathbb{E}[\hat{f}] - f \right)^2}_{\text{squared bias}} + \underbrace{\text{Var}(\hat{f})}_{\text{variance}} + \underbrace{\sigma^2}_{\text{irreducible error}}.$$





5 min break

MODEL SELECTION

Overfitting. If model \mathcal{H} is too big, then $\hat{f} \in \mathcal{H}$ performs

- well on training data, but poorly on test data.

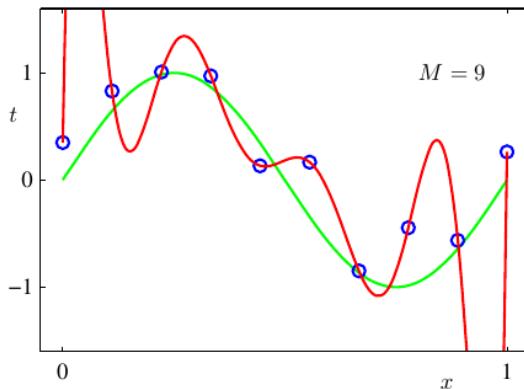
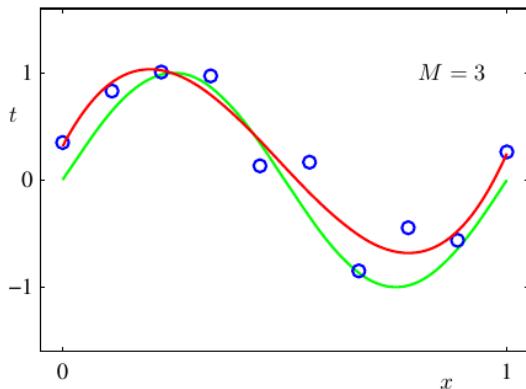
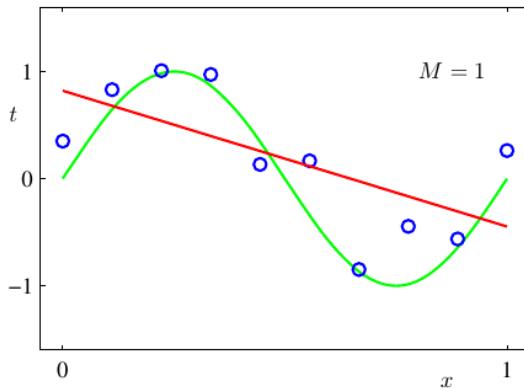
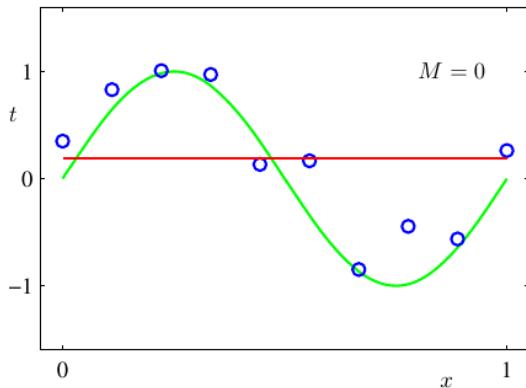
Underfitting. If model \mathcal{H} is too small, then $\hat{f} \in \mathcal{H}$ performs

- poorly on training data, and poorly on test data.

Finding a model with the right size is called **model selection**.



Model selection



Overfitting vs underfitting

Overfitting:

- Model fits the training data too well as it captures noise in addition to the underlying structure
- Low bias, high variance
- Training error low, testing error high

Underfitting:

- Model is too simple to capture the underlying structure of the data
- High bias, low variance
- Training error high, testing error high

Both lead to poor prediction performance on new input.

How to deal with underfitting or overfitting

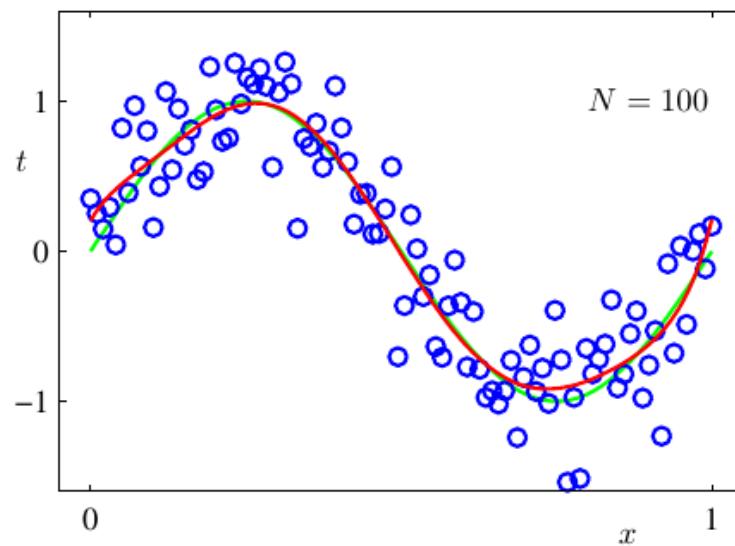
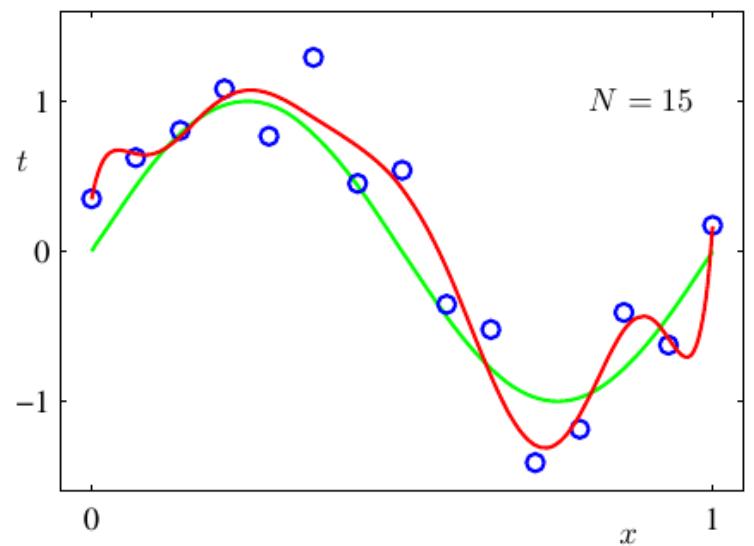
Underfitting:

- Increase model complexity
 - Adjust regularization

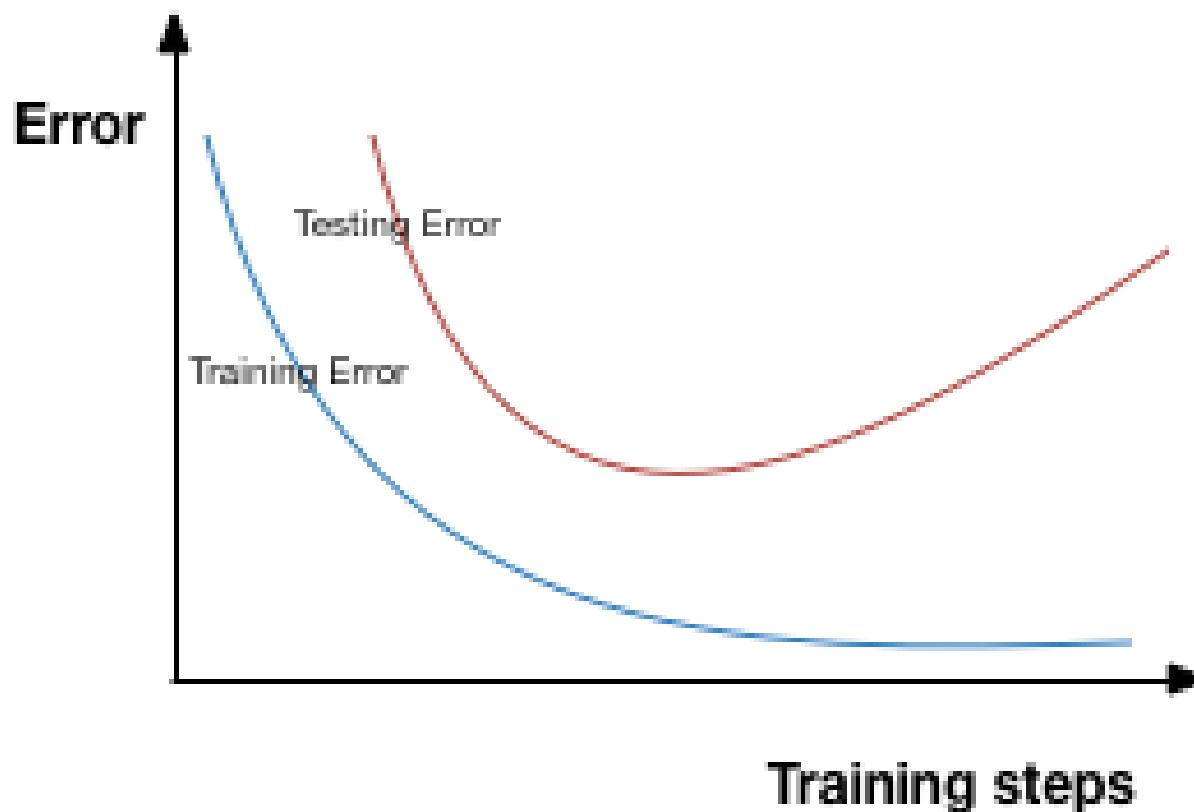
Overfitting:

- Increase data size
- Early stopping
- Decrease model complexity
 - Adjust regularization

Overfitting: Increase data size



Overfitting: Early stopping





REGULARIZATION



RIDGE REGRESSION

Add a penalty.

$$\mathcal{L}_{n,\lambda}(\theta) = \frac{1}{n} \sum_{\text{data } (x,y)} \frac{1}{2} (y - \theta^\top x)^2 + \frac{\lambda}{2} \|\theta\|^2$$

Pressure to fit data

Pressure to go to zero

Regularization parameter $\lambda \geq 0$

Regularizer

(Unfortunately, to include the parameter θ_0 , we cannot simply apply the constant feature trick. Why?)



TRAINING ALGORITHMS

Gradient

$$\nabla \mathcal{L}_{n,\lambda}(\theta) = \lambda\theta + \frac{1}{n}(X^\top X)\theta - \frac{1}{n}X^\top Y$$

Exact Solution

$$\begin{aligned}\nabla \mathcal{L}_{n,\lambda}(\hat{\theta}) = 0 &\Leftrightarrow \lambda\hat{\theta} + \frac{1}{n}(X^\top X)\hat{\theta} = \frac{1}{n}X^\top Y \\ &\Leftrightarrow \hat{\theta} = (n\lambda I + X^\top X)^{-1}X^\top Y\end{aligned}$$

This matrix is always invertible when $\lambda > 0$.



TRAINING ALGORITHMS

Gradient

$$\nabla \mathcal{L}_{n,\lambda}(\theta) = \lambda\theta + \frac{1}{n}(X^\top X)\theta - \frac{1}{n}X^\top Y$$

Gradient Descent

$$\theta \leftarrow (1 - \eta_k \lambda) \theta - \eta_k \left[\frac{1}{n}(X^\top X)\theta - \frac{1}{n}X^\top Y \right]$$

Without regularization,
i.e. $\lambda = 0$, this shrinkage
factor equals 1.



TRAINING LOSS VS TEST LOSS

Training Loss

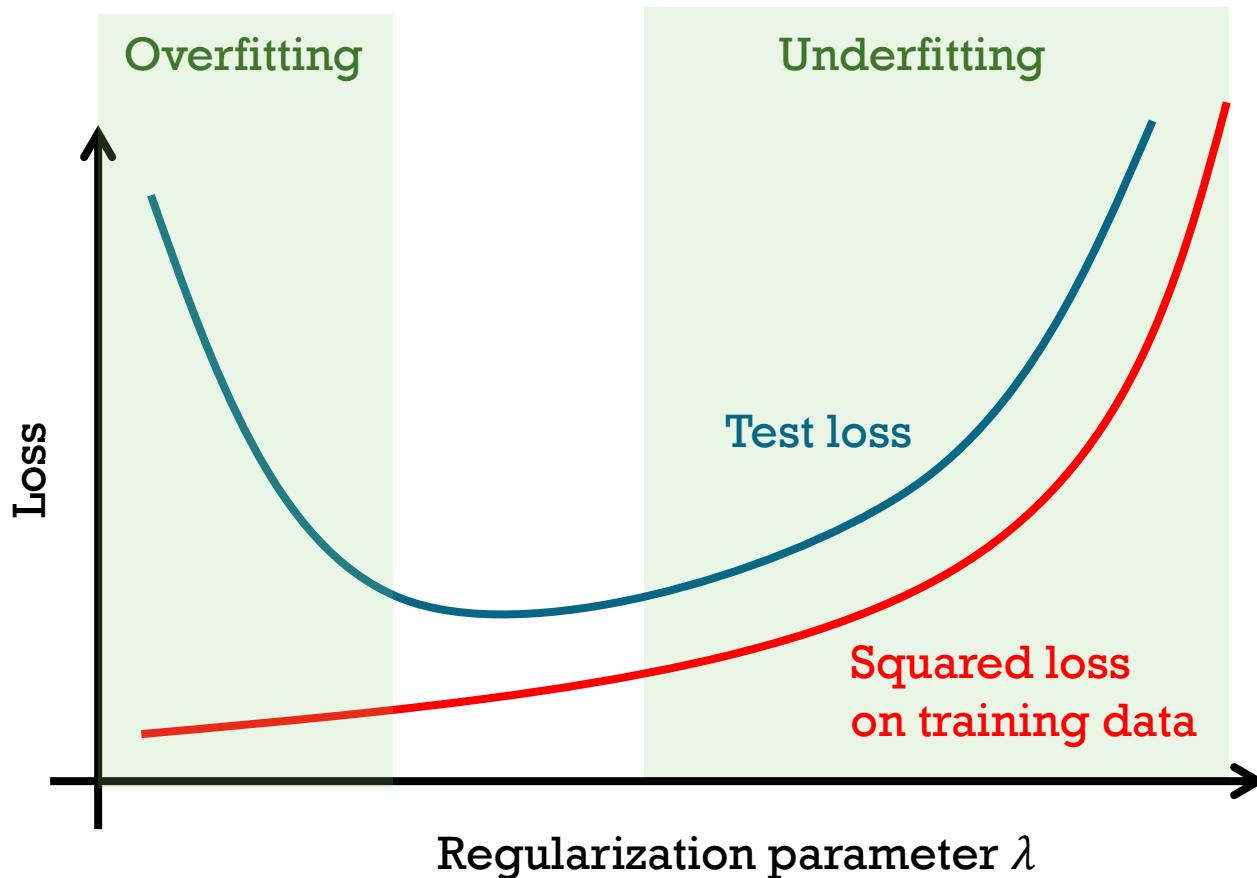
$$\mathcal{L}_{n,\lambda}(\theta) = \frac{1}{n} \sum_{\text{trg data } (x,y)} \frac{1}{2} (y - \theta^\top x)^2 + \frac{\lambda}{2} \|\theta\|^2$$

Test Loss

$$\mathcal{R}(\theta) = \frac{1}{n} \sum_{\text{test data } (x,y)} \frac{1}{2} (y - \theta^\top x)^2$$



EFFECT OF REGULARIZATION



Ridge regression vs LASSO

- LASSO (Least absolute shrinkage and selection operator) is similar to ridge regression, in that we have regression analysis performed with regularization.
- The difference is that the L_1 norm (hence the name L_1 regularization) is used to constrain the weights:

$$\arg \min_{\theta} \sum_{i=1}^n \left(\langle \theta, x^{(i)} \rangle - y^{(i)} \right)^2 + \lambda \sum_j |\theta_j|;$$

in contrast to the L_2 norm we have been seeing with ridge regression.

- With LASSO, the constraint on the weights typically sets some components of θ to 0, which can lead to sparser solutions, feature selection and better interpretability of the model.

We can explain this by reformulating the problems in an alternative but equivalent form:

Ridge regression:

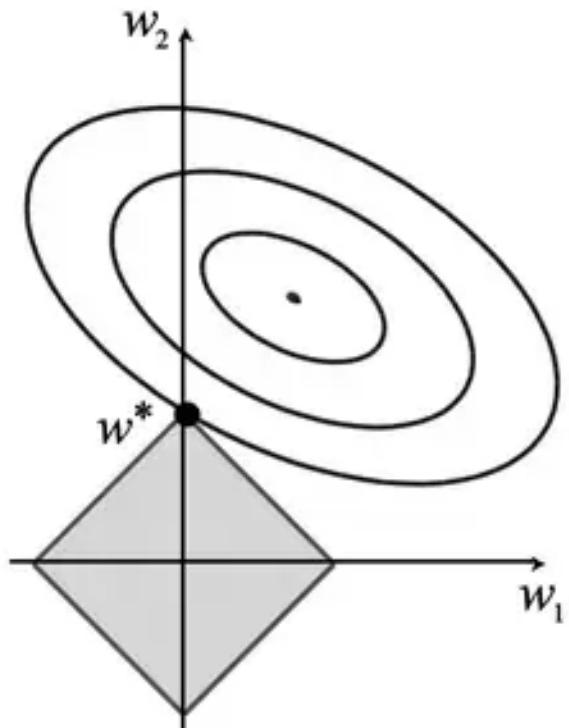
$$\arg \min_{\theta} \sum_{i=1}^n \left(\langle \theta, x^{(i)} \rangle - y^{(i)} \right)^2$$

subject to $\sum_j \theta_j^2 \leq t$

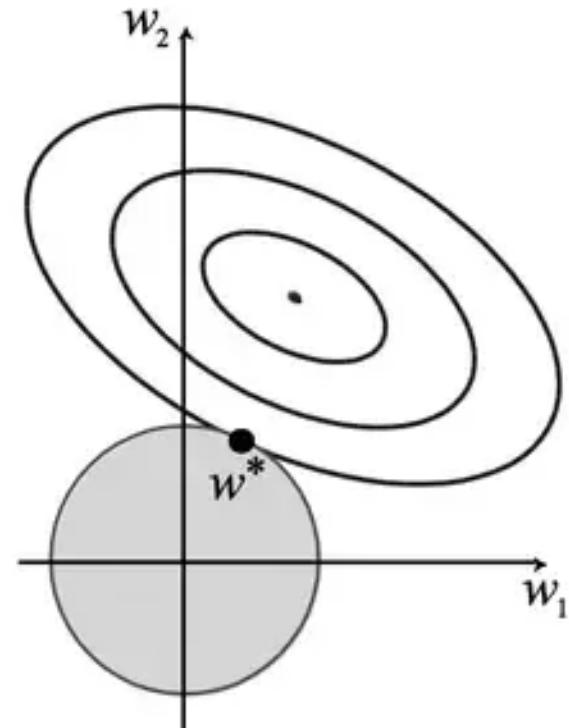
LASSO:

$$\arg \min_{\theta} \sum_{i=1}^n \left(\langle \theta, x^{(i)} \rangle - y^{(i)} \right)^2$$

subject to $\sum_j |\theta_j| \leq t$



L1



L2

CLASSIFICATION

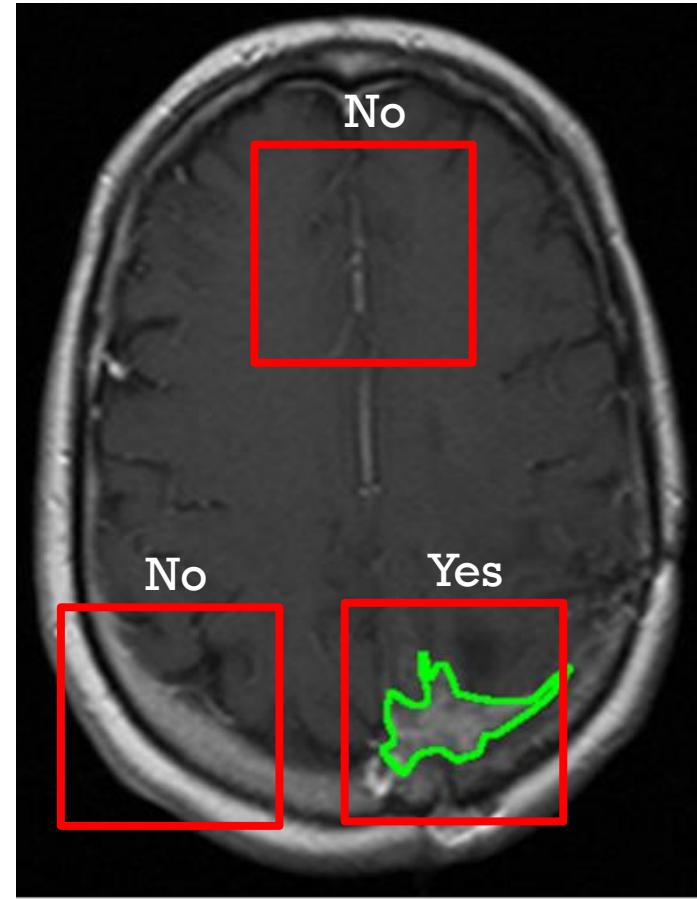


TUMOR CLASSIFICATION

Tumor?

Yes ~ +1

No ~ -1



SPAM FILTERS

Spam?

Yes ~ +1

No ~ -1



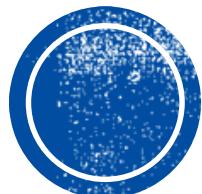
CLASSIFICATION

Machine Learning

> Supervised Learning
> Classification

- **Task.** Find $h: \mathbb{R}^d \rightarrow \{-1, +1\}$ such that $y \approx h(x; \theta)$
- **Experience.** Training data $(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})$
- **Performance.** Prediction error on test data





LINEAR CLASSIFICATION



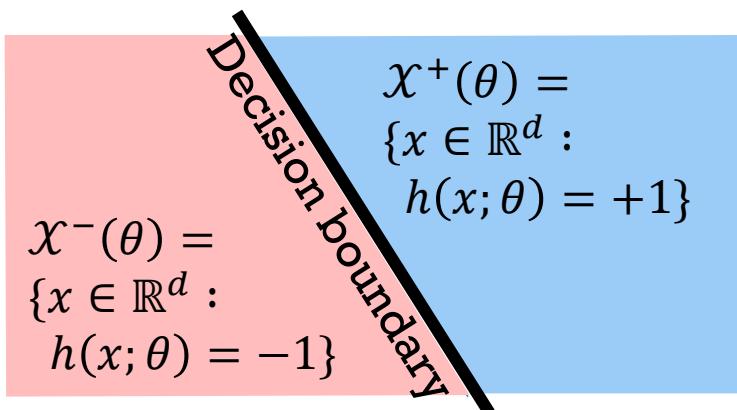
Training data

$$\mathcal{S}_n = \{ (x^{(i)}, y^{(i)}) \mid i = 1, \dots, n \}$$

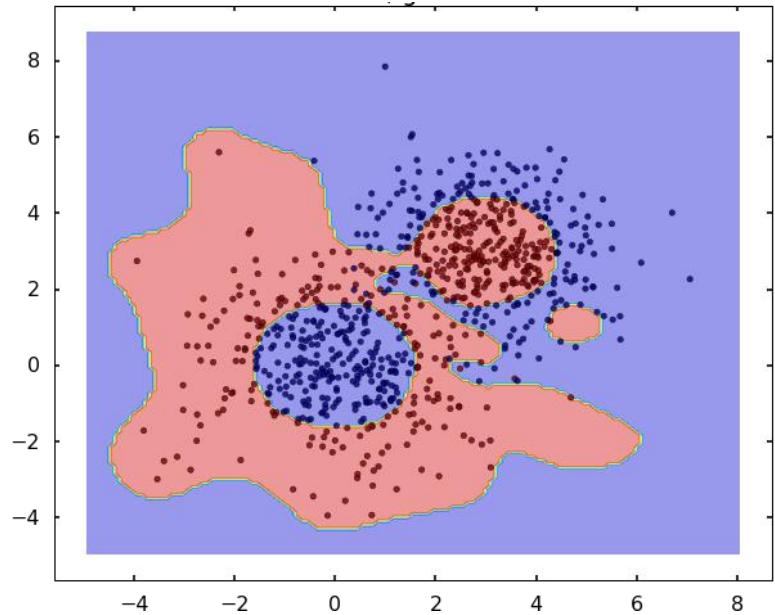
- **Features/Inputs** $x^{(i)} = (x_1^{(i)}, \dots, x_d^{(i)})^\top \in \mathbb{R}^d$
- **Labels/Output** $y^{(i)} \in \{-1, +1\}$



DECISION REGIONS



linear classifier



non-linear classifier

A classifier h partitions the space into **decision regions** that are separated by **decision boundaries**. In each region, all the points map to the same label. Many regions could have the same label.

For linear classifiers, these regions are **half spaces**.

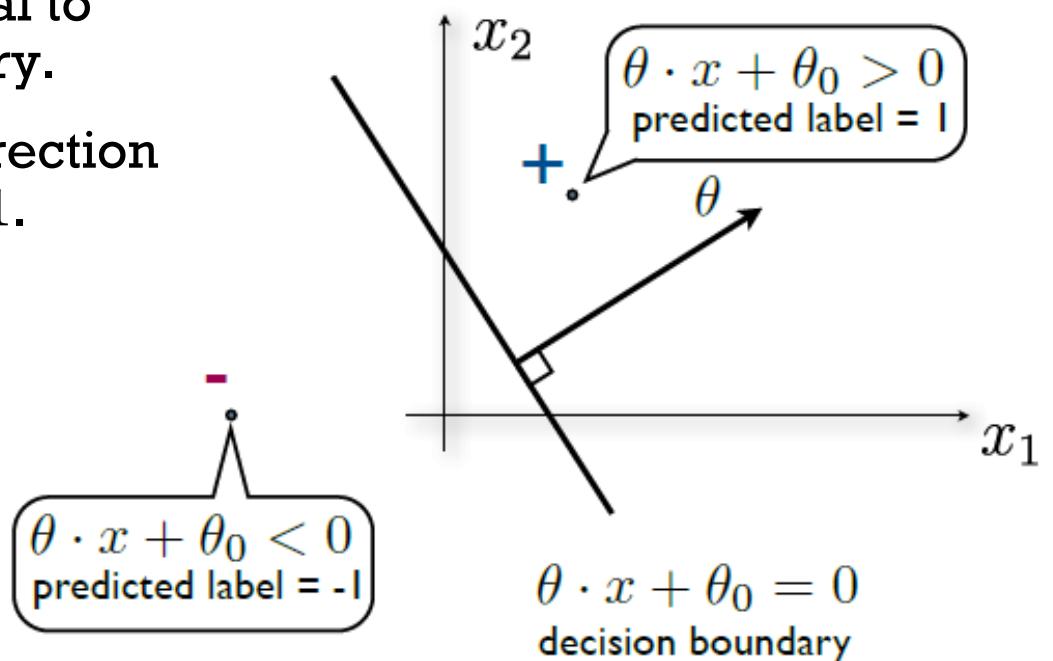


DECISION BOUNDARIES

For linear classifiers, the decision boundary is a **hyperplane** of dimension $d - 1$.

Vector θ is orthogonal to the decision boundary.

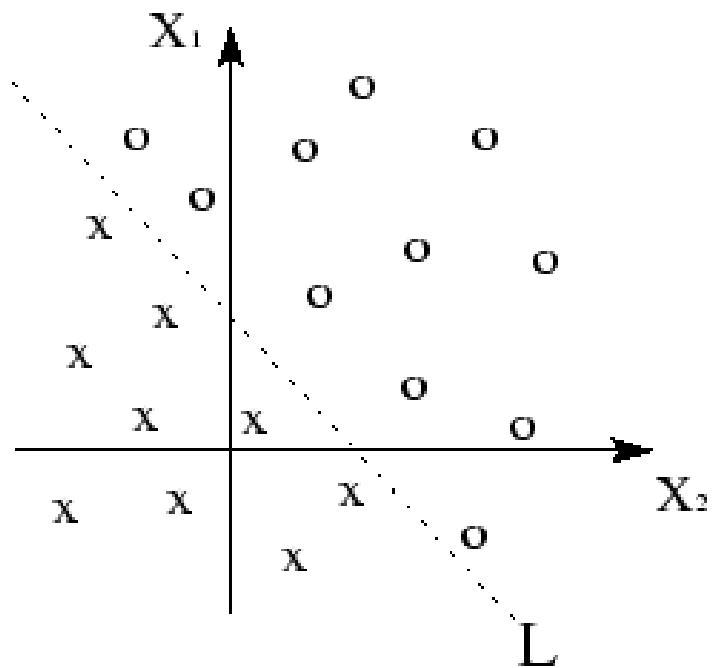
Vector θ points in direction of region labelled +1.



LINEARLY SEPARABLE

The training data \mathcal{S}_n is
linearly separable
if there exists a
parameters θ and θ_0 such
that for all $(x, y) \in \mathcal{S}_n$,

$$y(\theta^T x + \theta_0) > 0.$$



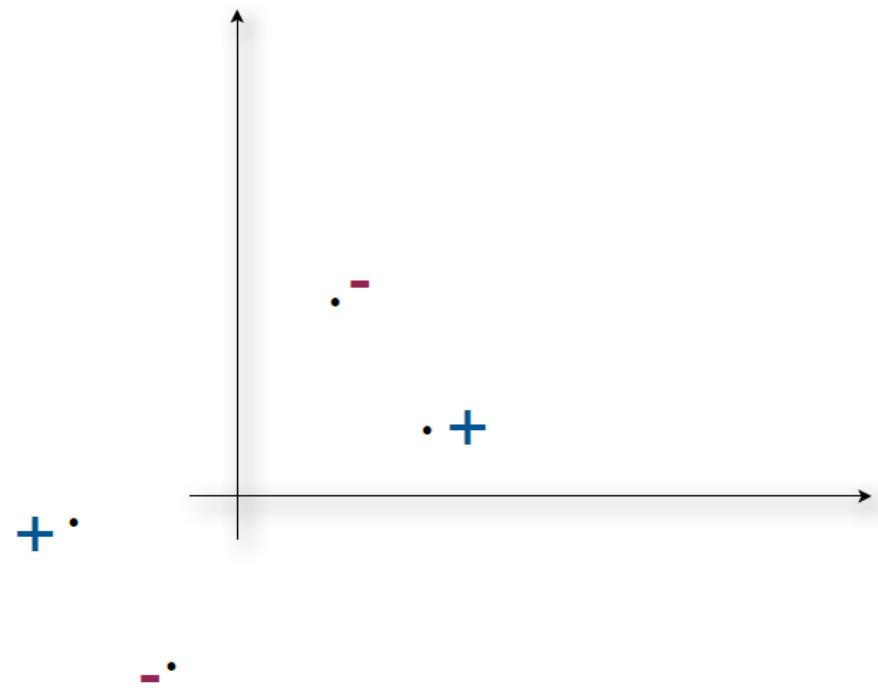
NOT LINEARLY SEPARABLE

Challenge

How do you prove
the training data
on the right is not
linearly separable?

Hint

Draw a line between
the points labelled $+1$,
and a line between
the points labelled -1 .



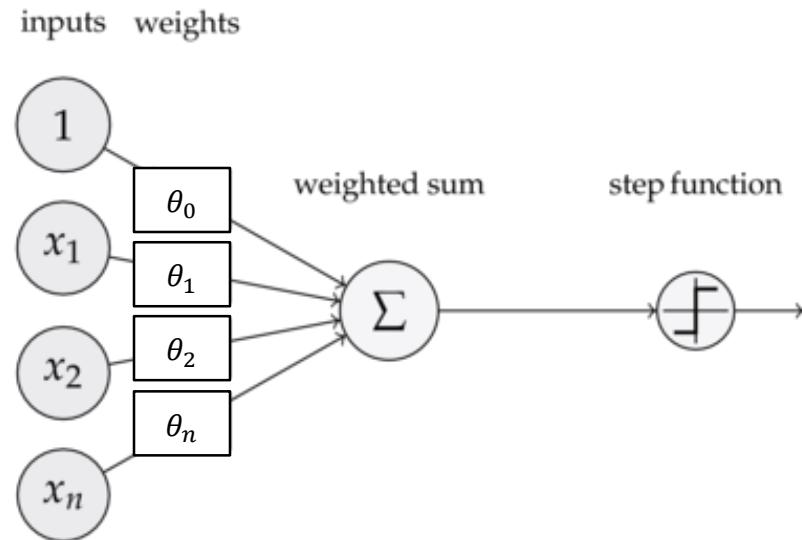


PERCEPTRON ALGORITHM



PERCEPTRON

Linear classifiers are often also called **perceptrons**.



Perceptrons (1957) were designed to resemble neurons.



The hypothesis function is given by

$$h_{\theta} (x^{(i)}) = \text{sgn} (\langle \theta, x^{(i)} \rangle),$$

where

$$\text{sgn}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0. \end{cases}$$

Perceptron criterion

- Using the count of the number of misclassified points as a loss function is not good as this is a piecewise constant, which means that its gradient is zero almost everywhere and gradient descent methods will not work.
- Instead we define the perceptron criterion:

$$\mathcal{L}_P(\theta) = - \sum_{i \in \mathcal{M}} y^{(i)} \langle \theta, x^{(i)} \rangle,$$

where \mathcal{M} is the set of misclassified points.

- Note that this is a piecewise-linear function.

Perceptron algorithm

- Step 1: pick a point $x^{(i)}$ and check if $y^{(i)} \langle \theta(t), x^{(i)} \rangle \geq 0$.
- Step 2: if yes, do nothing; if no perform the following update rule:

$$\begin{aligned}\theta(t+1) &= \theta(t) - \alpha \nabla \mathcal{L}_P(\theta(t)) \\ &= \theta(t) + \alpha x^{(i)} y^{(i)}\end{aligned}$$

- Cycle through the rest of the data with steps 1 and 2.

- The update rule reduces the error with respect to the selected point because

$$\begin{aligned}-y^{(i)} \langle \theta(t+1), x^{(i)} \rangle &= -y^{(i)} \langle \theta(t), x^{(i)} \rangle - y^{(i)} \langle \alpha x^{(i)} y^{(i)}, x^{(i)} \rangle \\ &< -y^{(i)} \langle \theta(t), x^{(i)} \rangle\end{aligned}$$

since $\alpha \|y^{(i)} x^{(i)}\|^2 > 0$.

- However, this does not guarantee that the total error function is reduced at each stage as:
 - the contribution to the error from other misclassified points may have increased;
 - previously correctly classified points may have become misclassified.

* proof not in syllabus

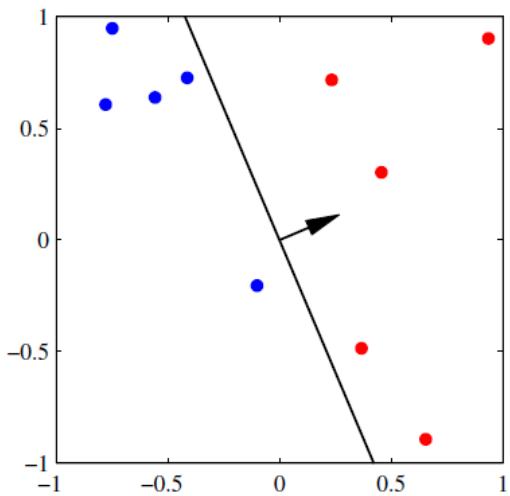
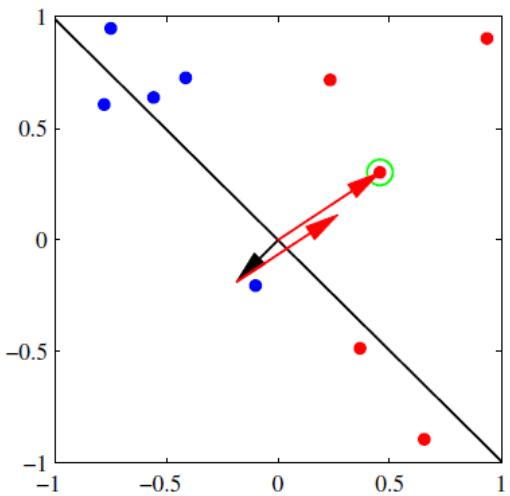
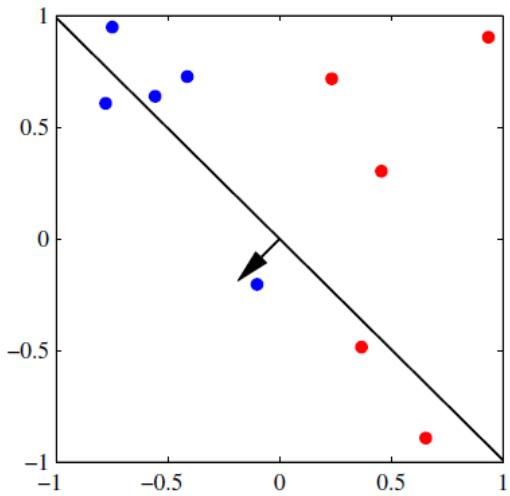
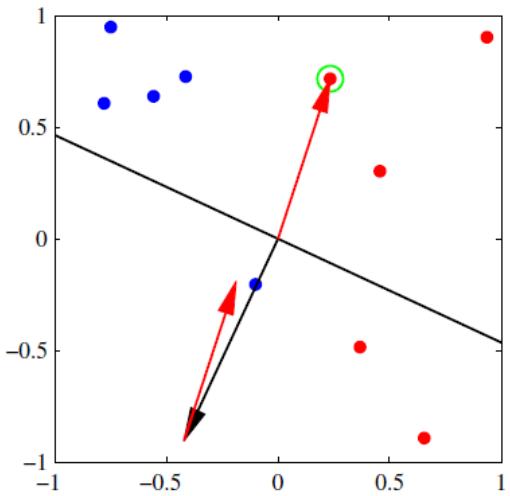
PERCEPTRON CONVERGENCE

Theorem. If the training data is linearly separable, then the perceptron algorithm terminates after a finite number of steps.

Proof. Not in the scope of this class, but it is not difficult. Basic idea is that with every mistake, lower bound for $\|\theta\|$ grows quickly but upper bound grows slowly. Eventually it must stop.

Non linearly-separable. In this case, the perceptron algorithm will never terminate, because there will always be a mistake for all values of θ . Other learning algorithms are needed.





Disadvantages of the perceptron

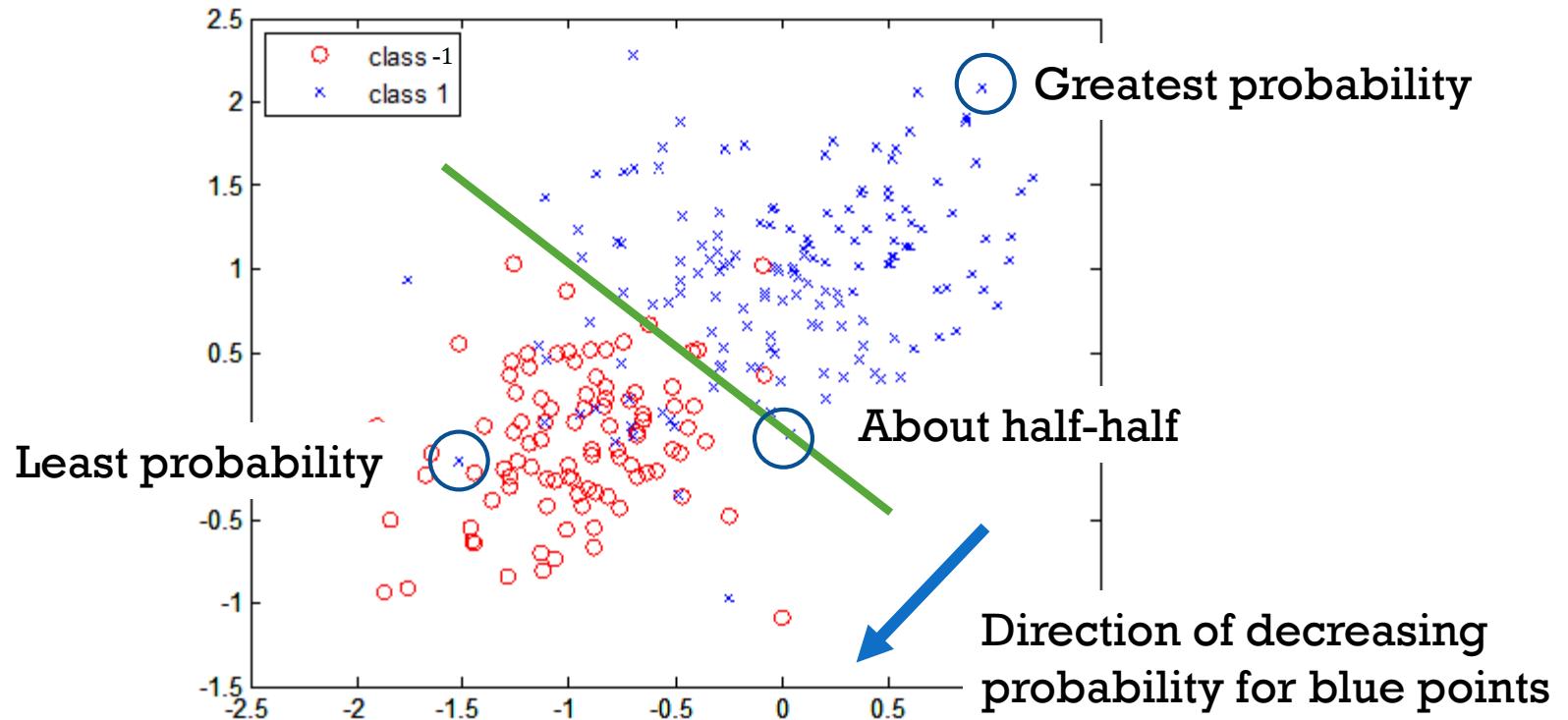
- Learning algorithm difficulties:
 - Not easy to differentiate slow convergence from cases where there will be no convergence due to not having linear separability.
 - Different initialization of the parameters and presentation of the data lead to different solutions.
- Does not provide probabilistic outputs.
- Does not generalize well to more than two classes.

5 min break



LOGISTIC REGRESSION

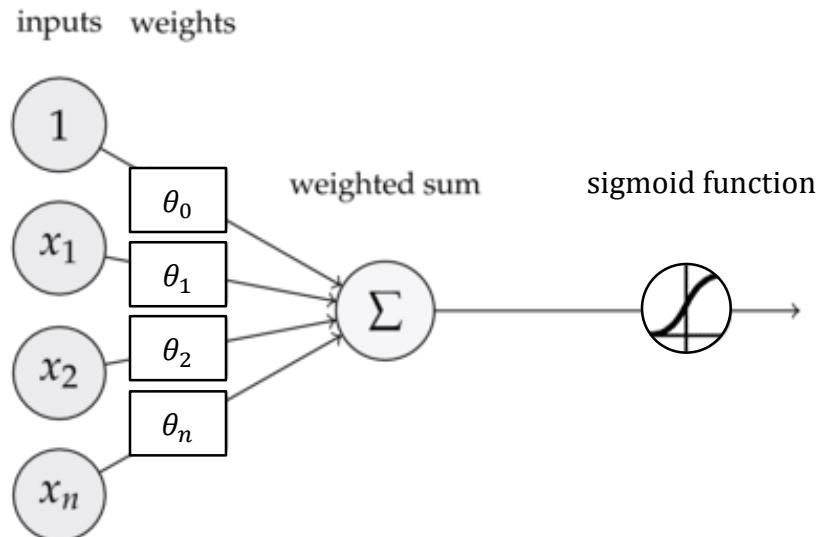




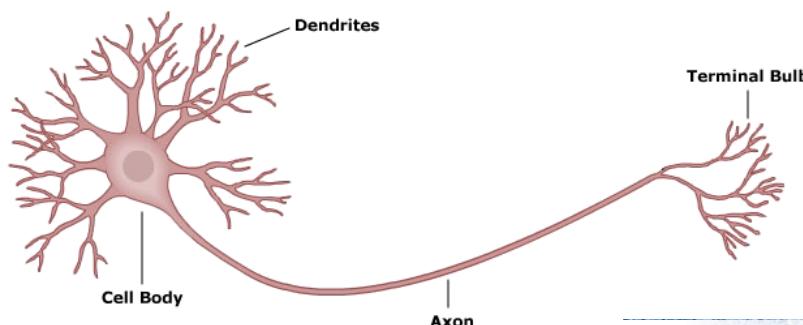
SIGMOID NEURONS

Model consists of
sigmoid neurons.

They were popular
in the early days of
deep learning (2006).



— A Typical Neuron —



PROBABILISTIC MODEL

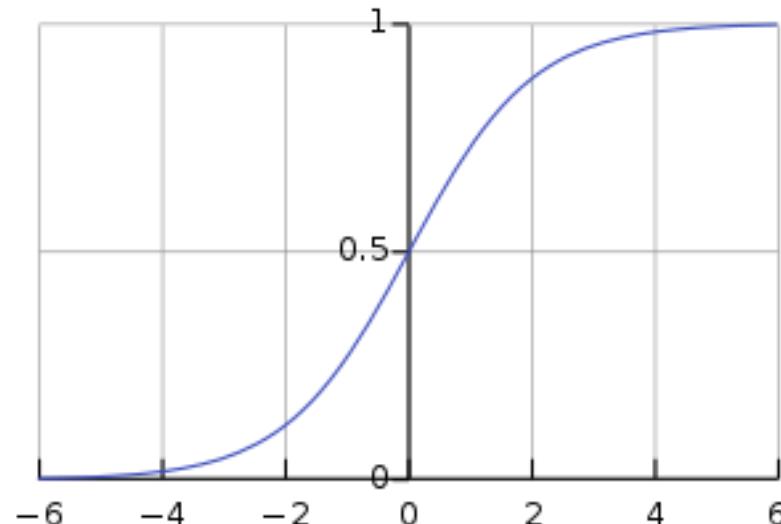
[0, 1] denotes the interval
 $\{a \in \mathbb{R} : 0 \leq a \leq 1\}$

Model the probability that the label y is +1 given the feature is x .

$$h: \mathbb{R}^d \rightarrow [0, 1]$$

$$h(x; \theta) = \mathbb{P}(y = +1 | x) = \text{sigmoid}(\theta^\top x)$$

For small $\theta^\top x$,
the label is very
likely to be -1.



For large $\theta^\top x$,
the label is very
likely to be +1.



Logistic regression

- The formula for the sigmoid function $\sigma(z)$ is given by

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

and the hypothesis function is thus

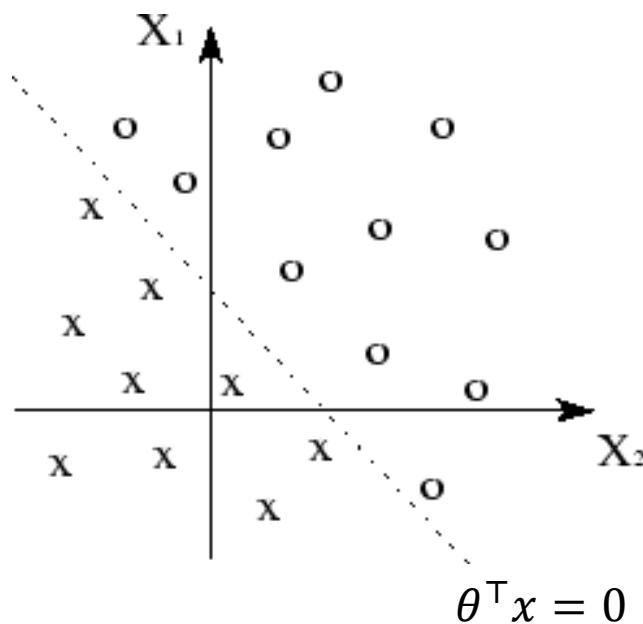
$$h_{\theta}(x^{(i)}) = \sigma(\langle \theta, x^{(i)} \rangle) = \frac{1}{1 + e^{-\langle \theta, x^{(i)} \rangle}}.$$

- The sigmoid function is also known as the logistic function.

DECISION BOUNDARY

$$h(x; \theta) \geq \frac{1}{2} \Leftrightarrow \text{sigmoid}(\theta^\top x) \geq \frac{1}{2} \Leftrightarrow \theta^\top x \geq 0$$

$$h(x; \theta) < \frac{1}{2} \Leftrightarrow \text{sigmoid}(\theta^\top x) < \frac{1}{2} \Leftrightarrow \theta^\top x < 0$$



The decision boundary
is described by $\theta^\top x = 0$.



Formulas for the sigmoid function

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

(ii)
$$\begin{aligned}\sigma'(z) &= \frac{e^{-z}}{(1 + e^{-z})^2} \\ &= \left(\frac{1}{1 + e^{-z}} \right) \left(\frac{e^{-z}}{1 + e^{-z}} \right) \\ &= \left(\frac{1}{1 + e^{-z}} \right) \left(1 - \frac{1}{1 + e^{-z}} \right) \\ &= \sigma(z)(1 - \sigma(z)) = \sigma(z)\sigma(-z)\end{aligned}$$

- With $z^{(i)}$ denoting $\langle \theta, x^{(i)} \rangle$, we have find loglikelihood

$$p_\theta(y = 1 \mid x^{(i)}) = \sigma(z^{(i)}),$$

which means that

$$p_\theta(y = 0 \mid x^{(i)}) = 1 - \sigma(z^{(i)}) = \sigma(-z^{(i)}).$$

- We can combine both expressions as

$$p_\theta(y \mid x^{(i)}) = \sigma(z^{(i)})^{y^{(i)}} \sigma(-z^{(i)})^{1-y^{(i)}}.$$

Log-likelihood function

The log-likelihood function is thus

$$\begin{aligned}\ell(\theta) &= \log \prod_{i=1}^m \sigma(z^{(i)})^{y^{(i)}} \sigma(-z^{(i)})^{1-y^{(i)}} \\ &= \sum_{i=1}^m y^{(i)} \log \left(\sigma(z^{(i)}) \right) + (1 - y^{(i)}) \log \left(\sigma(-z^{(i)}) \right).\end{aligned}$$

The gradient of this loss function is

$$\begin{aligned}\frac{\partial \ell(\theta)}{\partial \theta_j} &= \sum_{i=1}^m \frac{y^{(i)} x_j^{(i)}}{\sigma(z^{(i)})} \sigma(z^{(i)}) \sigma(-z^{(i)}) - \frac{(1-y^{(i)}) x_j^{(i)}}{\sigma(-z^{(i)})} \sigma(z^{(i)}) \sigma(-z^{(i)}) \\ &= \sum_{i=1}^m y^{(i)} x_j^{(i)} \sigma(-z^{(i)}) - (1-y^{(i)}) x_j^{(i)} \sigma(z^{(i)}) \\ &= \sum_{i=1}^m x_j^{(i)} [y^{(i)} (1 - \sigma(z^{(i)})) - (1-y^{(i)}) \sigma(z^{(i)})] \\ &= \sum_{i=1}^m x_j^{(i)} (y^{(i)} - \sigma(z^{(i)})).\end{aligned}$$

- We can then use the gradient to perform gradient ascent to maximize the likelihood:

$$\theta_j(t+1) = \theta_j(t) + \alpha \sum_{i=1}^m \left(y^{(i)} - \sigma \left(\langle \theta(t), x^{(i)} \rangle \right) \right) x_j^{(i)}, \quad j = 1, \dots, n.$$

- Do you see the similarity with linear regression?

Softmax classification

- $Y = \{1, 2, \dots, n\}$
- Given an input $x^{(i)}$, the softmax function which outputs the likelihood of Y being in class j is

$$p(Y = j \mid x^{(i)}) = \frac{e^{\langle \theta_j, x^{(i)} \rangle}}{\sum_{k=1}^n e^{\langle \theta_k, x^{(i)} \rangle}}$$

- Softmax classification is also known as multinomial logistic regression.
- The term "softmax" comes from the fact that if we introduce a parameter c into the formula in the previous slide, i.e. if

$$p(Y = j \mid x^{(i)}) = \frac{e^{c\langle \theta_j, x^{(i)} \rangle}}{\sum_{k=1}^n e^{c\langle \theta_k, x^{(i)} \rangle}},$$

then the softmax function converges to a max function as $c \rightarrow \infty$.

Overparameterized

- Softmax classification for n classes does not actually require n weight vectors $\theta_1, \dots, \theta_n$:

$$\begin{aligned} p(Y = j \mid x^{(i)}) &= \frac{e^{\langle \theta_j, x^{(i)} \rangle}}{\sum_{k=1}^n e^{\langle \theta_k, x^{(i)} \rangle}} \\ &= \frac{1}{1 + \sum_{k \neq j} e^{\langle \theta_k - \theta_j, x^{(i)} \rangle}} \\ &= \frac{1}{1 + \sum_{k \neq j} e^{-\langle \tilde{\theta}_k, x^{(i)} \rangle}}, \end{aligned}$$

where we denote $\tilde{\theta}_k$ as $\theta_j - \theta_k$. Hence, only $n - 1$ weight vectors are needed.

- In fact, in the 2-class case, this is precisely the formula for the sigmoid function, as thus softmax classification is a generalization of logistic regression:

$$p(Y = 1 \mid x^{(i)}) = \frac{e^{\langle \theta_1, x^{(i)} \rangle}}{e^{\langle \theta_1, x^{(i)} \rangle} + e^{\langle \theta_2, x^{(i)} \rangle}} = \frac{1}{1 + e^{-\langle \theta_1 - \theta_2, x^{(i)} \rangle}}$$

$$\begin{aligned} p(Y = 2 \mid x^{(i)}) &= \frac{e^{\langle \theta_2, x^{(i)} \rangle}}{e^{\langle \theta_1, x^{(i)} \rangle} + e^{\langle \theta_2, x^{(i)} \rangle}} = \frac{1}{1 + e^{-\langle \theta_2 - \theta_1, x^{(i)} \rangle}} \\ &= 1 - \frac{1}{1 + e^{-\langle \theta_1 - \theta_2, x^{(i)} \rangle}} \end{aligned}$$

Log-likelihood

$$\begin{aligned}\ell(\theta) &= \sum_{i=1}^m \log p(Y = y^{(i)} \mid x^{(i)}) \\ &= \sum_{i=1}^m \log \frac{e^{\langle \theta_{y^{(i)}}, x^{(i)} \rangle}}{\sum_{k=1}^n e^{\langle \theta_k, x^{(i)} \rangle}} \\ &= \sum_{i=1}^m \langle \theta_{y^{(i)}}, x^{(i)} \rangle - \log \sum_{k=1}^n e^{\langle \theta_k, x^{(i)} \rangle}\end{aligned}$$

Gradient of softmax log-likelihood

$$\begin{aligned}\nabla_{\theta_p} \ell(\theta) &= \sum_{i=1}^m \delta_{p y^{(i)}} x^{(i)} - \nabla_{\theta_p} \log \sum_{k=1}^n e^{\langle \theta_k, x^{(i)} \rangle} \\&= \sum_{i=1}^m \delta_{p y^{(i)}} x^{(i)} - \frac{\nabla_{\theta_p} \sum_{k=1}^n e^{\langle \theta_k, x^{(i)} \rangle}}{\sum_{k=1}^n e^{\langle \theta_k, x^{(i)} \rangle}} \\&= \sum_{i=1}^m \delta_{p y^{(i)}} x^{(i)} - \frac{e^{\langle \theta_p, x^{(i)} \rangle} x^{(i)}}{\sum_{k=1}^n e^{\langle \theta_k, x^{(i)} \rangle}} \\&= \sum_{i=1}^m \left(\delta_{p y^{(i)}} - p(Y = p \mid x^{(i)}) \right) x^{(i)}\end{aligned}$$

DEEP LEARNING



DEFINITIONS

ARTIFICIAL INTELLIGENCE

Early artificial intelligence stirs excitement.



MACHINE LEARNING

Machine learning begins to flourish.



DEEP LEARNING

Deep learning breakthroughs drive AI boom.



