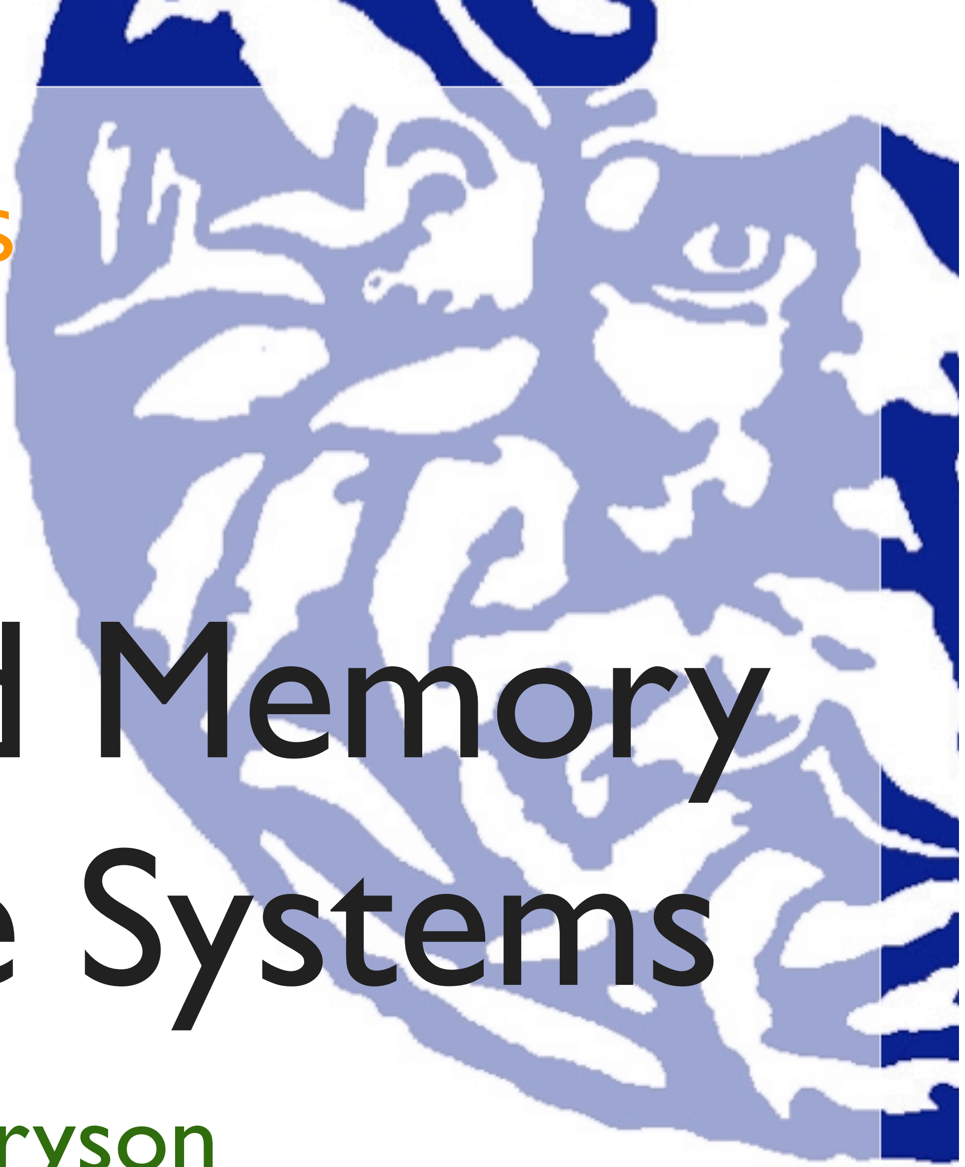
Intelligent Control and Cognitive Systems

brings you...



Learning and Memory in Cognitive Systems

Joanna J. Bryson

University of Bath, United Kingdom

Sensing vs Perception

- First week: Sensing – what information comes in.
- This week: Perception – what you think is going on.
  - Perception includes expectations.
  - Necessary for disambiguating noisy and impoverished sensory information.

Bayes' Theorem

"expectations"

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)}$$

$$p(X) = \sum_Y p(X|Y)p(Y)$$

posterior ∝ likelihood × prior

Given you've seen X, you can figure out if Y is likely true based on what you already know about the probability of experiencing: X independently, Y independently and X when you see Y.

One Application...

note to JB: copy X,Y to board,
useful later

- Y – potential action
- X – sensing
- priors = memory
- priors + sense = perception

Expectations

- For all cognitive systems, some priors are hard-coded: body shape, sensing array, even neural connectivity.
  - Derived from the experience of evolution or from a designer.
- Other expectations are derived from an individual's own experience – learning.

Learning

- Learning requires:
  - A representation.
  - A means of acting on current evidence.
  - A means of incorporating feedback concerning the outcome of the guess.
- AI learning calls incorporating feedback "error correction".

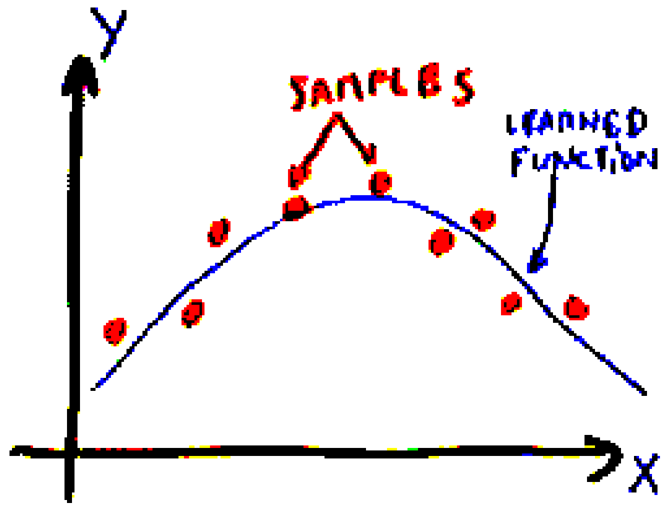
Learning is NOT Memorization

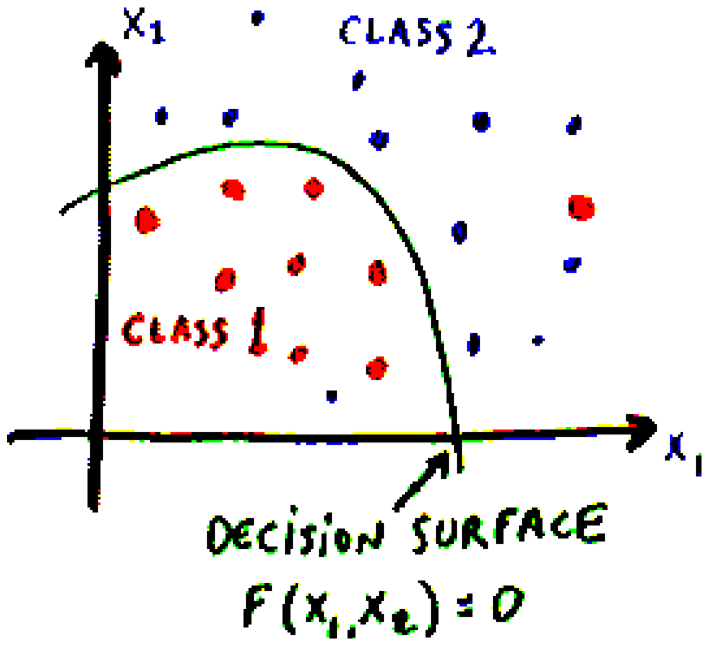
- rote learning is easy: just memorize all the training examples and their corresponding outputs.
- when a new input comes in, compare it to all the memorized samples, and produce the output associated with the matching sample.
- PROBLEM: in general, new inputs are different from training samples.
- The ability to produce correct outputs or behavior on previously unseen inputs is called GENERALIZATION.
- rote learning is memorization without generalization.
- The big question of Learning Theory (and practice): how to get good generalization with a limited number of examples.

Learning Outcomes

- Objective is to do the right thing at the right time (to be intelligent.)
- Doing the right thing often requires predicting likely possible sensory conditions so you can disambiguate situations that would otherwise be perceptually aliased.

Two Kinds of Supervised Learning





What we'll use as an example today.

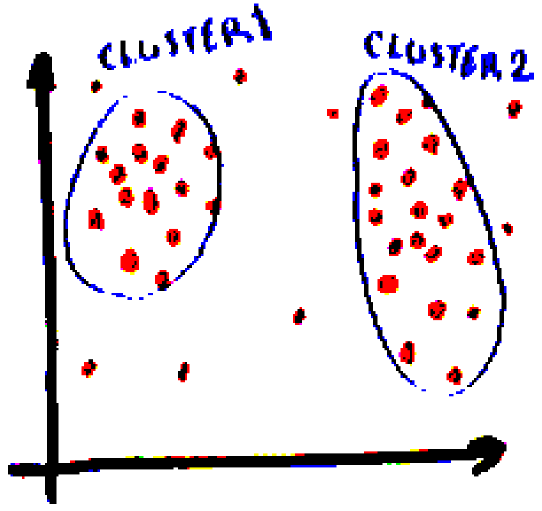
- Regression: also known as "curve fitting" or "function approximation". Learn a continuous input-output mapping from a limited number of examples (possibly noisy).
- Classification: outputs are discrete variables (category labels). Learn a decision boundary that separates one class from the other. Generally, a "confidence" is also desired (how sure are we that the input belongs to the chosen category).

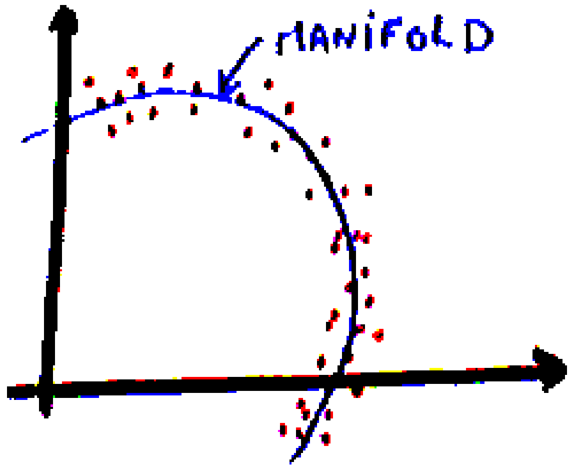
Includes kernel methods (not covered here.)

Unsupervised Learning

c.f. Lecture 5 “what the brain seems to be doing”

Unsupervised learning comes down to this: if the input looks like the training samples, output a small number, if it doesn’t, output a large number.

