Learning Algorithms

NEUR 531-603 Introduction to Computational Neuroscience

Curtis Baker

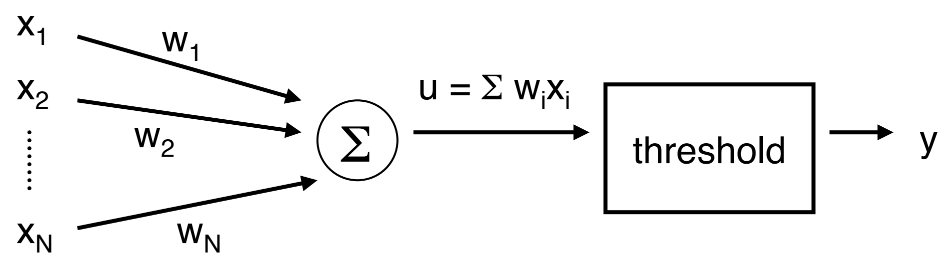
Primary Reading: Information Theory, Inference, and Learning Algorithms, by David MacKay
mainly chapter 39; also some chapters 38, 44

This book is freely available to download:
http://www.inference.phy.cam.ac.uk/mackay/itila/book.html

Individual chapters:
http://www.inference.phy.cam.ac.uk/mackay/itprnn/ps/

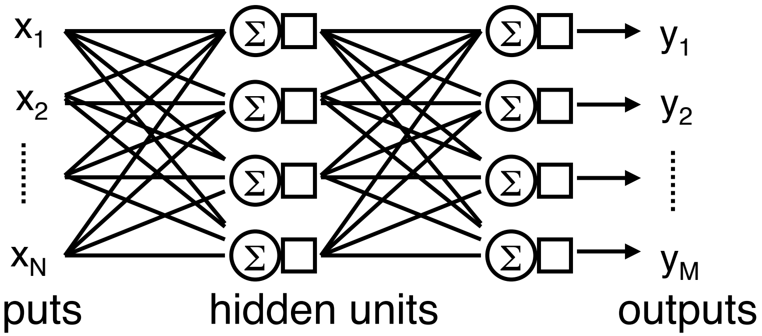
Neural Networks - a brief history

1950s-60s: McCollough-Pitts neuron; "feature-detector" neurons in optic tectum, A17



1960s-70s: Rosenblatt Perceptron: architecture (single-layer)
novelty at the time: learning; distributed memory; neural inspiration
Minsky & Papert critique, difficulty with multi-layer networks

1980s-90s: revenge of the neural networkers: back-prop, connectionism, etc
concurrent influences: neural plasticity, NMDA receptors;
Donald Hebb; David Marr; Rumelhart, Hinton, Sejnowski



90s, 00s: rise of the machines: machine learning (neural or not)
probabilistic models, statistical learning theory

Neural Networks

audiences

<u>cognitive science</u> - connectionist models

<u>neural network modeling</u> - from loose metaphors, to specific models
(e.g., development of ocular domiance stripes in visual cortex),
to a theoretical endeavour in its own right

<u>computer science</u>: robotics / machine intelligence, computer vision

example applications of "machine learning"

pattern recognition
- handwriting, fingerprints, faces, license plates

decoding
- "mind-reading" with fMRI

model parameter-fitting
- system identification

finding patterns in data
- cluster analysis
- data mining
- dimensionality reduction / data compression

Types of learning algorithms

1. supervised - *data = inputs, targets (from a "teacher")*
classification / recognition
output is discrete / categorical: e.g., Rosenblatt Perceptron

regression / function approximation
output is analog / functional: e.g., parameter-fitting

2. unsupervised - *data = set of multivariate values (without any "teacher")*
density estimation - clustering, EM algorithm, mixture of Gaussians

efficient coding of natural sensory info
decorrelation, PCA, ICA

3. reinforcement
output is an "action", optimized to maximize a "reward"
no access to examples of optimal or correct responses
instead, "agent" must *discover* them

common concepts throughout:
can often (optionally) be cast as "neural networks"
can often use as metaphorical neural models, *or* as data analysis tools
optimization algorithms - minimize an "error function" or "objective function"

Classification / recognition

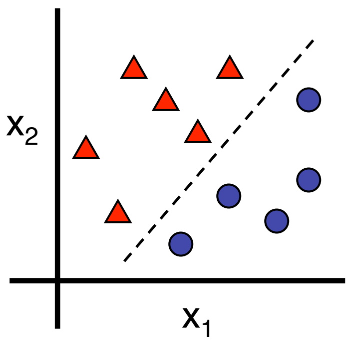
classify stimuli into categories

must be robust to variability in stimuli

dataset:

