Neural Networks: Supervised Learning Algorithms

Curtis Baker

Dept of Ophthalmology & Visual Sciences
Montreal General Hospital

readings:

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

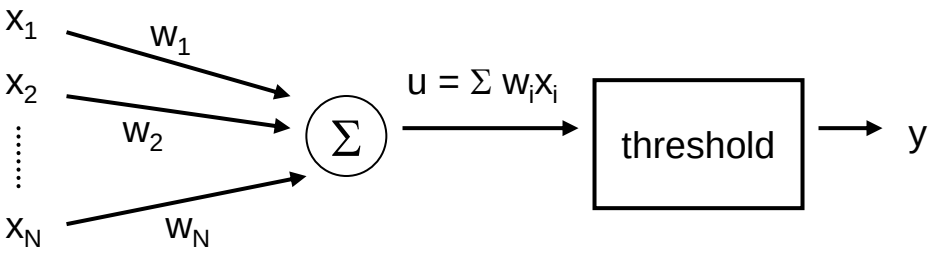
Pattern Recognition and Machine Learning, by Christopher Bishop
Chapter 1.1

***today*:**

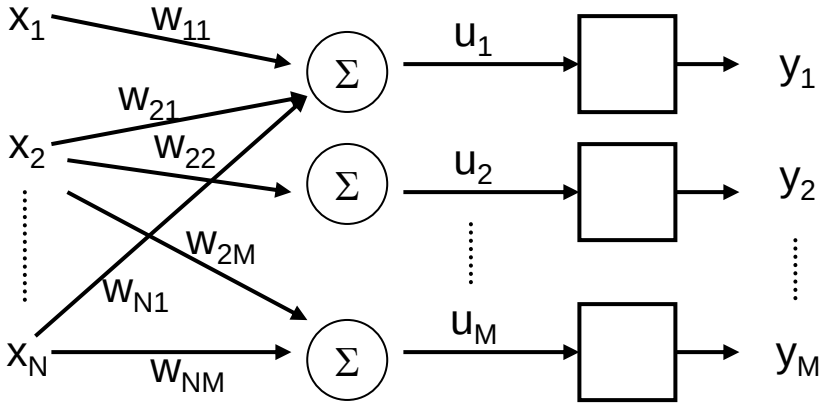
Neural Networks & Machine Learning, Classification
Regression
Deep Neural Networks

(Artificial) 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



inputs weights

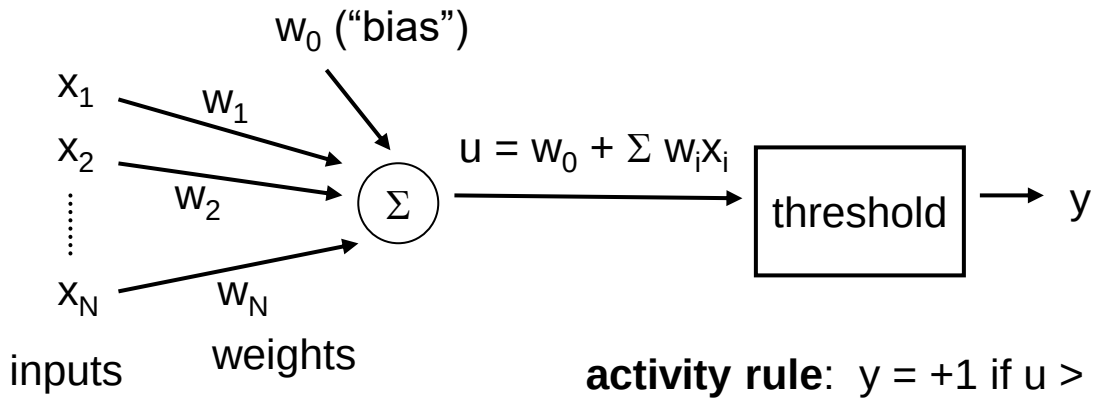
Frank Rosenblatt

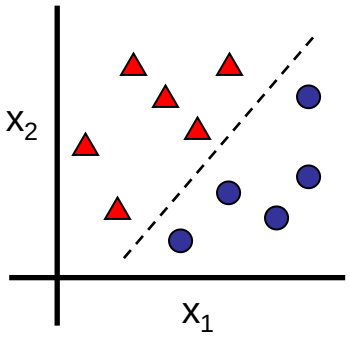
an early "binary classifier"
-> classify inputs X into 2 (or more) categories

https://blogs.umass.edu/comphon/2017/06/15/did-frank-rosenblatt-invent-deep-learning-in-1962/

Perceptron

Architecture: variables, and relationships between them





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

dataset:
inputs (x’s), e.g. photoreceptors / pixels, or higher-level features / receptive fields
corresponding classifications (T’s, i.e. the “teacher”)

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

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

if incorrect (y !=T):

$$\Delta w_i = 0$$

T = “teacher”

η (“eta”) = learning rate parameter

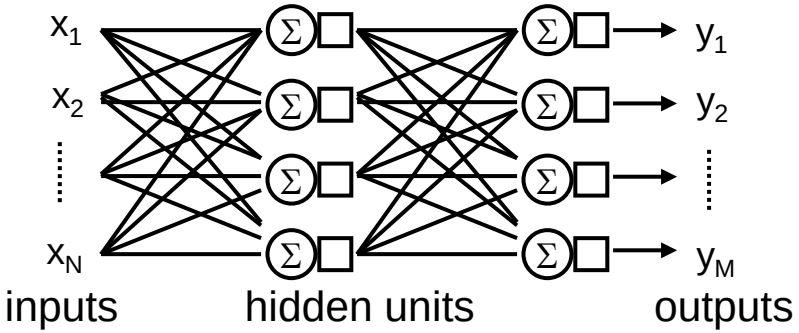
notes: weights initially random
sequential / on-line mode – up-date applied on each sequential trial
sensitive to η

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

Neural Networks - a brief history

1970s: Minsky & Papert critique of Perceptron
no way to train multi-layer networks