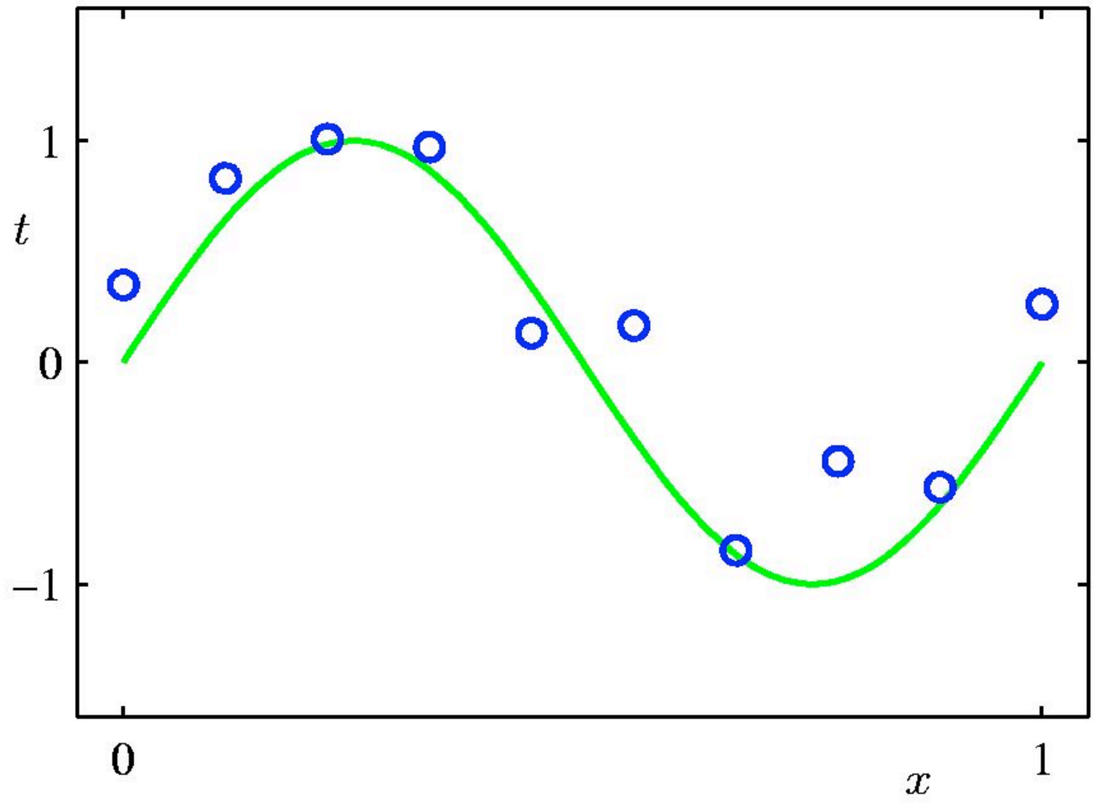




- This is a horrendously ill-posed problem in high dimension. To do it right, we must guess/discover the hidden structure of the inputs. Methods differ by their assumptions about the nature of the data.
- A Special Case: Density Estimation. Find a function f such $f(X)$ approximates the probability density of X, $p(X)$, as well as possible.
- Clustering: discover “clumps” of points
- Embedding: discover low-dimensional manifold or surface near which the data lives.
- Compression/Quantization: discover a function that for each input computes a compact “code” from which the input can be reconstructed.

Polynomial Curve Fitting

"Regression" via Chris Bishop

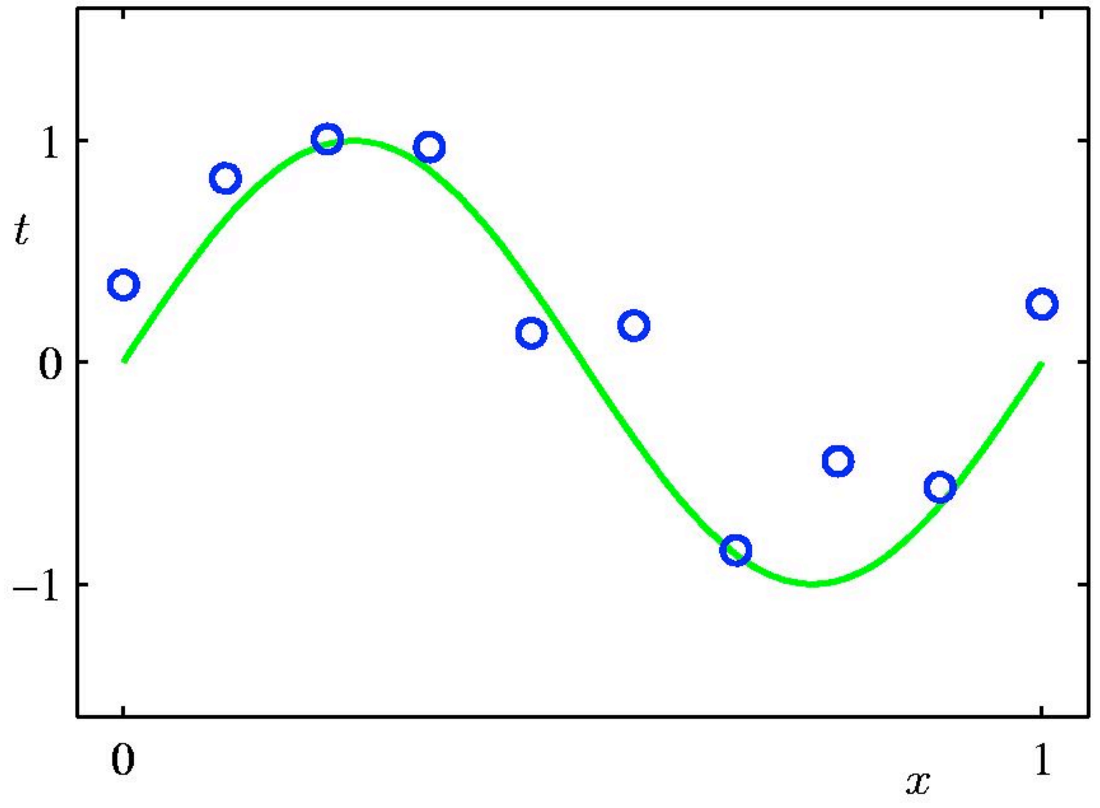


$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

Representation: Just a polynomial equation.

Example Application to Action Selection

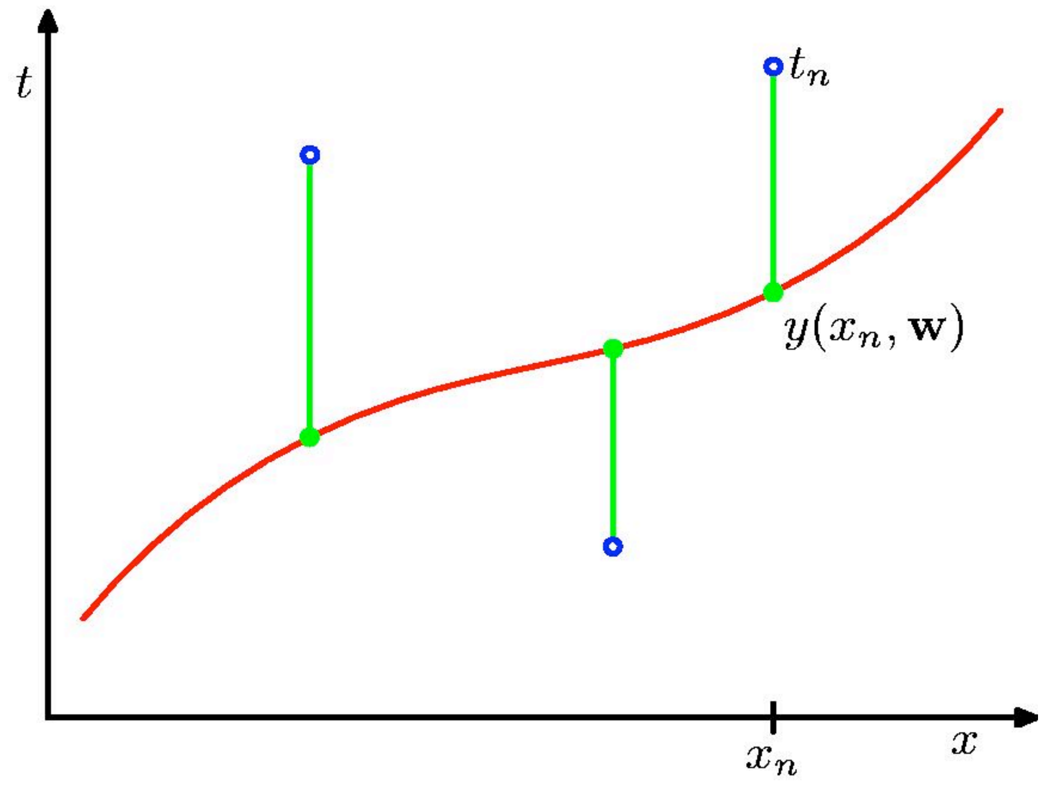
Where to drive your motor.



What you sensed

Sum-of-Squares Error Function

Use data to fix the world model currently held in the representation.



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

Error functions

- Based on some parameter **w** (for *weight* – more on why it's called that later).
- Objective is to minimise error function.
  - Take its derivative with respect to w.
  - Go down (take second deriv. if nec.)
- Linear functions gives a nice U function $\therefore$ you can tell when your done, derivative = 0.

Theory vs Practice

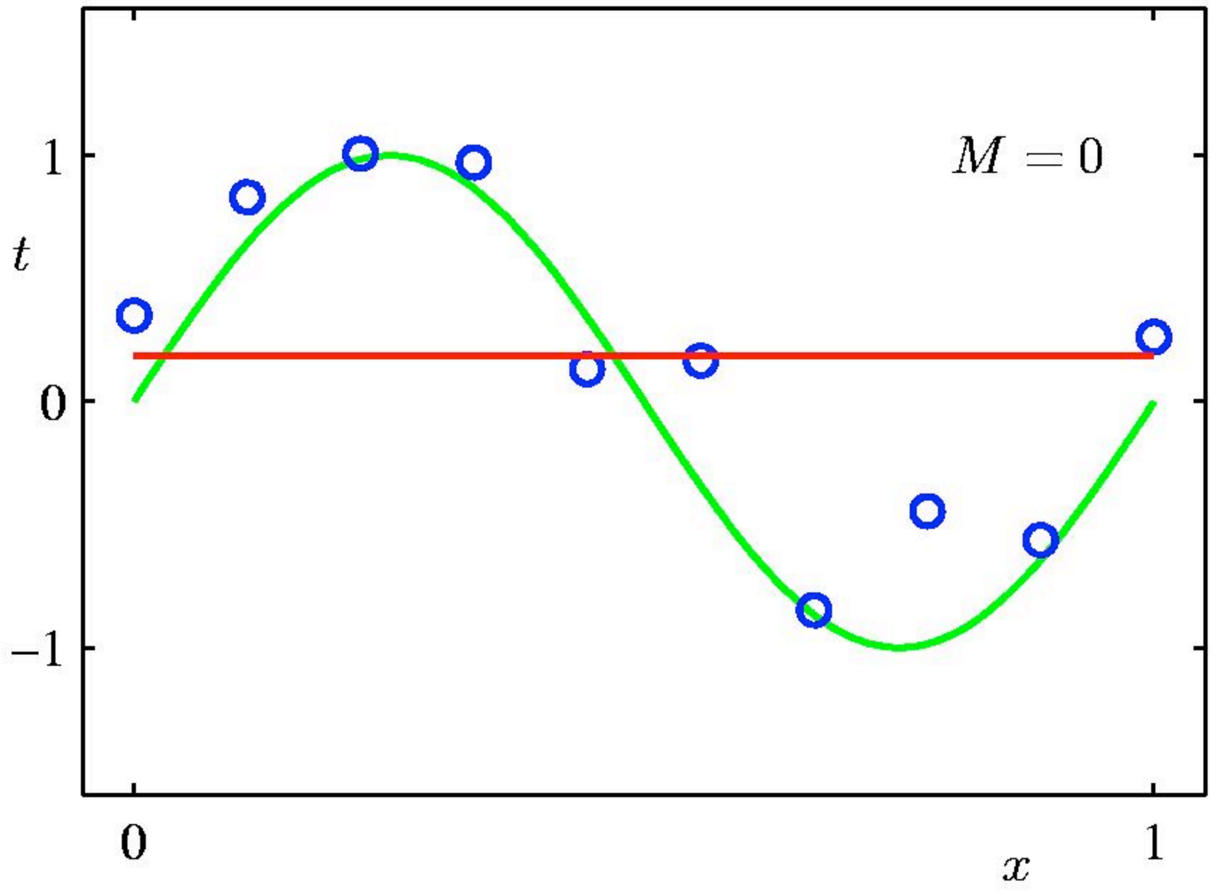
- If we assume that noise in signal is Normally distributed (with fixed variance), then least squares is equivalent to probabilistic methods (Per CM20220).
- Least squares is a lot easier to implement & lighter-weight to run.
- To the extent the assumption doesn’t hold, quality of results degrades – may be OK.

