02457 Signal Processing in Non-linear Systems: Lecture 5

Perceptrons and backpropagation

Ole Winther

Technical University of Denmark (DTU)

September 22, 2016

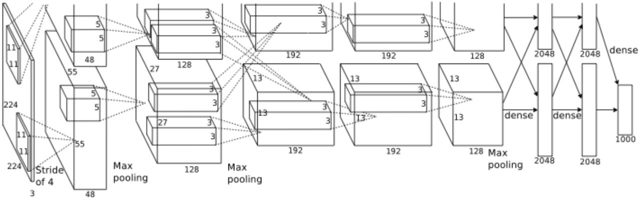
The deep learning revolution – 2012-present

ImageNet Classification with Deep Convolutional Neural Networks

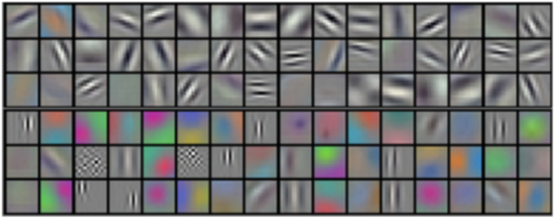
Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

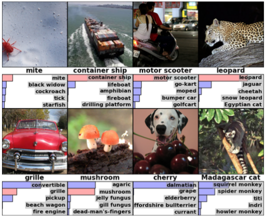
Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

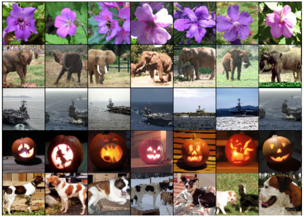


AlexNet

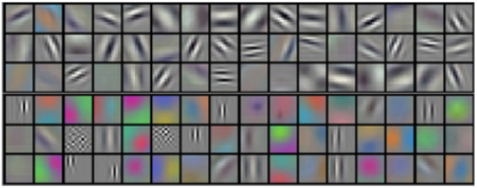


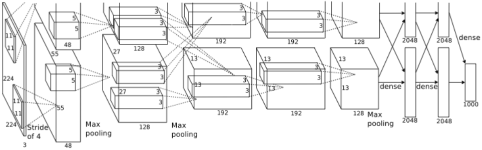


mite
container ship
motor scooter
leopard
mite
black widow
cockroach
tick
starfish
container ship
lifeboat
amphibian
fireboat
drilling platform
motor scooter
go-kart
moped
bumper car
golfcart
leopard
jaguar
cheetah
snow leopard
Egyptian cat
grille
mushroom
cherry
Madagascar cat
convertible
grille
pickup
beach wagon
fire engine
agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers
dalmatian
grape
elderberry
ffordshire bullterrier
currant
squirrel monkey
spider monkey
titi
indri
howler monkey



Understanding AlexNet





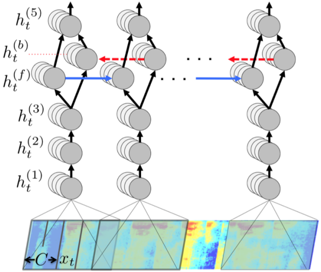
Yosinski et. al., ICML, `https://youtu.be/AgkfIQ4IGaM`