Milestones in AI and Deep Learning

- 1943: First artificial neural networks (McCulloch, Pitts)
- 1956: First conference defining “AI” in Dartmouth (Shannon, Minsky etc.)
- 1986: Training multi-layer neural nets using backpropagation (Hinton, Rumelhart, Williams)
- 1989: Convolutional neural networks (LeCun)
- 1997: Deep blue beats Kasporov in chess
- 2009: Big bang of deep learning, use of GPUs for accelerated training
- 2012: Numerous competitions won with superhuman performance in computer vision using deep CNNs (Krizhevsky, Ciresan etc.)
- 2016: Alphago beats Lee Sedol in Go

THE EXPANDING UNIVERSE OF MODERN AI

"THE BIG BANG"

Big Data
GPU
Algorithms

RESEARCH

Berkeley
Carnegie Mellon University
DEEPMIND
Massachusetts Institute of Technology
NYU
UNIVERSITY OF OXFORD
UNIVERSITY OF TORONTO

CORE TECHNOLOGY / FRAMEWORKS

Preferred Networks
facebook. torch
Université de Montréal
theano
Google TensorFlow
Berkeley Caffe
Microsoft CNTK
UNIVERSITY OF OXFORD cuDNN
NVIDIA cuDNN

AI-as-a-PLATFORM

amazon webservices

IBM Watson

Google

Microsoft Azure

START-UPS

api.ai

Personal Assistants
conversational interface

BLUE RIVER TECHNOLOGY

Agriculture
crop-yield optimization

clarifai

Tech
visual recognition platform

deep genomics

Genomics
genetic interpretation

drive.ai

Automotive
AI-as-a-service

SADAKO

Waste Management
sorting robots

MetaMind

eCommerce & Medical
recommendation engines

Morpho

Tech
computer vision

Orbital Insight

Geospatial
predictions from images

nervana

Tech
AI-as-a-service

SocialEyes*

Medical
diabetic retinopathy

HOW ARE YOU?

Education
teaching robots

1,000+ AI START-UPS

\$5B IN FUNDING

Source: Venture Scanner

INDUSTRY LEADERS

Ford

target

SIEMEN

GE

TESL

gsk

TOYO

Audi

Baidu

THE HOME DEPOT

Bloomberg

MASSACHUSETTS GENERAL HOSPITAL

UBER

Mercedes-Benz

charles SCHWAB

VOLVO

CISCO

MERCK

Pinterest

Walmart

ebay

YAHOO

Schlumberger

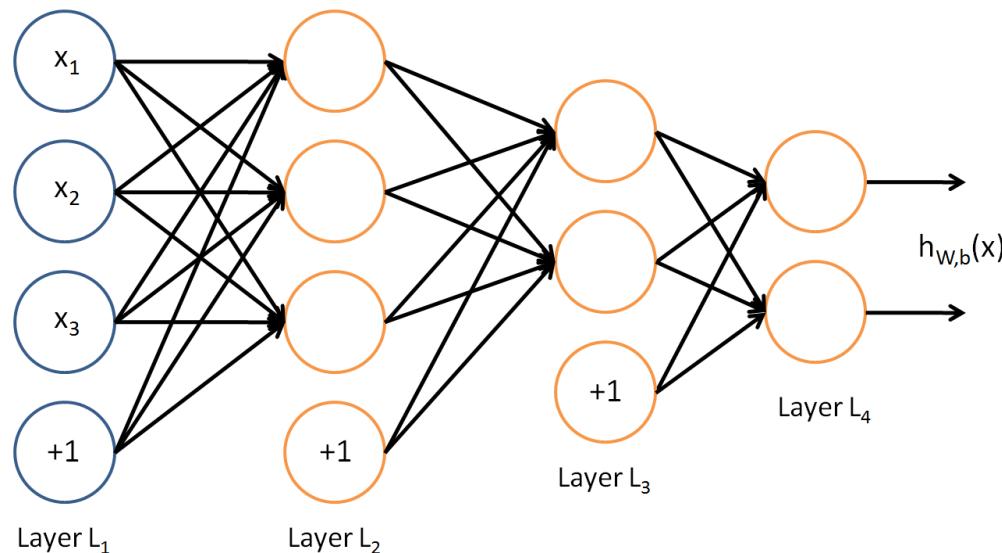
Yandex

PEPSICO

yelp

WHAT IS DEEP LEARNING?

Biologically-inspired multilayer neural networks

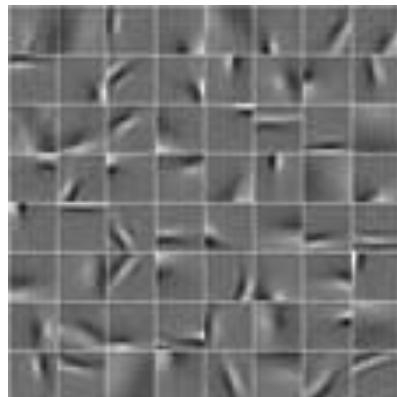


Both supervised and unsupervised

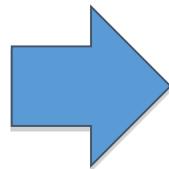


WHAT IS DEEP LEARNING?

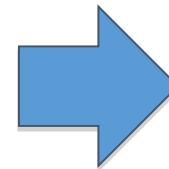
Example. Face recognition (Facebook)



Edges



Eyes, Noses, Mouths



Faces

Deeper layers learn higher-order features



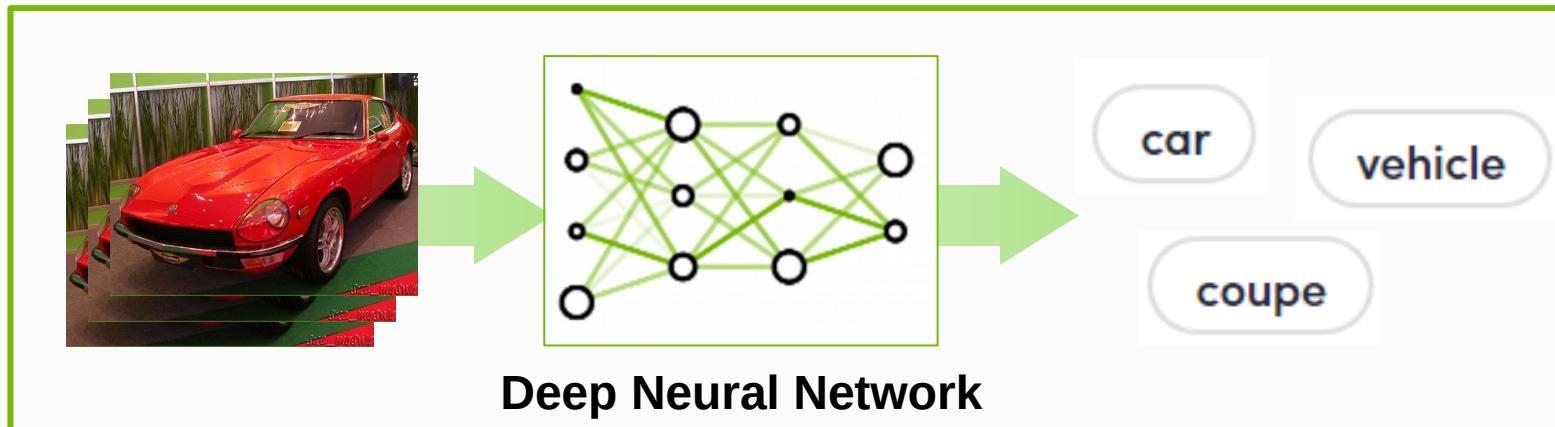
A NEW COMPUTING MODEL

Algorithms that Learn from Examples



Traditional Approach

- Requires domain experts
- Time consuming
- Error prone
- Not scalable to new problems



Deep Learning Approach

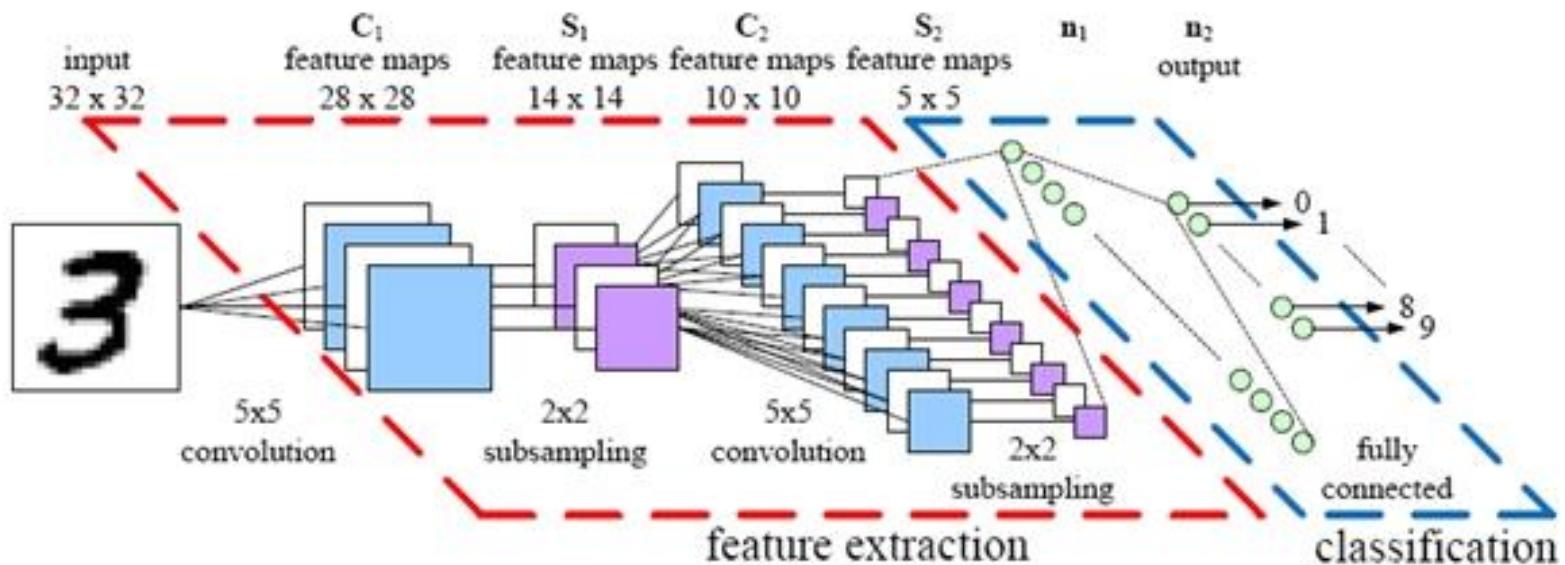
- ✓ Learn from data
- ✓ Easily to extend
- ✓ Speedup with GPUs



APPLICATIONS



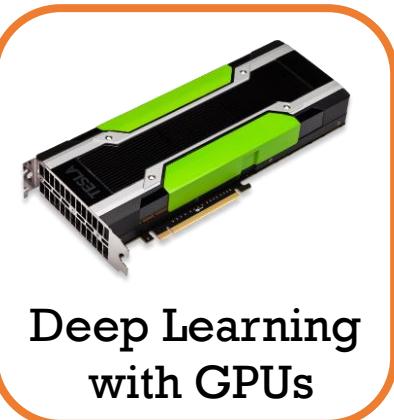
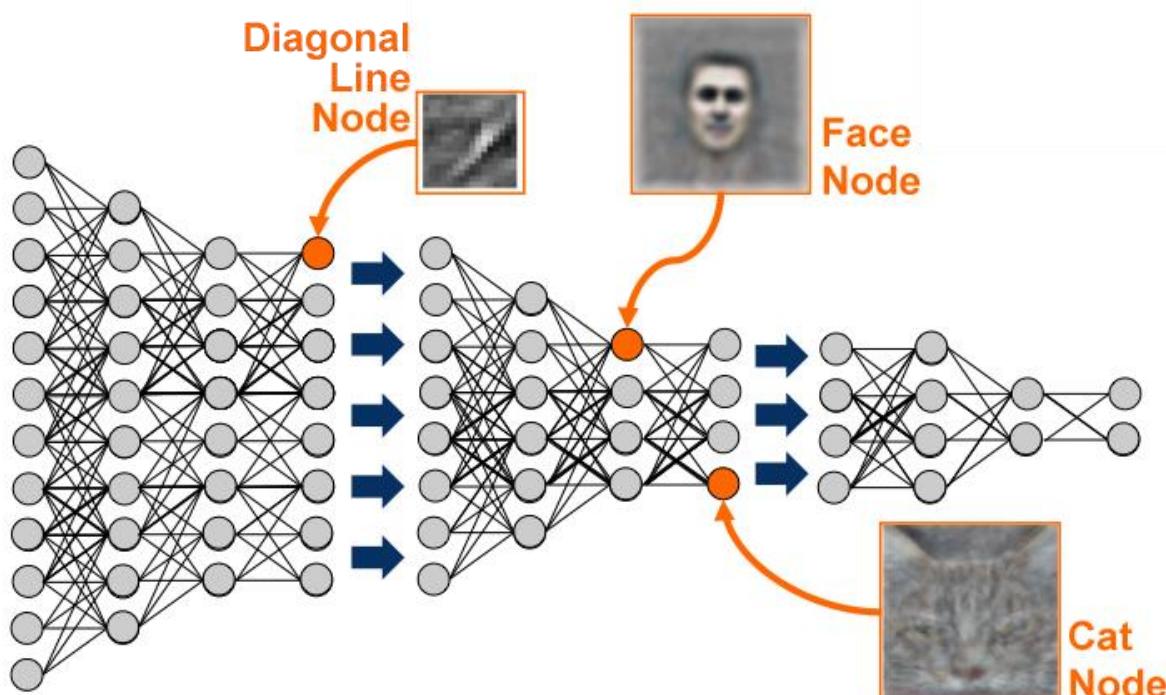
HANDWRITING RECOGNITION



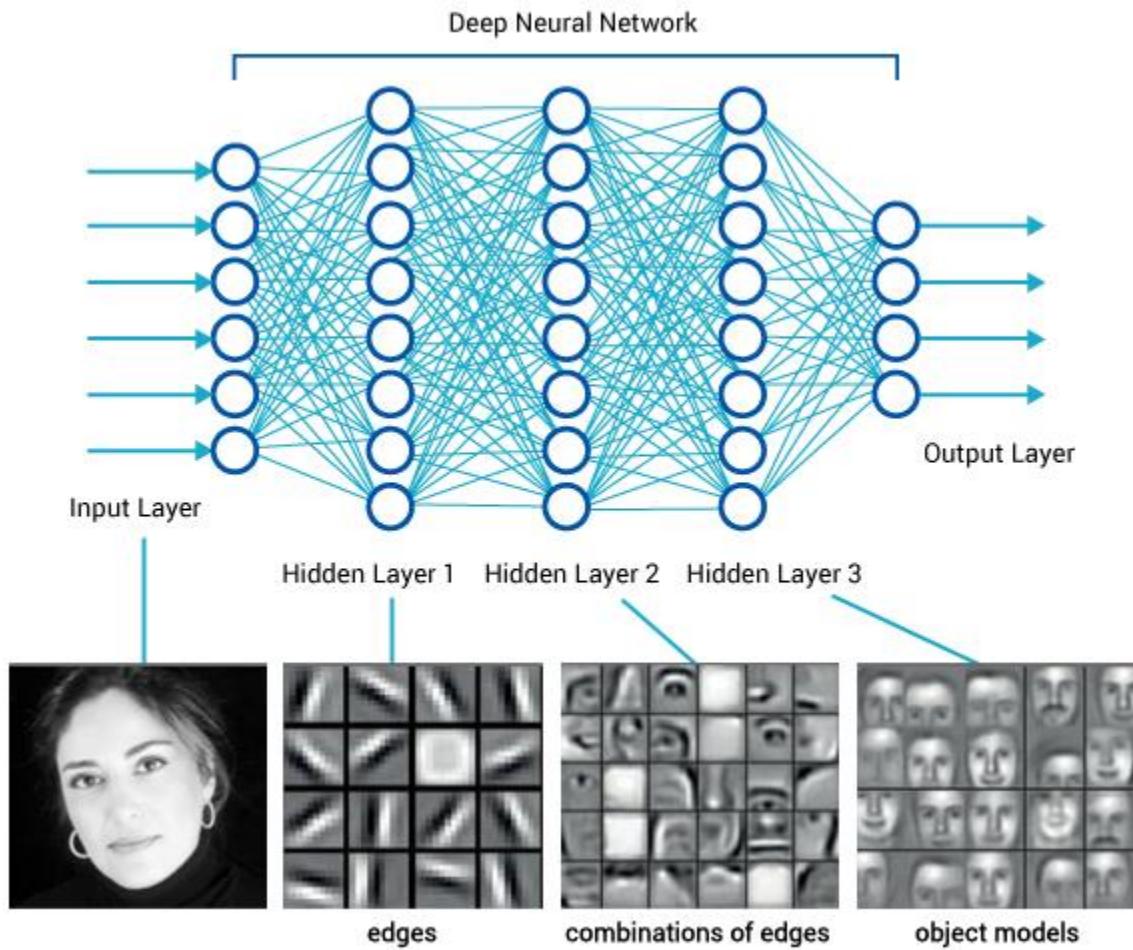
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9



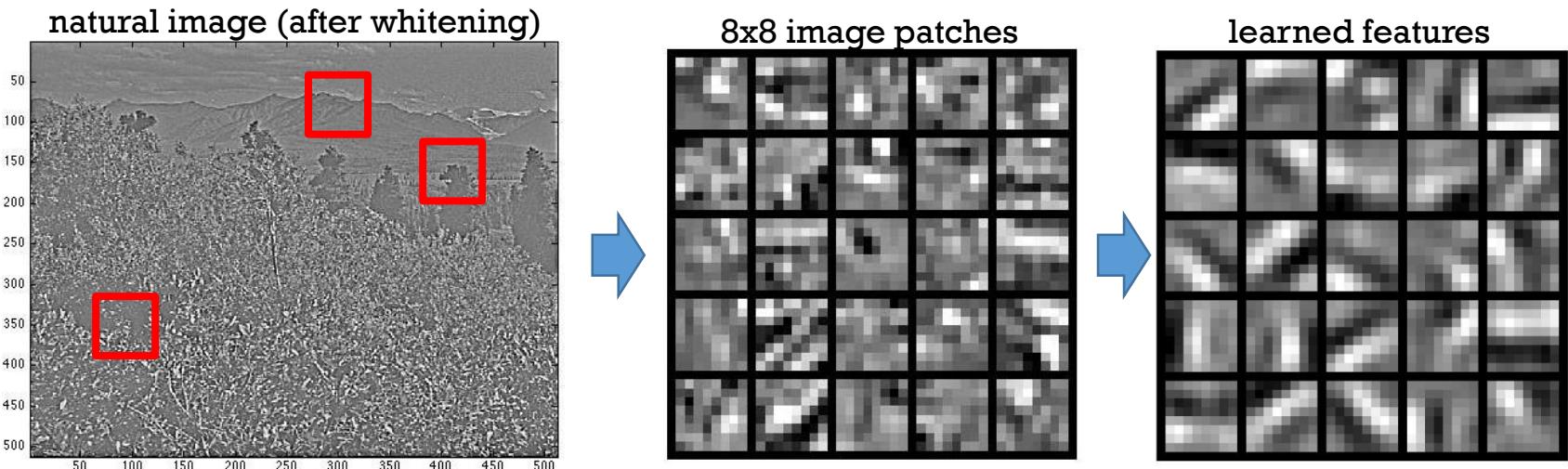
GOOGLE CAT VIDEOS



FACE RECOGNITION



NATURAL IMAGES



Edge features similar to those from neuroscience experiments
(see Hubel & Wiesel Cat Experiment, 1959).



SPEECH TRANSLATION



From Hidden Markov Models to Recurrent Neural Networks

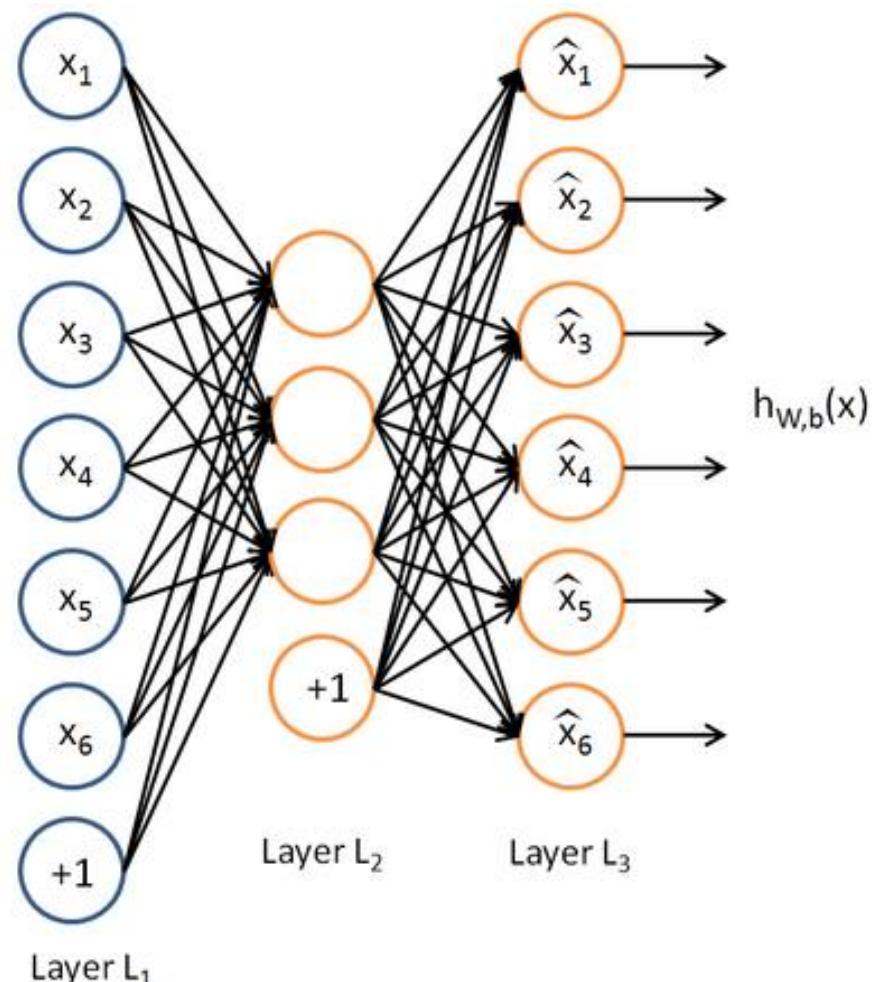


AUTOENCODERS

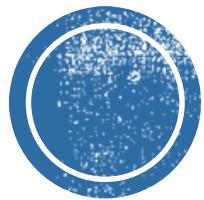
Training a multilayer neural network to reconstruct the input from a **reduced representation**.

Strategies for Dimensionality Reduction

- Few hidden neurons
- Sparse activations



5 min break

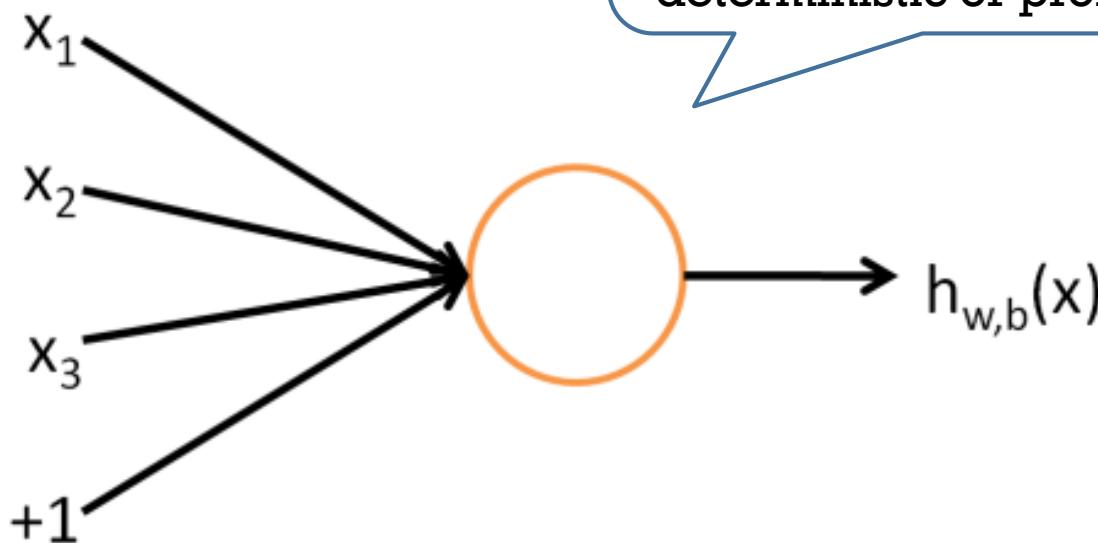


FEEDFORWARD NETWORKS



NEURON

Perceptron.

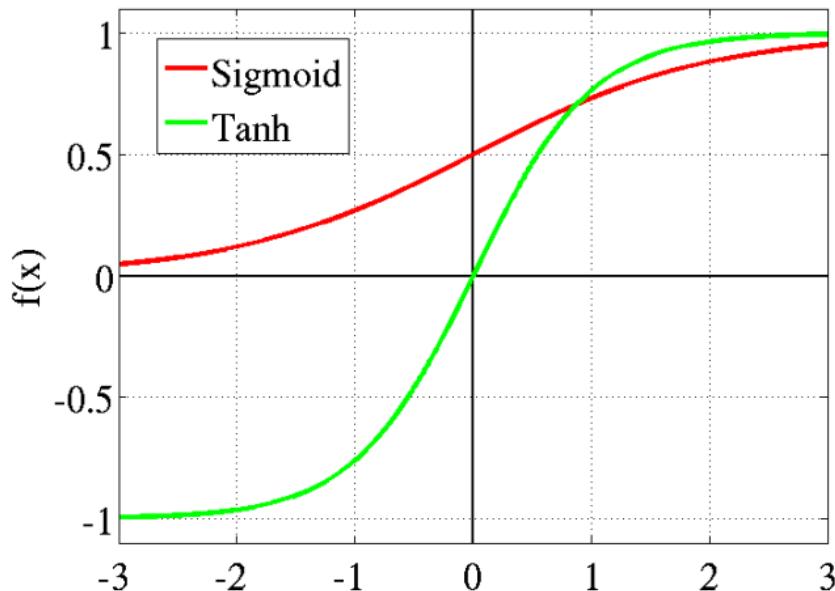


Depending on function f ,
neurons can be:
real-valued or binary-valued;
deterministic or probabilistic.

$$h_{w,b}(x) = f(w^\top x) = f\left(\sum_{i=1}^d w_i x_i + b\right)$$

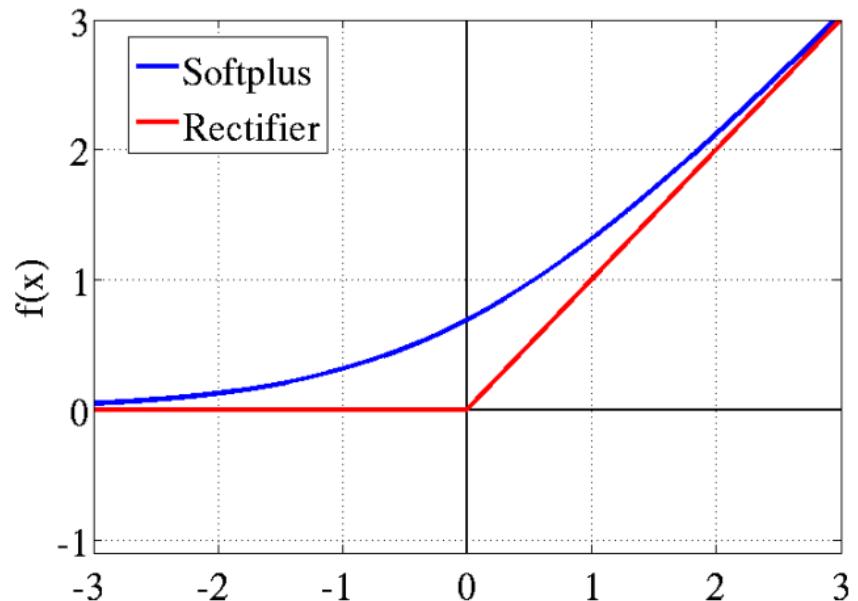


ACTIVATION FUNCTIONS



$$\text{sigmoid } f(z) = \frac{1}{1+e^{-z}}$$

$$\tanh f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

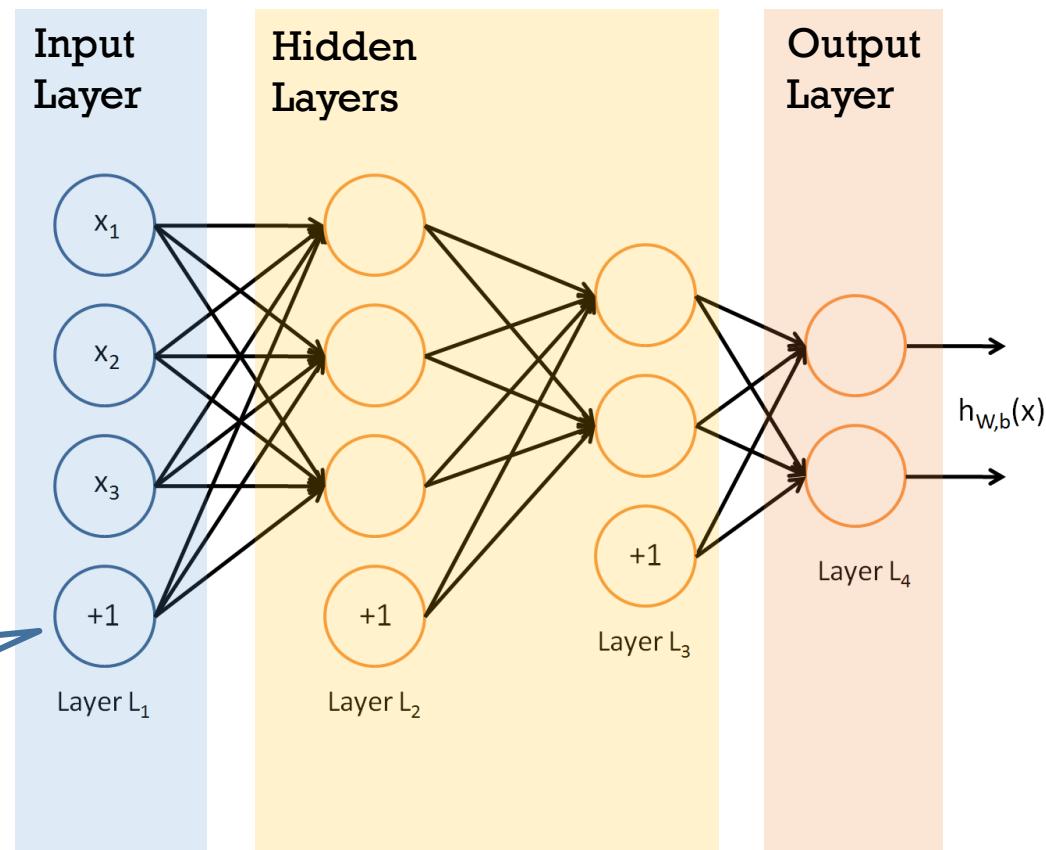


$$\text{softplus } f(z) = \ln(1 + e^{-z})$$

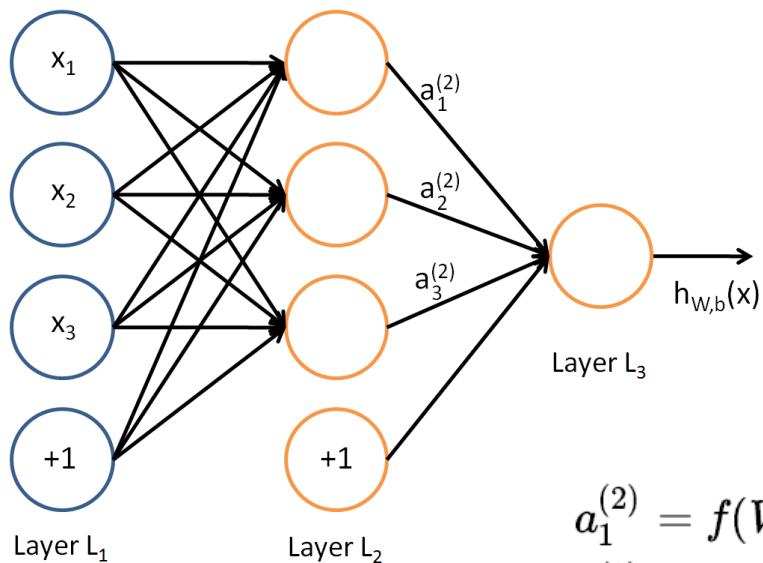
$$\begin{aligned} &\text{rectified} \\ &\text{linear unit} \\ &(\text{ReLU}) \end{aligned}$$
$$f(z) = \max(0, z)$$



MULTI-LAYER NEURAL NETWORK



MULTI-LAYER NEURAL NETWORK



$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$

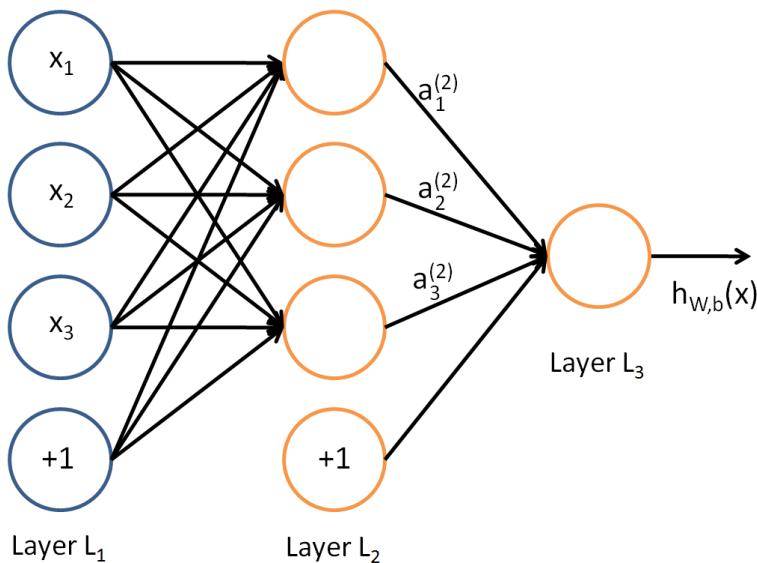
$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$

$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$

$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})$$



MULTI-LAYER NEURAL NETWORK



Forward Propagation.

$$z^{(2)} = W^{(1)}x + b^{(1)}$$

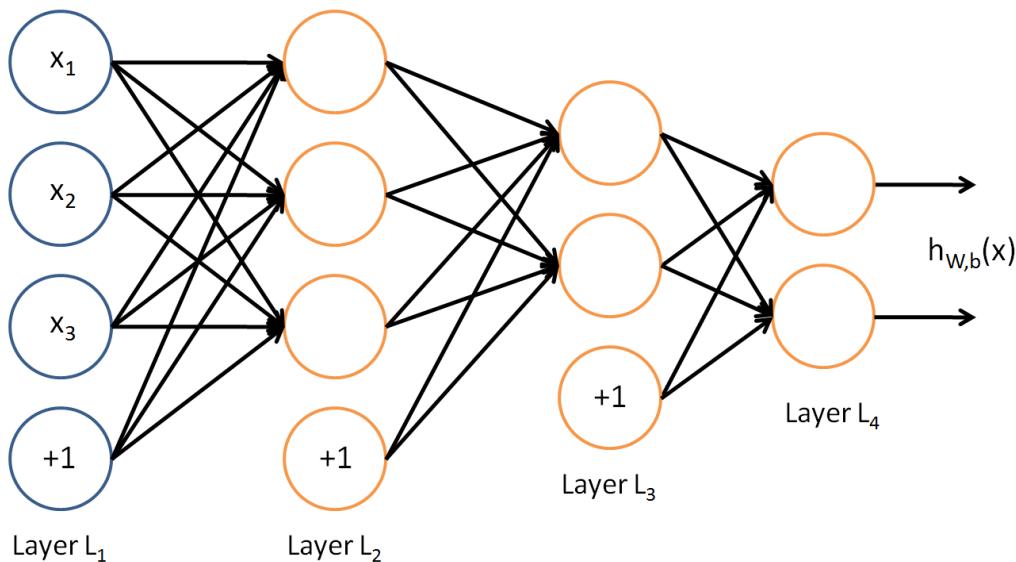
$$a^{(2)} = f(z^{(2)})$$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$

$$h_{W,b}(x) = a^{(3)} = f(z^{(3)})$$



MULTI-LAYER NEURAL NETWORK



Feedforward Neural Network.

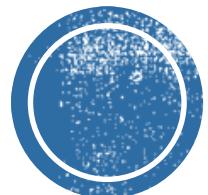
$$z^{(l+1)} = W^{(l)} a^{(l)} + b^{(l)}$$
$$a^{(l+1)} = f(z^{(l+1)})$$

Activation

Neural Network Architecture.

Arrangement of neurons, e.g. number of neurons in each layer.

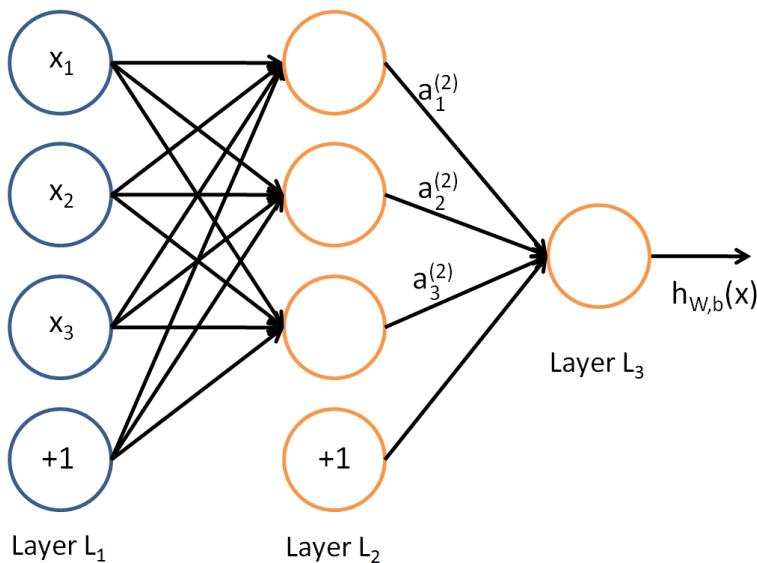




BACKPROPAGATION



MULTI-LAYER NEURAL NETWORK



Forward Propagation.

$$z^{(2)} = W^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$

$$h_{W,b}(x) = a^{(3)} = f(z^{(3)})$$



BACKPROPAGATION

Chain Rule for Neural Networks.

$$\begin{aligned} \frac{\partial}{\partial W_{ij}^{(l)}} \left(\frac{1}{2} \|a^{(n_l)} - y\|^2 \right) &= (a^{(n_l)} - y) \frac{\partial}{\partial W_{ij}^{(l)}} f(z^{(n_l)}) \\ &= (a^{(n_l)} - y) f'(z^{(n_l)}) \frac{\partial}{\partial W_{ij}^{(l)}} (W^{(n_l-1)} a^{(n_l-1)} + b^{(n_l-1)}) \\ &= (a^{(n_l)} - y) f'(z^{(n_l)}) W^{(n_l-1)} \frac{\partial}{\partial W_{ij}^{(l)}} a^{(n_l-1)} \\ &= (a^{(n_l)} - y) f'(z^{(n_l)}) W^{(n_l-1)} f'(z^{(n_l-1)}) \frac{\partial}{\partial W_{ij}^{(l)}} z^{(n_l-1)} \\ &\quad \underbrace{\qquad\qquad\qquad}_{\delta^{(n_l)}} \\ &\quad \underbrace{\qquad\qquad\qquad}_{\delta^{(n_l-1)}} \end{aligned}$$



Backpropagation algorithm

1. Perform forward propagation.
2. Compute δ for the output unit(s), e.g. $\delta = y - t$ in the case of MSE.
3. Backpropagate the δ 's using

$$\delta_j = f'(a_j) \sum_k w_{kj} \delta_k$$

for each hidden unit in the network.

4. Compute the required derivatives using

$$\frac{\partial L}{\partial w_{ji}^{(k)}} = \delta_j z_i.$$

WHAT WAS WRONG WITH BACKPROPAGATION IN 1986?



1. Our labeled datasets were thousands of times too small.
2. Our computers were millions of times too slow.
3. We initialized the weights in a stupid way.
4. We used the wrong type of non-linearity.



Xavier initialization

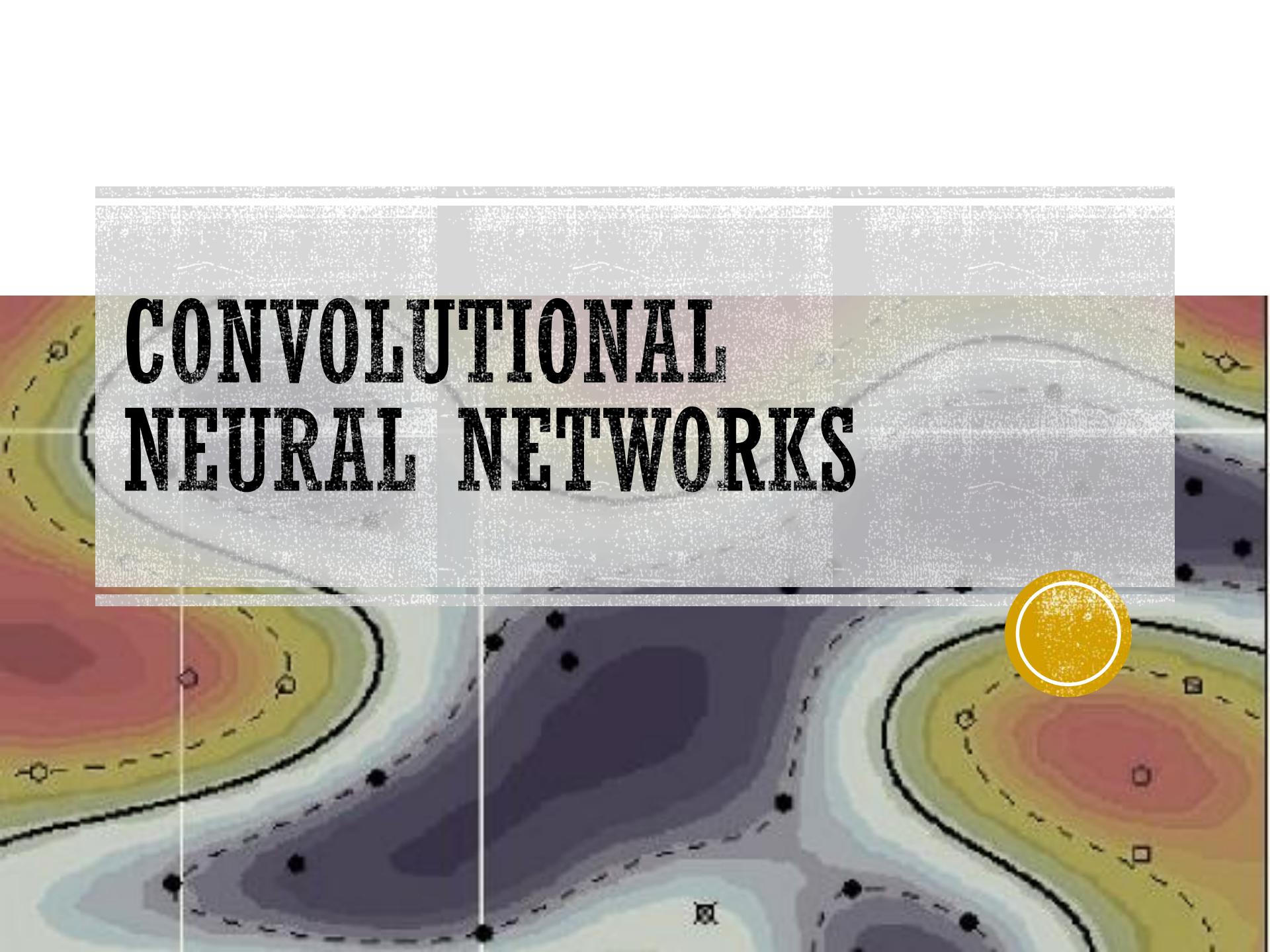
- Initialize weights using

$$w^{(l)} \sim \mathcal{N} \left(0, \sqrt{\frac{2}{n^{(l)} + n^{(l-1)}}} \right),$$

where $n^{(l)}$ is the number of neurons in layer l .

- This makes the variance of the activations in each layer similar to one another.

CONVOLUTIONAL NEURAL NETWORKS

A weather map featuring various colored contour lines (yellow, red, purple) and black dots representing data points. A prominent yellow circle highlights a circular region in the upper right quadrant of the map.

What are convolutional neural networks (CNNs)?

- Category of neural networks particularly effective for image classification
- Deep CNNs extract a hierarchy of features from the input:
 - Lower layers extract local features such as lines and curves
 - Higher layers extract composites of local features; e.g. parts of a face

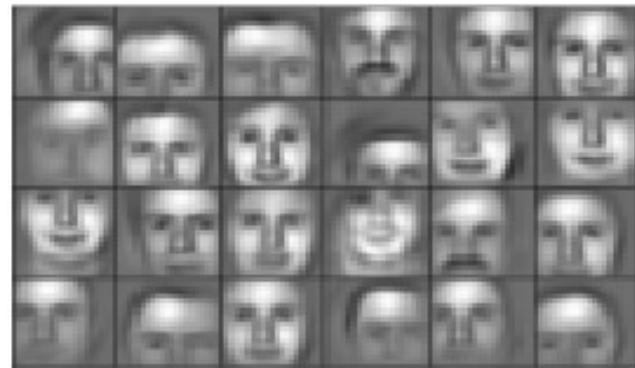
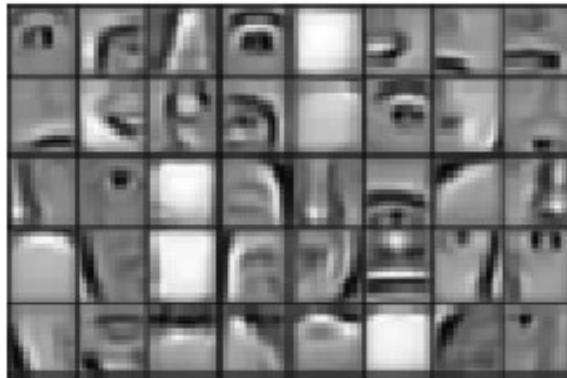
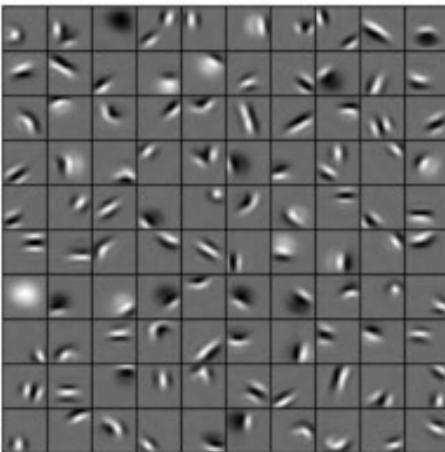
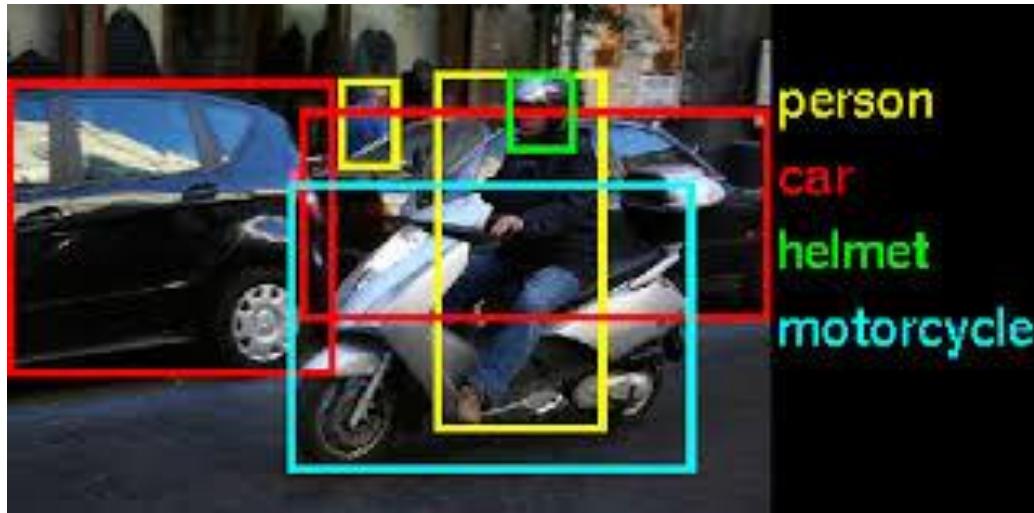


IMAGE RECOGNITION

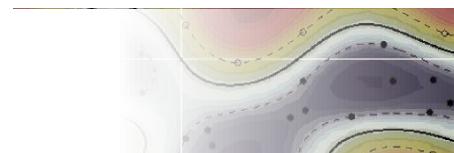


Too many parameters.

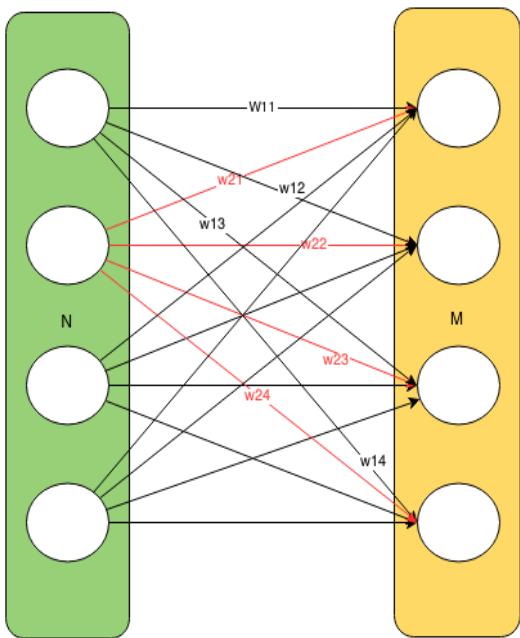
- $(\text{MNIST } 28 \times 28 \text{ pixels}) * (\text{100 features}) = 78,400 \text{ params}$

Translation invariance.

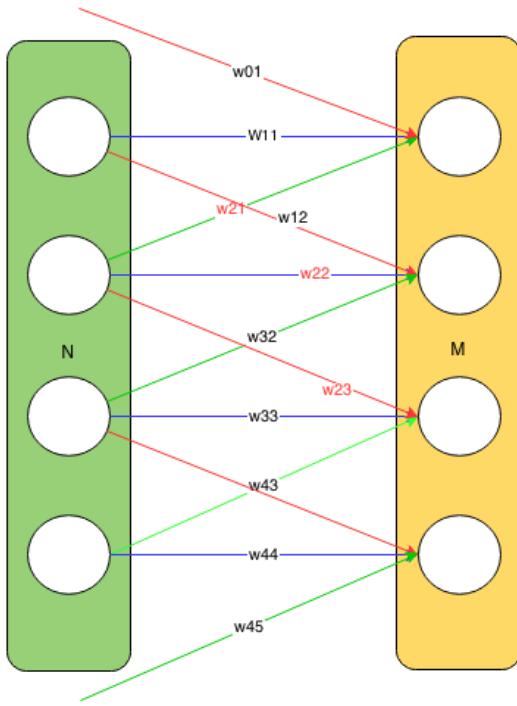
- Statistics of a patch (usually) does not depend on its location.



SHARED WEIGHTS



**Fully-Connected
(Dense)**

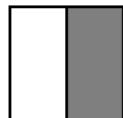


**Locally-Connected
with Shared Weights**

Edge detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

6×6



*

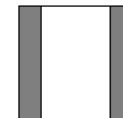
1	0	-1
1	0	-1
1	0	-1

3×3

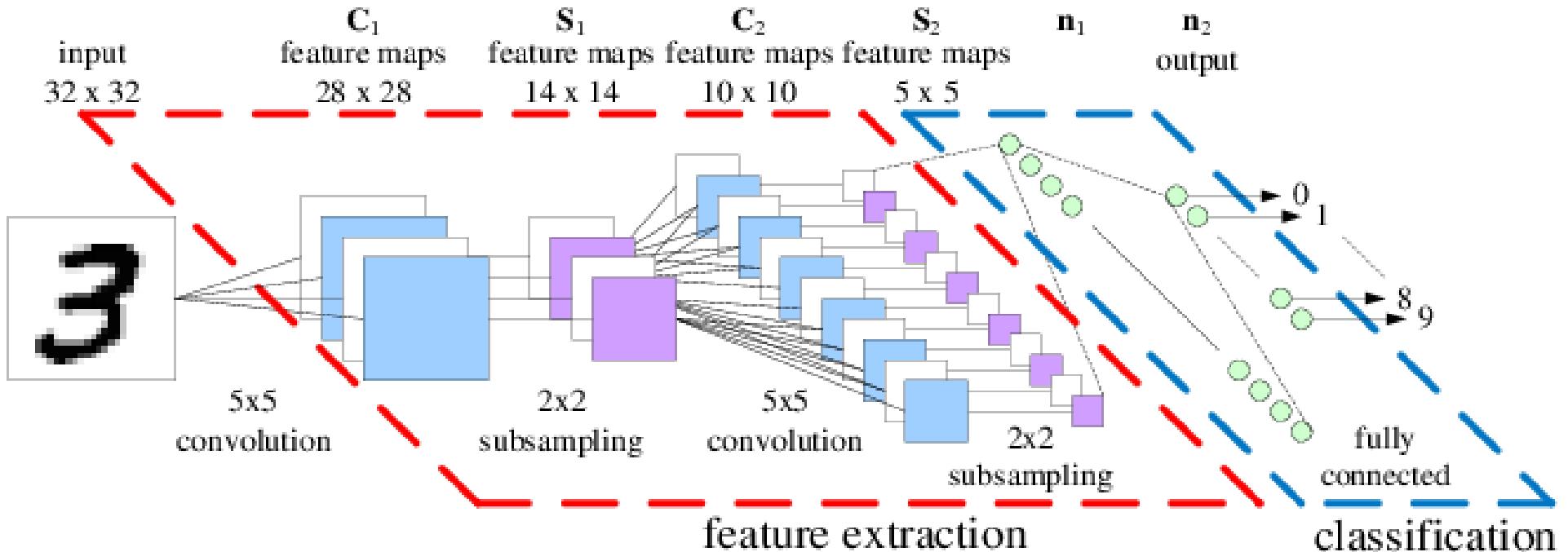
=

-0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

4×4



CNN architecture (e.g. LeNet)



PyTorch code

```
import torch.nn as nn
import torch.nn.functional as F

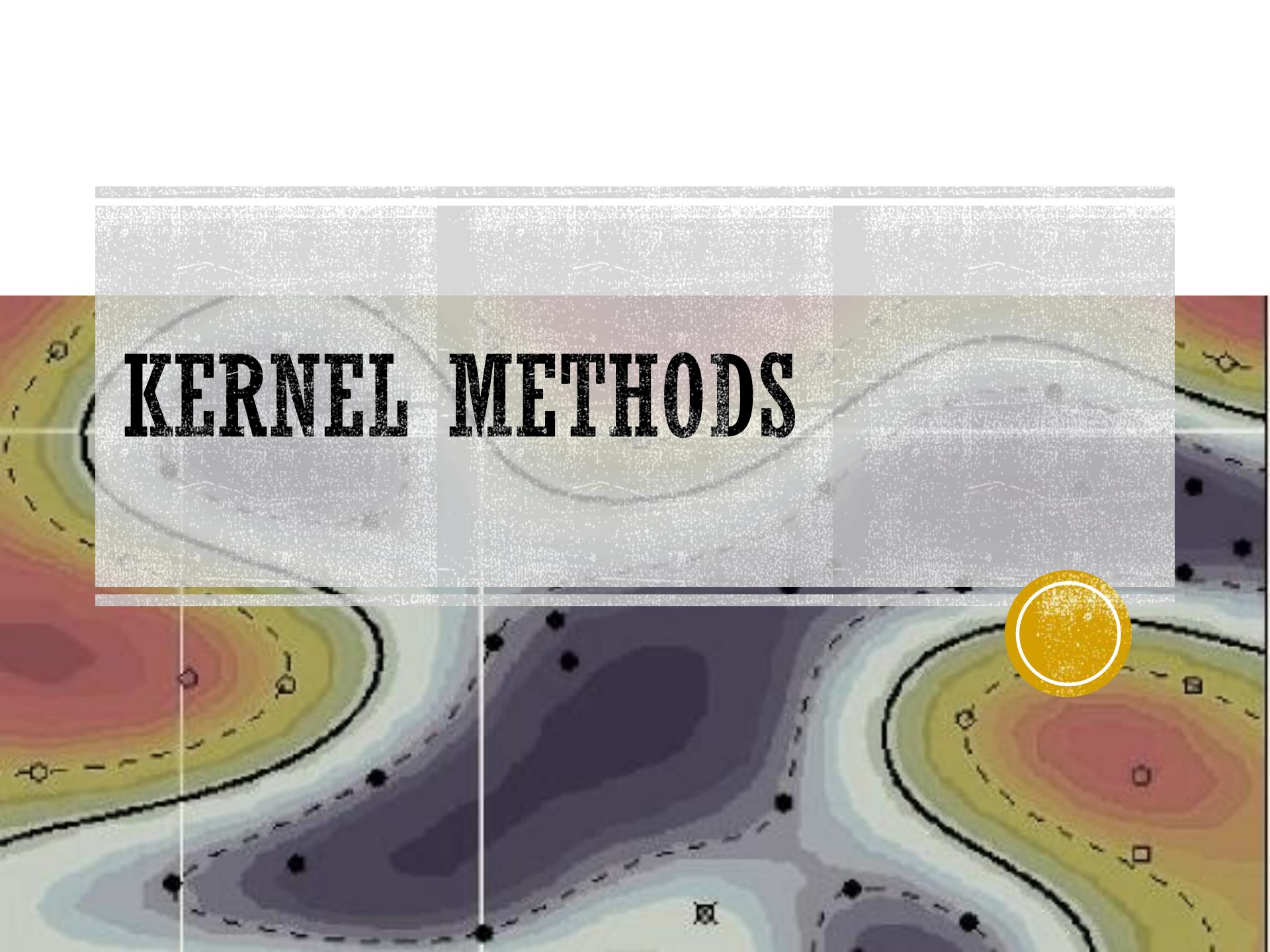
class convNet(nn.Module):
    def __init__(self):
        super(convNet, self).__init__()
        self.conv = nn.Conv2d(in_channels=1, out_channels=20,
kernel_size=5, stride=1, padding=1)

    def forward(self, x):
        x = F.relu(self.conv(x))
        return x
```

Input formats

- NCHW
 - "Number Channels Height Width" or "channels-first"
 - default on pyTorch
 - performs faster on GPUs
- NHWC
 - "Number Height Width Channels" or "channels-last"
 - default on Tensorflow, needed for CPUs

5 min break



KERNEL METHODS

FEATURE MAPS

Example. Non-linear classifiers.