Why Representations Matter

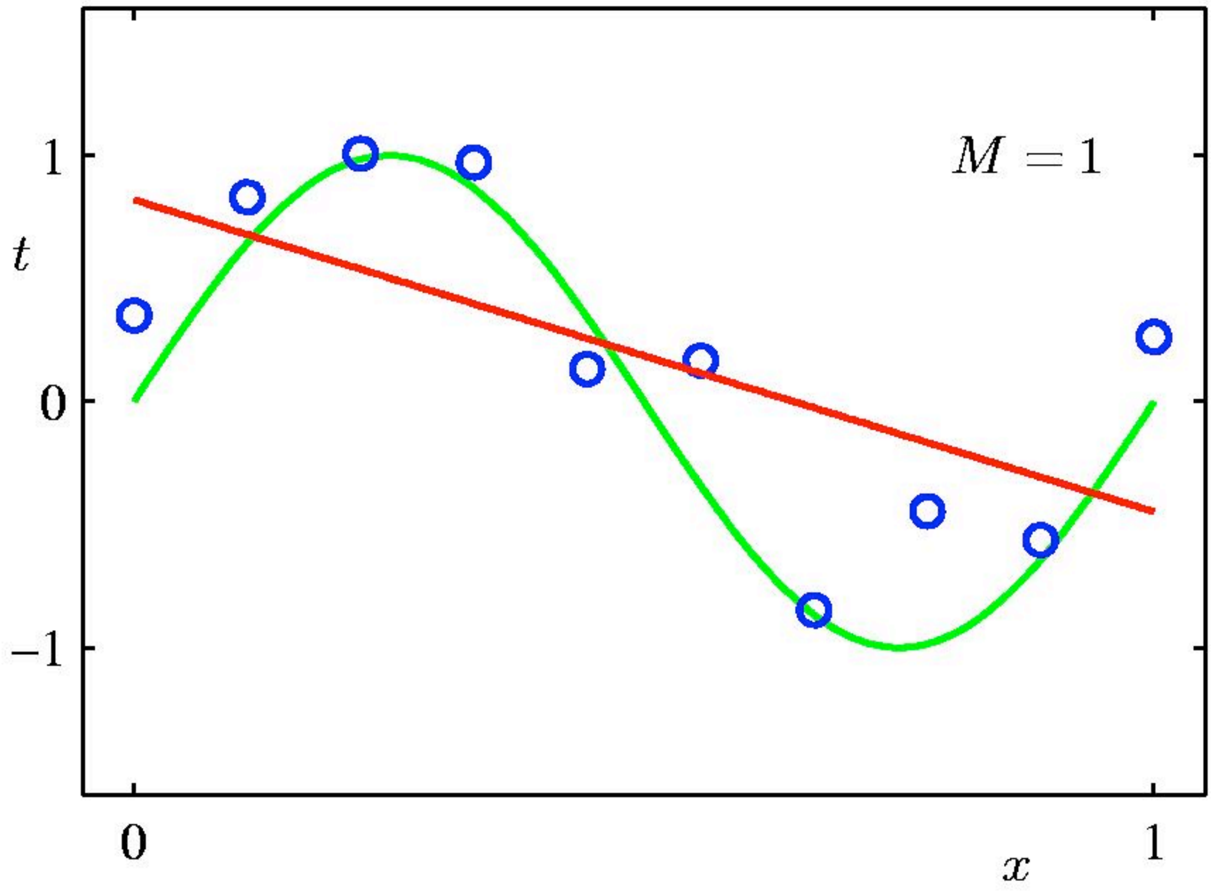
Green line is model used to generate data (in combination with noise).

Red line is the model learned from observing that data.

