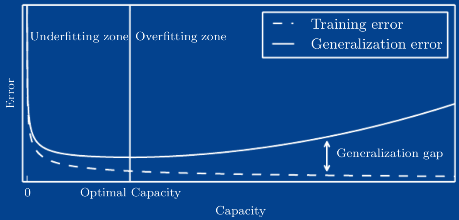
AARHUS
UNIVERSITY
SCHOOL OF ENGINEERING

LESSON 08: Model-capacity, Under/over-fitting, Generalization

CARSTEN EIE FRIGAARD

AUTUMN 2022



"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E." — Mitchell (1997).

L08: Model-capacity, Under/over-fitting, Generalization

Agenda

- J2 feedback (J3 Deadline => 13. Nov. 2022).
- Resumé af GD og NN's.
- Model Capacity,
- Under/over-fitting,

  Exercise: `L08/capacity_under_overfitting.ipynb`

  [OPTIONAL] (enten/eller mht. L09/regulizers.ipynb
- Generalization Error,

  Exercise: `L08/generalization_error.ipynb`

RESUMÉ: GD

The numerically Gradient decent [GD] method is based on the gradient vector

$$\nabla_{\mathbf{w}} J(\mathbf{w})$$

for the gradient oprator

$$\nabla_{\mathbf{w}} = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \dots, \frac{\partial}{\partial w_m}\right]^\top$$

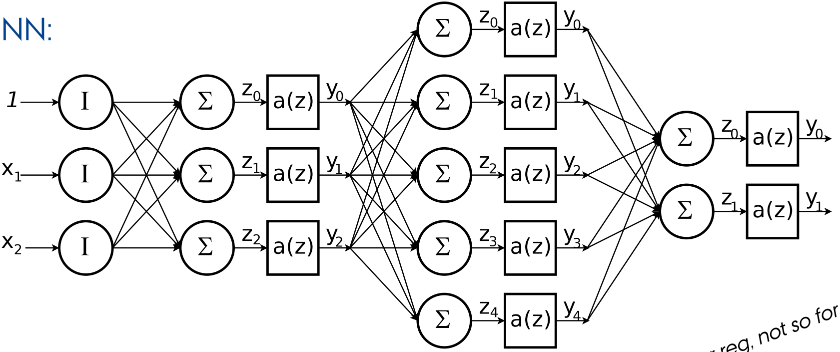
The algoritmn for updating via steps reads

$$\mathbf{w}^{(\text{next step})} = \mathbf{w} - \eta \nabla_{\mathbf{w}} J(\mathbf{w})$$

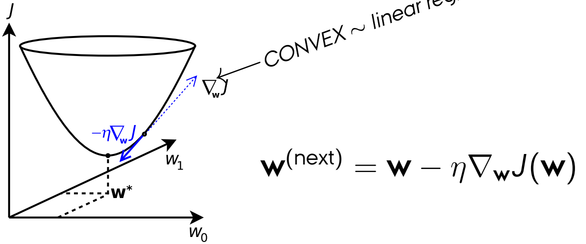
with η being the step size.

RESUMÉ: Training Deep Neural Networks

NN:



GD (via BPROP):

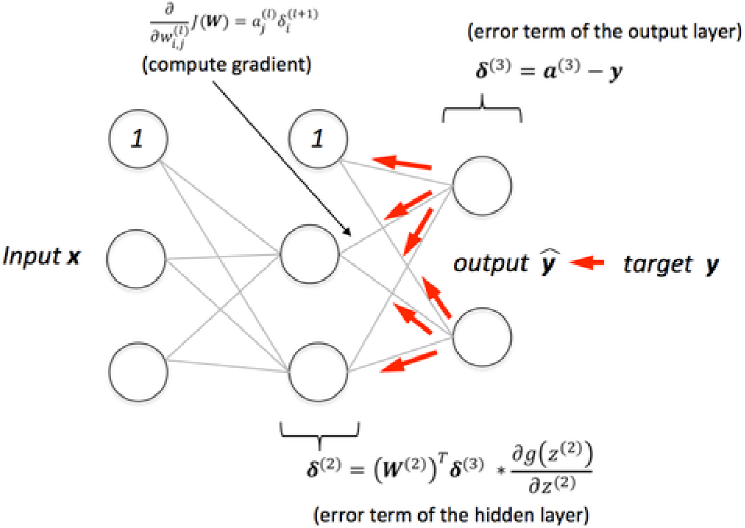


$$\mathbf{w}^{(\mathrm{next})} = \mathbf{w} - \eta \nabla_{\mathbf{w}} J(\mathbf{w})$$

NOTE: NN: Neural net, GD: Gradient Descent, BPROP: Back Propagation

Backpropagation (BProp)

Training MLPs



NOTE: [https://sebastianraschka.com/images/faq/visual-backpropagation/backpropagation.png

RESUMÉ: Training Deep Neural Networks

Equation 4-6. Gradient vector of the cost function

$$\nabla_{\boldsymbol{\theta}} \operatorname{MSE}(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \operatorname{MSE}(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1} \operatorname{MSE}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \operatorname{MSE}(\boldsymbol{\theta}) \end{pmatrix} = \frac{2}{m} \mathbf{X}^T (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

