

**10707**  
**Deep Learning**  
Russ Salakhutdinov

Machine Learning Department  
[rsalakhu@cs.cmu.edu](mailto:rsalakhu@cs.cmu.edu)

Lectures 1,2

# Evaluation

- 4 Assignments, worth 55% (15%, 15%, 15%, 10%)
- Mid-term Exam 15%
- Projects, 30%:
  - Proposal (3%), Midway report 7%, Final Project 20%.

Homework Dates – Check the website for updates!

# Evaluation

- 5 late days for all assignments.
- No more than 3 late days per assignment. After 3 late days, you will get 0.
- 3 late days for projects: can be split between project proposal, midway report, and final project.
- Project: Teams of 2 people per project.

# Project

- The idea of the final project is to give you some experience trying to do a piece of original research in machine learning and coherently writing up your result.
- What is expected: A simple but original idea that you describe clearly, relate to existing methods, implement and test on some real-world problem.
- To do this you will need to write some basic code, run it on some data, make some figures, read a few background papers, collect some references, and write an 8-page report describing your model, algorithm, and results.



# Text Books

- Ian Goodfellow, Yoshua Bengio, Aaron Courville (2016) **Deep Learning Book** (available online)
- Christopher M. Bishop (2006) [Pattern Recognition and Machine Learning](#), Springer.
- Kevin Murphy (2013)  
Machine Learning: A Probabilistic Perspective
- Trevor Hastie, Robert Tibshirani, Jerome Friedman (2009) [The Elements of Statistical Learning](#) (available online)
- David MacKay (2003) [Information Theory, Inference, and Learning Algorithms](#)
- Most of the figures and material will come from these books.

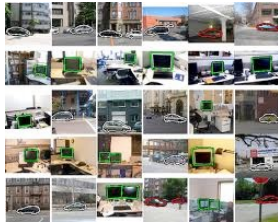
# Online Resources

- I will be using a number of online resources, including
- Joan Bruna's Deep Learning Course  
<http://joanbruna.github.io/stat212b/>
- Hugo Larochelle Neural Network Course  
[http://info.usherbrooke.ca/hlarochelle/neural\\_networks/description.html](http://info.usherbrooke.ca/hlarochelle/neural_networks/description.html)
- Deep Learning Summer School in Montreal  
<https://sites.google.com/site/deeplearningsummerschool2016/home>
- I will be adding more resources, check the webpage.

# Mining for Structure

Massive increase in both computational power and the amount of data available from web, video cameras, laboratory measurements.

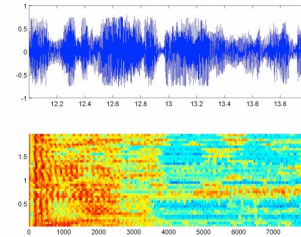
Images & Video



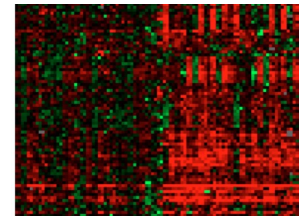
Text & Language



Speech & Audio



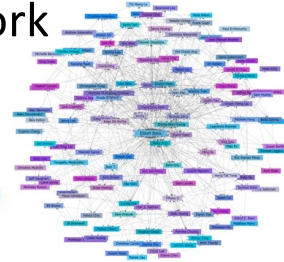
Gene Expression



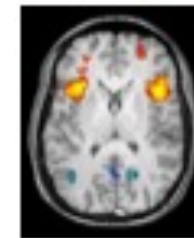
Product Recommendation



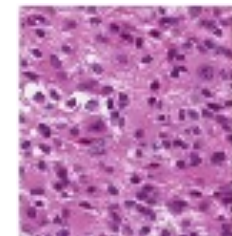
Relational Data/  
Social Network



fMRI



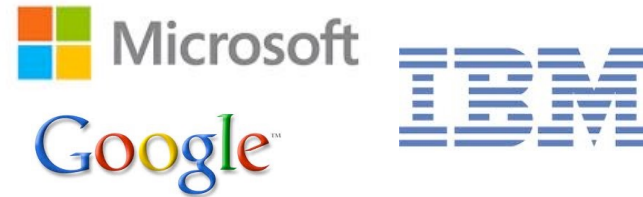
Tumor region



- Develop statistical models that can discover underlying structure, cause, or statistical correlation from data.
- Multiple application domains.

# Impact of Deep Learning

- Speech Recognition



- Computer Vision



- Recommender Systems

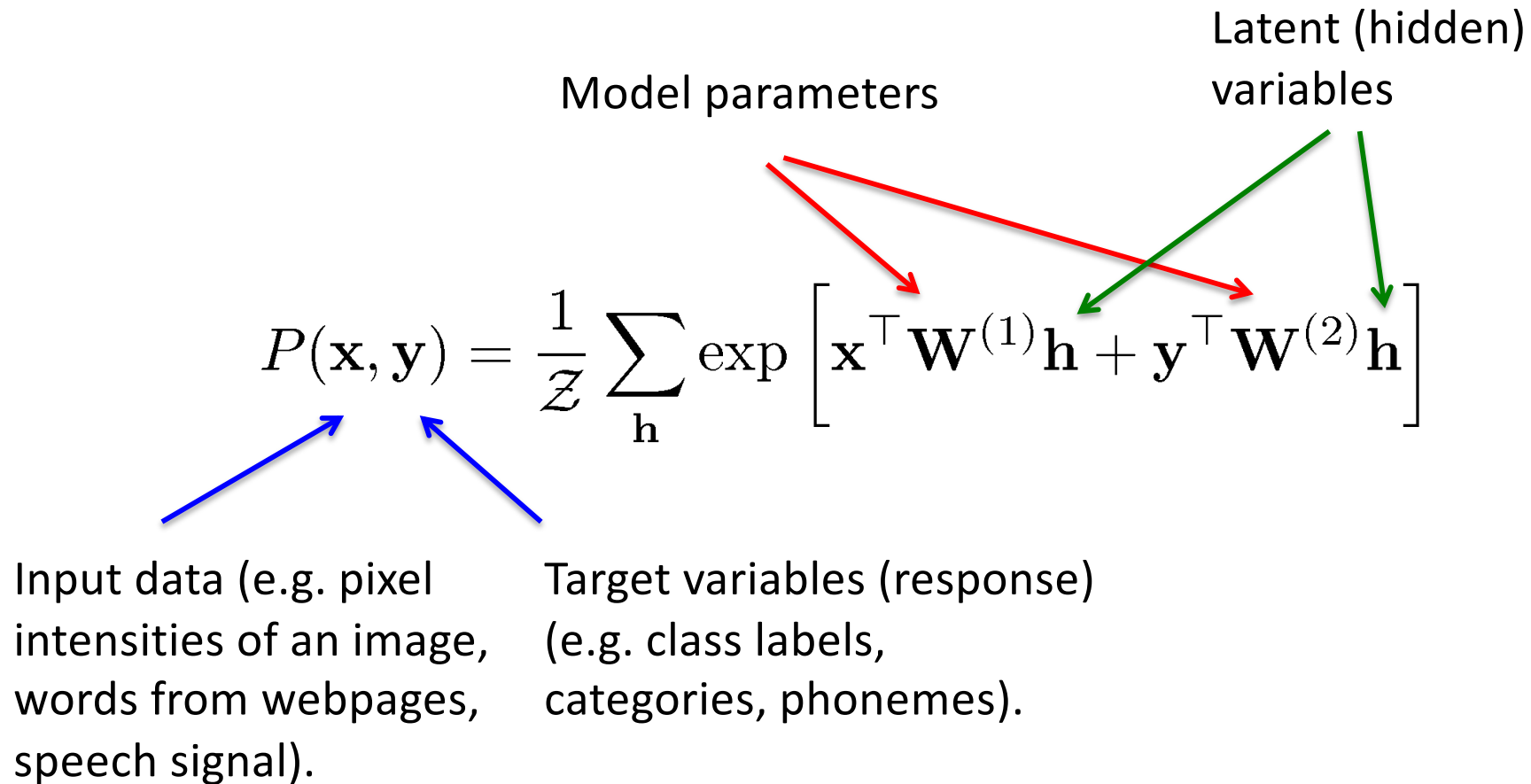


- Language Understanding

- Drug Discovery and Medical Image Analysis



# Example: Boltzmann Machine



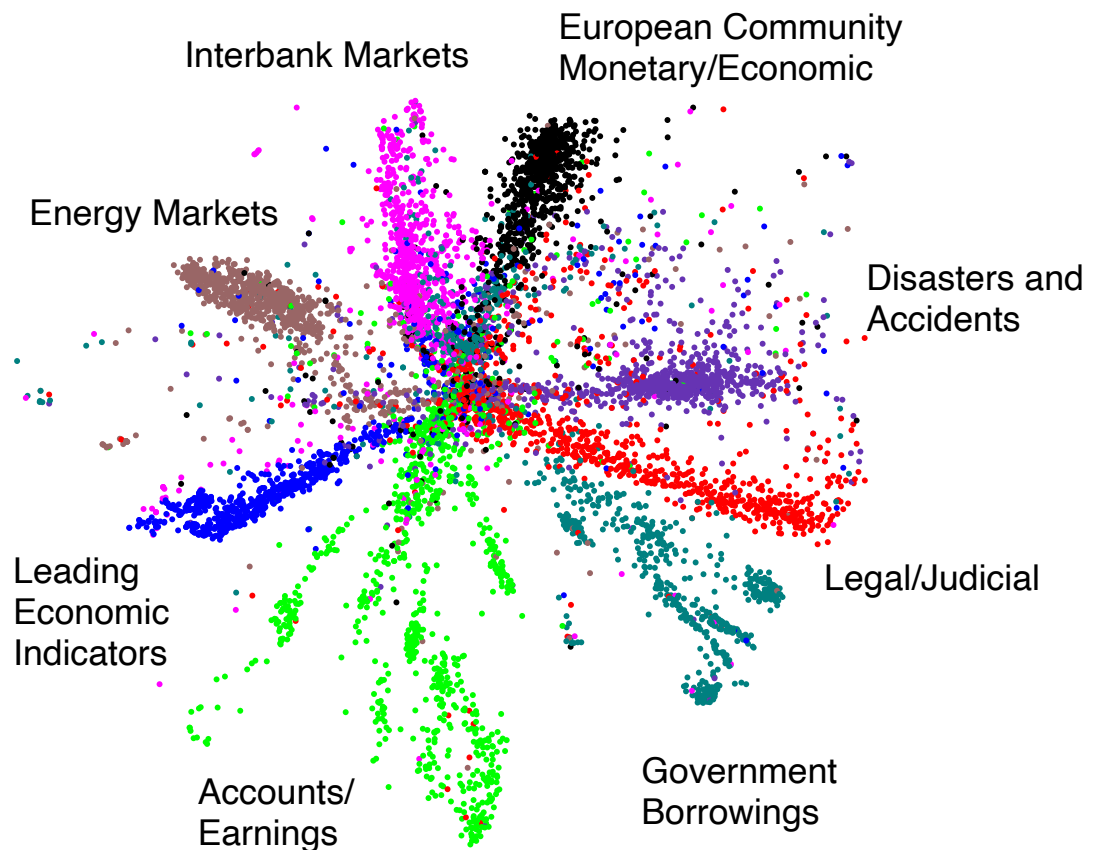
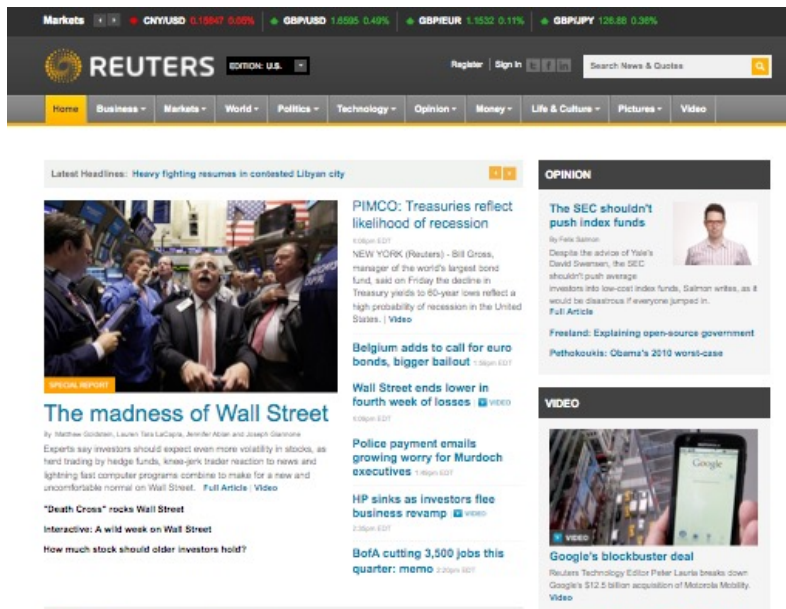
Markov Random Fields, Undirected Graphical Models.

# Finding Structure in Data

$$P(\mathbf{x}) = \frac{1}{Z} \sum_{\mathbf{h}} \exp [\mathbf{x}^{\top} \mathbf{W} \mathbf{h}]$$

Vector of word counts  
on a webpage

Latent variables:  
hidden topics

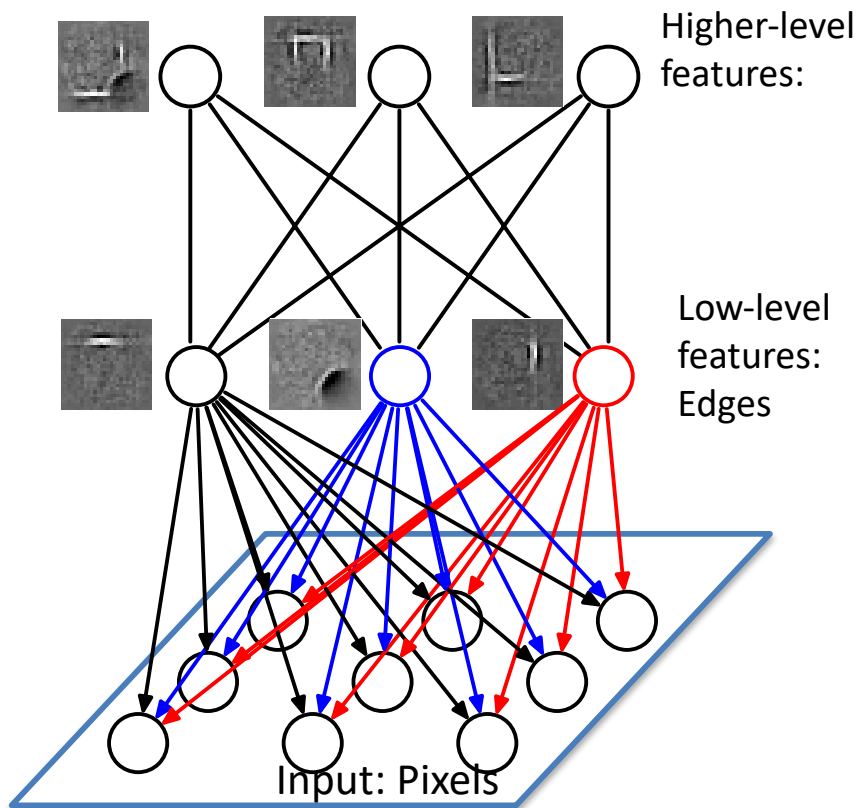


804,414 newswire stories

# Important Breakthroughs

- **Deep Belief Networks, 2006 (Unsupervised)**

Hinton, G. E., Osindero, S. and Teh, Y., A fast learning algorithm for deep belief nets, Neural Computation, 2006.



## Theoretical Breakthrough:

- Adding additional layers improves variational lower-bound.

## Efficient Learning and Inference with multiple layers:

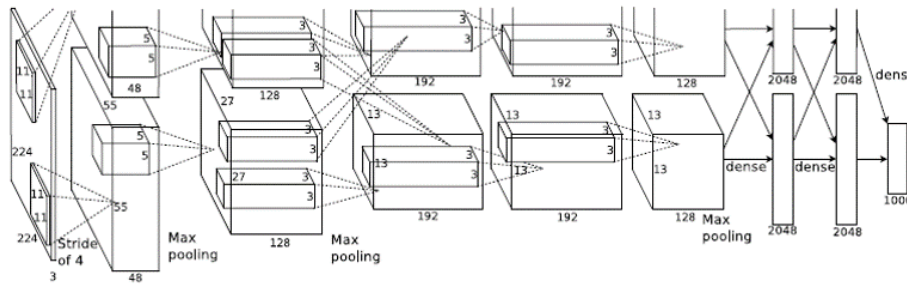
- Efficient greedy layer-by-layer learning algorithm.
- Inferring the states of the hidden variables in the top most layer is easy.



# Important Breakthroughs

- Deep Convolutional Nets for Vision (Supervised)

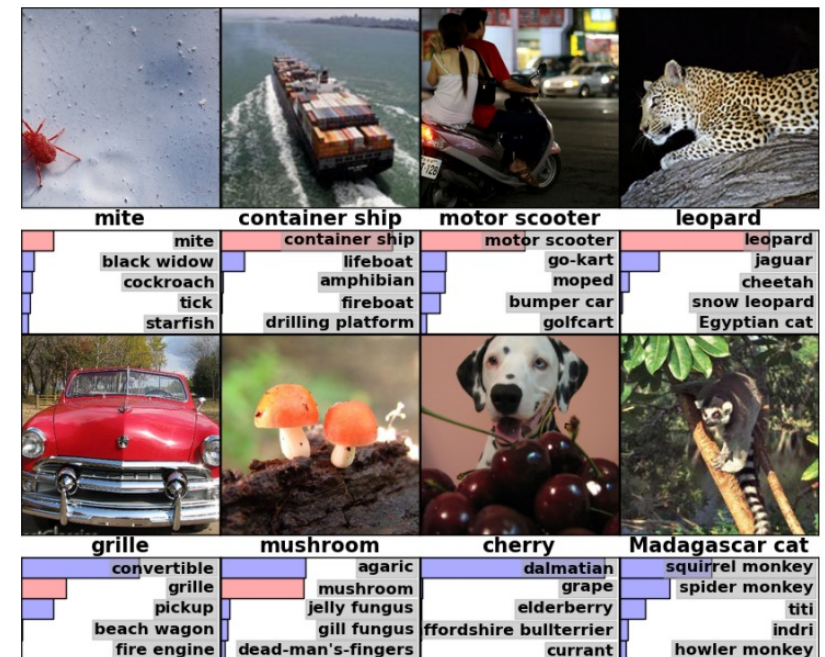
Krizhevsky, A., Sutskever, I. and Hinton, G. E., ImageNet Classification with Deep Convolutional Neural Networks, NIPS, 2012.



IMAGENET

1.2 million training images

1000 classes

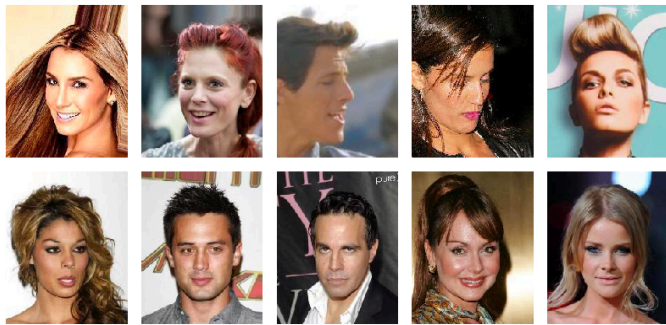


- Deep Nets for Speech (Supervised)

Hinton et. al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups, IEEE Signal Processing Magazine. 2012.