Recurrent neural networks – DeepSpeech

DeepSpeech: Scaling up end-to-end speech recognition

Awni Hannun,* Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, Andrew Y. Ng

Baidu Research – Silicon Valley AI Lab



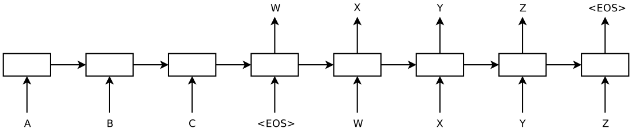
Encoder-decoder - machine translation

Sequence to Sequence Learning with Neural Networks

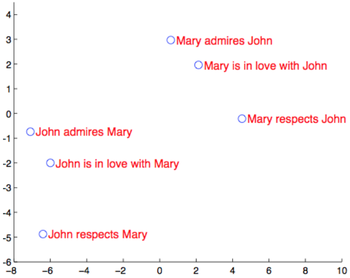
Ilya Sutskever
Google
ilyasu@google.com

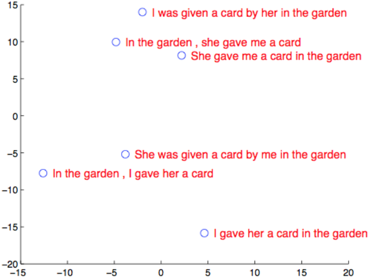
Oriol Vinyals
Google
vinyals@google.com

Quoc V. Le
Google
qvl@google.com



Machine translation - visualising latent space

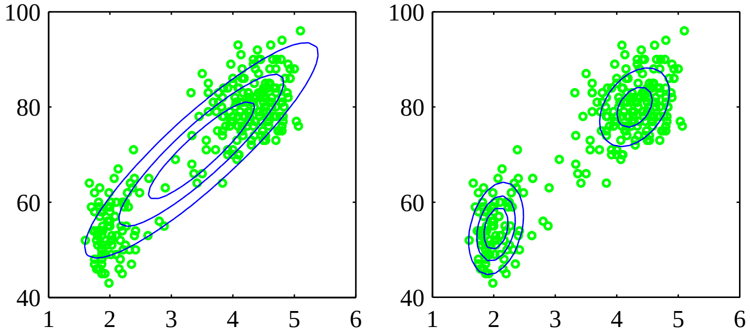




- Hour 1
  - Summary - learning and generalization
  - Exercise 4 - walk through
- Hour 2
  - Your turn! Getting some intuition about neural network
  - Multi-layer perceptrons aka feed-forward neural networks
  - Your turn! Universal approximator
- Hour 3
  - Neural network training
  - Summary and reading material
  - Your turn! Error backpropagation
- Next time:
  - Advanced non-linear optimization (use already this week)
  - Tricks of the trade
  - Neural networks for classification (signal detection)

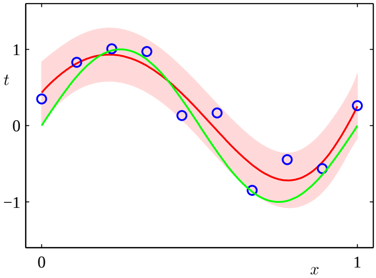
Unsupervised learning

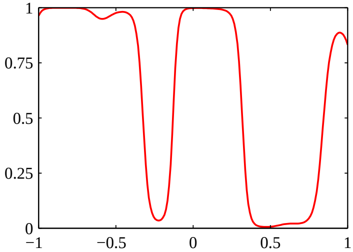
- Learning the distribution of a set of variables $p(\text{input})$.
- Or perhaps just some important characteristics of the distribution

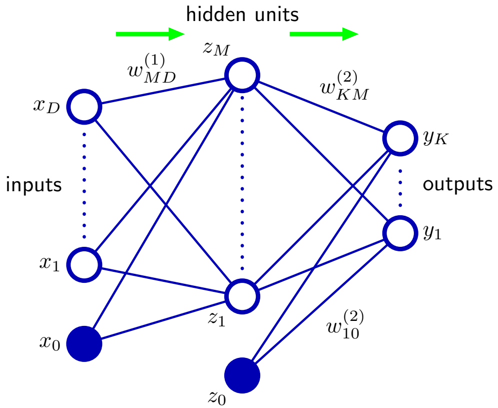


Supervised learning

- Learning the conditional distribution $p(\mathrm{output}|\mathrm{input})$.
- Regression – output continuous
- Classification – output discrete (e.g. positive diagnosis)





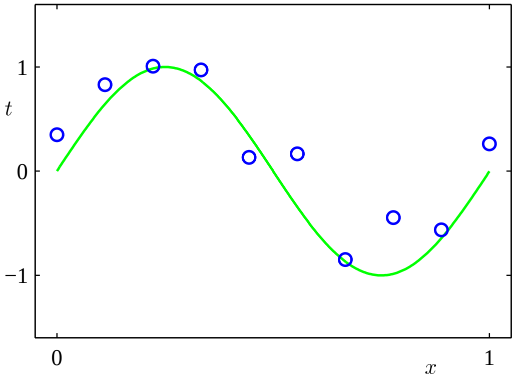


Neural networks aka multi-layer perceptrons today's topic, but first summary!

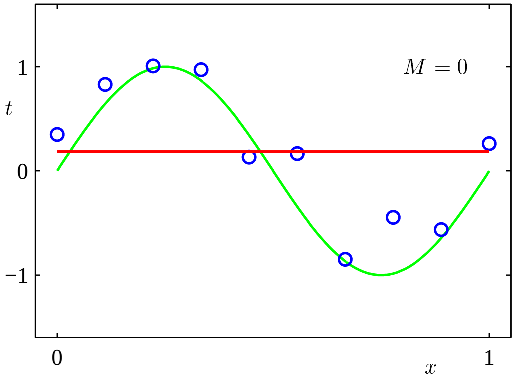
Summary previous lecture

- Generalization – the agenda of machine learning
  - Controlling model complexity
  - Bias-variance trade-off
  - Test set and cross-validation
- Training sets and models
- Likelihood function
- Linear regression as running example
- Bayesian approach
- Maximum likelihood

