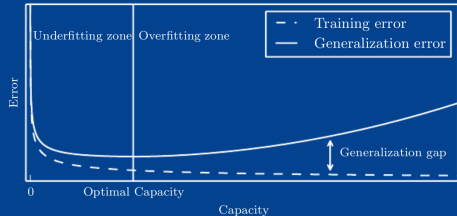




## LESSON 08: Model-capacity, Under/over-fitting, Generalization

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.
- ▶ Résumé 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 operator

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

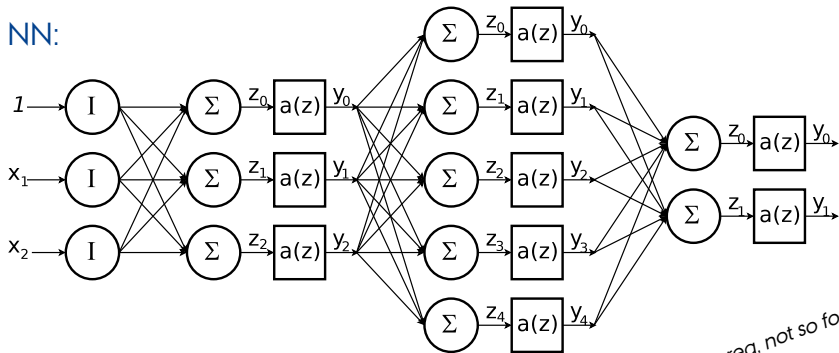
The algorithm for updating via steps reads

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

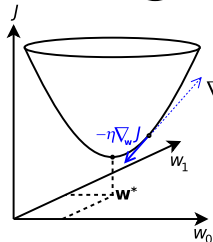
with  $\eta$  being the step size.

# RESUMÉ: Training Deep Neural Networks

NN:



GD (via BPROP):



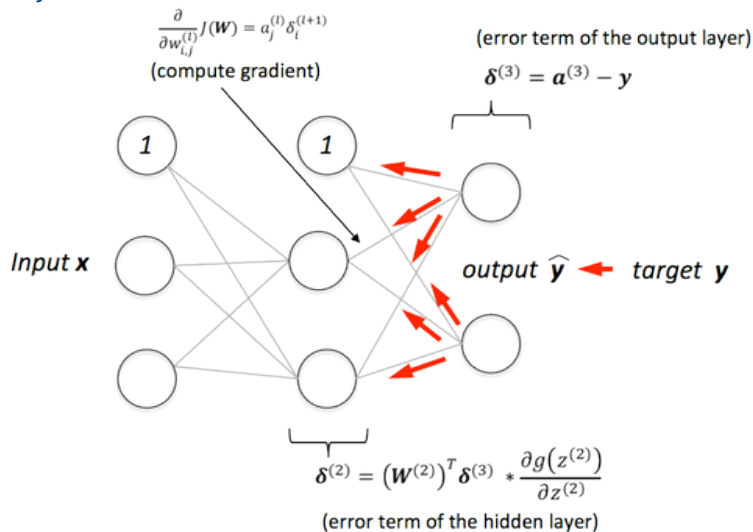
CONVEX  $\sim$  linear reg, not so for NNS

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

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

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



Notice that this formula involves calculation set  $\mathbf{X}$ , at each Gradient Descent step! This is called *Batch Gradient Descent*: it uses the whole data at every step (actually, *Full Gradient Descent* would be a better name). As a result it is terribly slow on large sets (but we will see much faster Gradient Descent algorithms shortly). However, Gradient Descent scales well with many features; training a Linear Regression model with hundreds of thousands of features is much faster with Gradient Descent than using the Normal Equation or Stochastic Gradient Descent.

Once you have the gradient vector, which points uphill, you need a way to go downhill. This means subtracting  $\nabla_{\theta} \text{MSE}(\theta)$  from the current parameters. The learning rate  $\eta$  comes into play:<sup>6</sup> multiply the gradient vector by the learning rate to get the size of the downhill step (Equation 4-7).

Equation 4-7. Gradient Descent step

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

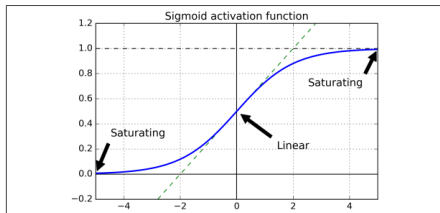


Figure 11-1. Logistic activation function saturation

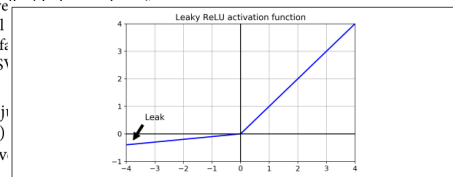
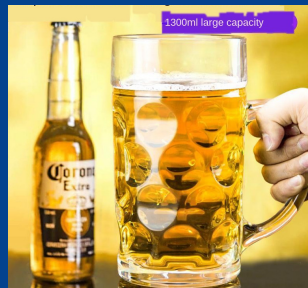


Figure 11-2. Leaky ReLU

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

# MODEL CAPACITY

---



# Model capacity

Exercise: `capacity_under_overfitting.ipynb`

Dummy and Paradox classifier:

*capacity* fixed  $\sim 0$ , 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 sapiens ("modern humans"):

*capacity*  $\propto$  the IQ 'score' function?