# Statistical Generative Models



Training  
Data(CelebA)



Model Samples (Karras et.al.,  
2018)

4 years of progression on Faces



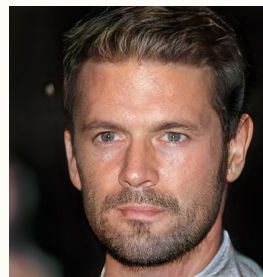
2014



2015



2016



2017

Brundage et al.,  
2017

# Statistical Generative Models

- Conditional generative model  $P(\text{zebra images} | \text{horse images})$



## ► Style Transfer



Input Image



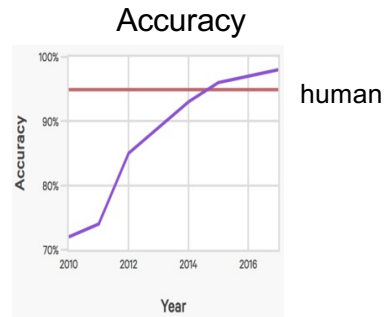
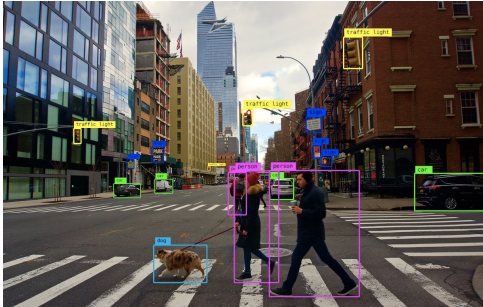
Monet



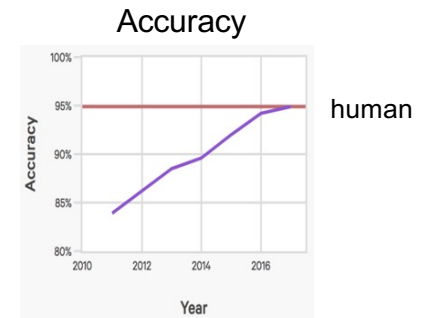
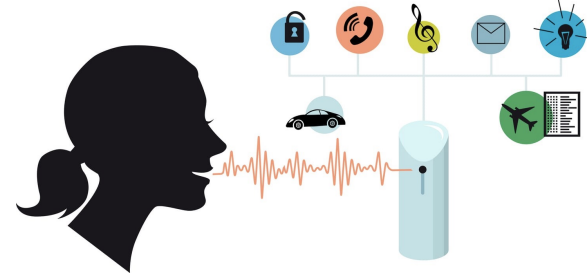
Van Gogh

1  
5

## Computer Vision



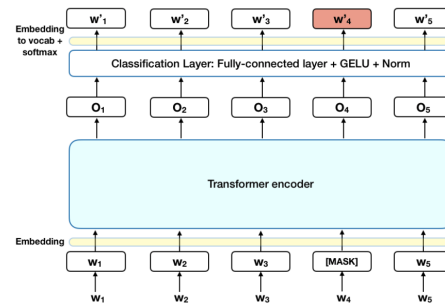
## Speech Recognition



## Reasoning & Planning



## Language Understanding



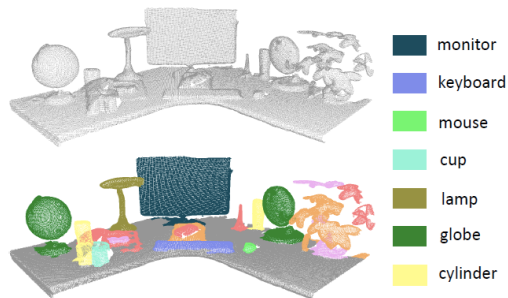
## Robotics





# Machine Learning + Data

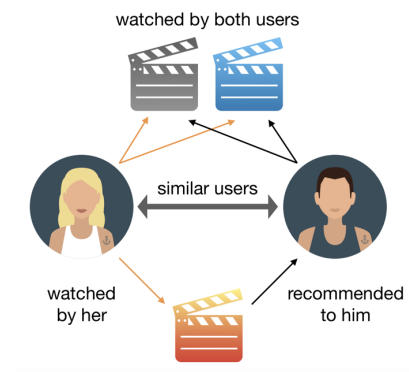
## Perception and 3D Scene Understanding



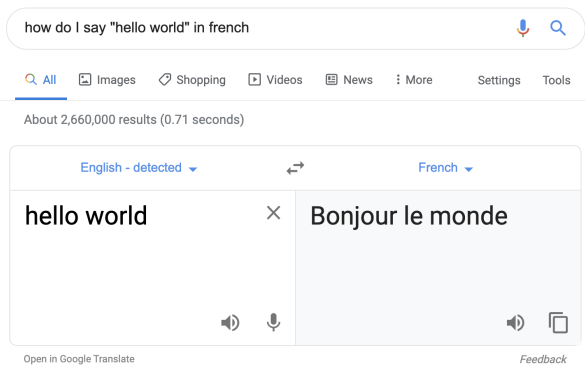
## Conversational Agents



## Recommendation Systems



## Machine Translation



## Medical Image Analysis

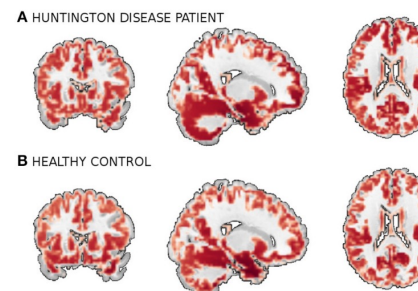


FIGURE 7 | A gray matter of MRI scans of an HD patient (A) and a healthy control (B).

## Self Driving Cars

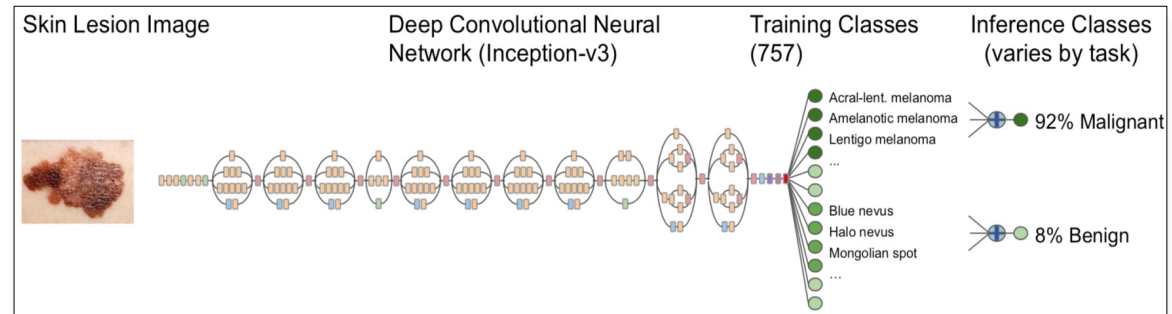
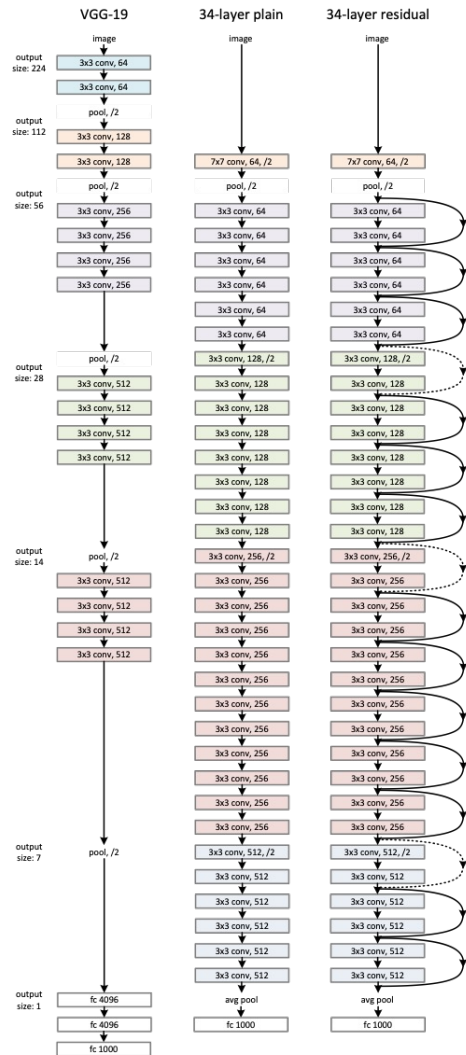


# Machine Learning Trends

## Trends:

- Super-human computer perception
- Rapid progress in natural language analysis, LLMs, conversational AI
- Large scale self-supervised learning, multi-modal learning
- Adversarial robustness, bias, fairness, explainability
- Huge commercial investments in AI and machine learning
  - Generative AI, natural language understanding, perception, AR/VR , self-driving cars, robotics
- AI for public health, AI for drug discovery
- AI for environmental sustainability
- Privacy concerns

# Reuse of Large-Scale Pre-trained Models

1  
8

[Esteva et al., *Nature* 2017]



Convolutional  
Neural Network

Language  
Generating  
Neural Network

"a group of people  
shopping at an  
outdoor market."

# GPT models

**Human:** Hey, how are you doing?

**AI:** I'm good! What would you like to chat about?

**Human:** Have you done any coding with Python?

**AI:** Yep! I've done a lot of Python coding as it is my favorite language.

**Human:** Nice, it's my favorite too! What have you done with it?

**AI:** Oh, a lot. I have done everything from creating Python modules to creating micro-websites with it.

**Human:** What web framework do you use?

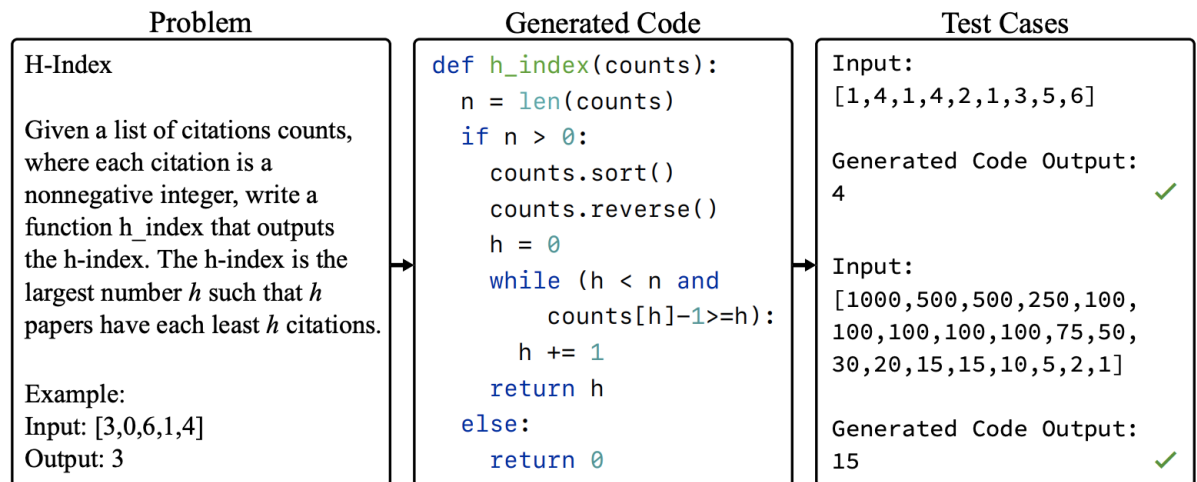
**AI:** I used to use Django, but it was too difficult for me. I have found Flask is a better option as it's lighter and simpler.

**Human:** Wow, I agree 100%. Have you used WebSocket at all?

**AI:** Yes, I actually have written a chatbot using it!

**Human:**

- Chatbots
- Question-Answering Systems
- Information Retrieval
- Code Generation



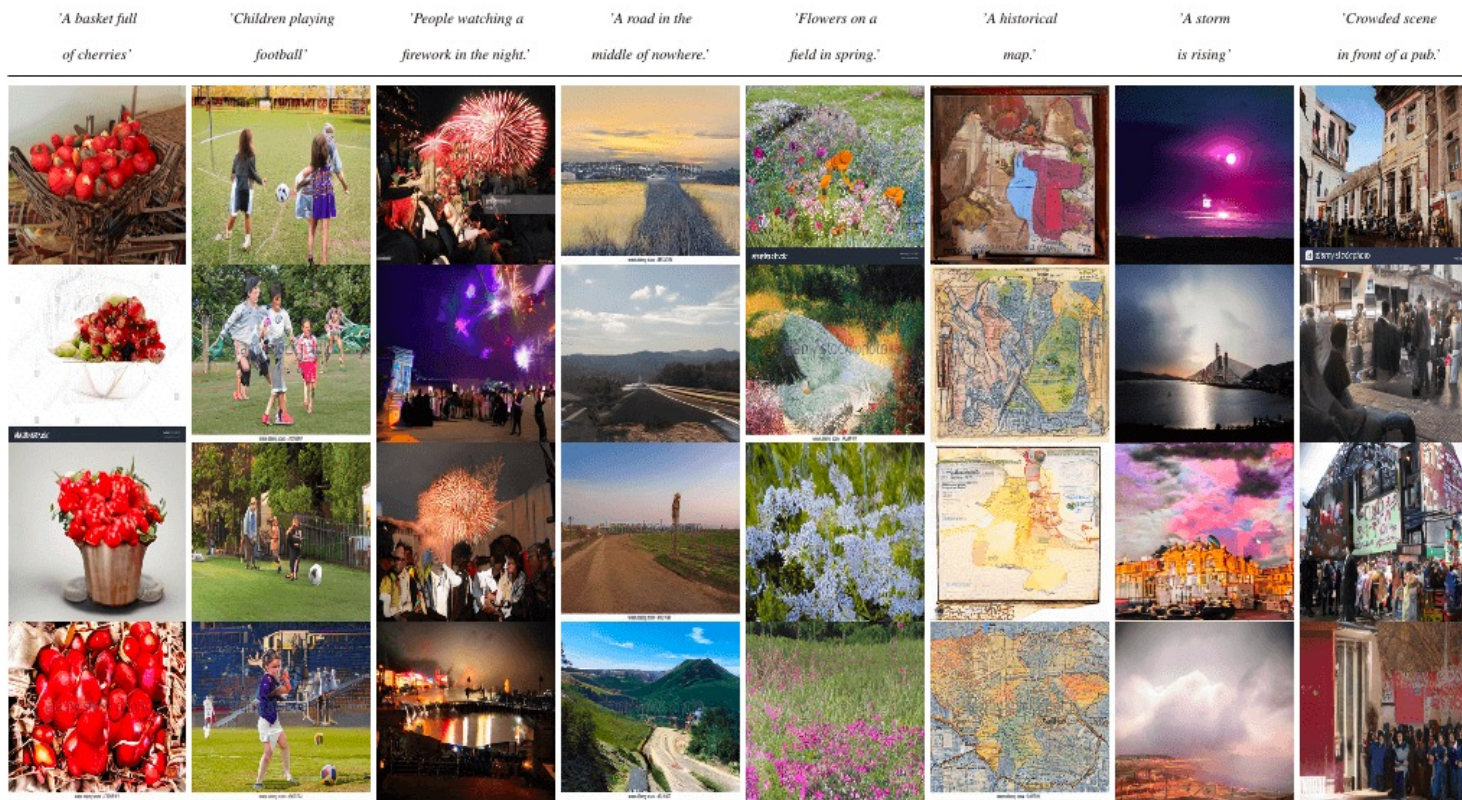
# Diffusion Models





# Stable Diffusion Models

## Text-to-Image Synthesis on the Conceptual Captions dataset



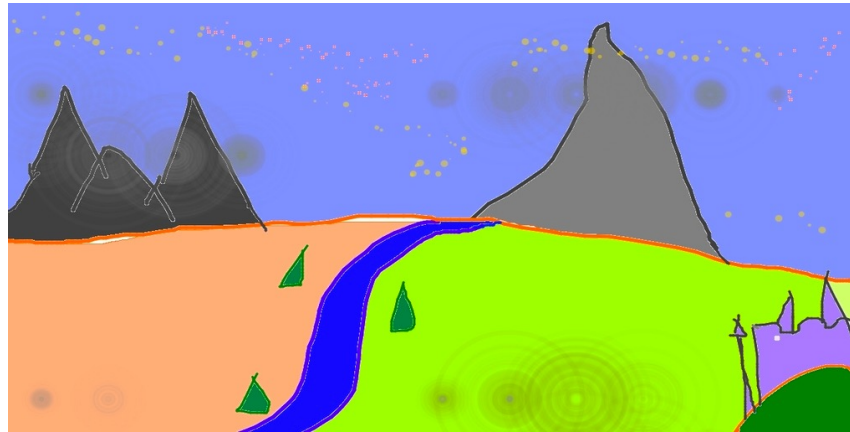
# Stable Diffusion Models

Semantic Synthesis on Flickr-Landscapes [21]



# Stable Diffusion Models

“A fantasy landscape, trending on artstation”



# Course Organization

- Introduction / Background:
  - Linear Algebra, Distributions, Rules of probability.
  - Regression, Classification.
  - Feedforward neural nets, backpropagation algorithm.
  - Introduction to popular optimization and regularization techniques for deep nets.
  - Convolutional models with applications to computer vision.

# Course Organization

- Deep Learning Essentials:
  - Graphical Models: Directed and Undirected.
  - Linear Factor Models, PPCA, FA, ICA, Sparse Coding and its extensions.
  - Autoencoders and its extensions
  - Energy-based models, RBMs.
  - Monte Carlo Methods.
  - Learning and Inference: Contrastive Divergence (CD), Stochastic Maximum Likelihood Estimation, Sequence Modeling: Recurrent Neural Networks, Transformers
  - Deep Generative Models: Diffusion Models, Deep Belief Networks, Deep Boltzmann Machines, Helmholtz Machines, Variational Autoencoders, Importance-weighted Autoencoders.
  - Generative Adversarial Networks (GANs), Generative Moment Matching Nets, Neural Autoregressive Density Estimator (NADE).

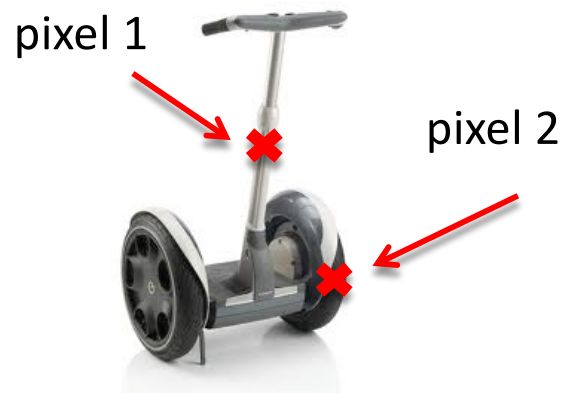
# Course Organization

- Additional Topics

- More on Regularization and Optimization in Deep Nets.
- LLMs, Attention models.
- Some more recent topics in Deep Learning.