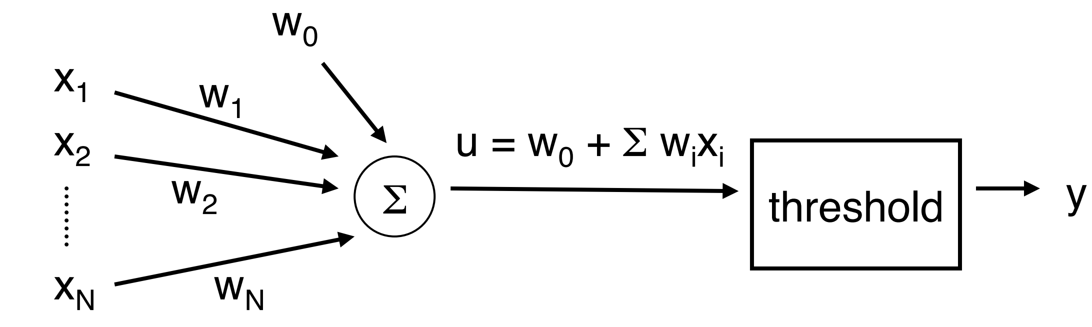
inputs (x's), e.g. photoreceptors / pixels, or higher-level features / receptive fields
corresponding classifications (T's, i.e. the "teacher")

Rosenblatt Perceptron - an early "binary classifier" -> classify into 2 categories

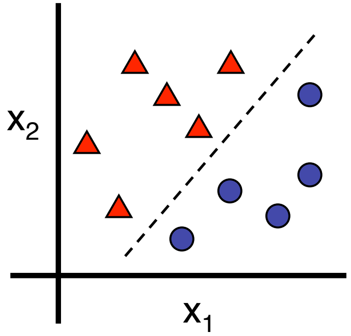


Classification

Architecture: variables, and relationships between them



inputs weights



activity rule: $y = +1$ if $u > 0$
$= -1$ if $u < 0$

learning (up-date) rule: if correct (y=T):

$$\Delta w_i = \eta x_i$$
$$\Delta w_0 = \eta$$

if incorrect (y !=T):

$$\Delta w_i = 0$$

← Feedback from teacher

T = "teacher"

η ("eta") = learning rate parameter

notes: weights initially random
sequential / on-line / stochastic mode
sensitive to η

problems: does not always converge; poor generalization; ad hoc (no underlying theory)

Not generalizable

(DEMO: Rosenblatt Perceptron)

Classification with gradient descent

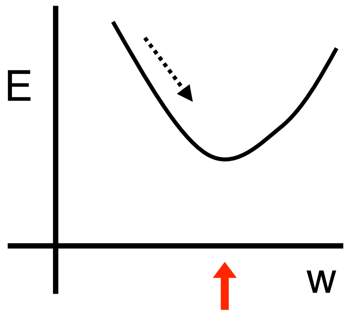
principle: specify an **error function**, which algorithm should try to minimize
(in general, an "**objective function**")

dataset:

inputs (x' s), e.g. photoreceptors / pixels, or higher-level features / receptive fields
corresponding classifications (T' s, i.e. the "teacher")

<u>gradient descent</u>:

Error



min: $\delta E / \delta w = 0$ -> $dw_i = -\eta \, \delta E / \delta w_i$

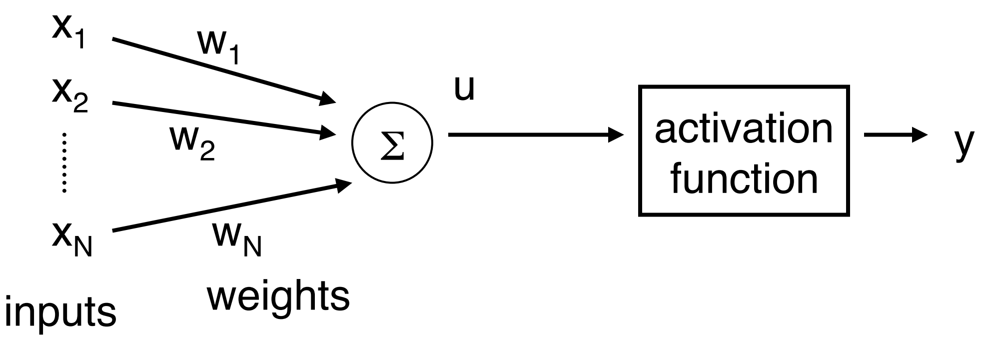
Slope

η ("eta") = learning rate / "step size"

E = error (objective) function -> gradient

Classification with LMS gradient descent

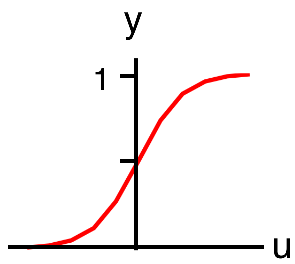
architecture



objective function: $E = 1/2\ \Sigma\ (T_j - y_j)^2$ "least mean squares"

where y_j = network response, on trial j
T_j = "teacher", i.e. desired or correct respose

(for probabilistic model, with Gaussian noise, this objective function is optimal) ?



activation function: must be differentiable
-> logistic (sigmoid): $y(u) = 1/(1 + e^{-u})$

