Neural Networks for Pattern Recognition

CSE 253

A.K.A. The “deep nets” course

Statistical Pattern Recognition

CSE 253

Complementary courses, different content:

Anything taught by Manmohan Chandraker, Hao Su, Kamalika Chaudhury, Sanjoy Dasgupta, Yoav Freund, Julian McAuley, Lawrence Saul, Zhuowen Tu (Cog Sci) or Nuno Vasconcelos (ECE)

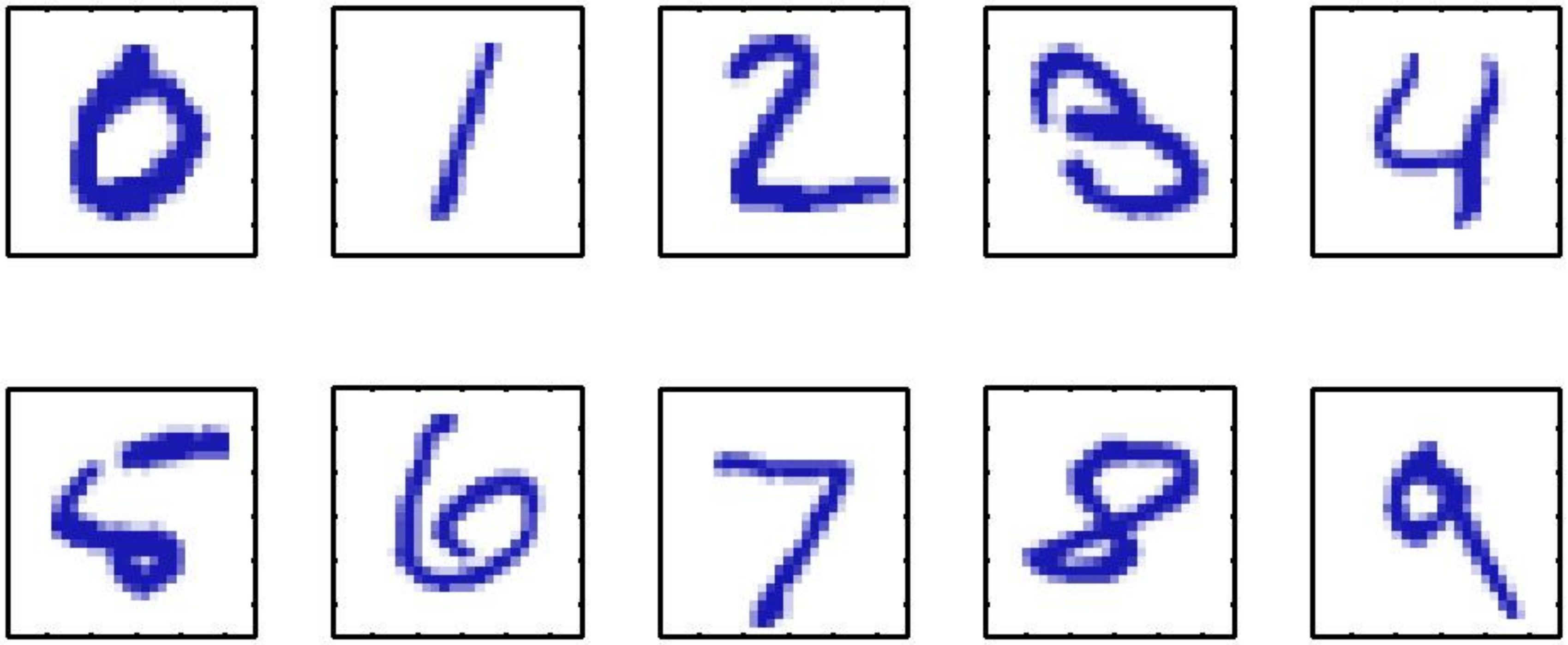
Related courses taught by

Virginia de Sa (CogSci), Ndapa Nakashole, Laurel Riek

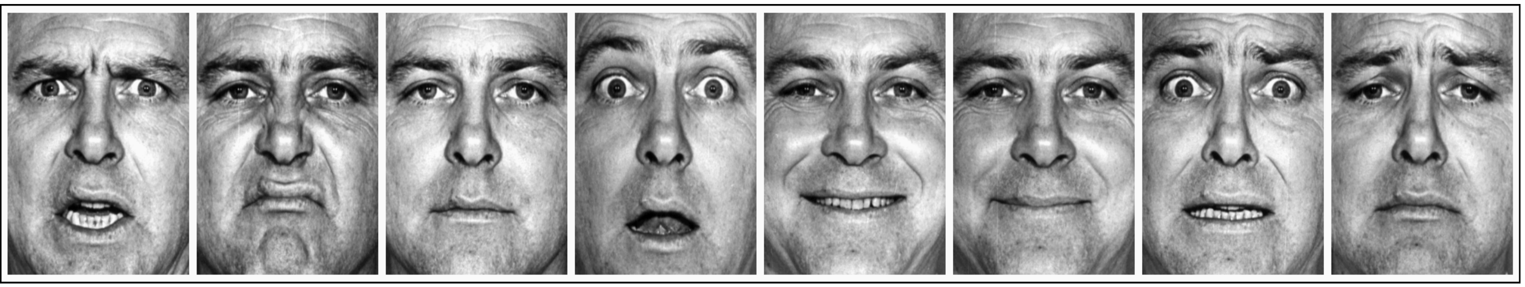
Outline of today

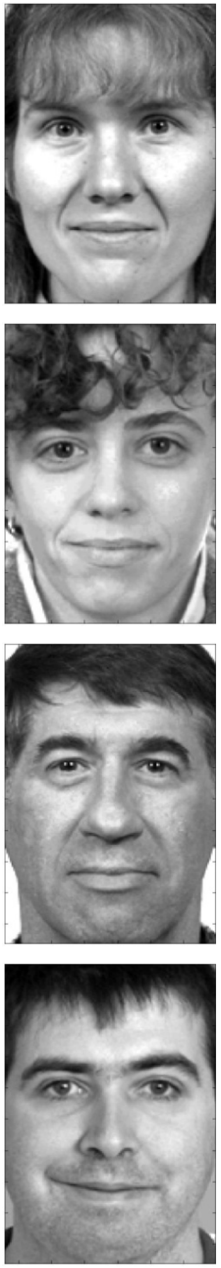
1. Pattern recognition: Some examples
2. Classification vs. Regression
3. Pre-processing and feature extraction
4. Polynomial curve fitting
5. Model complexity