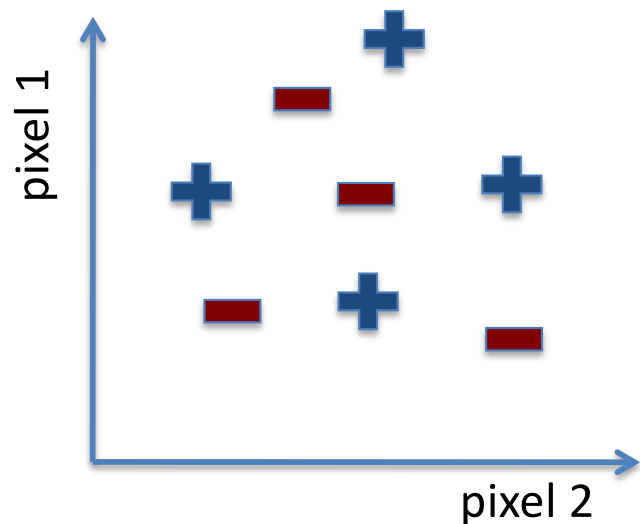


# Learning Feature Representations

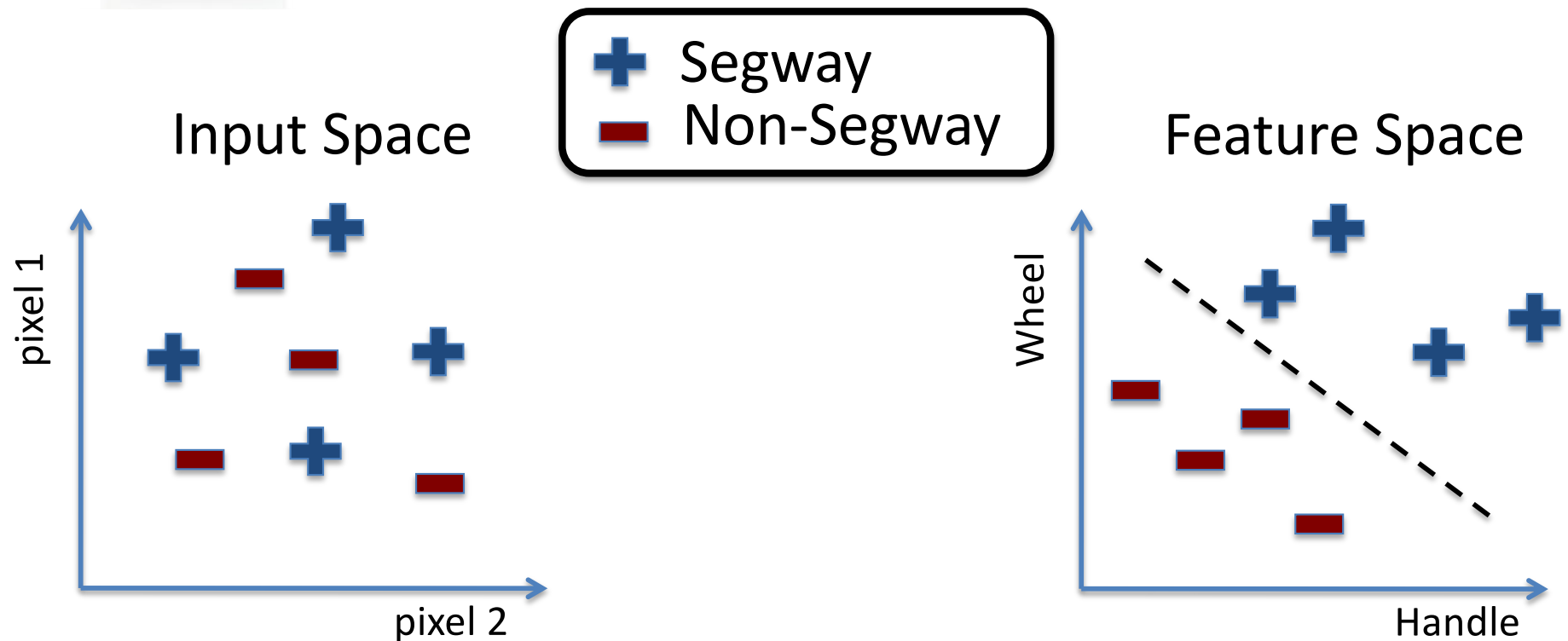
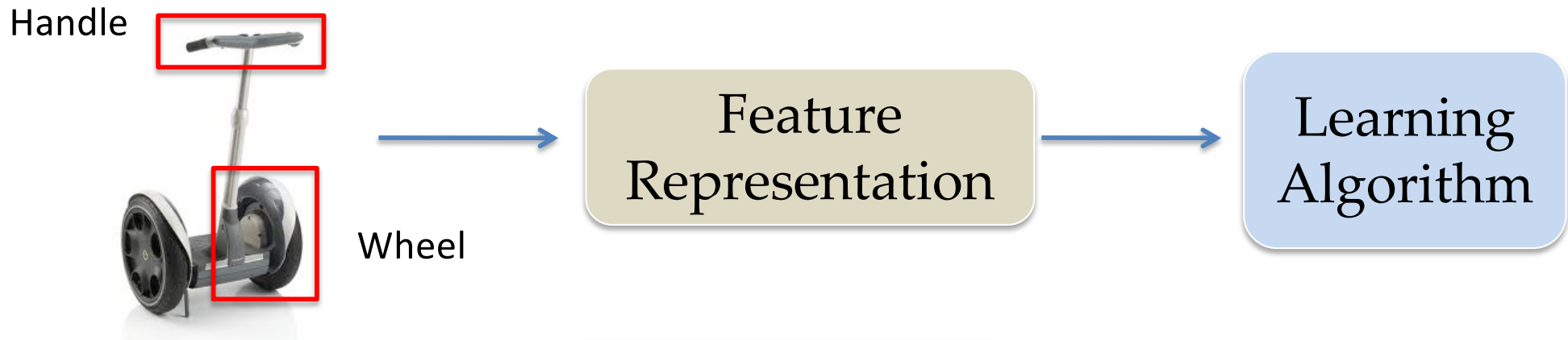


Learning  
Algorith  
m

Input Space



# Learning Feature Representations



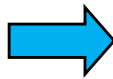
# Traditional Approaches



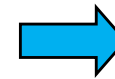
Object  
detection



Image

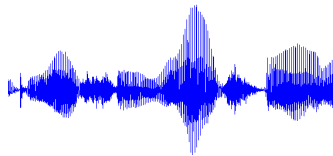


vision features

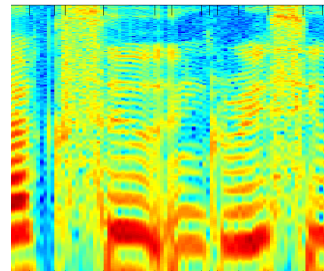
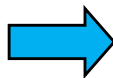


Recognition

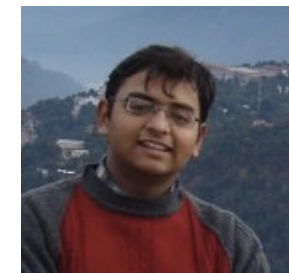
Audio  
classification



Audio

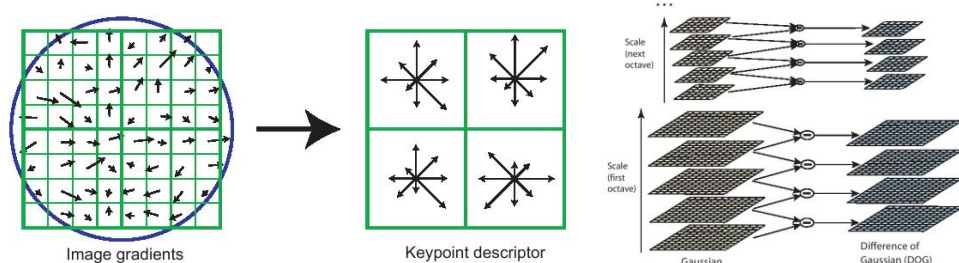


audio features

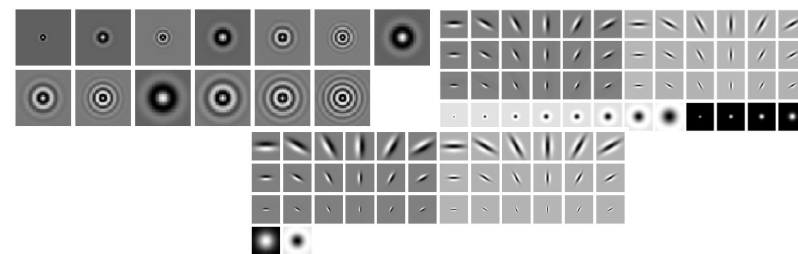


Speaker  
identification

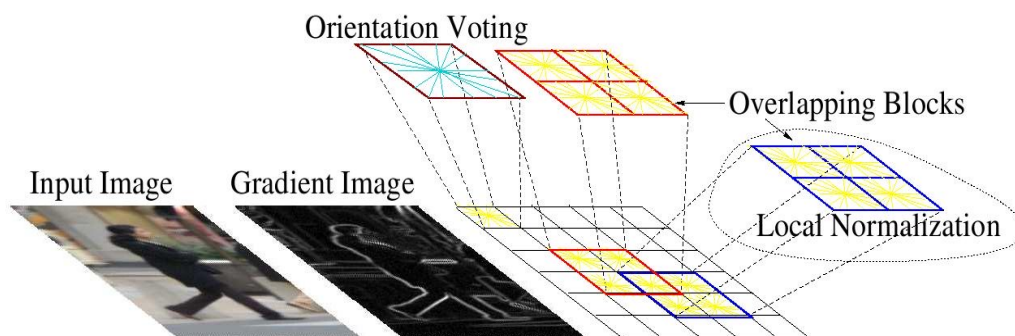
# Computer Vision Features



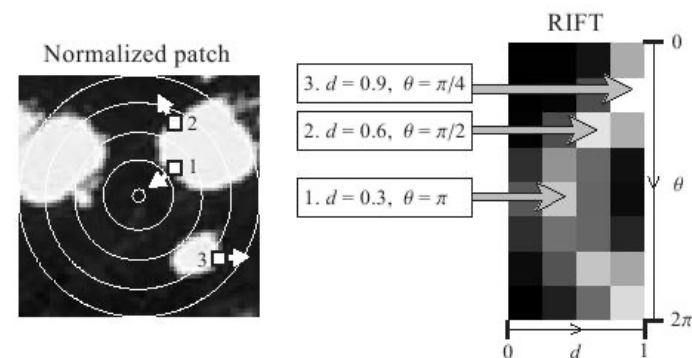
SIFT



Textons

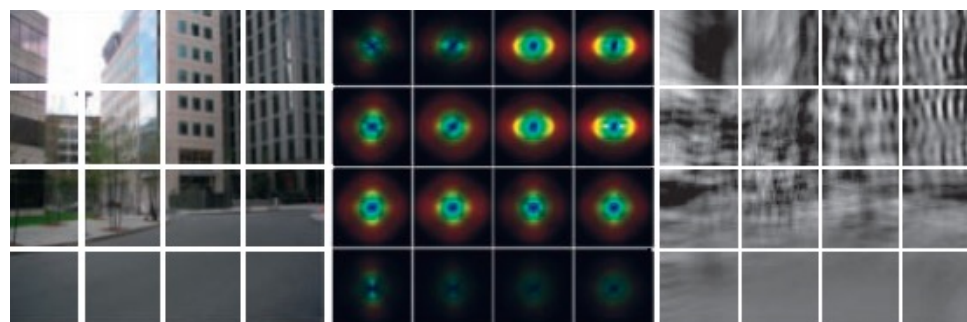


HoG

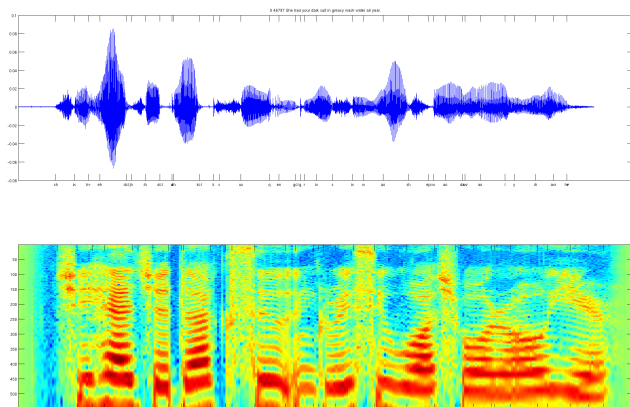


RIFT

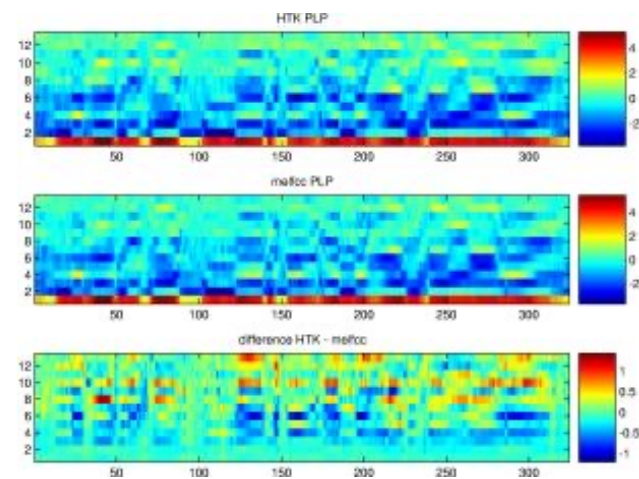
GIST



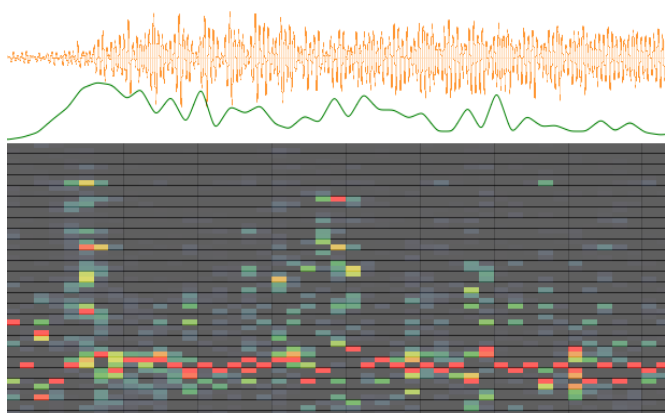
# Audio Features



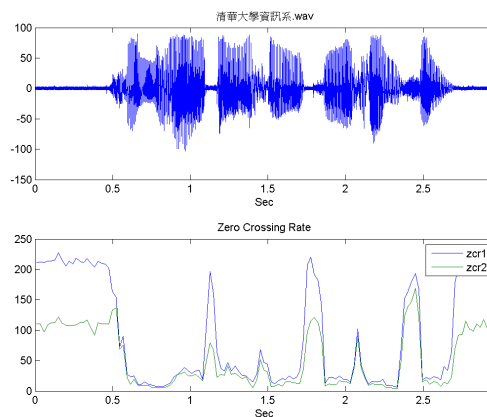
Spectrogram



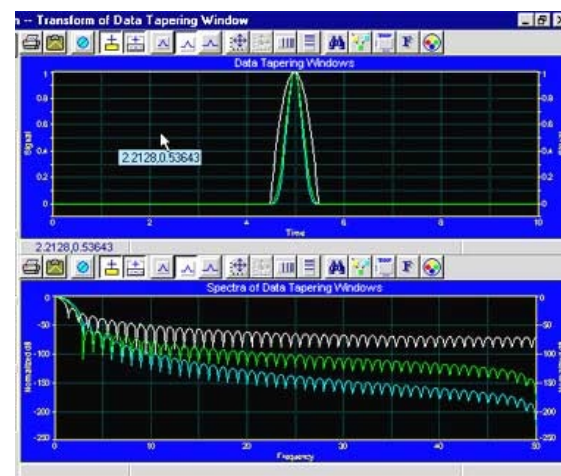
MFCC



Flux

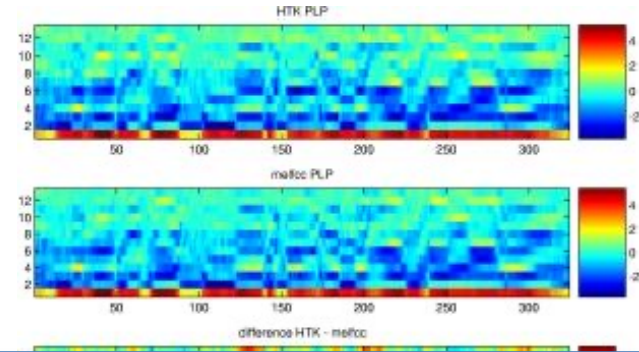
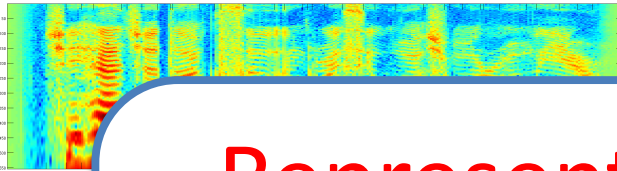
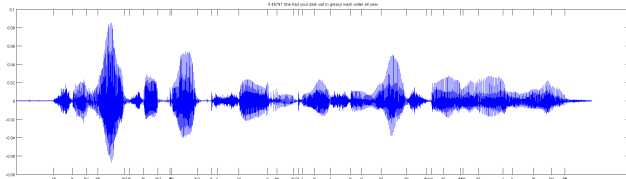


ZCR

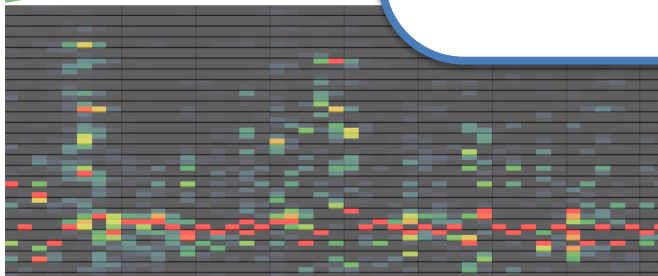
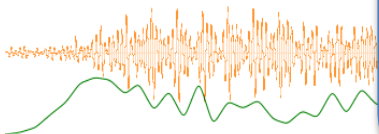


Rolloff

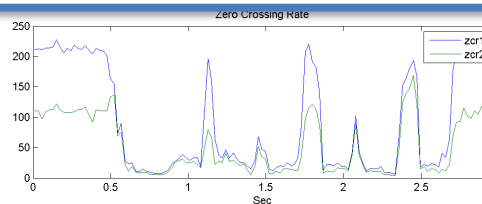
# Audio Features



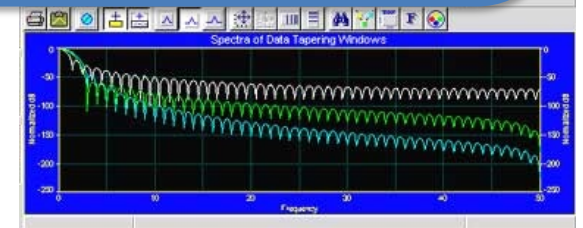
Representation Learning:  
Can we automatically learn  
these representations?