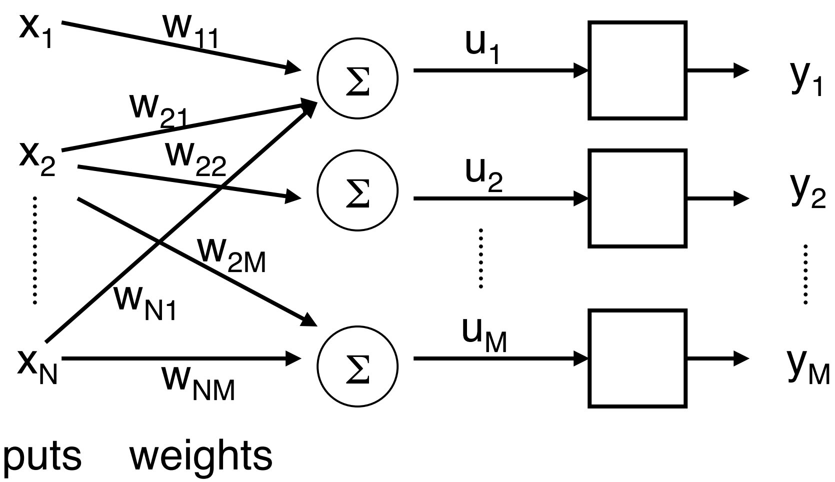
-> **learning rule**: $dw_i = -\eta\ \Sigma\ (T_j - y_j)\ x_i$

notes: for linear model, will always converge to unique minimum
best run in <u>batch</u> mode
problems: only gives good classification if categories are <u>linearly separable</u>

(DEMO: binary classifier with gradient descent)

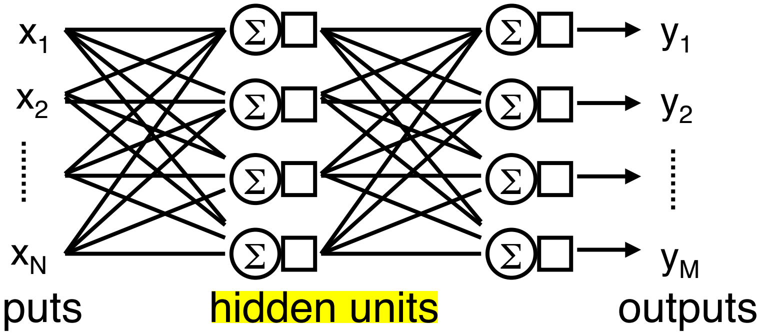
Classification: extensions

more than two categories: **multi-output networks**



outputs - e.g. liklihoods for each category

beyond linear separability: **multi-layer networks**



learning rule:
back-propagation

<u>problems</u>: multiple minima
how many hidden units ? too many -> *over-fitting*

Regularization

over-fitting problem:

weights eventually diverge to very large values,
giving only small improvements to error,
while degrading ability to generalize to new data

-> see MacKay, section 39.4 and Figure 39.5, 39.6 (-> Assignment)

regularization: modifiy error function, to place a "penalty" on large weight values
sometimes called "weight decay"

$$E = 1/2 \Sigma (T_j - y_j)^2 + \alpha \Sigma w_i^2$$

For decay

α = hyperparameter (not part of "activity rule" or model architecture -
it is a parameter of the learning algorithm)

Regression / function approximation

general regression problem: **y** = **f** (**x**)

given input vector, x, and vector of outputs, y,

-> find mapping function, **f**

compare to classification:
outputs are analog, not categorical

"teacher": try to optimize prediction of y-values

applications:
find best-fitting parameters of a model
system identification

linear regression: **y** = **w** · **x**

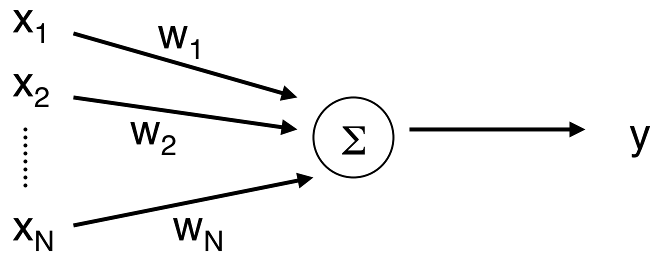
dataset: **x** = inputs: $x_1, x_2, ...x_N$
y = outputs: $y_1, y_2, ... y_M$

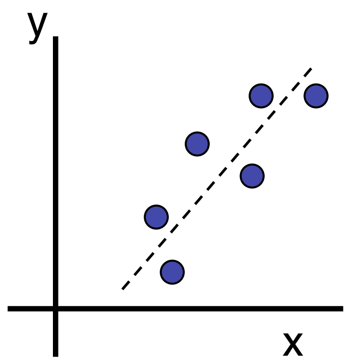
-> find **w** = weights: $w_{11}, w_{12}, ... w_{1M}, ... w_{N1}, ...w_{NM}$

Linear Regression

simplest case, single output: $y = \Sigma\, w_i x_i$

architecture





error function:

$$E = \Sigma\, (y_i - \text{predicted } y_i)^2$$

gradient descent:

min: $\delta E / \delta w = 0$ -> $dw_i = -\eta\, \delta E / \delta w_i$

η ("eta") = learning rate parameter

learning rule:

$$dw_i = \eta\, (y_i - \text{predicted } y_i)\, x_i$$

a "*general linear model*" (GLM) – guaranteed to have a unique minimum ?