The standard example: Handwritten digits (MNIST)

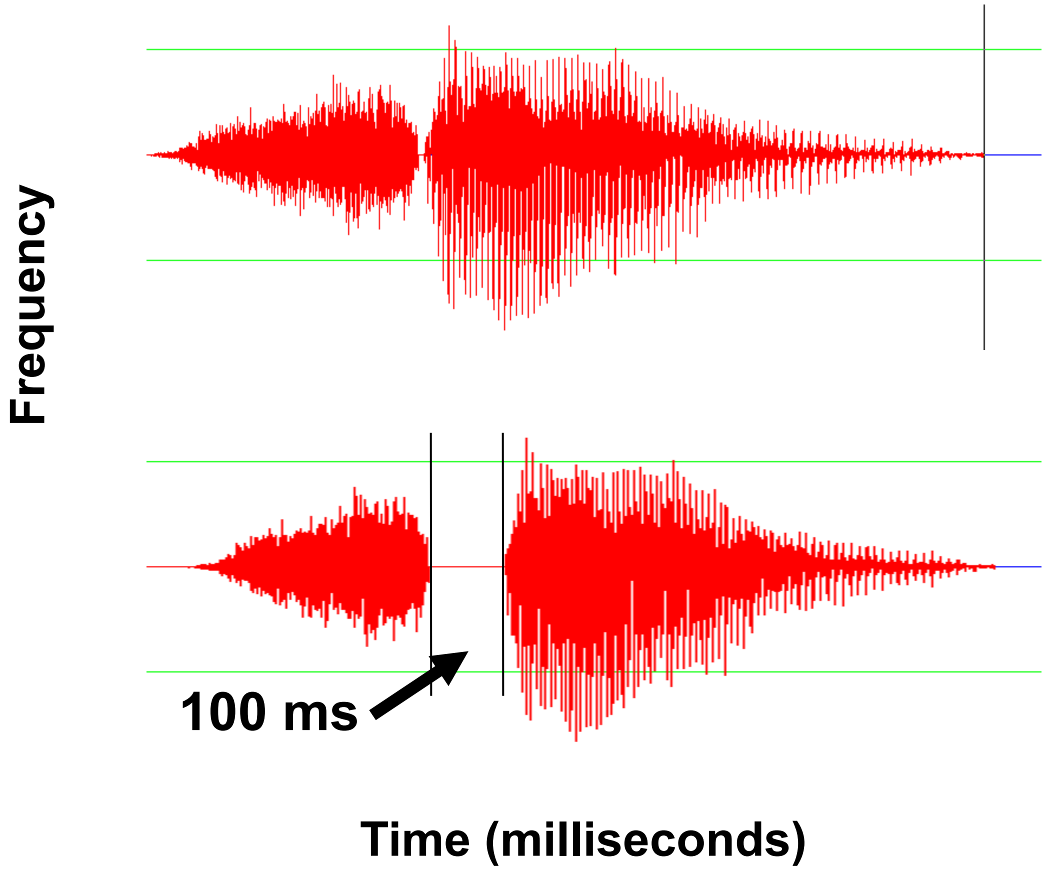


Facial Expression & Identity



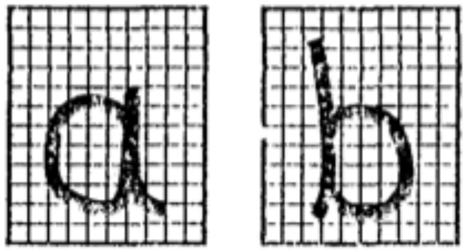


Speech recognition



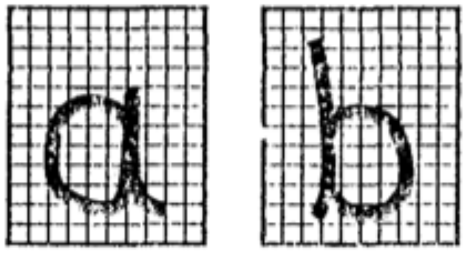
Requires subtle temporal information: Note these wave forms are nearly ***identical*** except for the artificially inserted gap!

A simpler example



- Distinguish handwritten "a"s from "b"s
  - Input: An array of *pixel values* (vector **x**)
  - Output: A *class label* Y_k, *k*=1, 2: often we use -1, 1
  - Training Set: A large number of input/output examples
  - Goal: Develop a *classifier* that assigns class labels to novel images with minimal mistakes. (i.e., a classifier that *generalizes* well)

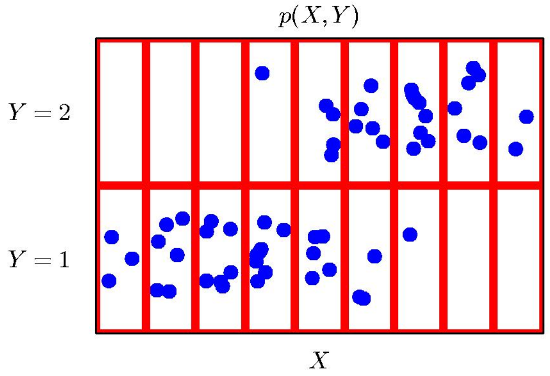
A simpler example



- Problem: High *dimensionality*:
    - Given a 256X256 image with 8 bit pixels =>
    - Inputs are of dimensionality 65,536 =>
    - There are approximately $10^{158,000}$ *possible* images
- This has been called the *curse of dimensionality*
- This implies we need to extract *features* that are of smaller dimension than the original images.
- Idea: Use ratio of height to width as a "feature". Call it x_1.

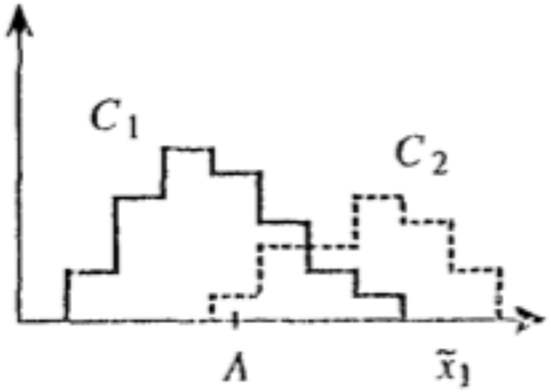
A simpler example

- Hypothetical distribution of values of x_1 from the training set:



- Classifer #1: Pick threshold A_1 that minimizes mistakes.