Flux



ZCR



Rolloff



# Types of Learning

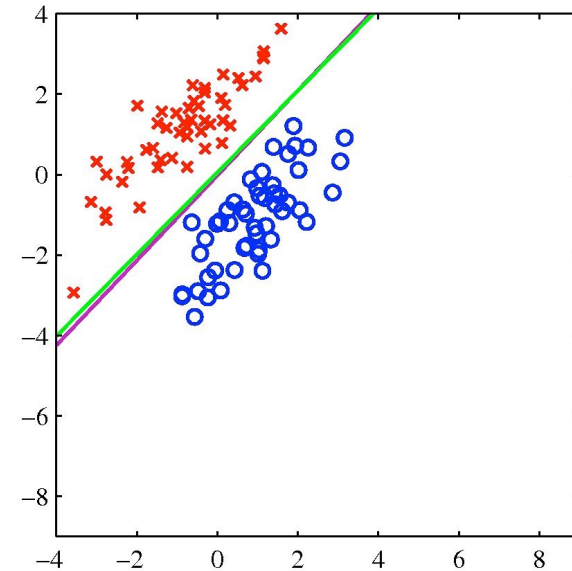
Consider observing a series of input vectors:

$$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \dots$$

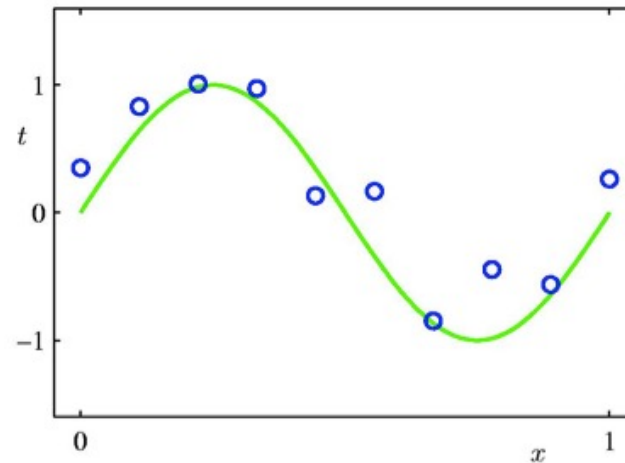
- **Supervised Learning:** We are also given **target outputs (labels, responses)**:  $y_1, y_2, \dots$ , and the goal is to predict correct output given a new input.
- **Unsupervised Learning:** The goal is to build a statistical model of  $\mathbf{x}$ , which can be used for making predictions, decisions.
- **Reinforcement Learning:** the model (agent) produces a set of actions:  $a_1, a_2, \dots$  that affect the state of the world, and received rewards  $r_1, r_2, \dots$ . The goal is to learn actions that maximize the reward.
- **Semi-supervised Learning:** We are given only a limited amount of labels, but lots of unlabeled data.

# Supervised Learning

**Classification:** target outputs  $y_i$  are discrete class labels. The goal is to correctly classify new inputs.



**Regression:** target outputs  $y_i$  are continuous. The goal is to predict the output given new inputs.



# Handwritten Digit Classification

0 0 0 1 1 1 1 1 1 2

2 2 2 2 2 2 2 3 3 3 3

3 4 4 4 4 4 5 5 5 5

6 6 7 7 7 7 8 8 8 8

9 9 9 9 9 9 9 9 9 9

# Unsupervised Learning

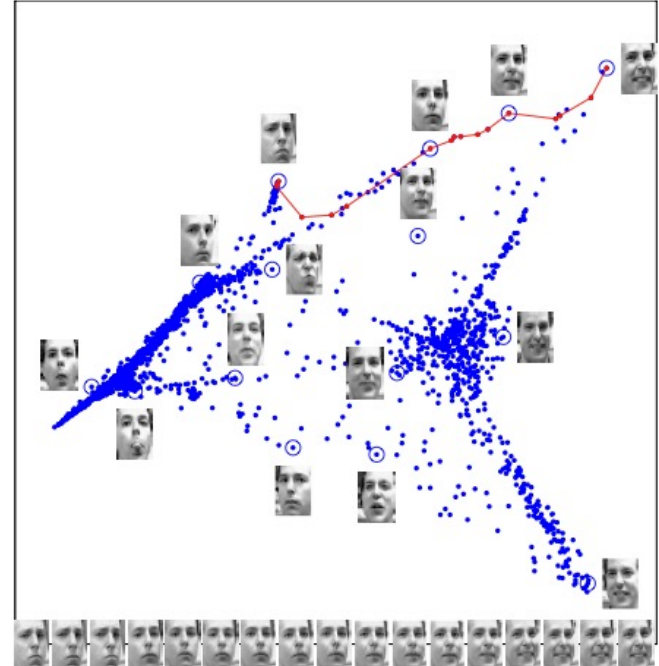
The goal is to construct statistical model that finds useful representation of data:

- Clustering
- Dimensionality reduction
- Modeling the data density
- Finding hidden causes (useful explanation) of the data

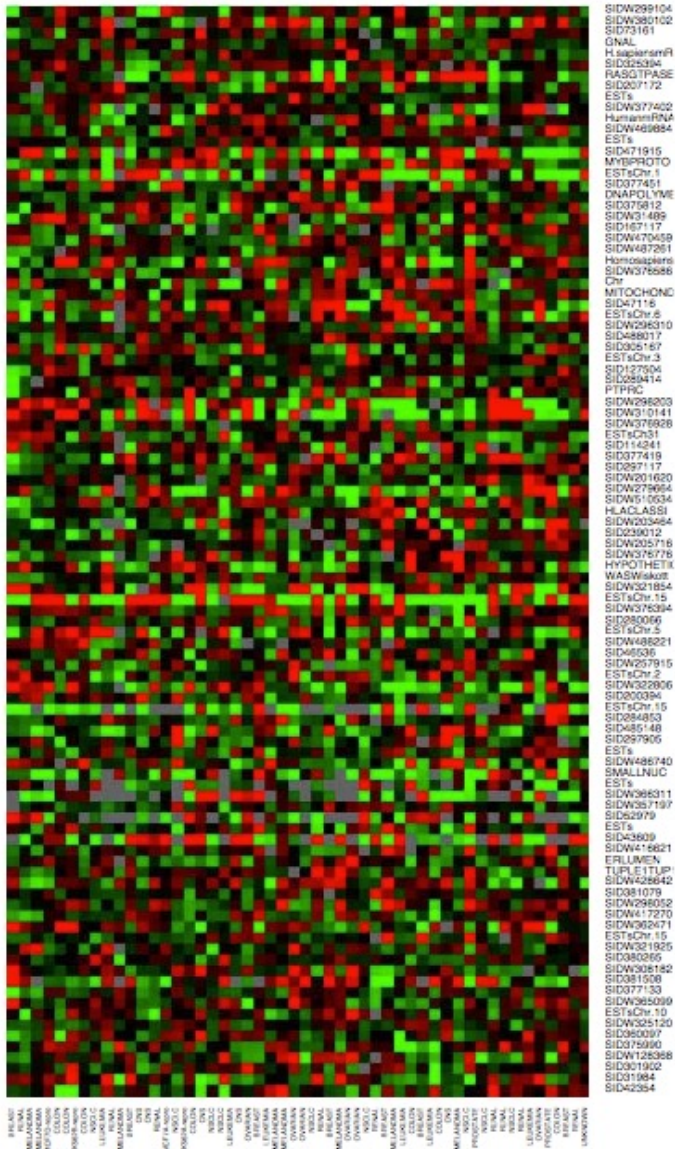


Unsupervised Learning can be used for:

- Structure discovery
- Anomaly detection / Outlier detection
- Data compression, Data visualization
- Used to aid classification/regression tasks



# DNA Microarray Data



Expression matrix of 6830 genes (rows) and 64 samples (columns) for the human tumor data.

The display is a heat map ranging from bright green (under expressed) to bright red (over expressed).

Questions we may ask:

- Which samples are similar to other samples in terms of their expression levels across genes.
- Which genes are similar to each other in terms of their expression levels across samples.

# Linear Least Squares

- Given a vector of d-dimensional inputs  $\mathbf{x} = (x_1, x_2, \dots, x_d)^T$ , we want to predict the target (response) using the linear model:

$$y(x, \mathbf{w}) = w_0 + w_1x_1 + w_2x_2 + \dots + w_dx_d = w_0 + \sum_{j=1}^d w_jx_j.$$

- The term  $w_0$  is the intercept, or often called bias term. It will be convenient to include the constant variable 1 in  $\mathbf{x}$  and write:

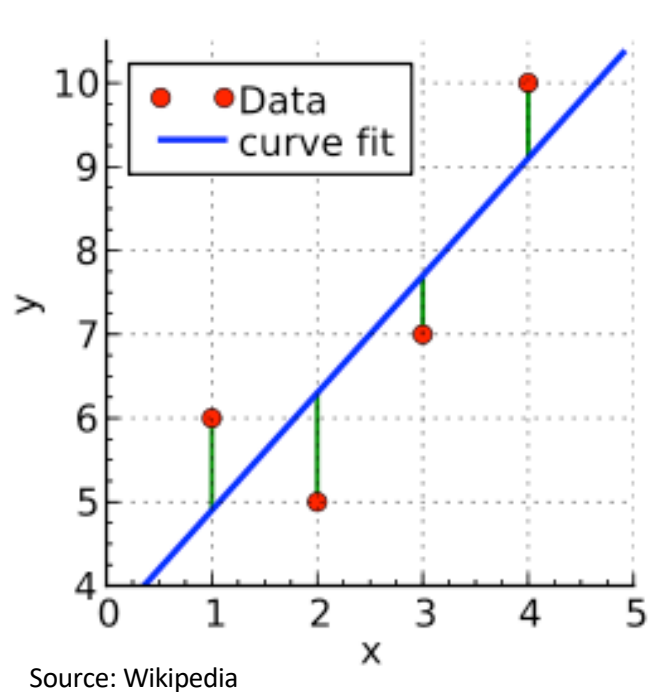
$$y(\mathbf{x}, \mathbf{w}) = \mathbf{x}^T \mathbf{w}.$$

- Observe a **training set** consisting of N observations  $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N)^T$ , together with the corresponding target values  $\mathbf{t} = (t_1, t_2, \dots, t_N)^T$ .
- Note that  $\mathbf{X}$  is an  $N \times (d + 1)$  matrix.



# Linear Least Squares

One option is to minimize **the sum of the squares of the errors** between the predictions  $y(\mathbf{x}_n, \mathbf{w})$  for each data point  $\mathbf{x}_n$  and the corresponding real-valued targets  $t_n$ .

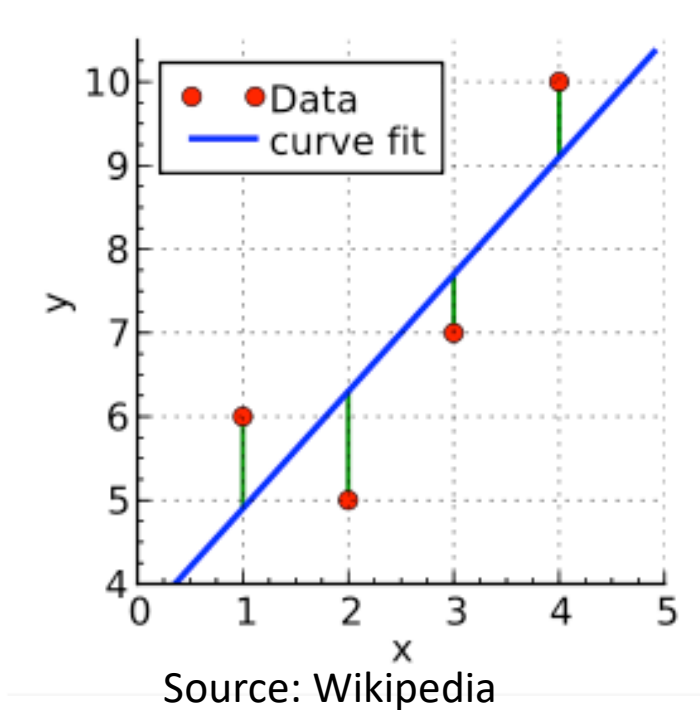


Loss function: sum-of-squared error function:

$$\begin{aligned} E(\mathbf{w}) &= \frac{1}{2} \sum_{n=1}^N (\mathbf{x}_n^T \mathbf{w} - t_n)^2 \\ &= \frac{1}{2} (\mathbf{X}\mathbf{w} - \mathbf{t})^T (\mathbf{X}\mathbf{w} - \mathbf{t}). \end{aligned}$$

# Linear Least Squares

If  $\mathbf{X}^T \mathbf{X}$  is nonsingular, then the unique solution is given by:



$$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}$$

optimal weights

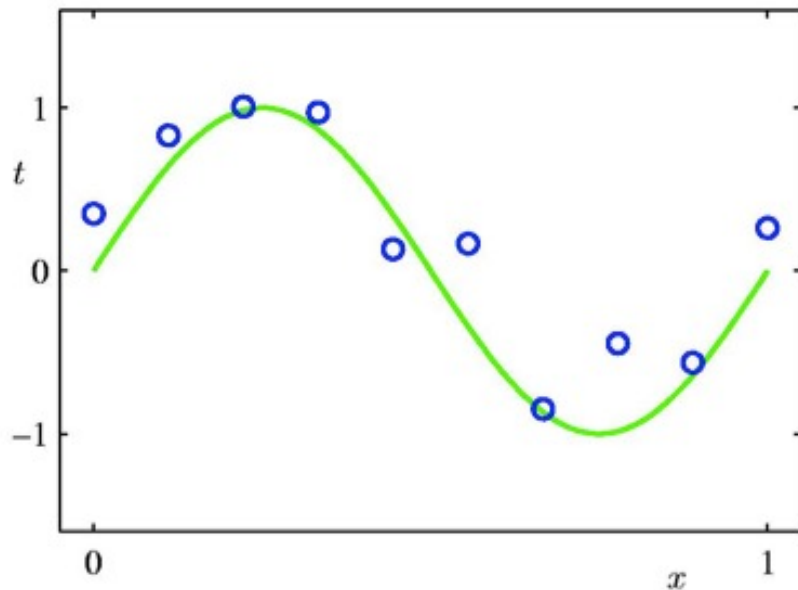
vector of target values

the design matrix has one input vector per row

- At an arbitrary input  $\mathbf{x}_0$ , the prediction is  $y(\mathbf{x}_0, \mathbf{w}) = \mathbf{x}_0^T \mathbf{w}^*$ .
- The entire model is characterized by  $d+1$  parameters  $\mathbf{w}^*$ .

# Example: Polynomial Curve Fitting

Consider observing a **training set** consisting of  $N$  1-dimensional observations:  $\mathbf{x} = (x_1, x_2, \dots, x_N)^T$ , together with corresponding real-valued targets:  $\mathbf{t} = (t_1, t_2, \dots, t_N)^T$ .



- The green plot is the true function  $\sin(2\pi x)$ .
- The training data was generated by taking  $x_n$  spaced uniformly between  $[0 \ 1]$ .
- The target set (blue circles) was obtained by first computing the corresponding values of the sin function, and then adding a small Gaussian noise.

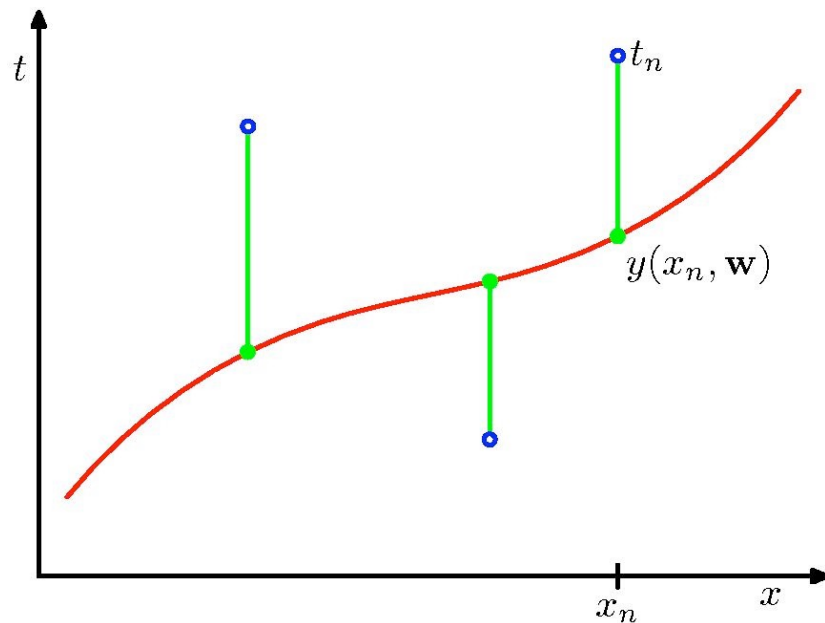
Goal: Fit the data using a polynomial function of the form:

$$y(x, \mathbf{w}) = w_0 + w_1x + w_2x^2 + \dots + w_Mx^M = \sum_{j=0}^M w_jx^j.$$

Note: the polynomial function is a nonlinear function of  $x$ , but it is a linear function of the coefficients  $\mathbf{w} \rightarrow$  **Linear Models**.

# Example: Polynomial Curve Fitting

- As for the least squares example: we can minimize the sum of the squares of the errors between the predictions  $y(x_n, \mathbf{w})$  for each data point  $x_n$  and the corresponding target values  $t_n$ .