(DEMO: linear regression with gradient descent)

Regression: extensions

faster algorithms to find optimum

- e.g. scaled conjugate gradient

basis functions (“features”) on front-end, e.g.:

Gaussian weightings
Gabor wavelets

regularization

Regularization

over-fitting problem: weights eventually diverge to very large values, giving only small improvements to error, while degrading ability to generalize to new data

simple example: fitting a *high*-order polynomial to a *small* number of data points

-> curve fits those few points extremely well,
but that curve-fit handles *new* points very poorly

-> general issue of model complexity

regularization: modifiy error function, to place a "penalty" on large weight values

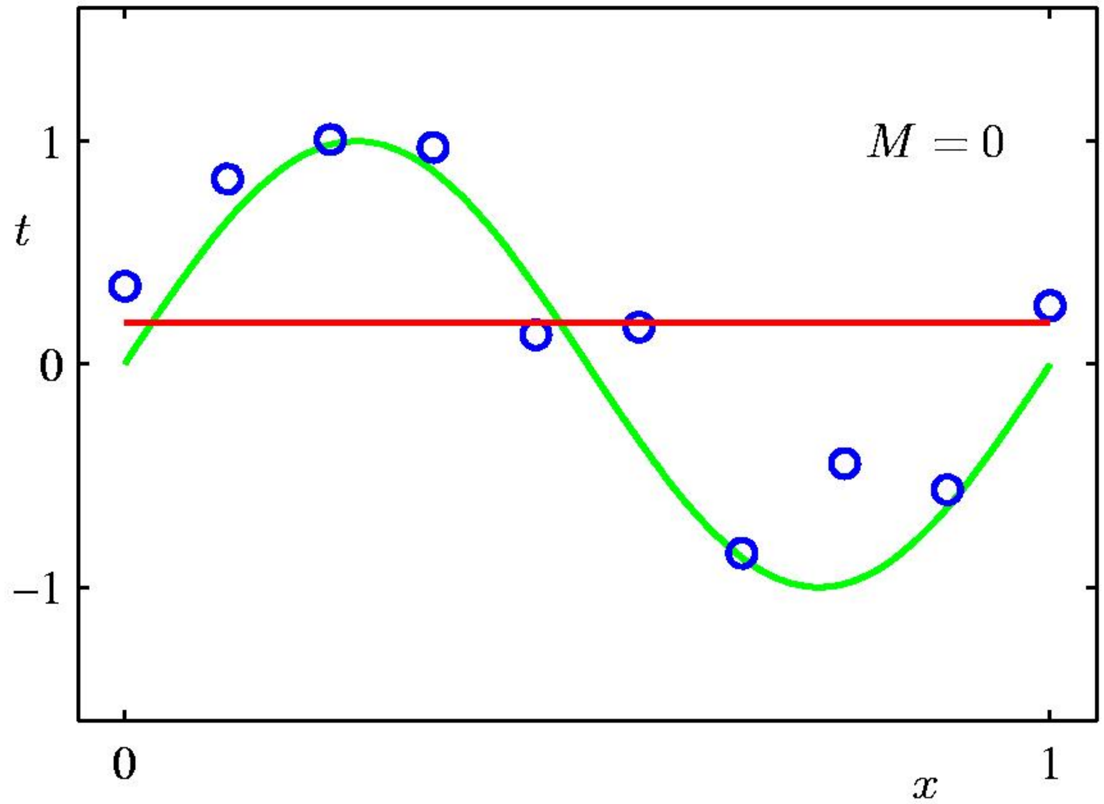
$$E = \Sigma (y_i - \text{predicted } y_i)^2 + \alpha \Sigma w_i^2$$

α = hyperparameter

sometimes called "ridge regression"