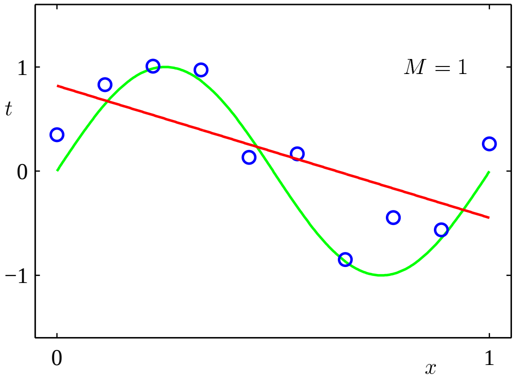
Learning with polynomial model



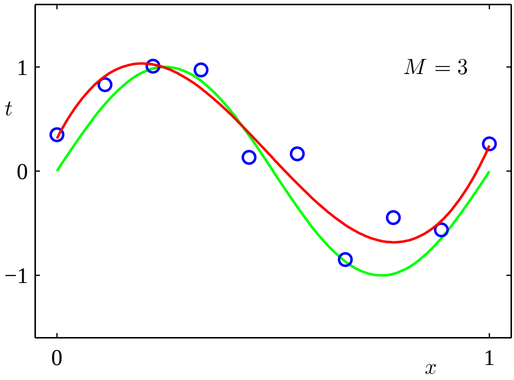
Learning with polynomial model $M = 0$



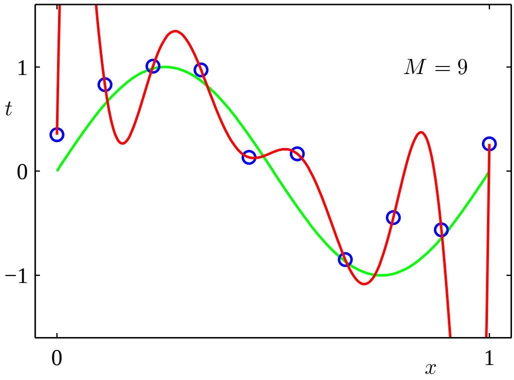
Learning with polynomial model $M = 1$



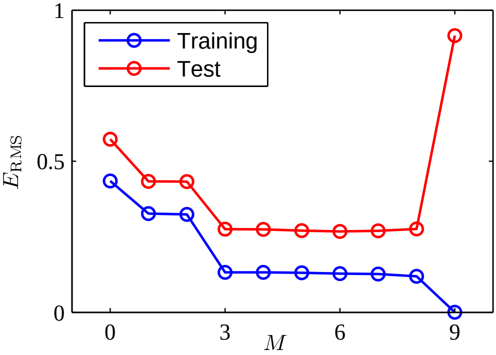
Learning with polynomial model $M = 3$



Learning with polynomial model $M = 9$

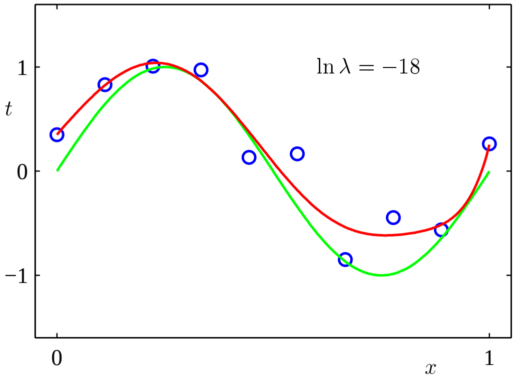


Performance versus model complexity

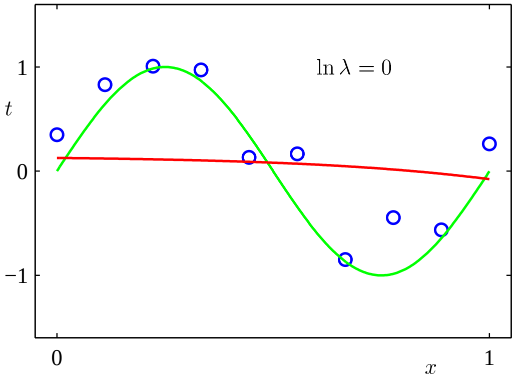


Turn to page 8 in Bishop: Explain Table 1.1.