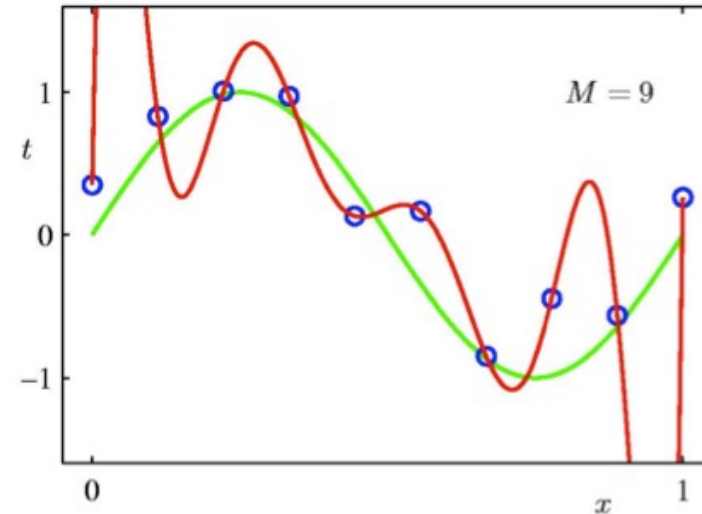
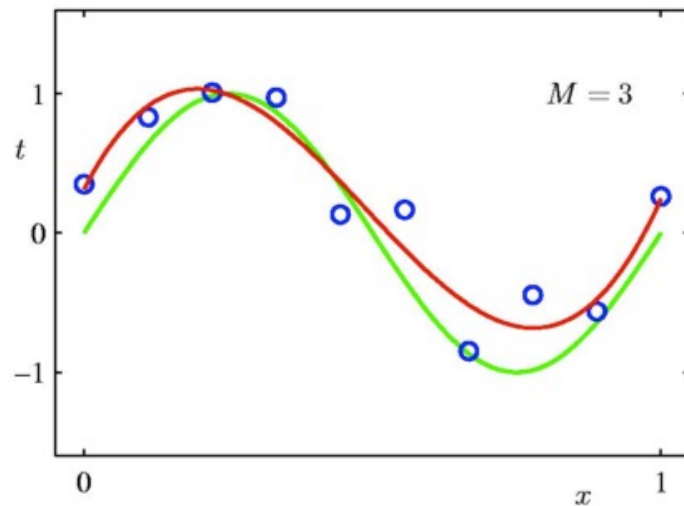
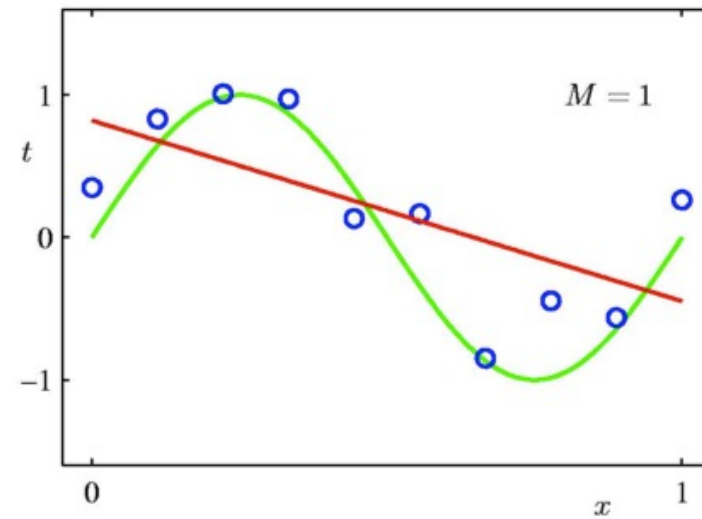
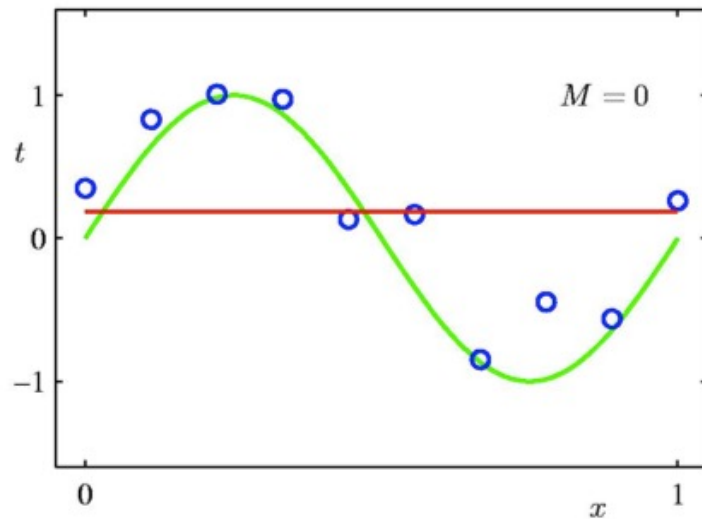


Loss function: sum-of-squared error function:

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

- Similar to the linear least squares: Minimizing sum-of-squared error function has a unique solution  $\mathbf{w}^*$ .
- The model is characterized by  $M+1$  parameters  $\mathbf{w}^*$ .
- How do we choose  $M$ ? → **Model Selection**.

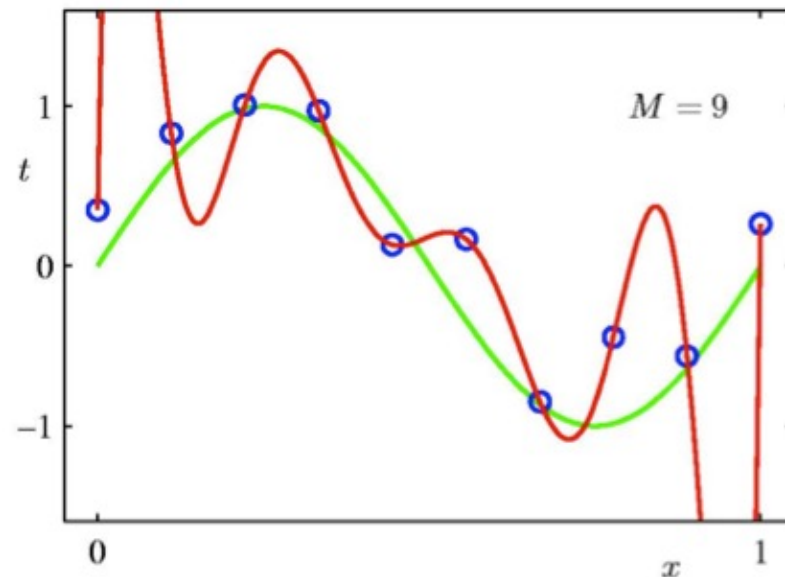
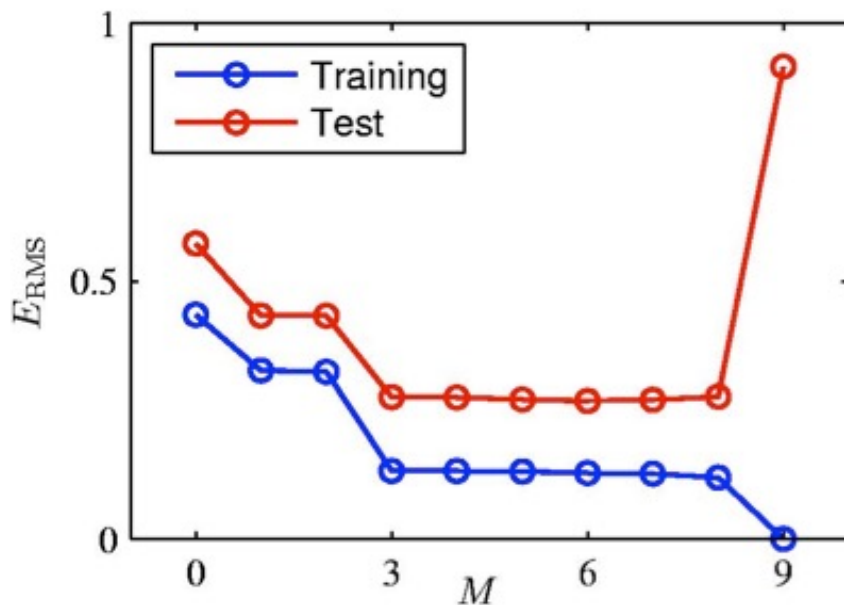
# Some Fits to the Data



For  $M=9$ , we have fitted the training data perfectly.

# Overfitting

- Consider a separate **test set** containing 100 new data points generated using the same procedure that was used to generate the training data.

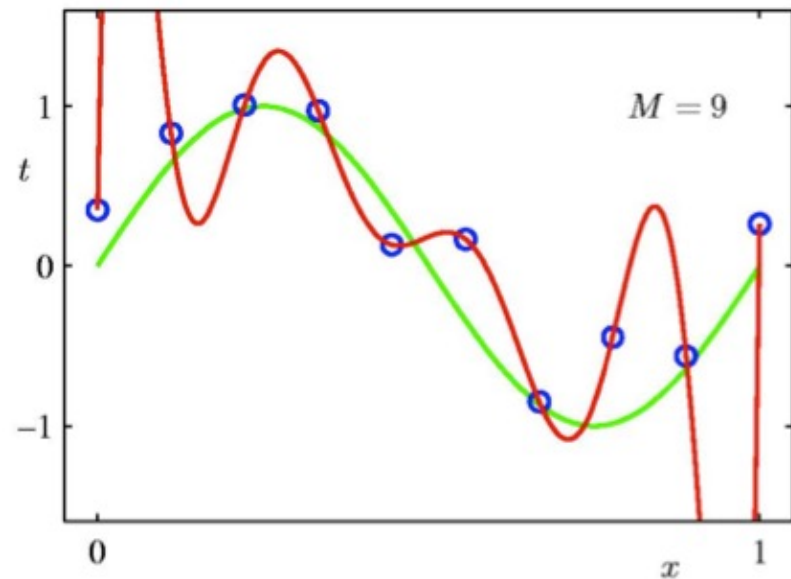


- For  $M=9$ , the training error is zero  $\rightarrow$  The polynomial contains 10 degrees of freedom corresponding to 10 parameters  $\mathbf{w}$ , and so can be fitted exactly to the 10 data points.
- However, the test error has become very large. Why?



# Overfitting

|         | $M = 0$ | $M = 1$ | $M = 3$ | $M = 9$     |
|---------|---------|---------|---------|-------------|
| $w_0^*$ | 0.19    | 0.82    | 0.31    | 0.35        |
| $w_1^*$ |         | -1.27   | 7.99    | 232.37      |
| $w_2^*$ |         |         | -25.43  | -5321.83    |
| $w_3^*$ |         |         | 17.37   | 48568.31    |
| $w_4^*$ |         |         |         | -231639.30  |
| $w_5^*$ |         |         |         | 640042.26   |
| $w_6^*$ |         |         |         | -1061800.52 |
| $w_7^*$ |         |         |         | 1042400.18  |
| $w_8^*$ |         |         |         | -557682.99  |
| $w_9^*$ |         |         |         | 125201.43   |

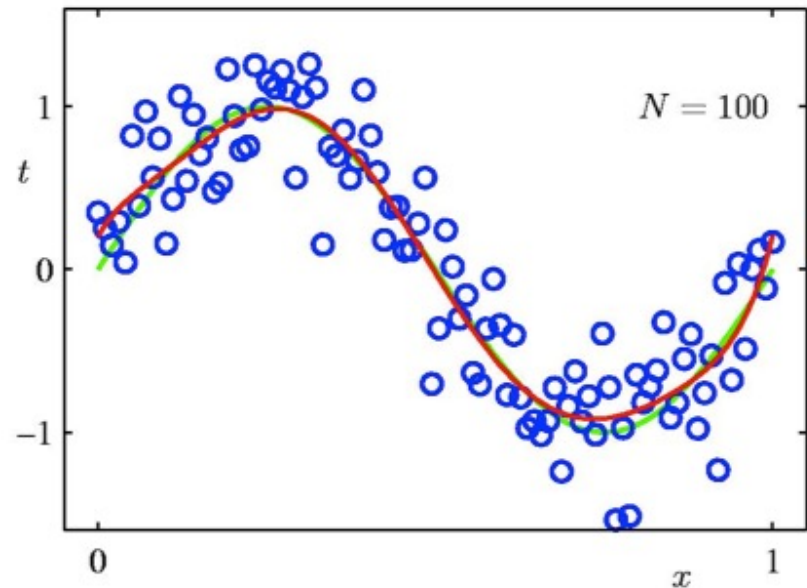
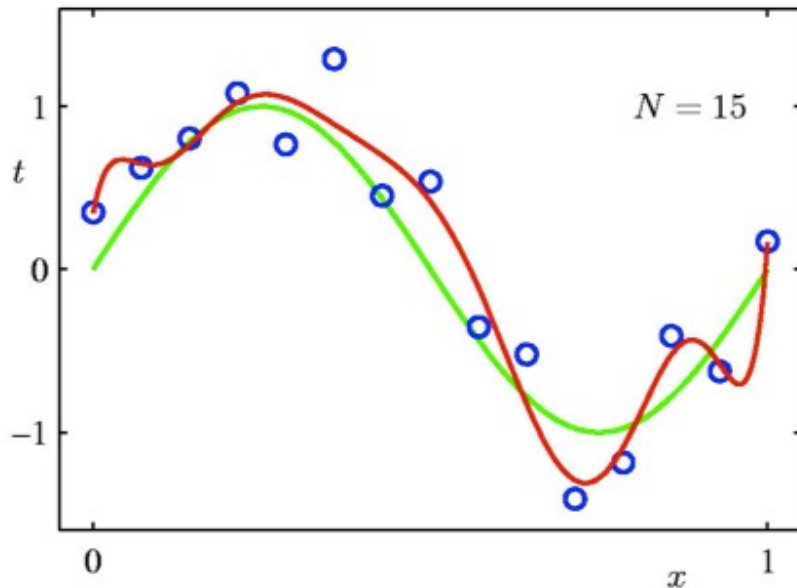


- As  $M$  increases, the magnitude of coefficients gets larger.
- For  $M=9$ , the coefficients have become finely tuned to the data.
- Between data points, the function exhibits large oscillations.

More flexible polynomials with larger  $M$  tune to the random noise on the target values.

# Varying the Size of the Data

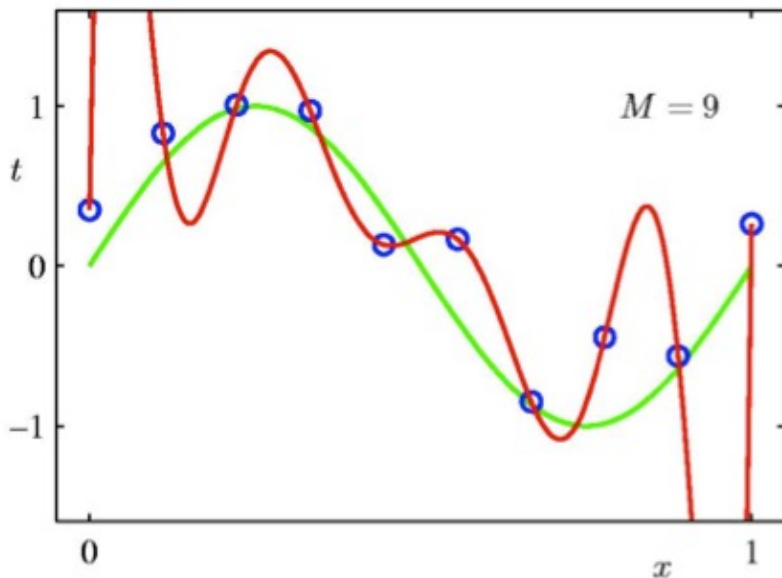
9th order polynomial



- For a given model complexity, the overfitting problem becomes less severe as the size of the dataset increases.
- However, the number of parameters is not necessarily the most appropriate measure of the model complexity.

# Generalization

- The goal is achieve good **generalization** by making accurate predictions for new test data that is not known during learning.
- Choosing the values of parameters that minimize the loss function on the training data may not be the best option.
- We would like to model the true regularities in the data and ignore the noise in the data:
  - It is hard to know which regularities are real and which are accidental due to the particular training examples we happen to pick.



- **Intuition:** We expect the model to generalize if it explains the data well given the complexity of the model.
- If the model has as many degrees of freedom as the data, it can fit the data perfectly. But this is not very informative.
- Some theory on how to control model complexity to optimize generalization.

# A Simple Way to Penalize Complexity

One technique for controlling over-fitting phenomenon is **regularization**, which amounts to adding a penalty term to the error function.

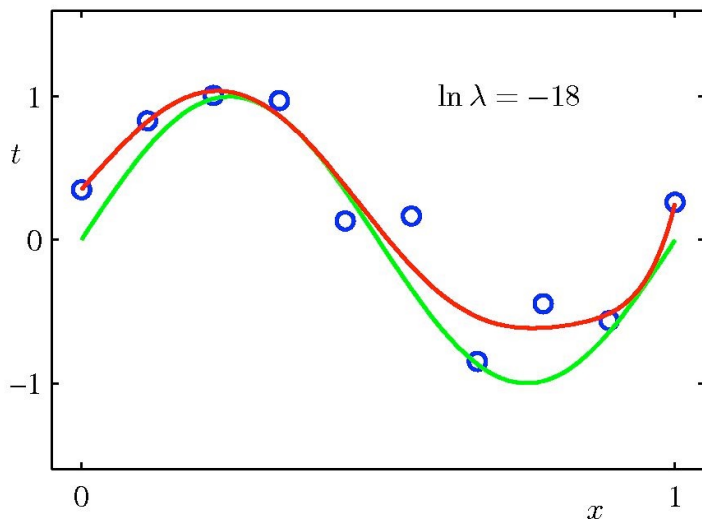
penalized error  
function

target value

regularization  
parameter

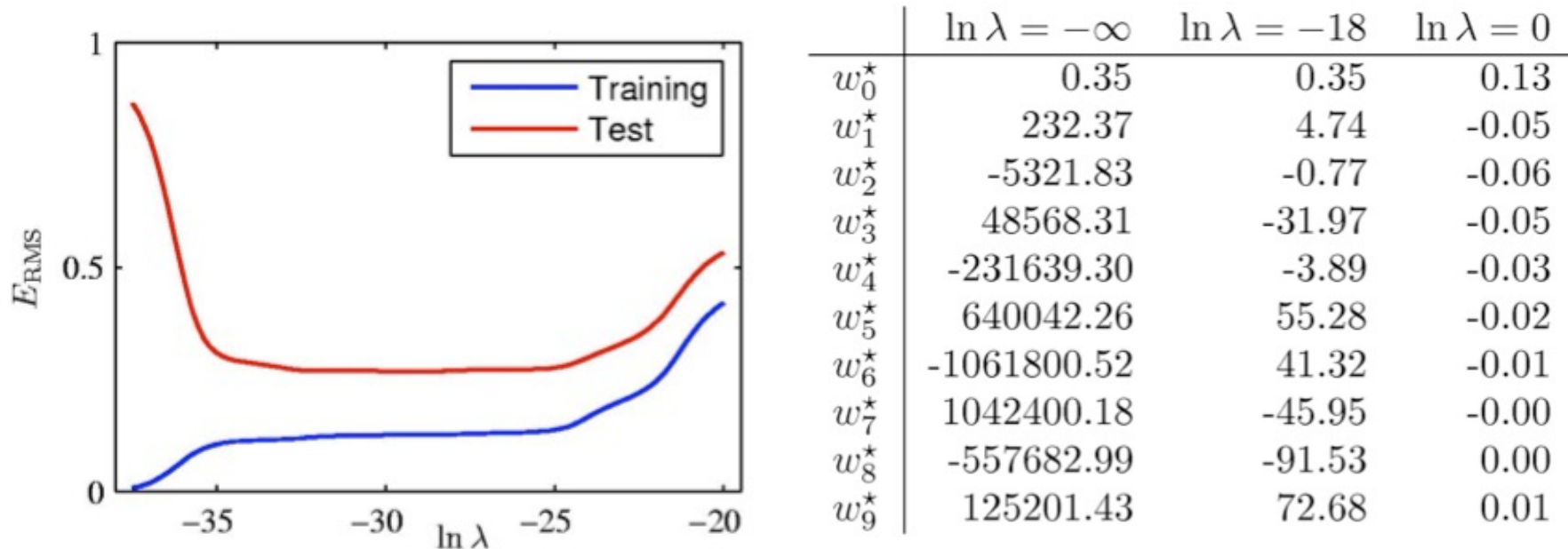
$$\tilde{E}(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

where  $\|\mathbf{w}\| = \mathbf{w}^T \mathbf{w} = w_1^2 + w_2^2 + \dots + w_M^2$  called the regularization term.  
Note that we do not penalize the bias term  $w_0$ .



- The idea is to “shrink” estimated parameters towards zero (or towards the mean of some other weights).
- Shrinking to zero: penalize coefficients based on their size.
- For a penalty function which is the sum of the squares of the parameters, this is known as “**weight decay**”, or “**ridge regression**”.