Notice that this formula involves calculation
set **X**, at each Gradient Descent step! This i
called *Batch Gradient Descent*: it uses the w
data at every step (actually, *Full Gradient D*
be a better name). As a result it is terribly sl
ing sets (but we will see much faster Gradient Descent algorithms
shortly). However, Gradient Descent scales we
features; training a Linear Regression model
dreds of thousands of features is much fa
Descent than using the Normal Equation or SV

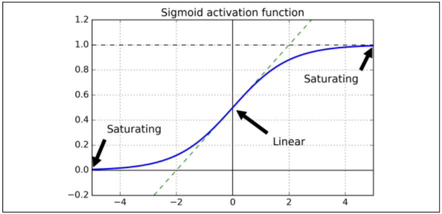


Figure 11-1. Logistic activation function saturation

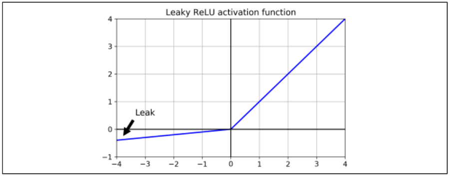


Figure 11-2. Leaky ReLU

Once you have the gradient vector, which points uphill, ju
tion to go downhill. This means subtracting $\nabla_{\boldsymbol{\theta}}\operatorname{MSE}(\boldsymbol{\theta})$
learning rate η comes into play:[6] multiply the gradient v
size of the downhill step (Equation 4-7).

Equation 4-7. Gradient Descent step

$$\boldsymbol{\theta}^{(\text{next step})} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \operatorname{MSE}(\boldsymbol{\theta})$$

$$\mathbf{w}^{(\text{next})} = \mathbf{w} - \eta \nabla_{\mathbf{w}} J(\mathbf{w})$$

Matrix Multiplication

Kunstig intelligens finder nye smarte måder til matrix-multiplikation

$$2 \times 3 + 0 \times 4 = 6$$

$$\begin{bmatrix} 2 & 0 \\ 1 & 9 \end{bmatrix} \times \begin{bmatrix} 3 & 9 \\ 4 & 7 \end{bmatrix} = \begin{bmatrix} 6 & 18 \\ 39 & 72 \end{bmatrix}$$

Multiplikation af to 2x2 matricer kræver to multiplikationer for hvert element i den endelige matrix udført på den sædvanlige måde. Men det kan udføres smartere med kun syv multiplikationer. (Illustration: arkiv)

Der findes mere effektive metoder til matrixmultiplikation end dem, man lærer i skolen. AI-program har fundet endnu bedre metoder end de, som i dag benyttes i computere.

Af Jens Ramskov Følg @jensramskov 5. okt 2022 kl. 17:00 15

[https://ing.dk/artikel/kunstig-intelligens-finder-nye-smarte-maader-matrix-multiplikation-261486]

Matrix Multiplication and Strassen's Algorithm

$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \times \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} = \begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{bmatrix}$$

Standard algorithm

$h_1 = a_{1,1}\, b_{1,1}$

$h_2 = a_{1,1}\, b_{1,2}$

$h_3 = a_{1,2}\, b_{2,1}$

$h_4 = a_{1,2}\, b_{2,2}$

$h_5 = a_{2,1}\, b_{1,1}$

$h_6 = a_{2,1}\, b_{1,2}$

$h_7 = a_{2,2}\, b_{2,1}$

$h_8 = a_{2,2}\, b_{2,2}$

$c_{1,1} = h_1 + h_3$

$c_{1,2} = h_2 + h_4$

$c_{2,1} = h_5 + h_7$

$c_{2,2} = h_6 + h_8$

Strassen's algorithm

$h_1 = (a_{1,1} + a_{2,2})\,(b_{1,1} + b_{2,2})$

$h_2 = (a_{2,1} + a_{2,2})\, b_{1,1}$

$h_3 = a_{1,1}\,(b_{1,2} - b_{2,2})$

$h_4 = a_{2,2}\,(-b_{1,1} + b_{2,1})$

$h_5 = (a_{1,1} + a_{1,2})\, b_{2,2}$

$h_6 = (-a_{1,1} + a_{2,1})\,(b_{1,1} + b_{1,2})$

$h_7 = (a_{1,2} - a_{2,2})(b_{2,1} + b_{2,2})$

$c_{1,1} = h_1 + h_4 - h_5 + h_7$

$c_{1,2} = h_3 + h_5$

$c_{2,1} = h_2 + h_4$

$c_{2,2} = h_1 - h_2 + h_3 + h_6$

MODEL CAPACITY

Model capacity

Exercise: `capacity_under_overfitting.ipynb`

Dummy and Paradox classifier:
capacity fixed ~ 0, cannot generalize at all!

Linear regression for a polynomial model:
capacity $\sim$ degree of the polynomial, x^n

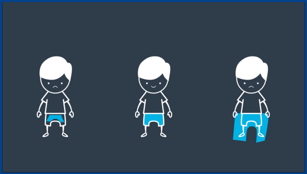
Neural Network model:
capacity $\propto$ number of neurons/layers

Homo sabiens ("modern humans"):
capacity $\propto$ the IQ 'score' function?