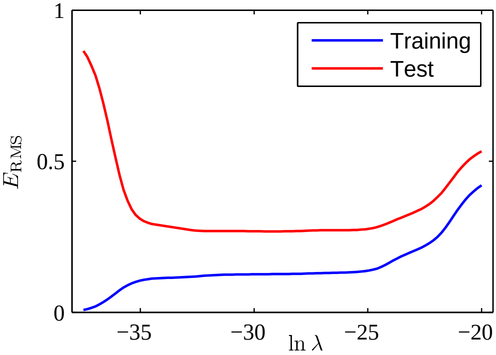
Weight decay regularization for $M = 9$



Weight decay regularization for $M = 9$

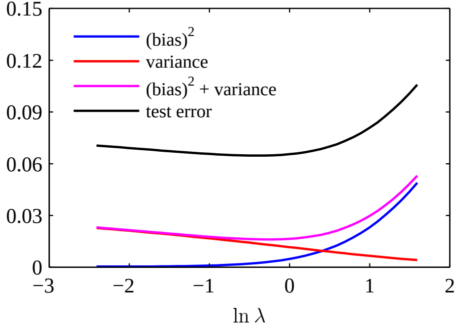


Performance versus model complexity



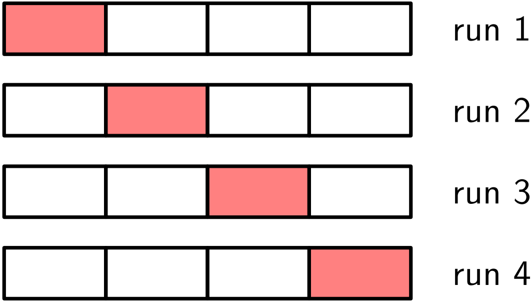
Turn to page 11 in Bishop: Explain Table 1.2.

Bias variance trade-off



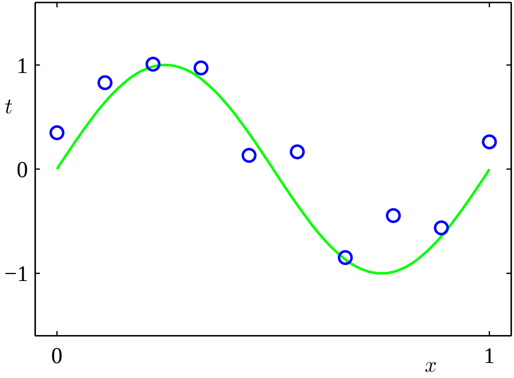
$$\text{Gen Err} = \text{Noise} + \text{Bias}^2 + \text{Variance}$$

Cross-validation

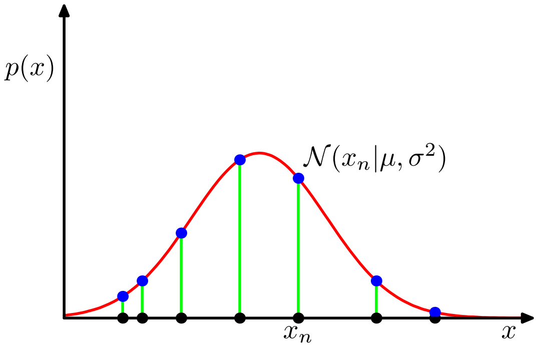


K-fold split (here 4) : train K times on $K-1$ parts and test on 1.