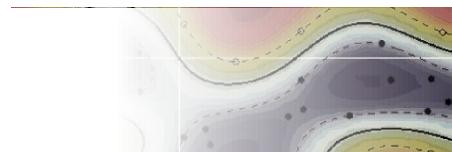
$$x = (x_1, x_2)$$

$$\phi(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2)$$

$$h(x; \theta, \theta_0)$$

$$= \text{sign}(\theta \cdot \phi(x))$$

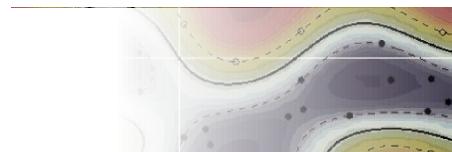
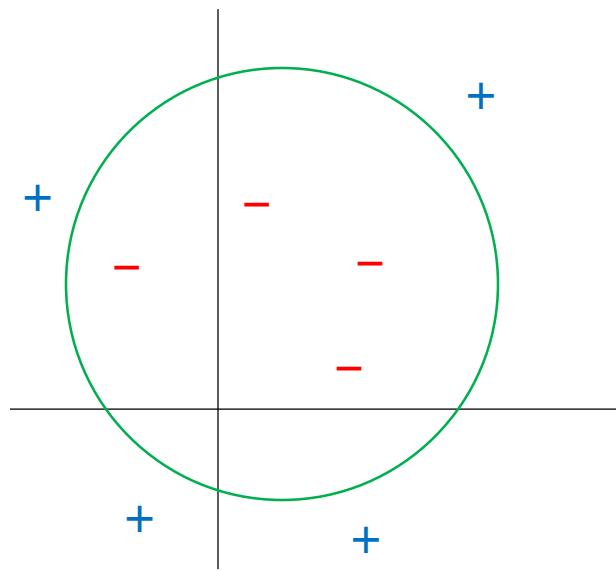
$$= \text{sign}(\theta_1 + \theta_2\sqrt{2}x_1 + \theta_3\sqrt{2}x_2 + \theta_4\sqrt{2}x_1x_2 + \theta_5x_1^2 + \theta_6x_2^2)$$



FEATURE MAPS

Example. Non-linear classifiers.

$$\begin{aligned} h(x; \theta, \theta_0) \\ = \text{sign}\left((x_1 - 1)^2 + (x_2 - 2)^2 - 9\right) \\ = \text{sign}\left(-4 - 2x_1 - 4x_2 + x_1^2 + x_2^2\right) \end{aligned}$$



CHALLENGES

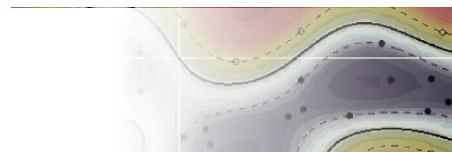
High-Dimensional Features.

$$x = (x_1, x_2, \dots, x_{1000}) \in \mathbb{R}^{1000}$$

$$\phi(x) = (1, \dots, \sqrt{2}x_i, \dots, \sqrt{2}x_i x_j, \dots, x_i^2, \dots) \in \mathbb{R}^{501501}$$

Inner Products.

Computing $\phi(x) \cdot \phi(x')$ for $x, x' \in \mathbb{R}^{1000}$ requires about 2,004,000 floating point operations.

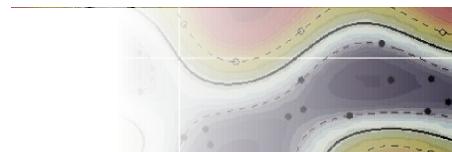


KERNEL FUNCTIONS

Fortunately, many inner products simplify nicely.

$$\begin{aligned} K(x, x') &= \phi(x) \cdot \phi(x') \\ &= 1 + 2 \sum_i x_i x'_i + 2 \sum_{i < j} x_i x_j x'_i x'_j + \sum_i x_i^2 x'^2_i \\ &= 1 + 2(\sum_i x_i x'_i) + (\sum_i x_i x'_i)^2 \\ &= (x \cdot x' + 1)^2 \end{aligned}$$

For $x, x' \in \mathbb{R}^{1000}$, computing this requires only about 2000 floating point operations, less than the 501,501 operations needed for $\phi(x)$.



Recall for linear regression with L2 regularization, we aim to minimize the loss function

$$\mathcal{L}(\theta) = \frac{1}{2} \sum_{i=1}^m \left(\langle \theta, \phi(x^{(i)}) \rangle - y^{(i)} \right)^2 + \frac{\lambda}{2} \|\theta\|^2.$$

Taking the gradient and setting to zero, we have

$$\begin{aligned}\theta &= -\frac{1}{\lambda} \sum_{i=1}^m \left(\langle \theta, \phi(x^{(i)}) \rangle - y^{(i)} \right) \phi(x^{(i)}) \\ &= \sum_{i=1}^m a_i \phi(x^{(i)}),\end{aligned}$$

where $a_i = -\frac{1}{\lambda} (\langle \theta, \phi(x^{(i)}) \rangle - y^{(i)})$.

Dual representation

We can write the expression above as $\Phi^T a$, where Φ is the design matrix whose i^{th} row is given by $\phi(x^{(i)})^T$, and where $a = (a_1, \dots, a_m)^T$. Substituting $\theta = \Phi^T a$ into the loss function, we have

$$\mathcal{L}(a) = \frac{1}{2} \langle \Phi \Phi^T a, \Phi \Phi^T a \rangle - \langle \Phi \Phi^T a, y \rangle + \frac{1}{2} \|y\|^2 + \frac{\lambda}{2} \langle a, \Phi \Phi^T a \rangle,$$

where $y = (y^{(1)}, \dots, y^{(m)})^T$.

Now define $K = \Phi\Phi^T$, where the ij^{th} element of K is given by

$$\langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle = k(x^{(i)}, x^{(j)}),$$

and we have

$$\mathcal{L}(a) = \frac{1}{2} \langle a, K^2 a \rangle - \langle Ka, y \rangle + \frac{1}{2} \|y\|^2 + \frac{\lambda}{2} \langle a, Ka \rangle.$$

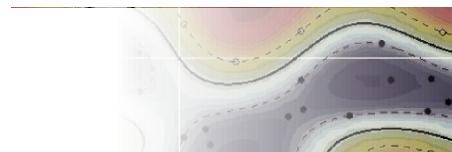
Taking the gradient with respect to a and setting it to zero, we obtain

$$a = (K + \lambda I)^{-1} y,$$

and we see that we can express the solution entirely in terms of the kernel.

KERNEL TRICK

The **kernel trick** refers to the strategy of converting a learning algorithm and the resulting predictor into ones that involve only the computation of the **kernel** $K(x, x') = \phi(x) \cdot \phi(x')$ but not of the **feature map** $\phi(x)$.



KERNEL FUNCTIONS

'Infinite dimensional'
positive definite
matrices

Definition.

A function $K: \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ is a *kernel function* if

1. $K(x, y) = K(y, x)$ for all $x, y \in \mathbb{R}^d$,
2. given $n \in \mathbb{N}$ and $x^{(1)}, x^{(2)}, \dots, x^{(n)} \in \mathbb{R}^d$,

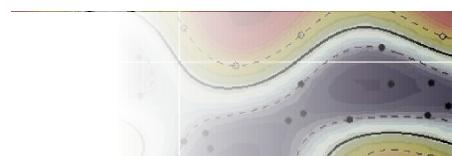
the *Gram matrix* K with entries

$$K_{ij} = K(x^{(i)}, x^{(j)})$$

is positive semidefinite.

Example. $K(x, x') = \phi(x) \cdot \phi(x')$

Can be shown that all kernel functions are of this form!



EXAMPLES

Linear Kernel.

$$K(x, x') = x \cdot x'$$

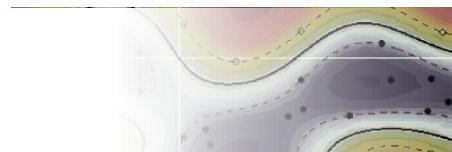
Polynomial Kernel.

$$K(x, x') = (x \cdot x' + 1)^k$$

Radial Basis Kernel.

$$K(x, x') = \exp\left(-\frac{1}{2}\|x - x'\|^2\right)$$

Feature map $\phi(x)$ is infinite dimensional!



Constructing kernels

Assuming $k_1(x, y)$ and $k_2(x, y)$ are valid kernels, the following kernels are also valid:

- (i) $k(x, y) = ak_1(x, y) + bk_2(x, y)$, $a, b > 0$
- (ii) $k(x, y) = k_1(x, y)k_2(x, y)$
- (iii) $k(x, y) = p(k_1(x, y))$, where p is polynomial with positive coefficients
- (iv) $k(x, y) = \exp(k_1(x, y))$
- (v) $k(x, y) = f(x)k_1(x, y)f(y)$, for any function f

Why infinite dimensional?

Theorem (Mercer)

Let \mathcal{X} be a closed and bounded subset of \mathbb{R}^d and μ be a finite measure on \mathcal{X} . Let $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ be a continuous symmetric positive-definite kernel on \mathcal{X} . Define the operator

$$T_K f(x) = \int_{\mathcal{X}} K(x, t) f(t) d\mu(t)$$

for all $f \in L^2(\mathcal{X}, \mu)$. Then there exists an orthonormal basis $\{\phi_i\}$ of eigenfunctions of T_K , with corresponding non-negative eigenvalues λ_i , such that

$$K(x, y) = \sum_{i=1}^{\infty} \lambda_i \phi_i(x) \phi_i(y)$$

where the convergence is absolute and uniform.

- Many feature mappings can be constructed without Mercer's theorem.
- However, Mercer's theorem guarantees a, possibly infinite-dimensional, feature vector $\tilde{\phi} = (\sqrt{\lambda_1}\phi_1(x), \sqrt{\lambda_2}\phi_2(x), \dots)$ and

$$K(x, y) = \langle \tilde{\phi}(x), \tilde{\phi}(y) \rangle.$$

(Note that if K is "simple" enough, it is possible that only a finite number of the eigenvalues are non-zero, making $\tilde{\phi}$ finite-dimensional.)

- Thus we get richer representations as we can work with arbitrarily high, and even infinite-dimensional, feature spaces.

Singular value decomposition (SVD)

Theorem

Given any real $m \times n$ matrix M , there exists a factorization of M of the form

$$M = U\Sigma V^T$$

where

- U is an orthogonal $m \times m$ matrix,
- Σ is a $m \times n$ matrix with non-negative real numbers on the diagonal and zero elsewhere,
- V is an orthogonal $n \times n$ matrix.

The diagonal entries of Σ are called the singular values of M .

SUPPORT VECTOR MACHINES

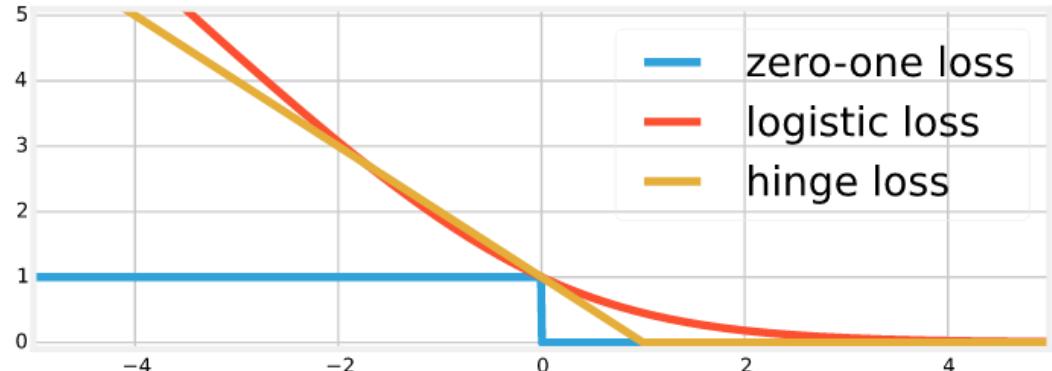




HINGE LOSS



LOSS FUNCTIONS



Training Loss

$$\mathcal{L}_n(\theta) = \frac{1}{n} \sum_{\text{data } (x,y)} \text{Loss}(y(\theta^\top x))$$

Zero-One Loss

$$\text{Loss}_{01}(z) = \llbracket z \leq 0 \rrbracket$$

Hinge Loss

$$\text{Loss}_{\text{H}}(z) = \max\{1 - z, 0\}$$

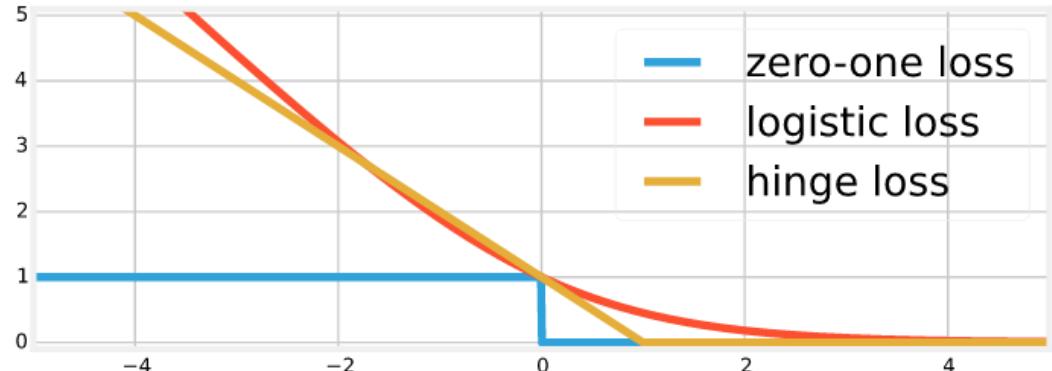
CONVEX!

Penalize large mistakes more.

Penalize near-mistakes, i.e. $0 \leq z \leq 1$.



HINGE LOSS



Find θ that minimizes

$$\begin{aligned}\mathcal{L}_n(\theta) &= \frac{1}{n} \sum_{(x,y) \in \mathcal{S}_n} \text{Loss}_H(z) \\ &= \frac{1}{n} \sum_{(x,y) \in \mathcal{S}_n} \max\{1 - y(\theta^\top x), 0\}\end{aligned}$$

Gradient

$$\nabla_z \text{Loss}_H(z) = \begin{cases} 0 & \text{if } z > 1, \\ -1 & \text{otherwise.} \end{cases}$$

$$\nabla_\theta \text{Loss}_H(y(\theta^\top x)) = \begin{cases} 0 & \text{if } y(\theta^\top x) > 1, \\ -yx & \text{otherwise.} \end{cases}$$





LAGRANGE MULTIPLIERS



CONSTRAINED OPTIMIZATION

Want to minimize some function $f(x)$, but there are some *constraints* on the values of x .

Method 1 (Dual Problem)

Solve a *dual optimization problem* where the constraints are nicer, and where it is easier to implement gradient descent.

Method 2 (Exact Solution)

Solve the *Lagrangian* system of equations.



EQUALITY CONSTRAINTS

Problem.

$$\text{minimize } f(x)$$

$$\text{subject to } h_1(x) = 0, \dots, h_l(x) = 0$$

Lagrangian.

$$L(x, \lambda) = f(x) + \lambda_1 h_1(x) + \dots + \lambda_l h_l(x)$$

Example.

$$\text{minimize } f(x) = n_1 \log x_1 + \dots + n_d \log x_d$$

$$\text{subject to } h(x) = x_1 + \dots + x_d - 1 = 0$$

$$L(x, \lambda) = n_1 \log x_1 + \dots + n_d \log x_d + \lambda(x_1 + \dots + x_d - 1)$$



TWO-PLAYER GAME

$$L(x, \lambda) = f(x) + \lambda_1 h_1(x) + \cdots + \lambda_l h_l(x)$$

Rules.

- You get to choose the value of x .
Your goal is to minimize $L(x, \lambda)$.
- Your adversary gets to choose the value of λ .
His goal is to maximize $L(x, \lambda)$.



PRIMAL GAME

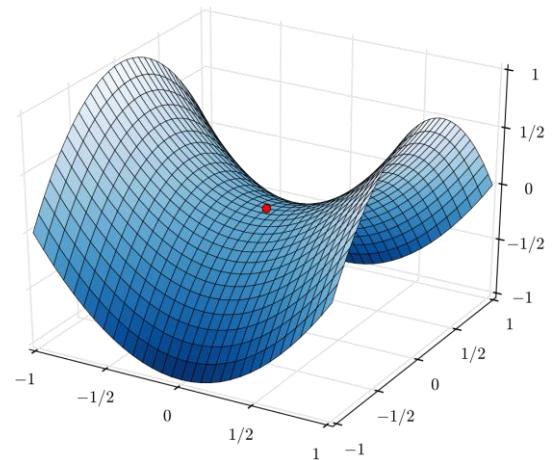
$$L(x, \lambda) = f(x) + \lambda_1 h_1(x) + \cdots + \lambda_l h_l(x)$$

Primal Game. You go first.

Your Strategy.

- Ensure that $h_1(x) = 0, \dots, h_l(x) = 0$.
- Find x that minimizes $f(x)$.

Final Score. $p^* = \min_x \max_{\lambda} L(x, \lambda)$



The optimal x^*, λ^* are
saddle points of $L(x, \lambda)$.



DUAL GAME

$$L(x, \lambda) = f(x) + \lambda_1 h_1(x) + \cdots + \lambda_l h_l(x)$$

Dual Game. You go second.

Adversary's Strategy.

- For each λ , compute $\ell(\lambda) = \min_x L(x, \lambda)$
- Find λ that maximizes $\ell(\lambda)$.

Final Score. $d^* = \max_{\lambda} \min_x L(x, \lambda)$



MAX-MIN INEQUALITY

Primal.

$$p^* = \min_x \max_{\lambda} L(x, \lambda)$$

Dual.

$$d^* = \max_{\lambda} \min_x L(x, \lambda)$$

“you do better if you have the last say”

$$\begin{aligned} p^* &= \min_x \max_{\lambda} L(x, \lambda) \\ &\geq \max_{\lambda} \min_x L(x, \lambda) = d^* \end{aligned}$$

If $p^* = d^*$, we can solve the primal by solving the dual.

Challenge. Can you prove $p^* \geq d^*$? (*not in syllabus)



MAX-MIN INEQUALITY

Example.

	$x = 1$	$x = 2$
$\lambda = 1$	1	4
$\lambda = 2$	3	2

Primal. $p^* = \min_x \max_{\lambda} L(x, \lambda) = 3$

Dual. $d^* = \max_{\lambda} \min_x L(x, \lambda) = 2$



EXACT SOLUTION

Problem.

$$\text{minimize } f(x)$$

$$\text{subject to } h_1(x) = 0, \dots, h_l(x) = 0$$

Lagrange multipliers.

1. Write down the Lagrangian.

$$L(x, \lambda) = f(x) + \lambda_1 h_1(x) + \dots + \lambda_l h_l(x)$$

2. Solve for critical points x, λ .

$$\nabla_x L(x, \lambda) = 0, \quad h_1(x) = 0, \dots, h_l(x) = 0$$

3. Pick critical point which gives global minimum.



EXAMPLE

minimize $f(x) = n_1 \log x_1 + \cdots + n_d \log x_d$
subject to $h(x) = x_1 + \cdots + x_d - 1 = 0$

Lagrangian

$$L(x, \lambda) = n_1 \log x_1 + \cdots + n_d \log x_d + \lambda(x_1 + \cdots + x_d - 1)$$

Critical points

$$\begin{aligned} 0 &= x_1 + \cdots + x_d - 1 && \Rightarrow && (-\lambda) = n_1 + \cdots + n_d \\ 0 &= n_i/x_i + \lambda && && x_i = n_i/(-\lambda) \end{aligned}$$



INEQUALITY CONSTRAINTS (PRIMAL-DUAL)

Primal Problem.

minimize $f(x)$

subject to $g_1(x) \leq 0, \dots, g_m(x) \leq 0$

Lagrangian.

$$L(x, \alpha) = f(x) + \alpha_1 g_1(x) + \dots + \alpha_m g_m(x)$$

Dual Problem.

maximize $\ell(\alpha)$

where $\ell(\alpha) = \min_{x \in \mathbb{R}^d} L(x, \alpha)$

subject to $\alpha_1 \geq 0, \dots, \alpha_m \geq 0$

Box constraints are
easier to work with!



INEQUALITY CONSTRAINTS (EXACT SOLN)

minimize $f(x)$
subject to $g_1(x) \leq 0, \dots, g_m(x) \leq 0$

Lagrangian.

$$L(x, \alpha) = f(x) + \alpha_1 g_1(x) + \dots + \alpha_m g_m(x)$$

Solve for x, α satisfying

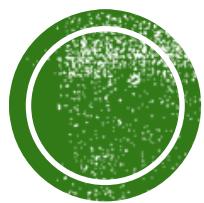
1. $\nabla_x L(x, \alpha) = 0$
2. $g_1(x) \leq 0, \dots, g_m(x) \leq 0$
3. $\alpha_1 \geq 0, \dots, \alpha_m \geq 0$
4. $\alpha_1 g_1(x) = 0, \dots, \alpha_m g_m(x) = 0$

Karush-Kuhn-Tucker (KKT) Conditions

Complementary Slackness



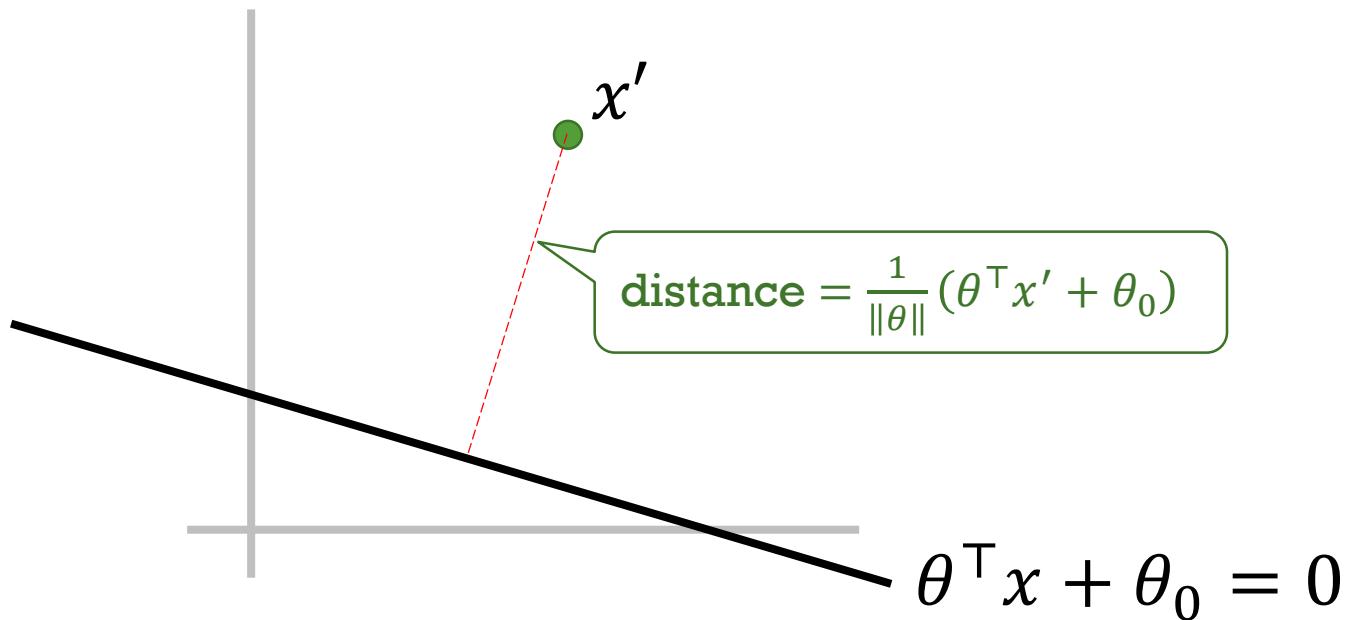
5 min break



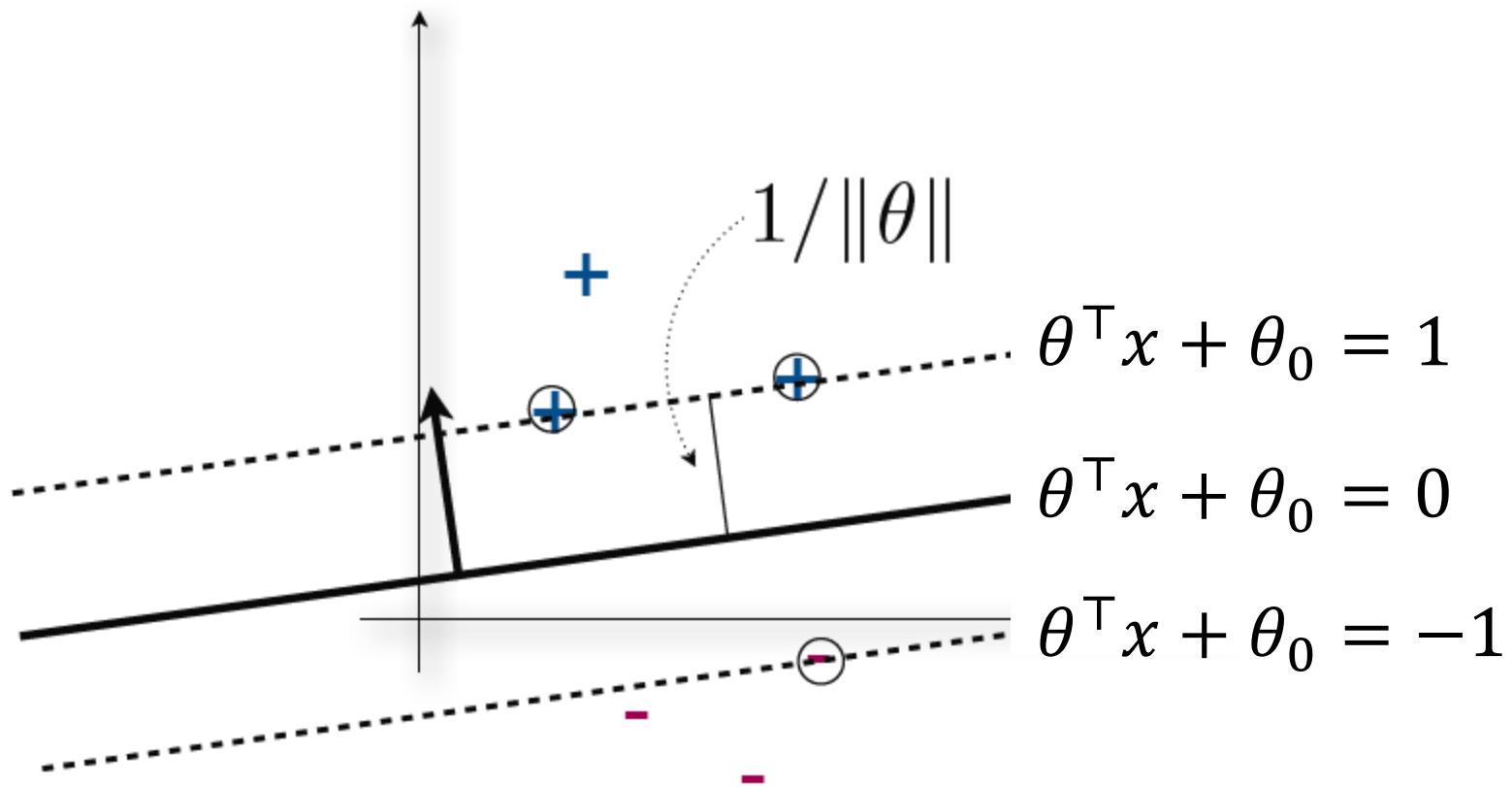
MAXIMUM MARGINS



COMPUTING THE MARGIN



COMPUTING THE MARGIN



MAXIMUM MARGIN

Unfortunately, this only applies to data that is linearly separable.

Our goal is to

$$\text{maximize } 1/\|\theta\|$$

$$\text{subject to } y(\theta^\top x + \theta_0) \geq 1 \text{ for all data } (x, y)$$

Or equivalently,

$$\text{minimize } \frac{1}{2} \|\theta\|^2$$

$$\text{subject to } y(\theta^\top x + \theta_0) \geq 1 \text{ for all data } (x, y)$$



LAGRANGIAN

$$\text{minimize} \quad \frac{1}{2} \|\theta\|^2$$

subject to $y(\theta^\top x) \geq 1$ for all data (x, y)

Drop θ_0 for now

Lagrangian. $L(\theta, \alpha) = \frac{1}{2} \|\theta\|^2 + \sum_{(x,y)} \alpha_{x,y} (1 - y(\theta^\top x))$

To find $\ell(\alpha) = \min_{\theta} L(\theta, \alpha)$, we solve

$$0 = \nabla_{\theta} L(\theta, \alpha) = \theta - \sum_{(x,y)} \alpha_{x,y} yx$$

to get $\theta = \sum_{(x,y)} \alpha_{x,y} yx$. Substituting into $L(\theta, \alpha)$ gives

$$\ell(\alpha) = \sum_{(x,y)} \alpha_{x,y} - \frac{1}{2} \sum_{(x,y)} \sum_{(x',y')} \alpha_{x,y} \alpha_{x',y'} yy' (x^\top x').$$



PRIMAL-DUAL

It can be shown that the primal and dual problems are equivalent (*strong duality*).

Primal.

$$\text{minimize} \quad \frac{1}{2} \|\theta\|^2$$

subject to $y(\theta^\top x) \geq 1$ for all data (x, y)

Dual.

$$\text{maximize} \quad \sum_{(x,y)} \alpha_{x,y} - \frac{1}{2} \sum_{(x,y)} \sum_{(x',y')} \alpha_{x,y} \alpha_{x',y'} y y' (x^\top x')$$

subject to $\alpha_{x,y} \geq 0$ for all (x, y)

After solving the dual to get the optimal $\alpha_{x,y}$'s, we obtain the optimal θ using $\theta = \sum_{(x,y)} \alpha_{x,y} y x$.



SUPPORT VECTORS

Complementary Slackness.

$$\hat{\alpha}_{x,y} > 0:$$

$$y(\hat{\theta}^\top x) = 1$$

Support Vectors

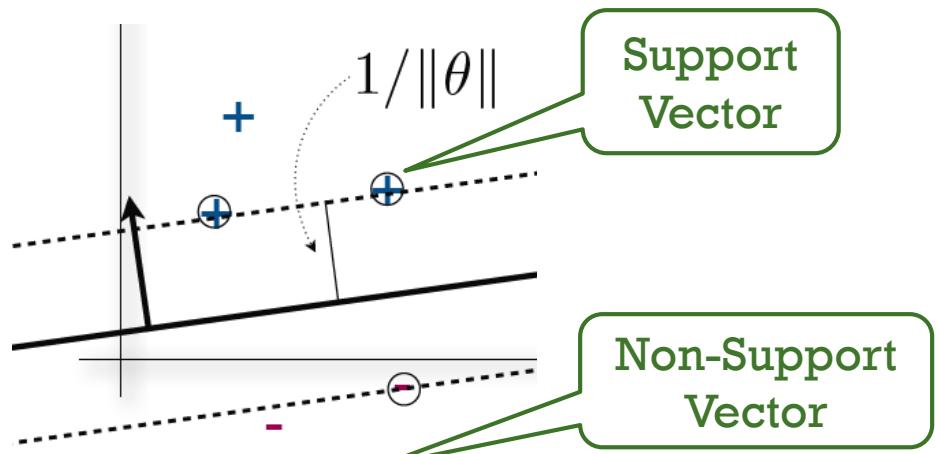
$$\hat{\alpha}_{x,y} = 0:$$

$$y(\hat{\theta}^\top x) > 1$$

Non-Support Vectors

Sparsity

Since very few data points are support vectors, most of the $\hat{\alpha}_{x,y}$ will be zero.



KERNEL TRICK

Learning.

$$\ell(\alpha) = \sum_{(x,y)} \alpha_{x,y} - \frac{1}{2} \sum_{(x,y)} \sum_{(x',y')} \alpha_{x,y} \alpha_{x',y'} y y' (\mathbf{x}^\top \mathbf{x}')$$

Prediction.

$$h(x; \theta) = \text{sign}(\theta^\top x) = \text{sign} \left(\sum_{(x',y')} \alpha_{x',y'} y' (\mathbf{x}^\top \mathbf{x}') \right)$$

For the dual, we don't need the feature vectors \mathbf{x}, \mathbf{x}' .
Knowing just the dot products $(\mathbf{x}^\top \mathbf{x}')$ is enough.

Recall that $(\mathbf{x}^\top \mathbf{x}')$ is a measure of similarity between x and x' .
This similarity function is also called a *kernel*.





EXTENSIONS



SVM WITH OFFSET

Primal.

$$\text{minimize} \quad \frac{1}{2} \|\theta\|^2$$

subject to $y(\theta^\top x + \theta_0) \geq 1$ for all data (x, y)

Dual.

$$\text{maximize} \quad \sum_{(x,y)} \alpha_{x,y} - \frac{1}{2} \sum_{(x,y)} \sum_{(x',y')} \alpha_{x,y} \alpha_{x',y'} y y' (x^\top x')$$

subject to $\alpha_{x,y} \geq 0$ for all (x, y)

$$\sum_{(x,y)} \alpha_{x,y} y = 0$$

Parameters. $\hat{\theta} = \sum_{(x,y)} \alpha_{x,y} y x$

$\hat{\theta}_0 = y - \hat{\theta}^\top x$ where (x, y) is a support vector



SVM WITH ERRORS

Primal.

$$\text{minimize} \quad \frac{\lambda}{2} \|\theta\|^2 + \frac{1}{n} \sum_{(x,y)} \xi_{x,y}$$

$$\begin{aligned} \text{subject to} \quad & y(\theta^\top x + \theta_0) \geq 1 - \xi_{x,y} \\ & \xi_{x,y} \geq 0 \end{aligned}$$

for all data (x, y)

for all data (x, y)

Slack variables allow constraints to be violated for a cost.

Equivalent Primal.

$$\text{minimize} \quad \frac{\lambda}{2} \|\theta\|^2 + \frac{1}{n} \sum_{(x,y)} \text{Loss}_H(y(\theta^\top x + \theta_0))$$

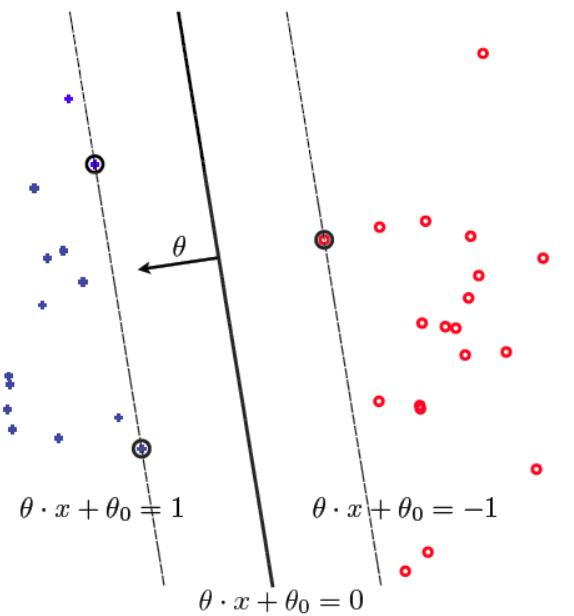
Hinge-loss classifier with regularization!



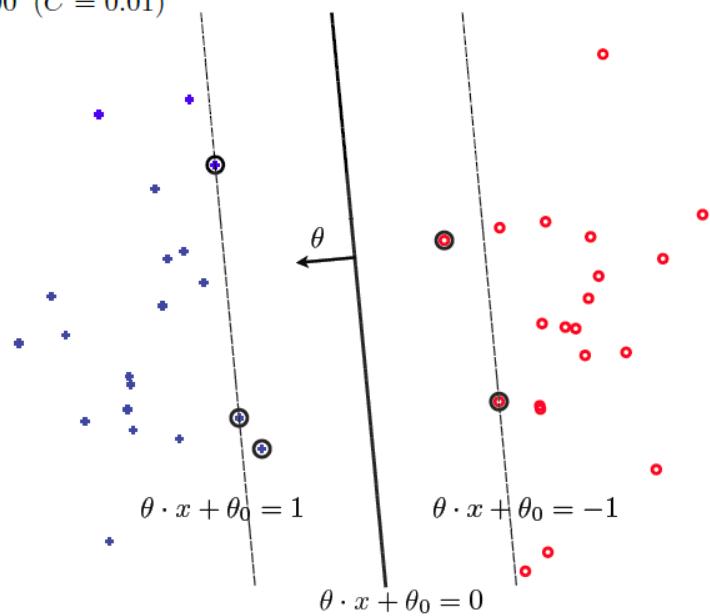
SVM WITH ERRORS

Linearly Separable.

$$\lambda = 1 \quad (C = 1)$$



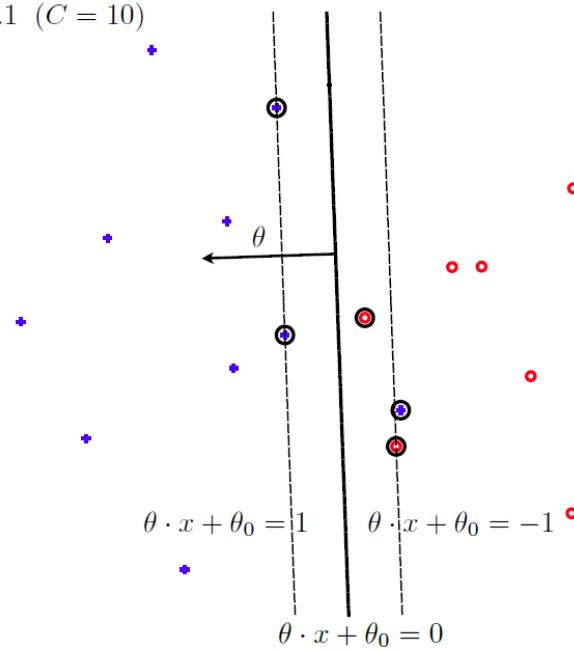
$$\lambda = 100 \quad (C = 0.01)$$



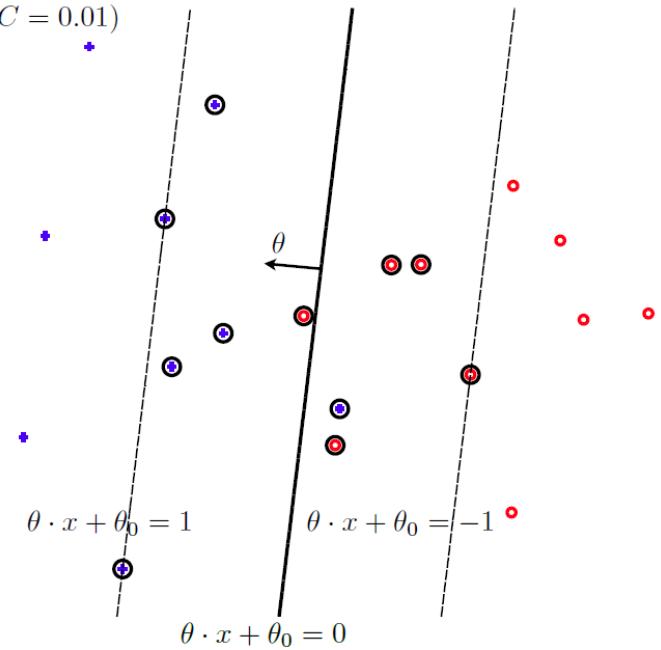
SVM WITH ERRORS

Not Linearly Separable.

$\lambda = 0.1 \ (C = 10)$



$\lambda = 100 \ (C = 0.01)$



SVM WITH ERRORS

Dual.

$$\text{maximize} \quad \sum_{(x,y)} \alpha_{x,y} - \frac{1}{2} \sum_{(x,y)} \sum_{(x',y')} \alpha_{x,y} \alpha_{x',y'} y y' (x^\top x')$$

$$\text{subject to} \quad 1/\lambda \geq \alpha_{x,y} \geq 0 \text{ for all } (x, y)$$

$$\sum_{(x,y)} \alpha_{x,y} y = 0$$

Putting limits on what the adversary can do.

There are many efficient solvers for quadratic problems with box constraints.



Probability Density Functions

A random variable X on a discrete space is well-defined if

$$\sum_{x \in \mathcal{X}} P(X = x) = 1. \quad (1)$$

If the state space \mathcal{X} is not discrete, for e.g. $\mathcal{X} = \mathbb{R}$ or \mathbb{R}^n , then a continuous random variable X is well-defined if there exists a probability density function (pdf) $f_X(x) \geq 0$ such that

$$\int_{\mathcal{X}} f_X(x) dx = 1. \quad (2)$$

Its cumulative distribution function (cdf)

$$P(X \leq a) = \int_{-\infty}^a f_X(x) dx \quad (3)$$

is a function of a , and is also denoted by $F(a)$.

Joint Distributions

A multivariate random variable $\mathbf{X} = (X_1, \dots, X_n)$ with state space $\mathcal{X}_1, \dots, \mathcal{X}_n$ is a joint distribution if

$$\sum_{x_1 \in \mathcal{X}_1, \dots, x_n \in \mathcal{X}_n} P(X_1 = x_1, \dots, X_n = x_n) = 1, \quad (4)$$

for discrete random variables. For continuous random variables, there exists a density function $f_{X_1, \dots, X_n}(x_1, \dots, x_n) \geq 0$ such that

$$\int_{x_1 \in \mathcal{X}_1, \dots, x_n \in \mathcal{X}_n} f_{X_1, \dots, X_n}(x_1, \dots, x_n) d\mathbf{x} = 1. \quad (5)$$

Marginal Distributions

With the joint distribution probabilities, one can derive the distribution of each individual X_i , or a subset of them. These distributions are known as marginal distributions.

For discrete random variables, the multivariate random variable (X_1, \dots, X_{n-1}) has the probability distribution

$$P(X_1 = x_1, \dots, X_{n-1} = x_{n-1}) = \sum_{x_n \in \mathcal{X}_n} P(X_1 = x_1, \dots, X_n = x_n), \quad (6)$$

while the random variable X_1 has the density function

$$P(X_1 = x_1) = \sum_{x_2 \in \mathcal{X}_2, \dots, x_n \in \mathcal{X}_n} P(X_1 = x_1, \dots, X_n = x_n). \quad (7)$$

For continuous random variables, the random variable X_1 has the density function

$$f_{X_1}(x_1) = \int_{x_2 \in \mathcal{X}_2, \dots, x_n \in \mathcal{X}_n} f_{X_1, \dots, X_n}(x_1, \dots, x_n) dx_2 \dots dx_n. \quad (8)$$

Conditional Distributions

Given a joint discrete distribution (X, Y) , the conditional probability function of X given Y is given by

$$P(X = x \mid Y = y) = \frac{P(X = x, Y = y)}{P(Y = y)}. \quad (9)$$

When (X, Y) is continuous, the probability density function, $f_{X|Y}(x \mid y)$, of X given Y has the expression

$$f_{X|Y}(x \mid y) = \frac{f_{X,Y}(x, y)}{f_Y(y)}. \quad (10)$$

Thus, the conditional distributions can be computed from the joint distributions and the marginal distributions.

Gaussian processes for regression and optimization

Conditional Gaussian distributions

Let $x \in \mathbb{R}^{n+p}$ be a Gaussian random vector which we will partition as

$$x = \begin{bmatrix} x_a \\ x_b \end{bmatrix},$$

where $x_a \in \mathbb{R}^n$ and $x_b \in \mathbb{R}^p$. Then the means and covariances can be represented as

$$\mu = \begin{bmatrix} \mu_a \\ \mu_b \end{bmatrix}, \quad \Sigma = \begin{bmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{bmatrix}. \quad (11)$$

Here, Σ_{aa} is $n \times n$, Σ_{ab} is $n \times p$, Σ_{ba} is $p \times n$ and Σ_{bb} is $p \times p$.

We will denote the precision matrix Σ^{-1} as

$$\Lambda = \Sigma^{-1} = \begin{bmatrix} \Lambda_{aa} & \Lambda_{ab} \\ \Lambda_{ba} & \Lambda_{bb} \end{bmatrix}.$$

Then we can expand the exponent of the Gaussian distribution as

$$\begin{aligned} & -\frac{1}{2} \langle (x - \mu), \Sigma^{-1}(x - \mu) \rangle \\ &= -\frac{1}{2} \langle (x_a - \mu_a), \Lambda_{aa}(x_a - \mu_a) \rangle - \frac{1}{2} \langle (x_a - \mu_a), \Lambda_{ab}(x_b - \mu_b) \rangle \\ & \quad - \frac{1}{2} \langle (x_b - \mu_b), \Lambda_{ba}(x_a - \mu_a) \rangle - \frac{1}{2} \langle (x_b - \mu_b), \Lambda_{bb}(x_b - \mu_b) \rangle. \end{aligned} \tag{12}$$

- To find the conditional distribution, we set x_b to be a constant in (12) and renormalize. The resulting distribution is still Gaussian, so all that remains is to find the conditional mean and conditional covariance.
- Using the fact that Λ_{aa} is symmetric and $\Lambda_{ba}^T = \Lambda_{ab}$, (12) can be rewritten as

$$-\frac{1}{2} \langle x_a, \Lambda_{aa} x_a \rangle + \langle x_a, \Lambda_{aa} \mu_a - \Lambda_{ab} (x_b - \mu_b) \rangle + \text{const}, \quad (13)$$

where the constant term does not contain x_a .

We know that the exponent of the conditional distribution must take the form

$$\begin{aligned}
 & -\frac{1}{2} \left\langle (x_a - \mu_{a|b}), \Sigma_{a|b}^{-1} (x_a - \mu_{a|b}) \right\rangle \\
 &= -\frac{1}{2} \left\langle x_a, \Sigma_{a|b}^{-1} x_a \right\rangle + \left\langle x_a, \Sigma_{a|b}^{-1} \mu_{a|b} \right\rangle + \text{const},
 \end{aligned} \tag{14}$$

so comparing (13) and (14), we must have

$$\begin{aligned}
 \Sigma_{a|b} &= \Lambda_{aa}^{-1}, \\
 \mu_{a|b} &= \mu_a - \Lambda_{aa}^{-1} \Lambda_{ab} (x_b - \mu_b).
 \end{aligned}$$

HW problem

Given a partitioned matrix, the following formula holds

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} M & -MBD^{-1} \\ -D^{-1}CM & D^{-1} + D^{-1}CMBD^{-1} \end{bmatrix},$$

where $M = (A - BD^{-1}C)^{-1}$.

Finally, using this formula and definition of the precision matrix, we can express the conditional mean and variance as

$$\begin{aligned}\mu_{a|b} &= \mu_a + \Sigma_{ab}\Sigma_{bb}^{-1}(x_b - \mu_b), \\ \Sigma_{a|b} &= \Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}.\end{aligned}$$

Marginal Gaussian distributions

Given $p(x_a, x_b)$ which is normally distributed with mean and covariance as in (11), the marginal distribution

$$p(x_a) = \int p(x_a, x_b) dx_b$$

is also Gaussian with

$$\begin{aligned}\mathbb{E}[x_a] &= \mu_a, \\ \text{Cov}[x_a] &= \Sigma_{aa}.\end{aligned}$$

Gaussian processes

Definition

A stochastic process $\{X_t; t \in T\}$ is a Gaussian process if for any finite set of indices $\{t_1, \dots, t_n\}$ of T , $(X_{t_1}, \dots, X_{t_n})$ is a multivariate normal random variable.

- If T is an infinite set, eg. a subset of \mathbb{R}^d , then this is an infinite-dimensional generalization of multivariate normal random variables.
- A Gaussian process is completely determined by its
 - Mean function $\mu(\cdot) : T \rightarrow \mathbb{R}$
 - Covariance function or kernel $k(\cdot, \cdot) : T \times T \rightarrow \mathbb{R}$
- The samples of a Gaussian process are paths if $T = \mathbb{R}$, or surfaces if $T = \mathbb{R}^d$, $d > 1$, and their smoothness is dependent on the kernel.

Examples of kernels

- Radial basis function (RBF or "Gaussian"):

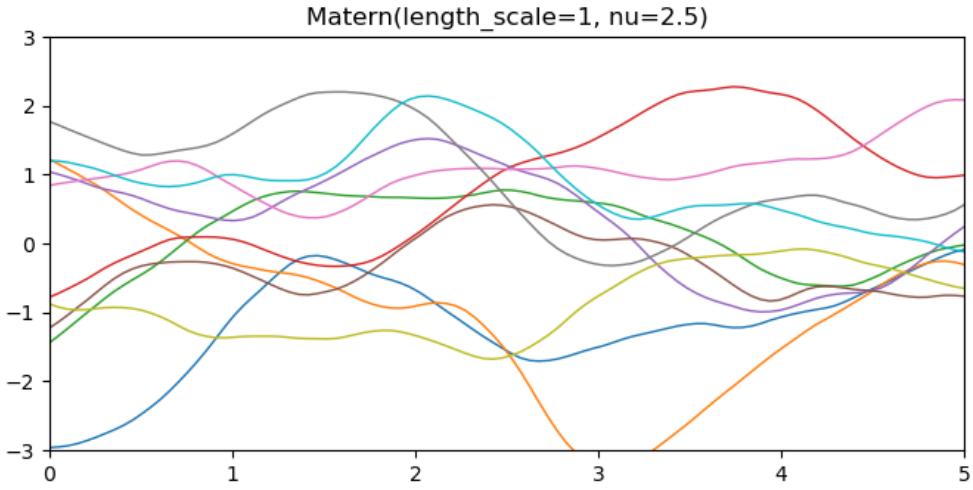
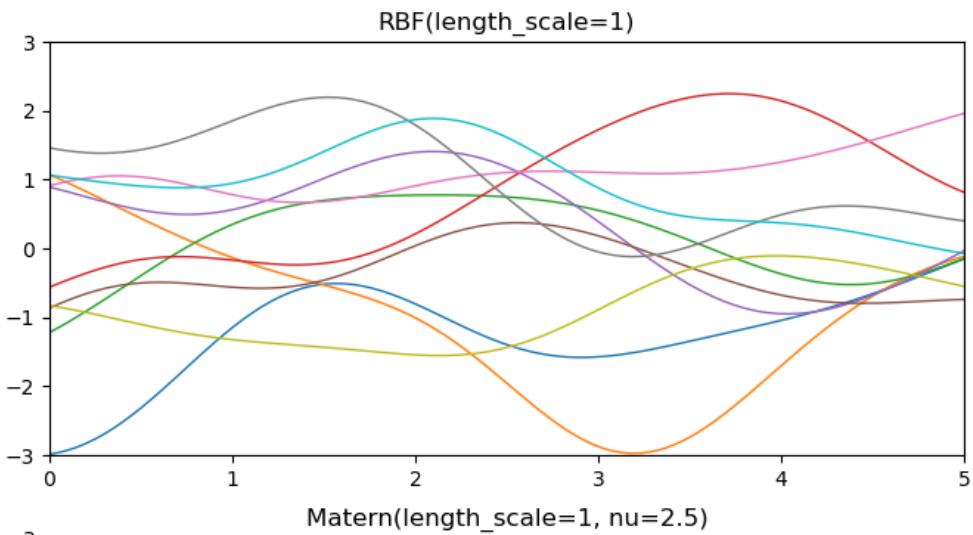
$$k(x, y) = e^{-\frac{\|x-y\|^2}{2\sigma^2}}, \quad x, y \in \mathbb{R}^d$$

- Matérn $\frac{5}{2}$ ($\frac{5}{2} = \nu + \frac{1}{2}$, $\nu = 2$):

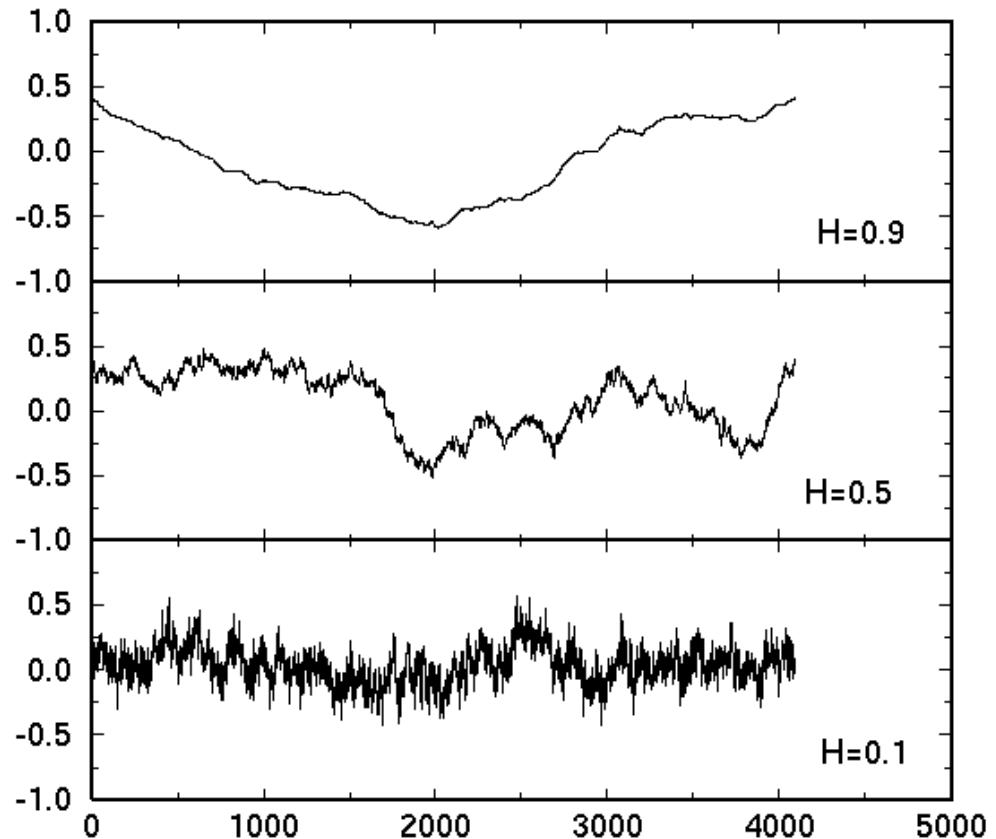
$$k(x, y) = \left(1 + \sqrt{5} \|x - y\| + \frac{5}{3} \|x - y\|^2 \right) e^{-\sqrt{5}\|x-y\|}, \quad x, y \in \mathbb{R}^d$$

- Fractional Brownian motion, with Hurst parameter $H \in (0, 1)$:

$$k(x, y) = \frac{1}{2} \left(x^{2H} + y^{2H} - |x - y|^{2H} \right), \quad x, y \in \mathbb{R}^+$$



Fractional Brownian motion



Gaussian processes for regression

- Recall that we model

$$y = f(x) + \epsilon, \quad (15)$$

where we assume $\epsilon \sim \mathcal{N}(0, \tau^2)$ and is independent between samples.