$\Rightarrow$ **Capacity** can be hard to express as a quantity for some models, but you need to choose..

$\Longrightarrow$ how to choose the **optimal** *capacity*?

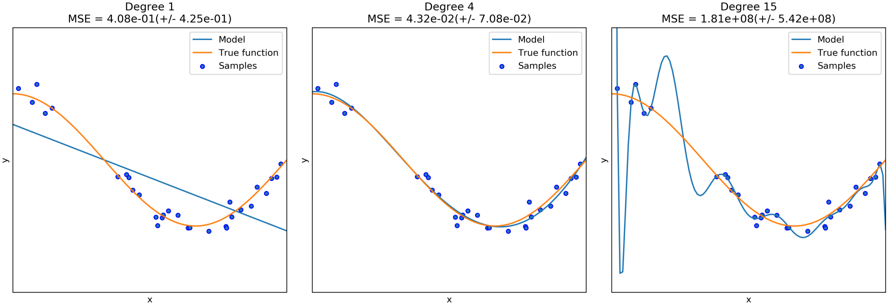
UNDER- AND OVERFITTING



Under- and overfitting

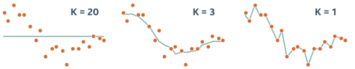
Exercise: `capacity_under_overfitting.ipynb`

Polynomial linear reg. fit for underlying model: `cos(x)`



- underfitting:
  capacity of model too low,
- overfitting:
  capacity to high.

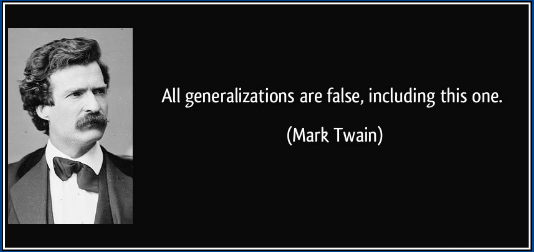
k-NN from L01:



$\Longrightarrow$ how to choose the **optimal** capacity?

NOTE: HOML: *Constraining a model [..] reduce risk of overfitting [via] regularization => L09*

GENERALIZATION ERROR

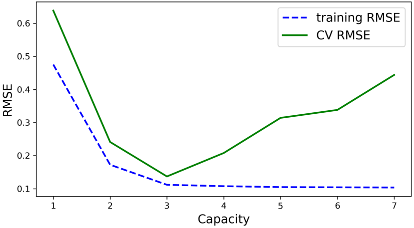


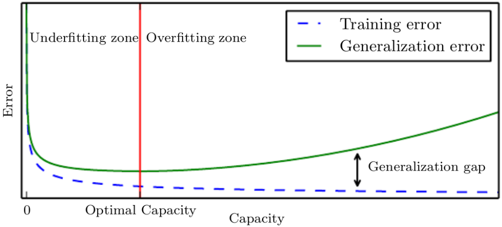
Generalization Error

Exercise: `generalization_error.ipynb`

RMSE-capacity plot for lin. reg. with polynomial features

(capacity ∼ degree of poly) (Figure 5.3 from [DL])





Inspecting the plots from the exercise (`.ipynb`) and [DL], extracting the concepts:

- training/generalization error,
- generalization gab,
- underfit/overfit zone,
- optimal capacity (best-model, early stop),
- (and the two axes: x/capacity, y/error.)

Generalization Error

Definition of ML:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T, as measured by P, improves with experience E."
>
> — Mitchell (1997).

Generalization Error

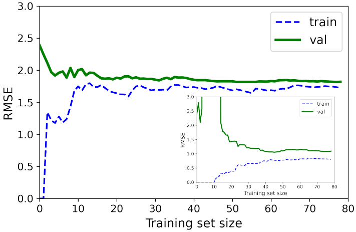
Exercise: `generalization_error.ipynb`

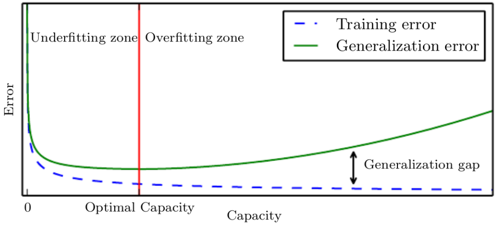
NOTE: three methods/plots:

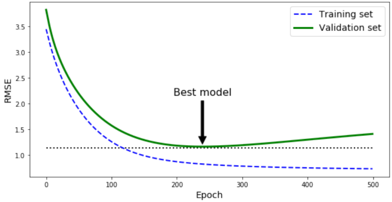
i) via **learning curves** as in [HOML],

ii) via an **error-capacity** plot as in [GITHOML] and [DL],

iii) via an **error-epoch** plot as in [GITHOML].







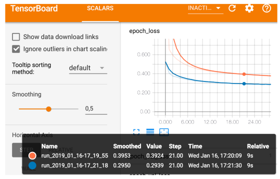


Figure 10-16. Visualizing Learning Curves with TensorBoard