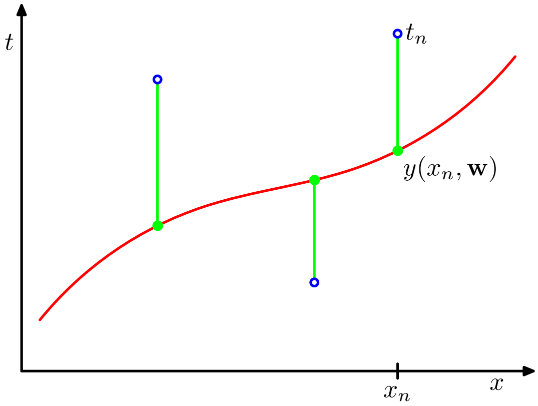
Training set and model



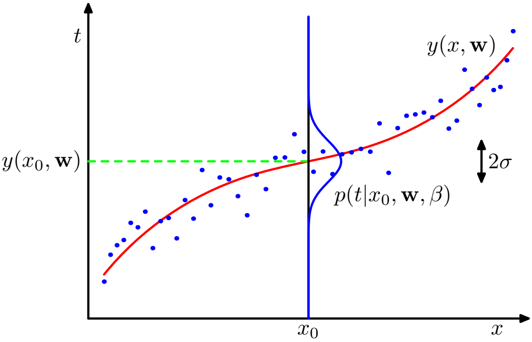
Likelihood function

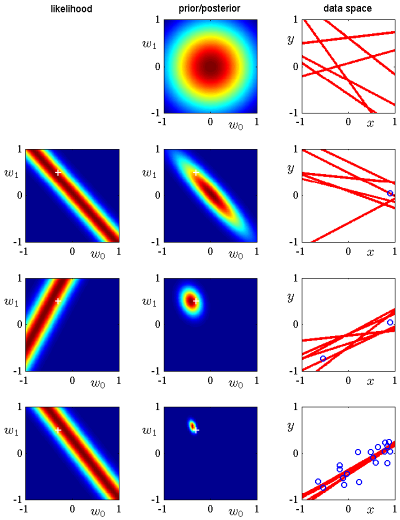


Likelihood function regression



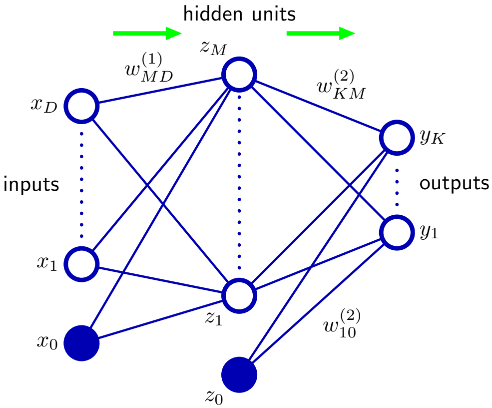
Regression




likelihood
prior/posterior
data space
w_1
w_0
y
x

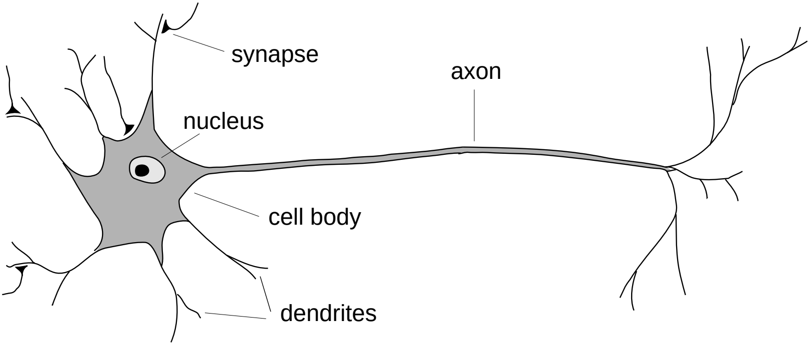
TensorFlow playground

- Your turn!
- Go to `http://playground.tensorflow.org`
- Questions:
  1. Change model so that it implements a linear decision boundary.
  2. Change model so that it overfits.
  3. Change model so that it underfits.
  4. Can you make training find local minima?
  5. What are the squares at each hidden and
  6. the arrows between units?
  7. Change to ReLU activation. Explain polyhedral like shape of decision boundary.
  8. Change to spiral dataset - is this a harder problem? Why?
  9. Can we say something about generalisation performance?

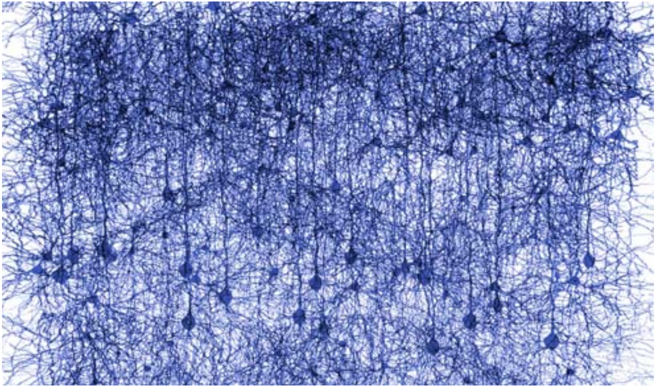


[Thanks to Anders Krogh for neural network slide material.]