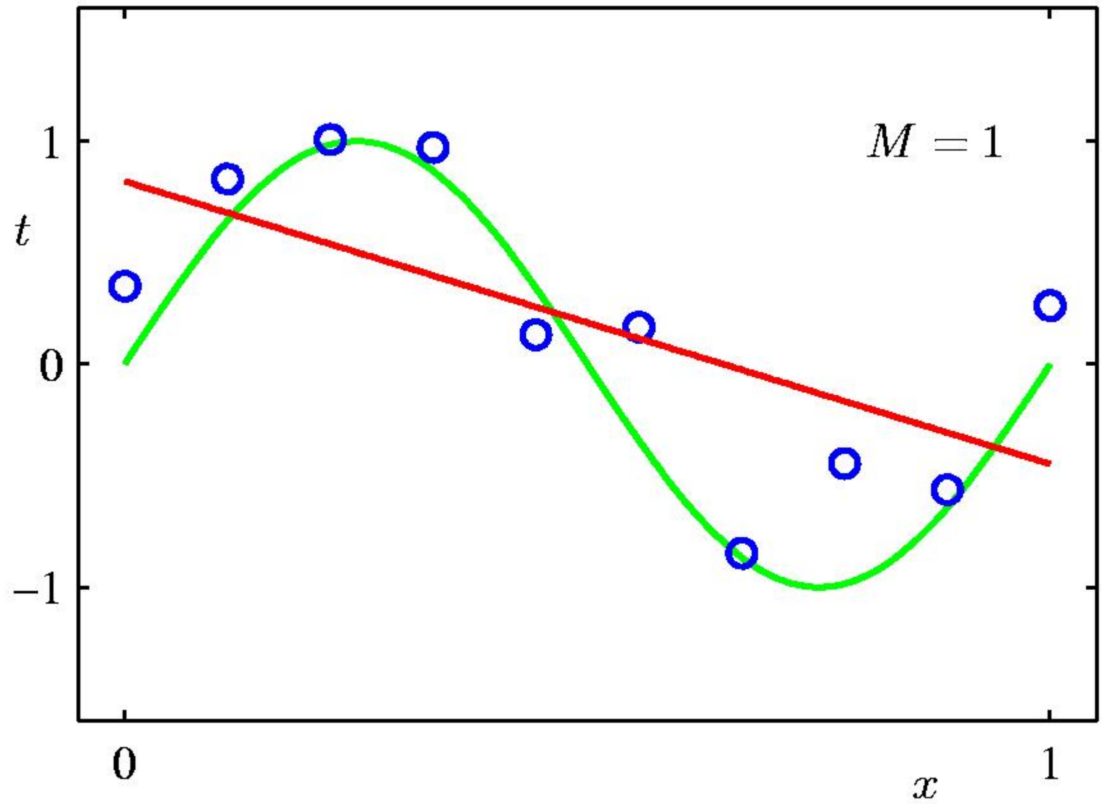
what value of α to use ? -> find best α, to best predict a "hold-back" dataset