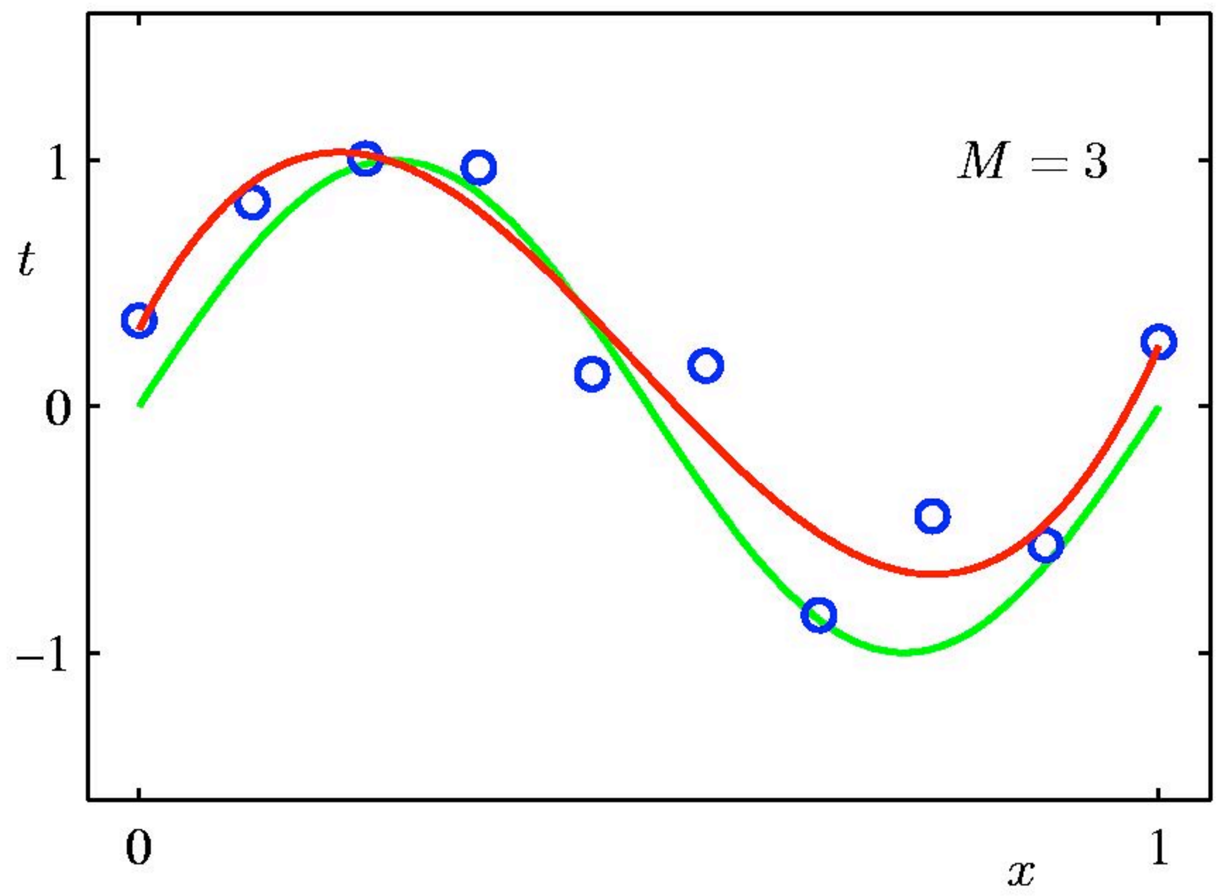
0th Order Polynomial



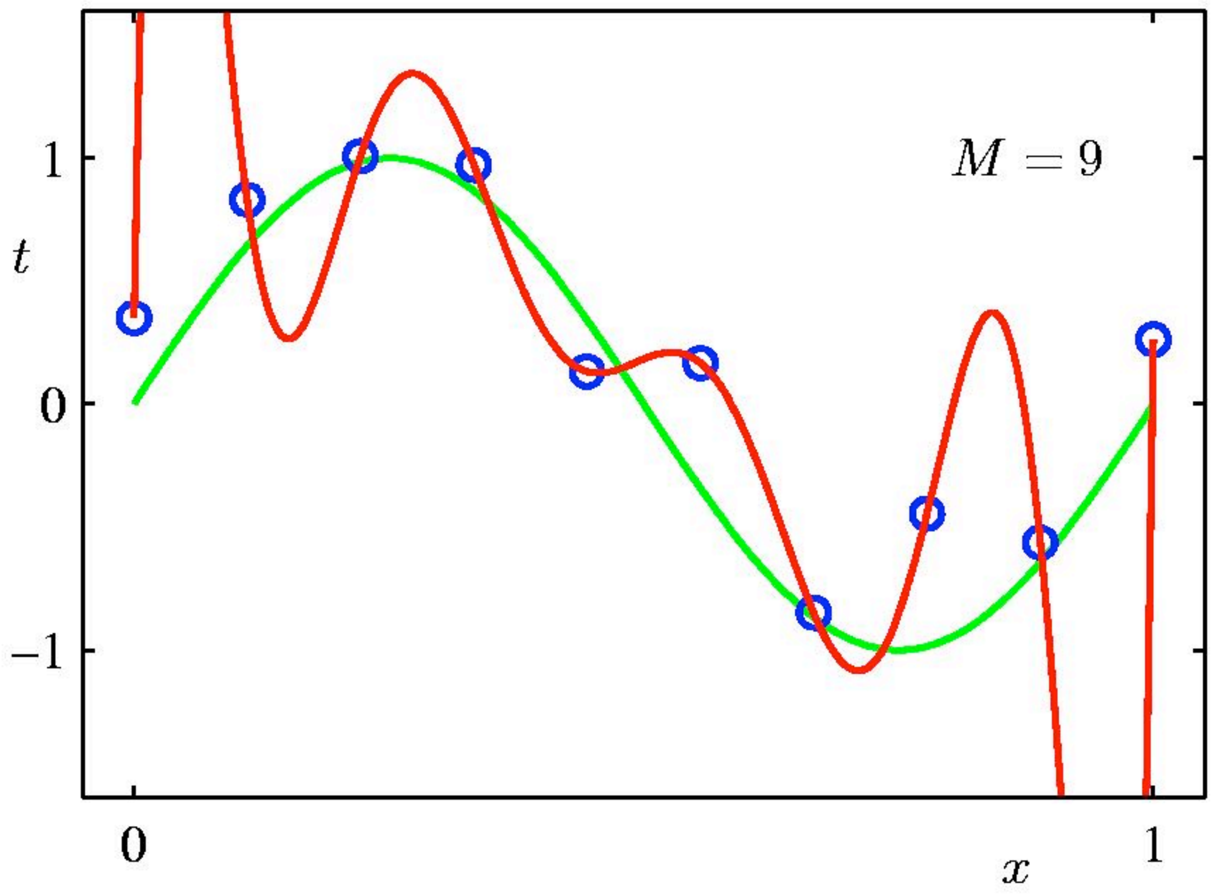
1st Order Polynomial



3rd Order Polynomial

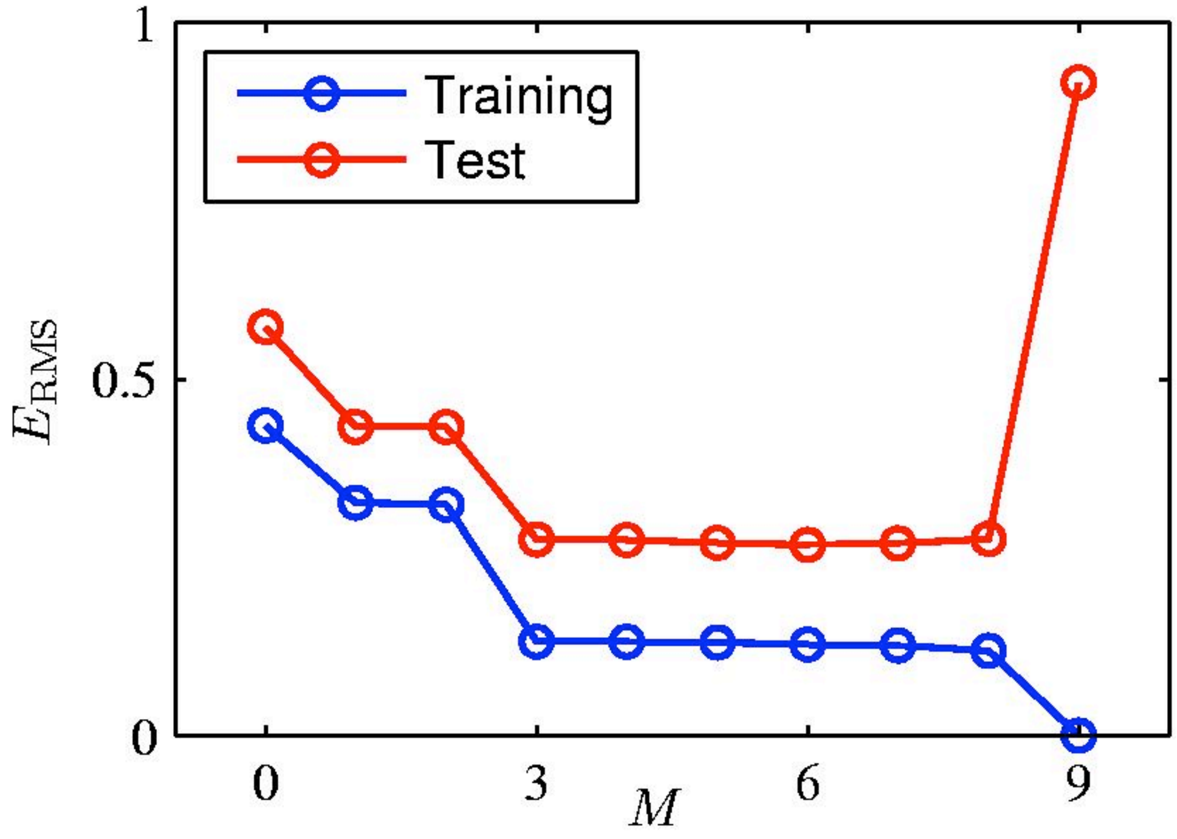


9th Order Polynomial



Over-fitting

When your model is too powerful for the data, it just "rote memorises" without generalising.



That means you get better on training data but worse on data you haven't seen.

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

Spot the indication of a problem.

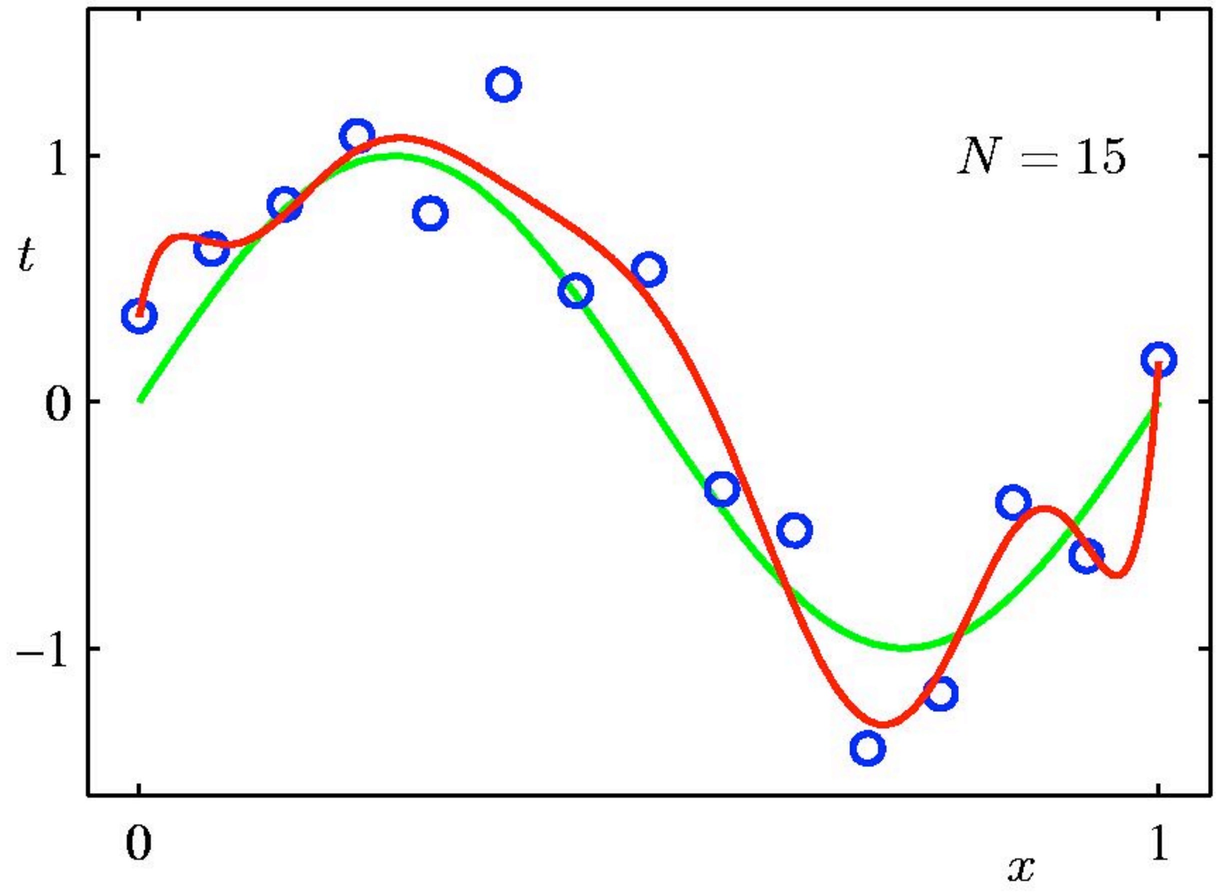
Polynomial Coefficients

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^\star$	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

More data makes it better...

Data Set Size: $N = 15$

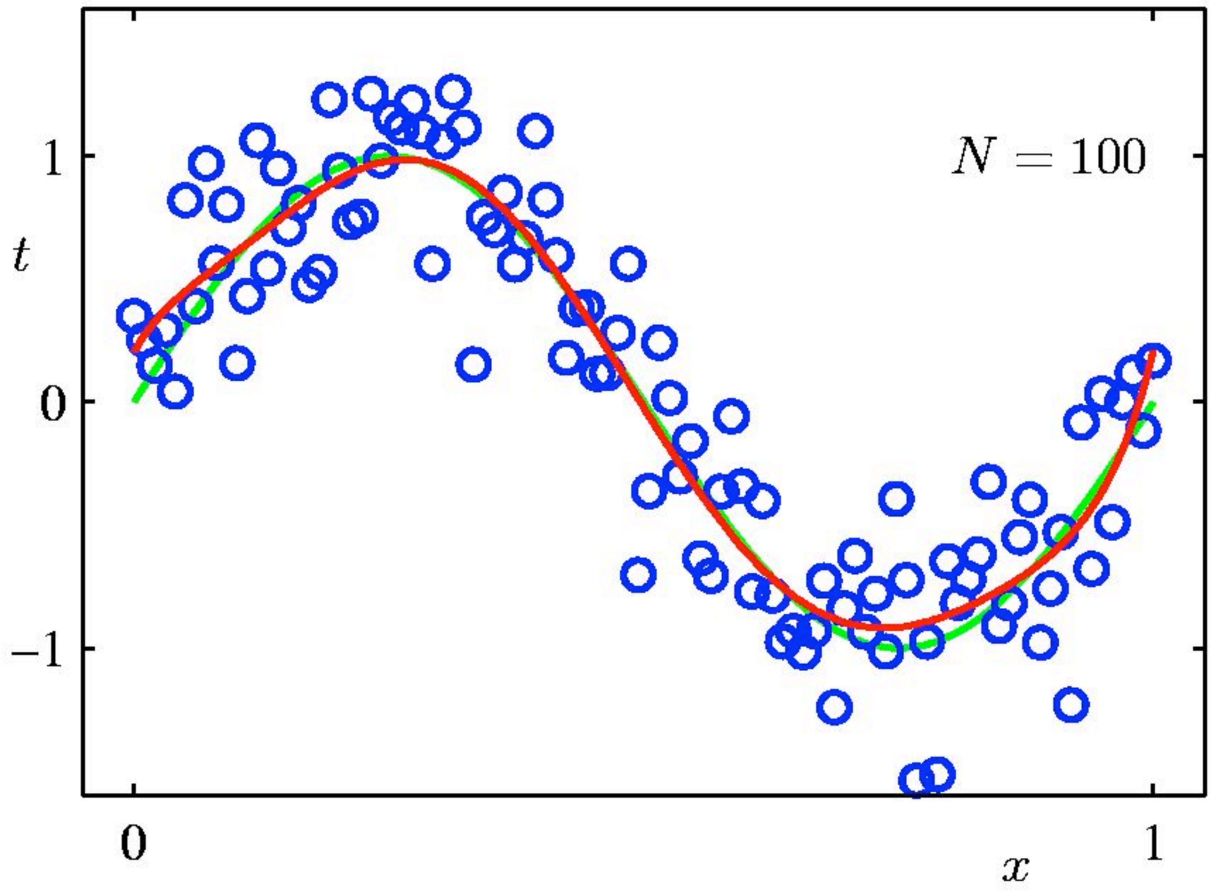
9th Order Polynomial



and better... more data is more information on the underlying model!

Data Set Size: $N = 100$

9th Order Polynomial



Learning is NOT Memorization

- rote learning is easy: just memorize all the training examples and their corresponding outputs.
- when a new input comes in, compare it to all the memorized samples, and produce the output associated with the matching sample.
- PROBLEM: in general, new inputs are different from training samples.
- The ability to produce correct outputs or behavior on previously unseen inputs is called GENERALIZATION.
- rote learning is memorization without generalization.
- The big question of Learning Theory (and practice): how to get good generalization with a limited number of examples.

Overfitting

- If you can memorise everything then you have no error signal to learn from, so you can't improve your model.
- If you can really memorise everything this doesn't matter. "Generalisation isn't the point of learning. Being right is the point of learning." – Will Lowe
- But mostly, it matters.

Spot the indication of a problem.