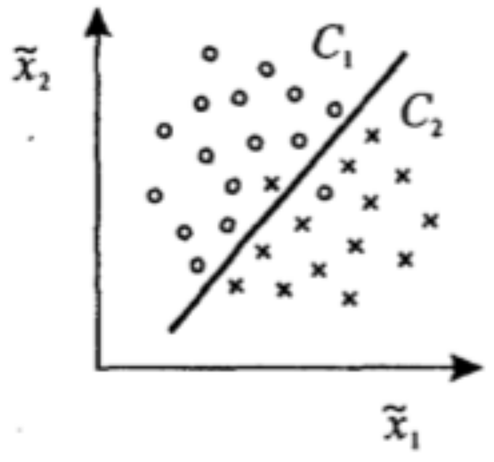
-



A simpler example

- Problem #2: Still will be a lot of mistakes!

- Idea 2: (we don't have a lot of ideas yet):
  - Add another feature. Call it x_2.
  - Now, in "feature space", we need a *decision boundary*, in this case a line:



A simpler example

- If we had to do all 26 letters, then we would need functions that give (for example) the probability of the category, $k = \{1,\ldots,c\}$, where c is the number of categories.

- We can do this by a set of k functions:

$$y_k = y_k(\mathbf{x}; \mathbf{w})$$

where the functions are parameterized by **w.**

The category decision is $k=\text{argmax}_j \, y_j(\mathbf{x};\mathbf{w})$ (i.e., the maximum y_k)

The curse of Dimensionality

Feature extraction is one way to deal with this curse:

- Feature extraction reduces the dimensionality of the data
- This reduces the size of the required training set
- Feature extraction recognizes that most data actually lies on a lower-dimensional manifold of the "ambient space."

Statistical Pattern Recognition: Classification vs. Regression

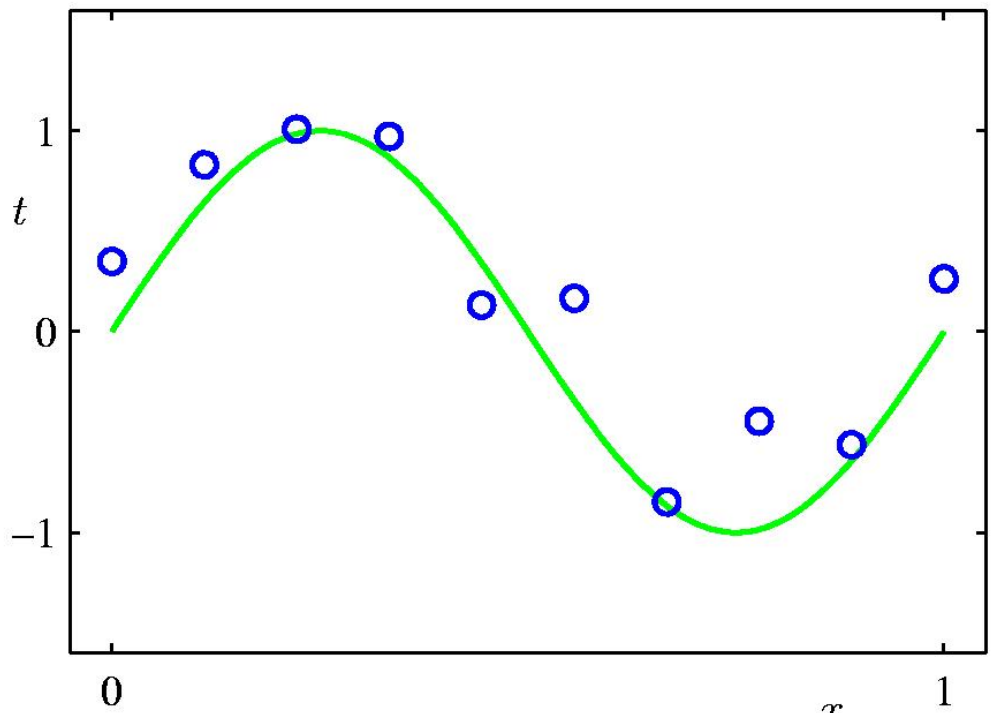
- These examples are *classification problems*
- This is in contrast to *regression problems*:
  - The output is the value of a *continuous variable*: exchange rates, amount of an emotion being expressed, etc.
  - Or: The average of a random quantity.
- Both of these (classification and regression) are examples of *function approximation*:
  - In classification, often we want the probability of class membership - a function approximation problem.

Statistical Pattern Recognition: Classification vs. Regression

- Usually, the data we are trying to approximate has:
  - Some underlying regularity
  - Some noise

- Often, the function we are using to approximate the data is not from the same class as the one that generated the data:
  - There will be a *loss*
  - There will be parameters we are trying to fit to minimize the loss.

A regression example: Polynomial Curve Fitting

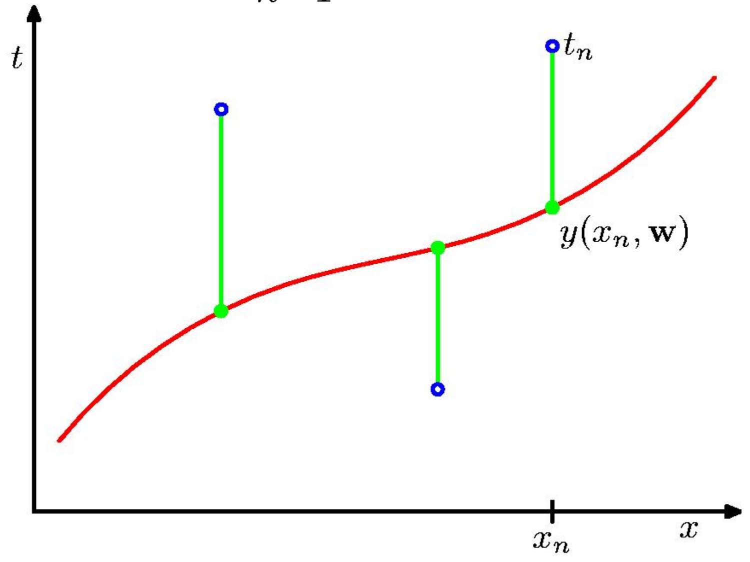


$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

Note: this is *linear* in the parameters *w:*
We can use linear regression techniques to solve for *w*