Biological CPU (neuron)

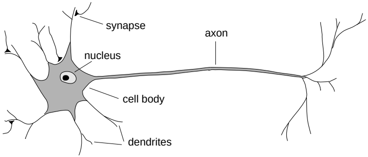


Brain: massively parallel computer

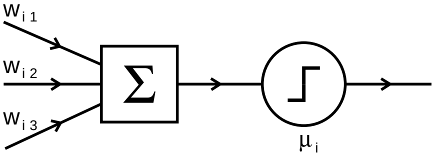


Approx. 10^{11} neurons and 10^{14} synapses in a human brain

- Neuron receives input through the synapses and dendrites
- Synapses are inhibitatory or exitatory
- If electric potential exceeds a threshold, the neuron “fires”
- Sends electric pulses to other neurons through the axon



- Early models of the brain is based on the McCulloch-Pitts model (1943)

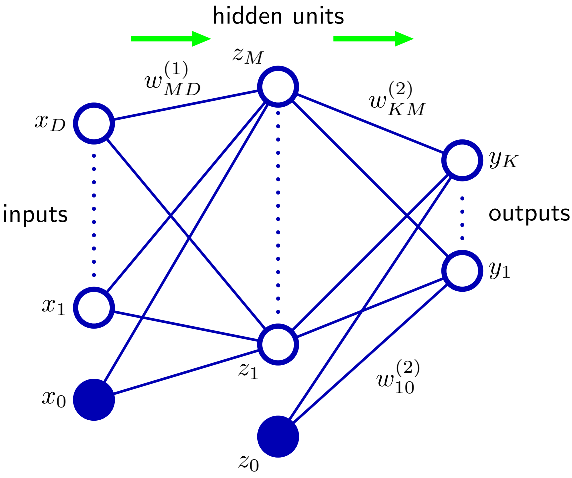


$$y(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{i=1}^{M} w_i x_i + w_0\right)$$

- σ is a threshold function
- $-w_0$ is the threshold.

- Rosenblatt (1950s) showed that perceptrons could “learn” from examples
- This research became unpopular – it didn’t work for XOR
- In the 1980s:
    - backpropagation learning is discovered for multi-layer perceptrons
    - learning in Boltmann Machines (recurrent networks) is formulated
    - The Hopfield model of associative memory
- ...

- ...
- This resulted in an explosion of research and lots of hype about clever thinking machines with intuition
- Currently:
  - NN research continues in the modeling of REAL neural networks
  - Absorbed as a method in pattern recognition and machine learning
  - NNs are disliked by some. This is a bit like disliking linear regression.
  - A huge revival rebranded as deep learning!


hidden units
inputs
outputs
z_M
$w^{(1)}_{MD}$
$w^{(2)}_{KM}$
x_D
y_K
x_1
y_1
z_1
x_0
$w^{(2)}_{10}$
z_0

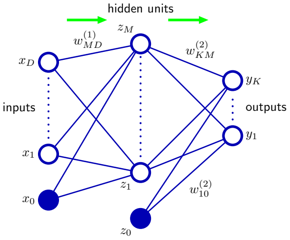
- Soft threshold units
- In each layer add an additional activation (x_0 or z_0) clamped to 1

$$\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

- Output k two-layer network:

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right)$$

- h activation function of the hidden units
- How do we count layers?



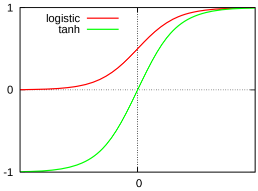
- Activation function
- Logistic function

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

- Hyperbolic tangent

$$\tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}$$

- Linear activation functions will give a linear network.
- Can be mixed, e.g. linear for output units and tanh for hidden units.

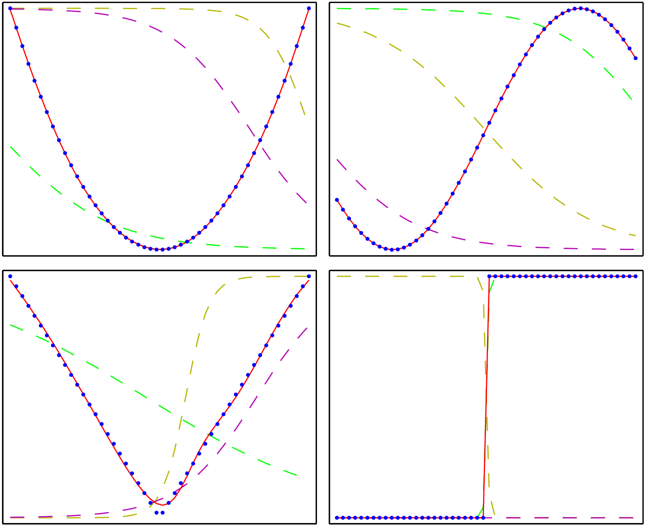


Your turn!

- Consider one input, one hidden layer and one linear output unit:

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=1}^{M} w_j^{(2)} \tanh\left(w_{j1}^{(1)} x_1 + w_{j0}^{(1)}\right) + w_0^{(2)}$$

- What weight values and how many hidden units should be used to learn the functions sketched on the black board and next slide?
- How will the function look for $\mathbf{x} \to \pm\infty$?
- This neural network is a universal approximator = can learn any function given enough hidden units.