0^{th} Order Polynomial

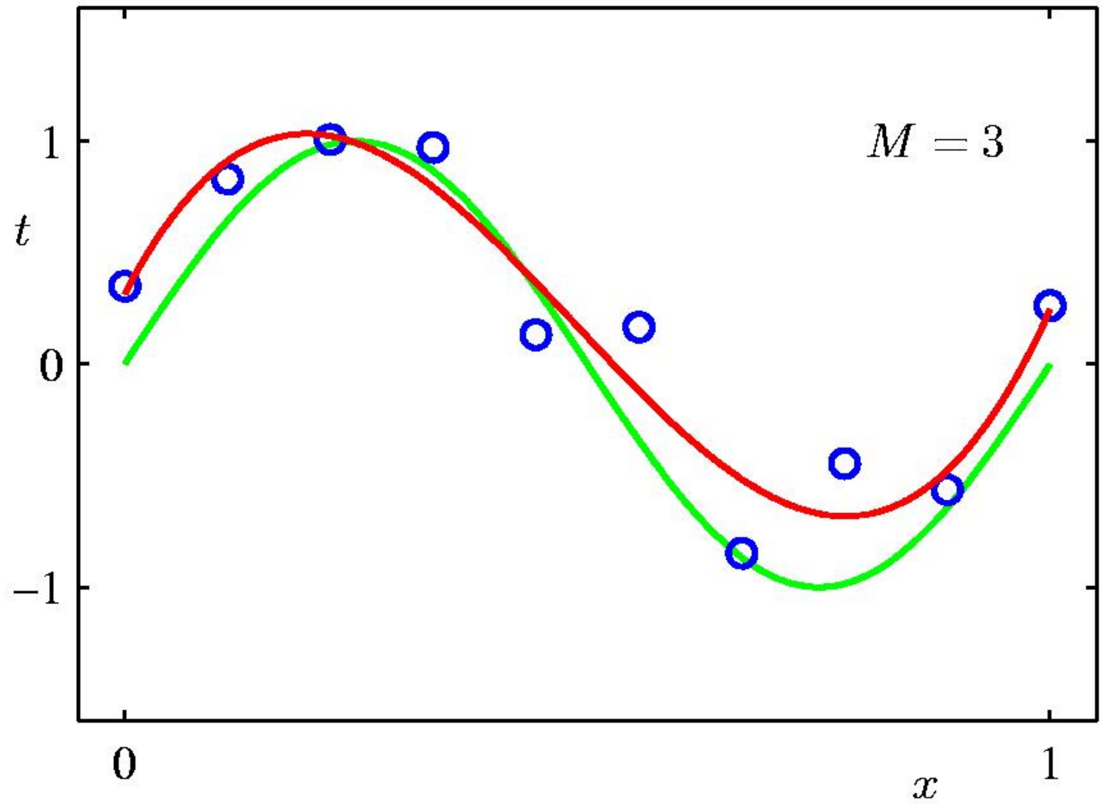


this and following slides, adapted from Bishop (2006), Fig.s 1.4 - 1.6

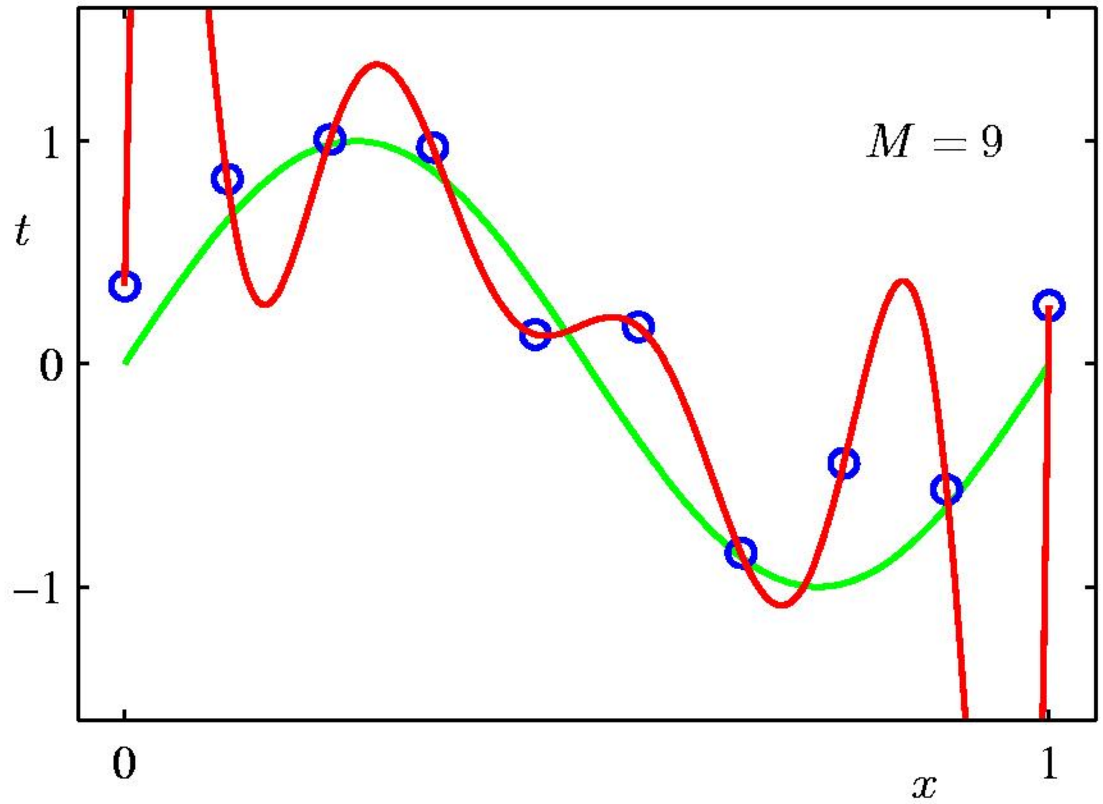
1^{st} Order Polynomial



3rd Order Polynomial



9th Order Polynomial



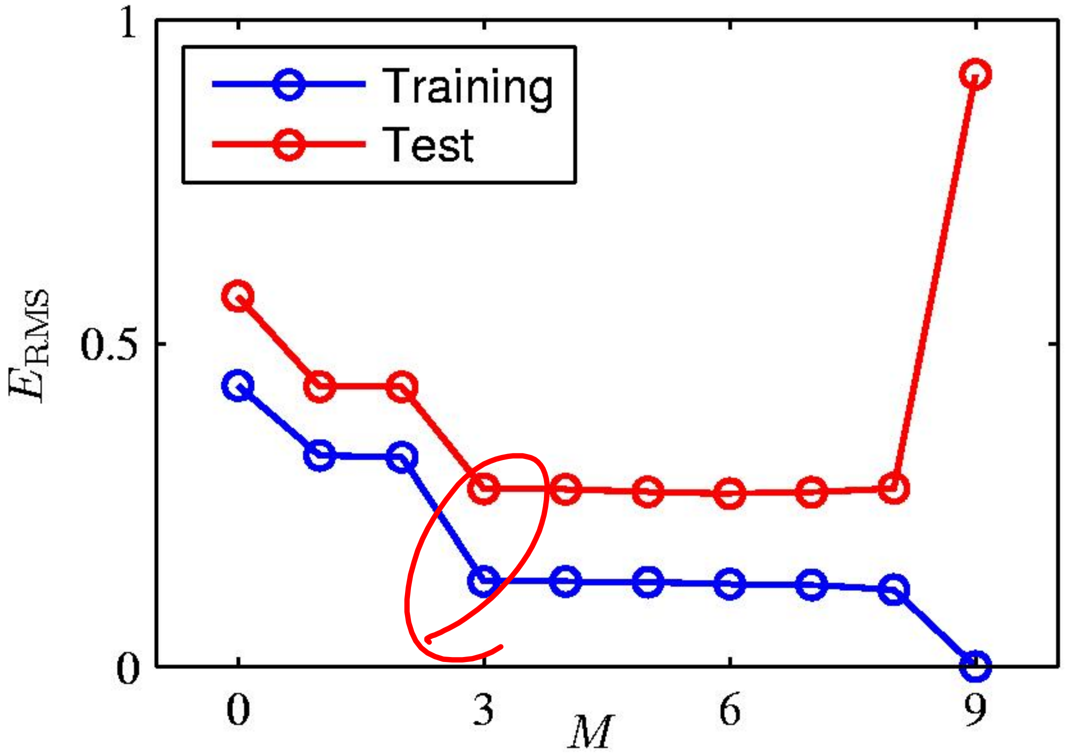
-> over-fitting !