- Previously, we have been assuming that the hypothesis function f is parametric, i.e. it can be described by a fixed number of parameters, e.g. $f(x) = \langle \theta, x \rangle$, which are to be learnt.

Parametric vs non-parametric methods

- In contrast, non-parametric methods have a flexible number of parameters that grows with the data.
- Furthermore, for non-parameteric algorithms, e.g. K-nearest neighbours (kNN), data points, or a subset of them, are kept and used in the prediction phase, resulting in a "memory-based" approach.
- In parametric algorithms, once data is used to learn the parameters, it can be discarded as only the learnt parameters are used for prediction.

- Returning to (15), we have

$$p(y|f(x)) \sim \mathcal{N}(f(x), \tau^2).$$

- Now instead of modeling f from a parameterized family of functions, we enforce a prior Gaussian distribution, with kernel K :

$$p(f(\cdot)) \sim \mathcal{N}(0, K).$$

Conditioned on input values x_1, \dots, x_n , we compute the marginal distribution of y by

$$p(y) = \int p(y|f)p(f)df.$$

We know this is normally distributed with mean

$$\mathbb{E}[y] = \mathbb{E}[f] + \mathbb{E}[\epsilon] = 0$$

and covariance C^n with entries

$$\begin{aligned} C_{ij}^n &= \mathbb{E}[y^{(i)}y^{(j)}] = \mathbb{E}\left[\left(f\left(x^{(i)}\right) + \epsilon^{(i)}\right)\left(f\left(x^{(j)}\right) + \epsilon^{(j)}\right)\right] \\ &= \mathbb{E}\left[f\left(x^{(i)}\right)f\left(x^{(j)}\right)\right] + \mathbb{E}\left[\epsilon^{(i)}\epsilon^{(j)}\right] \quad (16) \\ &= K(x_i, x_j) + \tau^2\delta_{ij}. \end{aligned}$$

Prediction

- Given $x^{(1)}, \dots, x^{(n)}$, and a new observation $x^{(n+1)}$, we predict $y^{(n+1)}$ by computing the posterior distribution

$$p\left(y^{(n+1)} \mid y^{(1)}, \dots, y^{(n)}\right).$$

- The joint distribution over $y^{(1)}, \dots, y^{(n)}, y^{(n+1)}$ has mean 0 and covariance C^{n+1} given by (16), which can be partitioned as

$$C^{n+1} = \begin{bmatrix} C^n & k \\ k^T & c \end{bmatrix},$$

where $k = [K(x^{(1)}, x^{(n+1)}), \dots, K(x^{(n)}, x^{(n+1)})]^T$ and $c = K(x^{(n+1)}, x^{(n+1)}) + \tau^2$.

Hence

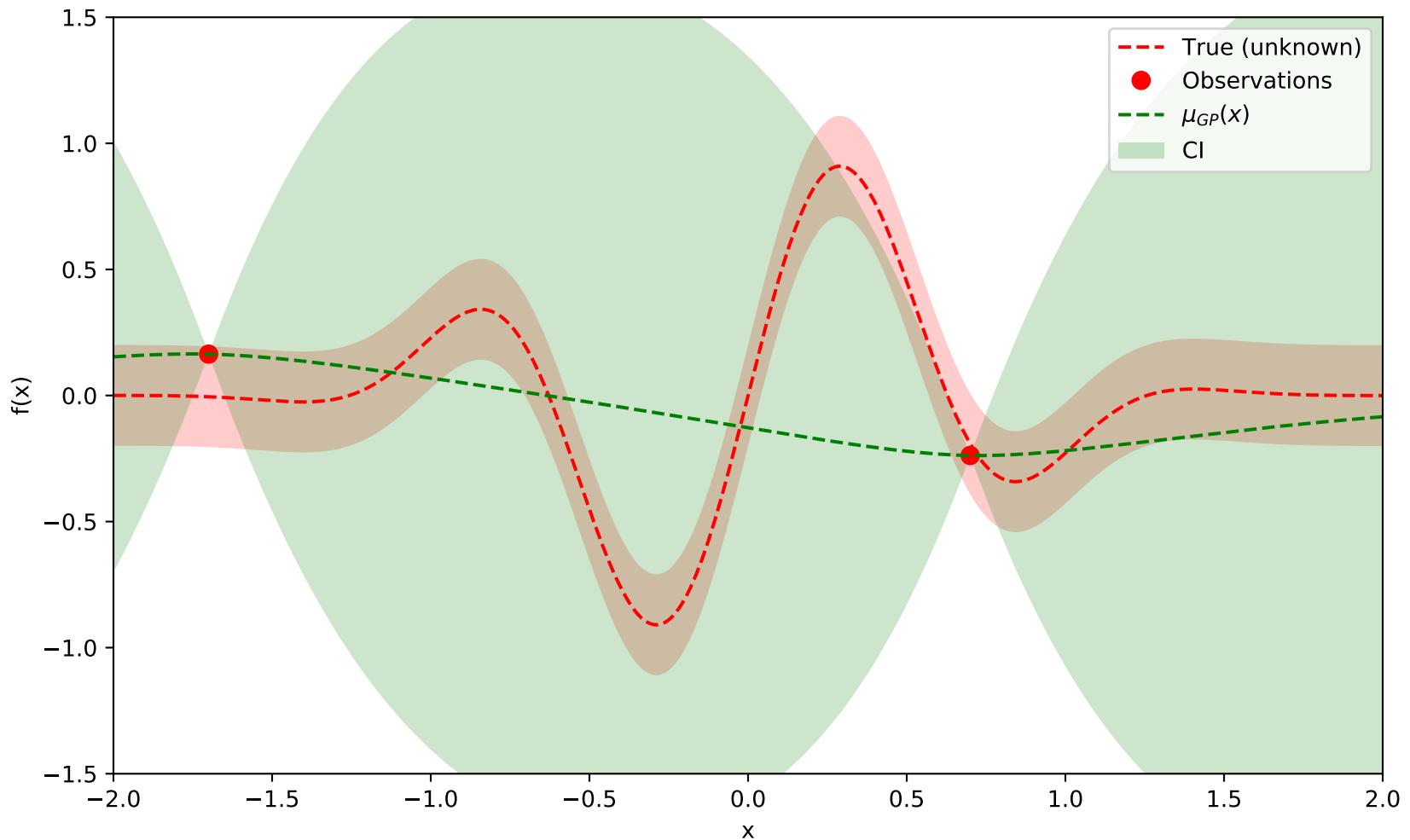
$$p\left(y^{(n+1)} \mid y^{(1)}, \dots, y^{(n)}\right) \sim \mathcal{N}(\mu, \sigma^2),$$

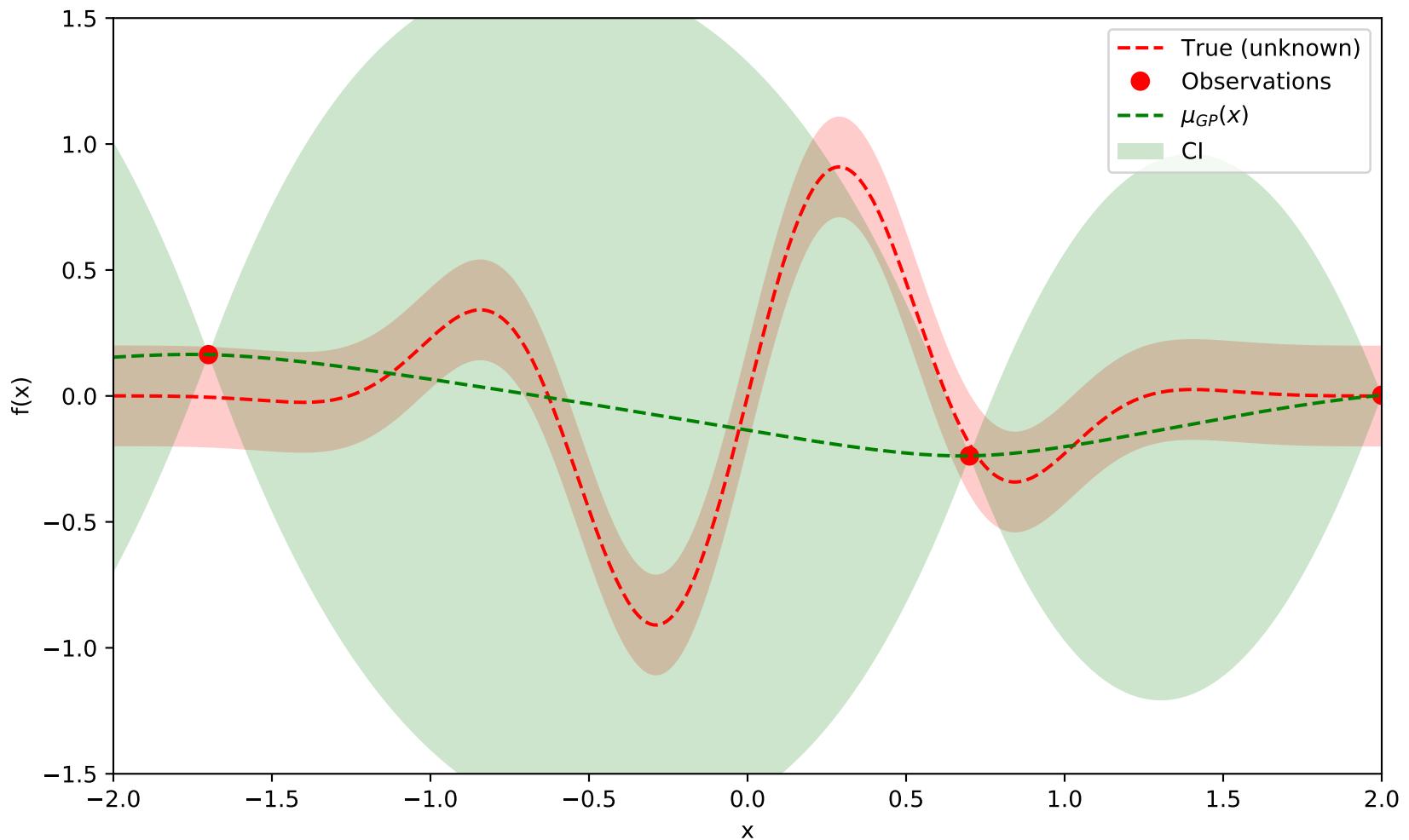
where

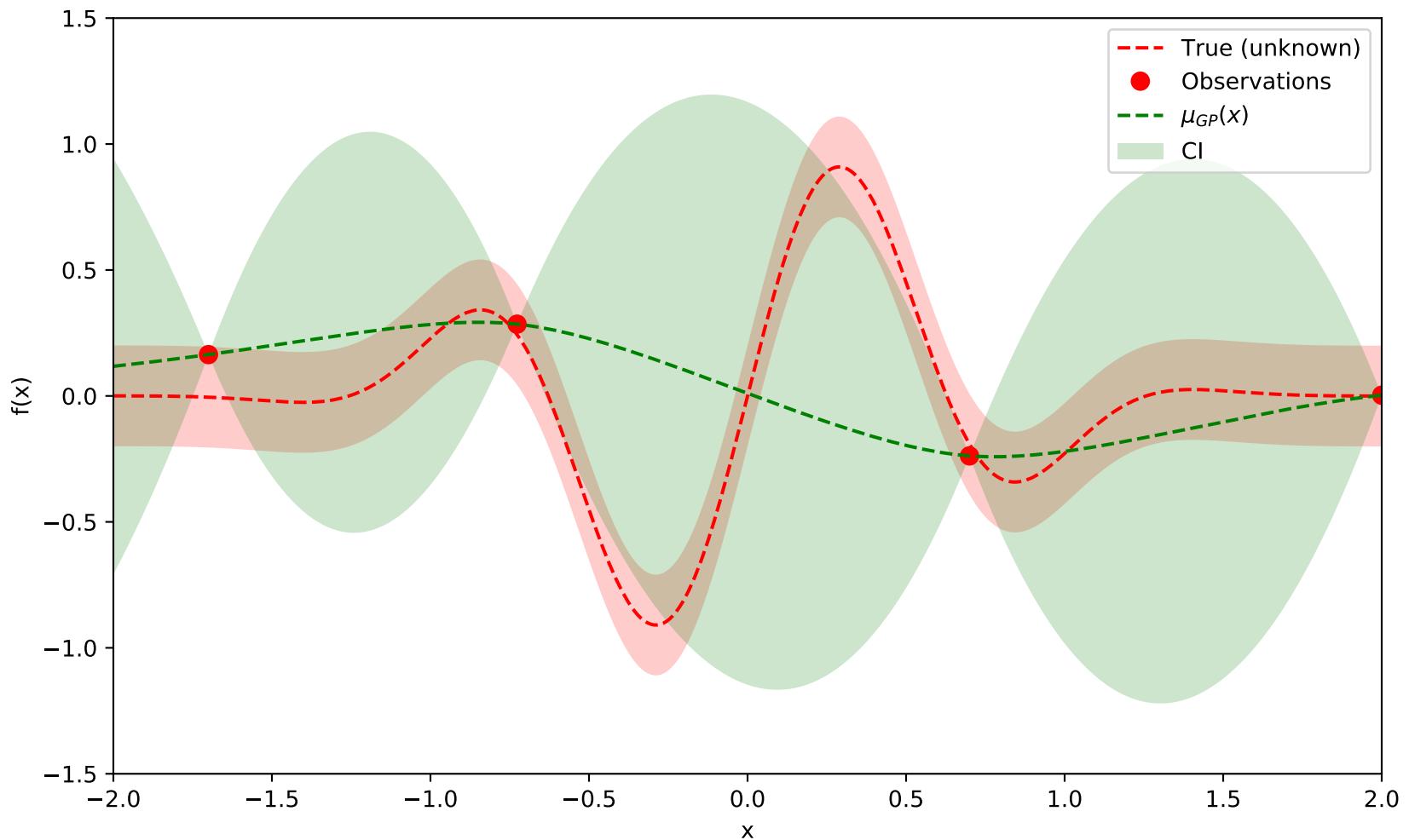
$$\begin{aligned}\mu &= k^T (C^n)^{-1} \mathbf{y}_n, \\ \sigma^2 &= c - k^T (C^n)^{-1} k,\end{aligned}$$

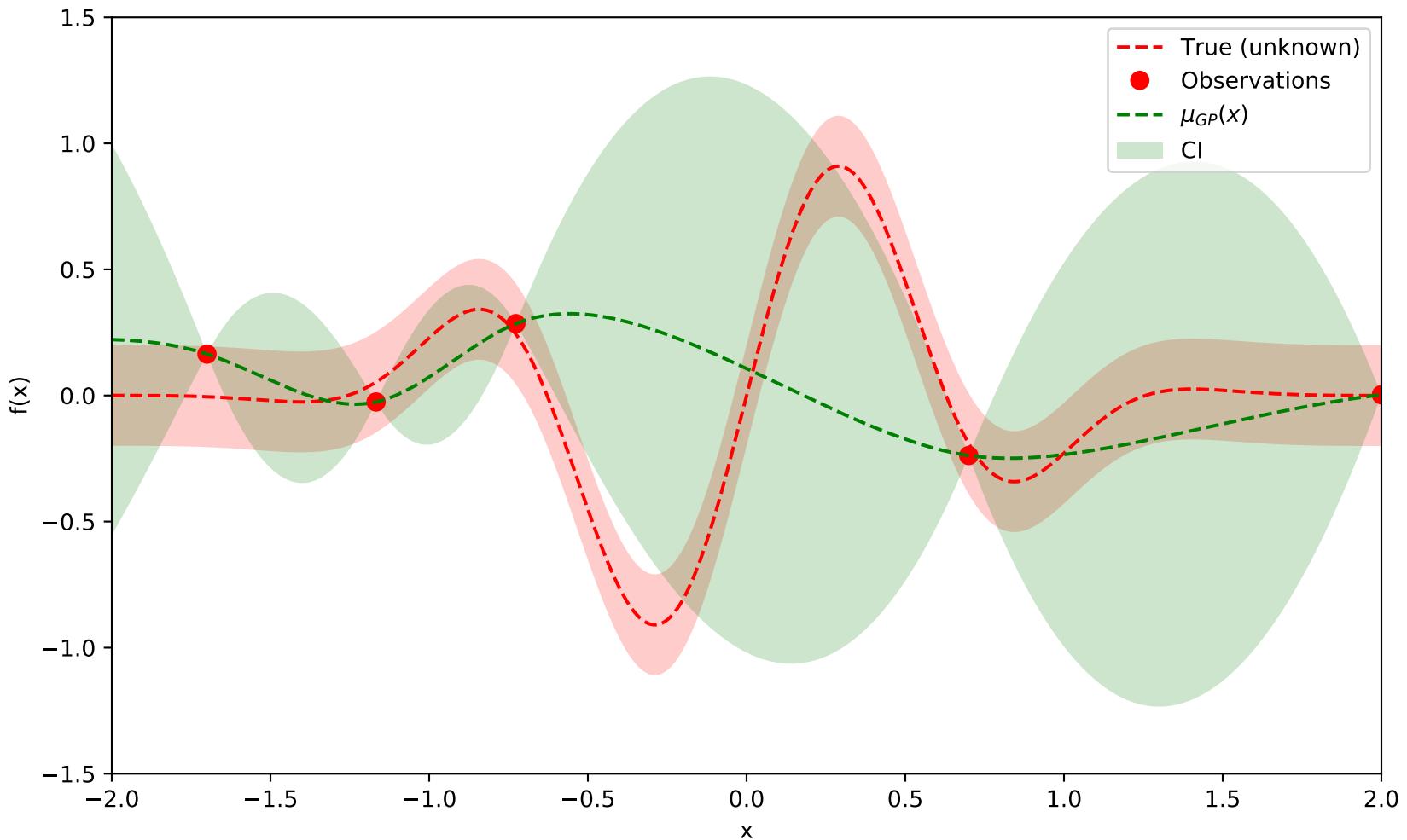
and we denote $\mathbf{y}_n = [y^{(1)}, \dots, y^{(n)}]^T$.

Note that μ and σ can be viewed as functions of $x^{(n+1)}$.









Prediction (summary)

Given $x^{(1)}, \dots, x^{(n+1)}$,

prior distribution $\xrightarrow{\{t^{(i)}\}_{i=1}^n}$ posterior distribution,

i.e.

$$p(y^{(1)}, \dots, y^{(n+1)}) \sim \mathcal{N}(0, C^{n+1}) \longrightarrow p(y^{(n+1)} \mid y^{(1)} = t^{(1)}, \dots, y^{(n)} = t^{(n)}) \sim \mathcal{N}(\mu, \sigma^2).$$

Hence, the prior distribution is entirely determined by the kernel K , whereas the the posterior distribution is continually being updated by the observations $t^{(i)}$.

Bayesian optimization with Gaussian processes

- Why?
 - In many machine learning problems, the objective function f is a black-box function which does not have an analytic expression (i.e. one cannot take derivatives), or has one that is too costly to compute.
 - Moreover, f may be expensive to evaluate, or its domain may be high-dimensional, which makes a grid-search over its domain prohibitively time-consuming (curse of dimensionality).
 - Eg. Hyper-parameter tuning in deep learning models

- Bayesian optimization techniques attempt to find the global optimum of f in as few steps as possible.
- How?
 - Define a *surrogate model* which approximates the objective function f , eg. a Gaussian process.
 - Use an *acquisition function* to direct sampling to areas where one will have an increased probability of finding the optimum.

Optimization algorithm

In the start, select a kernel to model the objective function and choose an acquisition function $A(x)$. Now assume we are at time n where the n^{th} data-point $x^{(n)}$ and the corresponding value $y^{(n)}$ has just been sampled.

1. Update the posterior distribution with $y^{(n)}$.
2. Find the next sampling point $x^{(n+1)}$ by optimizing

$$x^{(n+1)} = \arg \max_x A \left(x \mid x^{(1)}, \dots, x^{(n)} \right)$$

3. Obtain a (possibly noisy) sample $y^{(n+1)} = f(x^{(n+1)}) + \epsilon^{(n+1)}$.

Common acquisition functions

Let f^* denote the maximum value for f found so far, and let $\gamma_x = \frac{\mu_x - f^*}{\sigma_x}$.

1. Probability of Improvement:

$$A(x, f^*) = P(f_x > f^*) = \Phi(\gamma_x),$$

2. Expected Improvement:

$$A(x, f^*) = \mathbb{E}[\max\{f_x - f^*, 0\}] = \sigma_x \gamma_x \Phi(\gamma_x) - \phi(\gamma_x)$$

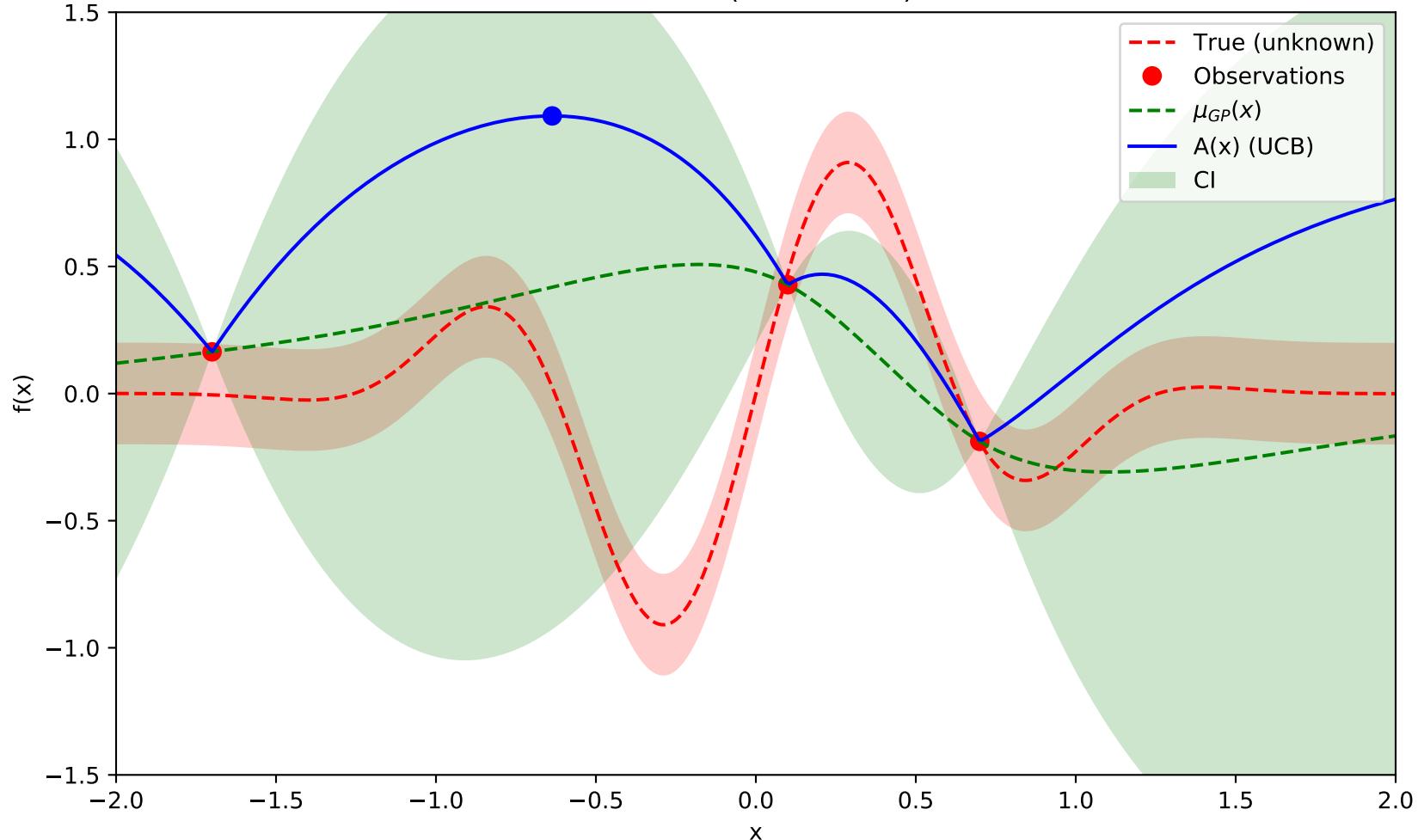
3. Upper Confidence Bound (UCB):

$$A(x) = \mu_x + \kappa \sigma_x$$

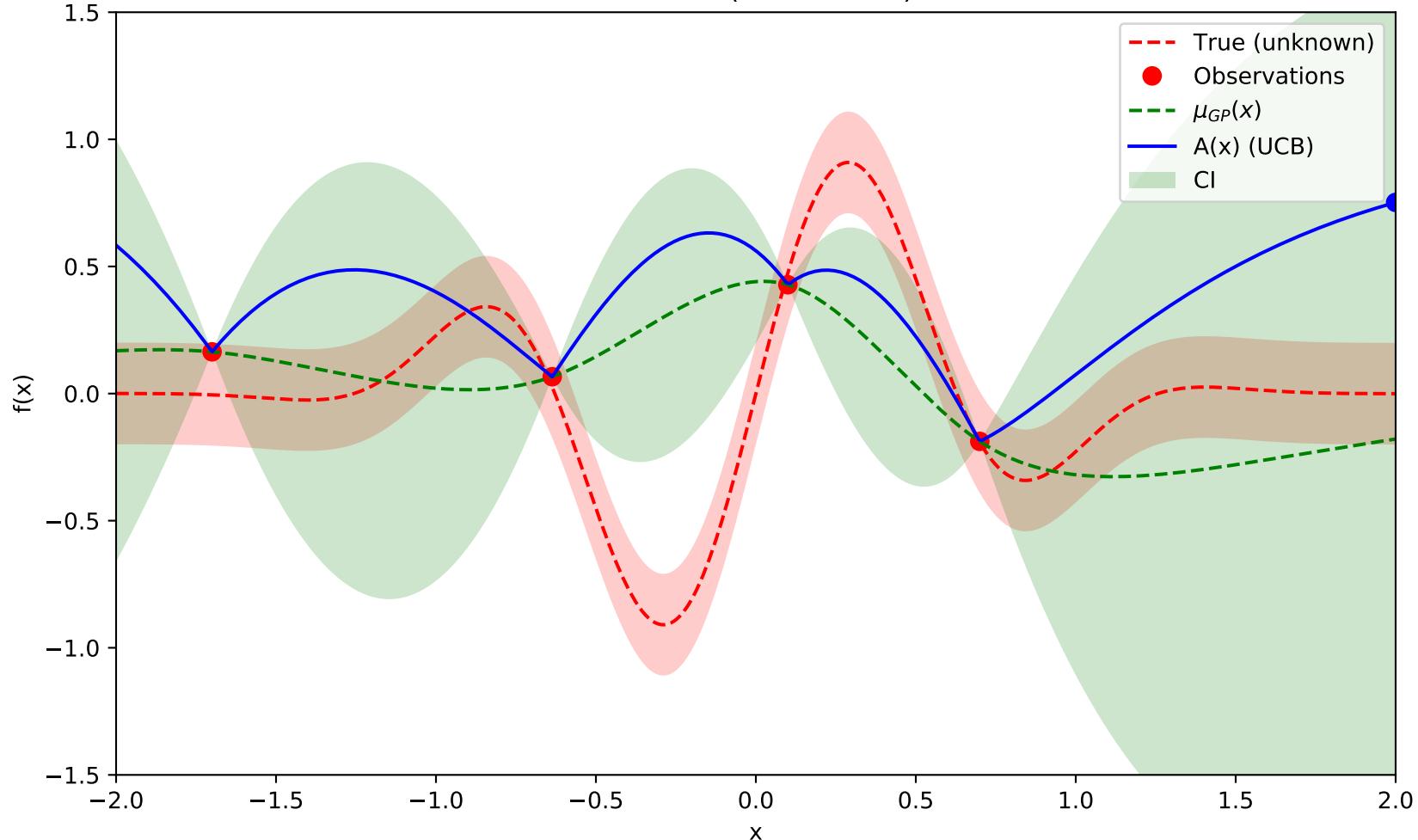
(Φ and ϕ denote the cdf and pdf of the standard normal distribution)

- The acquisition functions are relatively simple functions of μ_x and σ_x .
- Thus, Gaussian process regression replaces the original intractable objective function f with a tractable acquisition function which can be optimized by conventional methods, eg. gradient descent.

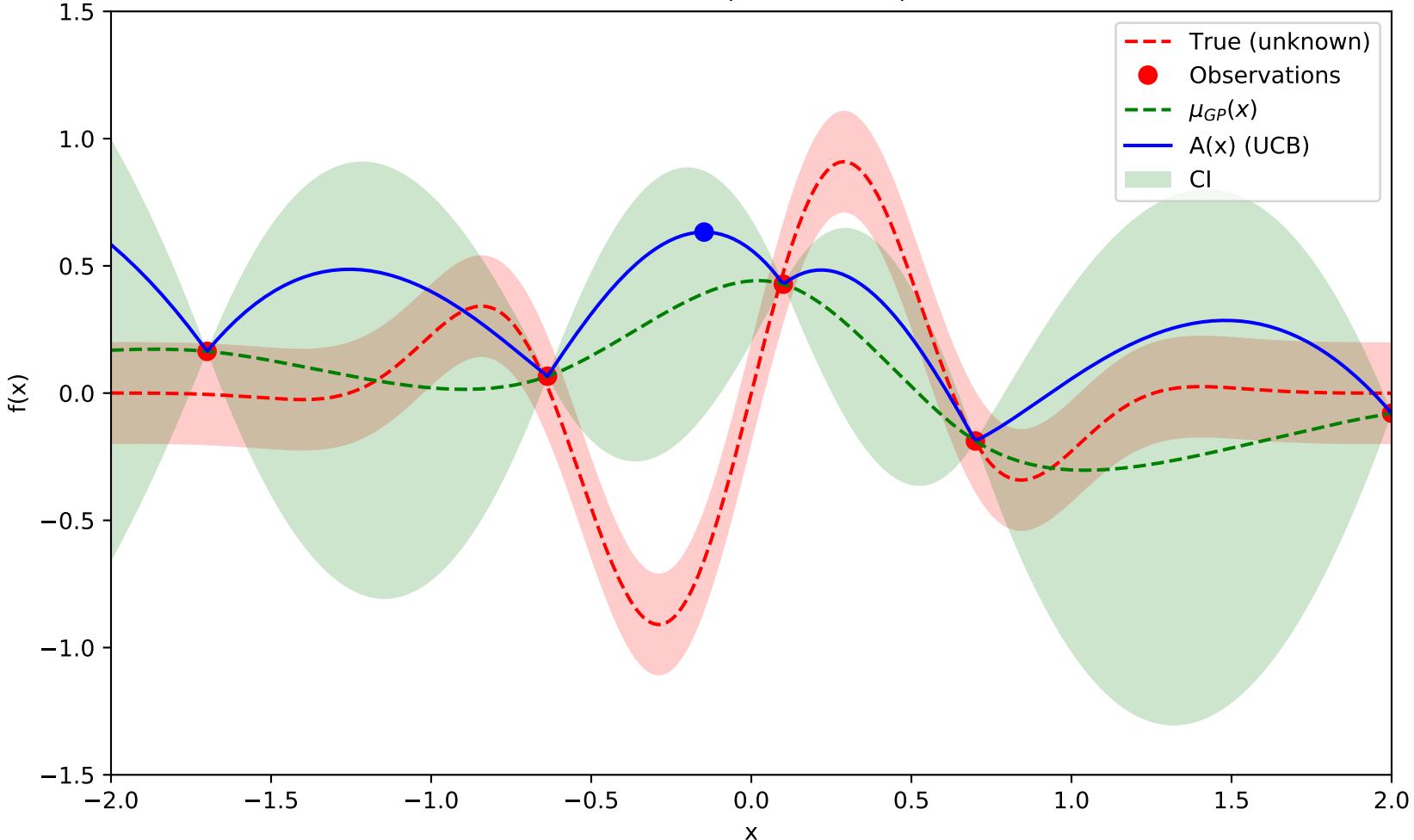
$$f^* = 0.43 \text{ (at } x^* = 0.10\text{)}$$



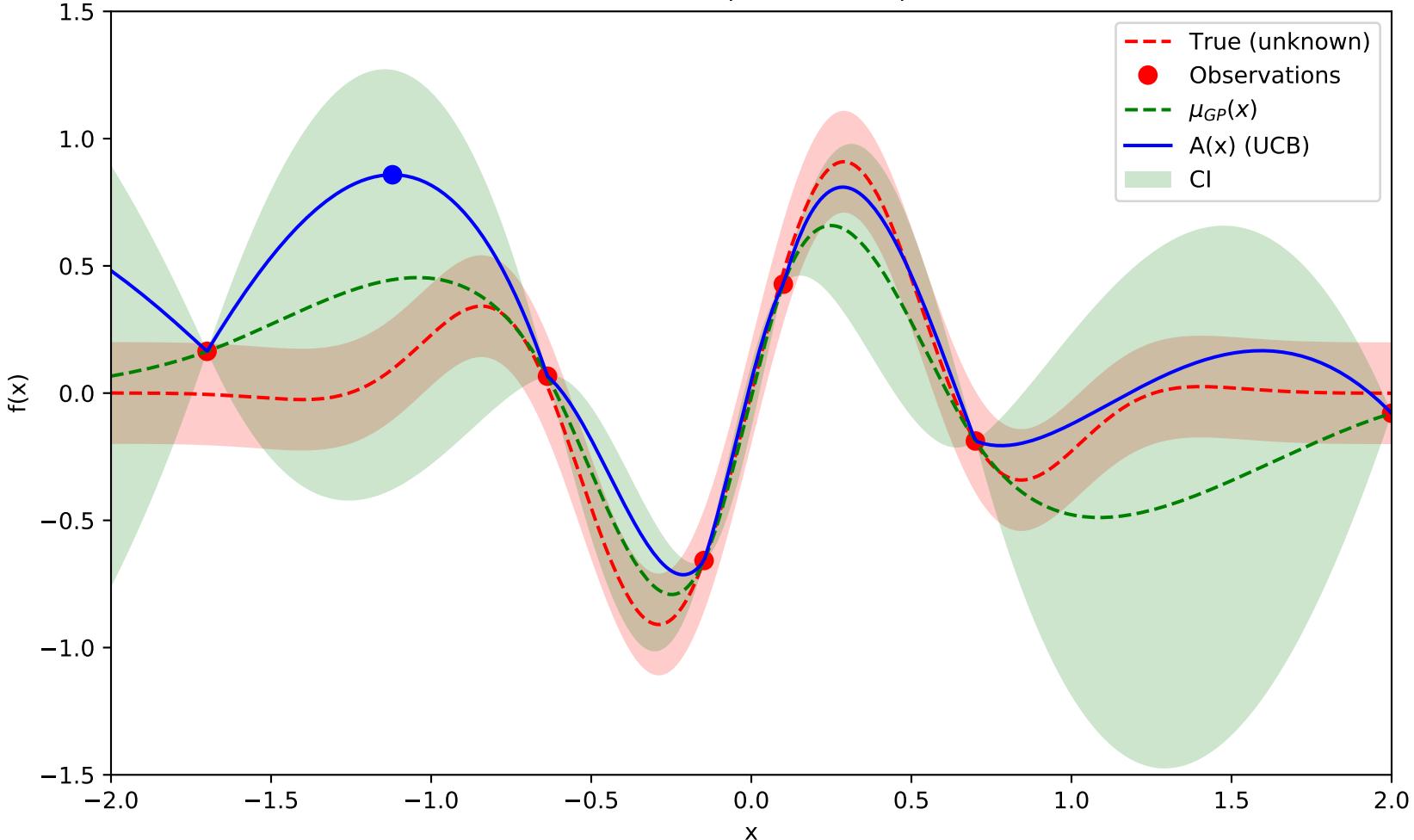
$$f^* = 0.43 \text{ (at } x^* = 0.10\text{)}$$



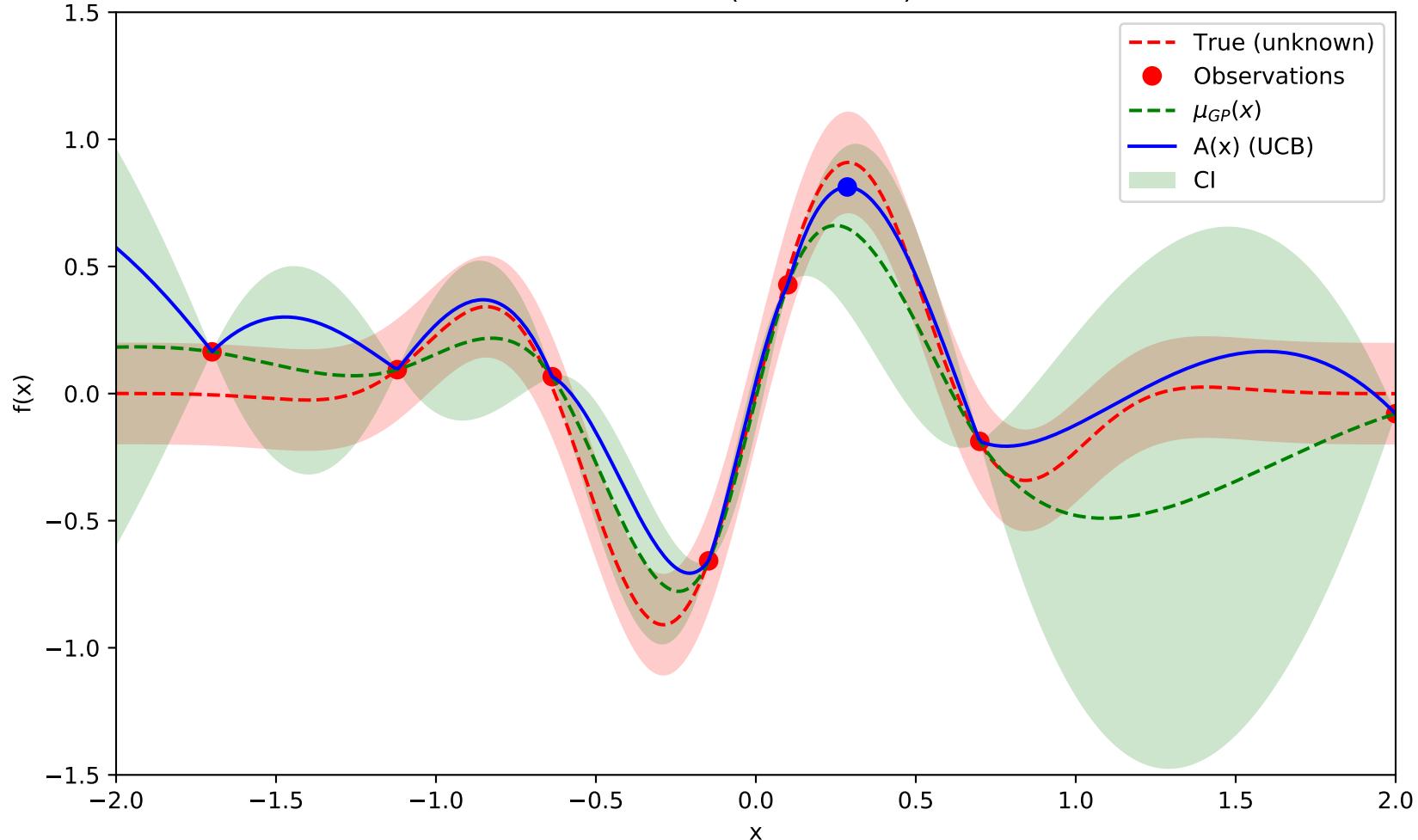
$$f^* = 0.43 \text{ (at } x^* = 0.10\text{)}$$



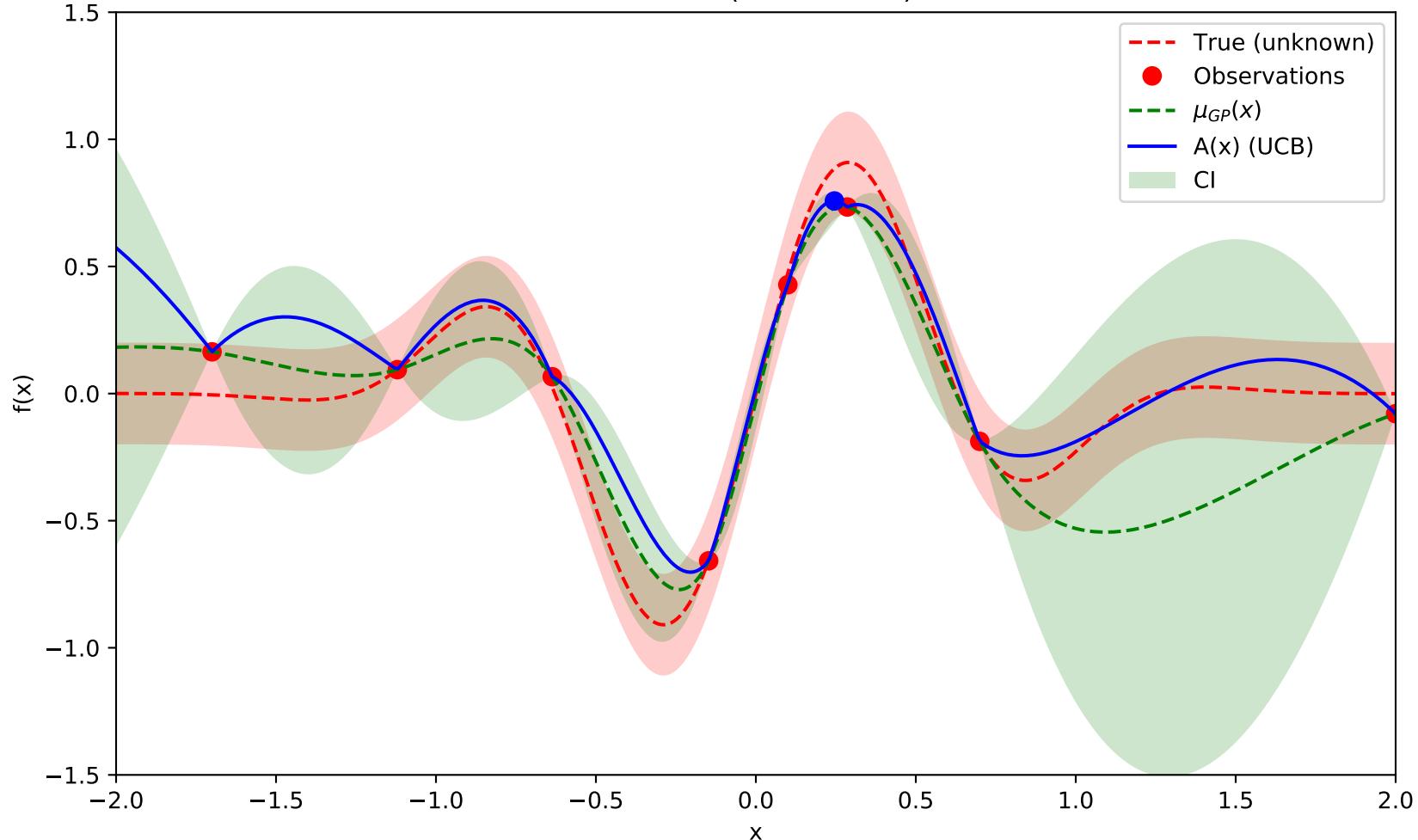
$$f^* = 0.43 \text{ (at } x^* = 0.10\text{)}$$



$$f^* = 0.43 \text{ (at } x^* = 0.10\text{)}$$



$$f^* = 0.73 \text{ (at } x^* = 0.29\text{)}$$



Code

```
from scipy.optimize import minimize
from scipy.stats import norm
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, Matern

gp = GaussianProcessRegressor(kernel=RBF(length_scale=1.0))
gp.fit(xi, yi)
mu_x, sigma_x = gp.predict(x, return_std=True)
```

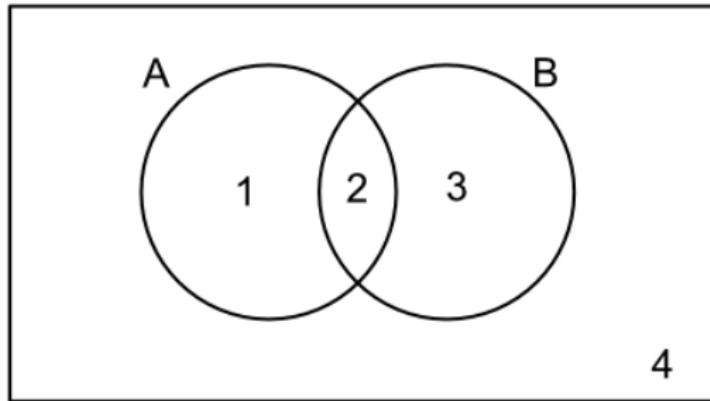
Graphical Models

March 4, 2019

Overview

- 1 Probability Review
- 2 Graph Theory
- 3 Undirected Graphical Models

Union Bound



$$P(A^c) = 1 - P(A) \quad (1)$$

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) \quad (2)$$

$$P(A \cup B) \leq P(A) + P(B) \quad (3)$$

Expectation

The expectation (mean) of a random variable X can be expressed as

$$E(X) = X_{\text{mean}} = \sum_{x \in \mathcal{X}} x P(X = x). \quad (4)$$

The variance and covariance can be defined therefore in terms of the expectation, where

$$\text{Var}(X) = E((X - E(X))^2) = E(X^2) - E(X)^2, \quad (5)$$

$$\text{Cov}(X) = E(XY) - E(X)E(Y). \quad (6)$$

Conditional expectations and variances follow from the conditional distributions

$$E(X | Y = y) = \sum_{x \in \mathcal{X}} x P(X = x | Y = y). \quad (7)$$

Independence

Two random variables are independent when the probability distribution of one random variable does not affect the other. More concretely, two random variables X and Y are independent, that is, $X \perp Y$, if and only if

$$P(X = x, Y = y) = P(X = x)P(Y = y), \quad \text{for all } x \in \mathcal{X}, y \in \mathcal{Y}. \quad (8)$$

If X and Y are continuous with joint density function $f_{X,Y}(x,y)$, then the above condition reduces to finding functions $h(x)$ and $g(y)$ such that

$$f_{X,Y}(x,y) = h(x)g(y). \quad (9)$$

Conditional Independence

Two random variables X and Y are conditionally independent given a third variable Z , denoted as $X \perp Y | Z$, if and only if

$$P(X = x, Y = y | Z = z) = P(X = x | Z = z)P(Y = y | Z = z), \quad (10)$$

for all $x \in \mathcal{X}, y \in \mathcal{Y}, z \in \mathcal{Z}$.

This is equivalent to saying

$$P(X = x | Y = y, Z = z) = P(X = x | Z = z).$$

Note that $X \perp Y | Z$ does not imply that $X \perp Y$, and vice versa.

Conditional independence relations

- Symmetry:

$$X \perp Y | Z \implies Y \perp X | Z$$

- Decomposition:

$$X \perp Y, W | Z \implies X \perp Y | Z \quad (\text{and } X \perp W | Z)$$

- Weak union:

$$X \perp Y, W | Z \implies X \perp Y | Z, W \quad (\text{and } X \perp W | Y, Z)$$

- Contraction:

$$X \perp Y | Z \text{ and } X \perp W | Y, Z \implies X \perp Y, W | Z$$

Graph Theory Preliminaries

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of:

- A set of nodes/vertices $\mathcal{V} = \{1, 2, \dots, n\}$. Illustrated by a dot or cross.
- A set of edges $\mathcal{E} = \{(u, v) \mid u, v \in \mathcal{V}\}$, which consists of node pairs. This is illustrated by a line connecting the two nodes in the node pair.
- If the pairs in \mathcal{E} are unordered, that is $(u, v) = (v, u)$, then graph is called undirected (the lines in the graph have no directional arrows). If the pairs are ordered, that is $(u, v) \neq (v, u)$, then the graph is called directed (the lines in the graph has arrows on them).
- We will be dealing with *simple* graphs here, which means we do not allow self loops and multiple edges between the same node pairs. We also will not be considering graphs with a mixture of ordered and unordered edges.

Directed and Undirected Graphs

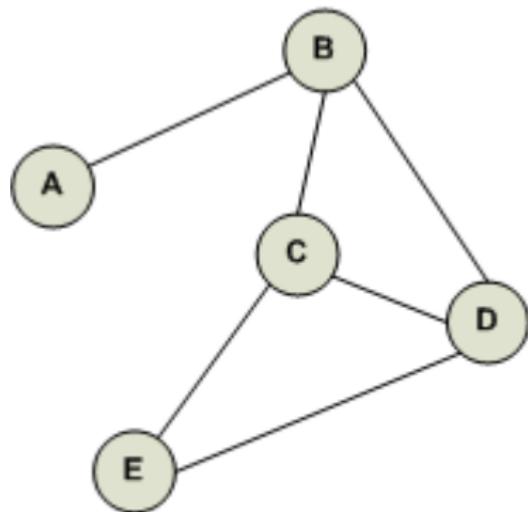


Fig 1. Undirected Graph

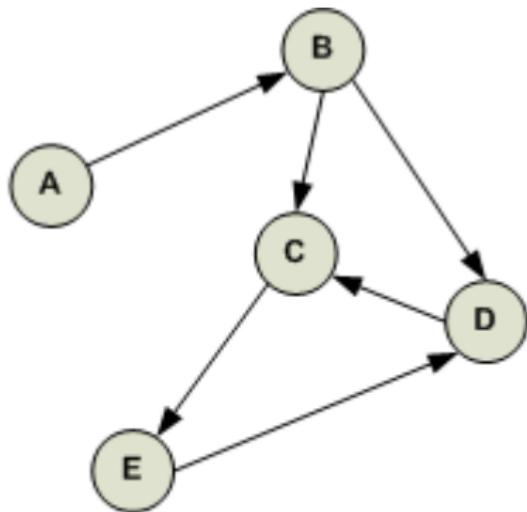
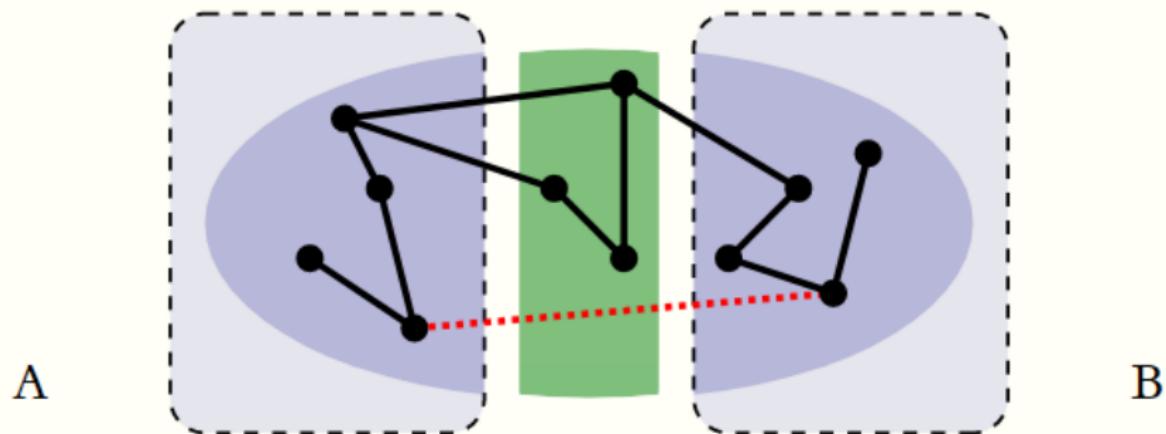


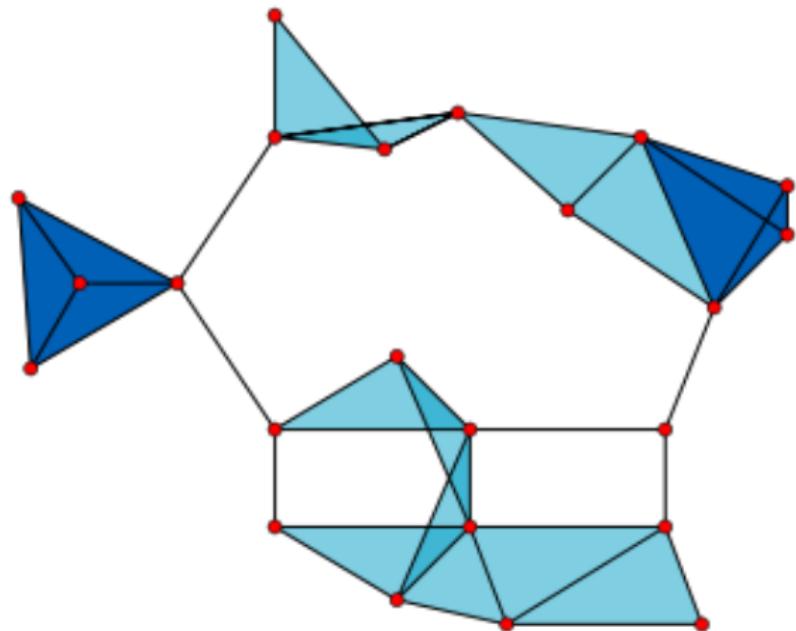
Fig 2. Directed Graph

Separator Node Sets

vertex separator



Graph Cliques



Graphical Models

A graphical model is a multivariate probability distribution associated with a graph representation, where the nodes of the graph represent the variables in the distribution and the edges represent the relationships between nodes.

If the associated graph is directed (undirected), then the graphical model is directed (undirected).

In this lecture, we will be discussing undirected graphical models.

Undirected Graphical Models

Which is more important, independence or conditional independence?

One main type of undirected graphical model is known as the Markov random field (MRF).

In a MRF, if no edge exists between two nodes u and v , then X_u must be conditionally independent of X_v given the rest of the variables.

Markov Random Fields

Markov Properties on Undirected Graphs

Fact: (F) \Rightarrow (G) \Rightarrow (L) \Rightarrow (P).

(F) P factorizes according to G : $p(x) = \prod_{C \in C} \phi_C(x_C)$.

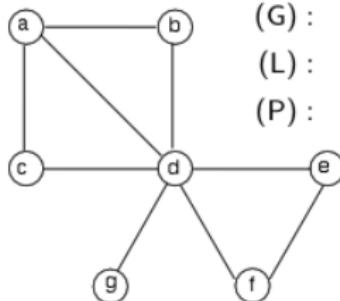
(G) For any disjoint subsets A, B, S of V , $A \perp B | S \Rightarrow X_A \perp X_B | X_S$.

(L) For all $v \in V$, $X_v \perp X_{V \setminus \text{cl}(v)} | X_{\text{bd}(v)}$.

(P) For all pairs of non-adjacent vertices (v, v') in G , $X_v \perp X_{v'} | V \setminus \{v, v'\}$.

Illustration:

G :



(F) : $p(x) = \phi_1(x_a, x_b, x_d) \phi_2(x_a, x_c, x_d) \phi_3(x_d, x_e, x_f) \phi_4(x_d, x_g)$

(This is the most general form of p that satisfies (F).)

(G) : $X_{\{a,b\}} \perp X_{\{e,g\}} | X_d$, $X_{\{c,g\}} \perp X_{\{b,f\}} | X_{\{a,d\}}$, etc.

(L) : $X_a \perp X_{\{e,f,g\}} | X_{\{b,c,d\}}$, $X_g \perp X_{\{a,b,c,e,f\}} | X_d$, etc.