# Regularization



Graph of the root-mean-squared training and test errors vs.  $\ln \lambda$  for the  $M=9$  polynomial.

How to choose  $\lambda$  ?

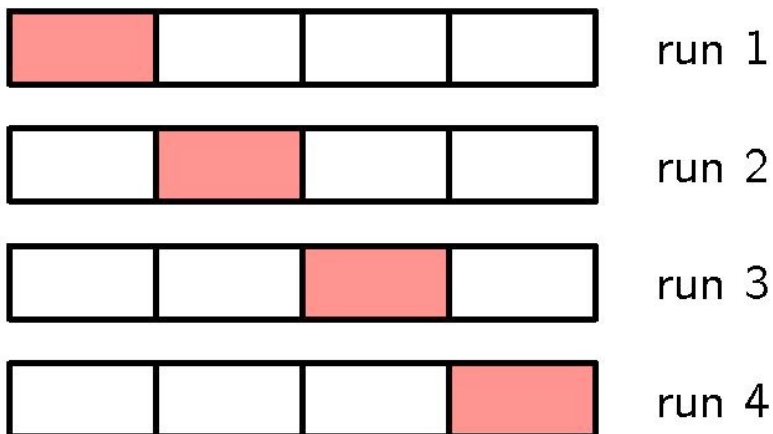
# Cross Validation

If the data is plentiful, we can divide the dataset into three subsets:

- **Training Data:** used to fitting/learning the parameters of the model.
- **Validation Data:** not used for learning but for selecting the model, or choosing the amount of regularization that works best.
- **Test Data:** used to get performance of the final model.

For many applications, the supply of data for training and testing is limited. To build good models, we may want to use as much training data as possible. If the validation set is small, we get noisy estimate of the predictive performance.

S fold cross-validation



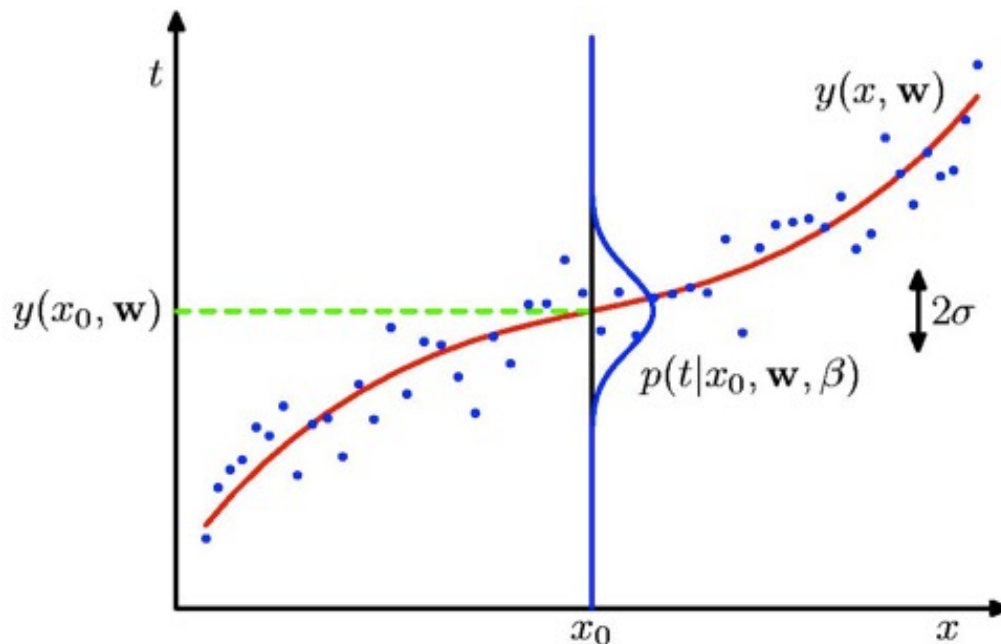
- The data is partitioned into  $S$  groups.
- Then  $S-1$  of the groups are used for training the model, which is evaluated on the remaining group.
- Repeat procedure for all  $S$  possible choices of the held-out group.
- Performance from the  $S$  runs are averaged.

# Probabilistic Perspective

- So far we saw that polynomial curve fitting can be expressed in terms of error minimization. We now view it from probabilistic perspective.
- Suppose that our model arose from a statistical model:

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon,$$

where  $\epsilon$  is a random error having Gaussian distribution with zero mean, and is independent of  $\mathbf{x}$ .



Thus we have:

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}),$$

where  $\beta$  is a precision parameter, corresponding to the inverse variance.

I will use probability distribution and probability density interchangeably. It should be obvious from the context.



# Sampling Assumption

- Assume that the training examples are drawn **independently** from the set of all possible examples, or from the same underlying distribution  $p(\mathbf{x}, t)$ .
- We also assume that the training examples are **identically distributed**  $\rightarrow$  i.i.d assumption.
- Assume that the test samples are drawn in exactly the same way -- i.i.d from the same distribution as the training data.
- These assumptions make it unlikely that some strong regularity in the training data will be absent in the test data.

# Maximum Likelihood

If the data are assumed to be independently and identically distributed (*i.i.d assumption*), the likelihood function takes form:

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{i=1}^N \mathcal{N}(t_n|y(\mathbf{x}_n, \mathbf{w}), \beta^{-1}).$$

It is often convenient to maximize the log of the likelihood function:

$$\ln p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = -\underbrace{\frac{\beta}{2} \sum_{n=1}^N (y(\mathbf{x}_n, \mathbf{w}) - t_n)^2}_{\beta E(\mathbf{w})} + \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi).$$

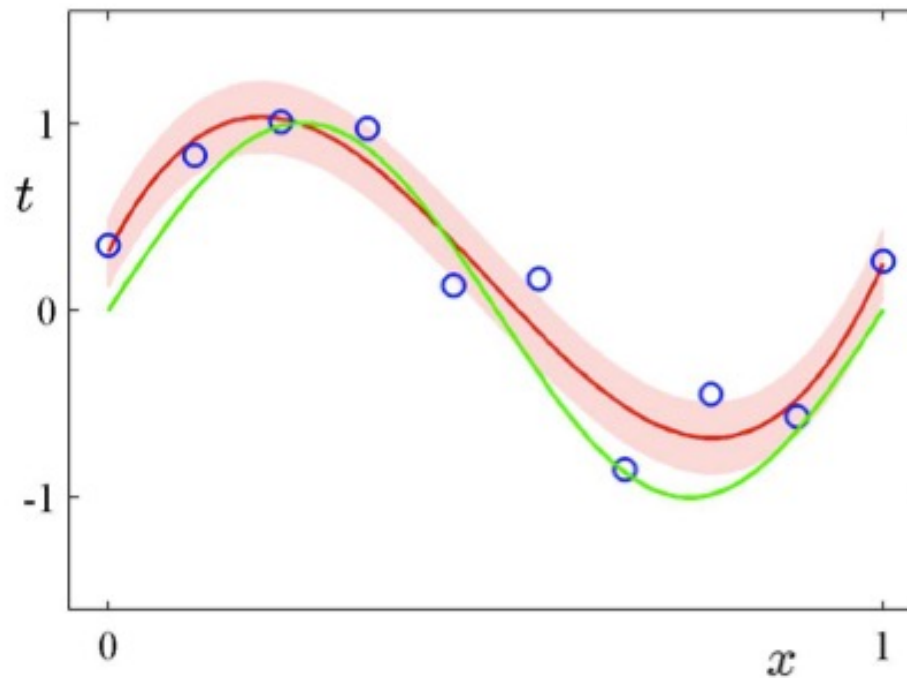
- Maximizing log-likelihood with respect to  $\mathbf{w}$  (under the assumption of a Gaussian noise) is equivalent to minimizing the *sum-of-squared error* function.
- Determine  $\mathbf{w}_{ML}$  by maximizing log-likelihood. Then maximizing w.r.t.  $\beta$ :

$$\frac{1}{\beta_{ML}} = \frac{1}{N} \sum_n (y(\mathbf{x}_n, \mathbf{w}_{ML}) - t_n)^2.$$

# Predictive Distribution

Once we determined the parameters  $\mathbf{w}$  and  $\beta$ , we can make prediction for new values of  $\mathbf{x}$ :

$$p(t|\mathbf{x}, \mathbf{w}_{ML}, \beta_{ML}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}_{ML}), \beta_{ML}^{-1}).$$



# Statistical Decision Theory

- We now develop a small amount of theory that provides a framework for developing many of the models we consider.
- Suppose we have a real-valued input vector  $\mathbf{x}$  and a corresponding target (output) value  $t$  with joint probability distribution:  $p(\mathbf{x}, t)$ .
- Our goal is predict target  $t$  given a new value for  $\mathbf{x}$ :
  - for regression:  $t$  is a real-valued continuous target.
  - for classification:  $t$  a categorical variable representing class labels.

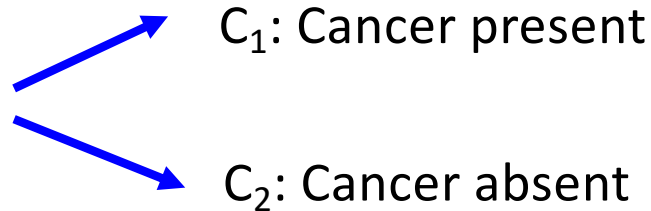
The joint probability distribution  $p(\mathbf{x}, t)$  provides a complete summary of uncertainties associated with these random variables.

Determining  $p(\mathbf{x}, t)$  from training data is known as the **inference problem**.

# Example: Classification

**Medical diagnosis:** Based on the X-ray image, we would like determine whether the patient has cancer or not.

- The input vector  $\mathbf{x}$  is the set of pixel intensities, and the output variable  $t$  will represent the presence of cancer, class  $C_1$ , or absence of cancer, class  $C_2$ .



$\mathbf{x}$  -- set of pixel intensities

- Choose  $t$  to be binary:  $t=0$  correspond to class  $C_1$ , and  $t=1$  corresponds to  $C_2$ .

**Inference Problem:** Determine the joint distribution  $p(\mathbf{x}, \mathcal{C}_k)$  or equivalently  $p(\mathbf{x}, t)$ . However, in the end, we must **make a decision** of whether to give treatment to the patient or not.

# Example: Classification

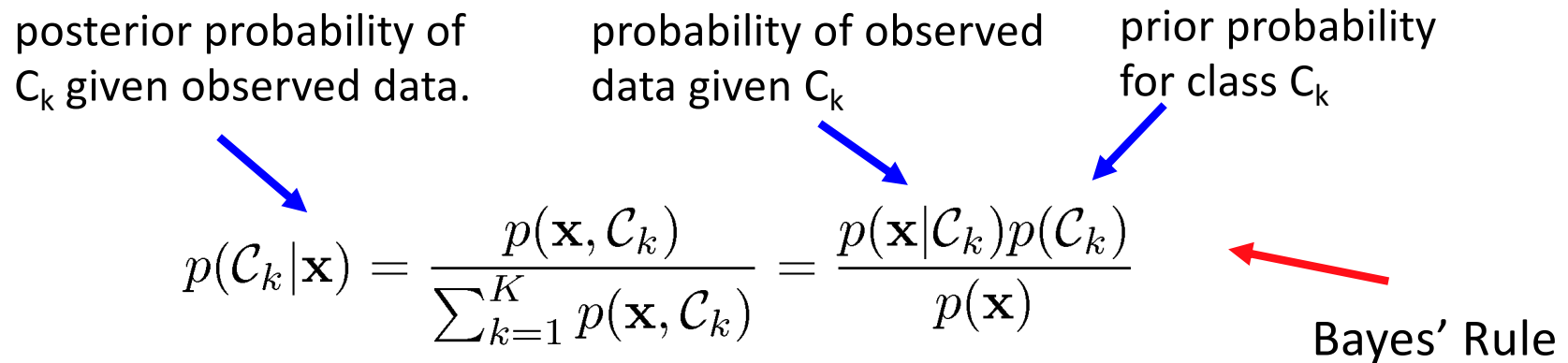
**Informally:** Given a new X-ray image, our goal is to decide which of the two classes that image should be assigned to.

- We could compute conditional probabilities of the two classes, given the input image:

posterior probability of  $C_k$  given observed data.

probability of observed data given  $C_k$

prior probability for class  $C_k$


$$p(C_k|\mathbf{x}) = \frac{p(\mathbf{x}, C_k)}{\sum_{k=1}^K p(\mathbf{x}, C_k)} = \frac{p(\mathbf{x}|C_k)p(C_k)}{p(\mathbf{x})}$$

Bayes' Rule

- If our goal to minimize the probability of assigning  $\mathbf{x}$  to the wrong class, then we should choose the class having the highest posterior probability.

# Expected Loss

- **Loss Function**: overall measure of loss incurred by taking any of the available decisions.
- Suppose that for  $\mathbf{x}$ , the true class is  $C_k$ , but we assign  $\mathbf{x}$  to class  $j$   
→ incur loss of  $L_{kj}$  ( $k,j$  element of a loss matrix).

Consider medical diagnosis example: example of a loss matrix:

|       |        | Decision |        |
|-------|--------|----------|--------|
|       |        | cancer   | normal |
| Truth | cancer | 0        | 1000   |
|       | normal | 1        | 0      |

Expected Loss:

$$\mathbb{E}[L] = \sum_k \sum_j \int_{\mathcal{R}_j} L_{kj} p(\mathbf{x}, C_k) d\mathbf{x}$$

Goal is to choose decision regions  $\mathcal{R}_j$  as to minimize expected loss.



# Regression

Let  $\mathbf{x} \in \mathbb{R}^d$  denote a real-valued input vector, and  $t \in \mathbb{R}$  denote a real-valued random target (output) variable with joint the distribution  $p(\mathbf{x}, t)$ .

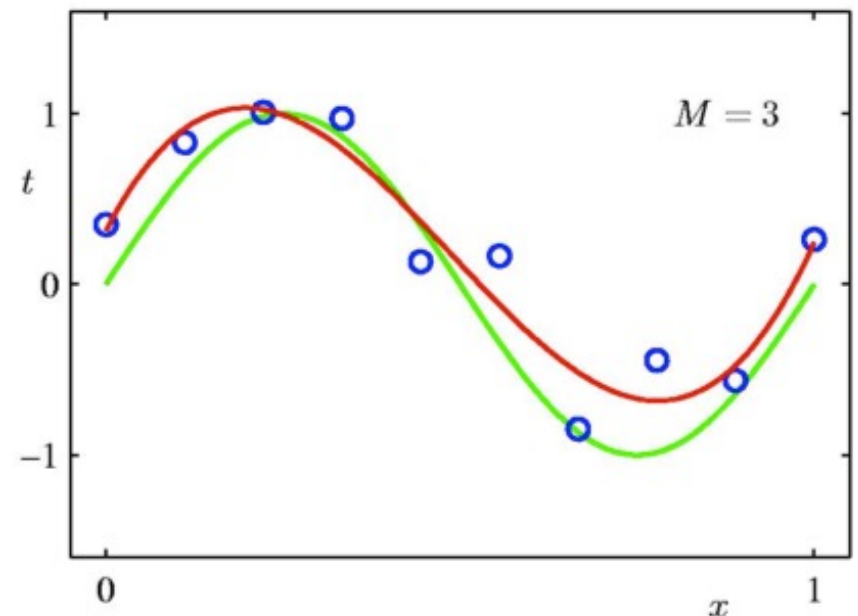
- The decision step consists of finding an estimate  $y(\mathbf{x})$  of  $t$  for each input  $\mathbf{x}$ .
- To quantify what it means to do well or poorly on a task, we need to define a loss (error) function:  $L(t, y(\mathbf{x}))$ .

- The average, or expected, loss is given by:

$$\mathbb{E}[L] = \int \int L(t, y(\mathbf{x})) p(\mathbf{x}, t) d\mathbf{x} dt.$$

- If we use squared loss, we obtain:

$$\mathbb{E}[L] = \int \int (t - y(\mathbf{x}))^2 p(\mathbf{x}, t) d\mathbf{x} dt.$$



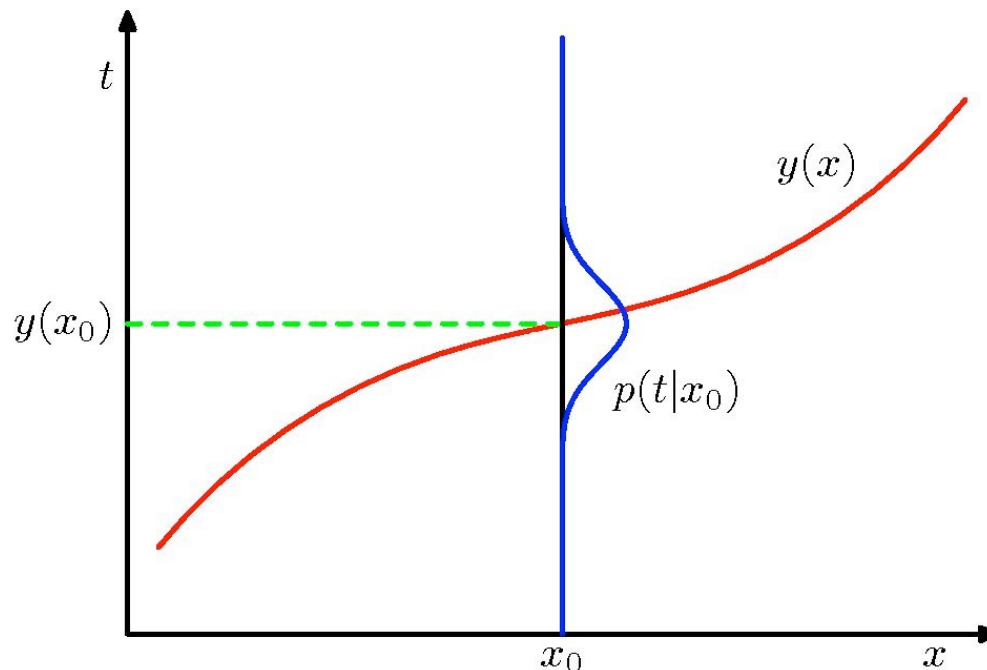
# Squared Loss Function

- If we use squared loss, we obtain:

$$\mathbb{E}[L] = \int \int (t - y(\mathbf{x}))^2 p(\mathbf{x}, t) d\mathbf{x} dt.$$

- Our goal is to choose  $y(\mathbf{x})$  so as to minimize the expected squared loss.
- The optimal solution (if we assume a completely flexible function) is the conditional average:

$$y(\mathbf{x}) = \int t p(t|\mathbf{x}) dt = \mathbb{E}[t|\mathbf{x}].$$



The regression function  $y(\mathbf{x})$  that minimizes the expected squared loss is given by the mean of the conditional distribution  $p(t|\mathbf{x})$ .