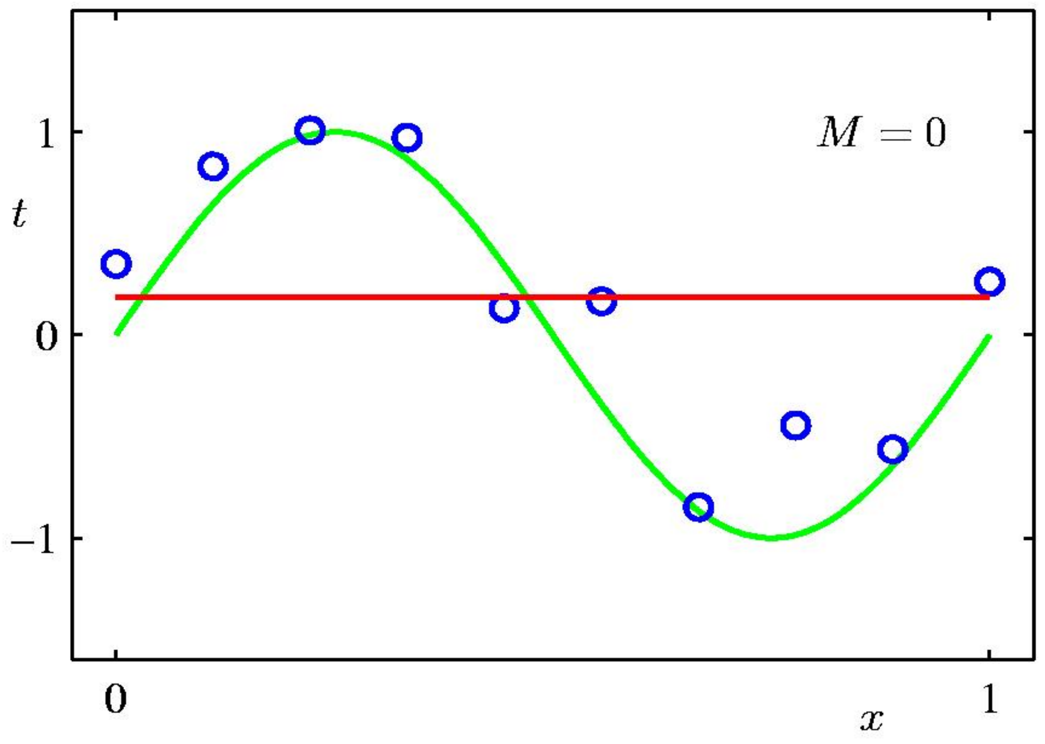
Now, we need a loss function: Sum-of-Squares Error Function

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

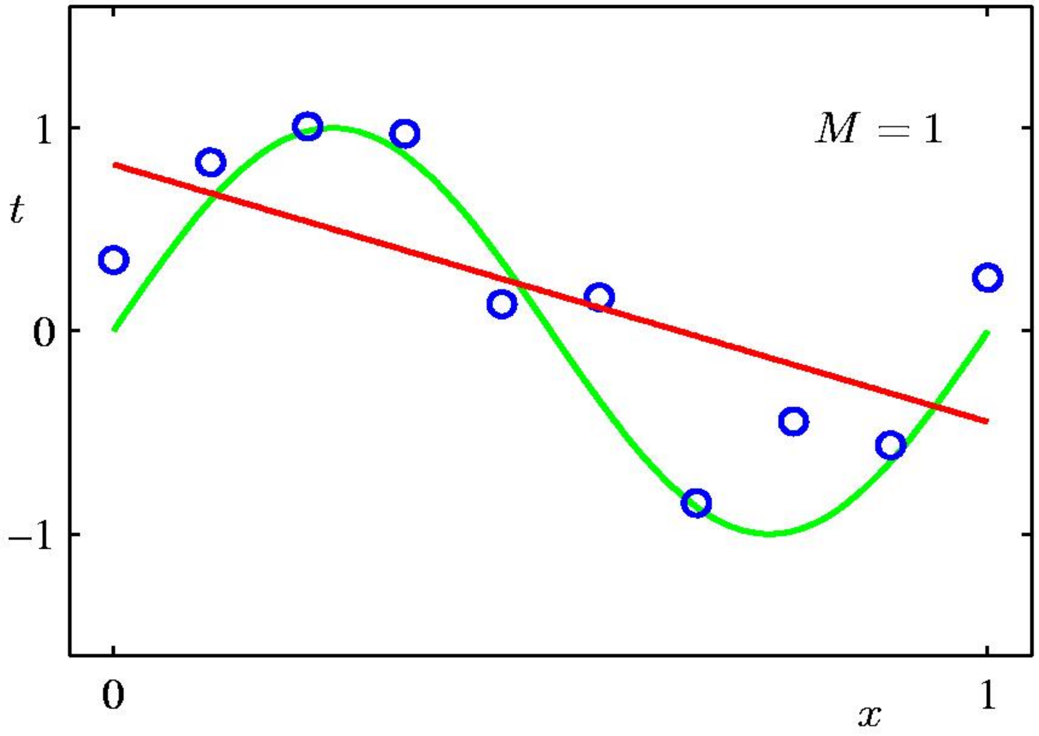


Note that what we want to do is minimize this:
take the derivative and set it to 0.
Since this is linear in the variables,
this leads to a closed form solution (more on that in the next lecture)