(P) : $X_a \perp X_e | X_{\{b,c,d,e,f\}}$, $X_c \perp X_b | X_{\{a,d,e,f,g\}}$, etc.



Multivariate Gaussian Distribution

The Gaussian distribution $\mathbf{X} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ has the density function,

$$f_{\mathbf{X}}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^p |\boldsymbol{\Sigma}|}} \exp \left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Omega} (\mathbf{x} - \boldsymbol{\mu}) \right\}, \quad (11)$$

where $\boldsymbol{\Omega} = \boldsymbol{\Sigma}^{-1}$, The edges of the associated graph \mathcal{G} are exactly those corresponding to the nonzero entries of $\boldsymbol{\Omega}$, the precision matrix.

Example

Let $\mathbf{X} = (X_1, X_2, X_3)$ be a multivariate Gaussian distribution with zero mean and covariance matrix

$$\boldsymbol{\Sigma} = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 1 & -1 \\ 0 & -1 & 3 \end{bmatrix}, \quad \boldsymbol{\Omega} = \begin{bmatrix} 2 & -3 & -1 \\ -3 & 6 & 2 \\ -1 & 2 & 1 \end{bmatrix}.$$

Observe that $X_1 \perp X_3$, or $X_1 \perp X_3 | \mathbf{X}_S$, where S is the null set. However, the graph according to the precision matrix is the complete graph, as the precision matrix has no off-diagonal zero entries. So while graph separability implies conditional independence, the converse does not necessarily hold.

Ising Models

The Ising model $\mathbf{X} = (X_1, \dots, X_n)$, used in the study of ferromagnetism, has state space $\mathcal{X} = \{-1, +1\}^n$ and density function

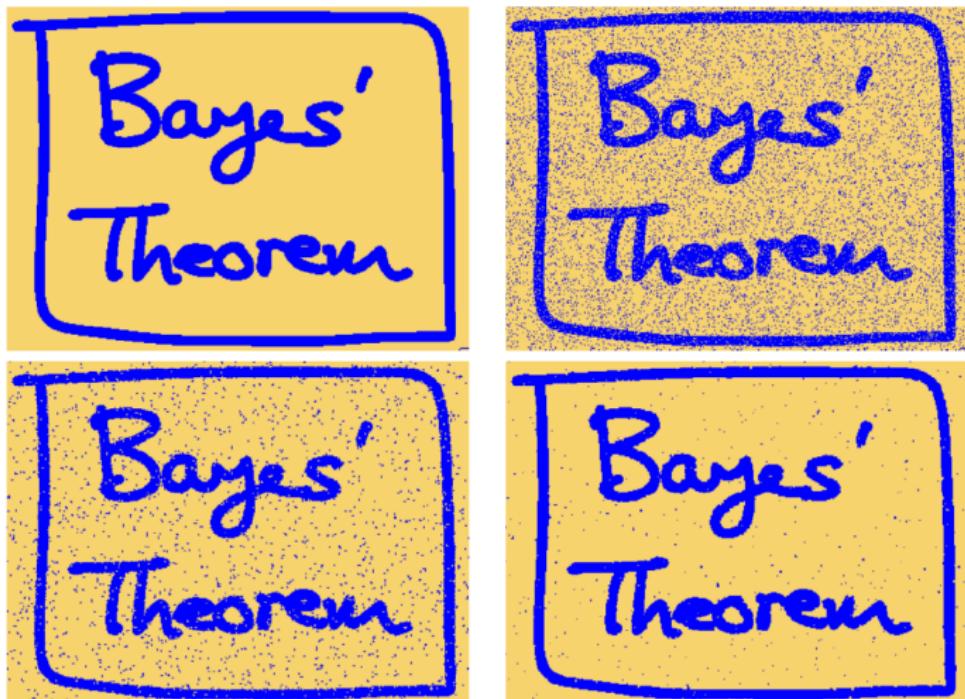
$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \exp \left\{ \sum_{ij} J_{ij} x_i x_j + \sum_i h_i x_i \right\}, \quad (12)$$

where the associated graph $\mathcal{G} = \mathcal{V}, \mathcal{E}$ has an edge between nodes i and j if and only if $J_{ij} \neq 0$.

The normalizing constant has the form

$$Z = \sum_{\mathbf{x} \in \mathcal{X}} \exp \left\{ \sum_{ij} J_{ij} x_i x_j + \sum_i h_i x_i \right\}. \quad (13)$$

Denoising



Graph Learning

The graph learning problem involves learning the underlying graph structure given samples from the distribution.

In the case of the Gaussian graphical model, we want to learn the non-zero structure of the precision matrix given the sample covariance matrix.

Difficult problem especially when dimension n is large.

Graphical Lasso

From samples $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}$, the sample covariance is defined by

$$S = \frac{1}{N-1} \sum_{i=1}^N \left(\mathbf{x}^{(i)} - \sum_{j=1}^N \mathbf{x}^{(j)}/N \right) \left(\mathbf{x}^{(i)} - \sum_{j=1}^N \mathbf{x}^{(j)}/N \right)^T. \quad (14)$$

The graphical lasso uses maximum likelihood and sparsity in the graph to estimate the precision matrix,

$$\hat{\Omega} = \min_{\Omega} -\log |\Omega| + \text{tr}(S\Omega) + \lambda \|\Omega\|_1. \quad (15)$$

Bayesian networks

Overview

- 1 Directed Acyclic Graphs
- 2 Directed Graphical Models
- 3 Independence and Markov Properties

Graph Terminology

$\mathcal{G} = (V, E)$ **directed graph**. V **vertices**. E **edges** (ordered pairs of vertices)

X, Y **adjacent** if $X \rightarrow Y$ edge. Y **child** of X . X **parent** of Y .

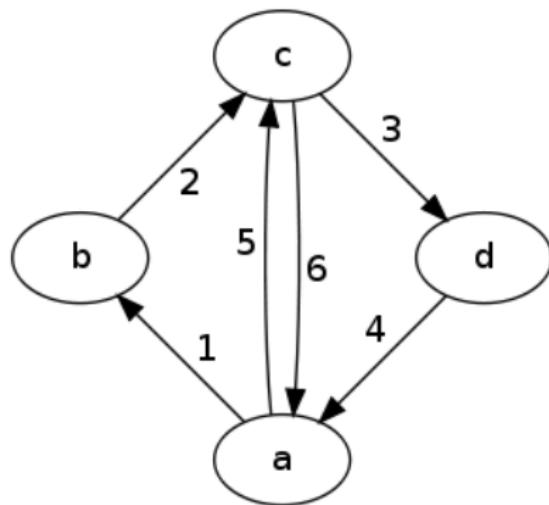
$X \rightarrow \dots \rightarrow Y$ **directed path**. Y **descendant** of X . X **ancestor** of Y .

$X \leftarrow \dots \rightarrow Y$ **undirected path** (ignore direction of arrows).

$X \rightarrow Y \leftarrow Z$ **collider**. $X \rightarrow Y \rightarrow Z$, $X \leftarrow Y \leftarrow Z$, $X \leftarrow Y \rightarrow Z$ **non-colliders**.

$X \rightarrow \dots \rightarrow X$ **cycle**. \mathcal{G} **directed acyclic graph** if no cycles.

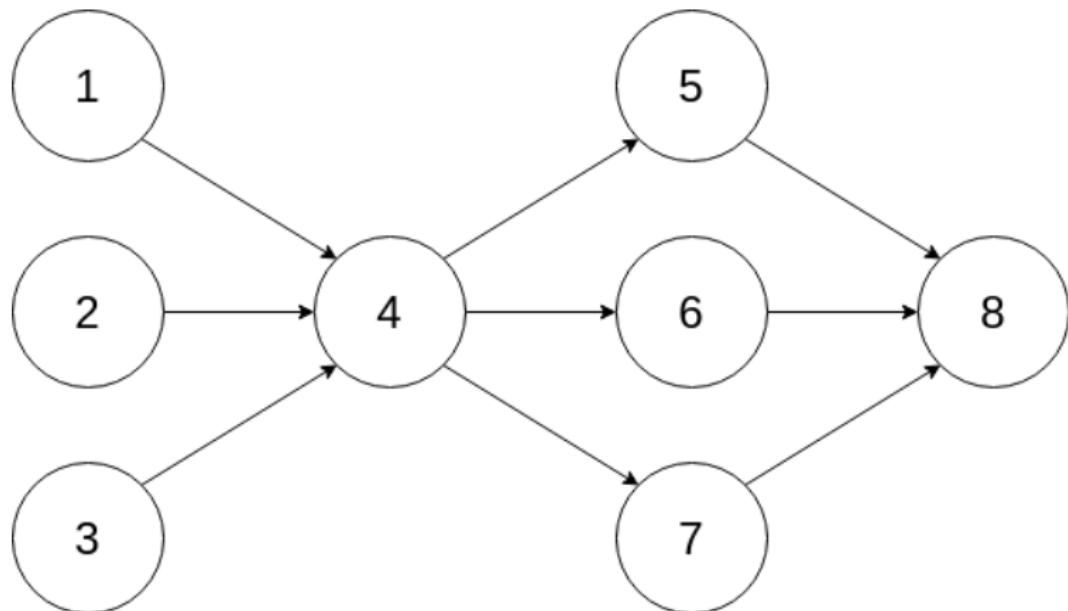
Directed Graphs with Cycles



$\{1, 2, 3, 4\}$ and $\{3, 4, 5\}$ are cycles.

$\{3, 4, 6\}$ is not a cycle.

Directed Acyclic Graphs (DAG)



Directed Graphical Models

A directed graphical model or Bayesian network consists of a multivariate random variable $\mathbf{X} = (X_1, \dots, X_n)$ and a corresponding graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where

- $\mathcal{V} = \{1, \dots, n\}$, where the variable X_i is represented by node i ,
- $(i, j) \in \mathcal{E}$ is denoted by an arrow connecting i to j ,
- the probability mass (density) function of \mathbf{X} satisfies the factorization property.

Factorization

We denote the parents of node i by $\text{Pa}(i)$.

A probability mass function $P(\mathbf{X} = \mathbf{x})$ satisfies the factorization property with respect to a DAG if

$$P(\mathbf{X} = \mathbf{x}) = \prod_{i=1}^n P(X_i = x_i \mid \text{Pa}(i)). \quad (1)$$

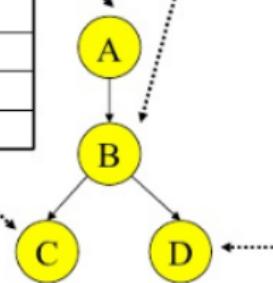
Example

A Set of Tables for Each Node

A	P(A)
false	0.6
true	0.4

A	B	P(B A)
false	false	0.01
false	true	0.99
true	false	0.7
true	true	0.3

B	C	P(C B)
false	false	0.4
false	true	0.6
true	false	0.9
true	true	0.1



Each node X_i has a conditional probability distribution $P(X_i | \text{Parents}(X_i))$ that quantifies the effect of the parents on the node

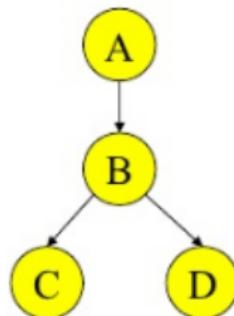
The parameters are the probabilities in these conditional probability tables (CPTs)

B	D	P(D B)
false	false	0.02
false	true	0.98
true	false	0.05
true	true	0.95

Using a Bayesian Network Example

Using the network in the example, suppose you want to calculate:

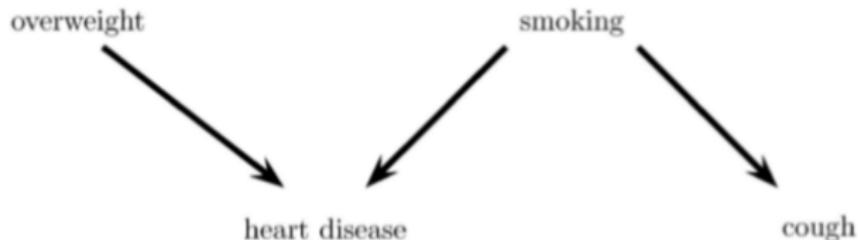
$$\begin{aligned} & P(A = \text{true}, B = \text{true}, C = \text{true}, D = \text{true}) \\ &= P(A = \text{true}) * P(B = \text{true} | A = \text{true}) * \\ &\quad P(C = \text{true} | B = \text{true}) P(D = \text{true} | B = \text{true}) \\ &= (0.4)*(0.3)*(0.1)*(0.95) \end{aligned}$$



Examples

EXAMPLES

Smoking



$$f(\text{overweight}, \text{smoking}, \text{heart disease}, \text{cough})$$

joint probability

$$= f(\text{overweight}) \times f(\text{smoking})$$

root probabilities

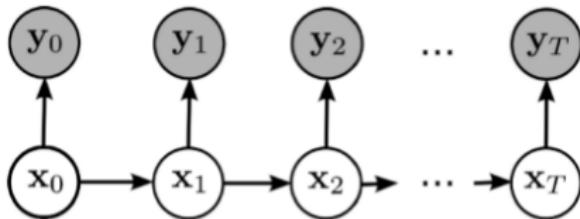
$$\times f(\text{heart disease} | \text{overweight}, \text{smoking})$$

transition
probabilities

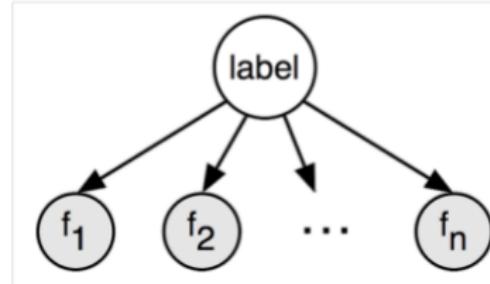
$$\times f(\text{cough} | \text{smoking}).$$

Example

Hidden Markov Model

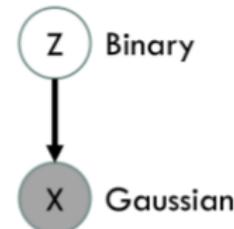


Naïve Bayes

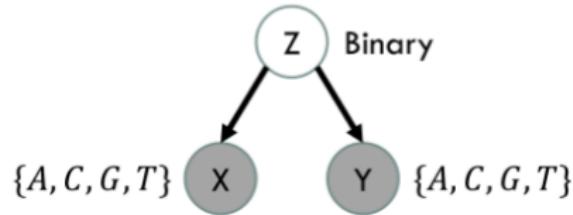


Example

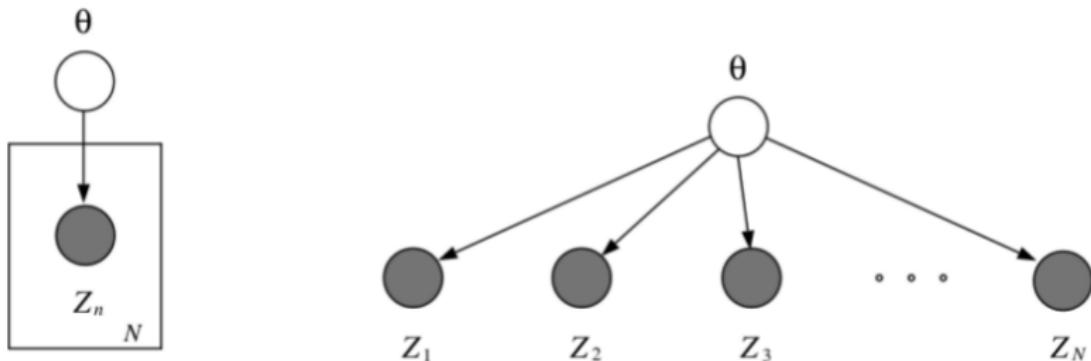
Gaussian Mixtures



Phylogenetic Models



Examples

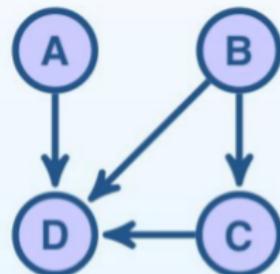


When N is large, the joint distribution is difficult to analyze. A directed graphical model helps us to factorize the model so that analysis can be done more efficiently.

$$P(\mathbf{Z} = \mathbf{z}, \theta = x) = P(\theta = x) \prod_{i=1}^N P(Z_i = z_i \mid \theta = x). \quad (2)$$

Directed Gaussian Graphical Model

Parametrize the model using *structural equation modelling (SEM)*.



Gaussian

$$A = \varepsilon_A, \quad \varepsilon_A, \varepsilon_B, \varepsilon_C, \varepsilon_D \sim \mathcal{N}(0, 1)$$

$$B = \varepsilon_B$$

$$C = \lambda_{BC}B + \varepsilon_C$$

$$D = \lambda_{AD}A + \lambda_{BD}B + \lambda_{CD}C + \varepsilon_D$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -\lambda_{BC} & 1 & 0 \\ -\lambda_{AD} & -\lambda_{BD} & -\lambda_{CD} & 1 \end{pmatrix} \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} \sim \mathcal{N}(0, \text{Id})$$

Directed Gaussian Graphical Model

Let \boldsymbol{K} be the matrix of the left hand side of the previous expression, and let $\mathbf{A} = (A, B, C, D)$ be the multivariate random variable. Then the graphical model in the previous slide implies that \boldsymbol{KA} is the standard normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$, which further implies that

$$\mathbf{A} \sim \mathcal{N}\left(\mathbf{0}, \boldsymbol{K}^{-1} \left(\boldsymbol{K}^{-1}\right)^T\right). \quad (3)$$

The Half Way Point

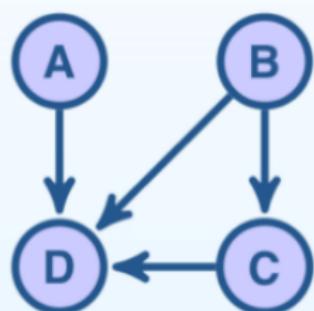
5 Minutes Break

Local Markov Property

We say that a distribution \mathbb{P} satisfies the **local Markov property** with respect to \mathcal{G} if for all variables W ,

$$W \perp \tilde{W} \mid \pi_W$$

where π_W are the parents of W , and \tilde{W} are the variables which are neither parents nor descendants of W .



$$\begin{aligned}A &\perp\!\!\!\perp B, C \\B &\perp\!\!\!\perp A \\C &\perp\!\!\!\perp A \mid B \\D &\perp\!\!\!\perp \emptyset \mid A, B, C\end{aligned}$$

D-Separation

Consider the following undirected path from X to Z :

$$X - Y_1 - Y_2 - \cdots - Y_n - Z.$$

Let W be some subset of vertices that do not contain X or Z .



Think of each intermediate vertex as a gate, and W a set of keys.

1. Collider gates are usually closed; all other gates are usually open.
2. If a collider or one of its descendants is in W , then that gate is opened.
3. If a non-collider is in W , then that gate is closed.

If all the gates are open, we say that X and Z are **d-connected** given W .

If we cannot find any such path, then X and Z are **d-separated** given W .

Sets S and T are **d-separated** given W if it is true for all $X \in S, Z \in T$.

D-Separation

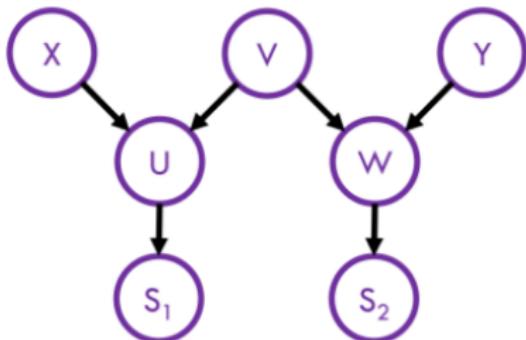
EXAMPLES

Three layer DAG

X and Y are d-separated (given the empty set);

X and Y are d-connected given $\{S_1, S_2\}$;

X and Y are d-separated given $\{S_1, S_2, V\}$.

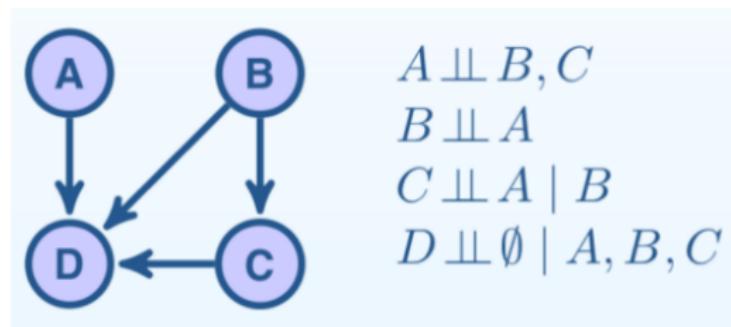


Global Markov Property

We say that a distribution \mathbb{P} satisfies the **global Markov property** with respect to \mathcal{G} if

$$S \perp T \mid W$$

for all disjoint subsets S, T, W such that S and T are d-separated given W .



$$\begin{aligned} A &\perp\!\!\!\perp B \mid C \\ A &\perp\!\!\!\perp C \end{aligned}$$

Hammersley-Clifford Theorem

The following are equivalent:

1. \mathbb{P} satisfies the **factorization property** with respect to \mathcal{G} .
2. \mathbb{P} satisfies the **local Markov property** with respect to \mathcal{G} .
3. \mathbb{P} satisfies the **global Markov property** with respect to \mathcal{G} .

Explaining Away

EXPLAINING AWAY



Why does conditioning on a collider lead to dependence?

If you don't know your friend is late:

$$\mathbb{P}(\text{Aliens}|\text{Watch}) = \mathbb{P}(\text{Aliens})$$

Aliens \perp Watch

If you now know your friend is late:

$$\mathbb{P}(\text{Aliens}|\text{Watch}, \text{Late}) < \mathbb{P}(\text{Aliens}|\text{Late})$$

Aliens $\not\perp$ Watch | Late

Knowing his broken watch made him late explains away the possibility that he is late because he was abducted by aliens.

Advantages of Bayesian Network Representations

- Conditional Independence relations can be read off the underlying DAG.
- Can describe causal relationships between variables.
- Can handle incomplete data or latent variables.

REGRESSION



Intended learning outcomes

- Understand the difference between training data and test data.
- Explain bias-variance tradeoff.
 - high bias=underfitting
cannot capture the underlying trend of the data.
 - captures the noise of the data
- Explain what is underfitting and overfitting, how to identify them, and how they determine model selection.
- Give different ways to solve overfitting and underfitting.

INTENDED LEARNING OUTCOMES

Optimization

- Give examples of loss functions, and define empirical risk in terms of the loss function.
- List two general types of algorithms used in optimization, e.g. exact solution, and gradient descent. Outline the broad steps involved in each of them.
- Explain why framing a problem as convex optimization is highly desirable, in terms of speed and local minima.
- Explain the motivation behind performing stochastic gradient descent, rather than traditional gradient descent.



INTENDED LEARNING OUTCOMES

Multivariate Linear Regression

- State the model and the training loss.
- Explain how the ‘constant feature’ trick can be used to reduce the problem to one without the constant parameter θ_0 .
- Describe two training algorithms that may be applied.
- Derive the gradient of the training loss.
- Derive the formula for the exact solution.
- Describe two potential weaknesses of the exact solution, and possible solutions for these weaknesses.
- Apply the above algorithms to a given data set.



INTENDED LEARNING OUTCOMES

Regularization

- Explain why regularization can help with generalization.
- State the training loss and test loss in ridge regression.
- Identify the regularizer and regularization parameter in the training loss of a given machine learning problem.
- Explain why regularization solves the invertibility problem in traditional linear regression.
- Describe the difference in gradient descent between traditional and regularized linear regression.
- Describe how the test loss and training loss varies with the regularization parameter.



CLASSIFICATION



Intended learning outcomes

- Understand what decision boundaries and regions are with respect to a classifier.
- Define what it means for a dataset to be linearly separable, and give an example of a dataset that is not.

INTENDED LEARNING OUTCOMES

Logistic Regression

- Write down the sigmoid function and logistic loss function.
- Describe the model as a set of sigmoid neurons which predict the probability of the labels given the features.
- Derive the label predictions from the label probabilities.
Describe the decision boundary.
- Derive the training loss from the likelihood of the data, and write it in terms of the logistic loss.
- Derive the training gradient, and describe an algorithm for performing logistic regression.



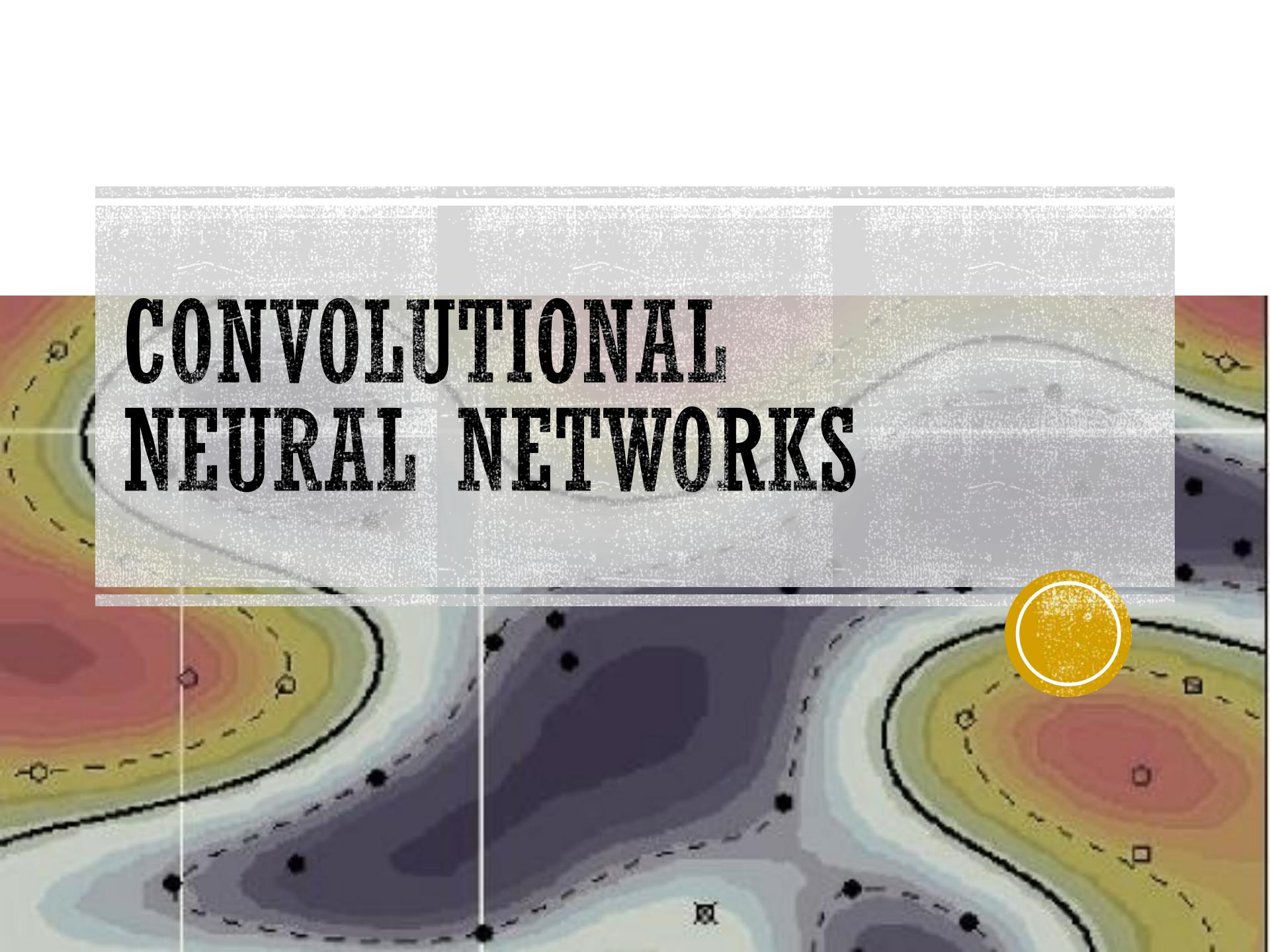
DEEP LEARNING



Intended learning outcomes

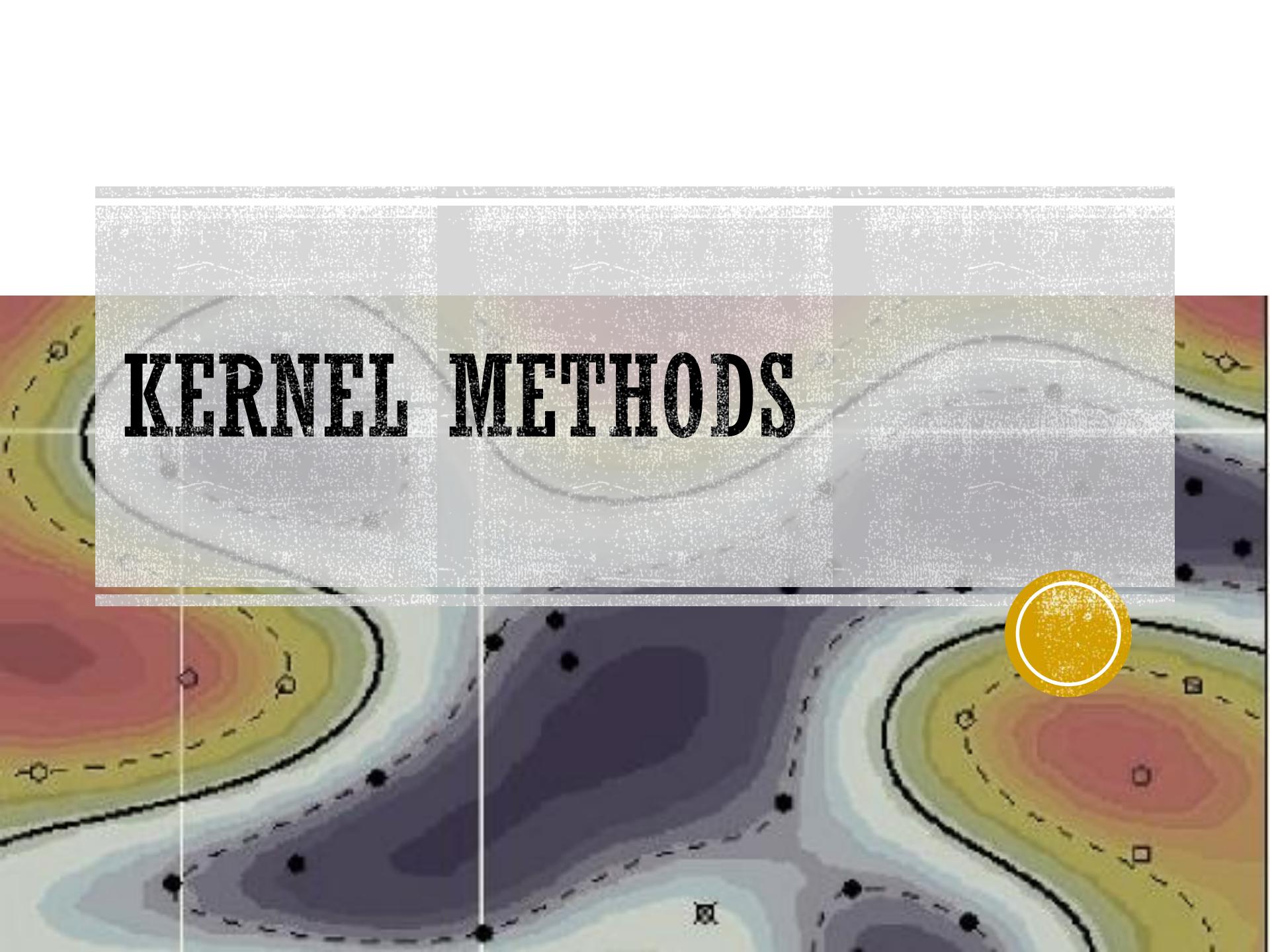
- Describe how the output of an artificial neuron relates to its inputs.
- Give examples of commonly used activation functions.
- Know how to efficiently compute the error signal in backpropagation, and be able to explain why it is done this way.
- Understand the statement of the universal approximation theorem.
- Give reasons for the recent successes of deep learning, and explain how it differs from classical machine learning.

CONVOLUTIONAL NEURAL NETWORKS

A weather map featuring various colored contour lines (yellow, red, purple) and black dots representing data points. A prominent yellow circle highlights a circular region in the upper right quadrant of the map.

Intended learning outcomes

- Understand how convolution is performed on 2D arrays.
- Explain why convolution is used for image recognition, and describe how it builds up a hierarchy of features.
- Know the definitions of the terms "stride", "padding" and "filter size".

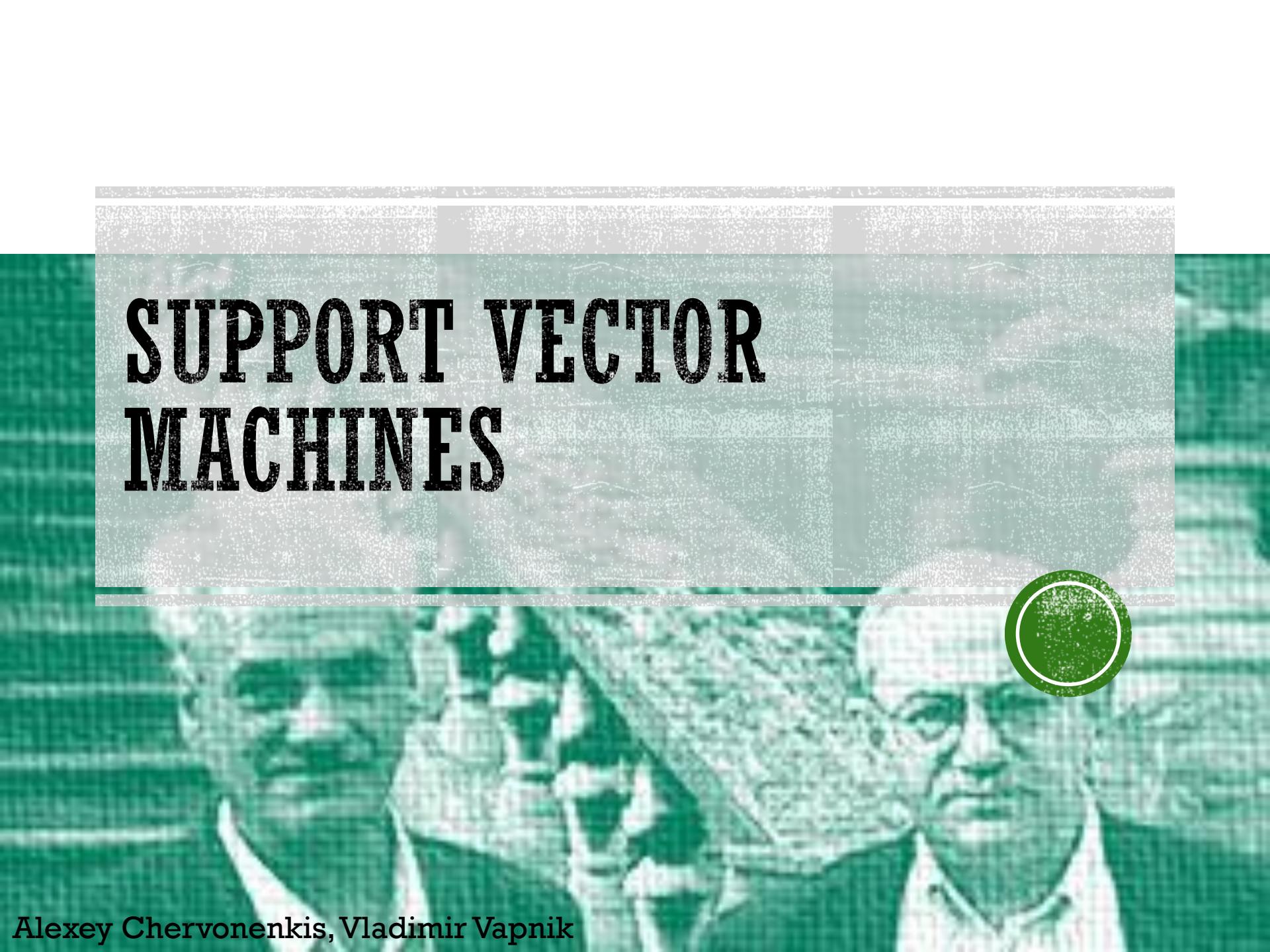


KERNEL METHODS

Intended learning outcomes

- Give the definition of a kernel function, and be able to ascertain what a valid kernel function is.
- Describe how feature maps change a linear algorithm into a non-linear one.
- Explain what the kernel trick is, and how it bypasses having to define a computationally intensive feature map.
- Understand how kernels typically arise in dual representations of the problem.

SUPPORT VECTOR MACHINES



Alexey Chervonenkis, Vladimir Vapnik

INTENDED LEARNING OUTCOMES

Support Vector Machines

- Write down the primal problem, and explain how it is derived from the maximum margin problem.
- Write down the dual problem, and identify the kernel. Describe how the optimal θ is derived from the $\alpha_{x,y}$'s. Describe in terms of the $\alpha_{x,y}$'s, how to do prediction.
- Define support vectors, both geometrically and in terms of the $\alpha_{x,y}$'s. Recognize that most of the $\alpha_{x,y}$'s are zero.



INTENDED LEARNING OUTCOMES

Extensions

- Describe the dual problem for the SVM with offset.
- Describe the primal problem for SVM with slack variables.
Show that the primal is equivalent to regularized hinge loss.
Explain how the regularizing parameter λ affects the margins.
Describe the dual problem in terms of box constraints.



Gaussian processes for regression and optimization

Intended learning outcomes

- Define a Gaussian process and understand them as infinite-dimensional generalizations of multivariate Gaussian random variables.
- Understand the difference between parametric and non-parametric methods.
- Use the formulas for conditional Gaussian distributions to perform Gaussian process prediction.
- Define what an acquisition function is and list commonly used examples.
- Understand how acquisition functions are used in conjunction with Gaussian process prediction to do Bayesian optimization.

Graphical models

Intended learning outcomes

- Understand conditional independence of random variables.
- Understand what a Markov random field is, and the properties its graph and joint distribution must satisfy.
- Describe what d-separation is.
- Describe a Bayesian network, and the properties it must fulfill.

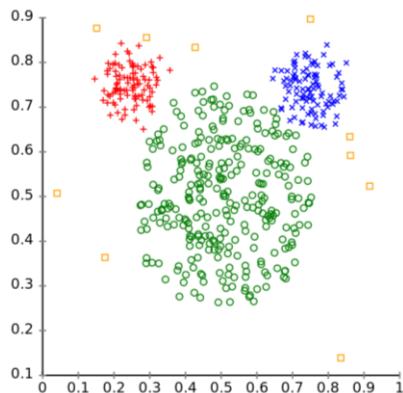


CLUSTERING



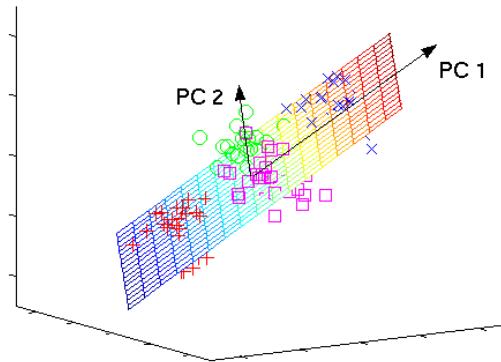
UNSUPERVISED LEARNING

- No labels/responses. Finding structure in data.
- Dimensionality Reduction.



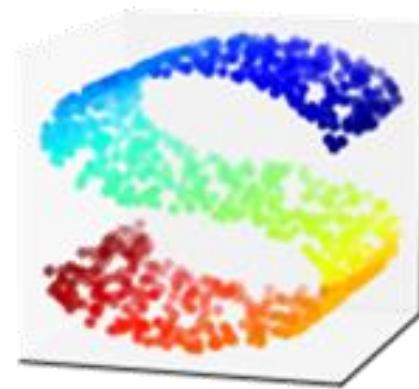
Clustering

$$T: \mathbb{R}^d \rightarrow \{1, 2, \dots, k\}$$



Subspace Learning

$$T: \mathbb{R}^d \rightarrow \mathbb{R}^m$$

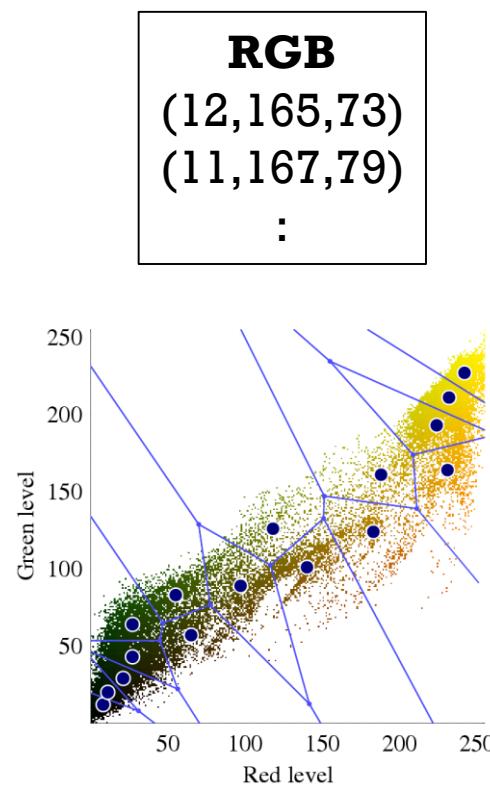


Manifold Learning



USES OF UNSUPERVISED LEARNING

- Data compression



Labels

3
43
:

Dictionary

1 ~ (10, 160, 70)
2 ~ (40, 240, 20)
:



USES OF UNSUPERVISED LEARNING

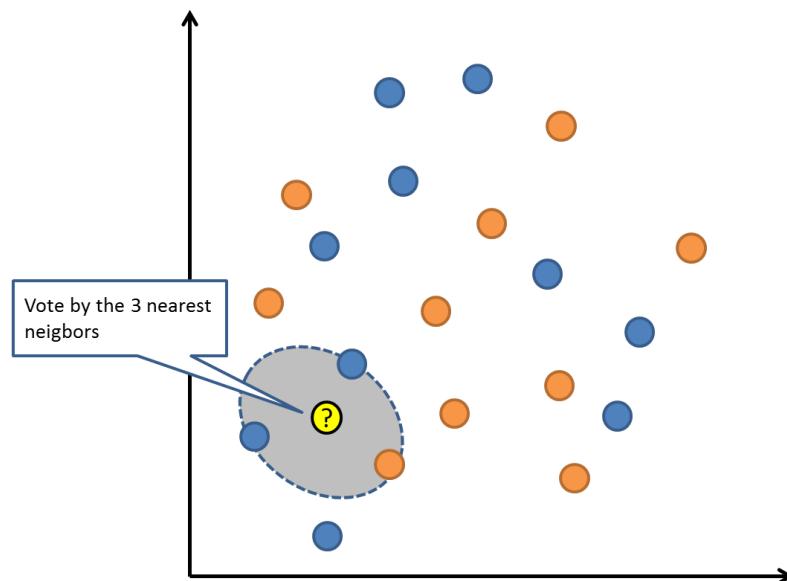
- Improve classification/regression (semi-supervised learning)
1. From *unlabeled data*, learn a good features $T: \mathbb{R}^d \rightarrow \mathbb{R}^m$.
 2. To *labeled data*, apply transformation $T: \mathbb{R}^d \rightarrow \mathbb{R}^m$.
$$(T(x^{(1)}), y^{(1)}), \dots, (T(x^{(n)}), y^{(n)})$$
 3. Perform classification/regression on transformed data.



K-NEAREST NEIGHBORS

Main Idea.

- Find a few users b_1, \dots, b_k (neighbors) that are similar to user a
- Use information from users b_1, \dots, b_k to predict ratings of user a



K-means vs K-nearest neighbours

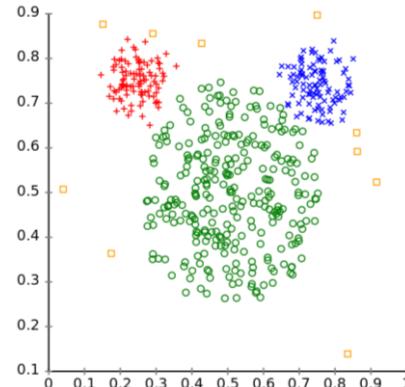
- K-means clustering is an unsupervised learning method, whereas KNN is a supervised learning technique.
- The 'K' in K-means refers to the number of clusters the algorithm is trying to learn from the data. The 'K' in KNN is the number of nearest neighbours used to classify a test sample.
- K-means has a training phase whereas KNN does not.

WHAT IS CLUSTERING

Clustering Problem.

Input. Training data $\mathcal{S}_n = \{x^{(1)}, x^{(2)}, \dots, x^{(n)}\}$, each $x^{(i)} \in \mathbb{R}^d$.
Integer k

Output. Clusters $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_k \subset \{1, 2, \dots, n\}$ such that
every data point is in one and only one cluster.



Some clusters
could be empty!

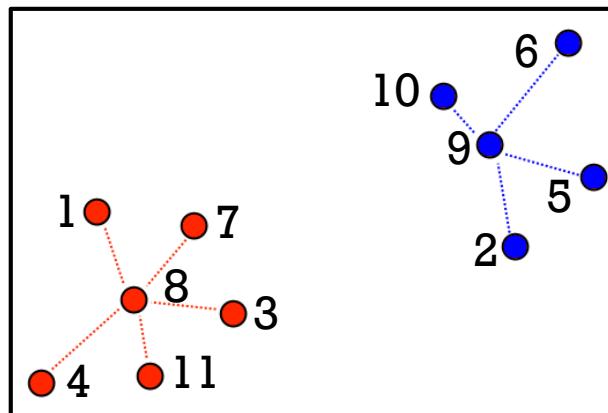


HOW TO SPECIFY A CLUSTER

- By listing all its elements

$$\mathcal{C}_1 = \{1,3,4,7,8,11\}$$

$$\mathcal{C}_2 = \{2,5,6,9,10\}$$



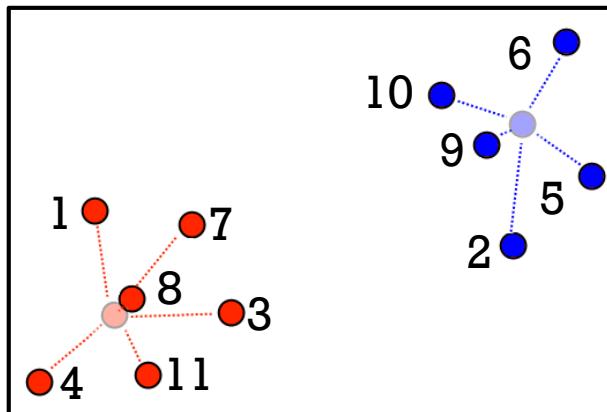
HOW TO SPECIFY A CLUSTER

Each point $x^{(i)}$ will be assigned the closest representative.

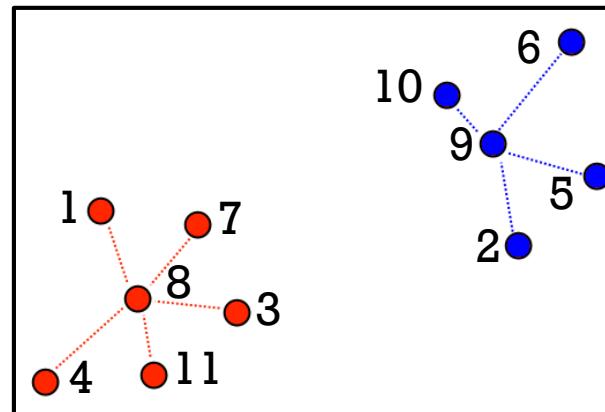
- Using a representative
 - a. A point in center of cluster (centroid)
 - b. A point in the training data (exemplar)

$$z^{(1)} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, z^{(2)} = \begin{pmatrix} 5 \\ 4 \end{pmatrix}$$

$$z^{(1)} = 8, z^{(2)} = 9$$



centroid

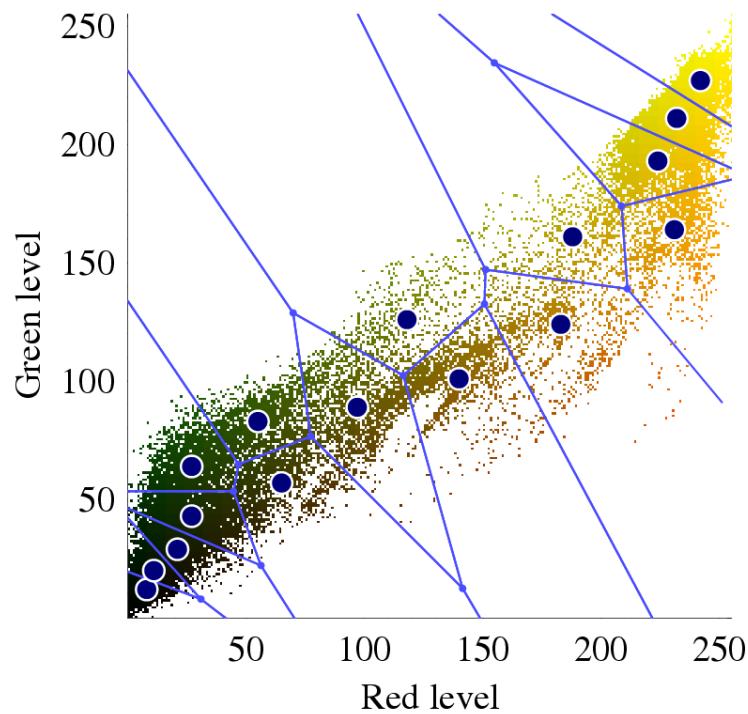


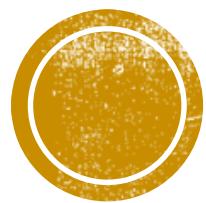
exemplar



VORONOI DIAGRAM

We can partition all the points in the space into regions, according to their closest representative.





TRAINING LOSS

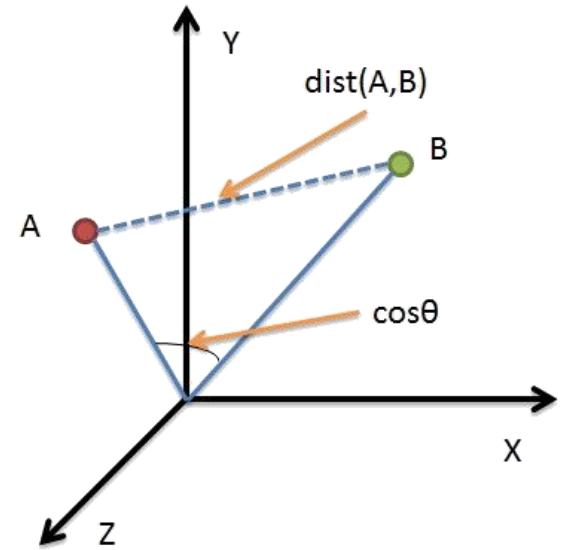


DISTANCE METRICS

(sometimes called *loss functions*)

A measure of how close two data points are.
Nearby points (i.e. distance is *small*) are
more likely they belong to the same cluster.

- Euclidean Distance $\text{dist}(x, y) = \|x - y\|^2$

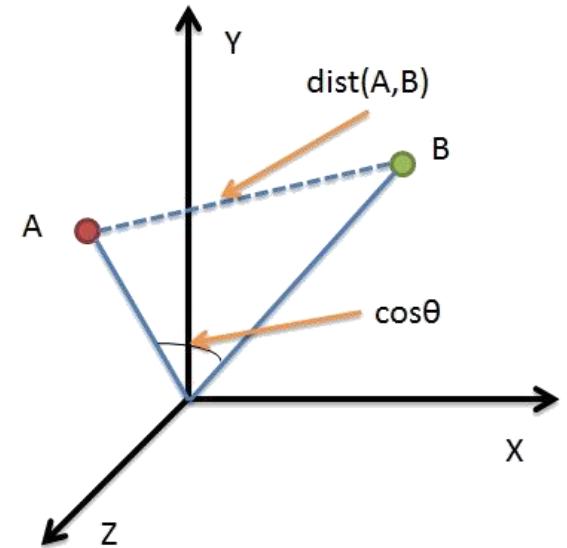


SIMILARITY FUNCTIONS

(sometimes called *kernels, correlation*)

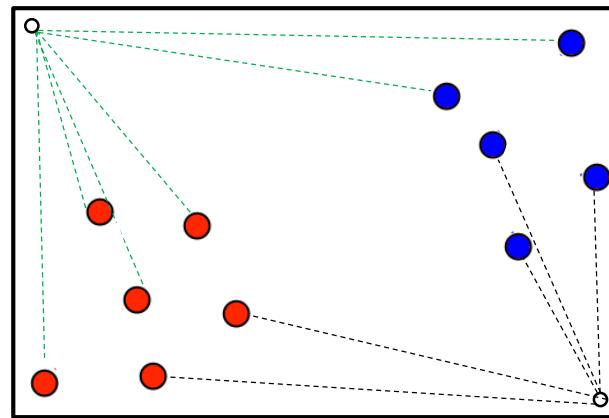
A measure of how alike two data points are.
Similar points (i.e. similarity is **large**) are
more likely they belong to the same cluster.

- Cosine Similarity $\cos(x, y) = \frac{x^T y}{\|x\| \|y\|}$



TRAINING LOSS

Sum of squared distances to closest representative.



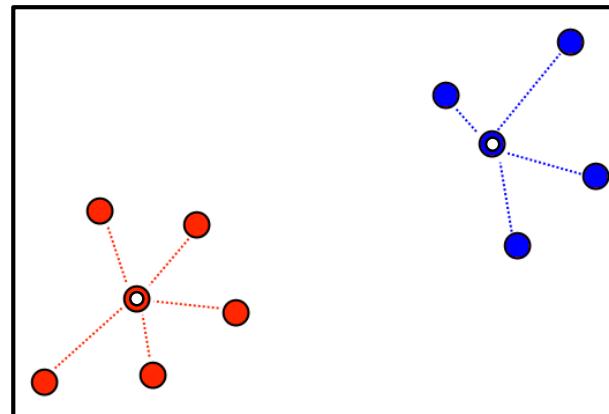
$$\text{loss} \approx 11 \times (1)^2 = 11$$

assume length of each
edge is about 1



TRAINING LOSS

Sum of squared distances to closest representative.



$$\text{loss} \approx 9 \times (0.1)^2 = 0.09$$

assume length of each
edge is about 0.1



TRAINING LOSS

Optimizing over representatives.

How do I use a
similarity function
instead?

$$\mathcal{L}_{n,k}(z^{(1)}, \dots, z^{(k)}; \mathcal{S}_n) = \sum_{i=1}^n \min_{1 \leq j \leq k} \|x^{(i)} - z^{(j)}\|^2.$$



TRAINING LOSS

Optimizing over clusters.

$$\mathcal{L}_{n,k}(\mathcal{C}_1, \dots, \mathcal{C}_n; \mathcal{S}_n) = \sum_{j=1}^n \sum_{i \in \mathcal{C}_j} \left\| x^{(i)} - \frac{1}{|\mathcal{C}_j|} \sum_{i' \in \mathcal{C}_j} x^{(i')} \right\|^2.$$



TRAINING LOSS

Optimizing both clusters and representatives.

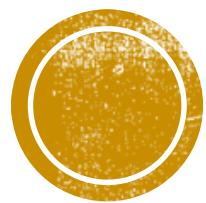
Instead of the distance metric, you can use the *negative similarity* function.

$$\mathcal{L}_{n,k}(\mathcal{C}_1, \dots, \mathcal{C}_k, z^{(1)}, \dots, z^{(k)}; \mathcal{S}_n) = \sum_{j=1}^k \sum_{i \in \mathcal{C}_j} \|x^{(i)} - z^{(j)}\|^2$$

These clusters need not consist of points closest to the representatives.

These representatives need not be the centroids of the clusters.