# Squared Loss Function

- If we use squared loss, we obtain:

$$\begin{aligned}(y(\mathbf{x}) - t)^2 &= (y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}] + \mathbb{E}[t|\mathbf{x}] - t)^2 \\ &= (y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}])^2 + 2(y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}])(\mathbb{E}[t|\mathbf{x}] - t) + (\mathbb{E}[t|\mathbf{x}] - t)^2.\end{aligned}$$

- Plugging into expected loss:

$$\mathbb{E}[L] = \underbrace{\int \{y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}]\}^2 p(\mathbf{x}) d\mathbf{x}}_{\text{expected loss is minimized when } y(\mathbf{x}) = \mathbb{E}[t|\mathbf{x}]} + \underbrace{\int \text{var}[t|\mathbf{x}] p(\mathbf{x}) d\mathbf{x}}_{\text{intrinsic variability of the target values.}}$$

expected loss is minimized  
when  $y(\mathbf{x}) = \mathbb{E}[t|\mathbf{x}]$ .

intrinsic variability of the  
target values.

Because it is independent noise, it  
represents an irreducible minimum  
value of expected loss.

# Other Loss Function

- Simple generalization of the squared loss, called the *Minkowski* loss:

$$\mathbb{E}[L] = \int \int (t - y(\mathbf{x}))^q p(\mathbf{x}, t) d\mathbf{x} dt.$$

- The minimum of  $\mathbb{E}[L]$  is given by:
  - the conditional mean for  $q=2$ ,
  - the conditional median when  $q=1$ , and
  - the conditional mode for  $q \rightarrow 0$ .

# Discriminative vs. Generative

- Generative Approach:

Model the joint density:  $p(\mathbf{x}, t) = p(\mathbf{x}|t)p(t)$ ,  
or joint distribution:  $p(\mathbf{x}, \mathcal{C}_k) = p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)$ .

Infer conditional  
density:

$$p(t|\mathbf{x}) = \frac{p(\mathbf{x}|t)p(t)}{p(\mathbf{x})}.$$

- Discriminative Approach:

Model conditional density  $p(t|\mathbf{x})$  directly.

# Linear Basis Function Models

- Remember, the simplest linear model for regression:

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1x_1 + w_2x_2 + \dots + w_dx_d = w_0 + \sum_{j=1}^d w_jx_j,$$

where  $\mathbf{x} = (x_1, x_2, \dots, x_d)^T$  is a d-dimensional input vector (covariates).

**Key property:** linear function of the parameters  $w_0, w_1, \dots, w_d$ .

- However, it is also a linear function of the input variables.

Instead consider:

$$y(\mathbf{x}, \mathbf{w}) = w_0\phi_0(\mathbf{x}) + w_1\phi_1(\mathbf{x}) + \dots + w_{M-1}\phi_{M-1}(\mathbf{x}) = \sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}),$$

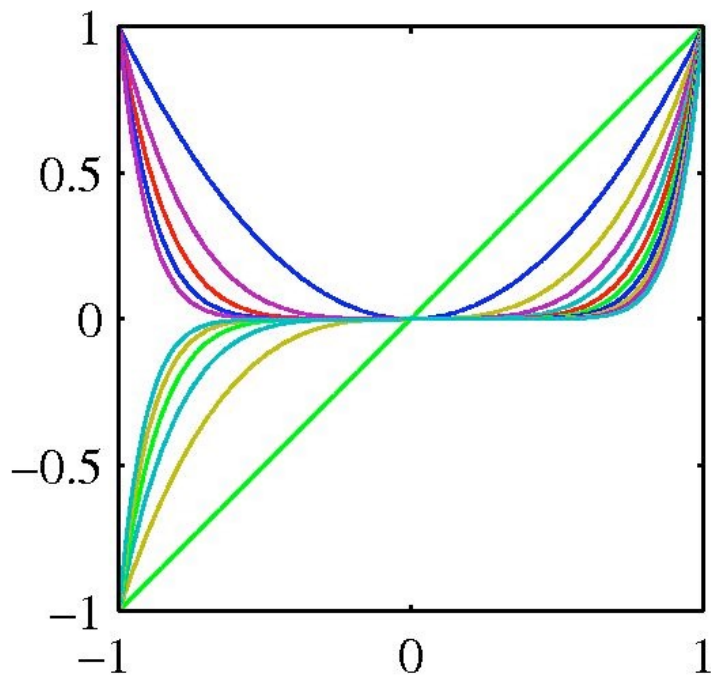
where  $\phi_j(\mathbf{x})$  are known as basis functions.

- Typically  $\phi_0(\mathbf{x}) = 1$  so that  $w_0$  acts as a bias (or intercept).
- In the simplest case, we use linear bases functions:  $\phi_j(\mathbf{x}) = x_j$ .
- Using nonlinear basis allows the functions  $y(\mathbf{x}, \mathbf{w})$  to be nonlinear functions of the input space.

# Linear Basis Function Models

Polynomial basis functions:

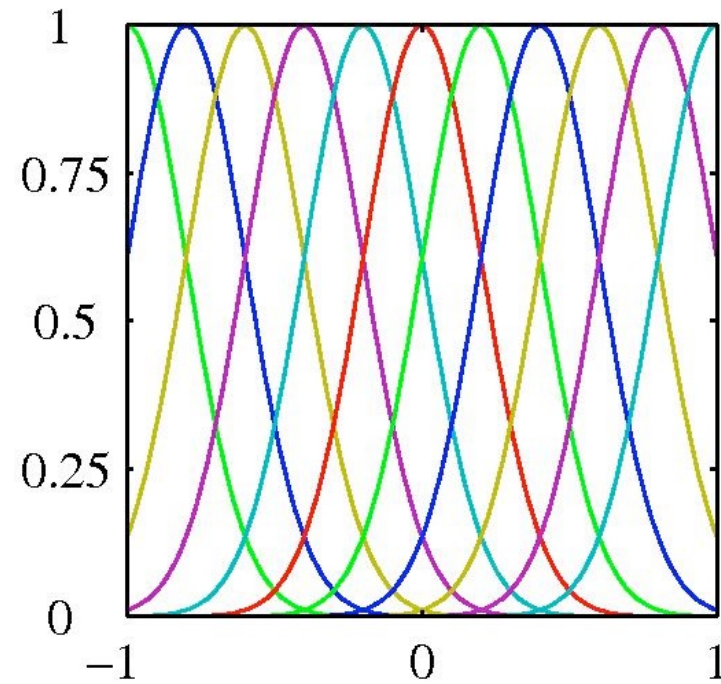
$$\phi_j(x) = x^j.$$



Basis functions are global: small changes in  $\mathbf{x}$  affect all basis functions.

Gaussian basis functions:

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



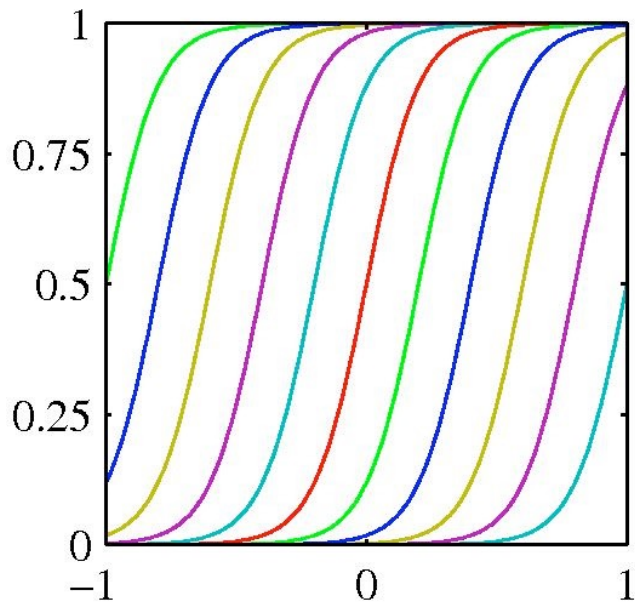
Basis functions are local: small changes in  $\mathbf{x}$  only affect nearby basis functions.  
 $\mu_j$  and  $s$  control location and scale (width).



# Linear Basis Function Models

Sigmoidal basis functions

$$\phi_j(x) = \sigma\left(\frac{x - \mu_j}{s}\right), \text{ where } \sigma(a) = \frac{1}{1 + \exp(-a)}.$$

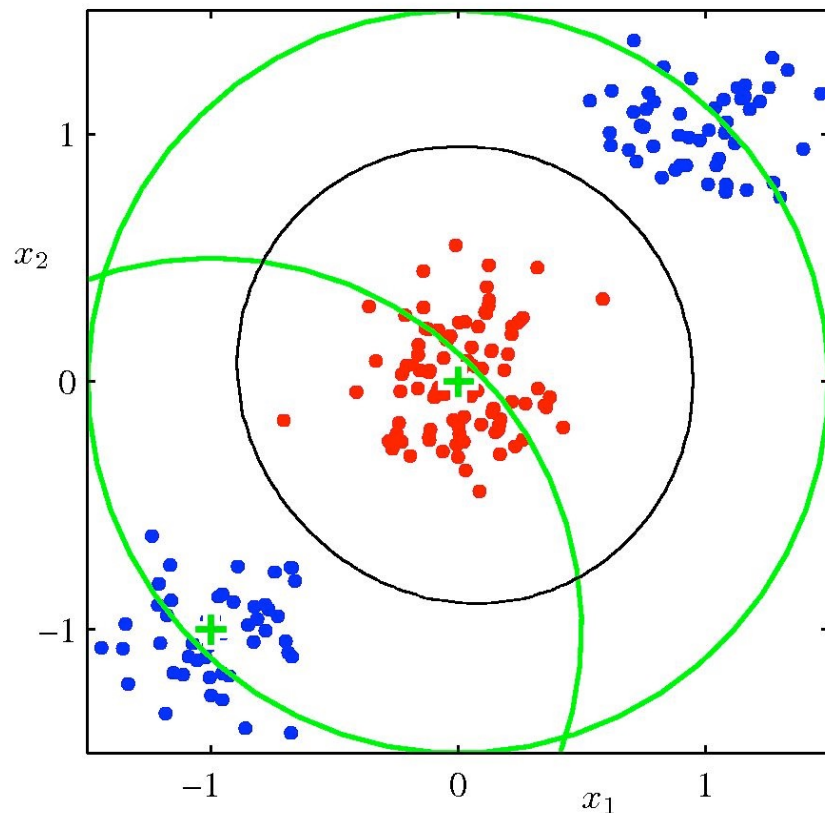


Basis functions are local: small changes in  $\mathbf{x}$  only affect nearby basis functions.  $\mu_j$  and  $s$  control location and scale (slope).

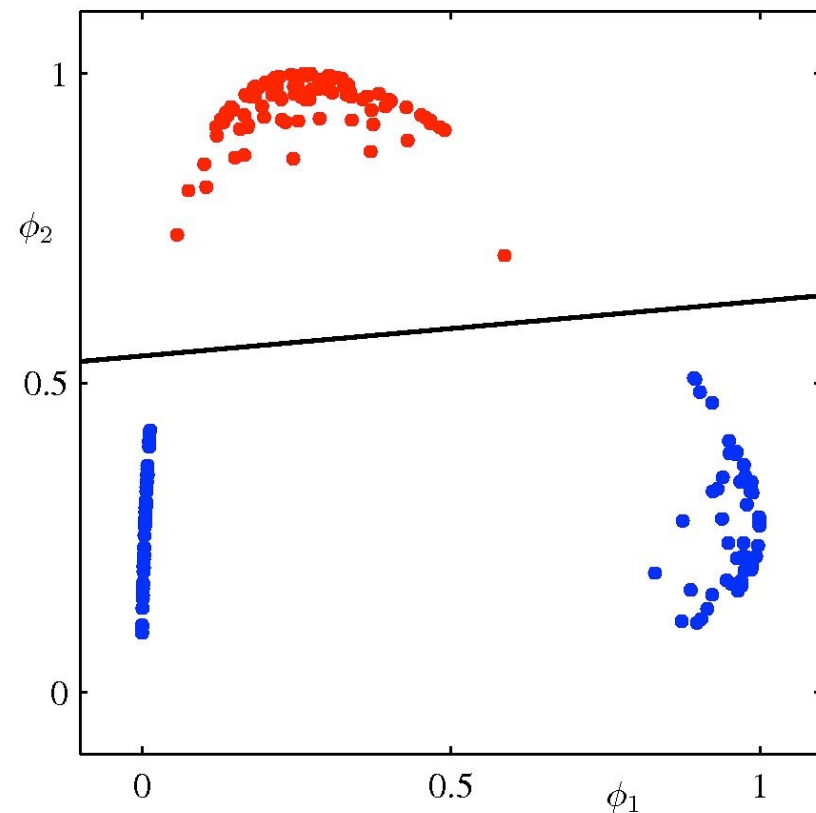
- Decision boundaries will be linear in the feature space  $\phi$ , but would correspond to nonlinear boundaries in the original input space  $\mathbf{x}$ .
- Classes that are linearly separable in the feature space  $\phi(\mathbf{x})$  need not be linearly separable in the original input space.

# Linear Basis Function Models

Original input space



Corresponding feature space using two Gaussian basis functions



- We define two Gaussian basis functions with centers shown by green the crosses, and with contours shown by the green circles.
- Linear decision boundary (right) is obtained using logistic regression, and corresponds to nonlinear decision boundary in the input space (left, black curve).

# Maximum Likelihood

- As before, assume observations arise from a deterministic function with an additive Gaussian noise:

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon,$$

which we can write as:

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}).$$

- Given observed inputs  $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ , and corresponding target values  $\mathbf{t} = [t_1, t_2, \dots, t_N]^T$ , under i.i.d assumption, we can write down the likelihood function:

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{i=1}^N \mathcal{N}(t_n|\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n), \beta),$$

where  $\boldsymbol{\phi}(\mathbf{x}) = (\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), \dots, \phi_{M-1}(\mathbf{x}))^T$ .

# Maximum Likelihood

Taking the logarithm, we obtain:

$$\begin{aligned}\ln p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) &= \sum_{i=1}^N \ln \mathcal{N}(t_n | \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n), \beta) \\ &= -\frac{\beta}{2} \underbrace{\sum_{n=1}^N (t_n - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n))^2}_{\text{sum-of-squares error function}} + \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi).\end{aligned}$$

Differentiating and setting to zero yields:

$$\nabla_{\mathbf{w}} \ln p(\mathbf{t}|\mathbf{w}, \beta) = \beta \sum_{n=1}^N \{t_n - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n)\} \boldsymbol{\phi}(\mathbf{x}_n)^T = \mathbf{0}.$$

# Maximum Likelihood

Differentiating and setting to zero yields:

$$\nabla_{\mathbf{w}} \ln p(\mathbf{t}|\mathbf{w}, \beta) = \beta \sum_{n=1}^N \{t_n - \mathbf{w}^T \phi(\mathbf{x}_n)\} \phi(\mathbf{x}_n)^T = \mathbf{0}.$$

Solving for  $\mathbf{w}$ , we get:

$$\mathbf{w}_{\text{ML}} = \left( \Phi^T \Phi \right)^{-1} \Phi^T \mathbf{t}$$

The Moore-Penrose pseudo-inverse,  $\Phi^\dagger$ .

where  $\Phi$  is known as the **design matrix**:

$$\Phi = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix}.$$

# Sequential Learning

- The training data examples are presented one at a time, and the model parameters are updated after each such presentation (online learning):

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla E_n$$

Diagram illustrating the weight update equation:

- $\mathbf{w}^{(t+1)}$ : weights after seeing training case  $t+1$  (indicated by a blue arrow)
- $\mathbf{w}^{(t)}$ : weights before seeing training case  $t+1$  (indicated by a blue arrow)
- $\eta$ : learning rate (indicated by a blue arrow)
- $\nabla E_n$ : vector of derivatives of the squared error w.r.t. the weights on the training case presented at time  $t$ . (indicated by a red arrow)

- For the case of sum-of-squares error function, we obtain:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \left( t_n - \mathbf{w}^{(t)T} \phi(\mathbf{x}_n) \right) \phi(\mathbf{x}_n).$$

- **Stochastic gradient descent**: The training examples are picked at random (dominant technique when learning with very large datasets).
- Care must be taken when choosing learning rate to ensure convergence.

# Regularized Least Squares

- Let us consider the following error function:

$$E_D(\mathbf{w}) + \lambda E_W(\mathbf{w})$$

Data term + Regularization term

$\lambda$  is called the regularization coefficient.

- Using sum-of-squares error function with a quadratic penalization term, we obtain:

$$\frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^T \phi(\mathbf{x}_n)\}^2 + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$$

which is minimized by setting:

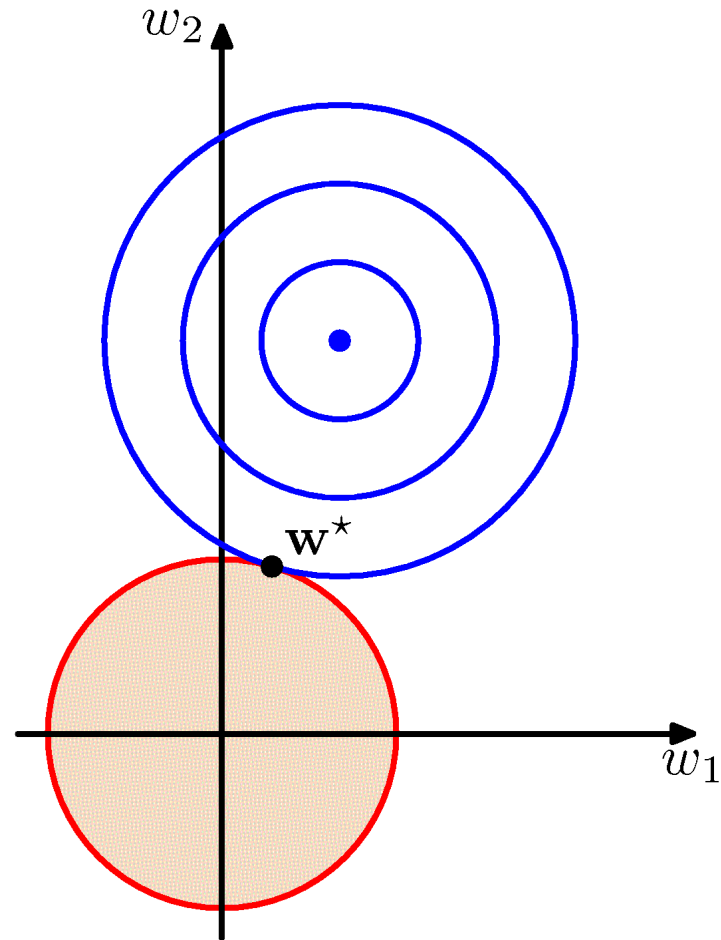
$$\mathbf{w} = \left( \lambda \mathbf{I} + \Phi^T \Phi \right)^{-1} \Phi^T \mathbf{t}.$$

Ridge regression

The solution adds a positive constant to the diagonal of  $\Phi^T \Phi$ . This makes the problem nonsingular, even if  $\Phi^T \Phi$  is not of full rank (e.g. when the number of training examples is less than the number of basis functions).

# Effect of Regularization

- The overall error function is the sum of two parabolic bowls.
- The combined minimum lies on the line between the minimum of the squared error and the origin.
- The regularizer shrinks model parameters to zero.