Over-fitting

how to detect over-fitting:

see how well we can predict an independent ("hold-back") test dataset:

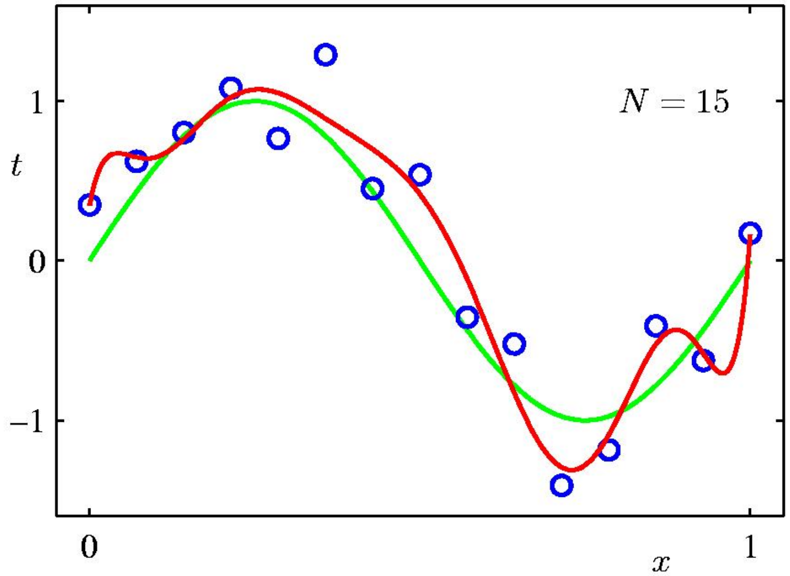
Error

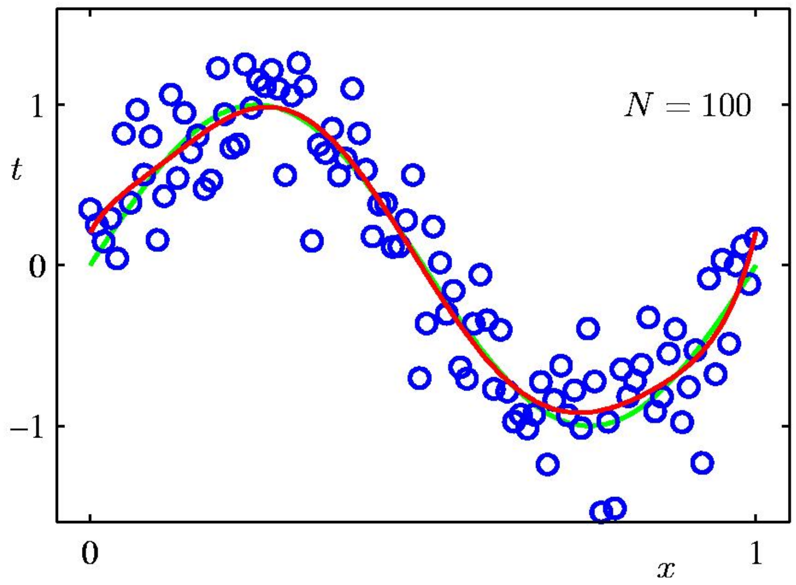


Root-Mean-Square (RMS) Error: $E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^\star)/N}$

Data Set Size:

M = 9,   i.e. fit 9th order polynomial

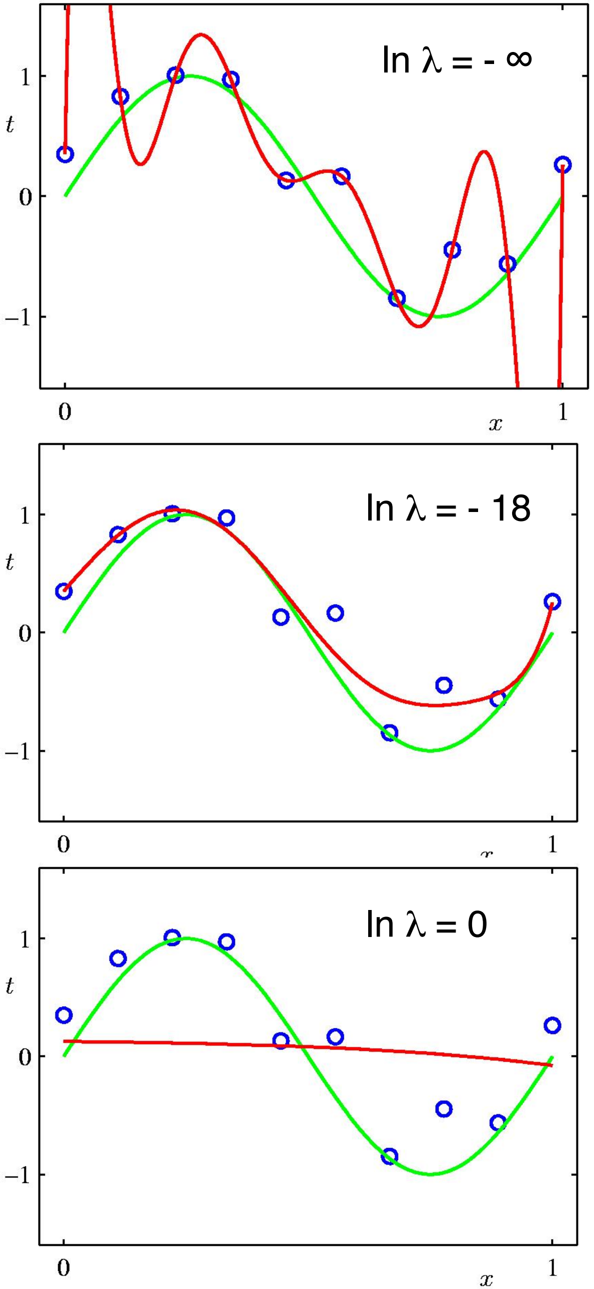




With more data, over-fitting becomes less of a problem.

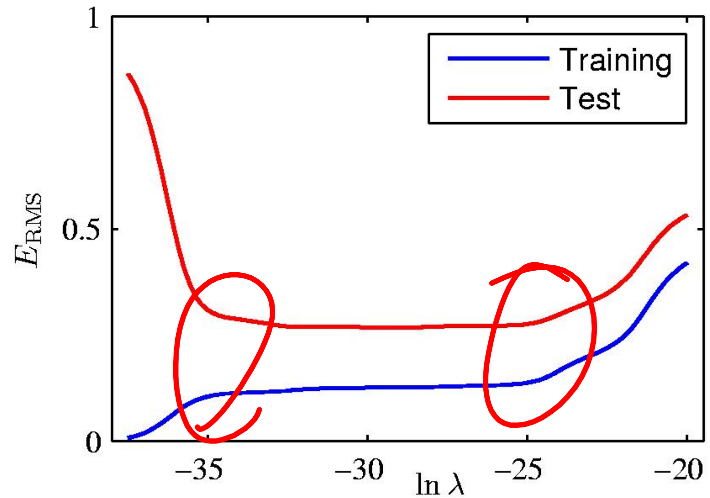
But, gathering more data can sometimes be difficult or expensive ...

Regularization



penalize large coefficient values:

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



adapted from Bishop (2006), Fig.s 1.4, 1.7, 1.8

(DEMO: system identification with gradient descent)

Optional supplementary reading:

Theoretical Neuroscience   Peter Dayan & L.F. Abbott;   MIT Press, 2001
Chapter 8: Plasticity and Learning - e.g., 8.4, "Supervised Learning"
Chapter 10: Representational Learning

Pattern Recognition and Machine Learning   Christopher Bishop
suppl materials ->   http://research.microsoft.com/en-us/um/people/cmbishop/PRML/index.htm

journal articles:

•Wu MCK, David SV, Gallant JL (2006) Complete functional characterization of sensory neurons by system identification. Ann Rev Neurosci 29:477-505.

•Nishimoto S, Gallant JL (2011) A three-dimensional spatiotemporal receptive field model explains responses of area MT neurons to naturalistic movies. J Neurosci 31:14551-14564.

•Talebi V, Baker CL (2012) Natural versus synthetic stimuli for estimating receptive field models: A comparison of predictive robustness. J Neurosci 32:1560-1576.

related courses at McGill: “Machine Learning” (COMP-652) - Doina Precup

On-line resources:
NetLab - http://www.ncrg.aston.ac.uk/netlab/
STRFlab - http://strflab.berkeley.edu/
Bruno Olshausen's course (UCBerkeley): redwood.berkeley.edu/wiki/VS298:_Neural_Computation
Michael Jordan's course (UCBerkeley): www.cs.berkeley.edu/%7Easimma/294-fall06/
Andrew Moore's slides (CMU):   www.autonlab.org/tutorials/
Wikipedia pages: Neural Networks, Machine Learning, ...