5 min break



K-MEANS



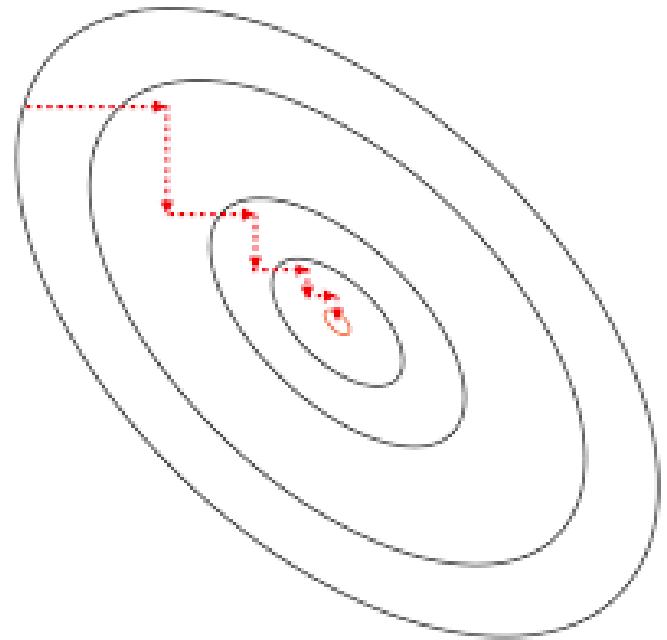
OPTIMIZATION ALGORITHM

Goal. Minimize $\mathcal{L}(x, y)$.

Coordinate Descent (Gradient).

Repeat until convergence:

1. Move in direction of $\partial\mathcal{L}/\partial x$.
2. Move in direction of $\partial\mathcal{L}/\partial y$.



Coordinate Descent (Optimization).

Repeat until convergence:

1. Find optimal x while holding y constant.
2. Find optimal y while holding x constant.



OPTIMIZATION ALGORITHM

Coordinate Descent (Optimization)

Repeat until convergence:

- Find best clusters given centroids
- Find best centroid given clusters

$$\mathcal{L}_{n,k}(\mathcal{C}_1, \dots, \mathcal{C}_k, z^{(1)}, \dots, z^{(k)}; \mathcal{S}_n) = \sum_{j=1}^k \sum_{i \in \mathcal{C}_j} \|x^{(i)} - z^{(j)}\|^2$$



OPTIMIZATION ALGORITHM

1. Initialize centroids $z^{(1)}, \dots, z^{(k)}$ from the data.
2. Repeat until no further change in training loss:

- a. For each $j \in \{1, \dots, k\}$,

$$\mathcal{C}_j = \{ i \text{ such that } x^{(i)} \text{ is closest to } z^{(j)} \}.$$

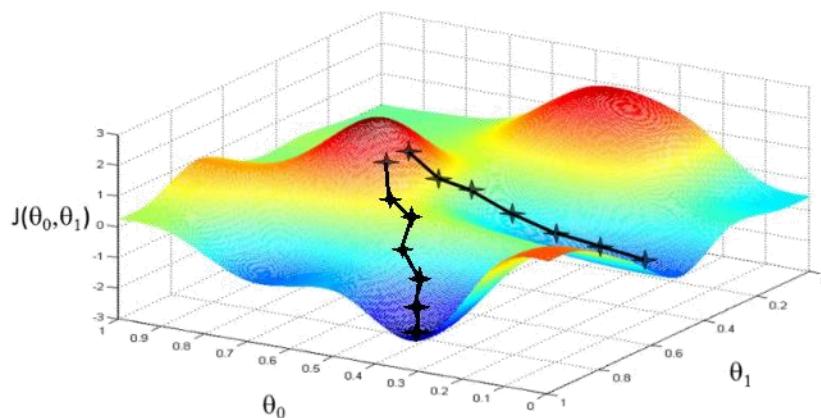
- b. For each $j \in \{1, \dots, k\}$,

$$z^{(j)} = \frac{1}{|\mathcal{C}_j|} \sum_{i \in \mathcal{C}_j} x^{(i)} \text{ (cluster mean)}$$



CONVERGENCE

- Training loss always decreases in each step (coordinate descent).
- Converges to local minimum, not necessarily global minimum.



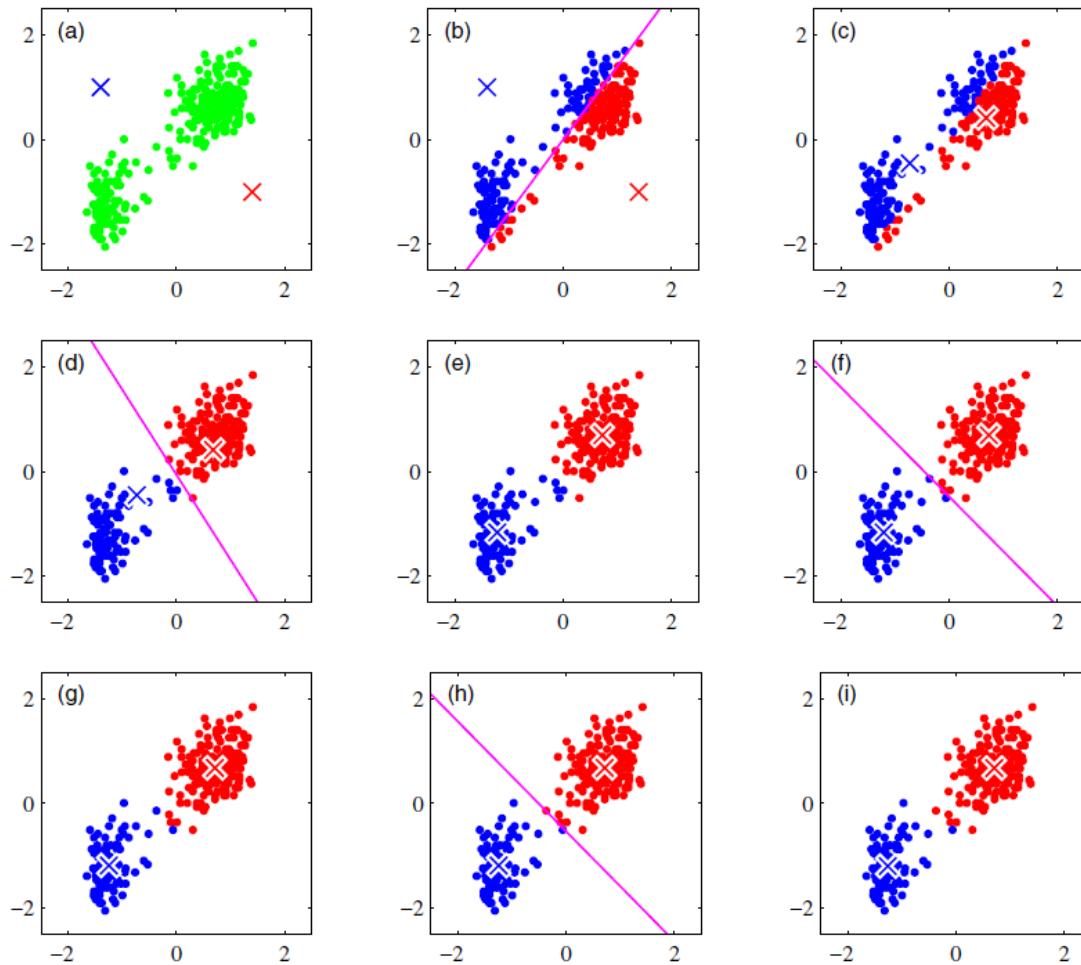
Challenge.

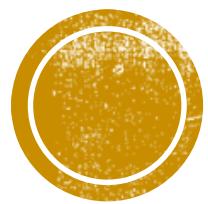
Why does the algorithm terminate in a finite number of steps?

* not in syllabus

Repeat algorithm over many initial points, and pick the configuration with the smallest training loss.

Illustration





DISCUSSION



INITIALIZATION

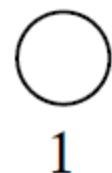
Optimization

- Empty clusters
 - Pick data points to initialize clusters
- Bad local minima
 - Initialize many times and pick solution with smallest training loss
 - Pick good starting positions



INITIALIZATION

Optimization



Starting position of centroids



Final position of centroids

Problem.

How to choose good starting positions?

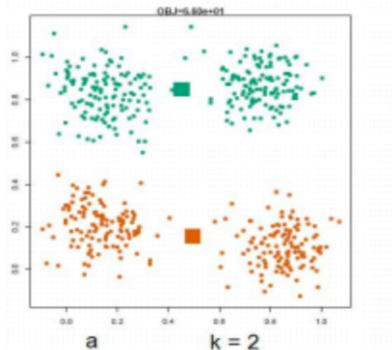
Solution.

Place them far apart with high probability.

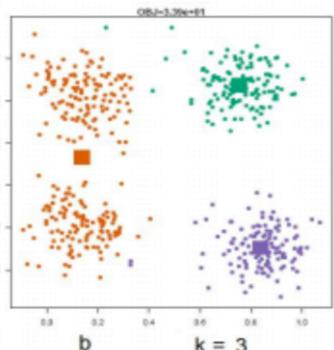


NUMBER OF CLUSTERS

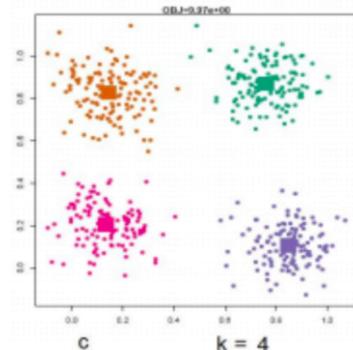
Generalization



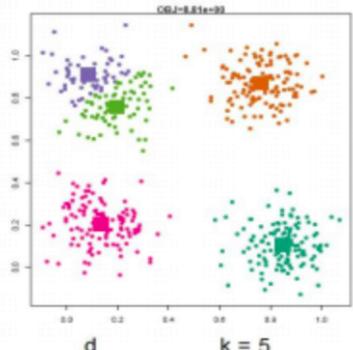
a $k = 2$



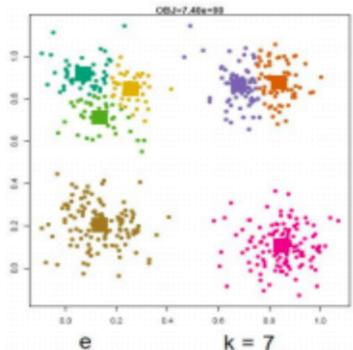
b $k = 3$



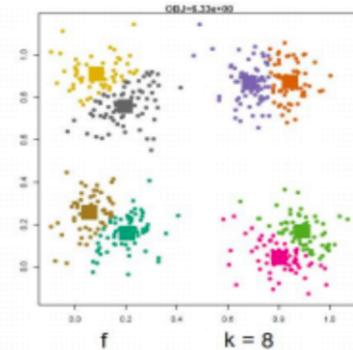
c $k = 4$



d $k = 5$



e $k = 7$



f $k = 8$



NUMBER OF CLUSTERS

Generalization

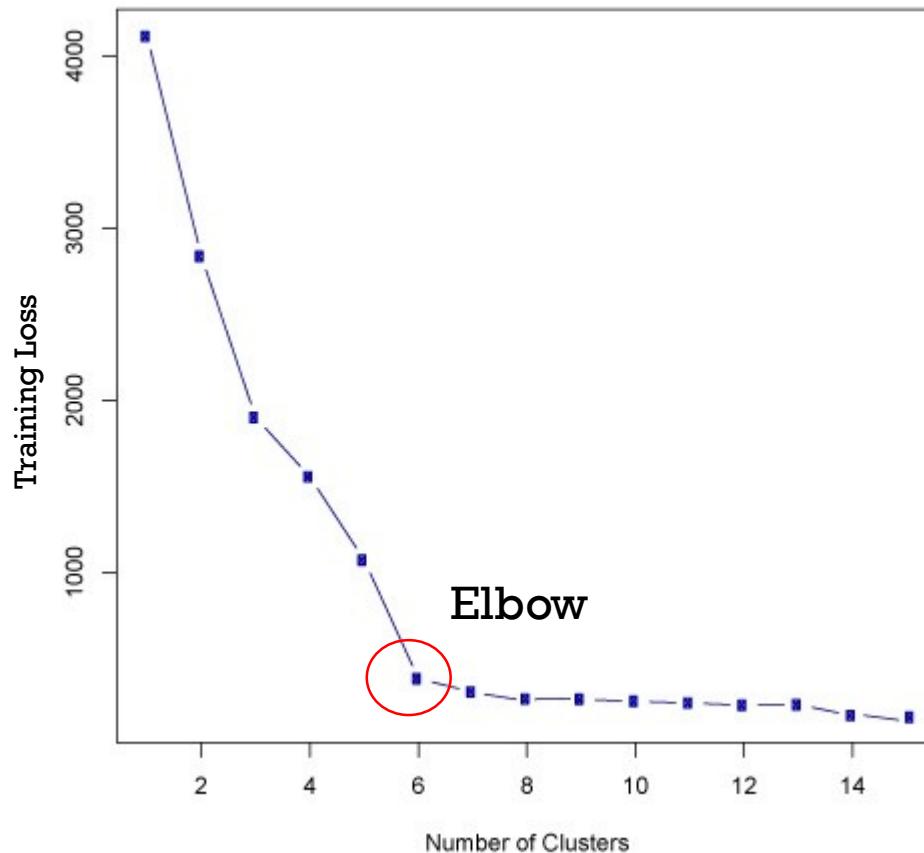
How do we choose k , the optimal number of clusters?

- Elbow method
 - Training Loss
 - Validation Loss
- Semi-supervised learning
 - Accuracy in supervised task



ELBOW METHOD

Generalization

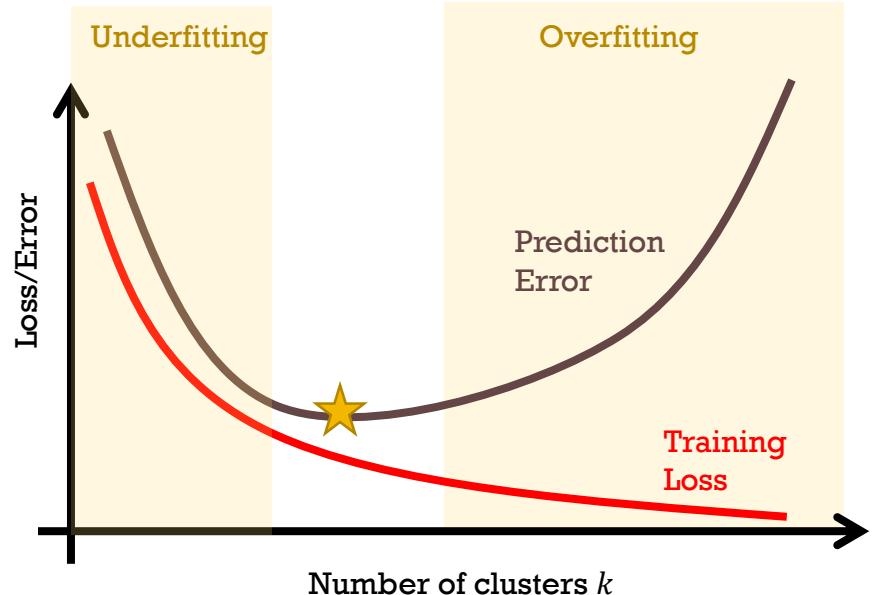


SEMI-SUPERVISED LEARNING

Generalization

Supervised task with small *labeled* data S'

- For each number of clusters k ,
 - Perform k -means on *unlabeled* data.
 - Transform S' using learned clusters e.g. compute distance to each centroid.
 - Use new features for supervised task, and compute prediction error.
- Pick k with smallest prediction error.



K-MEDROIDS

Use exemplars
instead of centroids.

e.g. Google News.

Repeat until convergence:

- Find best clusters given exemplars
- Find best exemplars given clusters

 People Are Drilling Headphone Jacks Into the iPhone 7
Fortune - 1 hour ago
He then takes the bit to the iPhone 7 and drills a hole into the device. ... Instead, Apple shipped iPhone 7 units with an adapter that lets users ...
[iPhone 7 review: Not Apple's best](#)
Expert Reviews - 2 hours ago
[Please don't drill a headphone jack into your iPhone 7](#)
BGR - 2 hours ago
[Apple iPhone 7 Users: Please DO NOT Drill a 3.5mm Hole on it to ...](#)
News18 - 7 hours ago
Video claiming drilling into iPhone 7 will reveal hidden headphone ...
Highly Cited - The Guardian - 1 hour ago
[Clueless iPhone 7 owners tricked into DRILLING hole in their ...](#)
Highly Cited - The Sun - 24 Sep 2016


Expert Reviews BGR The Guardian News18 International ... Herald Sun

[View all](#)

 Pegatron CEO slams analysts, 'cautiously optimistic' about Apple ...
AppleInsider (press release) (blog) - 3 hours ago
The CEO of Apple's manufacturing partner Pegatron notes that the iPhone 7 is exceeding estimates on the strength of the phone alone, and ...
[Google Nexus 2016' Specs: Solution to Apple iPhone 7 ...](#)
University Herald - 3 hours ago
[Apple Supplier Pegatron Hints of Higher iPhone 7 Demand while ...](#)
Patently Apple - 2 hours ago
[iPhone 7 vs Samsung Galaxy S7: Which is the best smartphone to ...](#)
Alphr - 5 hours ago
[Samsung Galaxy Note 7 Explosions Boost iPhone 7 Sales, Top ...](#)
Softpedia News - 8 hours ago


University He... Patently Apple Alphr Softpedia News Expert Reviews

[View all](#)

Information theory

Entropy

- Consider a random variable X on the set $\{a, b, c, d\}$, with probabilities $P(X = a) = p_a$, $P(X = b) = p_b$, ...
- What is the optimal number of bits to encode the possible values of X ?

- Since there are 4 possibilities, we can use 00 for a , 01 for b , 10 for c and 11 for d ; i.e. 2 bits.
- If $p_a = p_b = p_c = p_d = \frac{1}{4}$, then on average we expect to use 2 bits to transmit a message containing just the value of X .
- Should we adopt the same encoding scheme if if $p_a = \frac{1}{2}, p_b = \frac{1}{4}, p_c = \frac{1}{8} = p_d$?

- Intuitively we should use fewer bits to encode the more frequently occurring values, and more bits to encode the less frequently occurring ones.
- Eg., we can use 0 for a , 10 for b , 110 for c and 111 for d . Note that we cannot use shorter codes for b , c or d because we need to be able to unambiguously parse a concatenation of the strings, eg. 1110110 decodes uniquely into dac .
- With this encoding scheme, on average we use

$$\left(\frac{1}{2} \times 1\right) + \left(\frac{1}{4} \times 2\right) + \left(\frac{1}{8} \times 3\right) + \left(\frac{1}{8} \times 3\right) = 1.75$$

bits.

Definition

The entropy, $H(X)$ of a discrete random variable is given by

$$H(X) = - \sum_i p_i \log p_i,$$

where we adopt the convention that $0 \log 0 = 0$.

If we use base 2 for the logarithm, the units of entropy are given in *bits*; if the natural logarithm is used, the units are called *nats*.

- Thus, when $p_a = \frac{1}{2}, p_b = \frac{1}{4}, p_c = p_d = \frac{1}{8}$, then

$$H(X) = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{4} \log_2 \frac{1}{4} - \frac{1}{8} \log_2 \frac{1}{8} - \frac{1}{8} \log_2 \frac{1}{8} = 1.75$$

bits, which is the same as the average number of bits we computed earlier with our encoding scheme.

- In fact, Shannon's source coding theorem (1948) (or noiseless coding theorem) tells us that we cannot do better; i.e. we cannot find a lossless encoding scheme that uses on average fewer bits than the entropy of X ; i.e. entropy gives us a lower bound.

- Recall for HW 2 that entropy is maximized when X is a uniform distribution; for n classes, we need

$$\log_2 n$$

bits on average to transmit X , and this is the most bandwidth required amongst all possible distributions of X .

- In contrast, if we know that $p_i = 1$ for some i , then $H(X) = 0$, and we do not need any bandwidth for transmission since we already know the outcome!

Cross entropy

Definition

The cross entropy of two discrete distributions p and q , such that $q_i = 0 \implies p_i = 0$, is given by

$$H(p, q) = - \sum_i p_i \log q_i.$$

If $q_i = 0$ for some i but $p_i > 0$, then $H(p, q) = \infty$.

We can also write $H(X, Y)$ instead when we have two random variables X and Y with distributions p and q respectively.

- We know that $H(p, q) \geq H(p, p)$ for all q and equality occurs when $q = p$.
- Recall that cross entropy loss is used in logistic/softmax regression, where p denotes the target distribution (typically $p_i = 1$ for some i and 0 otherwise; this is the one-hot encoding of $t = i$), and q is the prediction of the model.
- Thus cross entropy gives a measure of how dissimilar q is from p .
- It is not symmetric; i.e. $H(p, q) \neq H(q, p)$ in general.

Kullback-Leibler (KL) divergence (or relative entropy)

Definition

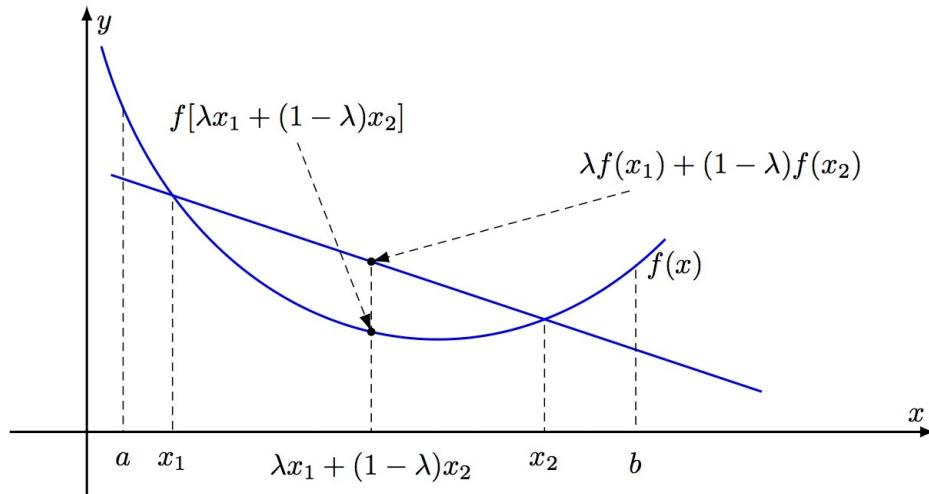
The KL divergence of two discrete distributions p and q such that $q_i = 0 \implies p_i = 0$, is given by

$$\begin{aligned} D_{KL}(p|q) &= H(p, q) - H(p, p) \\ &= \sum_i p_i \log \frac{p_i}{q_i}. \end{aligned}$$

If $q_i = 0$ for some i but $p_i > 0$, then $H(p, q) = \infty$.

- KL divergence measures the number of extra bits required to transmit X with distribution p , as compared to the optimal code, when we use the sub-optimal coding scheme associated with distribution q .
- As with cross entropy, it is not symmetric.
- We can use source coding theorem to infer that KL divergence is always non-negative, but there is a more direct proof using Jensen's inequality.

Convex functions



Definition

A function $\phi : (a, b) \rightarrow \mathbb{R}$ is convex if for all $x, y \in (a, b)$, $\phi(\lambda x + (1 - \lambda)y) \leq \lambda\phi(x) + (1 - \lambda)\phi(y)$ for all $\lambda \in (0, 1)$.

Proposition

If a function ϕ is convex, then it is continuous.

Not all convex functions are differentiable, eg. $\phi(x) = |x|$, but we have the following proposition.

Proposition

If a function ϕ has a non-negative second derivative on (a, b) , then it is convex.

Jensen's inequality

Theorem

Let $\phi(x)$ be a convex function. If μ is a probability measure, and $f(x)$ and $\phi(f(x))$ are integrable, then

$$\phi \left(\int f(x) d\mu(x) \right) \leq \int \phi(f(x)) d\mu(x).$$

Example

$$\begin{aligned} \text{Var}(X) \geq 0 &\iff \mathbb{E}[X^2] - (\mathbb{E}[X])^2 \geq 0 \\ &\iff \mathbb{E}[X^2] \geq (\mathbb{E}[X])^2 \end{aligned}$$

We know the second line is true by applying Jensen's inequality with $\phi(x) = x^2$ and $f(x) = x$.

Proposition

For any two distributions p and q , $D_{KL}(p|q) \geq 0$, and is equal to 0 when $p = q$.

Proof.

$$\begin{aligned} D_{KL}(p|q) &= \sum_i p_i \left(-\log \frac{q_i}{p_i} \right) \quad (\text{sum runs over all } i \text{ such that } p_i > 0) \\ &\geq -\log \sum_i p_i \left(\frac{q_i}{p_i} \right) \quad (\text{by Jensen's inequality}) \\ &= -\log \sum_i q_i \geq 0. \end{aligned}$$



EXPECTATION MAXIMIZATION

Lesson 1

Epitaph

moulee-rah = money

jujiminmee = kidnap

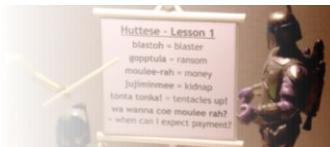
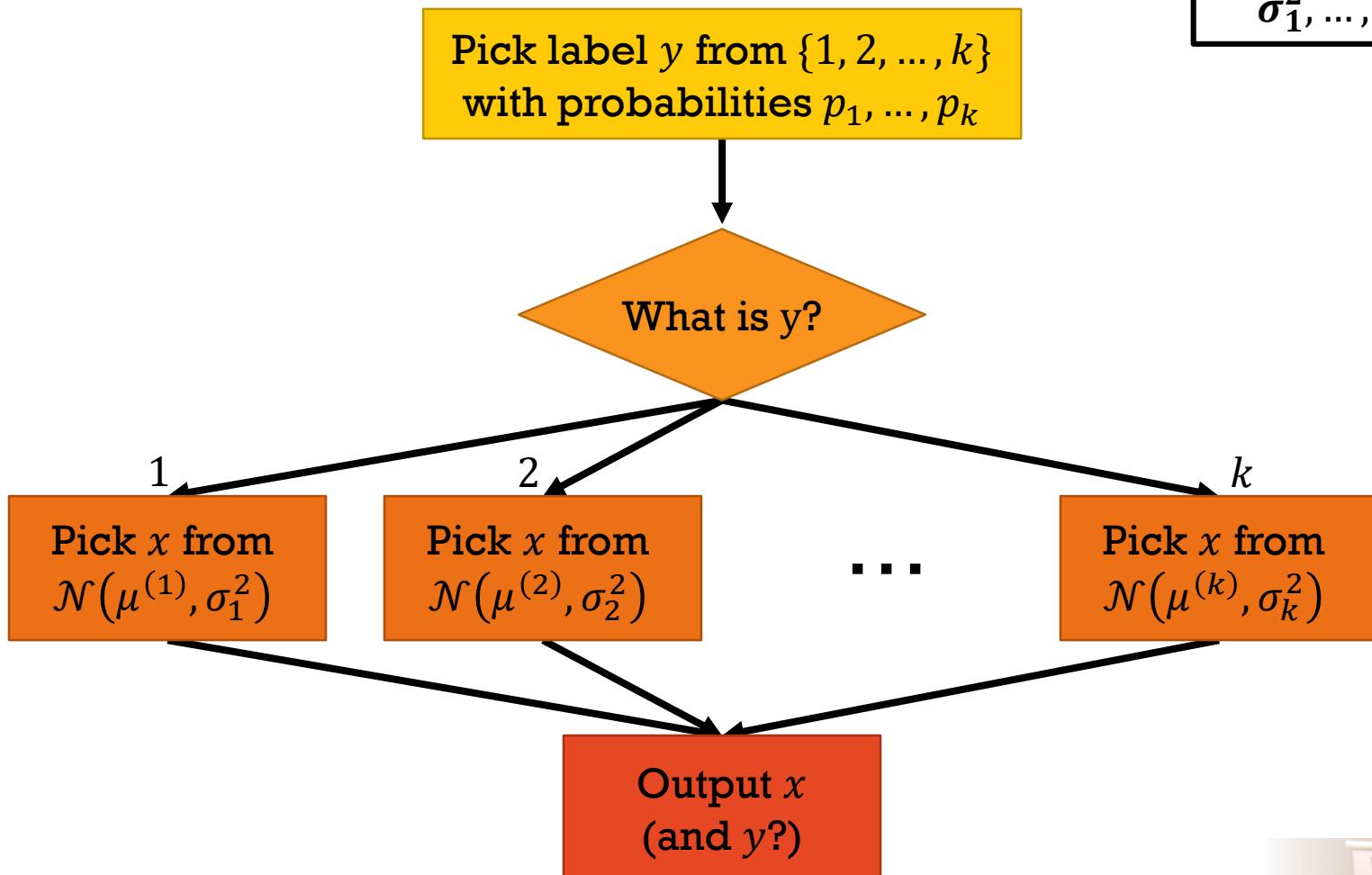
tonta tonka! = tentacles up!

wa wanna coe moulee rah?
= when can I expect payment?

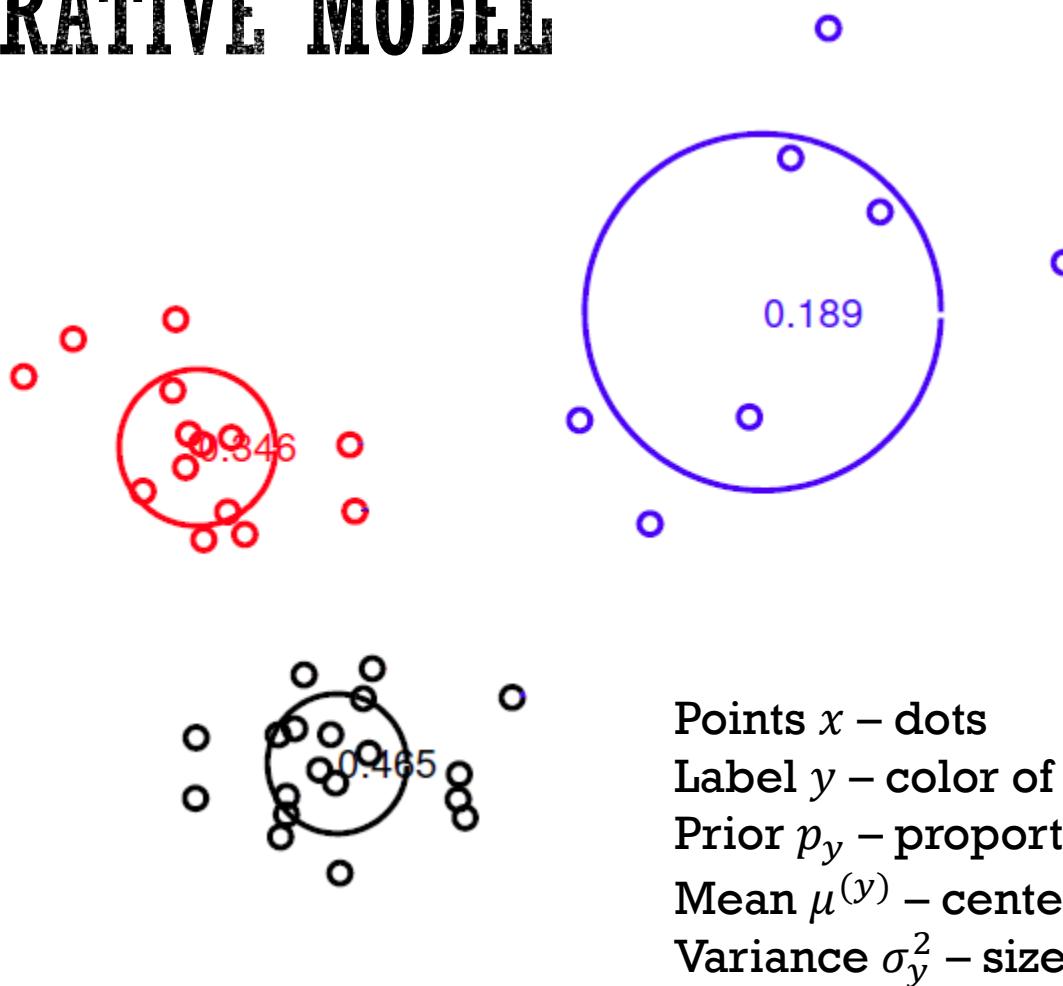


GENERATIVE MODEL

Model Parameters

$$p_1, \dots, p_k$$
$$\mu^{(1)}, \dots, \mu^{(k)}$$
$$\sigma_1^2, \dots, \sigma_k^2$$


GENERATIVE MODEL



Points x – dots

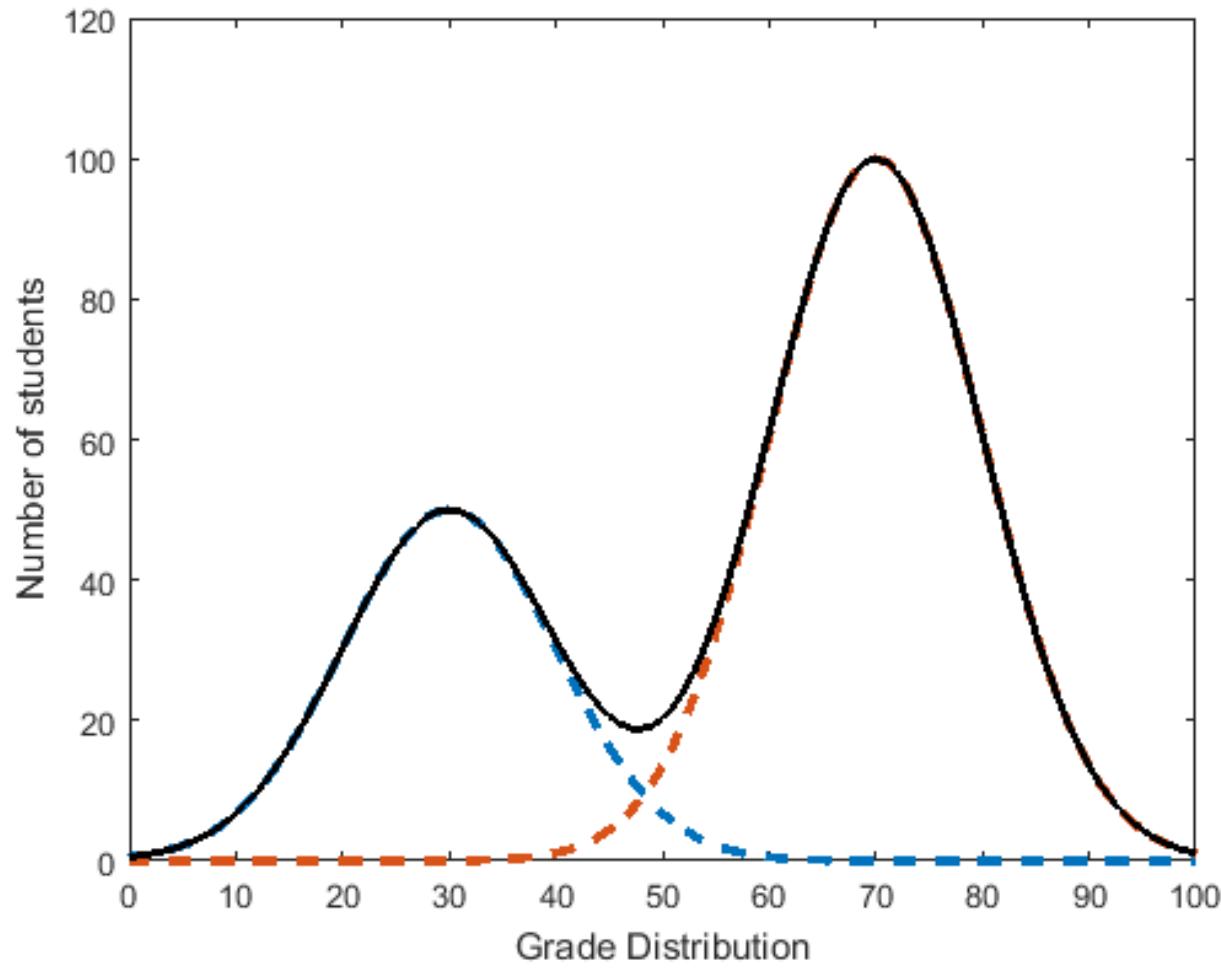
Label y – color of dots

Prior p_y – proportion of dots

Mean $\mu^{(y)}$ – center of circle

Variance σ_y^2 – size of circle

GENERATIVE MODEL



HutteSe - Lesson 1
blastoh + blaster
gooptula + ramon
moulee-rahy = money
Julimimeme = kidnap
tonka tonka! = tentacles up!
wa wanna coe moulee rah?
- when can I expect payment?



Gaussian Mixture Models

Given training set $x^{(1)}, \dots, x^{(N)}$, we wish to model the data by specifying the joint distribution of X with a latent variable Z :

$$p(x, z) = p(x|z)p(z).$$

Here $Z \sim \text{Multinomial}(\pi)$, i.e.

$$P(Z = j) = \pi_j, \quad j = 1, \dots, m,$$

and $\sum_{j=1}^m \pi_j = 1$, and we have

$$X | \{Z = j\} \sim \mathcal{N}(\mu_j, \Sigma_j).$$

- This means that the log-likelihood of the data is given by

$$\begin{aligned}
 \ell(\pi, \mu, \Sigma) &= \sum_{i=1}^N \log p(x^{(i)}) = \sum_{i=1}^N \log \sum_{j=1}^m p(x^{(i)} \mid Z=j) P(Z=j) \\
 &= \sum_{i=1}^N \log \sum_{j=1}^m \pi_j \left(C_j \exp^{-\frac{1}{2} \langle x^{(i)} - \mu_j, \Sigma_j^{-1} (x^{(i)} - \mu_j) \rangle} \right)
 \end{aligned}$$

- Unfortunately, there is no closed-form solution to optimizing this log-likelihood. Let's see why by examining the conditions we get when we try to optimize $\ell(\pi, \mu, \Sigma)$.

- Differentiating with respect to μ_k and setting to zero, we get

$$\sum_{i=1}^N \frac{\pi_k \left(C_j \exp^{-\frac{1}{2} \langle x^{(i)} - \mu_j, \Sigma_j^{-1} (x^{(i)} - \mu_j) \rangle} \right) \Sigma^{-1} (x^{(i)} - \mu_k)}{\sum_{j=1}^m \pi_j \left(C_j \exp^{-\frac{1}{2} \langle x^{(i)} - \mu_j, \Sigma_j^{-1} (x^{(i)} - \mu_j) \rangle} \right)} = 0$$

- We will denote

$$\gamma \left(z_k^{(i)} \right) := \frac{\pi_k \left(C_j \exp^{-\frac{1}{2} \langle x^{(i)} - \mu_k, \Sigma_k^{-1} (x^{(i)} - \mu_k) \rangle} \right)}{\sum_{j=1}^m \pi_j \left(C_j \exp^{-\frac{1}{2} \langle x^{(i)} - \mu_j, \Sigma_j^{-1} (x^{(i)} - \mu_j) \rangle} \right)},$$

and rearranging, we get

$$\mu_k = \frac{\sum_{i=1}^N x^{(i)} \gamma \left(z_k^{(i)} \right)}{\sum_{i=1}^N \gamma \left(z_k^{(i)} \right)}. \quad (1)$$

- Similarly, differentiating with respect to Σ_k^{-1} and setting to zero, we get

$$\Sigma_k^{-1} = \frac{\sum_{i=1}^N \gamma(z_k^{(i)}) \|x^{(i)} - \mu_k\|^2}{\sum_{i=1}^N \gamma(z_k^{(i)})}. \quad (2)$$

- Since π satisfies the constraint $\sum_{j=1}^m \pi_j = 1$, we'll have to use Lagrange multipliers to optimize with respect to π . We have

$$\frac{d}{d\pi_k} \left[\sum_{i=1}^N \log \sum_{j=1}^m \pi_j \left(C_j \exp^{-\frac{1}{2} \langle x^{(i)} - \mu_j, \Sigma_j^{-1} (x^{(i)} - \mu_j) \rangle} \right) - \lambda \left(\sum_{j=1}^m \pi_j - 1 \right) \right] = 0,$$

- which implies that

$$\sum_{i=1}^N \frac{\left(C_k \exp^{-\frac{1}{2} \langle x^{(i)} - \mu_k, \Sigma_k^{-1} (x^{(i)} - \mu_k) \rangle} \right)}{\sum_{j=1}^m \pi_j \left(C_j \exp^{-\frac{1}{2} \langle x^{(i)} - \mu_j, \Sigma_j^{-1} (x^{(i)} - \mu_j) \rangle} \right)} = \lambda.$$

- Multiplying by π_k on both sides and summing over k , we get $\lambda = N$, which gives

$$\pi_k = \frac{1}{N} \sum_{i=1}^N \gamma \left(z_k^{(i)} \right). \quad (3)$$

As suggested in the notation, $\gamma(z_k^{(i)})$ represents an important quantity related to the latent variable Z . Indeed, if we compute

$$P(Z = k | x^{(i)}) = \frac{p(x^{(i)} | Z = k) P(Z = k)}{p(x^{(i)})}$$

using Bayes' rule, we get

$$\begin{aligned} P(Z = k | x^{(i)}) &= \frac{\pi_k \left(C_j \exp^{-\frac{1}{2} \langle x^{(i)} - \mu_k, \Sigma_k^{-1} (x^{(i)} - \mu_k) \rangle} \right)}{\sum_{j=1}^m \pi_j \left(C_j \exp^{-\frac{1}{2} \langle x^{(i)} - \mu_j, \Sigma_j^{-1} (x^{(i)} - \mu_j) \rangle} \right)} \\ &= \gamma(z_k^{(i)}). \end{aligned}$$

As $\gamma(z_k^{(i)})$ contains the parameters μ , Σ and π in a complex way, (1), (2) and (3) cannot be solved in closed-form. However, it suggests the following two-step algorithm:

- E-step: Set

$$\gamma_{t+1}(z_k^{(i)}) = P_{\mu(t), \Sigma(t), \pi(t)}(Z = k \mid x^{(i)})$$

- M-step: Set

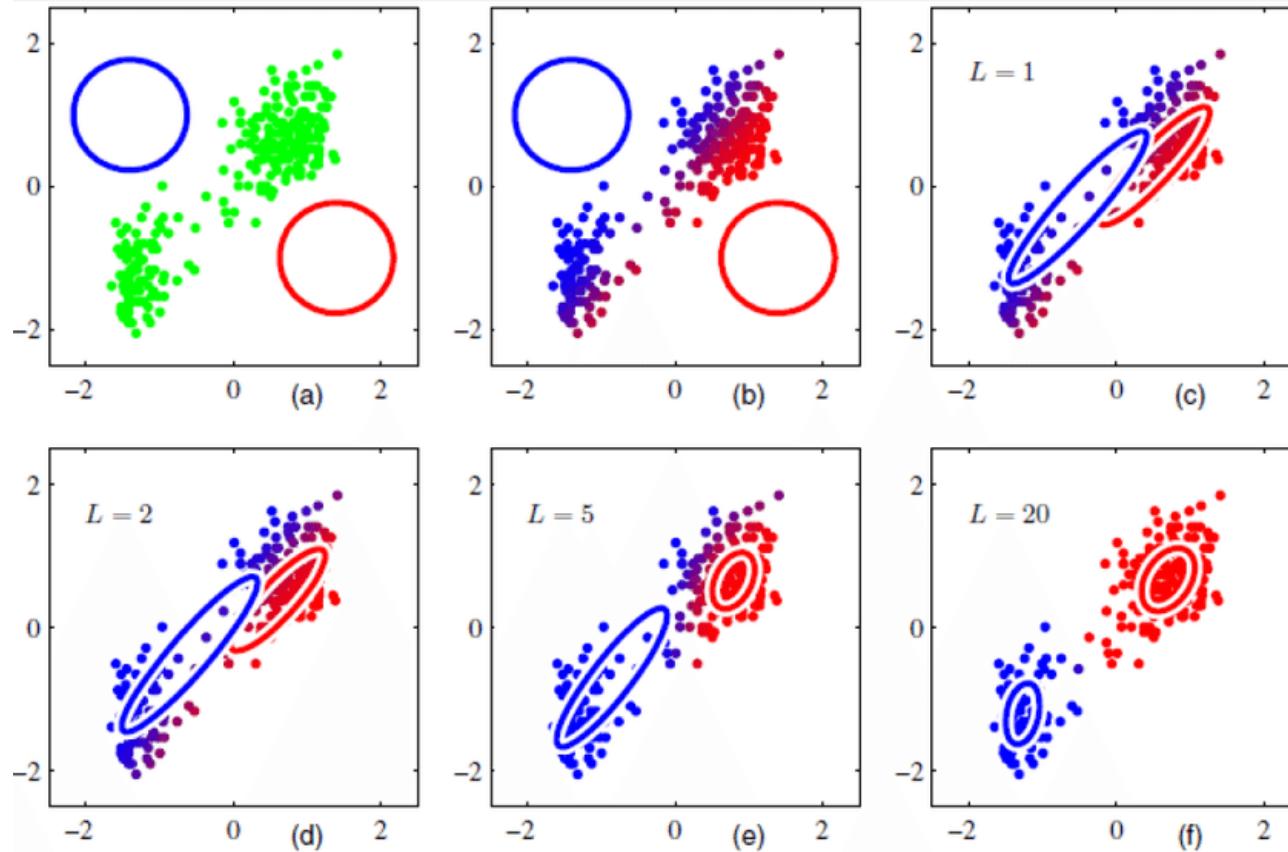
$$\pi_k(t+1) = \frac{1}{N} \sum_{i=1}^N \gamma_{t+1}(z_k^{(i)})$$

$$\mu_k(t+1) = \frac{\sum_{i=1}^N x^{(i)} \gamma_{t+1}(z_k^{(i)})}{\sum_{i=1}^N \gamma_{t+1}(z_k^{(i)})}$$

$$\Sigma_k^{-1}(t+1) = \frac{\sum_{i=1}^N \gamma_{t+1}(z_k^{(i)}) \|x^{(i)} - \mu_k(t)\|^2}{\sum_{i=1}^N \gamma_{t+1}(z_k^{(i)})}$$

- Repeat until convergence.

EM on Old Faithful dataset



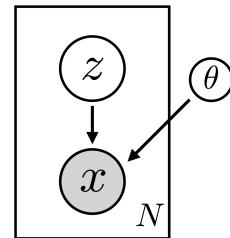
COMPARISON WITH K-MEANS

- Like k-means, EM clustering may get stuck in local minima.
- Unlike k-means, the local minima are more favorable because soft labels allow points to move between clusters slowly.



EM algorithm

- Given a training set $\{x^1, \dots, x^{(N)}\}$ which we hypothesize to be generated from latent variables z



we wish to maximize the log-likelihood

$$\begin{aligned} l_\theta(\mathbf{x}) &= \sum_{i=1}^N \log p_\theta(x_i) \\ &= \sum_{i=1}^N \log \int p_\theta(x_i, z) dz \end{aligned}$$

EM algorithm

- E-step: Construct a lower bound
- M-step: Optimize the lower bound

E-step

Construct a lower bound:

Given any distribution $q(z)$, we have

$$\begin{aligned} \sum_{i=1}^N \log \int p_\theta(x^{(i)}, z) dz &= \sum_{i=1}^N \log \int q(z) \frac{p_\theta(x^{(i)}, z)}{q(z)} dz \\ &= \sum_{i=1}^N \log \mathbb{E}_{q(z)} \left[\frac{p_\theta(x^{(i)}, z)}{q(z)} \right] \\ &\geq \sum_{i=1}^N \mathbb{E}_{q(z)} \left[\log \frac{p_\theta(x^{(i)}, z)}{q(z)} \right] = \sum_{i=1}^N \int q(z) \log \frac{p_\theta(x^{(i)}, z)}{q(z)} dz, \end{aligned}$$

where the last line follows by Jensen's inequality.

- The lower bound $\sum_{i=1}^N \int q(z) \log \frac{p_\theta(x^{(i)}, z)}{q(z)} dz$ holds for all distributions $q(z)$, but which one is the best?
- We shall fix i and look at

$$\log \int p_\theta(x^{(i)}, z) dz - \int q(z) \log \frac{p_\theta(x^{(i)}, z)}{q(z)} dz$$

for a single data-point, then sum over all the data-points later.

- Since $p_\theta(z|x^{(i)}) = \frac{p_\theta(x^{(i)}, z)}{p_\theta(x^{(i)})}$, this implies that

$$\begin{aligned}
\log p_\theta(x^{(i)}) &= \log p_\theta(x^{(i)}, z) - \log p_\theta(z|x^{(i)}) \\
&= \log p_\theta(x^{(i)}, z) - \log q(z) + \log q(z) - \log p_\theta(z|x^{(i)}) \\
&= \log \frac{p_\theta(x^{(i)}, z)}{q(z)} + \log \frac{q(z)}{p_\theta(z|x^{(i)})},
\end{aligned}$$

which means that

$$\log p_\theta(x^{(i)}) - \log \frac{p_\theta(x^{(i)}, z)}{q(z)} = \log \frac{q(z)}{p_\theta(z|x^{(i)})}.$$

- Integrating both sides with respect to $q(z) dz$, we have

$$\int \log p_{\theta}(x^{(i)}) q(z) dz - \int q(z) \log \frac{p_{\theta}(x^{(i)}, z)}{q(z)} dz = \int q(z) \log \frac{q(z)}{p_{\theta}(z|x^{(i)})} dz,$$

which gives us the formula

$$\log_{\theta} p(x^{(i)}) - \int q(z) \log \frac{p_{\theta}(x^{(i)}, z)}{q(z)} dz = D_{KL} [q(z) | p_{\theta}(z|x^{(i)})].$$

- Recall that the KL-divergence is ≥ 0 , and equals 0 when $q(z) = p_{\theta}(z|x^{(i)})$.

EM algorithm

(i) E-step: Set

$$q_t(z) := p_{\theta_t} \left(z \mid x^{(i)} \right)$$

(ii) M-step: Set

$$\theta_{t+1} := \arg \max_{\theta} \sum_{i=1}^N \int q_t(z) \log \frac{p_{\theta}(x^{(i)}, z)}{q_t(z)} dz$$

(iii) Go back to step (i) until the increase in $\ell(\theta)$ falls below some predetermined threshold.

Monotone convergence theorem

Theorem

Let $\{a_n\}$ be an monotonically non-decreasing sequence; i.e. $a_{n+1} \geq a_n$ for all n . If $\{a_n\}$ is bounded above by some constant c , then the sequence converges.

Convergence

- Note that

$$\begin{aligned}\ell(\theta_{t+1}) &\geq \sum_{i=1}^N \int q_t(z) \log \frac{p_{\theta_{t+1}}(x^{(i)}, z)}{q_t(z)} dz \\ &\geq \sum_{i=1}^N \int q_t(z) \log \frac{p_{\theta_t}(x^{(i)}, z)}{q_t(z)} dz \\ &= \ell(\theta_t).\end{aligned}$$

- The first inequality follows from the definition of the lower bound, the second follows from the M-step, and the third equality is a result of the E-step which sets $D_{KL}[q(z) \mid p_{\theta_t}(z|x_i)]$ to 0.
- Thus, we get convergence from Monotone convergence theorem since we have a monotonically non-decreasing sequence which is bounded above by 0.

MODEL SELECTION

- By setting $p_{k+1} = 0$, we see that (mixture model with k clusters) contained in (mixture model with $k + 1$ clusters).
- Therefore, likelihood for (mixture model with $k + 1$ clusters) is greater or equal to that of (mixture model with k clusters).
- How to choose the right k and prevent over-/under-fitting?



VALIDATION VS CROSS-VALIDATION

Method 1 (Simulation)

Estimate testing error using simple validation or cross-validation.

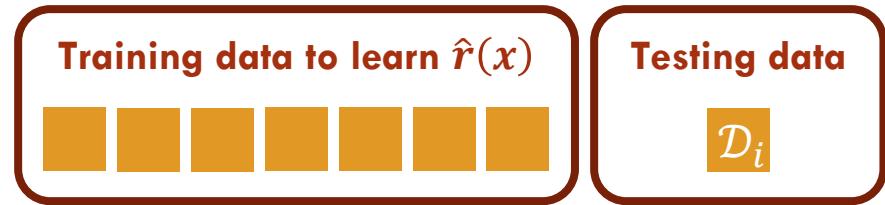
testing error

- $\hat{R}(\mathcal{D})$



k-fold cross-validation.

- $\hat{R}_{\text{CV}} = \frac{1}{m} \sum_{i=1}^m \hat{R}(\mathcal{D}_i)$



BAYESIAN INFORMATION CRITERION

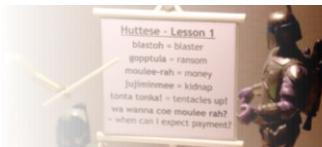
Method 2 (Marginal Likelihood)

Maximize the **marginal likelihood integral**. But computing this integral is tedious, so we approximate it using the BIC.