1980s: resurgence of the neural networkers: back-propagation algorithm, connectionism
concurrent influences: neural plasticity, NMDA receptors; Donald Hebb



1990s: difficulties with multi-layer networks
how many hidden units ? too many -> *over-fitting*

2000s: machine learning; probabilistic models, statistical learning theory

2010s: deep neural networks, deep learning

Types of learning algorithms

1. supervised - *data = inputs, targets (from a "teacher")*

classification / recognition
output is categorical: e.g., fMRI decoding of which stimulus is being viewed

regression
output is continuous: e.g., prediction of neuron's firing frequency

2. unsupervised - *data = set of multivariate values (without any "teacher")*
density estimation – clustering, 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
animal learning, e.g. operant conditioning
game-playing, e.g. Alpha-Go

common concepts throughout:
optimization algorithms - minimize an "error function" or "objective function"
can often be used as metaphorical neural models, or as data analysis tools
for single neurons, MUA, LFPs, EEG/MEG, fMRI, …

Optimization with gradient descent

principle: specify an **error function**, E, 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

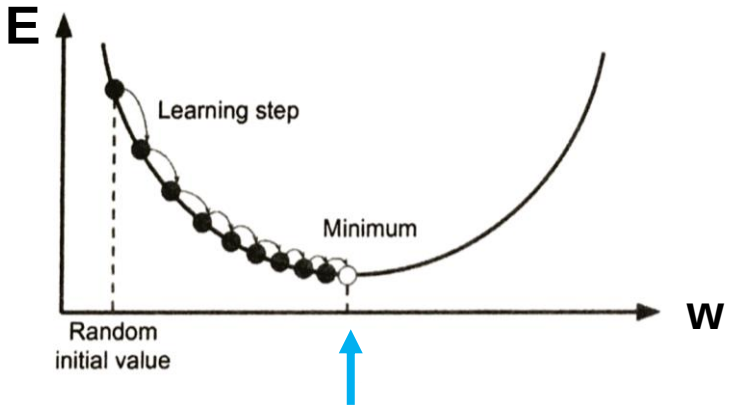
corresponding classifications (T's, i.e. the "teacher")

-> use algorithm, that iteratively modifies the parameters, to mimimize the error function

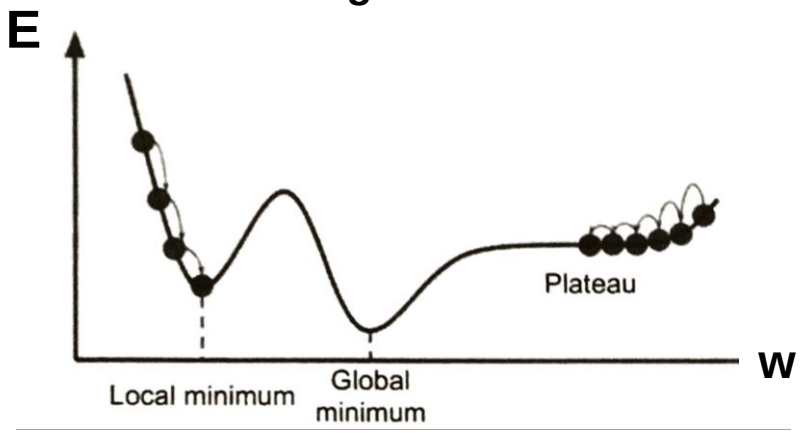
gradient descent: min: $\delta E / \delta w = 0$ E = error (objective) function

-> learning rule: $dw_i = \eta (y_i$ - predicted $y_i) x_i$

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



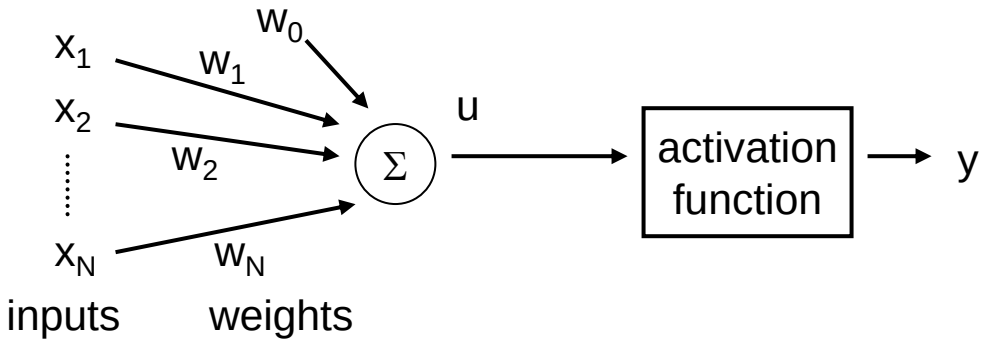
local vs global minima

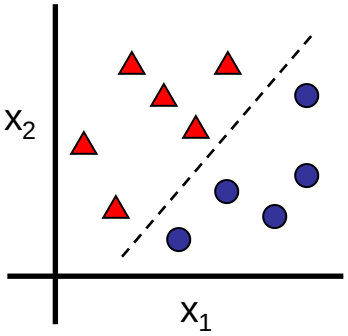


Géron, 2019

Classification with LMS gradient descent

architecture



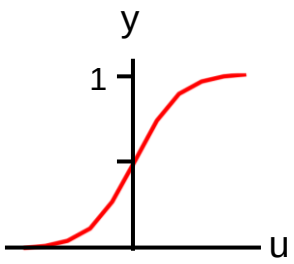


best run in batch mode (use all data in parallel, , as a "batch")

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

limitation: only gives good classification if categories are linearly separable

Over-fitting

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"