- Let a training set be given by $\mathcal{D} = \{(t_n, \mathbf{x}_n) | n = 1, \ldots, N\}$.
- Regression $t_n \in \mathcal{R}$.
- The mean square error of the model $y(\mathbf{x}, \mathbf{w})$ is given by

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

- Remember relation to maximum likelihood

$$-\log p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2) = \frac{1}{\sigma^2} E(\mathbf{w}) + \frac{N}{2} \log(2\pi\sigma^2)$$

Finding the minimum by gradient descent

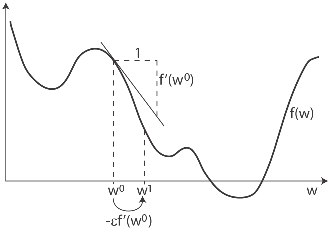
- Iterate:

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} + \Delta\mathbf{w}^{\tau}$$

with

$$\Delta w_{ji}^{\tau} = -\eta \frac{dE(\mathbf{w})}{dw_{ji}}$$

- leads to error back-propagation learning
- which is nothing but clever book-keeping and chain rule of derivatives



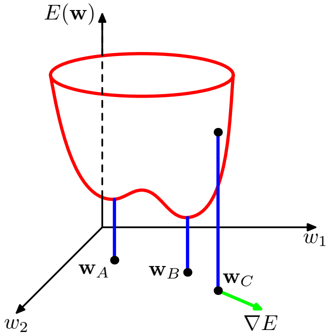
- Objective: to solve the equation

$$\nabla E = 0$$

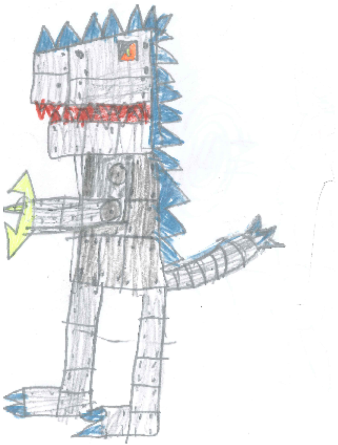
- Gradient descent:

$$\begin{aligned} \mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)} \\ \Delta\mathbf{w}^{(\tau)} &= -\eta \nabla E|_{\mathbf{w}=\mathbf{w}^{(\tau)}} \end{aligned}$$

- η is the learning parameter (rate)
- η can be too small: convergence very slow
- η can be too large: oscillatory behavior



- Summary of course so far:
  - Learning from data
  - Likelihood function, Bayesian and maximum likelihood learning
- Feed-forward neural networks aka multi-layer perceptrons
- Regression today (and classification next week)
- Iterative parameter optimization – error backpropagation



- Neural networks – Bishop 5.1-5.4. (most focus on 5-5.2.1)
- Alternative free pdf books:
- Hastie, Tibshirani and Friedman, The Elements of Statistical Learning, Springer and
- MacKay, Information Theory, Inference, and Learning Algorithms, Cambridge

Your turn! Error backpropagation

- Consider one input, one hidden layer and one linear output unit and cost function

$$
\begin{aligned}
y(x, \mathbf{w}) &= \sum_{j=1}^{M} w_j^{(2)} \tanh\left(w_{j1}^{(1)} x_1 + w_{j0}^{(1)}\right) + w_0^{(2)} \\
E(\mathbf{w}) &= \frac{1}{2} \sum_{n=1}^{N} (y(\mathbf{x}_n, \mathbf{w}) - t_n)^2
\end{aligned}
$$

- Calculate $\frac{\partial E}{\partial w_j^{(2)}}$ and $\frac{\partial E}{\partial w_{ji}^{(1)}}$.
- Explain how these derivatives are used in the error backpropagation rule? (Hint: See Bishop)
- Advanced question: Write down the general backprop rule.

- The multi-layer perceptron (MLP)
- Linear output and one hidden layer

$$
\begin{aligned}
y_l(\mathbf{x}) &= \sum_{j=0}^{M} w_{lj}^{(2)} z_j \\
z_j &= \tanh\left(\mathbf{w}_j^{(1)\,T} \mathbf{x}\right) \\
z_0 &= 1
\end{aligned}
$$

- We compute the gradient w.r.t. any weight in first or second layer u

$$
\begin{aligned}
E &= \sum_{n=1}^{N} (y(\mathbf{x}_n, \mathbf{w}) - t_n)^2 \\
\frac{\partial E}{\partial u} &= \sum_{n=1}^{N} \frac{\partial}{\partial u} (y(\mathbf{x}_n, \mathbf{w}) - t_n)^2 \\
&= 2 \sum_{n=1}^{N} (y(\mathbf{x}_n, \mathbf{w}) - t_n) \frac{\partial y(\mathbf{x}_n, \mathbf{w})}{\partial u}
\end{aligned}
$$

- ...