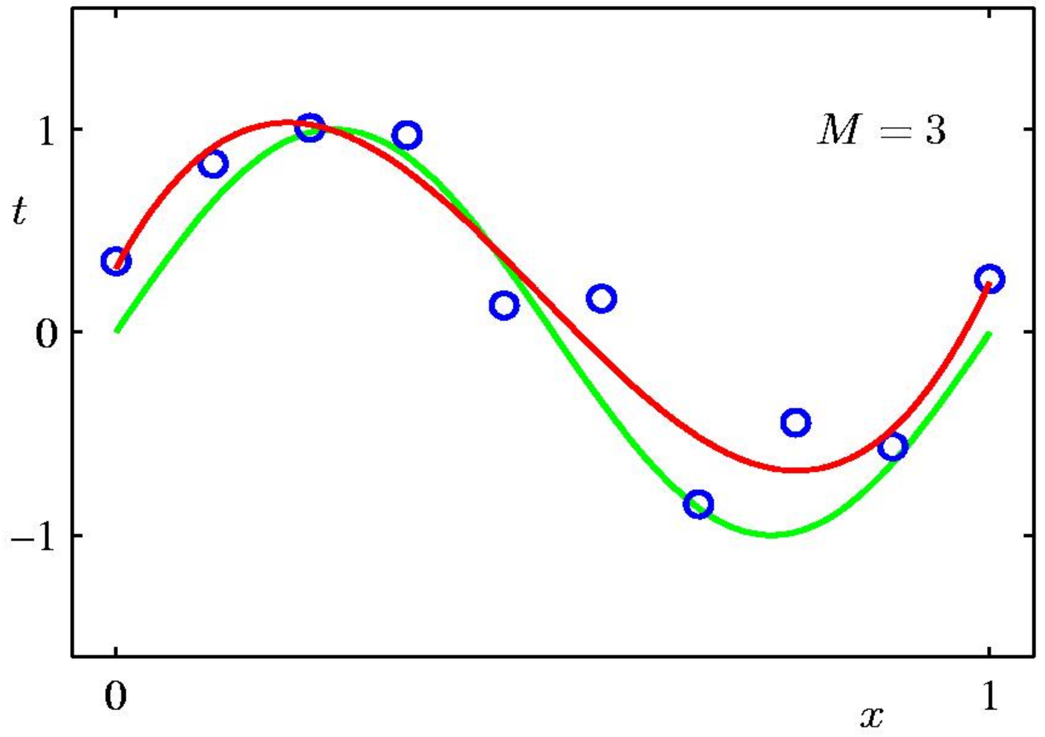
0th Order Polynomial



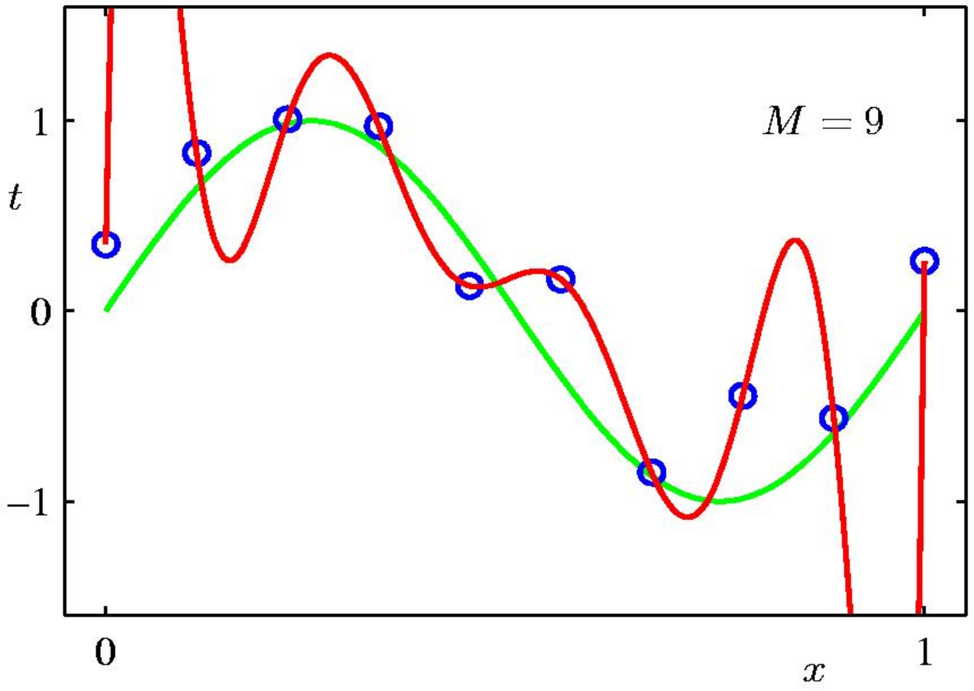
1st Order Polynomial



3rd Order Polynomial



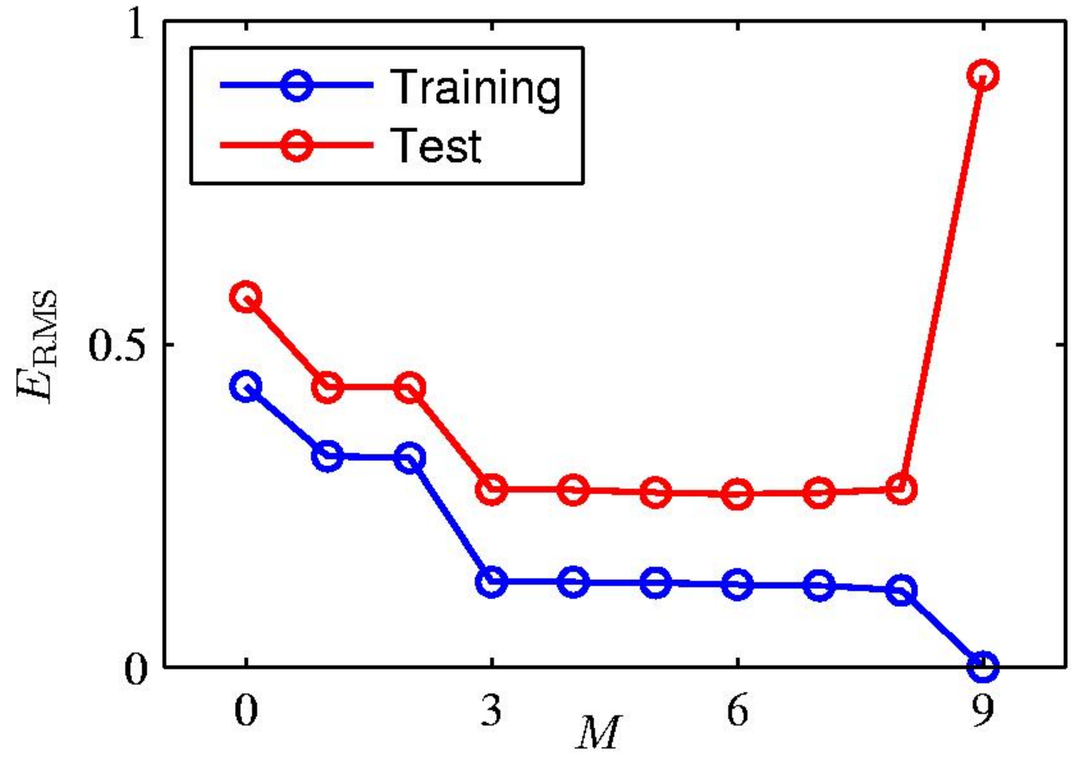
9th Order Polynomial



Note, we have fit the training data *perfectly*.