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

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

Regression

general regression problem: $\mathbf{y} = \mathbf{f}(\mathbf{x})$

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

-> find mapping function, **f**

compare to classification:

outputs are continuous, not categorical

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

applications:

system identification

find best-fitting parameters of an encoding model

linear regression: $\mathbf{y} = \mathbf{w} \cdot \mathbf{x}$

training dataset: **x** = inputs: $x_1, x_2, \ldots x_N$

y = outputs: $y_1, y_2, \ldots y_M$

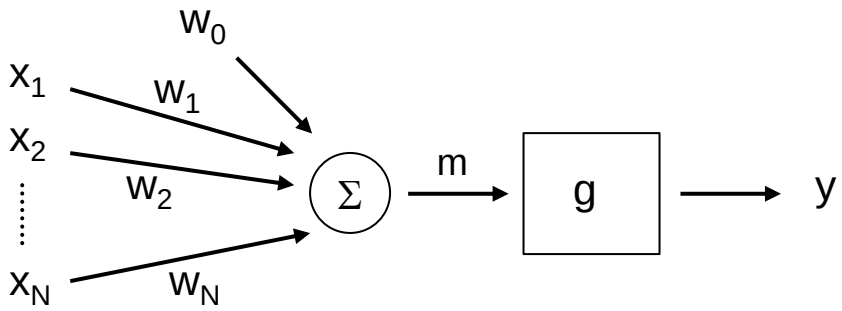
-> find **w** = weights: $w_{11}, w_{12}, \ldots w_{1M}, \ldots w_{N1}, \ldots w_{NM}$

Generalized Linear Model (GLM)

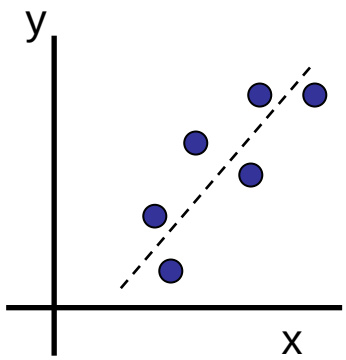
$y = g(m)$

$m = \Sigma\, w_i x_i + w_0$ = linear weighted sum of the inputs

model architecture



simple case: $y = w_1 x + w_0$



x_i = firing rates of afferents
w_i = synaptic weights
m = membrane potential of neuron
g = nonlinearity, w. threshold
y = response of neuron (firing rate)

g = nonlinearity, e.g. half-square:

$y = m^a$ if $m \geq 0$

$y = 0$ if $m < 0$

model parameters: weights, w_i , parameters of the nonlinearity, e.g. *a*

"*generalized linear model*" (GLM): guaranteed to have a <u>unique minimum</u>

Over-fitting

- too many parameters, too little data ... -> "over-fitting"

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

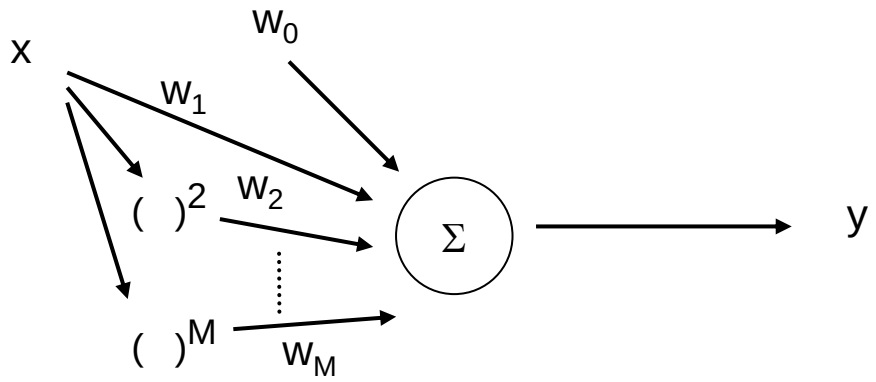
simple example: fitting a *high*-order polynomial to a *small* number of data points

-> the fitted curve will fit those few points extremely well,
but that it handles *new* points very poorly

"model" – polynomial, a linear weighted sum of input values, raised to different powers – note that it is linear in the coefficients w_i :

$$y = f(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \ldots + w_M x^M$$

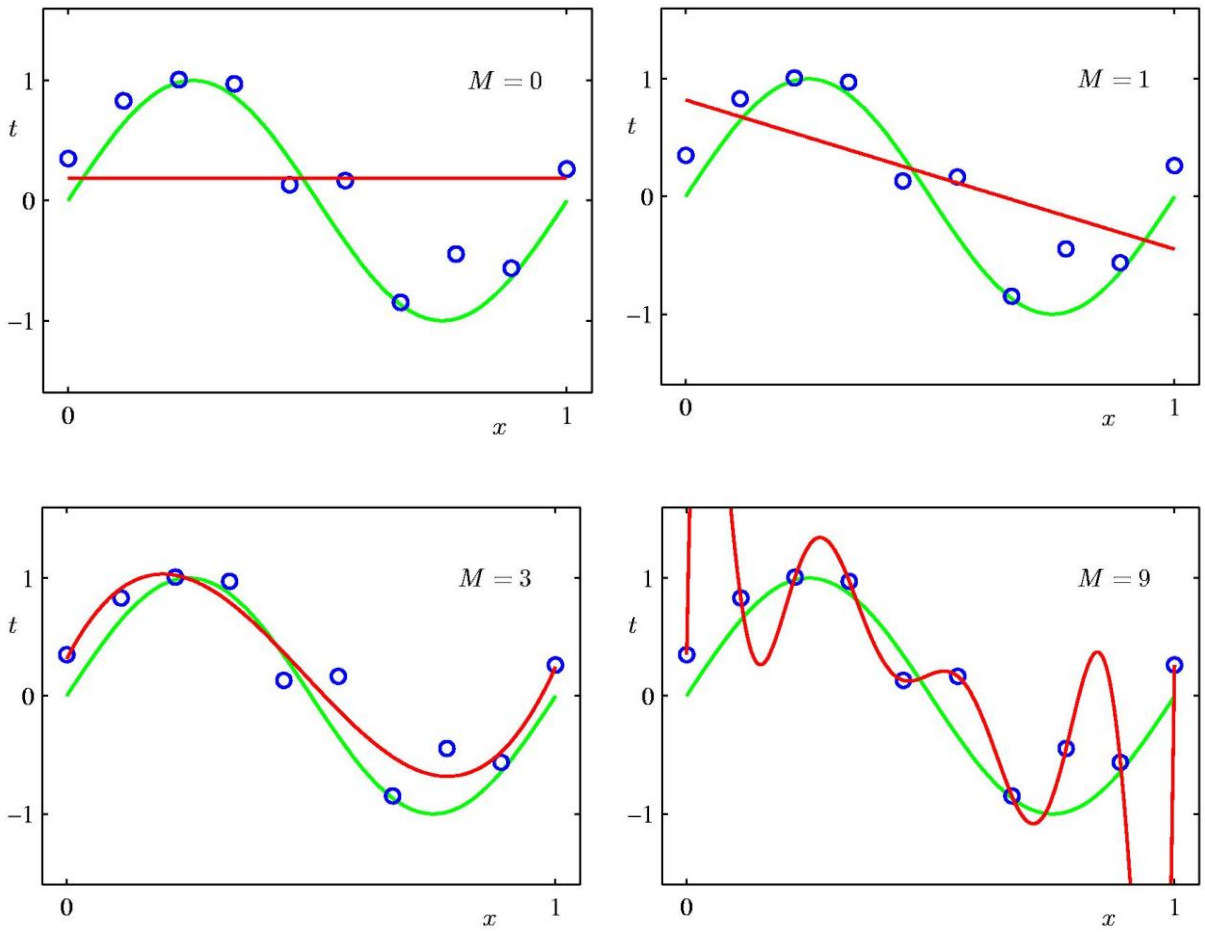
Note, this is still a linear regression problem !