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

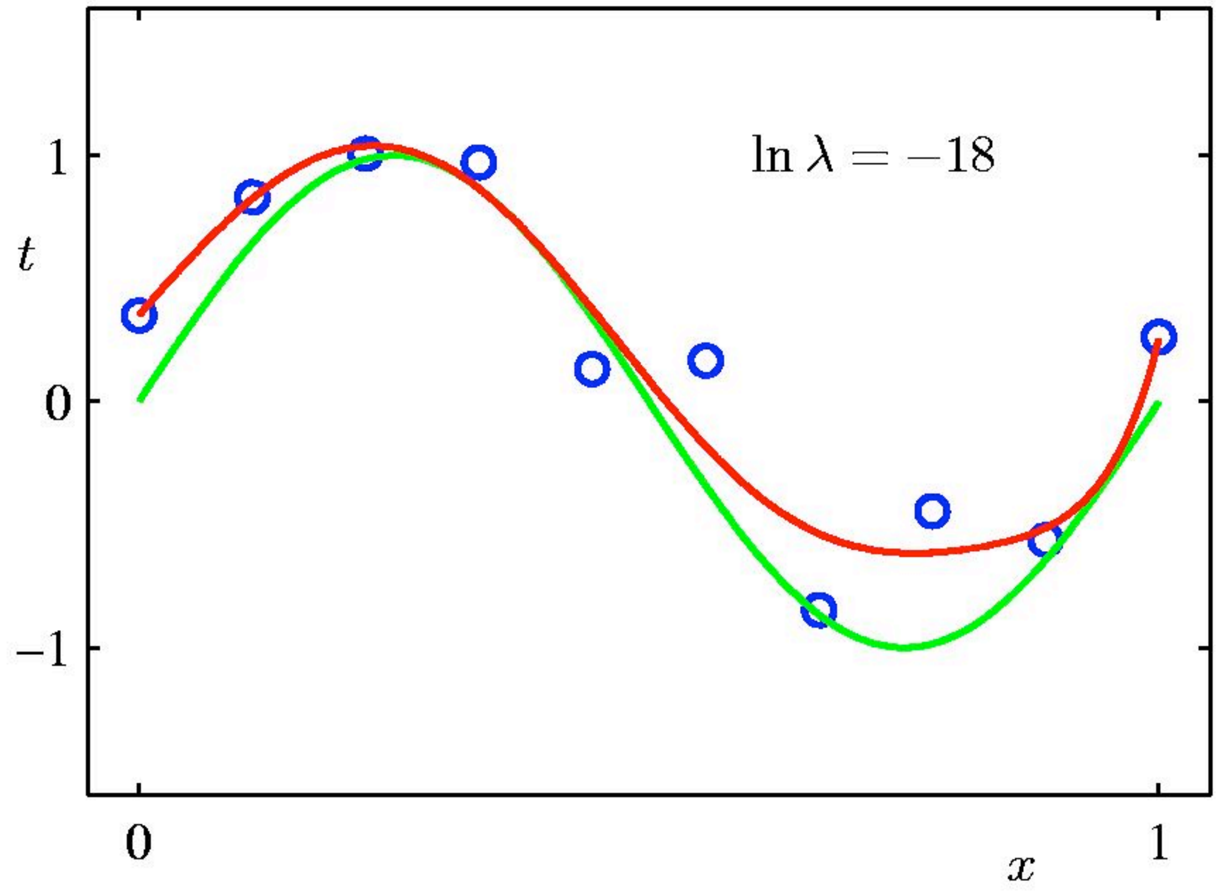
Another solution (when you can't get more data)