But we would like the solution to *generalize* to new examples, and here we are fitting the noise!

The problem: Over-fitting



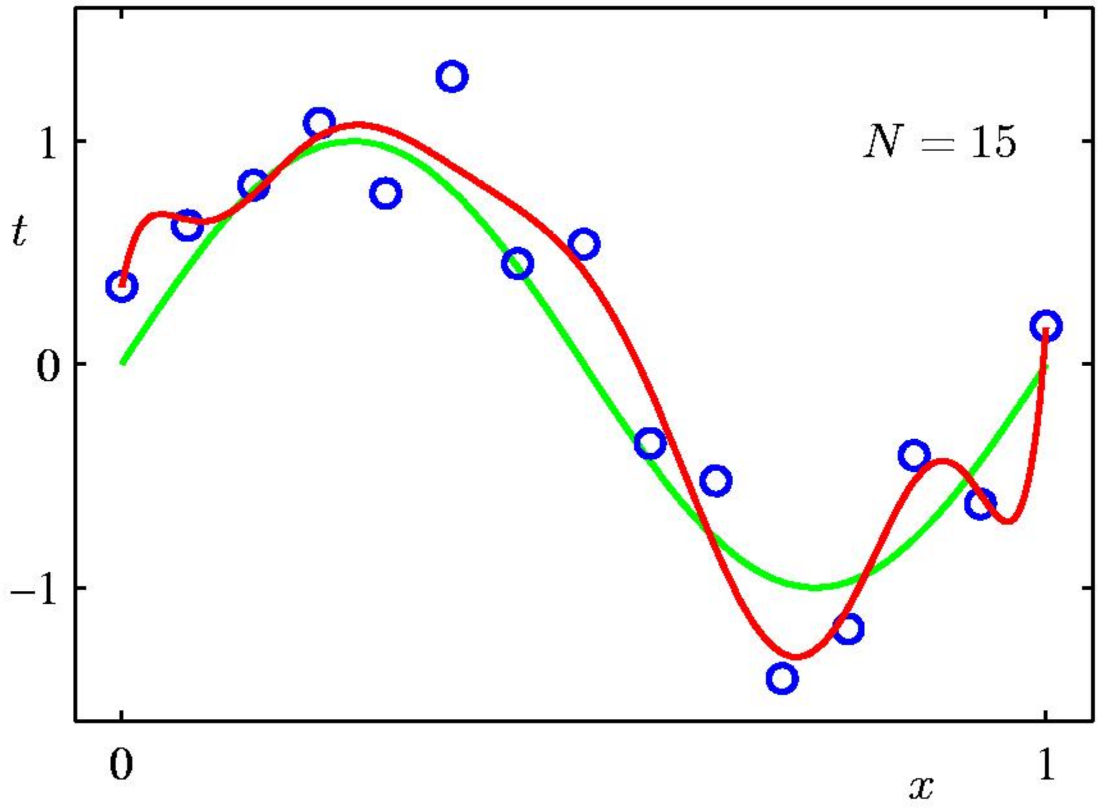
Root-Mean-Square (RMS) Error: $E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^\star)/N}$

The Polynomial Coefficients

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

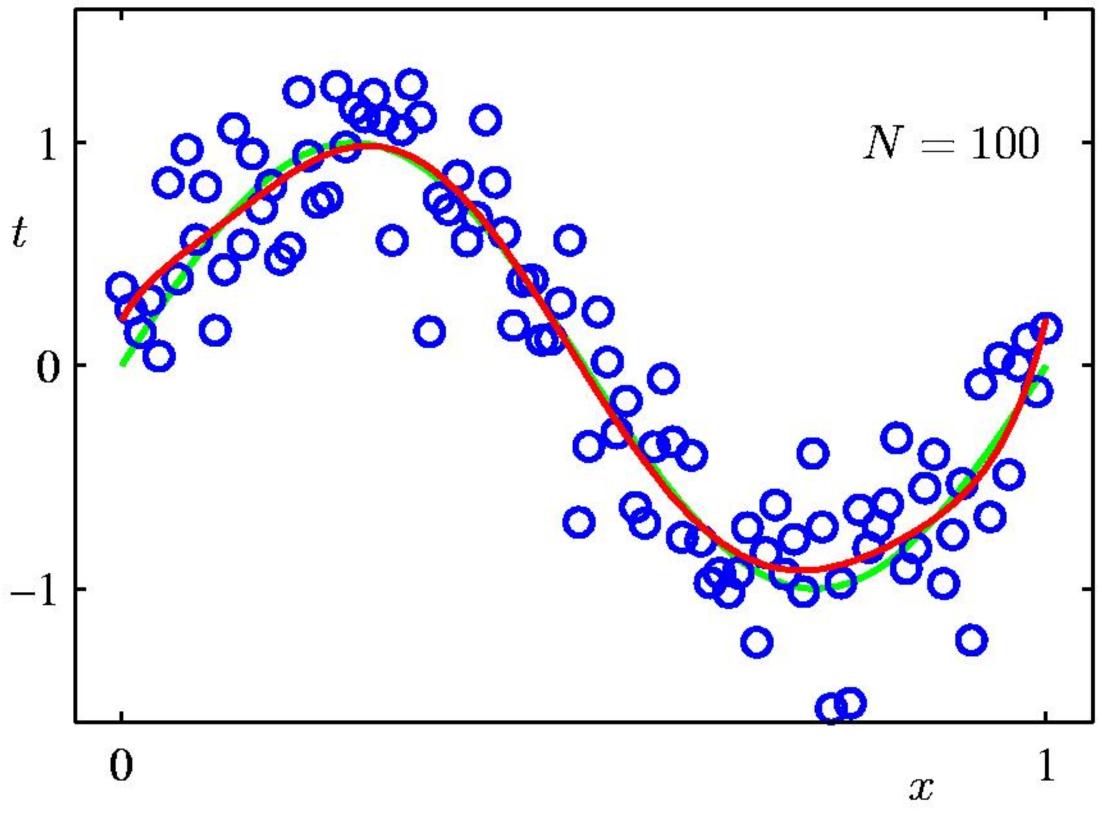
Effects of Data Set Size

9th Order Polynomial



Effects of Data Set Size

9th Order Polynomial



How to deal with overfitting?