Over-fitting

"model" – polynomial, a linear weighted sum of input values, raised to different powers :

$$y = f(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \ldots + w_M x^M$$



N = 10 data points

M = order of polynomial
-> "model complexity"

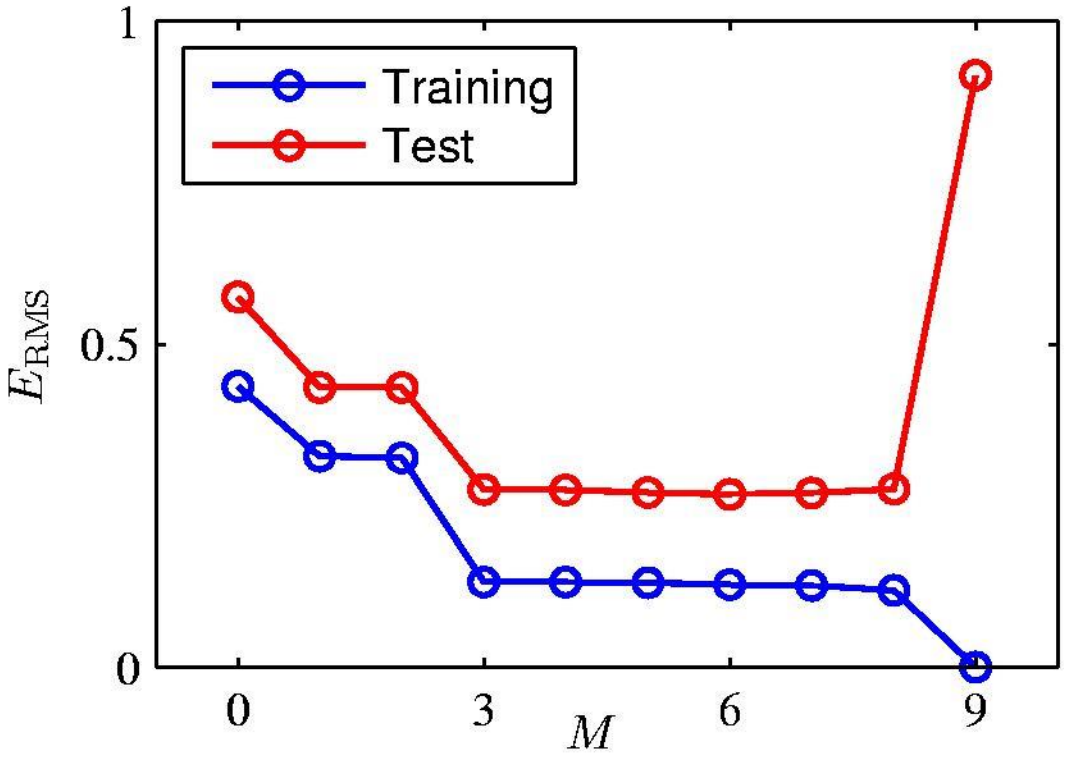
-> over-fitting !

adapted from Bishop (2006), Fig.s 1.4 - 1.6

over-fitting

how to detect over-fitting:

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



N = 10 data points

M = order of polynomial

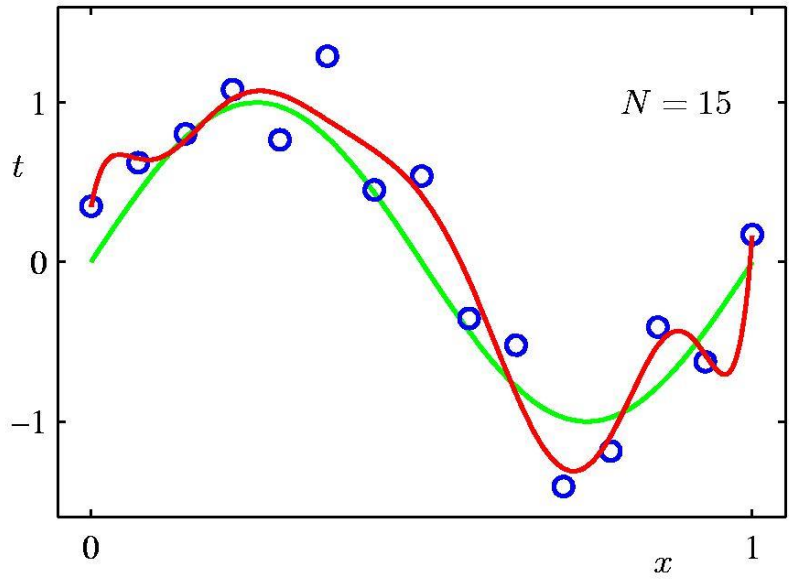
-> "model complexity"