Regularization

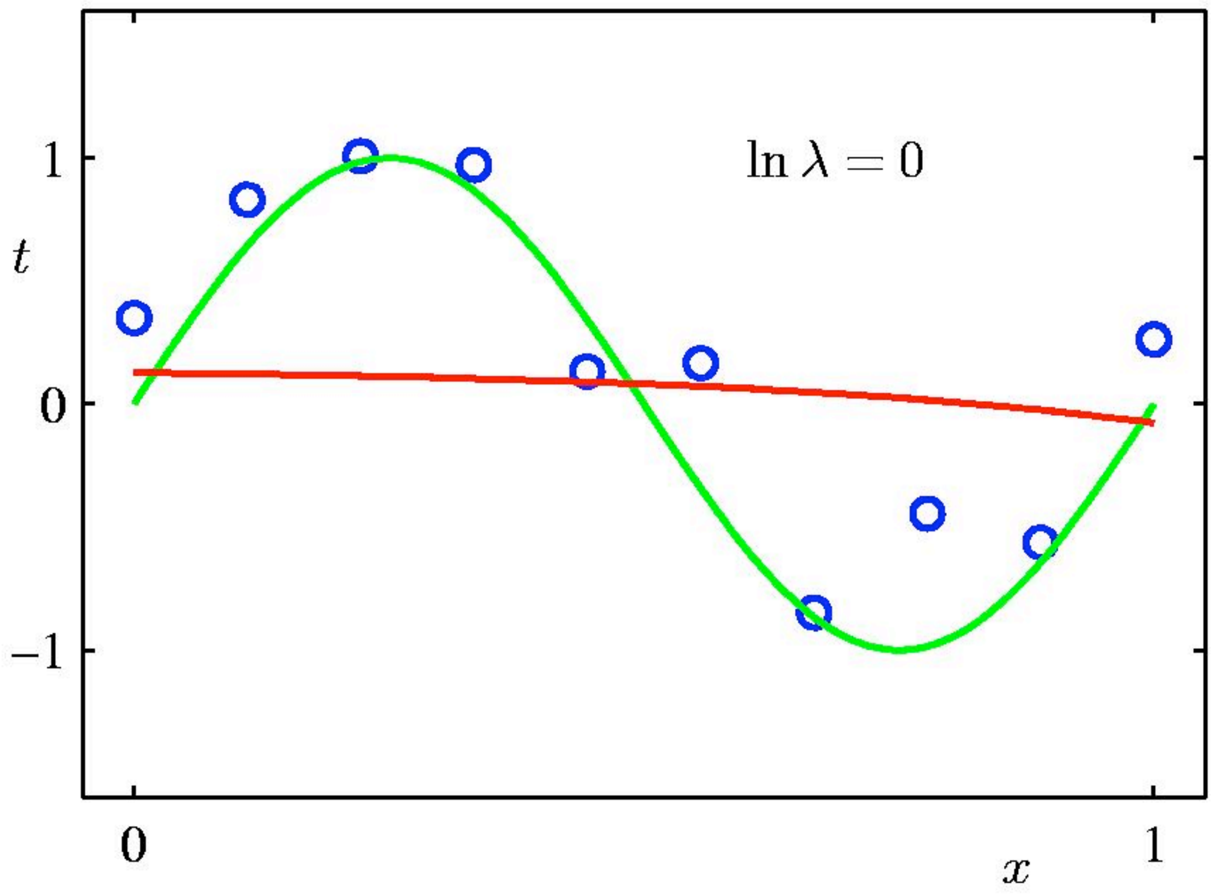
Penalize large coefficient values

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

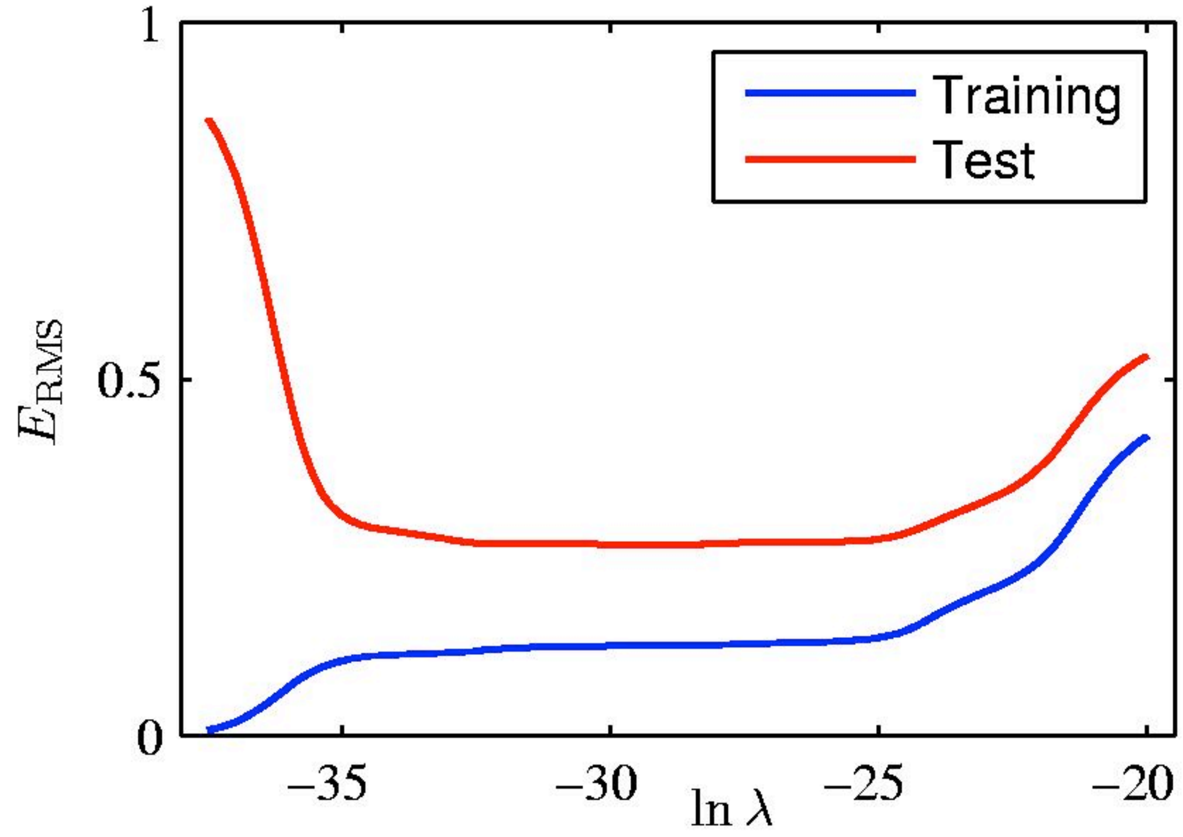
Regularization: $\ln \lambda = -18$



Regularization: $\ln \lambda = 0$

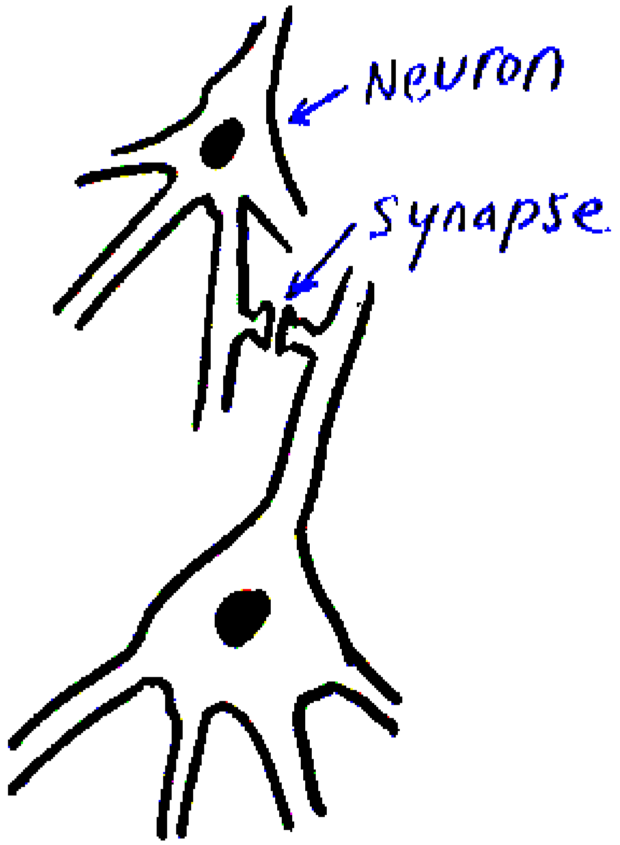


Regularization: E_{RMS} vs. $\ln \lambda$



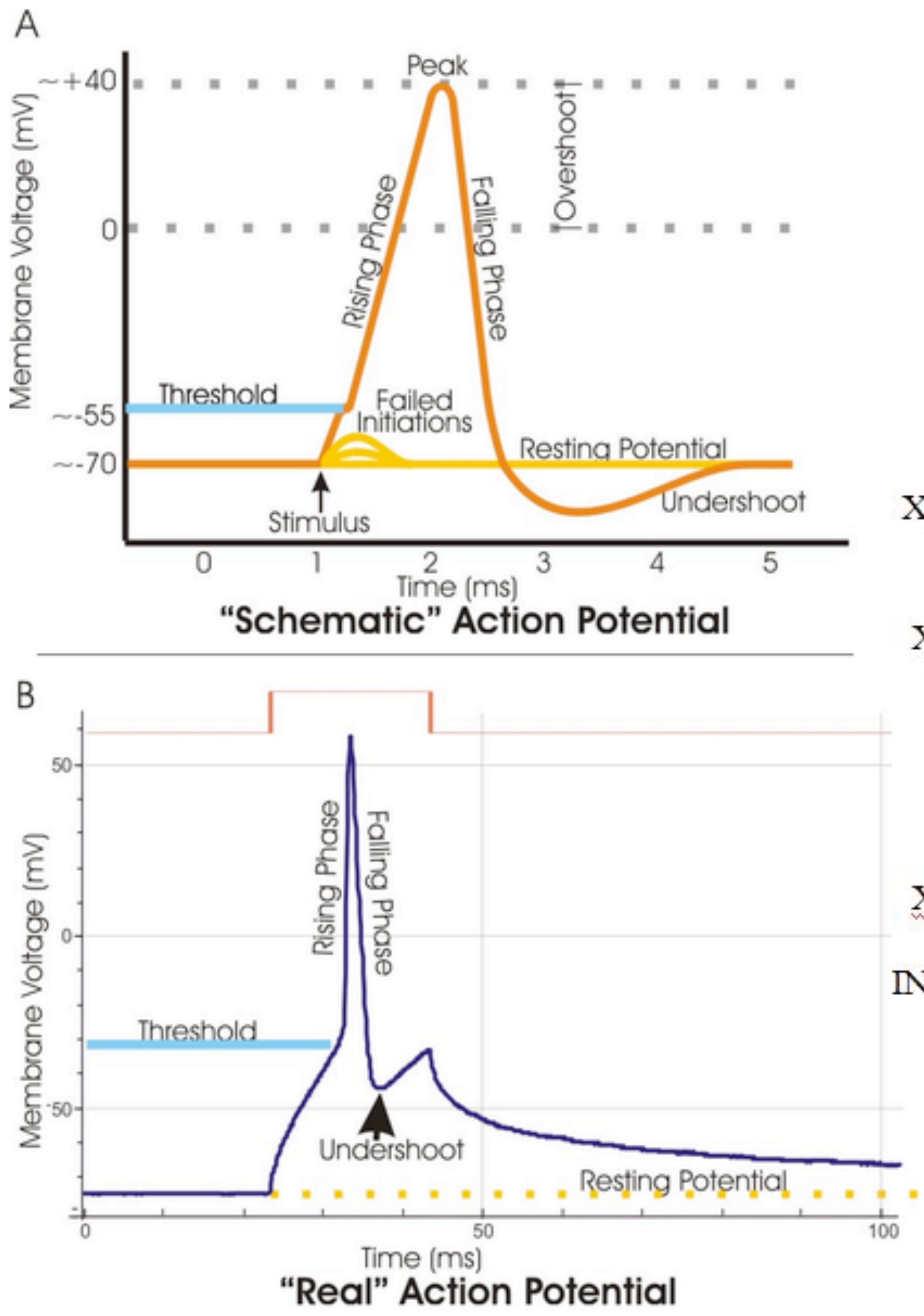
How Biology Does It

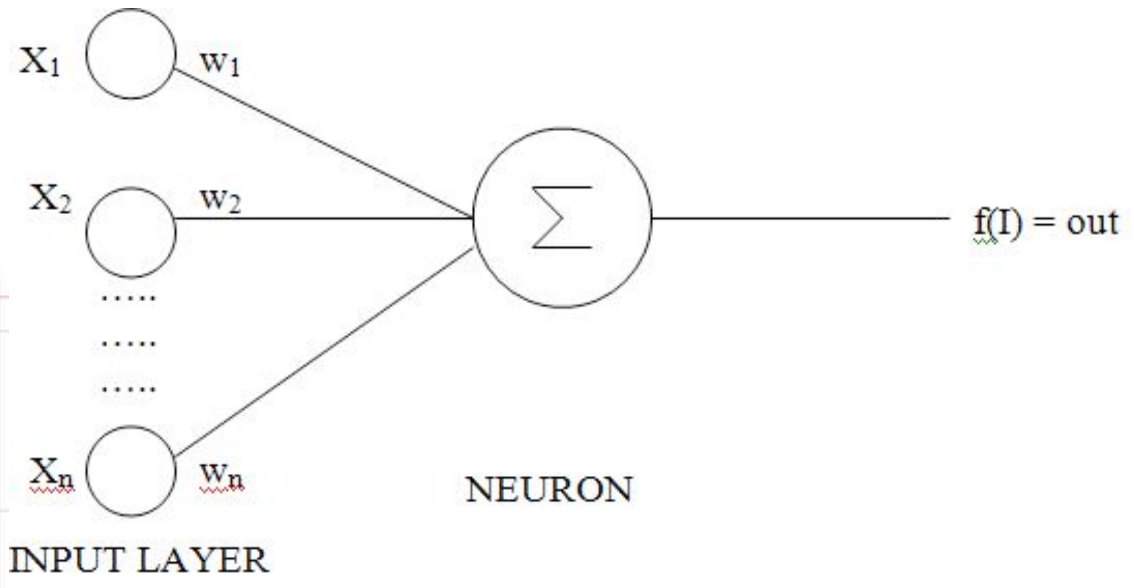
As per last time...



- The first attempts at machine learning in the 50's, and the development of artificial neural networks in the 80's and 90's were inspired by biology.
- Nervous Systems are networks of neurons interconnected through synapses
- Learning and memory are changes in the "efficacy" of the synapses
- HUGE SIMPLIFICATION: a neuron computes a weighted sum of its inputs (where the weights are the synaptic efficacies) and fires when that sum exceeds a threshold.
- Hebbian learning (from Hebb, 1947): synaptic weights change as a function of the pre- and post-synaptic activities.
- orders of magnitude: each neuron has 10^3 to 10^5 synapses. Brain sizes (number of neurons): house fly: 10^5; mouse: 5.10^6, human: 10^{10}.

Perceptron

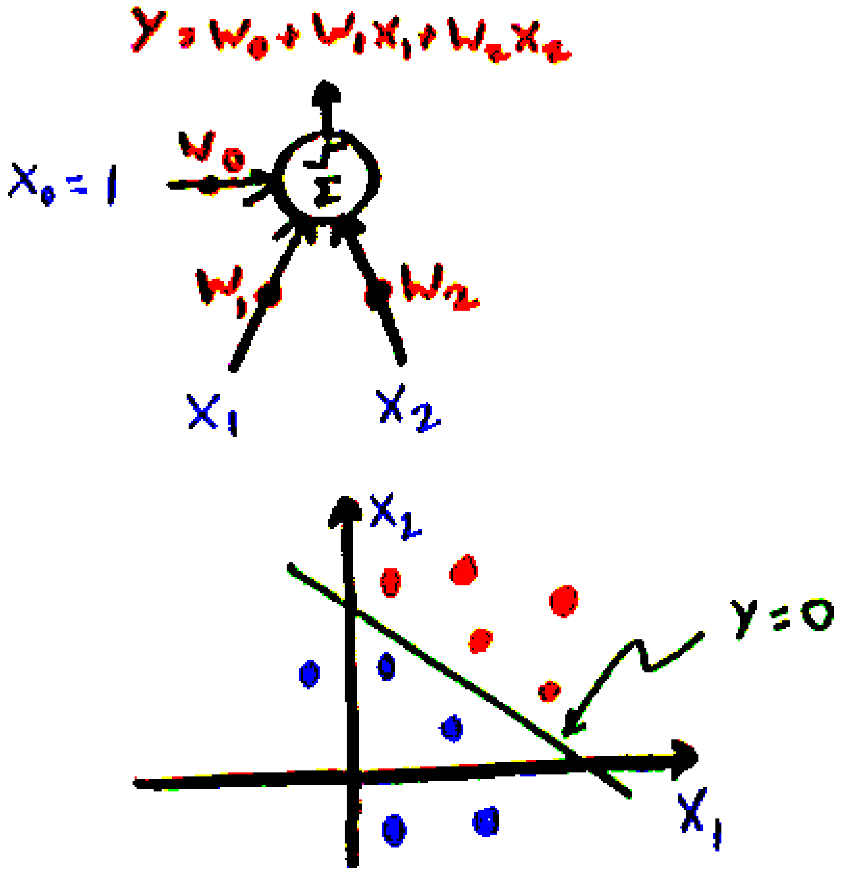




wikipedia

The Linear Classifier (originally: The Perceptron)

Historically, the Linear Classifier was designed as a highly simplified model of the neuron (McCulloch and Pitts 1943, Rosenblatt 1957):



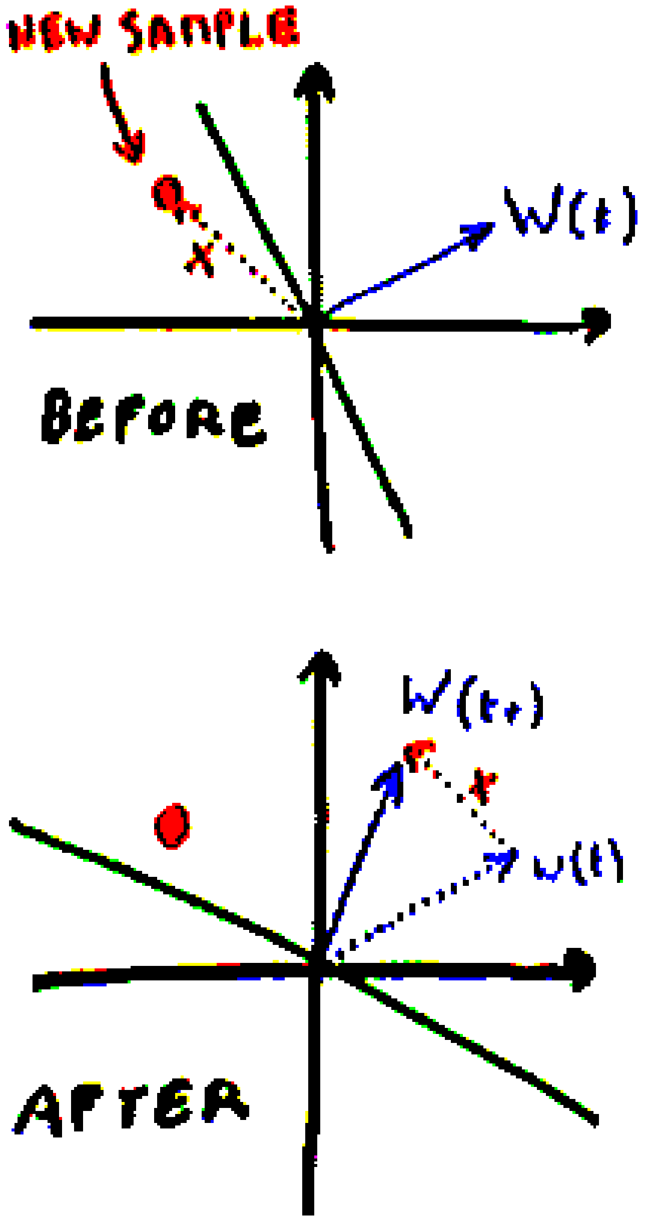
$$y = f\left(\sum_{i=0}^{i=N} w_i x_i\right)$$

With f is the threshold function: $f(z) = 1$ iff $z > 0$, $f(z) = -1$ otherwise. x_0 is assumed to be constant equal to 1, and w_0 is interpreted as a bias.
In vector form: $W = (w_0, w_1 .... w_n), X = (1, x_1 ... x_n)$:

$$y = f(W'X)$$

The hyperplane $W'X = 0$ partitions the space in two categories. W is orthogonal to the hyperplane.

A Simple Idea for Learning: Error Correction



Perceptron Learning Algorithm

We have a **training set** $\mathcal{S}$consisting of P input-output pairs: $\mathcal{S} = (X^1, y^1), (X^2, y^2), ...(X^P, y^P)$.
A very simple algorithm:
- show each sample in sequence repetitively
- if the output is correct: do nothing
- if the output is -1 and the desired output +1: increase the weights whose inputs are positive, decrease the weights whose inputs are negative.
- if the output is +1 and the desired output -1: decrease the weights whose inputs are positive, increase the weights whose inputs are negative.

More formally, for sample p:

$$w_i(t+1) = w_i(t) + (y_i^p - f(W'X^p))x_i^p$$

This simple algorithm is called the Perceptron learning procedure (Rosenblatt 1957).