- The best way: *Get more data!* (Or, manufacture more data...)
- Otherwise, to eliminate overfitting, one idea it to use “Occam’s (or Ockham’s) Razor”:
- The simplest hypothesis is the best.
- Applying this to a model, we change our objective function:
- Minimize $J=E+\lambda C$ where E is the error and C is a measure of model complexity.
- This is called *regularization.*
- In neural networks, for example, we might use:
- $C = w^2/ (w^2+1)$
- (Penalizes big weights very little, small weights are eliminated.)

How do we prevent overfitting? Regularization

- Penalize large coefficient values

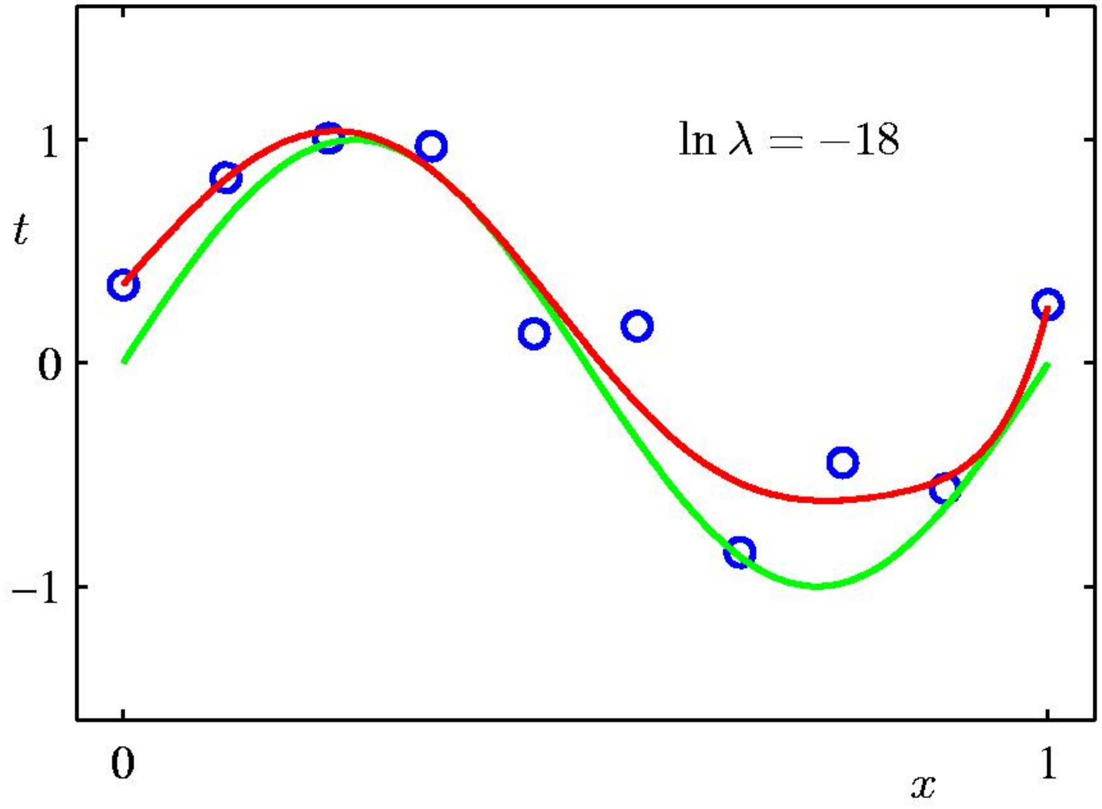
$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

Regularization:

(9th order polynomial)

$$\ln \lambda = -18$$

(.00000015)

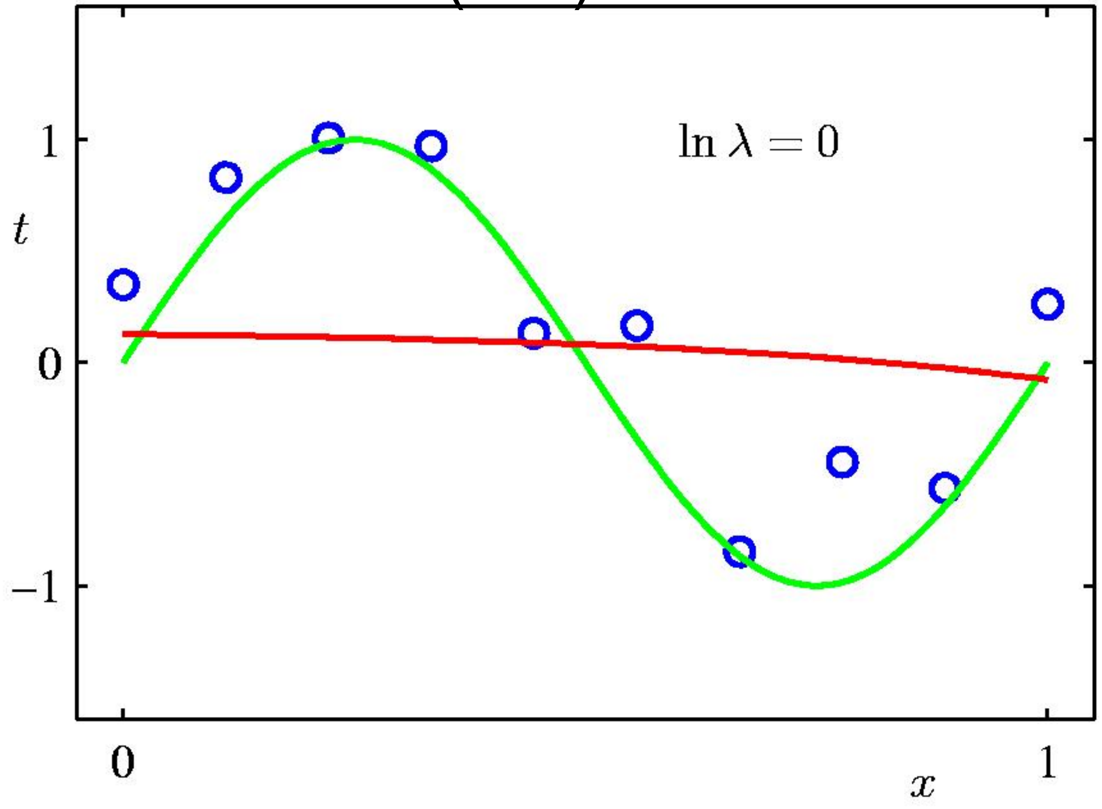


Regularization:

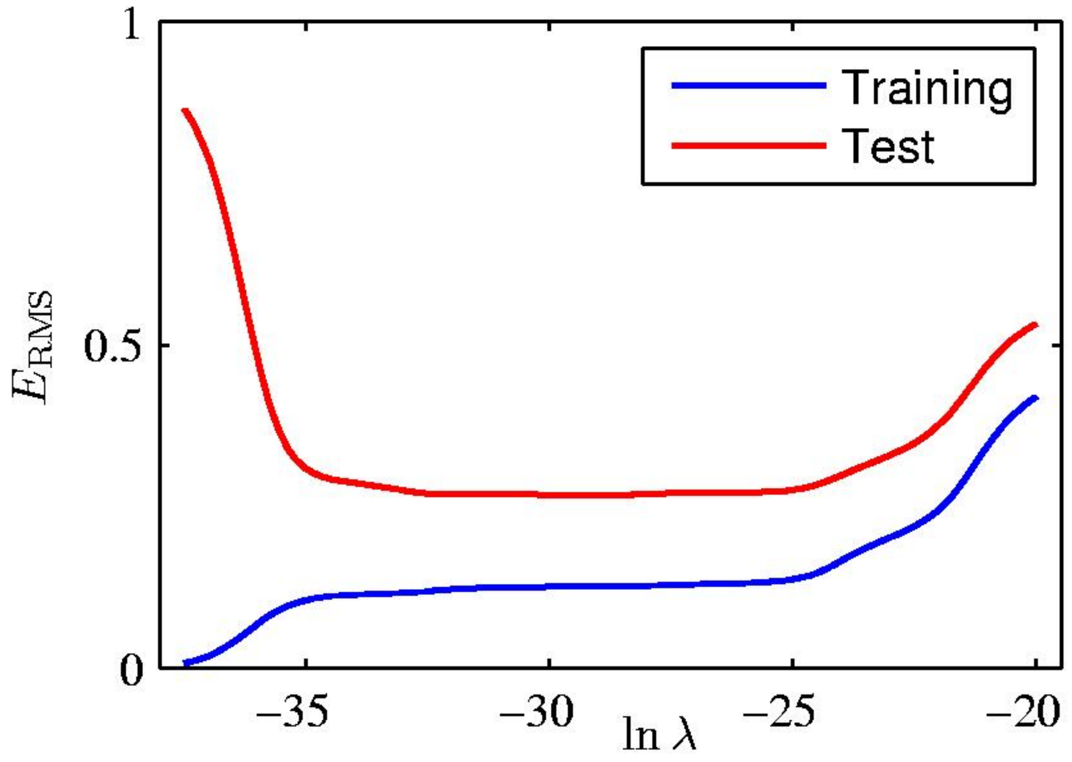
(9th order polynomial)

$\ln \lambda = 0$

(λ=1)



Regularization: E_{RMS} vs. $\ln \lambda$



Polynomial Coefficients

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

How to deal with overfitting?

- The best way: *Get more data!* (Or, manufacture more data...)
- Minimize $J=E+\lambda C$ where E is the error and C is a measure of model complexity (*regularization*).
- Early stopping:
  - Have a hold out set (some fraction of the training set) – this is a stand-in for the unseen test set
  - Use the remaining portion of the training set to change the weights
  - Watch the error on the holdout set and stop when it starts to rise.

Statistical Pattern Recognition: Pre-processing and feature extraction

- Pattern classsification is usually a multi-step process:
  - Preprocessing (normalization, feature extraction)
  - processing by the neural network
  - possible post-processing of the network output.

Statistical Pattern Recognition: Pre-processing and feature extraction

- For example, in face recognition, in the bad old days, before deep nets, we first:
  - Normalize the brightness (and possibly variance) of the image.
  - Find particular feature points in the image (mouth, eyes)
  - Normalize these features to fixed positions (translation, size invariance)
  - Crop the image (eliminate irrelevant features)
  - Extract features from the resulting image for processing by the neural network (e.g., Gabor filtering followed by PCA).
  - Use a linear classifier on the result (e.g., softmax regression)
- This is what computer vision did for the 50 years or so before 2012.

Statistical Pattern Recognition: Pre-processing and feature extraction

- Nowadays, with deep nets, we might just subtract the mean of the pixels, and divide by the standard deviation (*z-scoring*)

- Or just subtract the mean over the whole dataset.

- Then we let the deep network *learn* the features.

Final Points/Summary

- Neural networks are function approximators that can do both classification and regression: This class focuses on classification
- Logistic regression and linear regression are examples of very simple neural networks
- Regression is trying to fit a function as closely as possible, while generalizing well to new points.
- Classifiers create *decision boundaries,* such as a line, that separate classes in a feature space.
- Pattern recognition involves
  - Extracting features from the input to reduce dimensionality
  - The resulting function returns a value that could be
    - A real number in the case of regression
    - A probability of being in a category, or a hard decision, in the case of classification

Final Points/Summary

- Typical confusion:

  Logistic regression is called *regression* because you are fitting a continuous function that is the probability of being in a category – so we use it for *classification*!

Final Points/Summary

- Since there is usually noisy data, we will make errors. Our goal is to minimize them.
- We achieve this by defining an objective function (a.k.a. a loss or cost function), such as sum-squared error that when minimized, minimizes errors in some sense.
- However, we can't just minimize errors on the training set! This can lead to overfitting (as in the case of the 9th order polynomial, which reduces errors to 0 on the training set).
- To deal with overfitting, we can
  - Get more data
  - Penalize complexity of our function approximator (regularization)
  - Use a hold-out set as a stand-in for the unseen test set, and (for example) choose the complexity of our model based on it.