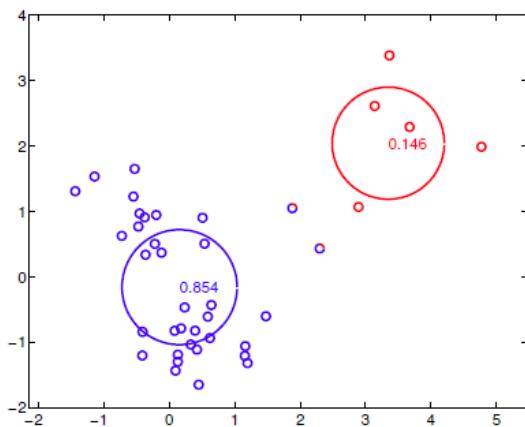
$$\text{BIC}(\theta) = \mathcal{L}_n(\theta) - \frac{\# \text{ of free params}}{2} \log n$$

For Gaussian mixtures, we have $k(d + 2) - 1$ free parameters.

$$\text{BIC}(\theta) = \mathcal{L}_n(\theta) - \frac{k(d+2)-1}{2} \log n$$

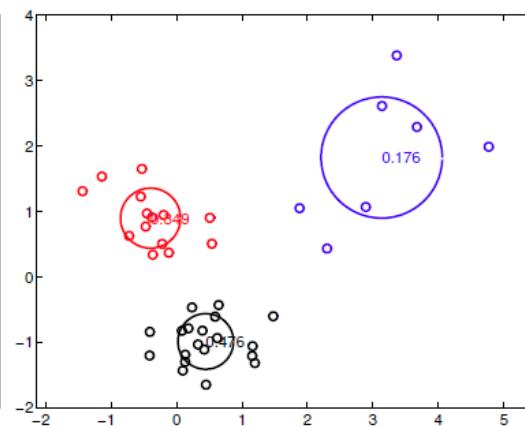


BAYESIAN INFORMATION CRITERION



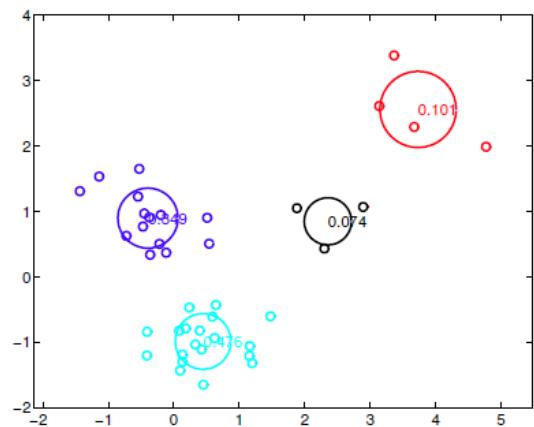
$$l(D; \hat{\theta}) = -118.25$$

$$BIC(D; \hat{\theta}) = -131.16$$



$$l(D; \hat{\theta}) = -98.64$$

$$BIC(D; \hat{\theta}) = -118.93$$



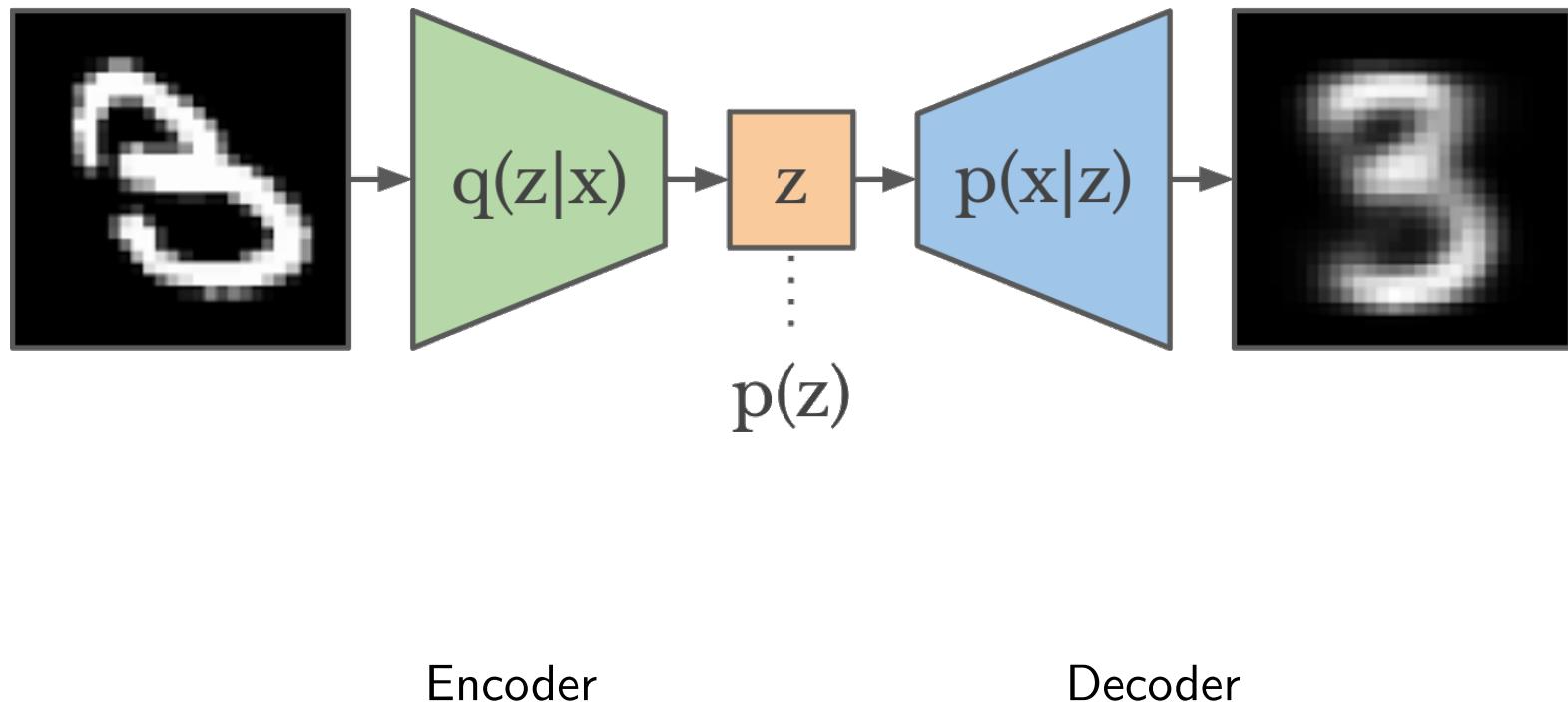
$$l(D; \hat{\theta}) = -94.11$$

$$BIC(D; \hat{\theta}) = -121.78$$

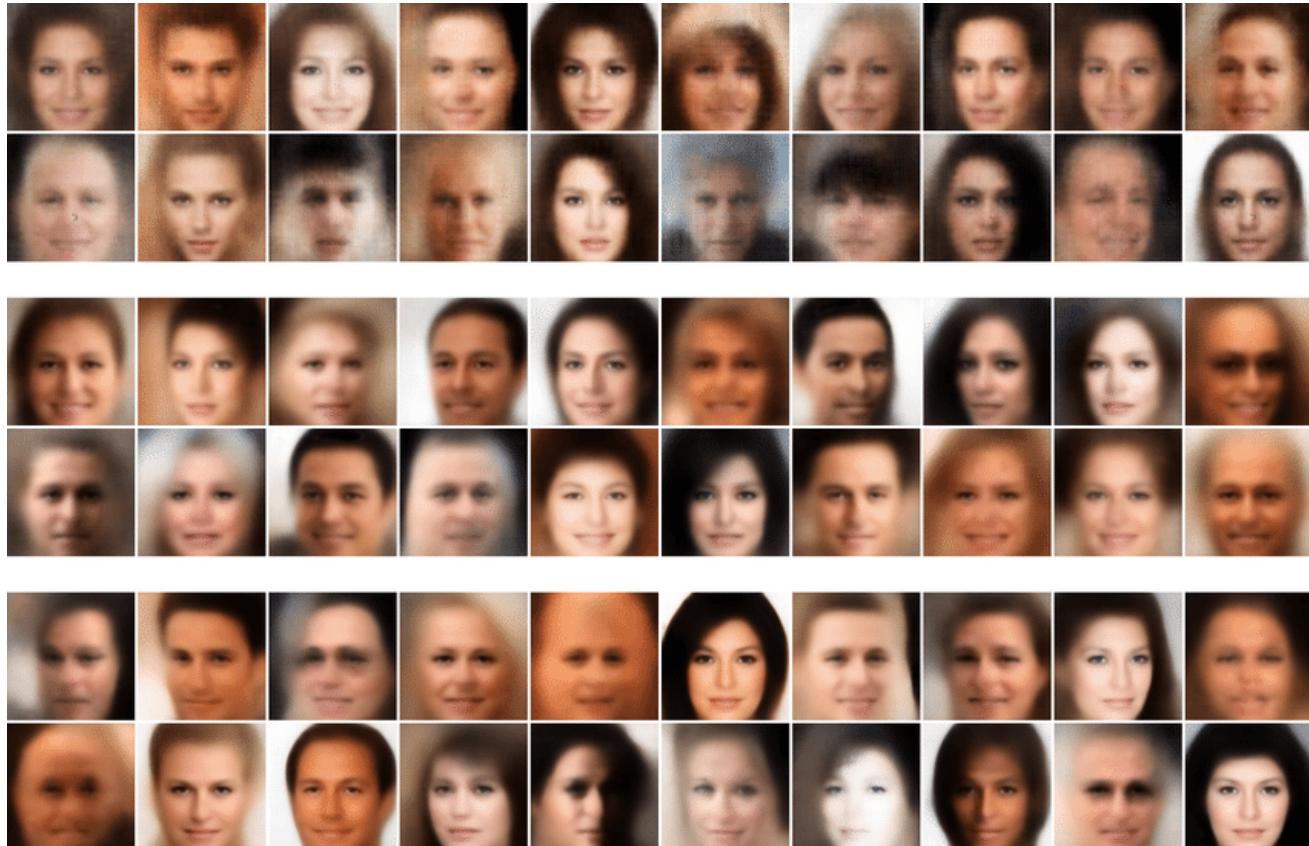


Variational auto-encoders (VAEs)

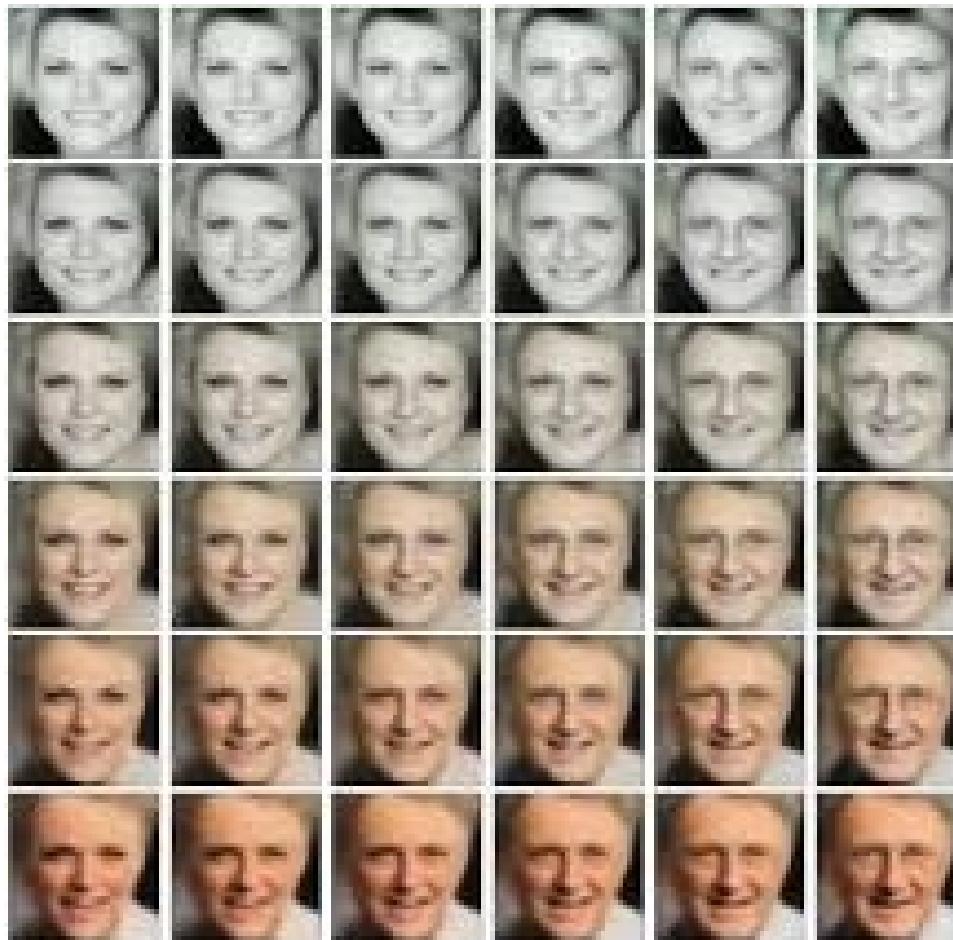
Variational autoencoder



Computer generated faces

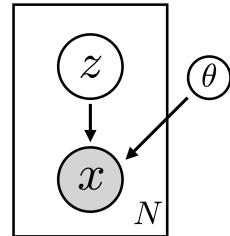


Interpolation between sampled points



EM algorithm in general

- Given a training set $\{x^1, \dots, x^{(N)}\}$ which we hypothesize to be generated from latent variables z



we wish to maximize the log-likelihood

$$\begin{aligned} l_\theta(\mathbf{x}) &= \sum_{i=1}^N \log p_\theta(x^{(i)}) \\ &= \sum_{i=1}^N \log \int p_\theta(x^{(i)}, z) dz \end{aligned}$$

- The expectation-maximization (EM) algorithm in general is a technique for finding maximum likelihood solutions for probabilistic models with latent variables.
- In general, the *incomplete data likelihood function* $p_\theta(x)$ is hard to optimize, but the *complete data likelihood function* $p_\theta(x, z)$ is easier to work with.

Lower bound

Given any distribution $q(z)$, we have

$$\begin{aligned} \sum_{i=1}^N \log \int p_\theta(x^{(i)}, z) dz &= \sum_{i=1}^N \log \int q(z) \frac{p_\theta(x^{(i)}, z)}{q(z)} dz \\ &= \sum_{i=1}^N \log \mathbb{E}_{q(z)} \left[\frac{p_\theta(x^{(i)}, z)}{q(z)} \right] \\ &\geq \sum_{i=1}^N \mathbb{E}_{q(z)} \left[\log \frac{p_\theta(x^{(i)}, z)}{q(z)} \right] = \sum_{i=1}^N \int q(z) \log \frac{p_\theta(x^{(i)}, z)}{q(z)} dz, \end{aligned}$$

where the last line follows by Jensen's inequality.

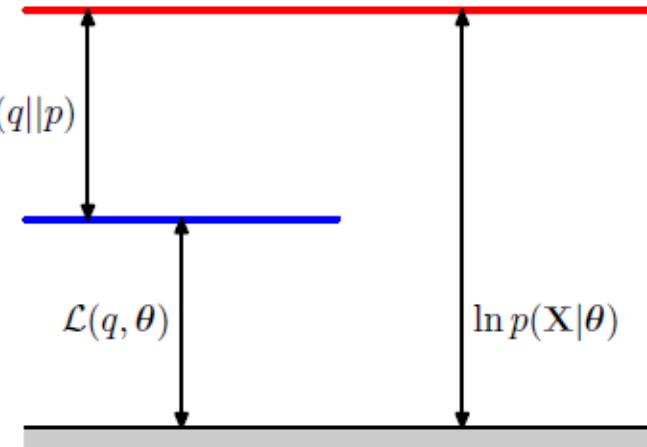
- The lower bound

$$\mathcal{L}(q, \theta) = \sum_{i=1}^N \int q(z) \log \frac{p_\theta(x^{(i)}, z)}{q(z)} dz$$

holds for all distributions $q(z)$, but which one is the best?

- We have the following formula which gives the difference between the log-likelihood and the lower bound:

$$\log_\theta p(x^{(i)}) - \mathcal{L}(q, \theta) = D_{KL} [q(z) \mid\mid p_\theta(z \mid x^{(i)})].$$



- Recall that the KL-divergence is ≥ 0 , and equals 0 when $q(z) = p_\theta(z|x^{(i)})$, in which case the lower bound is equal to the log-likelihood.

EM algorithm

- (i) E-step: Optimize lower bound with respect to q

$$q_{t+1}(z) := \arg \max_q \mathcal{L}(q, \theta_t)$$

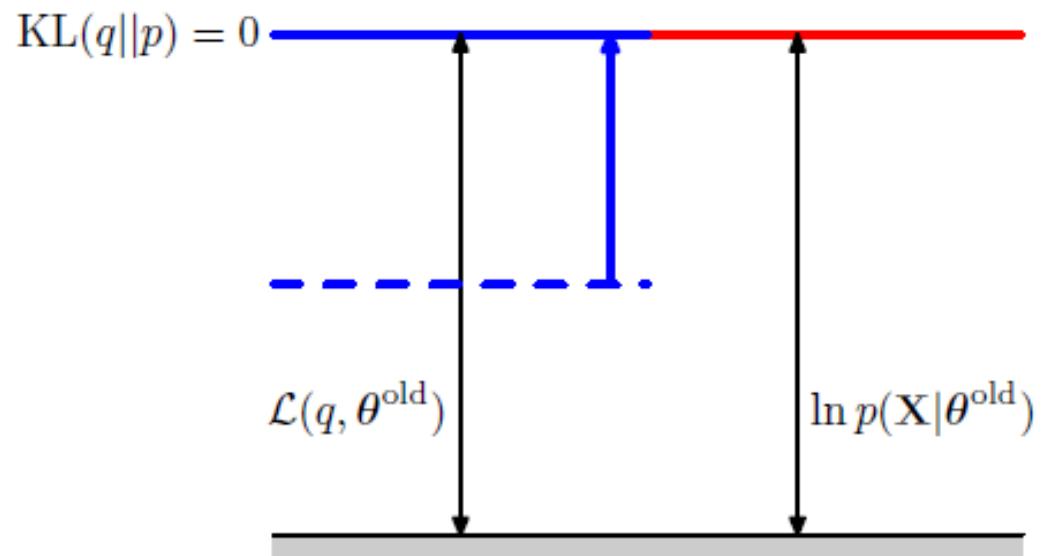
- (ii) M-step: Optimize lower bound with respect to θ

$$\begin{aligned} \theta_{t+1} &:= \arg \max_{\theta} \mathcal{L}(q_{t+1}, \theta) \\ &= \arg \max_{\theta} \sum_{i=1}^N \int q_{t+1}(z) \log \frac{p_{\theta}(x^{(i)}, z)}{q_{t+1}(z)} dz \end{aligned}$$

- (iii) Go back to step (i) until the increase in $\ell_{\theta}(\mathbf{x})$ falls below some predetermined threshold.

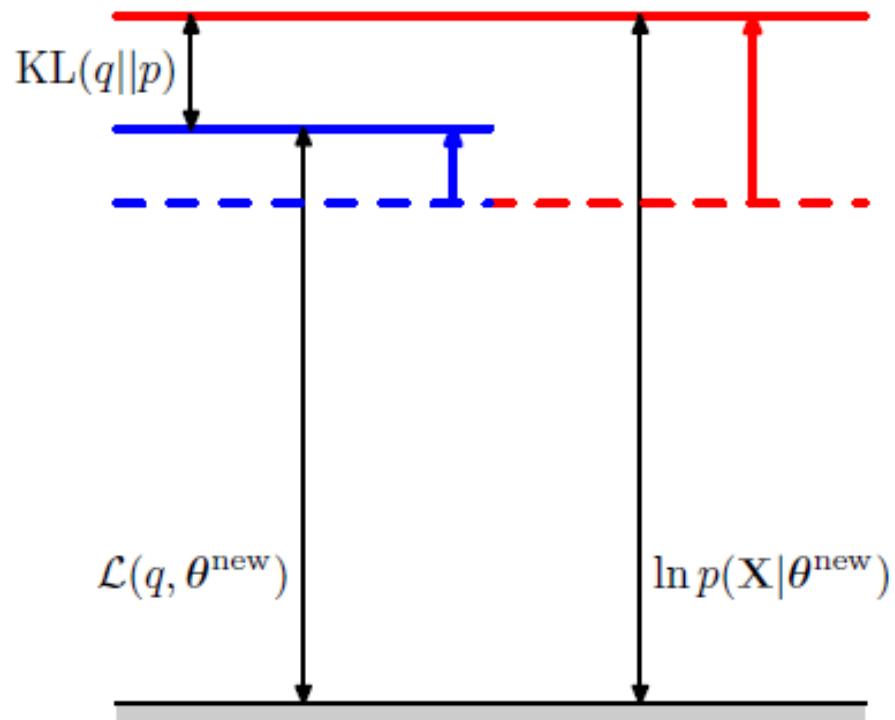
E-step

Illustration of the E step of the EM algorithm. The q distribution is set equal to the posterior distribution for the current parameter values θ^{old} , causing the lower bound to move up to the same value as the log likelihood function, with the KL divergence vanishing.



M-step

Illustration of the M step of the EM algorithm. The distribution $q(Z)$ is held fixed and the lower bound $\mathcal{L}(q, \theta)$ is maximized with respect to the parameter vector θ to give a revised value θ^{new} . Because the KL divergence is nonnegative, this causes the log likelihood $\ln p(X|\theta)$ to increase by at least as much as the lower bound does.



Monotone convergence theorem

Theorem

Let $\{a_n\}$ be an monotonically non-decreasing sequence; i.e. $a_{n+1} \geq a_n$ for all n . If $\{a_n\}$ is bounded above by some constant c , then the sequence converges.

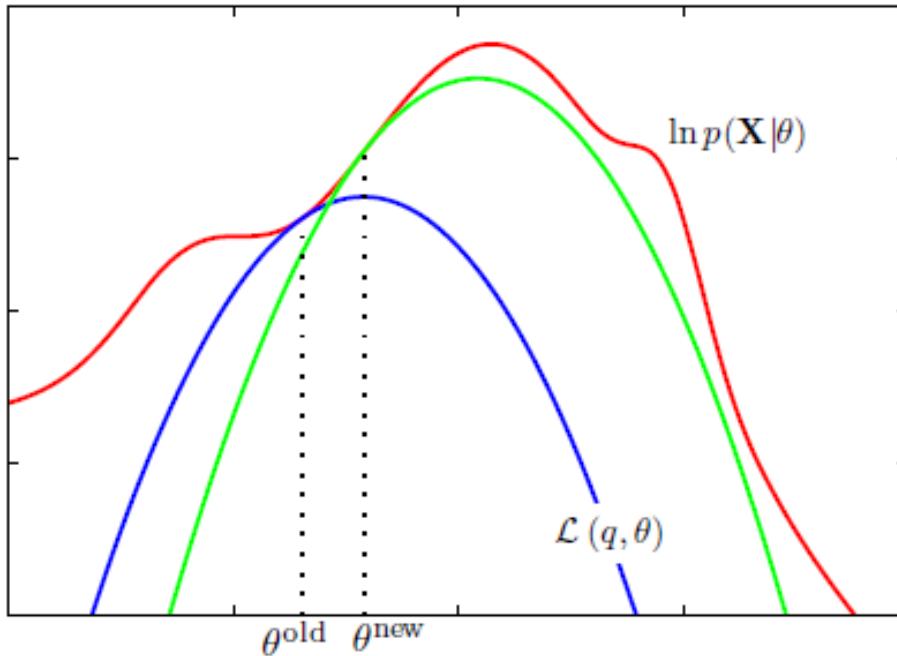
Convergence

- Note that

$$\begin{aligned}\ell_{\theta_{t+1}}(\mathbf{x}) &\geq \sum_{i=1}^N \int q_{t+1}(z) \log \frac{p_{\theta_{t+1}}(x^{(i)}, z)}{q_{t+1}(z)} dz \\ &\geq \sum_{i=1}^N \int q_{t+1}(z) \log \frac{p_{\theta_t}(x^{(i)}, z)}{q_{t+1}(z)} dz \\ &= \ell_{\theta_t}(\mathbf{x}).\end{aligned}$$

- The first inequality follows from the definition of the lower bound, the second follows from the M-step, and the third equality is a result of the E-step which sets $D_{KL}[q(z) \mid p_{\theta_t}(z|x_i)]$ to 0.
- Thus, we get convergence from Monotone convergence theorem since we have a monotonically non-decreasing sequence which is bounded above by 0.

Another view of EM



- Blue curve: Lower bound after E-step at previous iteration
- Green curve: Lower bound after E-step at current iteration

- In a complex model like a VAE, $p_\theta(z|x^{(i)})$ is intractable, so we cannot directly set

$$q_{t+1}(z) := p_{\theta_t}(z \mid x^{(i)}),$$

which also means the KL-divergence is never exactly 0.

- Instead, we approximate the conditional distribution by considering a restricted family of (parameterized) distributions for q . For VAEs, q is modeled using a neural network with parameters ϕ and the lower bound

$$\mathbb{E}_{q_\phi(z|x^{(i)})} \left[\log \frac{p_\theta(x^{(i)}, z)}{q_\phi(z \mid x^{(i)})} \right]$$

is maximized with respect to θ and ϕ together.

- Furthermore, we have

$$\begin{aligned} & \mathbb{E}_{q_\phi(z|x^{(i)})} \left[\log \frac{p_\theta(x^{(i)}, z)}{q_\phi(z | x^{(i)})} \right] \\ &= \mathbb{E}_{q_\phi(z|x^{(i)})} \left[\log p_\theta(x^{(i)}|z) \right] - D_{KL} \left[q_\phi \left(z|x^{(i)} \right) \middle| p(z) \right]. \end{aligned}$$

- The second term in can be computed analytically, and the first term can be approximated by

$$\frac{1}{L} \sum_{l=1}^L \log p_\theta \left(x^{(i)} \mid z^{(i,l)} \right),$$

where $z^{(i,l)}$ is drawn (L times) from the distribution of Z .

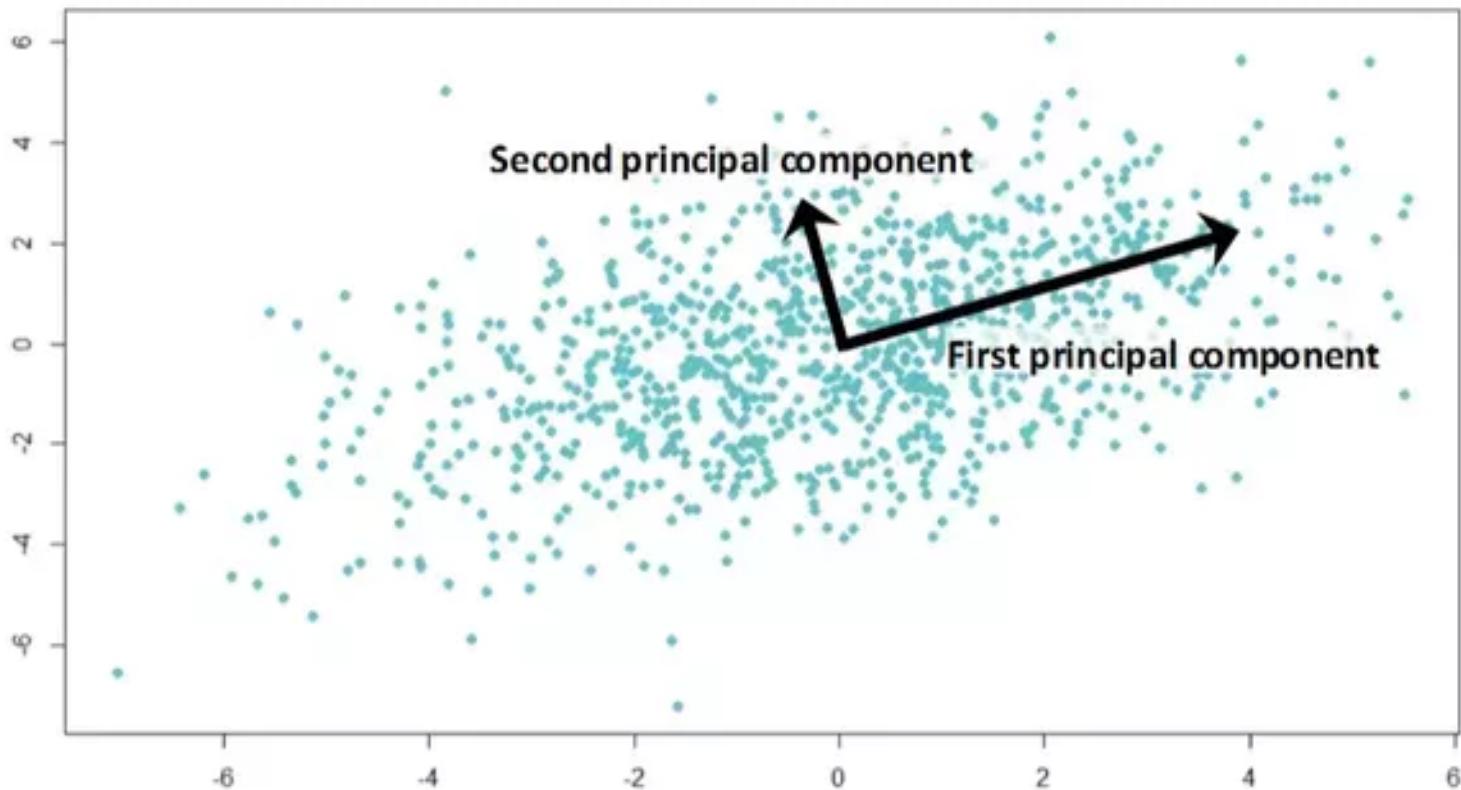
Principal Component Analysis (PCA)

- Imagine we want to predict the results of the next election. There are many variables to base our prediction on, eg. the recent trend in housing prices, the local and global stock markets, crime rate, wages, recent changes in taxes etc.
- From our lesson on regression, we know we can fit all this data into a design matrix X , where the data-points $x^{(i)}$ are the rows of X , each of which are vectors in \mathbb{R}^p (thus X has p columns).

- Each component in $x^{(i)}$, and there are p of them, represents a particular variable you want to model.
- What if p is too large and we want to only consider the most important L ($< p$) variables?
- What if the variables are not independent from one another but are correlated in a way that is unknown to us?

- Rather than simply eliminating some of the variables haphazardly, we want to perform a transformation on the data-points such that after the transformation, the new variables/components are
 - (i) independent from one another
 - (ii) ranked in terms of their importance (axes with the most variation), so we can choose the L most important ones (dimensionality reduction)
- Note that one potential drawback is that such a transformation would lead to a loss in interpretability of the data.

Principal components



Singular value decomposition (SVD)

Theorem

Given any real $n \times p$ matrix M , there exists a factorization of M of the form

$$M = U\Sigma V^T$$

where

- U is an orthogonal $n \times n$ matrix,
- Σ is a $n \times p$ matrix with non-negative real numbers on the diagonal and zero elsewhere,
- V is an orthogonal $p \times p$ matrix.

The diagonal entries of Σ are called the singular values of M .

PCA steps

Given a design matrix X , perform the following steps:

- (1) For each column of X , compute the mean and subtract it from each entry in the column. At the same time, compute the standard deviation and divide each entry in the column by it (this ensures that each column is normalized to have mean 0 and standard deviation 1).
- (2) Do SVD on X to yield $X = U\Sigma V^T$, then multiply X by V to get $XV = U\Sigma$.
- (3) Retain the first L columns of XV and remove the remaining columns.

Why does PCA work?

We want to find a transformation $x^{(i)}Z = t^{(i)} = [t_1^{(i)}, \dots, t_p^{(i)}]$
where the variance of the first component

$$t_1^{(i)} = \langle x^{(i)}, z_1 \rangle \quad (z_1 \text{ is the first column of } Z)$$

is maximized, i.e.

$$\begin{aligned} z_1 &= \arg \max_{\|z\|=1} \sum_{i=1}^n \left(t_1^{(i)} \right)^2 = \arg \max_{\|z\|=1} \sum_{i=1}^n \left(\langle x^{(i)}, z \rangle \right)^2 \\ &= \arg \max_{\|z\|=1} \|Xz\|^2 = \arg \max_{\|z\|=1} z^T X^T X z. \end{aligned}$$

- Let $\lambda_1, \dots, \lambda_p$ denote the eigenvalues of $X^T X$ arranged in descending order, and let v_1, \dots, v_p denote the corresponding eigenvectors. As the eigenvectors form an orthonormal basis, we can represent

$$z = a_1 v_1 + \dots + a_p v_p,$$

where $a_i = \langle z, v_i \rangle$ for all i .

- Then we have

$$\begin{aligned} z^T X^T X z &= \langle z, X^T X z \rangle \\ &= \langle a_1 v_1 + \dots + a_p v_p, a_1 \lambda_1 v_1 + \dots + a_p \lambda_p v_p \rangle \\ &= a_1^2 \lambda_1 + \dots + a_p^2 \lambda_p, \end{aligned}$$

and essentially we need to find (a_1, \dots, a_p) with $\sum_i a_i^2 = 1$ which maximizes this expression.

- Hence, the vector which maximizes the variance of the first component must be the eigenvector v_1 , where $(a_1, \dots, a_p) = (1, \dots, 0)$, of $X^T X$ corresponding to its largest eigenvalue λ_1 .
- Similarly, to compute the next most variable component $t_2^{(i)}$ one must compute the second eigenvector of $X^T X$, and so on.
- $X^T X$ is proportional to the sample covariance matrix of the original dataset, and recall that we need to first compute it to obtain V in the SVD of X .
- Thus, we see that V is precisely the transformation Z we are looking for.

Hidden Markov Models (HMMs)

More details can be found in

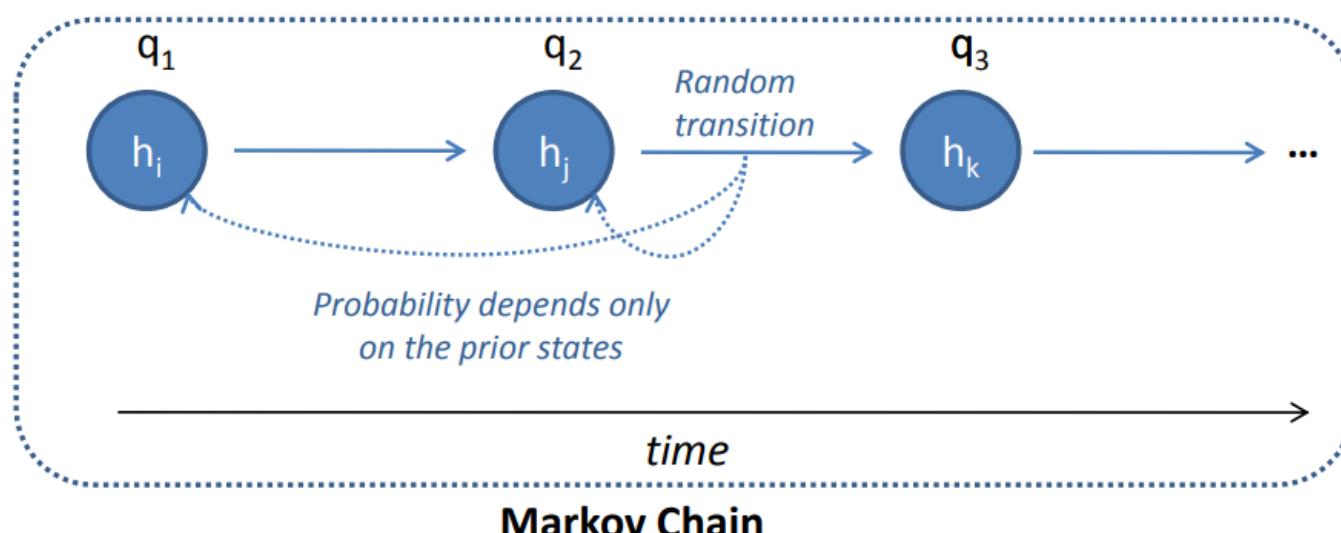
- Chapter 13 "Sequential Data" of Bishop's *Pattern Recognition and Machine Learning*

Applications

- Speech generation, speech recognition
- Financial forecasting: stock prices, currency exchange rates etc.
- DNA sequencing
- Weather prediction
- Video analytics
- Machine translation

Markov Chains

- Markov Chains model sequential processes.
- Consider a discrete random variable q with states $\{h_1, \dots, h_n\}$.
- State of q changes randomly in discrete time steps.
- Transition probability depends only on the k previous states.
 - Markov Property



Transition Probabilities

■ Most simplest Markov chain:

- Transition Probability depends only on the previous state (i.e. k=1):

$$P(q_t = h_i | q_{t-1}, \dots, q_1) = P(q_t = h_i | q_{t-1})$$

- Transition Probability is time invariant:

$$P(q_t = h_i | q_{t-1}) = P(q_{t-1} = h_i | q_{t-2})$$

■ In this case, the Markov chain is defined by:

- An $(n \times n)$ Matrix T containing state change probabilities:

$$T_{ij} = P(q_t = h_i | q_{t-1} = h_j)$$

- An n-dimensional vector π containing initial state probabilities:

$$\pi_i = P(q_1 = h_i)$$

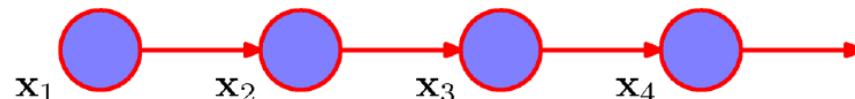
- Since π and K contain probabilities, they have to be normalized:

$$\sum_{i=1}^n \pi_i = 1 \quad \forall a : \sum_{i=1}^n K_{ai} = 1$$

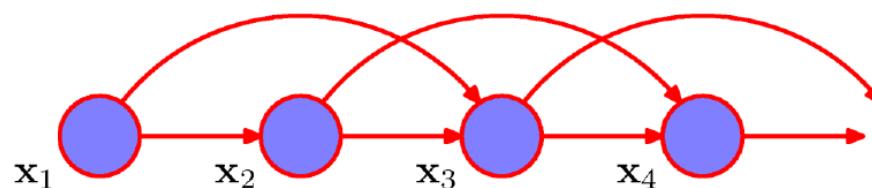
Nth Order Markov Chain

- Markov Assumption

1st order $p(\mathbf{X}) = \prod_{i=1}^n p(X_n | X_{n-1})$



2nd order $p(\mathbf{X}) = \prod_{i=1}^n p(X_n | X_{n-1}, X_{n-2})$



Prelude

- Given observations (O_1, \dots, O_n) , hidden states (S_1, \dots, S_n) , we want to define a model by providing structure to our joint distribution $p(O_1, \dots, O_n, S_1, \dots, S_n)$. We have

$$p(O_1, \dots, O_n, S_1, \dots, S_n) = p(S_1, \dots, S_n)p(O_1, \dots, O_n | S_1, \dots, S_n)$$

using chain rule.

- Repeated application of chain rule to $p(S_1, \dots, S_n)$ yields

$$\begin{aligned} p(S_1, \dots, S_n) &= p(S_1, \dots, S_{n-1})p(S_n | S_1, \dots, S_{n-1}) \\ &= p(S_1, \dots, S_{n-2})p(S_{n-1} | S_1, \dots, S_{n-2})p(S_n | S_1, \dots, S_{n-1}) \\ &\quad \vdots \\ &= p(S_1)p(S_2 | S_1)p(S_3 | S_1, S_2) \cdots p(S_n | S_1, \dots, S_{n-1}). \end{aligned}$$

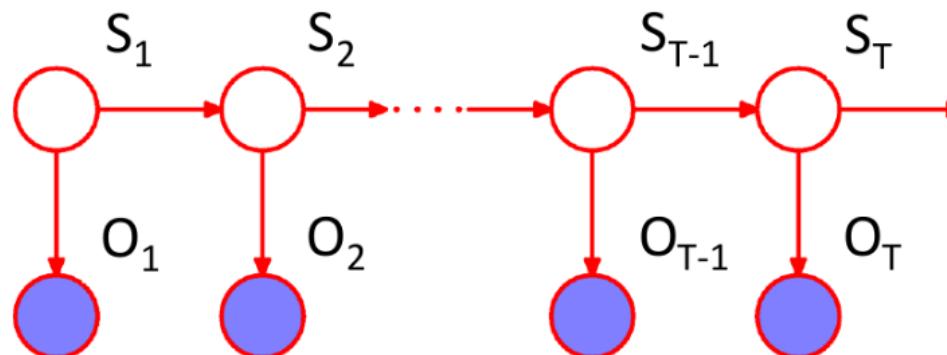
- Similarly for $p(O_1, \dots, O_n | S_1, \dots, S_n)$, we have

$$\begin{aligned} p(O_1, \dots, O_n | S_1, \dots, S_n) \\ = p(O_1 | S_1, \dots, S_n)p(O_2 | O_1, S_1, \dots, S_n) \cdots p(O_n | O_1, \dots, O_{n-1}, S_1, \dots, S_n). \end{aligned}$$

- So far we have enforced no assumptions; we will do so in the next few slides.

HMM definition

- Distributions that characterize sequential data with few parameters but are not limited by strong Markov assumptions.



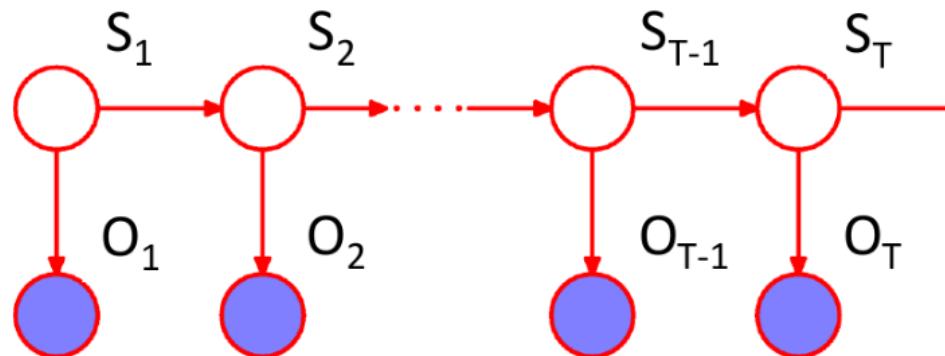
Observation space

$$O_t \in \{y_1, y_2, \dots, y_K\}$$

Hidden states

$$S_t \in \{1, \dots, I\}$$

Joint Distribution Factorization



$$p(S_1, \dots, S_T, O_1, \dots, O_T) = \prod_{t=1}^T p(O_t | S_t) \prod_{t=1}^T p(S_t | S_{t-1})$$

1st order Markov assumption on hidden states $\{S_t\}$ $t = 1, \dots, T$
(can be extended to higher order).

Note: O_t depends on all previous observations $\{O_{t-1}, \dots, O_1\}$

Model Parameters

- Parameters – stationary/homogeneous markov model
(independent of time t)

Initial probabilities

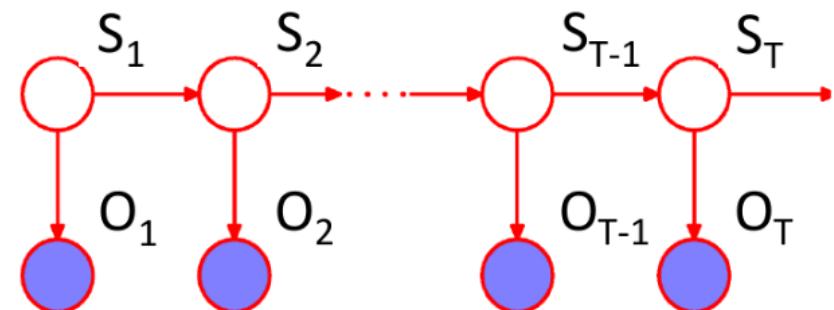
$$p(S_1 = i) = \pi_i$$

Transition probabilities

$$p(S_t = j | S_{t-1} = i) = p_{ij}$$

Emission probabilities

$$p(O_t = y | S_t = i) = q_i^y$$



$$p(\{S_t\}_{t=1}^T, \{O_t\}_{t=1}^T) =$$

$$p(S_1) \prod_{t=2}^T p(S_t | S_{t-1}) \prod_{t=1}^T p(O_t | S_t)$$

- The Dishonest Casino

A casino has two die:

Fair dice

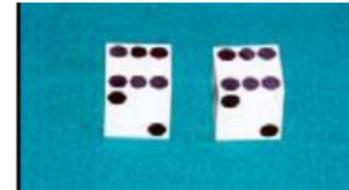
$$P(1) = P(2) = P(3) = P(5) = P(6) = 1/6$$

Loaded dice

$$P(1) = P(2) = P(3) = P(5) = 1/10$$

$$P(6) = \frac{1}{2}$$

Casino player switches back-&-forth between fair and loaded die once every 20 turns



HMM Example

GIVEN: A sequence of rolls by the casino player

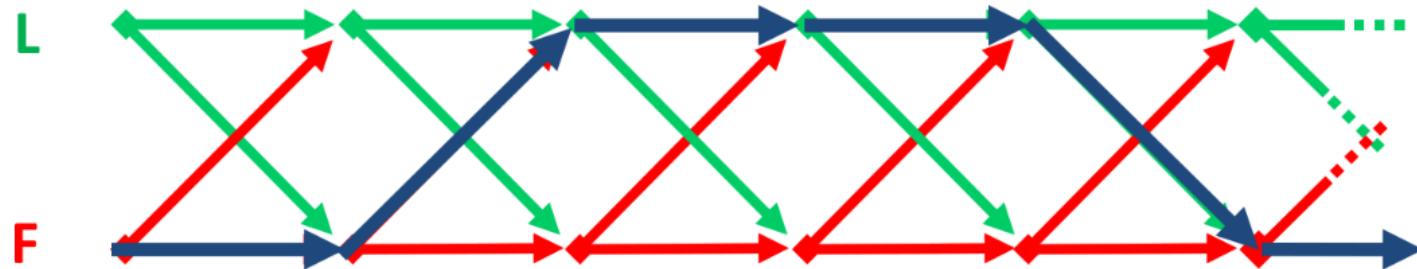
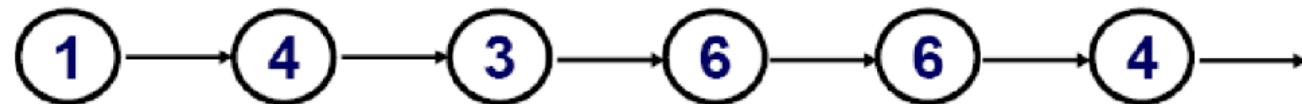
1245526462146146136136661664661636616366163616515615115146123562344

QUESTION

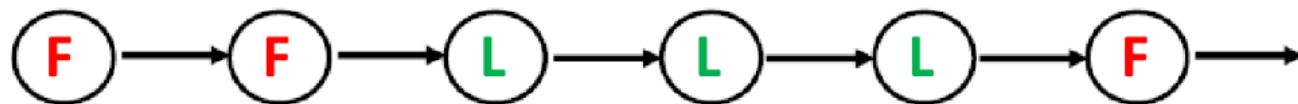
- How likely is this sequence, given our model of how the casino works?
 - This is the **EVALUATION** problem in HMMs
- What portion of the sequence was generated with the fair die, and what portion with the loaded die?
 - This is the **DECODING** question in HMMs
- How “loaded” is the loaded die? How “fair” is the fair die? How often does the casino player change from fair to loaded, and back?
 - This is the **LEARNING** question in HMMs

HMM Example

- Observed sequence: $\{O_t\}_{t=1}^T$

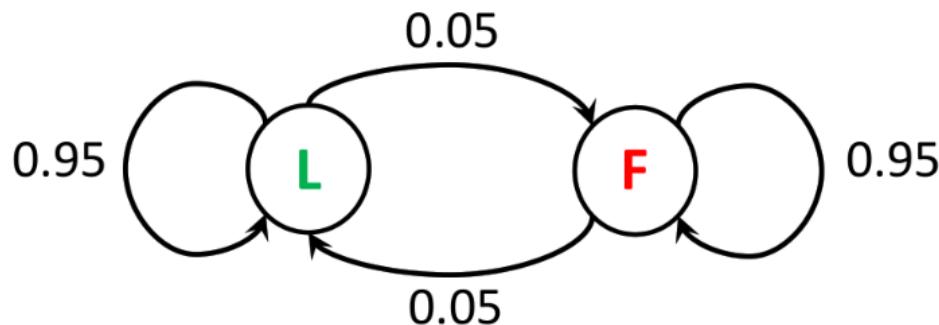


- Hidden sequence $\{S_t\}_{t=1}^T$ (or segmentation):



HMM Example

- Switch between **F** and **L** once every 20 turns ($1/20 = 0.05$)



- HMM Parameters

Initial probs

$$P(S_1 = \text{L}) = 0.5 = P(S_1 = \text{F})$$

Transition probs

$$P(S_t = \text{L/F} | S_{t-1} = \text{L/F}) = 0.95$$

$$P(S_t = \text{F/L} | S_{t-1} = \text{L/F}) = 0.05$$

Emission probabilities

$$P(O_t = y | S_t = \text{F}) = 1/6 \quad y = 1, 2, 3, 4, 5, 6$$

$$\begin{aligned} P(O_t = y | S_t = \text{L}) &= 1/10 \quad y = 1, 2, 3, 4, 5 \\ &= 1/2 \quad y = 6 \end{aligned}$$

Three Main Learning Problems

- **Evaluation** – Given HMM parameters & observation seqn $\{O_t\}_{t=1}^T$
find $p(\{O_t\}_{t=1}^T)$ prob of observed sequence
- **Decoding** – Given HMM parameters & observation seqn $\{O_t\}_{t=1}^T$
find $\arg \max_{s_1, \dots, s_T} p(\{S_t\}_{t=1}^T | \{O_t\}_{t=1}^T)$ most probable sequence of hidden states
- **Learning** – Given HMM with unknown parameters and $\{O_t\}_{t=1}^T$ observation sequence
find $\arg \max_{\theta} p(\{O_t\}_{t=1}^T | \theta)$ parameters that maximize likelihood of observed data

Three Main Learning Problems

- **Evaluation** – What is the probability of the observed sequence? [Forward Algorithm](#)
- **Decoding** – What is the probability that the third roll was loaded given the observed sequence? [Forward-Backward Algorithm](#)
 - What is the most likely die sequence given the observed sequence? [Viterbi Algorithm](#)
- **Learning** – Under what parameterization is the observed sequence most probable? [Baum-Welch Algorithm \(EM\)](#)

Evaluation: Forward Algorithm

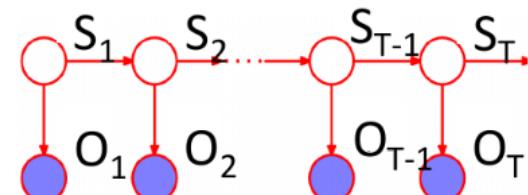
- Given HMM parameters $p(S_1), p(S_t|S_{t-1}), p(O_t|S_t)$ & observation sequence $\{O_t\}_{t=1}^T$

find probability of observed sequence

$$\begin{aligned} p(\{O_t\}_{t=1}^T) &= \sum_{S_1, \dots, S_T} p(\{O_t\}_{t=1}^T, \{S_t\}_{t=1}^T) \\ &= \sum_{S_1, \dots, S_T} p(S_1) \prod_{t=2}^T p(S_t|S_{t-1}) \prod_{t=1}^T p(O_t|S_t) \end{aligned}$$

requires summing over all possible hidden state values at all times – K^T exponential # terms!

Instead: $p(\{O_t\}_{t=1}^T) = \sum_k \underbrace{p(\{O_t\}_{t=1}^T, S_T = k)}_{\alpha_T^k} \text{ Compute recursively}$



Evaluation: Forward Algorithm

$$p(\{O_t\}_{t=1}^T) = \sum_k p(\{O_t\}_{t=1}^T, S_T = k) = \sum_k \alpha_T^k$$

Compute forward probability α_t^k recursively over t

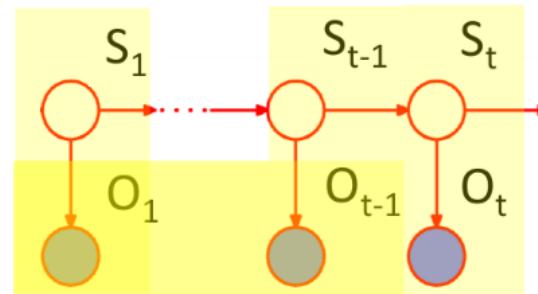
$$\alpha_t^k := p(O_1, \dots, O_t, S_t = k)$$

Introduce S_{t-1}

Chain rule

Markov assumption

$$= p(O_t | S_t = k) \sum_i \alpha_{t-1}^i p(S_t = k | S_{t-1} = i)$$



Evaluation: Forward Algorithm

Can compute α_t^k for all k, t using dynamic programming:

- Initialize: $\alpha_1^k = p(O_1 | S_1 = k) p(S_1 = k)$ for all k
- Iterate: for $t = 2, \dots, T$
$$\alpha_t^k = p(O_t | S_t = k) \sum_i \alpha_{t-1}^i p(S_t = k | S_{t-1} = i) \quad \text{for all } k$$
- Termination: $p(\{O_t\}_{t=1}^T) = \sum_k \alpha_T^k$

Decoding: Backward Algorithm

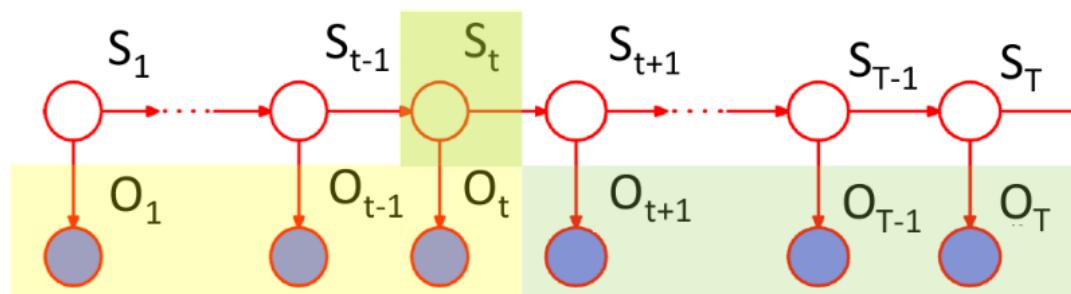
- Given HMM parameters $p(S_1), p(S_t|S_{t-1}), p(O_t|S_t)$ & observation sequence $\{O_t\}_{t=1}^T$
find probability that hidden state at time t was k $p(S_t = k|\{O_t\}_{t=1}^T)$

$$\begin{aligned} p(S_t = k, \{O_t\}_{t=1}^T) &= p(O_1, \dots, O_t, S_t = k, O_{t+1}, \dots, O_T) \\ &= p(O_1, \dots, O_t, S_t = k)p(O_{t+1}, \dots, O_T | S_t = k) \end{aligned}$$

Compute recursively

α_t^k

β_t^k



Decoding: Backward Algorithm

$$p(S_t = k, \{O_t\}_{t=1}^T) = p(O_1, \dots, O_t, S_t = k)p(O_{t+1}, \dots, O_T | S_t = k) = \alpha_t^k \beta_t^k$$

Compute forward probability β_t^k recursively over t

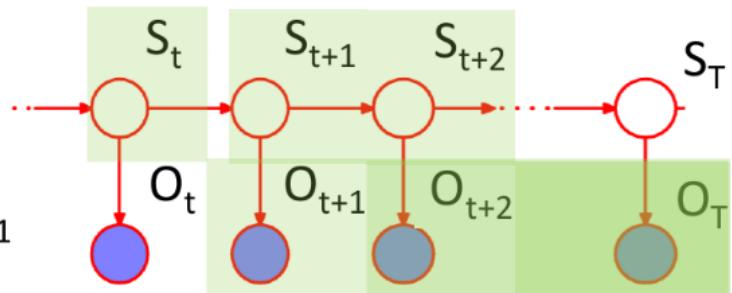
$$\beta_t^k := p(O_{t+1}, \dots, O_T | S_t = k)$$

Introduce S_{t+1}

Chain rule

Markov assumption

$$= \sum_i p(S_{t+1} = i | S_t = k) p(O_{t+1} | S_{t+1} = i) \beta_{t+1}^i$$



Decoding: Backward Algorithm

Can compute β_t^k for all k, t using dynamic programming:

- Initialize: $\beta_T^k = 1$ for all k

- Iterate: for $t = T-1, \dots, 1$

$$\beta_t^k = \sum_i p(S_{t+1} = i | S_t = k) p(O_{t+1} | S_{t+1} = i) \beta_{t+1}^i \quad \text{for all } k$$

- Termination: $p(S_t = k, \{O_t\}_{t=1}^T) = \alpha_t^k \beta_t^k$

$$p(S_t = k | \{O_t\}_{t=1}^T) = \frac{p(S_t = k, \{O_t\}_{t=1}^T)}{p(\{O_t\}_{t=1}^T)} = \frac{\alpha_t^k \beta_t^k}{\sum_i \alpha_t^i \beta_t^i}$$

Decoding: Viterbi Algorithm

- Most likely state assignment at time t

$$\arg \max_k p(S_t = k | \{O_t\}_{t=1}^T) = \arg \max_k \alpha_t^k \beta_t^k$$

E.g. Which die was most likely used by the casino in the third roll given the observed sequence?

- Most likely assignment of state sequence

$$\arg \max_{\{S_t\}_{t=1}^T} p(\{S_t\}_{t=1}^T | \{O_t\}_{t=1}^T)$$

E.g. What was the most likely sequence of die rolls used by the casino given the observed sequence?

Not the same solution !

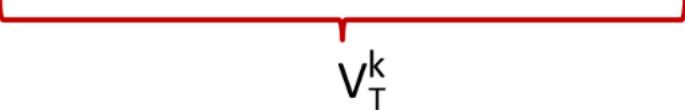
MLA of x?
MLA of (x,y)?

x	y	$P(x,y)$
0	0	0.35
0	1	0.05
1	0	0.3
1	1	0.3

Decoding: Viterbi Algorithm

- Given HMM parameters $p(S_1), p(S_t|S_{t-1}), p(O_t|S_t)$ & observation sequence $\{O_t\}_{t=1}^T$
find most likely assignment of state sequence

$$\begin{aligned}\arg \max_{\{S_t\}_{t=1}^T} p(\{S_t\}_{t=1}^T | \{O_t\}_{t=1}^T) &= \arg \max_{\{S_t\}_{t=1}^T} p(\{S_t\}_{t=1}^T, \{O_t\}_{t=1}^T) \\ &= \arg \max_k \max_{\{S_t\}_{t=1}^{T-1}} p(S_T = k, \{S_t\}_{t=1}^{T-1}, \{O_t\}_{t=1}^T)\end{aligned}$$


 v_T^k

Compute recursively

v_T^k - probability of most likely sequence of states ending at state $S_T = k$

Decoding: Viterbi Algorithm

$$\max_{\{S_t\}_{t=1}^T} p(\{S_t\}_{t=1}^T, \{O_t\}_{t=1}^T) = \max_k V_T^k$$

Compute probability V_t^k recursively over t

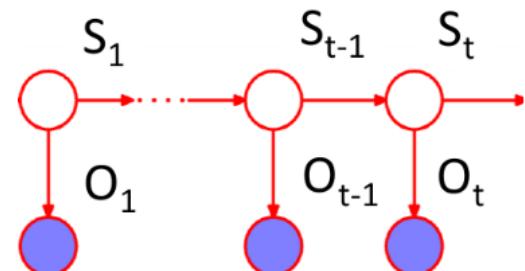
$$V_t^k := \max_{S_1, \dots, S_{t-1}} p(S_t = k, S_1, \dots, S_{t-1}, O_1, \dots, O_t)$$

.

Bayes rule

.