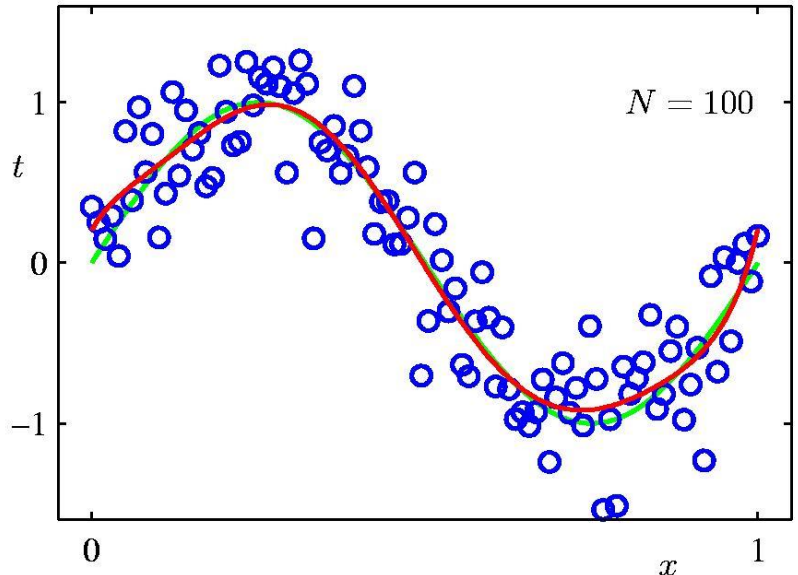
Root-Mean-Square (RMS) Error: $E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^\star)/N}$

Bishop (2006)

preventing over-fitting: more data

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





N = number of measurements

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

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

preventing over-fitting: early stopping

- divide up data into two sets

  Training data – use this data for fitting parameters

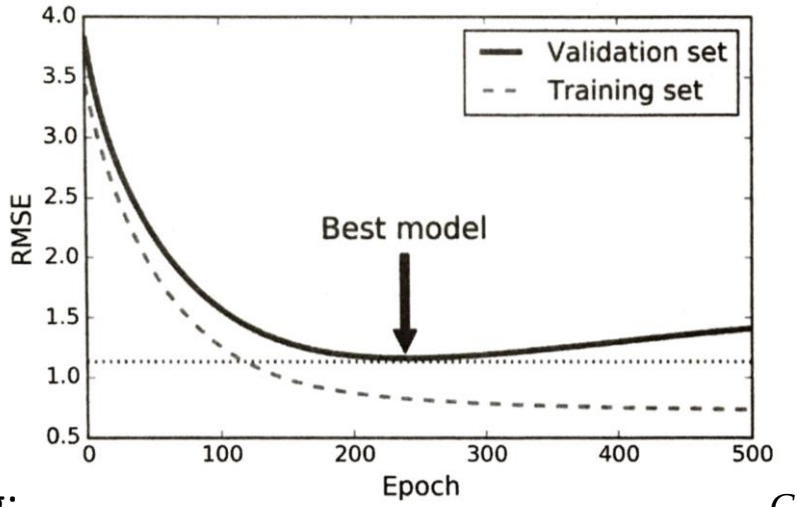
  Validation data – "hold-back" data, to measure generalization

  - can the trained model predict this dataset ?

- for every iteration of training, evaluate the estimated model (so far), on both the Training and Validation datasets, and compare

- if Validation errors > Training errors: over-fitting !

  - the greater the difference, the greater the over-fitting



Géron (2019)

- therefore, to minimize overfitting:

  - when error on Validation data starts to <u>increase</u>, STOP

  (if you keep going, over-fitting will get *worse*)

preventing over-fitting: use "prior knowledge"

When over-fitting occurs, often the parameter values grow to abnormally large values. To minimize this, try to discourage large parameter values

-> modify 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"

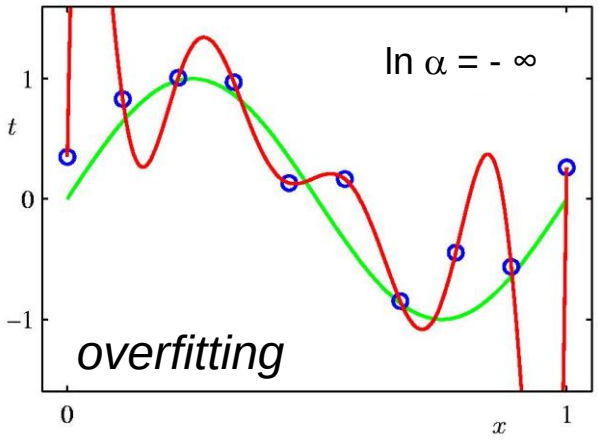
-> apply penalty only during training, not to measurement of error in training or validation

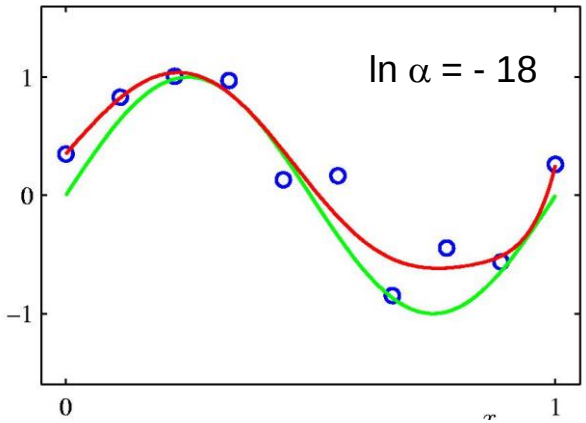
what value of α to use ?