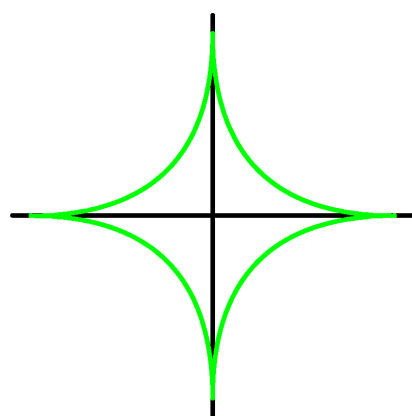




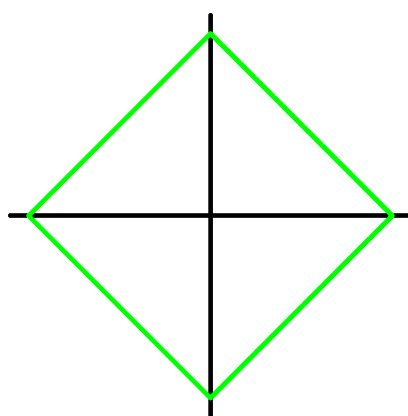
# Other Regularizers

Using a more general regularizer, we get:

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

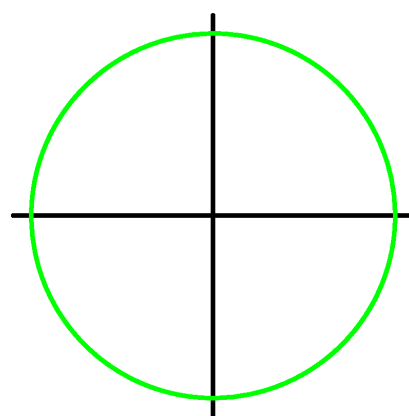


$q = 0.5$



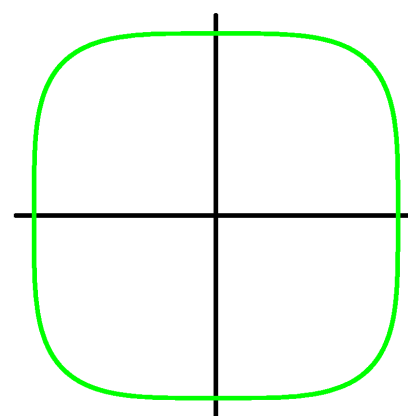
$q = 1$

Lasso



$q = 2$

Quadratic



$q = 4$

# The Lasso

- Penalize the absolute value of the weights:

$$\mathbf{w}^{lasso} = \underset{\mathbf{w}}{\operatorname{argmin}} \left[ \frac{1}{2} \sum_{n=1}^N (t_n - \mathbf{w}^T \phi(\mathbf{x}_n))^2 + \frac{\lambda}{2} \sum_{j=1}^{M-1} |w_j| \right].$$

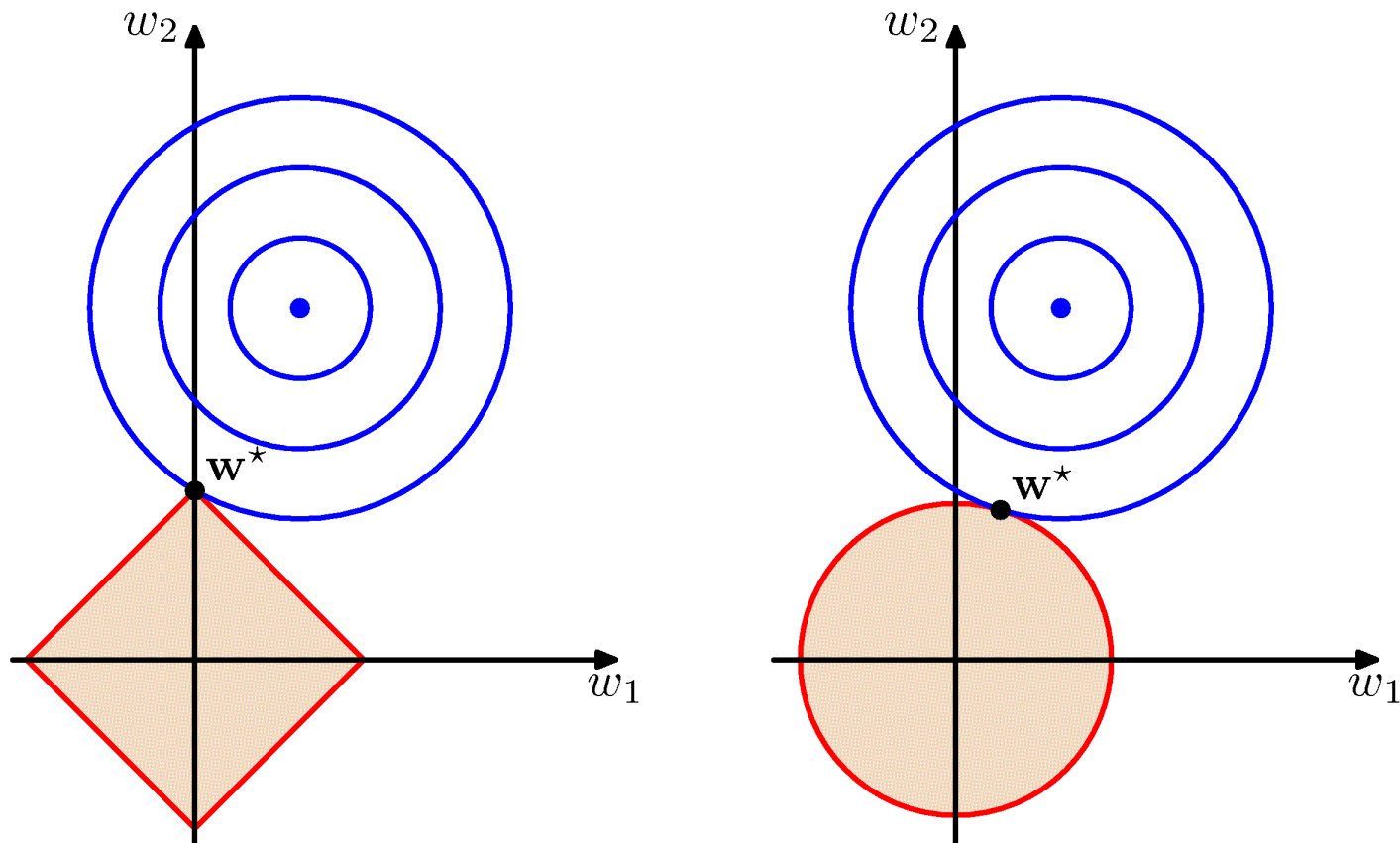
- For sufficiently large  $\lambda$ , some of the coefficients will be driven to exactly zero, leading to a sparse model.
- The above formulation is equivalent to:

$$\mathbf{w}^{lasso} = \underset{\mathbf{w}}{\operatorname{argmin}} \underbrace{\frac{1}{2} \sum_{n=1}^N (t_n - \mathbf{w}^T \phi(\mathbf{x}_n))^2}_{\text{unregularized sum-of-squares error}}, \text{ subject to } \sum_{j=1}^{M-1} |w_j| \leq \tau.$$

- The two approaches are related using Lagrange multipliers.
- The Lasso solution is a quadratic programming problem: can be solved efficiently.

# Lasso vs. Quadratic Penalty

Lasso tends to generate sparser solutions compared to a quadratic regularizer (sometimes called  $L_1$  and  $L_2$  regularizers).



# Bias-Variance Decomposition

- Introducing a regularization term can help us control overfitting. But how can we determine a suitable value of the regularization coefficient?
- Let us examine the expected squared loss function. Remember:

$$\mathbb{E}[L] = \int \{y(\mathbf{x}) - h(\mathbf{x})\}^2 p(\mathbf{x}) d\mathbf{x} + \underbrace{\int \int \{h(\mathbf{x}) - t\}^2 p(\mathbf{x}, t) d\mathbf{x} dt}_{\text{intrinsic variability of the target values: The minimum achievable value of expected loss}}$$

for which the optimal prediction is given by the conditional expectation:

$$h(\mathbf{x}) = \mathbb{E}[t|\mathbf{x}] = \int tp(t|\mathbf{x}) dt.$$

intrinsic variability of the target values: The minimum achievable value of expected loss

- If we model  $h(\mathbf{x})$  using a parametric function  $y(\mathbf{x}, \mathbf{w})$ , then from a Bayesian perspective, the uncertainty in our model is expressed through the posterior distribution over parameters  $\mathbf{w}$ .
- We first look at the frequentist perspective.

# Bias-Variance Decomposition

- From a frequentist perspective: we make a point estimate of  $\mathbf{w}^*$  based on the dataset  $D$ .
- We next interpret the uncertainty of this estimate through the following thought experiment:
  - Suppose we had a large number of datasets, each of size  $N$ , where each dataset is drawn independently from  $p(\mathbf{x}, t)$ .
  - For each dataset  $D$ , we can obtain a prediction function  $y(\mathbf{x}; \mathcal{D})$ .
  - Different datasets will give different prediction functions.
  - The performance of a particular learning algorithm is then assessed by taking the average over the ensemble of these datasets.
- Let us consider the expression:

$$\{y(\mathbf{x}; \mathcal{D}) - h(\mathbf{x})\}^2.$$

- Note that this quantity depends on a particular dataset  $D$ .

# Bias-Variance Decomposition

- Consider:

$$\{y(\mathbf{x}; \mathcal{D}) - h(\mathbf{x})\}^2.$$

- Adding and subtracting the term  $\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})]$ , we obtain

$$\begin{aligned} & \{y(\mathbf{x}; \mathcal{D}) - h(\mathbf{x})\}^2 \\ &= \{y(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})] + \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})] - h(\mathbf{x})\}^2 \\ &= \{y(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})]\}^2 + \{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})] - h(\mathbf{x})\}^2 \\ &\quad + 2\{y(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})]\}\{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})] - h(\mathbf{x})\}. \end{aligned}$$

- Taking the expectation over  $\mathcal{D}$ , the last term vanishes, so we get:

$$\begin{aligned} & \mathbb{E}_{\mathcal{D}} [\{y(\mathbf{x}; \mathcal{D}) - h(\mathbf{x})\}^2] \\ &= \underbrace{\{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})] - h(\mathbf{x})\}^2}_{(\text{bias})^2} + \underbrace{\mathbb{E}_{\mathcal{D}} [\{y(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})]\}^2]}_{\text{variance}}. \end{aligned}$$

# Bias-Variance Trade-off

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise}$$

Average predictions over all datasets differ from the optimal regression function.

Solutions for individual datasets vary around their averages -- how sensitive is the function to the particular choice of the dataset.

Intrinsic variability of the target values.

$$(\text{bias})^2 = \int \{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})] - h(\mathbf{x})\}^2 p(\mathbf{x}) d\mathbf{x}$$

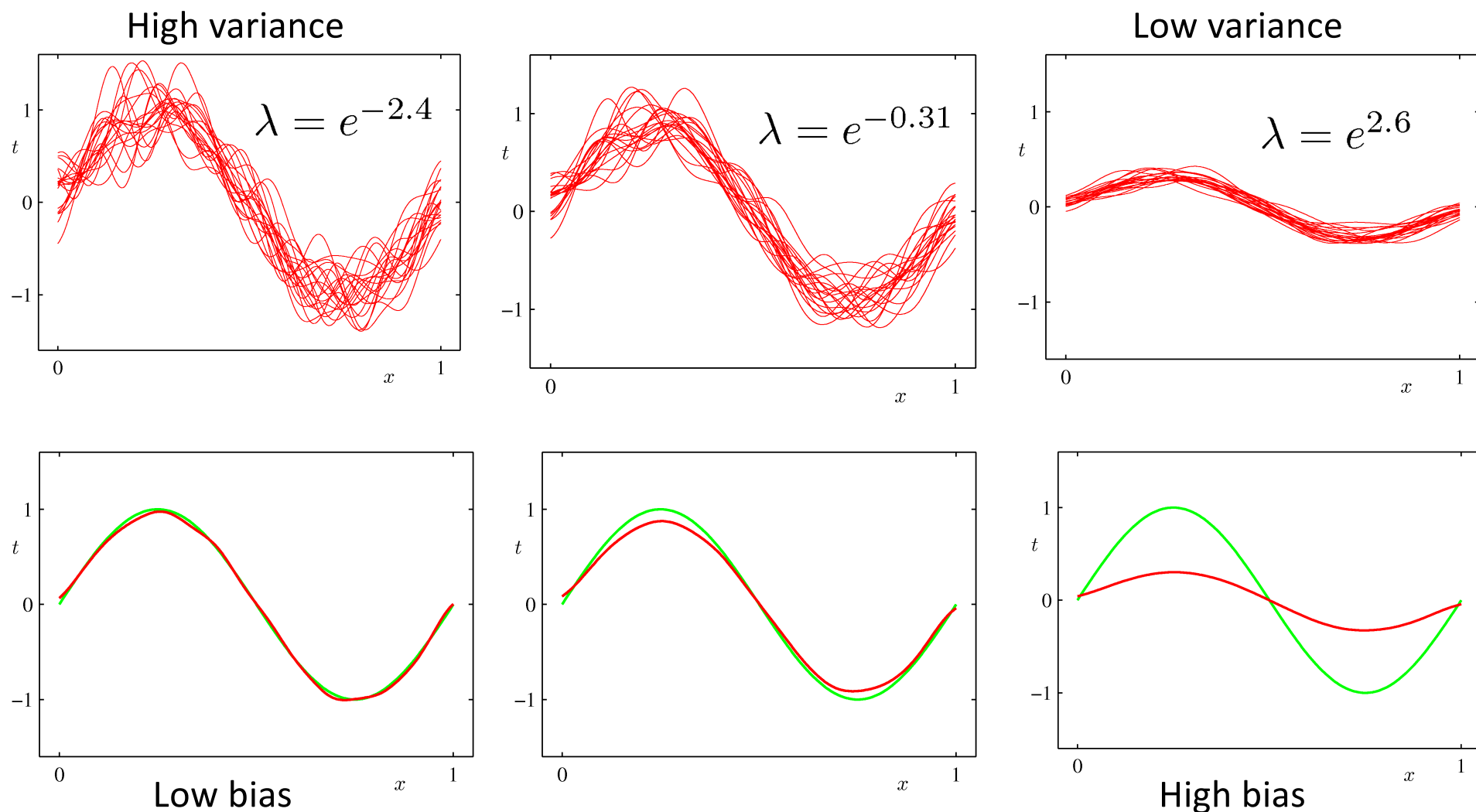
$$\text{variance} = \int \mathbb{E}_{\mathcal{D}} [\{y(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})]\}^2] p(\mathbf{x}) d\mathbf{x}$$

$$\text{noise} = \iint \{h(\mathbf{x}) - t\}^2 p(\mathbf{x}, t) d\mathbf{x} dt$$

- Trade-off between bias and variance: With very flexible models (high complexity) we have low bias and high variance; With relatively rigid models (low complexity) we have high bias and low variance.
- The model with the optimal predictive capabilities has to balance between bias and variance.

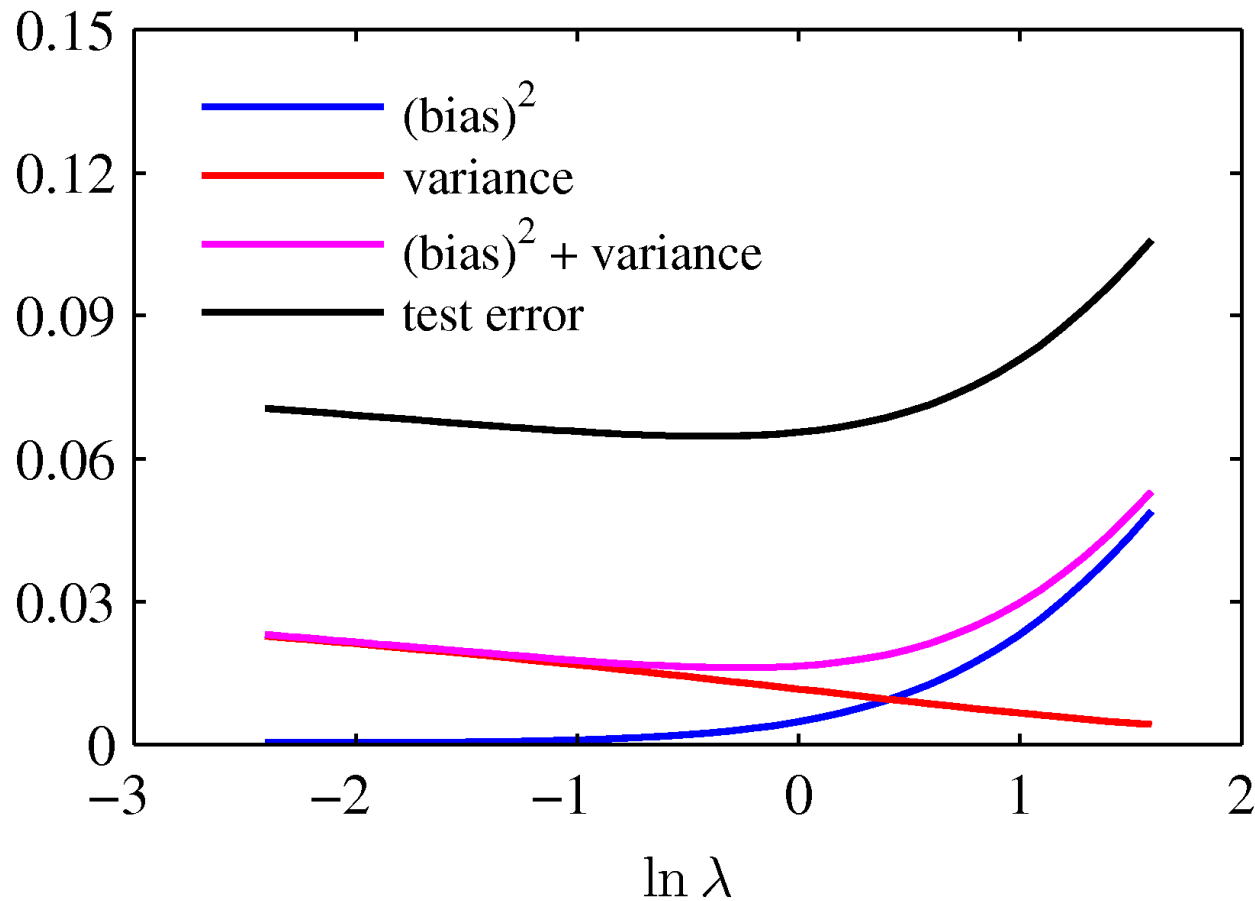
# Bias-Variance Trade-off

- Consider the sinusoidal dataset. We generate 100 datasets, each containing  $N=25$  points, drawn independently from  $h(x) = \sin 2\pi x$ .





# Bias-Variance Trade-off



From these plots note that over-regularized model (large  $\lambda$ ) has high bias, and under-regularized model (low  $\lambda$ ) has high variance.

# Beating the Bias-Variance Trade-off

- We can reduce the variance by averaging over many models trained on different datasets:
  - In practice, we only have a single observed dataset. If we had many independent training sets, we would be better off combining them into one large training dataset. With more data, we have less variance.
- Given a standard training set  $D$  of size  $N$ , we could generate new training sets,  $N$ , by sampling examples from  $D$  uniformly and with replacement.
  - This is called bagging and it works quite well in practice.
- Given enough computation, we could also resort to the Bayesian framework:
  - Combine the predictions of many models using the posterior probability of each parameter vector as the combination weight.