Markov assumption



$$= p(O_t | S_t = k) \max_i p(S_t = k | S_{t-1} = i) V_{t-1}^i$$

Decoding: Viterbi Algorithm

Can compute V_t^k for all k, t using dynamic programming:

- Initialize: $V_1^k = p(O_1 | S_1=k)p(S_1 = k)$ for all k

- Iterate: for $t = 2, \dots, T$

$$V_t^k = p(O_t | S_t = k) \max_i p(S_t = k | S_{t-1} = i) V_{t-1}^i \quad \text{for all } k$$

- Termination: $\max_{\{S_t\}_{t=1}^T} p(\{S_t\}_{t=1}^T, \{O_t\}_{t=1}^T) = \max_k V_T^k$

Traceback: $S_T^* = \arg \max_k V_T^k$

$$S_{t-1}^* = \arg \max_i p(S_t^* | S_{t-1} = i) V_{t-1}^i$$

Learning: Baum-Welch (EM) Algorithm

- Given HMM with unknown parameters $\theta = \{\{\pi_i\}, \{p_{ij}\}, \{q_i^k\}\}$ and observation sequence $\mathbf{O} = \{O_t\}_{t=1}^T$

find parameters that maximize likelihood of observed data

$$\arg \max_{\theta} p(\{O_t\}_{t=1}^T | \theta)$$

But likelihood doesn't factorize since observations not i.i.d.

hidden variables – state sequence $\{S_t\}_{t=1}^T$

EM (Baum-Welch) Algorithm:

E-step – Fix parameters, find expected state assignments

M-step – Fix expected state assignments, update parameters

Learning: Baum-Welch (EM) Algorithm

- Start with random initialization of parameters
- **E-step** – Fix parameters, find expected state assignments

$$\gamma_i(t) = p(S_t = i | O, \theta) = \frac{\alpha_t^i \beta_t^i}{\sum_j \alpha_t^j \beta_t^j}$$

Forward-Backward algorithm

$$\xi_{ij}(t) = p(S_{t-1} = i, S_t = j | O, \theta)$$

$$= \frac{p(S_{t-1} = i | O, \theta) p(S_t = j, O_t, \dots, O_T | S_{t-1} = i, \theta)}{p(O_t, \dots, O_T | S_{t-1} = i, \theta)}$$

$$= \frac{\gamma_i(t-1) p_{ij} q_j^{O_t} \beta_t^j}{\beta_{t-1}^i}$$

Learning: Baum-Welch (EM) Algorithm

- Start with random initialization of parameters
- E-step:

- $\gamma_i(t) = p_\theta(S_t = i \mid O) = \frac{\alpha_t^i \beta_t^i}{\sum_j \alpha_t^j \beta_t^j}$
- $\xi_{ij}(t) = p_\theta(S_{t-1} = i, S_t = j \mid O) = \frac{\gamma_i(t-1) p_{ij} q_j^{O_t} \beta_t^j}{\beta_{t-1}^i}$

- M-step:

- $\pi_i = \frac{\gamma_i(1)}{\sum_{j=1}^K \gamma_j(1)}$
- $p_{ij} = \frac{\sum_{t=2}^T \xi_{ij}(t)}{\sum_{k=1}^K \sum_{t=2}^T \xi_{ik}(t)} = \frac{\sum_{t=2}^T \xi_{ij}(t)}{\sum_{t=2}^T \gamma_i(t)}$
- $q_i^k = \frac{\sum_{t=1}^T \delta_{O_t=k} \gamma_i(t)}{\sum_{t=1}^T \gamma_i(t)}$

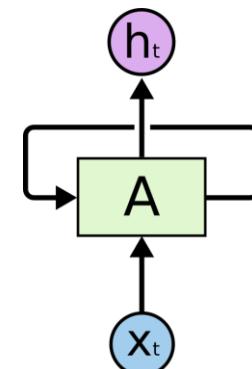
RECURRENT NEURAL NETWORKS



RECURRENT NEURAL NETWORKS

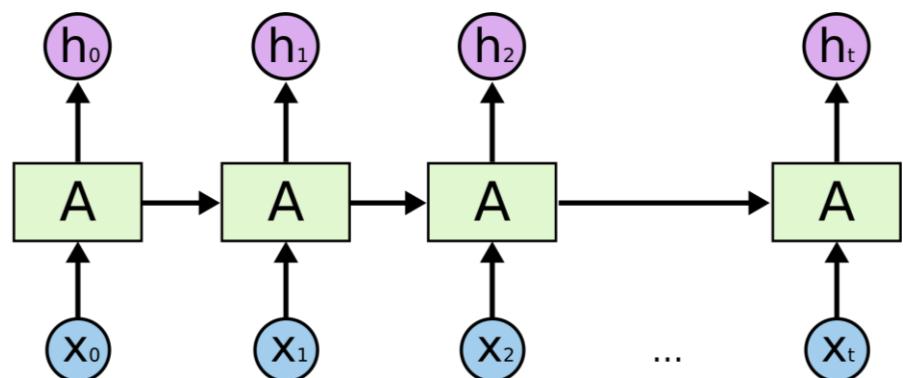
Definition.

Neural networks where the connections form a directed cycle. As opposed to feedforward neural networks.



Redefinition.

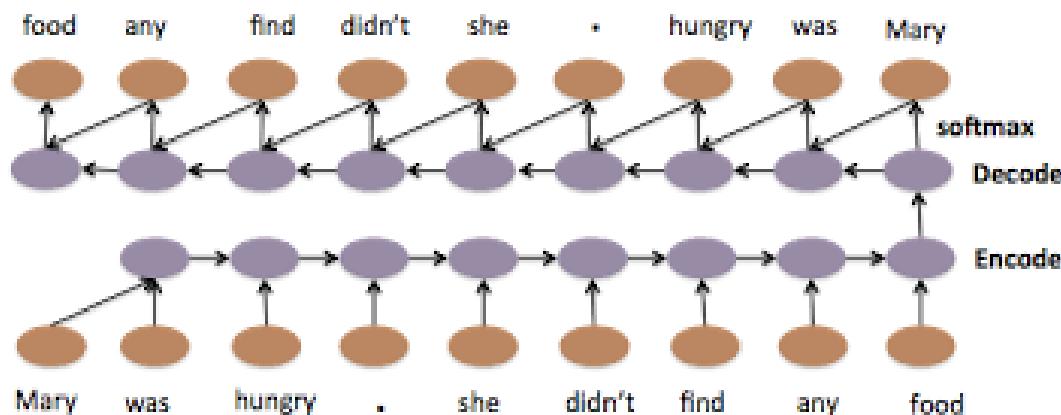
May be unrolled as a feedforward network with shared weights propagating through time.



RECURRENT NEURAL NETWORKS

Temporal State.

- RNN dynamics encode some time-varying state.
- Perfect for modeling sequences, e.g. speech recognition, natural language processing, financial modeling.



PROBLEM WITH VERY DEEP NETWORKS

Vanishing or Exploding Gradient Problem.

- Difficult to train very deep networks because of issues with backpropagation algorithm
- Error signals $\delta^{(l)}$ either shrink exponentially, or grow exponentially with the number of layers

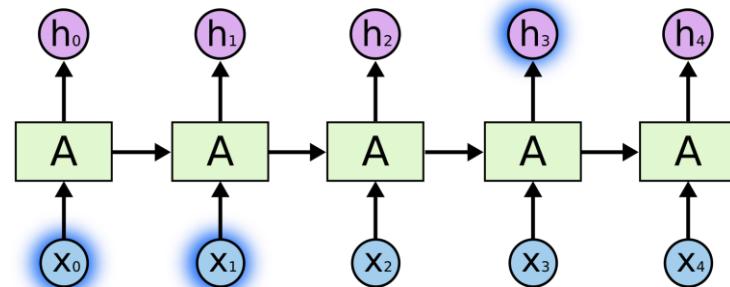
Recurrent Neural Networks are Very Deep!

- We want RNNs to learn long-term dependencies
- But for a long time, nobody knew how to train them

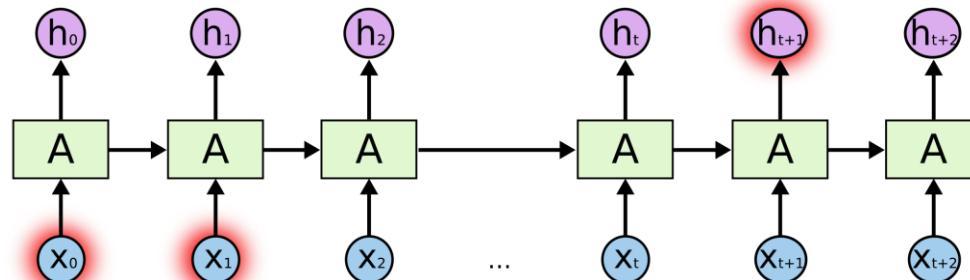


LONG-TERM DEPENDENCIES

“the clouds are in the sky”

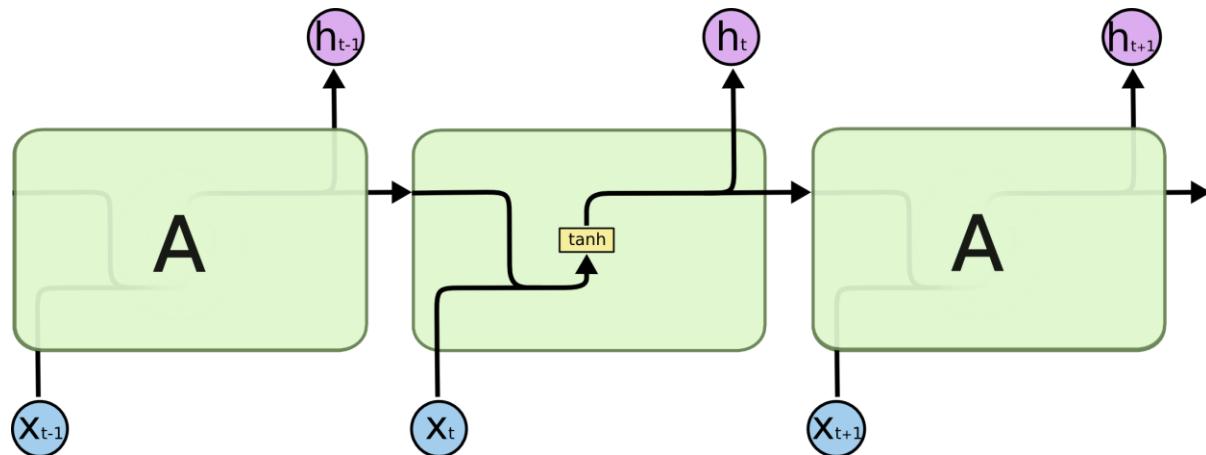


I grew up in France... I speak fluent *French*.



STANDARD RNN

Single Layer RNN.



Neural Network Layer

Pointwise Operation

Vector Transfer

Concatenate

Copy



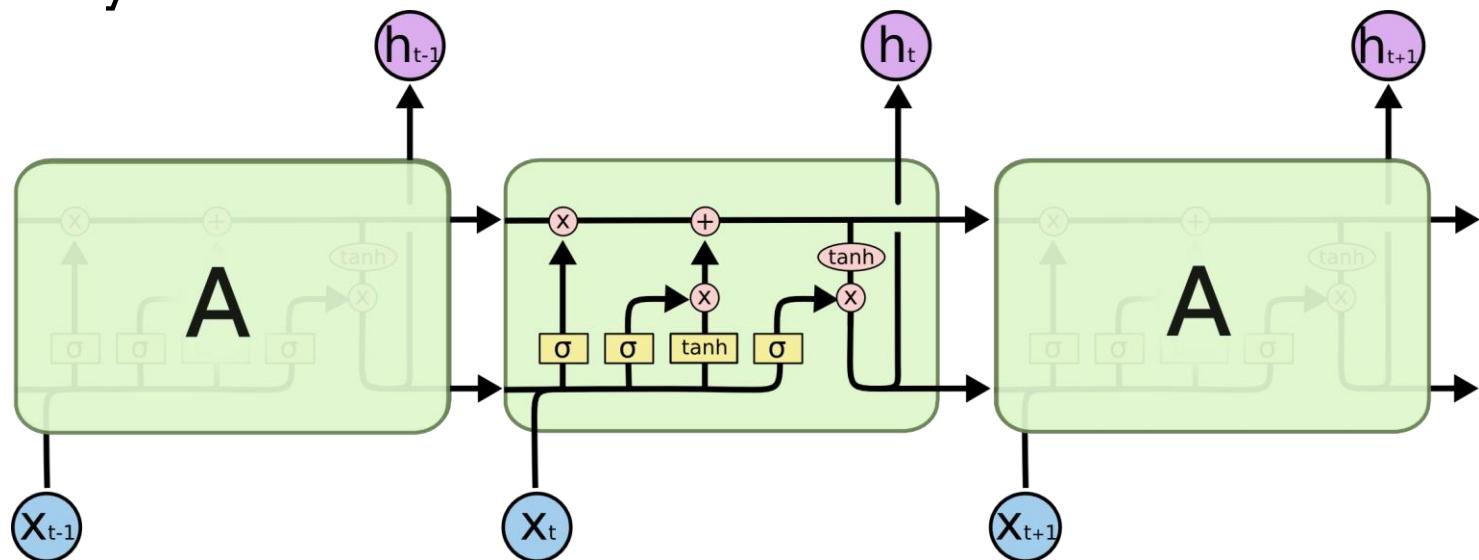


LONG SHORT TERM MEMORY (LSTM)



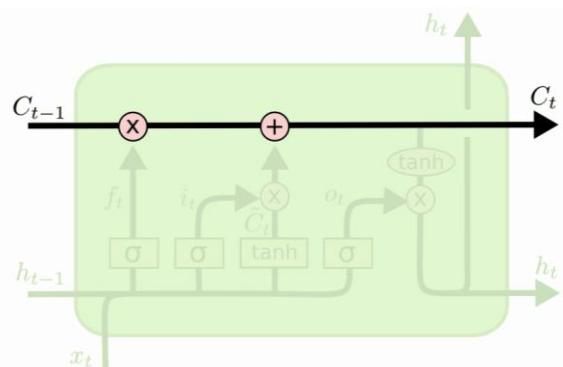
LONG SHORT TERM MEMORY

Four-Layer RNN.

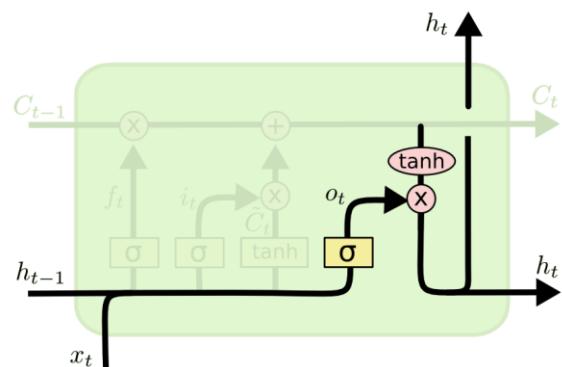


STATES AND GATES

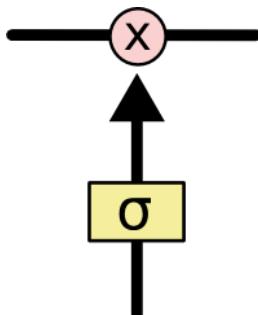
Cell State



Output State



Gates

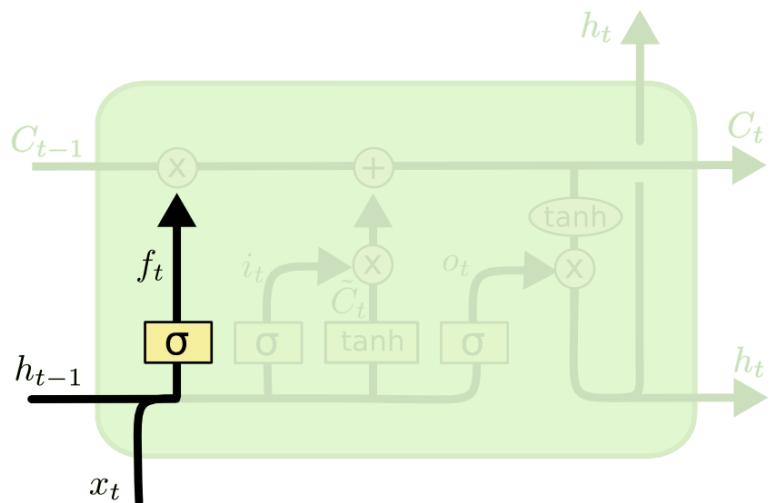


Output of sigmoid
is between 0 and 1
0 – forget past state
1 – keep past state



LAYER 1 – FORGET PAST STATE

Example. Forget gender of subject when subject changes.



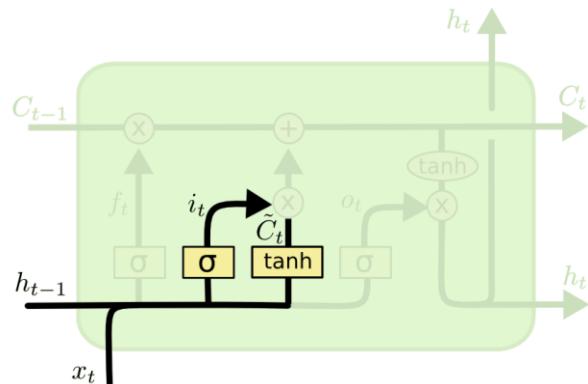
How much to forget?

$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$



LAYER 2,3 – ADD NEW INFORMATION

Example. Add gender of new subject.

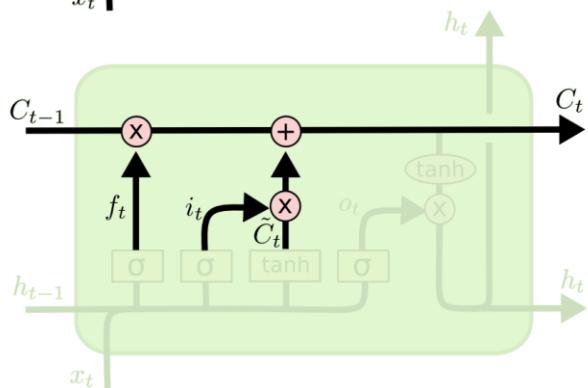


How much to add?

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

What info to add?

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$



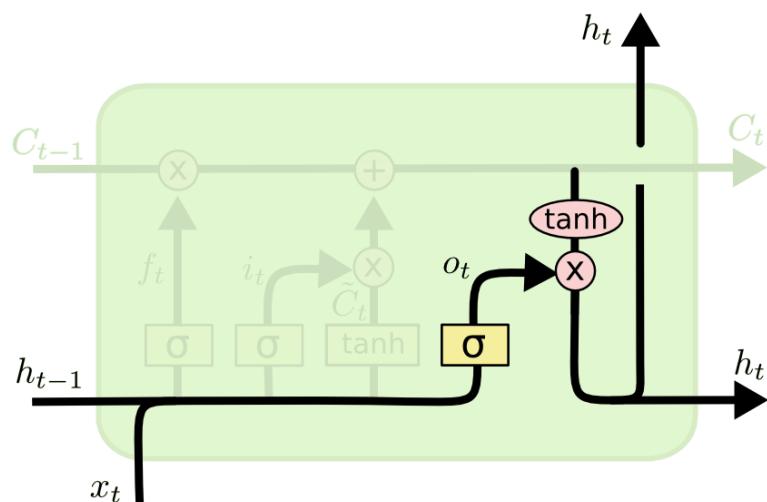
New cell state.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



LAYER 4 – SELECTIVE OUTPUT

Example. Singular or plural form of verb? Output and forward.



How much to output?

$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

What to output?

$$h_t = o_t * \tanh (C_t)$$



Reinforcement learning

Textbook

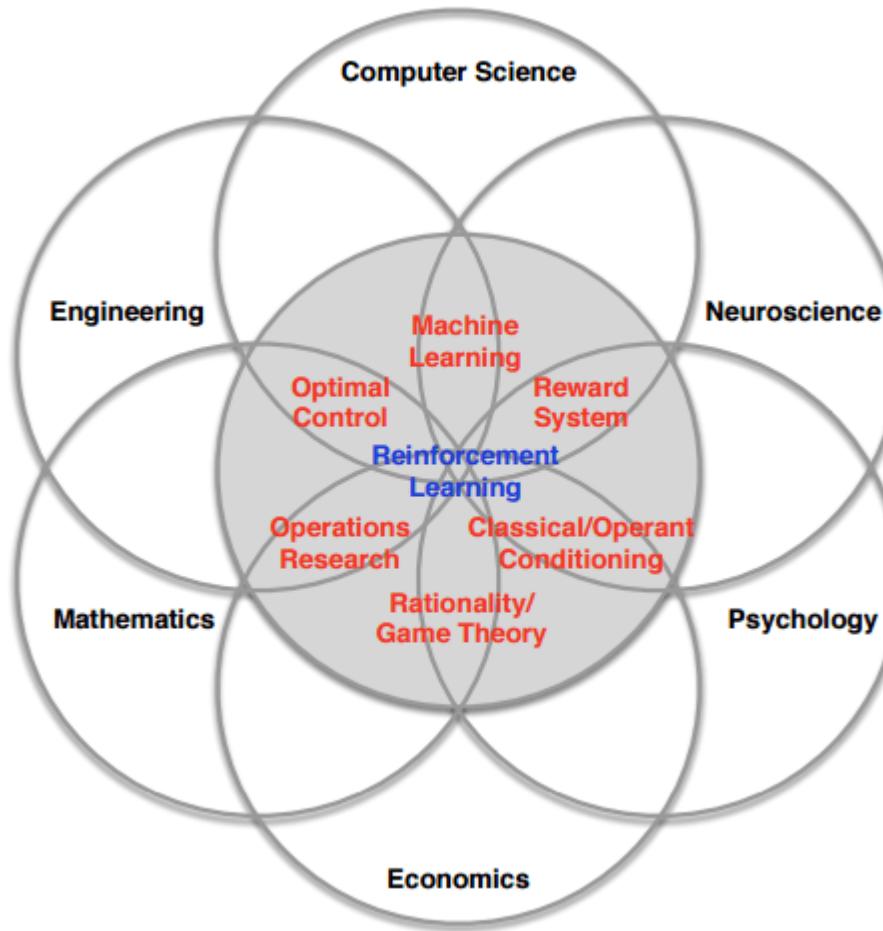
Read the first few chapters (up to and including "Dynamic programming") of

- *Reinforcement Learning: An Introduction* by Sutton and Barto (2018)

What is reinforcement learning?

- Learning how to map situations to actions, so as to maximize a numerical reward.
- Features:
 - Trial and error search
 - Delayed rewards
 - Dilemma of exploration vs exploitation
- Applications:
 - Games
 - Robotics

Reinforcement learning from different views



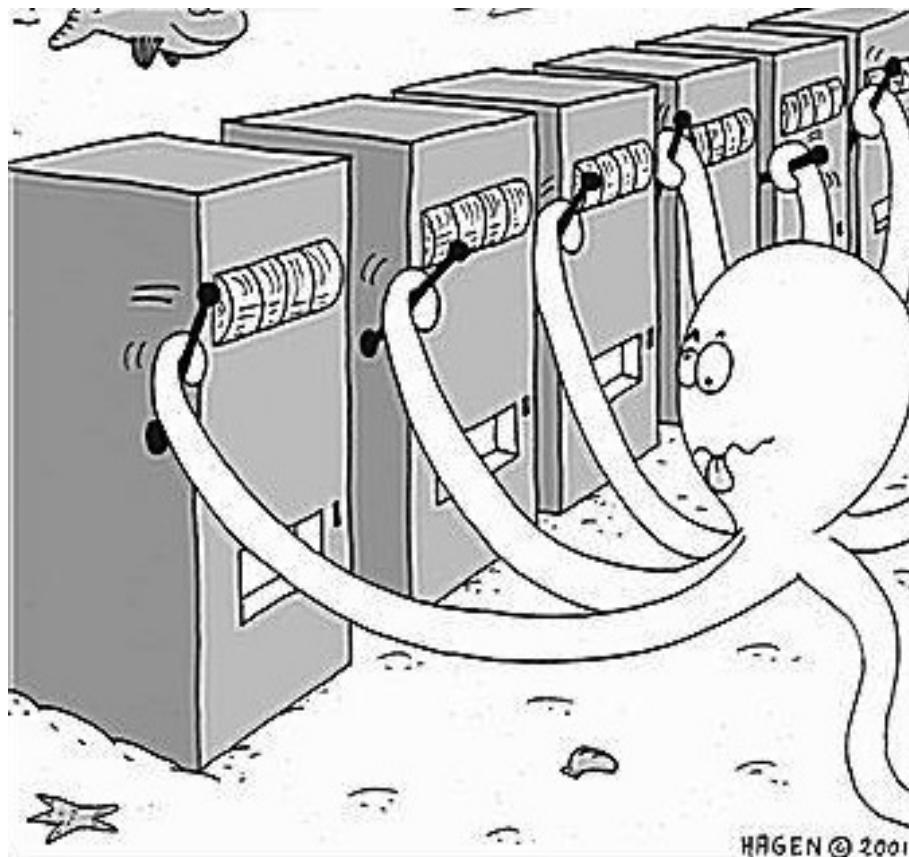
Elements of reinforcement learning

- Policy:
 - defines the agent's behaviour at a given time
 - it is a mapping from states to actions
 - can be stochastic
- Reward signal:
 - at each time step, the environment sends a number to the agent called the reward
 - ultimate goal of the agent is to maximize total reward

Elements of reinforcement learning cont.

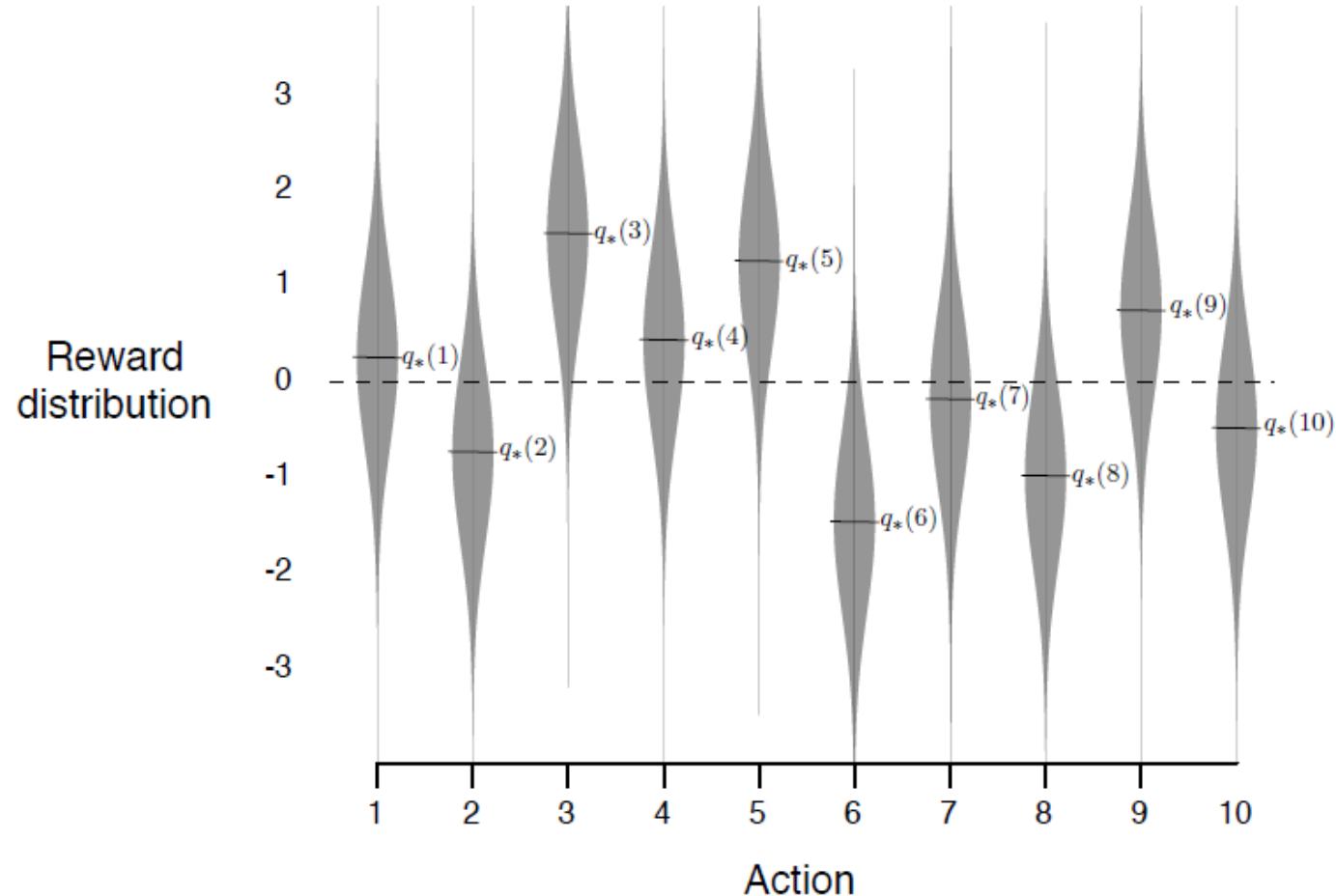
- Value function:
 - the value of a state is the total amount of reward the agent can expect to accumulate over the future, starting from that state
 - reward signal indicates what is good in the immediate sense; the value function indicates what is good in the long run
- Model (optional):
 - approximation of the environment that can be used to predict the next state and reward given the current state and action
 - used for planning
 - model-free methods vs model-based methods

Multi-armed bandit problem



- Imagine there are k jackpot machines in front of you. When you pull the lever of machine i , with some unknown probability p_i you will win \$1, and with probability $1 - p_i$ you will receive nothing.
- The average payouts of the machines may all be different, and the casino affords you 1000 lever pulls before asking you to leave. What should your strategy be?
- Here we are assuming the rewards are distributed as Bernoulli random variables, but they can also be modeled with other distributions (eg. Gaussian).

Gaussian rewards



Action-values

- Before choosing the strategy, let's keep track of our current estimates of how good pulling lever i is, or in more general terms, how good taking action i is. We calculate

$$Q_t(i) = \frac{\text{Sum of rewards from action } i \text{ before } t}{\text{Number of times action } i \text{ is taken prior to } t}$$

for each i and at all times t .

- This is an estimate of the true action-value $Q_*(i)$, which is the expected reward one were to obtain if one were to keep selecting action i .

- By the law of large numbers, we have

$$Q_t(i) \xrightarrow{t \rightarrow \infty} q_*(i)$$

for all i .

- Do not confuse action-values with values: action-values are associated with actions, whereas values are associated with states; i.e.

$$V(s) = \mathbb{E}[R_t | S_t = s],$$

and denotes the reward the agent is expected to receive (following some policy) starting from some state s .

Incremental implementation

$$\begin{aligned} Q_{n+1} &= \frac{1}{n+1} \sum_{i=1}^{n+1} R_i \\ &= \frac{1}{n+1} \left(\sum_{i=1}^n R_i + R_{n+1} \right) \\ &= \frac{1}{n+1} \left(n \left(\frac{1}{n} \right) \sum_{i=1}^n R_i + R_{n+1} \right) \\ &= \frac{1}{n+1} (nQ_n + Q_n - Q_n + R_{n+1}) \\ &= Q_n + \frac{1}{n+1} (R_{n+1} - Q_n) \end{aligned}$$

- The formula for incrementally updating the mean estimate is of the form

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize}[\text{Target} - \text{OldEstimate}].$$

- The expression $[\text{Target} - \text{OldEstimate}]$ is called the *error* of the estimate. It is reduced by taking a step in the direction of *Target*.

Greedy policy

- The simplest strategy is the following, at time t , take the action with the best return so far:

$$A_t = \arg \max_i Q_t(i).$$

- This strategy fully exploits, but does not explore.

Exploitation-Exploration trade-off

- Exploitation:
 - We exploit our current knowledge of the action-values to select the best action.
 - This is the right thing to do to maximize expected reward in one step, or when there is no more uncertainty.
- Exploration:
 - When we select a non-greedy action, we are exploring.
 - We do so despite receiving a smaller reward in the short term, in the hopes of finding better actions, which we can then exploit at later times to maximize reward in the long run.

5 min break

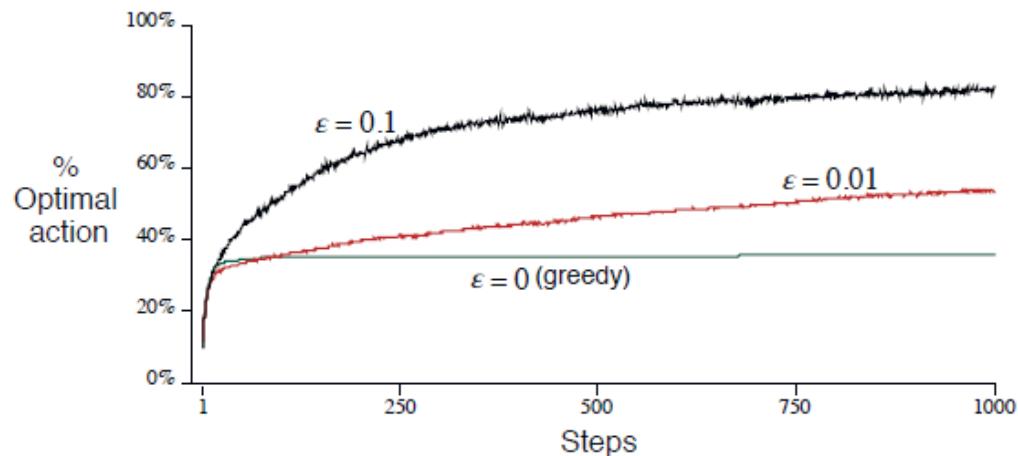
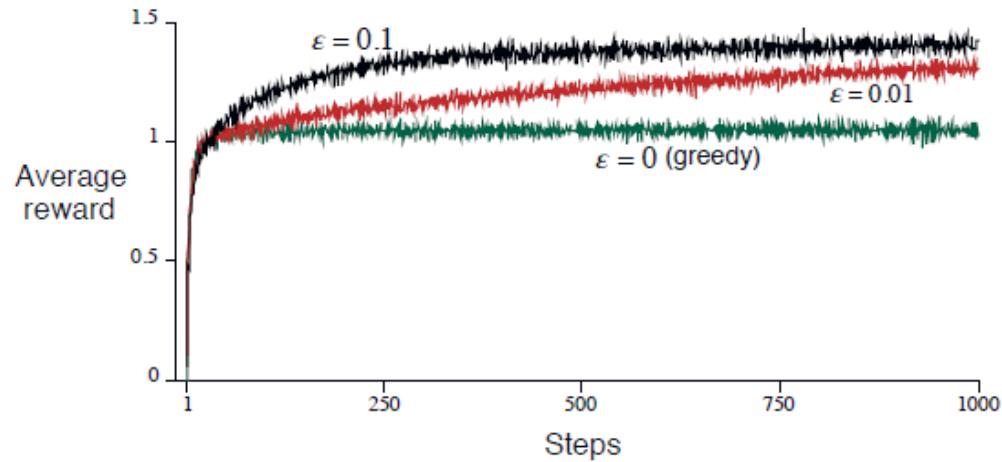
ϵ -greedy policy

- Behave greedily most of the time, but explore once in a while:

$$A_t = \begin{cases} i^* = \arg \max_i Q_t(i) & \text{with probability } 1 - \epsilon \\ j, j \neq i^* & \text{each with probability } \frac{\epsilon}{k-1} \end{cases}$$

- Balances exploitation vs exploration, but does not select intelligently between the $k - 1$ non-greedy actions.

Performance



Upper confidence bound (UCB or UCB1)

- "Optimism in the face of uncertainty."
- Select action at time t according to

$$A_t = \arg \max_i \left(Q_t(i) + c \sqrt{\frac{\log t}{N_t(i)}} \right).$$

- $N_t(i)$ denotes the number of times action i has been selected prior to time t .
- c is the exploration constant; increasing it favours exploration and decreasing it favours exploitation.

Hoeffding's inequality

Theorem

Let X_1, \dots, X_n be i.i.d. random variables with mean μ such that $X_i \in [0, 1]$ for all i . Then

$$P(\bar{X} \geq \mu + U) \leq e^{-2nU^2},$$

where $\bar{X} = \frac{1}{n}(X_1 + \dots + X_n)$.

Explaining the exploration term

- We want our upper bound U to be set such that the probability that our sample estimate \bar{X} exceeds the true mean plus U is very low.
- We also expect our estimate to get better with time, so, although somewhat arbitrary, we can demand that

$$P(\bar{X} \geq \mu + U) \leq \frac{1}{t^k},$$

for some k . Here t denotes the total number of actions taken so thus far.

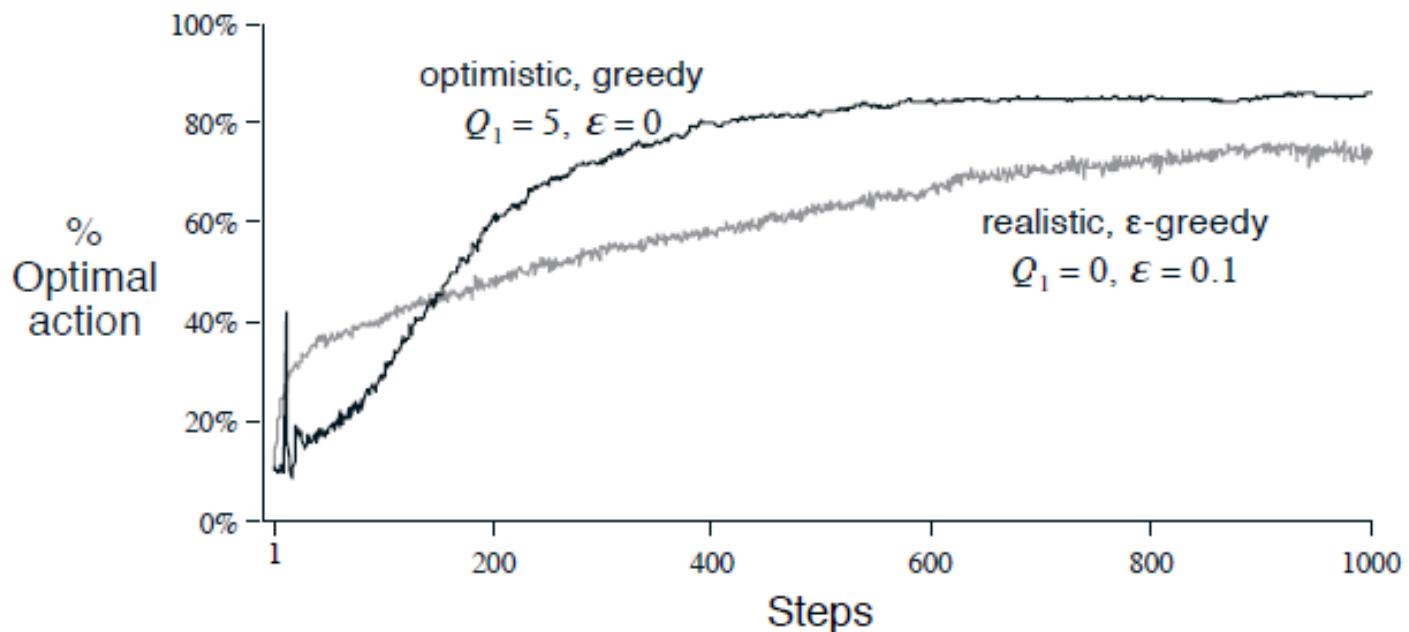
Explaining the exploration term

- Thus using Hoeffding's inequality, we require

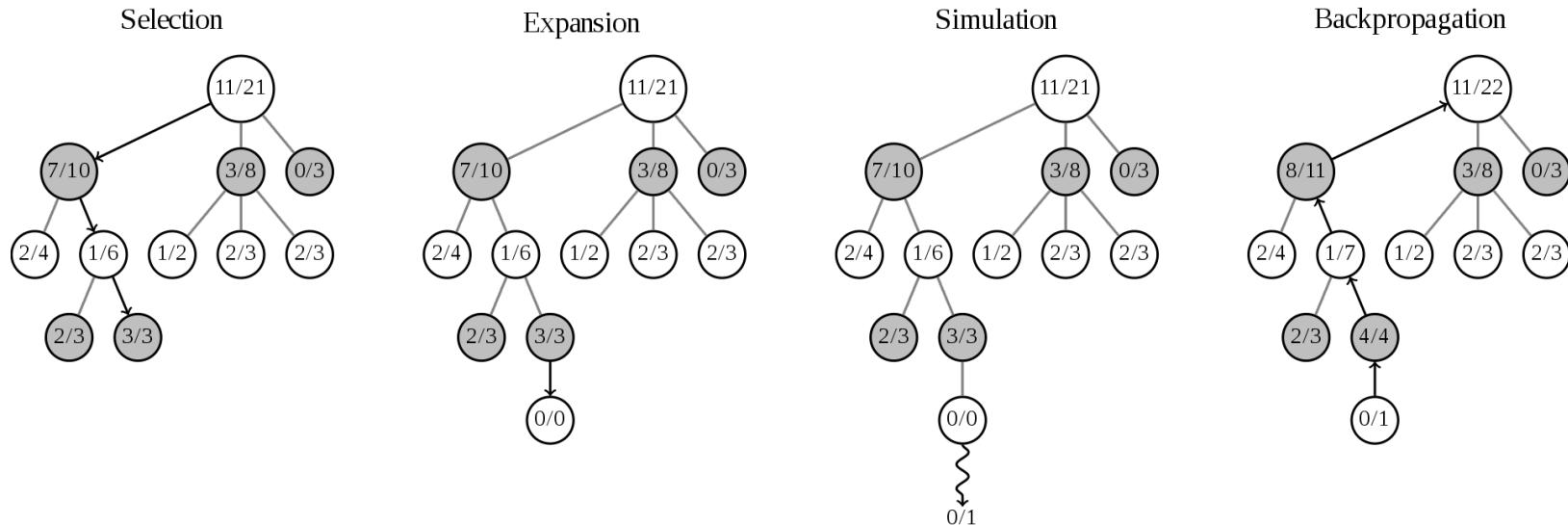
$$\begin{aligned} e^{-2nU^2} \leq \frac{1}{t^k} &\iff -2nU^2 \leq -k \log t \\ &\iff U \geq \sqrt{\frac{k}{2}} \sqrt{\frac{\log t}{n}} \end{aligned}$$

Optimistic initial values

- Set $Q_0(i)$ to be something high, rather than 0 or the mean.
- This encourages exploration in the initial stages.



Monte-Carlo tree search (MCTS)



Steps of MCTS

- (i) Selection: Starting from the root node, the search process descends down the tree by successively selecting child nodes according to the *tree policy*, the most common of which is UCB1.
- (ii) Expansion: When the simulation phase reaches a leaf node, children of the leaf node are added to the tree, and one of them is selected by UCB1.
- (iii) Simulation: One random playout, or multiple if parallel processing is employed, is performed until a terminal node is reached.
- (iv) Back-propagation: The result of the playout is computed and used to update the nodes visited in the selection phase.

- Pros:
 - Does not require an evaluation function at leaf nodes.
 - Compared to alpha-beta search, which primarily uses depth-first search, MCTS is a best-first search algorithm, and search can be terminated at any time with relatively good results.
 - More robust than minimax or alpha-beta search.
- Cons:
 - At each node, UCB1 assumes a stationary distribution for the children (actions) of the node, but more often than not the actual distribution is non-stationary.
 - Simple averaging of the rewards to determine the means of the child nodes causes convergence to the optimal action to be slow, particularly in minimax trees.

Non-stationary problems

- Give more weight to recent rewards than older rewards:

$$Q_{n+1} = Q_n + \alpha[R_{n+1} - Q_n], \quad \alpha \in (0, 1].$$

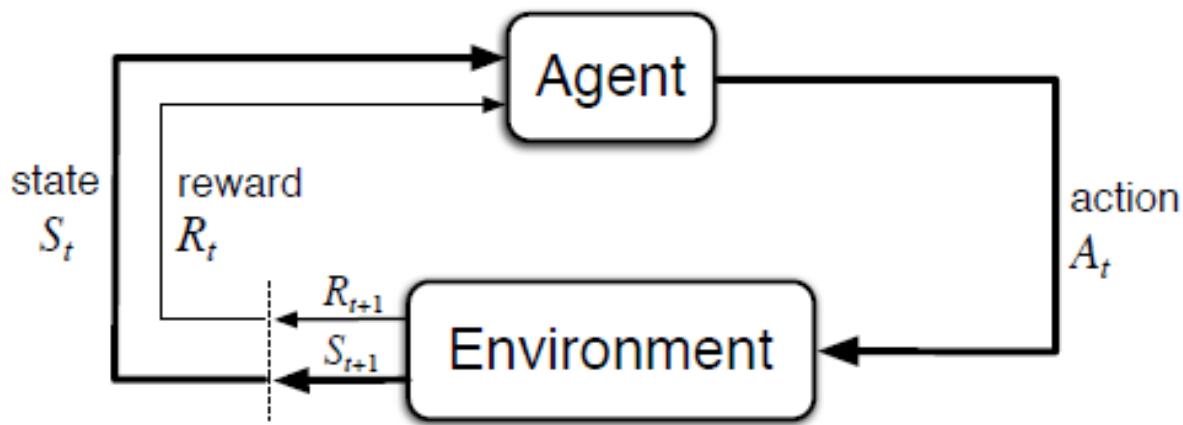
- Expanding the expression, we have

$$\begin{aligned} Q_{n+1} &= Q_n + \alpha[R_{n+1} - Q_n] \\ &= \alpha R_{n+1} + (1 - \alpha)Q_n \\ &= \alpha R_{n+1} + (1 - \alpha)[\alpha R_n + (1 - \alpha)Q_{n-1}] \\ &= \alpha R_{n+1} + (1 - \alpha)\alpha R_n + (1 - \alpha)^2 Q_{n-1} \\ &= \alpha R_{n+1} + (1 - \alpha)\alpha R_n + (1 - \alpha)^2 R_{n-1} \\ &\quad + \cdots + (1 - \alpha)^n \alpha R_1 + (1 - \alpha)^{n+1} Q_0. \end{aligned}$$

- This is also called exponential recency-weighted average.

Markov Decision Processes (MDPs)

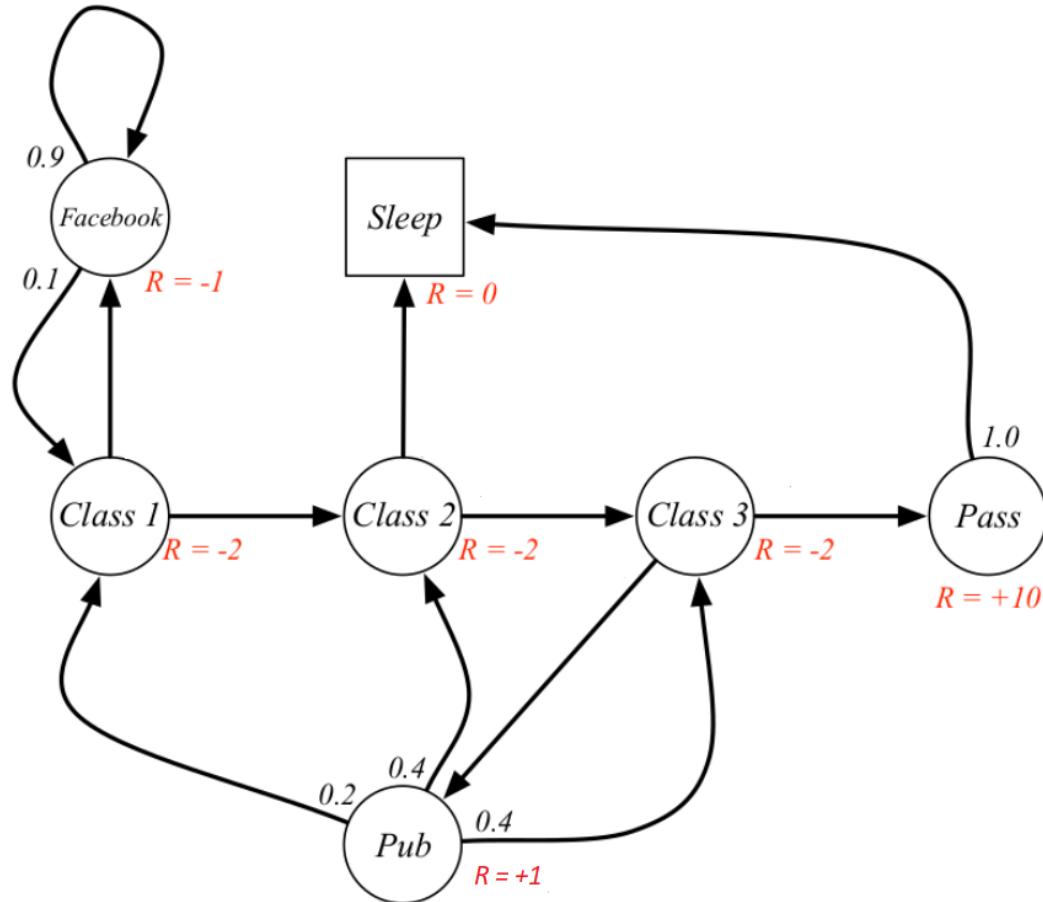
Agent-Environment interaction



Formal definition

- A Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p)$ where:
 - \mathcal{S} is the set of states
 - \mathcal{A} is the set of actions
 - \mathcal{R} is the set of rewards
 - $p(s', a|s, a) = P(S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a)$ governs the dynamics of the MDP.
- $S_t \in \mathcal{S}$, $R_t \in \mathcal{R}$ and $A_t \in \mathcal{A}$ for all t .
- $\sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a) = 1$

Student MDP



Markovity

- State-transition probabilities:

$$p(s' | s, a) = P(S_t = s' | S_{t-1} = s, A_{t-1} = a) = \sum_{r \in \mathcal{R}} p(s', r | s, a)$$

- Expected rewards:

$$r(s, a) = \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a] = \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r | s, a)$$

Objective/goal

- To maximize the expected sum of rewards $E(G_0)$:

$$G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- $\gamma \in [0, 1]$ is called the discount factor.
- For infinite episodes with bounded rewards, we must have $\gamma < 1$ so that $\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ converges.
- For finite episodes (γ can be 1, no discounting):
$$G_t = \sum_{k=0}^T \gamma^k R_{t+k+1}.$$
- $G_t = R_{t+1} + \gamma G_{t+1}.$

Policy and Value functions

- Policy function $\pi(a|s)$: the probability that $A_t = a$ given that $S_t = s$
- State-value function for policy π :
 - this is the value of a state s under policy π , i.e. the expected return when starting in s and following π thereafter
 - $v_\pi(s) := \mathbb{E}_\pi [G_t | S_t = s]$
 - By convention, for terminal states, if any, their value is 0.
- Action-value function for policy π :
 - the value of taking action a in state s under policy π (and thereafter)
 - $q_\pi(s, a) := \mathbb{E}_\pi [G_t | S_t = s, A_t = a]$

Bellman equation (wrt a policy π)

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) \left[r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s'] \right] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) \left[r + \gamma v_\pi(s') \right], \quad \text{for all } s \in \mathcal{S}, \end{aligned}$$

- v_π is the solution to its own Bellman equation
- This is a system of $|\mathcal{S}|$ linear equations in $|\mathcal{S}|$ unknowns.

Policy evaluation (prediction)

- When $|S|$ is small, we can directly solve the linear equations.
- When the state-space is large, it is more efficient to do iteration using the following update formula:

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_{\pi} [R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) [r + \gamma v_k(s')] \end{aligned}$$

Iterative policy evaluation

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π , the policy to be evaluated

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation

Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop:

$$\Delta \leftarrow 0$$

Loop for each $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until $\Delta < \theta$

Optimal policies and value functions

- $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in \mathcal{S}$.
- Optimal state-value function:

$$v_*(s) := \max_{\pi} v_{\pi}(s)$$

for all s .

- Optimal action-value function:

$$q_*(s, a) := \max_{\pi} q_{\pi}(s, a)$$

for all s, a .

Optimal Bellman equation (state-value)

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_*(s')]. \end{aligned}$$

Optimal Bellman equation (action-value)

$$\begin{aligned} q_*(s, a) &= \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a\right] \\ &= \sum_{s', r} p(s', r | s, a) [r + \gamma \max_{a'} q_*(s', a')] \end{aligned}$$

- Both optimal Bellman equations are non-linear and cannot be solved explicitly.

Policy improvement

- Given the state-value function of a policy π , how do we find a new policy π' that is better than it?
- We can adopt the greedy approach, which does a one-step look-ahead (for each state s):

$$\begin{aligned}\pi'(s) &:= \arg \max_a q_\pi(s, a) \\ &= \arg \max_a \mathbb{E} [R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \arg \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')]\end{aligned}$$

Policy iteration

- $\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$
- Here, \xrightarrow{E} denotes policy evaluation and \xrightarrow{I} denotes policy improvement.

Policy iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization

$V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation

Loop:

$$\Delta \leftarrow 0$$

Loop for each $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s)) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement

policy-stable \leftarrow true

For each $s \in \mathcal{S}$:

$$\text{old-action} \leftarrow \pi(s)$$

$$\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

If $\text{old-action} \neq \pi(s)$, then *policy-stable* \leftarrow false

If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Value iteration

- Combines policy evaluation and improvement into one step.
- Convert optimal Bellman equation into an update rule:

$$\begin{aligned}v_{k+1}(s) &:= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\&= \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')]\end{aligned}$$

Value iteration

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation

Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop:

$$\Delta \leftarrow 0$$

Loop for each $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that

$$\pi(s) = \arg \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

- Greedy policy with respect to optimal value function:

$$\pi(s) \approx \arg \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_*(s')] = \arg \max_a q_*(s,a) = \pi_*(s)$$

Final review

K-means clustering

Intended learning outcomes:

- Describe the differences between distance metrics and similarity functions.
- Describe the steps of the K-means algorithm, and understand how it is based on coordinate descent.
- Explain why it is important to run the algorithm several times with various starting points.
- Recall the elbow method for estimating the optimal number for K given the dataset.
- Understand what Voronoi regions are and how they are related to their corresponding clusters.

Information theory

Intended learning outcomes:

- Memorize the formulas for entropy, cross-entropy and KL-divergence, and be able to use them for simple computations.
- Understand these quantities in the context of optimal coding theory.
- Know what their properties are; e.g. non-symmetry, KL-divergence is always non-negative and is zero when $p = q$, etc.

EM algorithm

Intended learning outcomes:

- Be familiar with the distribution of the Gaussian mixture model.
- Describe the EM algorithm, and understand how it differs from K-means when used for clustering.
- Know what are the potential problems that may arise when using EM.
- Know what Bayesian Information Criterion is, and how it is used for model selection.
- Understand the EM algorithm in the general case, and how optimizing the lower bound is the same as optimizing the log-likelihood.

PCA

Intended learning outcomes:

- Understand that PCA is a form of dimensionality reduction.
- Know the steps of PCA, and the role singular value decomposition (SVD), $X = U\Sigma V^T$, plays in it.
- Understand the significance of the transformation $T = XV = U\Sigma$.

Hidden Markov Models

Intended learning outcomes:

- Write down the joint-probability distribution of HMM in terms of the initial, transition and emission probabilities.
- Know that EM algorithm is used to learn the parameters.
- Understand the forward-backward algorithm, and how it is used to solve the evaluation problem in HMMs, as well as to compute the conditional probabilities in the E-step when learning the parameters.
- Know that the Viterbi algorithm is used to solve the decoding problem in HMMs.
- Understand dynamic programming in general, and how it is used in the forward-backward and Viterbi algorithms.

Multi-armed bandit problem

Intended learning outcomes:

- Understand the exploration-exploitation trade-off in reinforcement learning.
- Know what are action-values, and how to incrementally update their estimates.
- Know what are the greedy, ϵ -greedy and UCB policies.
- Understand how Monte-Carlo tree search (MCTS) works (each of its four phases per iteration), and how it compares to minimax search.
- Understand how the exponential recency-weighted average update allows one to handle non-stationary cases.

Markov decision processes

Intended learning outcomes:

- Understand what a Markov decision process is.
- Know what policy, state-value and action-value functions are.
- Understand the Bellman equations for the value functions under a policy, and how it can be solved directly as a linear system of equations for small state spaces.
- Understand the optimal Bellman equations which the optimal value function must satisfy, and how they are non-linear.
- Understand how to iteratively converge to the solution of the Bellman equation for policy evaluation, and how to use this approximate solution for policy improvement.
- Understand policy iteration and value iteration.