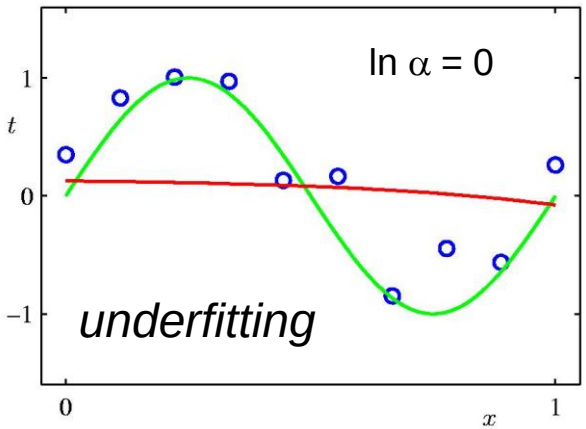
-> α that best predicts another "hold-back" dataset (not used for training)

- need to optimize α - typically done with a grid-search (i.e. try each of a series of values, and use the one that works best

ridge regression for the polynomial fitting example



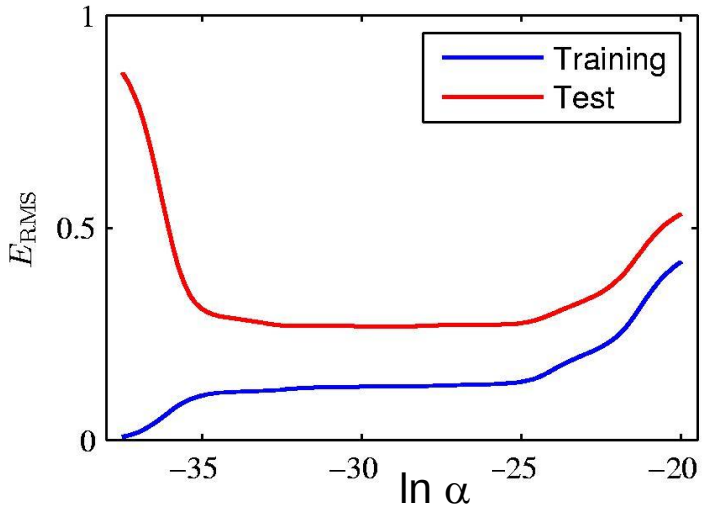




increasing alpha (α)

penalize large coefficient values:

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



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

regression vs. reverse correlation

for a linear system: $Y = H X$ where Y = output, X = input, H = parameter vector (RF)

solve: $H = YX^{-1}$ -> *numerical problems !*

regression (least-squares) solution: $H = (X^T X)^{-1} X^T Y$

if input (X) = white noise, then $X^T X$ is just the identity matrix, and we have:
$H = X^T Y$ = cross-correlation between input and output.

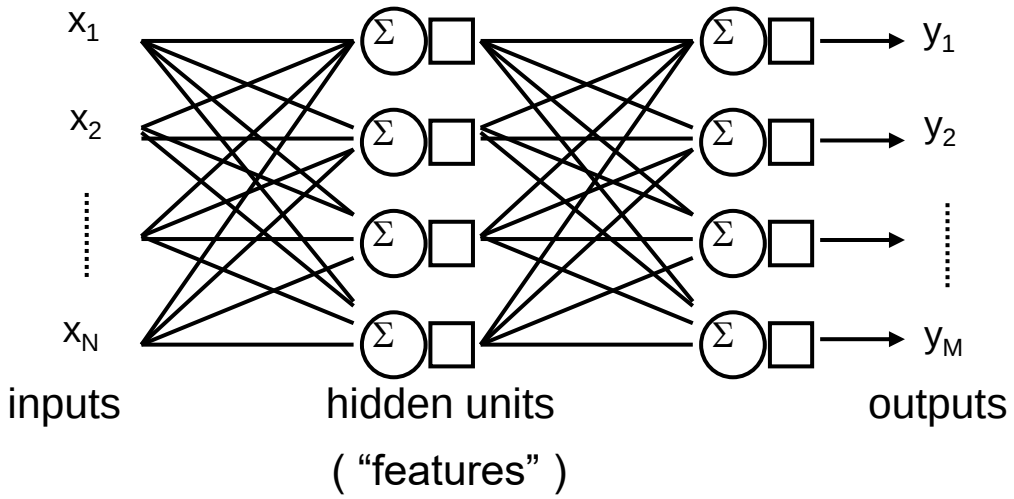
If the output is a spike train, this is equivalent to *reverse correlation*.

reverse correlation - special case of regression
inputs *must* be uncorrelated, i.e. white noise

regression
inputs *can* be correlated (not "white")
stimuli can be filtered by preprocessors
natural images, sounds, etc

regression with gradient descent – why not just "do the math" ?
many parameters, large datasets -> matrix too big to invert
matrix might be difficult to invert, e.g. natural images
more ways to apply regularization
more ways to do optimization

deep neural networks



training: "back-propagation"