Historical Note

- Our understanding of linear classifiers and probability-based learning came from our attempts to understand what neural networks (NN) could & couldn’t do.
- NN are intuitive, easy, algorithmic & attractive, biologically inspired.
- But these days, most (not all) real action is happening in straight maths.

Common Learning Algorithm Tricks

- How much you add or subtract from the weight determines how fast you learn: learning rate.
- If you learn too fast you can overshoot the ideal value, do this a lot and you dither forever.
- Want learning to converge on right values.

The Perceptron Learning Procedure

Provably works iff linearly separable.

Theorem: If the classes are linearly separable (i.e. separable by a hyperplane), then the Perceptron procedure will converge to a solution in a finite number of steps.
Proof: Let's denote by W^* a normalized vector in the direction of a solution. Suppose all X are within a ball of radius R. Without loss of generality, we replace all X^p whose y^p is -1 by $-X^p$, and set all y^p to 1. Let us now define the margin $M = min_p W^* X^p$. Each time there is an error, $W.W^*$ increases by at least $X.W^* \geq M$. This means $W_{final}.W^* \geq NM$ where N is the total number of weight updates (total number of errors). But, the change in square magnitude of W is bounded by the square magnitude of the current sample X^p, which is itself bounded by R^2. Therefore, $|W_{final}|^2 \leq NR^2$. combining the two inequalities $W_{final}.W^* \geq NM$ and $|W_{final}| \leq \sqrt{N}R$, we have

$$W_{final}.W^*/|W_{final}| \geq \sqrt{(N)}M/R$$

. Since the left hand side is upper bounded by 1, we deduce

$$N \leq R^2/M^2$$

Proof by Minsky
(long story)

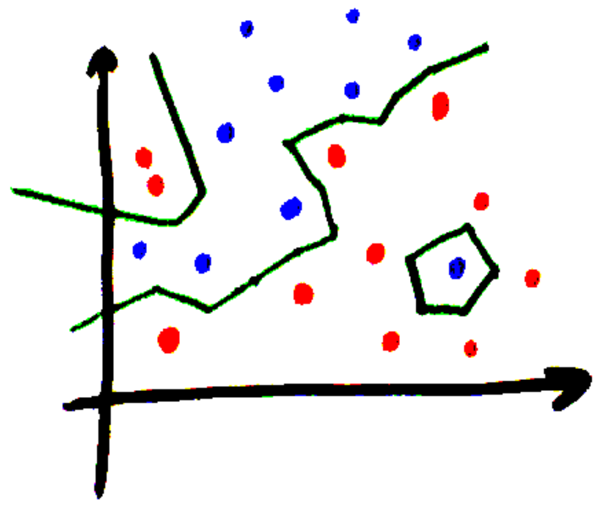
Neat vs Scruffy

- How can you be sure your problem is linearly separable?
  - You can't. Just try it. Scruffy.
  - Only use provably cool stuff. Neat.

Neats + Scruffies

- A collection of hacks is more likely to win if it is motivated by theory – if each hack is a reasonable approximation of what a sound system would do.
- A systems approach will look for indicators of fail states for scruffy solutions (e.g. coefficients blowing up earlier.)

A Simple Trick: Nearest Neighbor Matching



- Instead of insisting that the input be exactly identical to one of the training samples, let's compute the "distances" between the input and all the memorized samples (aka the prototypes).
- 1-Nearest Neighbor Rule: pick the class of the nearest prototype.
- K-Nearest Neighbor Rule: pick the class that has the majority among the K nearest prototypes.
- PROBLEM: What is the right distance measure?
- PROBLEM: This is horrendously expensive if the number of prototypes is large.
- PROBLEM: do we have any guarantee that we get the best possible performance as the number of training samples increases?

Problem, problem, problem but it works really well.

Can often also interpolate between stored solutions (Atkins, Schaal)

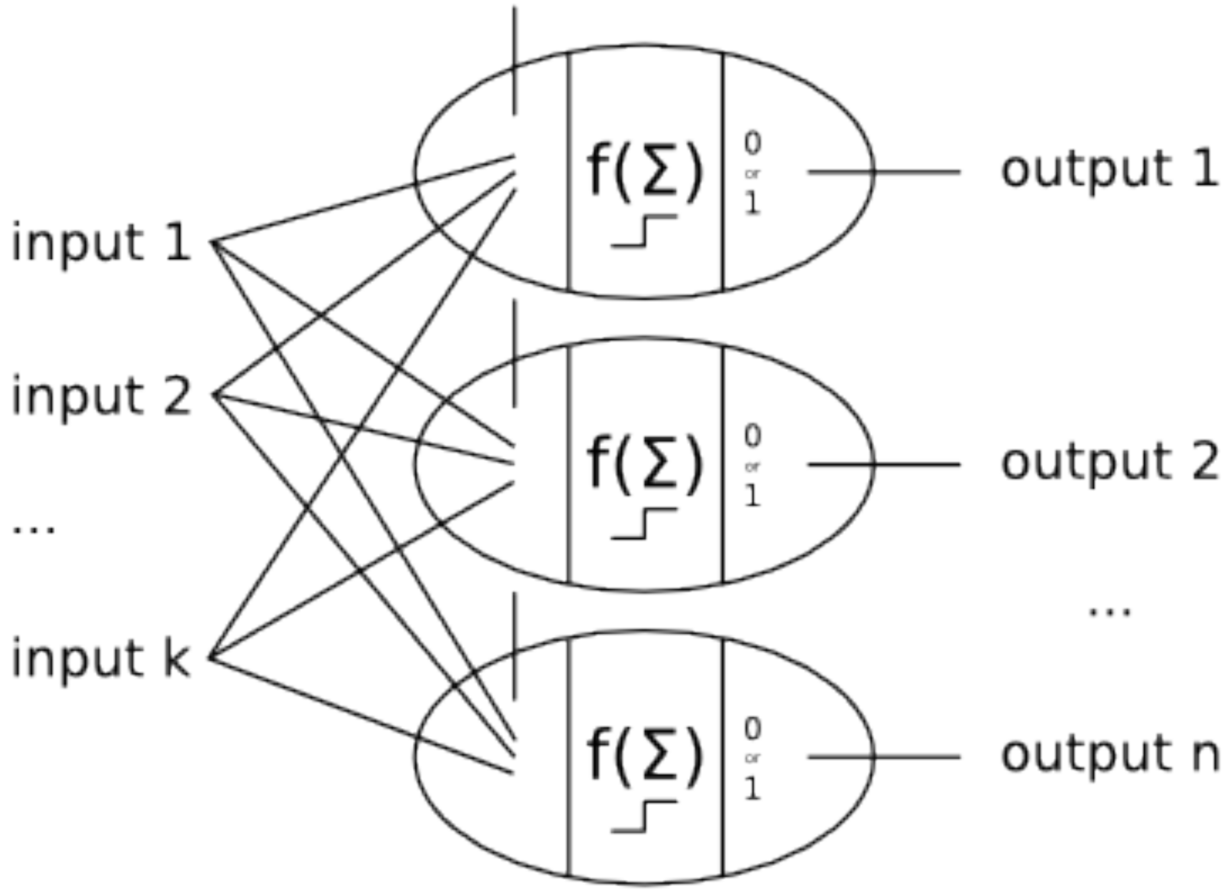
Single Layer Perceptron Network

Note:

mutual inhibition

"winner take all"

WTA



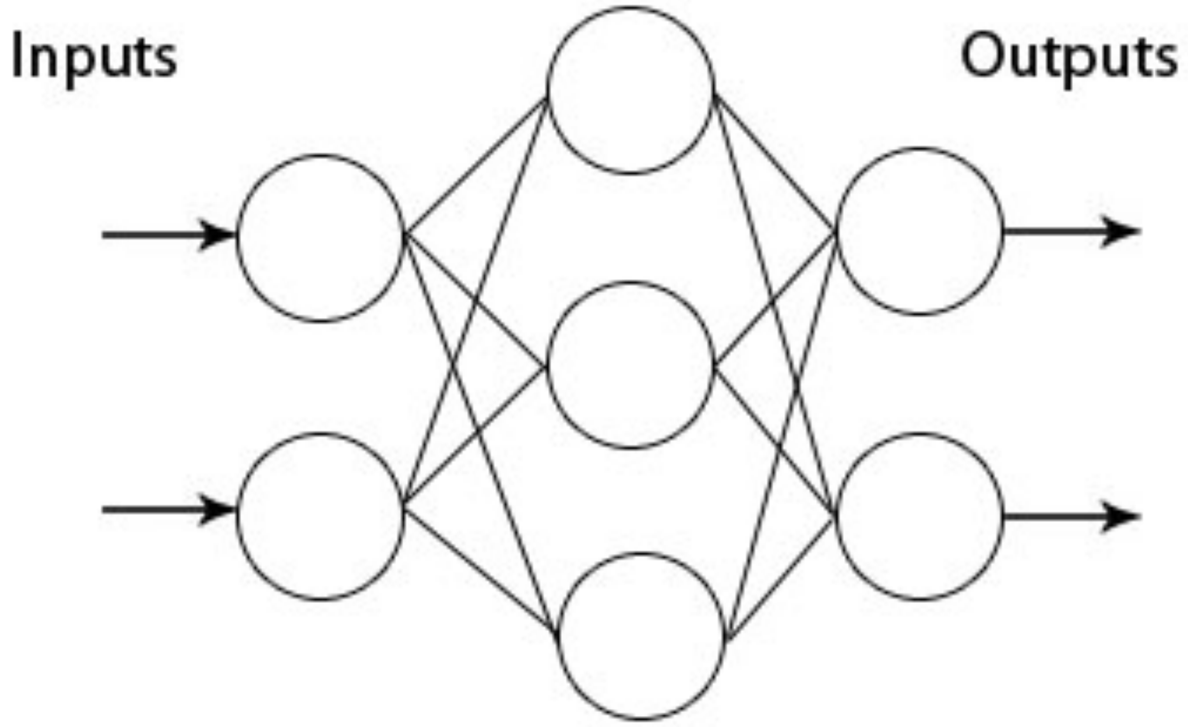
Neats vs Scruffies: Multilayer Perceptrons

- NN “learned like people” will solve AI.
- Minsky & Papert (1969) proved single-layered perceptron networks can’t solve some pretty basic problems.
- No one knew how to train multi-layer perceptrons, funding dried up, field almost died.

AI Winter

Multi Layered Perceptron

- Would solve the problem!
- But if there's an error, which weight caused it?

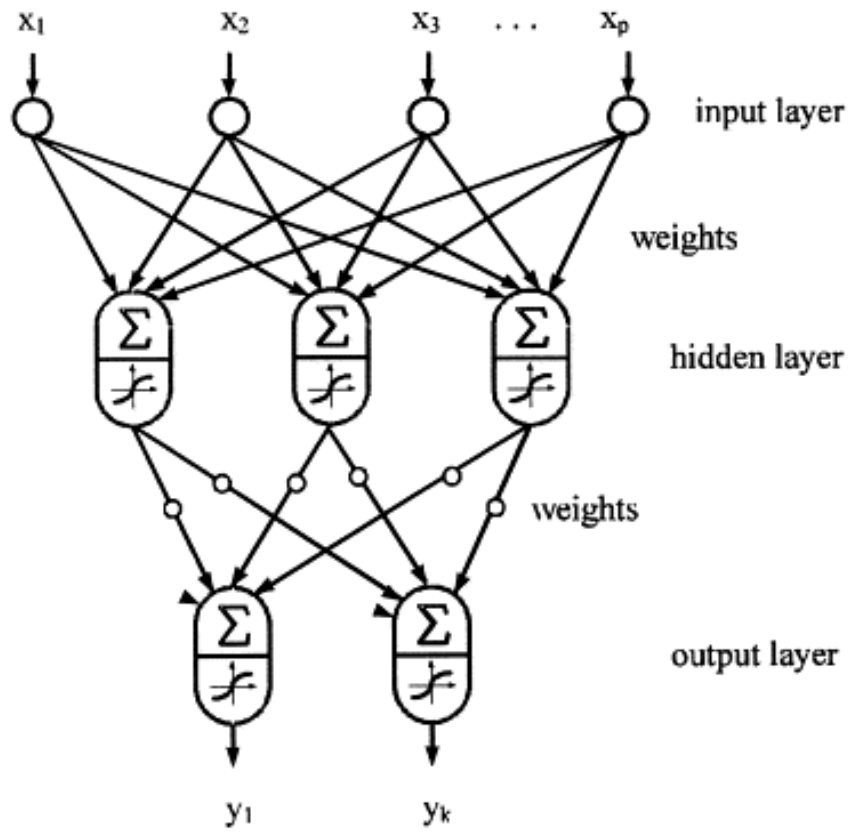


Neats vs Scruffies: Backpropagation

- In the 1980s, several people realised if the threshold was a sigmoid not a step function, you could assign "credit" across layers using calculus – backpropagation.
- But then they realised they could do *lots* of things with calculus & statistics – serious machine learning academics do Bayes now.