- ...
- The network derivative for an output unit weight $w_{j'}^{(2)}$:

$$
\begin{aligned}
\frac{\partial y(\mathbf{x}_n, \mathbf{w})}{\partial w_{j'}^{(2)}} &= \frac{\partial}{\partial w_{j'}^{(2)}} \sum_{j=0}^{M} \mathbf{w}_j^{(2)} z_j \\
&= \sum_{j=0}^{M} \frac{\partial}{\partial w_{j'}^{(2)}} w_j^{(2)} z_j \\
&= z_{j'}
\end{aligned}
$$

- The network derivative for a hidden unit weight $w_{j'k'}^{(1)}$ is given by

$$
\begin{aligned}
\frac{\partial y(\mathbf{x}_n, \mathbf{w})}{\partial w_{j'k'}^{(1)}} &= \frac{\partial}{\partial w_{j'k'}^{(1)}} \sum_{j=0}^{M} w_j^{(2)} z_j = \sum_{j=0}^{M} w_j^{(2)} \frac{\partial}{\partial w_{j'k'}^{(1)}} z_j \\
&= w_{j'}^{(2)} \frac{\partial}{\partial w_{j'k'}^{(1)}} \tanh\left(\sum_{k=0}^{d} w_{jk}^{(1)} x_{nk} \right) \\
&= w_{j'}^{(2)} \left(1 - z_{j'}^2\right) \frac{\partial}{\partial w_{j'k'}^{(1)}} \sum_{k=0}^{d} w_{jk}^{(1)} x_k^n \\
&= w_{j'}^{(2)} \left(1 - z_{j'}^2\right) x_{nk'}
\end{aligned}
$$

- Used $\frac{\partial \tanh(a)}{\partial a} = 1 - tanh^2(a)$

- Combining we get for the output weight

$$\begin{aligned}\frac{\partial E}{\partial w_j} &= 2\sum_{n=1}^{N}(y(\mathbf{x}_n)-t_n)z_{nj} \equiv 2\sum_{n=1}^{N}\delta_n z_{nj}\\ \delta_n &= y(\mathbf{x}_n)-t_n\end{aligned}$$

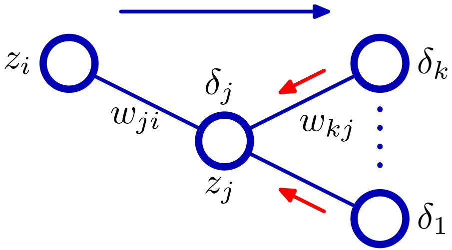
- and for the hidden weight

$$\frac{\partial E}{\partial w_{jk}} = 2\sum_{n=1}^{N}(y(\mathbf{x}_n)-t_n)w_j^{(2)}\left(1-z_{nj}^2\right)x_{nk} \equiv 2\sum_{n=1}^{N}\delta_{nj}x_{nk}$$

- The delta-rule:

$$\delta_{nj} \equiv (y(\mathbf{x}_n)-t_n)\,w_j^{(2)}\left(1-z_{nj}^2\right) = \delta_n\,w_j^{(2)}\,\left(1-z_{nj}^2\right)$$

The general Backprop rule



- Consider a hidden unit $z_j = g(a_j)$, where $a_{nj} = \sum_i w_{ji} z_{ni}$
- ... then the derivative can be expressed

$$\frac{\partial E}{\partial w_{ji}} = \sum_{j',n} \frac{\partial E_n}{\partial a_{nj'}} \frac{\partial a_{nj'}}{\partial w_{ji}} = \sum_n \frac{\partial E_n}{\partial a_{nj}} \frac{\partial a_{nj}}{\partial w_{ji}}$$

- ...
- Let $\delta_{nj} = \frac{\partial E_n}{\partial a_{nj}}$, note also $\frac{\partial a_{nj}}{\partial w_{ji}} = z_{ni}$, this leads to

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \delta_{nj} z_{ni}$$

- Computing the $\delta' s$

$$\begin{aligned} \delta_{nj} &\equiv \frac{\partial E_n}{\partial a_{nj}} = \sum_k \frac{\partial E_n}{\partial a_{nk}} \frac{\partial a_{nk}}{\partial a_{nj}} \\ &= g'(a_{nj}) \sum_k w_{kj} \, \delta_{nk} \end{aligned}$$