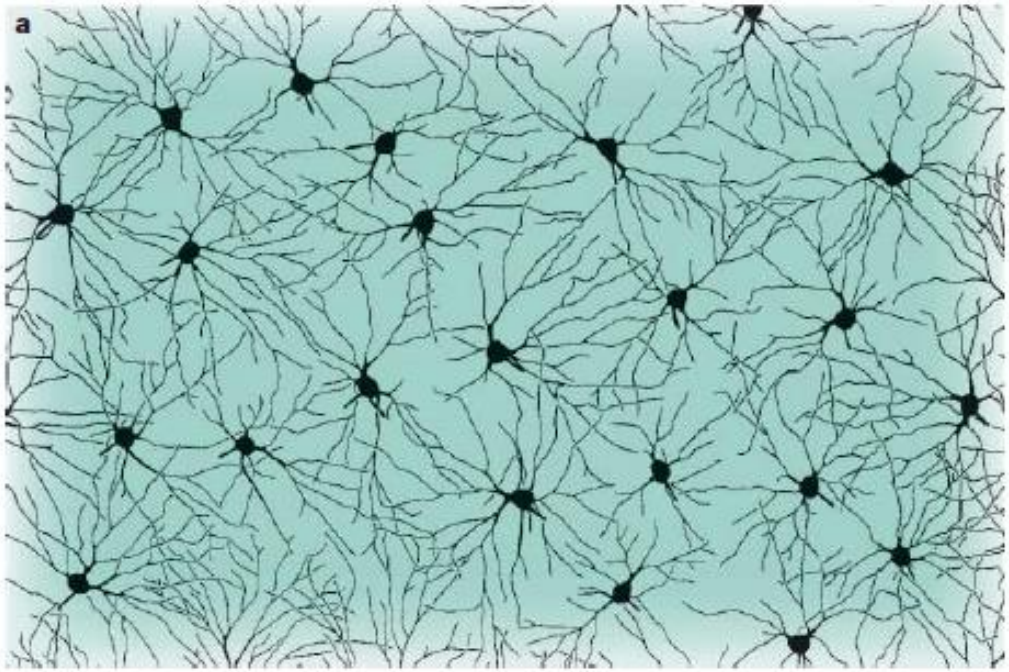
problems with multi-layer ("deep") neural networks

- too many connections -> too many weights (parameters)
- over-fitting
- convergence, local minima, etc
- computation demands

one approach: don't try to learn the hidden units, use "hand-crafted" features
-> only need to learn the final set of weights (just a GLM)

or ... deep learning ...

inspiration from neurobiology: “tiling” of visual receptive fields



from Wässle (2004)

dendritic fields, and therefore receptive fields, of ON-alpha retinal ganglion cells:

•homogeneous – identical, but in different positions

•complete coverage

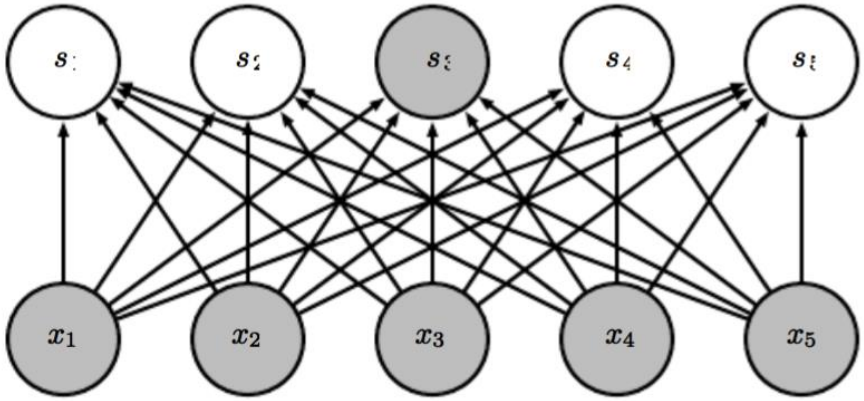
•minimally overlapping

key ideas to use for deep neural networks:

- output of array of such cells = convolution of a filter (RF) with the visual stimulus
- a given layer (e.g. retina) might contain several such filters
  (e.g. ON and OFF-M, P cells), acting in parallel
- hierarchy of successive layers (e.g. retina, LGN, V1, V2, ...),
  each having its own set of homogeneous filters

advantages of *convolutional* networks:

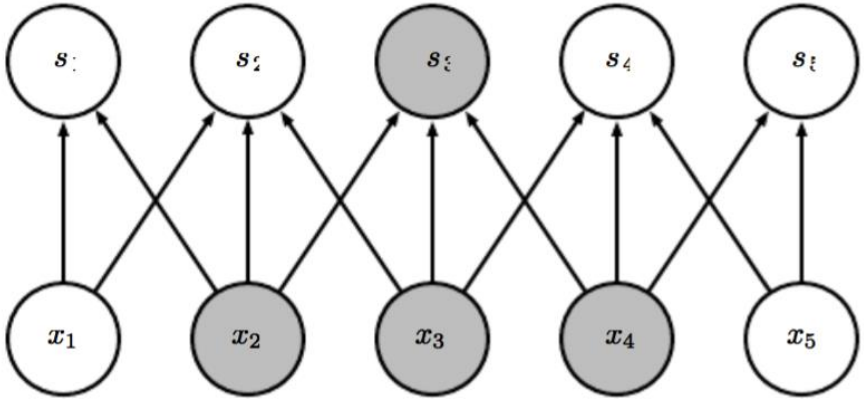
traditional neural network -> many parameters to learn



"dense" / "fully connected":

- each unit gets input from all lower-level units
- no homogeneity
- many weights to learn

"parameter-sharing" -> many fewer parameters to learn



convolutional:

- each unit gets input from only limited set of lower-level units
- homogeneity – "parameter sharing"
- fewer weights to learn

from Deep Learning, Goodfellow et al, Fig 9.3 (2016)

Pooling

reduce dimensionality, by combining adjacent “neurons”