(Backpropagation is essentially the chain rule.)

Backpropagation



Geoff Hinton

- One of the (independent) backprop inventors.
- cf. deep learning, Boltzman Machines

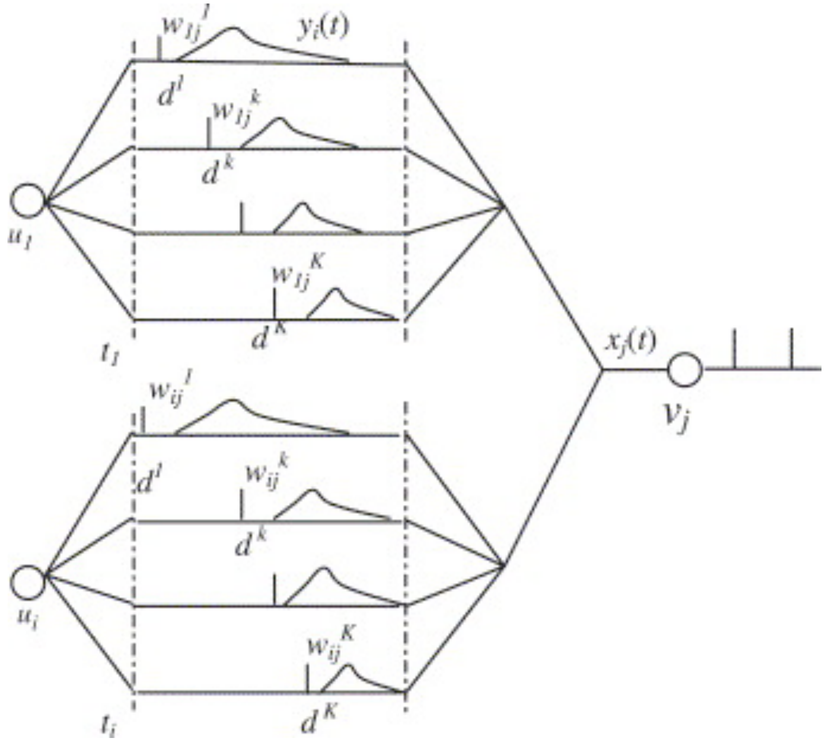
Neats vs Scruffies: Theory vs Practice

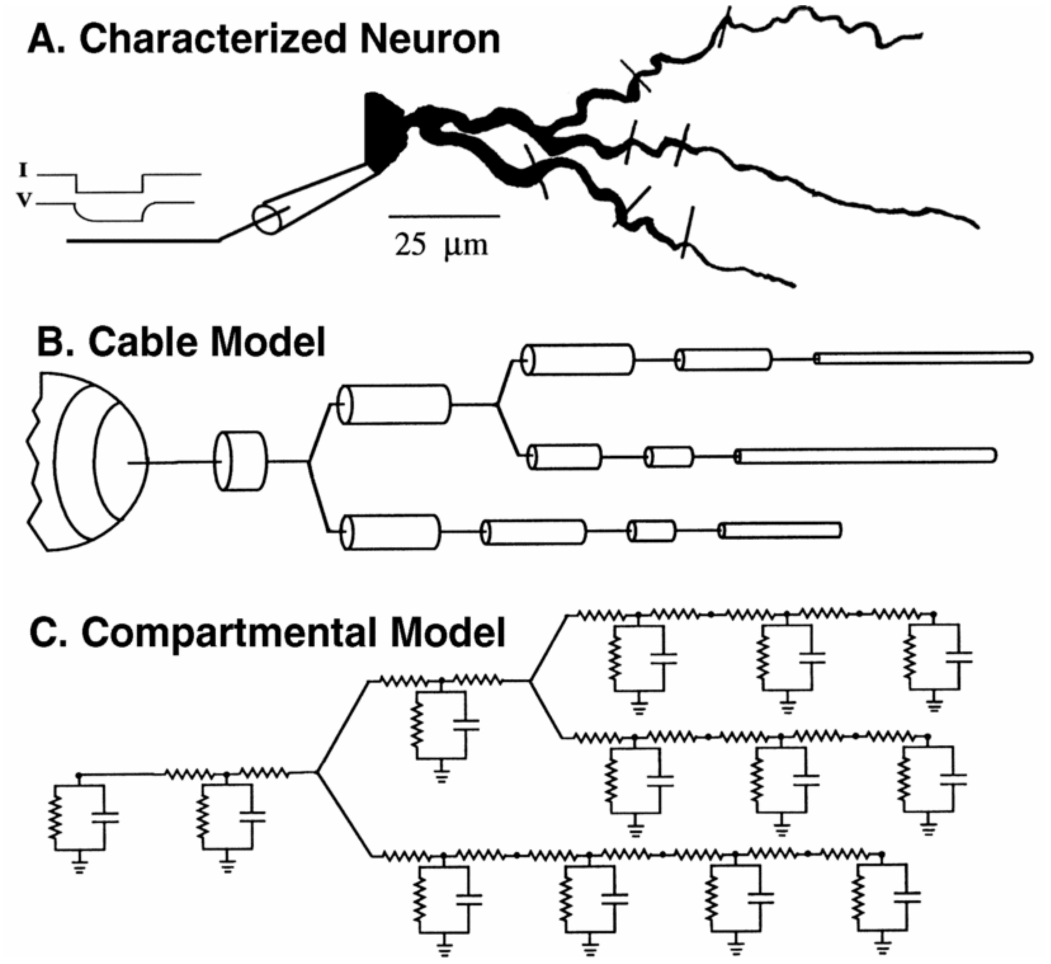
- Serious fast applied stuff e.g. Google do the serious neat stuff (though sometimes scruffily hacked together).
- But many, many, many applications of backpropagation on 3-layer networks in ordinary industry by students like you.
- 2013 "NN still used by psychologists, some artificial life researchers."

Other Topical NN Research

Compartmental models

Also 2013

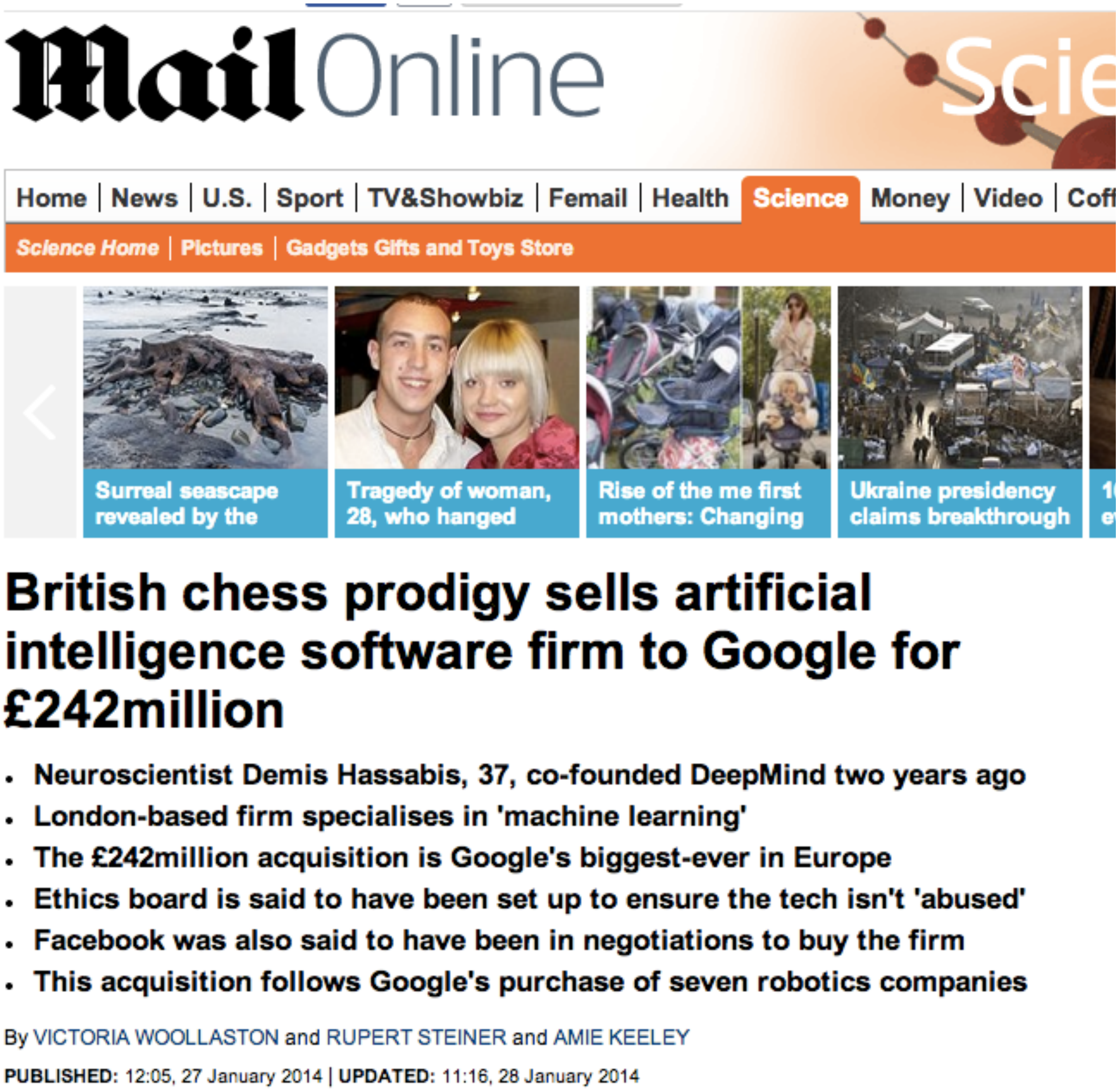




Spike timing networks

2014 Deep Mind

See also lecture notes…


MailOnline
Scie
Home | News | U.S. | Sport | TV&Showbiz | Femail | Health | Science | Money | Video | Coff
Science Home | Pictures | Gadgets Gifts and Toys Store
Surreal seascape revealed by the
Tragedy of woman, 28, who hanged
Rise of the me first mothers: Changing
Ukraine presidency claims breakthrough

British chess prodigy sells artificial intelligence software firm to Google for £242million

- Neuroscientist Demis Hassabis, 37, co-founded DeepMind two years ago
- London-based firm specialises in 'machine learning'
- The £242million acquisition is Google's biggest-ever in Europe
- Ethics board is said to have been set up to ensure the tech isn't 'abused'
- Facebook was also said to have been in negotiations to buy the firm
- This acquisition follows Google's purchase of seven robotics companies

By VICTORIA WOOLLASTON and RUPERT STEINER and AMIE KEELEY

PUBLISHED: 12:05, 27 January 2014 | **UPDATED:** 11:16, 28 January 2014