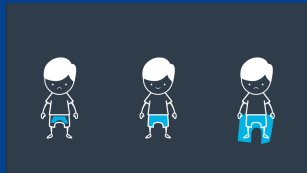
$\Rightarrow$  **Capacity** can be hard to express as a quantity for some models, but you need to choose..

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



# UNDER- AND OVERFITTING

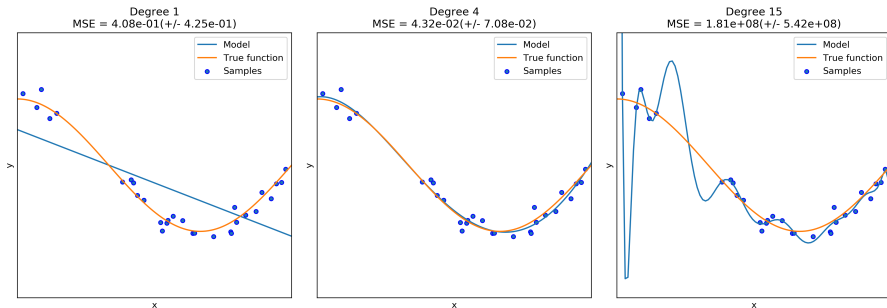
---



# Under- and overfitting

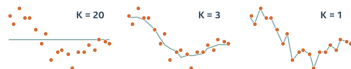
Exercise: `capacity_under_overfitting.ipynb`

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



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

k-NN from L01:

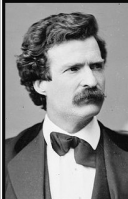


⇒ how to choose the **optimal** capacity?

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

# GENERALIZATION ERROR

---



All generalizations are false, including this one.

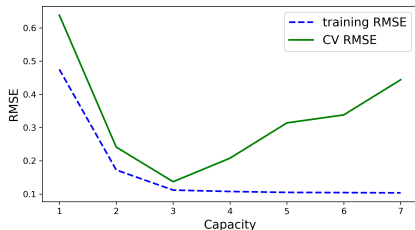
(Mark Twain)

# Generalization Error

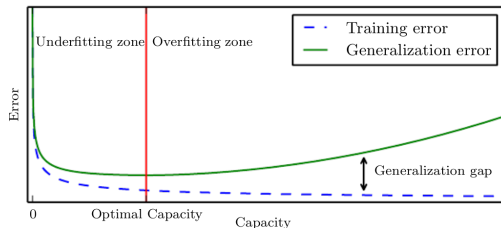
Exercise: `generalization_error.ipynb`

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

(capacity  $\sim$  degree of poly)



(Figure 5.3 from [DL])



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

- ▶ training/generalization error,
- ▶ generalization gap,
- ▶ 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:

- via **learning curves** as in [HOML],
- via an **error-capacity** plot as in [GITHOML] and [DL],
- via an **error-epoch** plot as in [GITHOML].

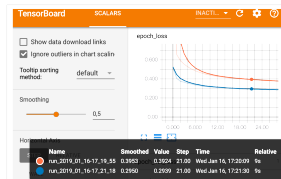
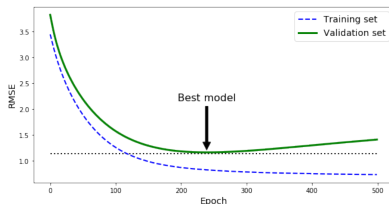
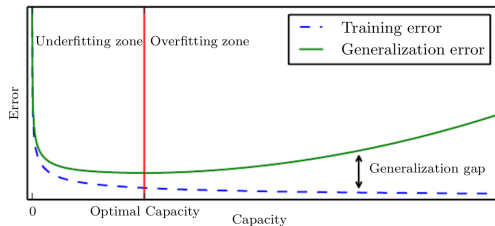
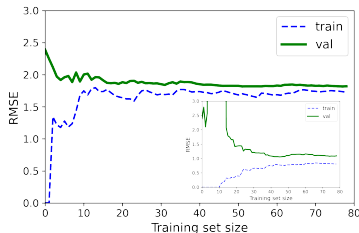


Figure 19.16. Visualizing Learning Curves with TensorBoard