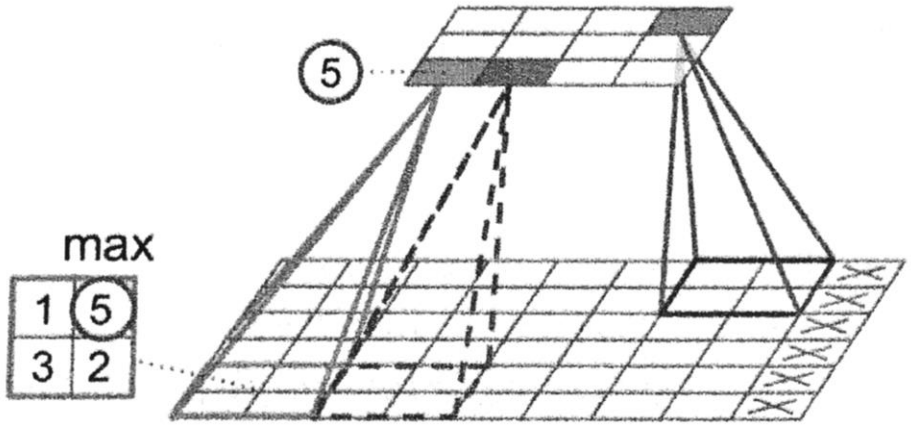
e.g. replace each 2x2 block by just one

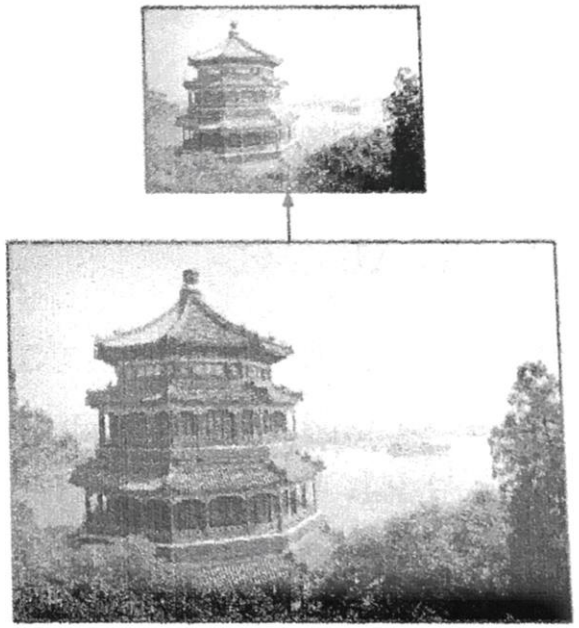
benefits: fewer model parameters to learn -> less overfitting

less computing, memory load

most common kinds of pooling:

- average pooling: take the mean
- max pooling: take the largest





Geron, Fig 13-8, p 364

making multi-layer (deep) neural networks work:

- too many connections / too many weights (parameters)

  -> *convolutional filters, pooling*

- over-fitting

  -> *regularization, e.g. dropout*

- convergence, local minima, etc

  -> *better activation functions (without saturation), e.g. half-wave rectification (ReLU)*

- computation demands

  -> *faster computers, bigger data*

  -> *better algorithms (e.g. stochastic gradient descent)*

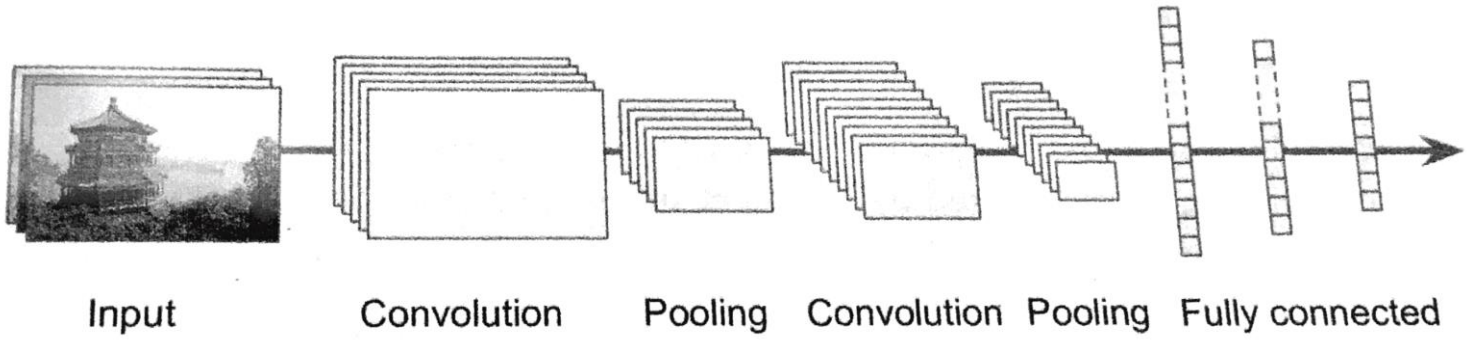
  -> *GPUs (graphical processing units)*

https://www.slideshare.net/AliKheyrollahi/deep-learning-for-developers-oredev

model architecture for deep neural networks

succession of “layers”, alternating between convolution and pooling layers,
followed by one or more fully connected layers

each convolution layer has one or more convolution filters,
each with its own pooling



Geron, Fig 13-9, p 365

Optional supplementary reading, references:

textbook: 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

related courses at McGill: "Fundamentals of Machine Learning" (Comp 451)
"Applied Machine Learning" (COMP 551)
"Machine Learning" (COMP 652 / ECSE 608)

books:

Information Theory, Inference, and Learning Algorithms, by David MacKay
http://www.inference.phy.cam.ac.uk/mackay/itila/book.html

Pattern Recognition and Machine Learning by Christopher Bishop. Springer, 2011.
suppl materials -> http://research.microsoft.com/en-us/um/people/cmbishop/PRML/index.htm

Deep Learning by Ian Goodfellow, Yoshua Bengio & Aaron Courville. MIT Press, 2016.

Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Ed) by Aurélien Géron. O'Reilly, 2019.
-> available as eBook from McGill Library

journal articles:

•LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521:436-444.

•LeCun Y, Kavukcuoglu K, Farabet (2010) Convolutional networks and applications in vision. Proc IEEE Intern Symp Circuits and Systems. DOI 10.1109/ISCAS.2010.5537907

•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.

•Wässle H (2004) Parallel processing in the mammalian retina. Nature Reviews Neuroscience 5:1-11.

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

On-line resources:

NetLab - http://www.aston.ac.uk/eas/research/groups/ncrg/resources/netlab/
STRFlab - http://strflab.berkeley.edu/
Bruno Olshausen's course (UCBerkeley): redwood.berkeley.edu/wiki/VS298:_Neural_Computation
Pascal Vincent (UdeM), slides for "Deep Learning 101": http://www.crm.umontreal.ca/2017/MAIN2017/pdf/Vincent_MAIN_2017.pdf
TensorFlow: https://www.tensorflow.org/
ConvNet course at Stanford (many course materials): http://cs231n.stanford.edu/