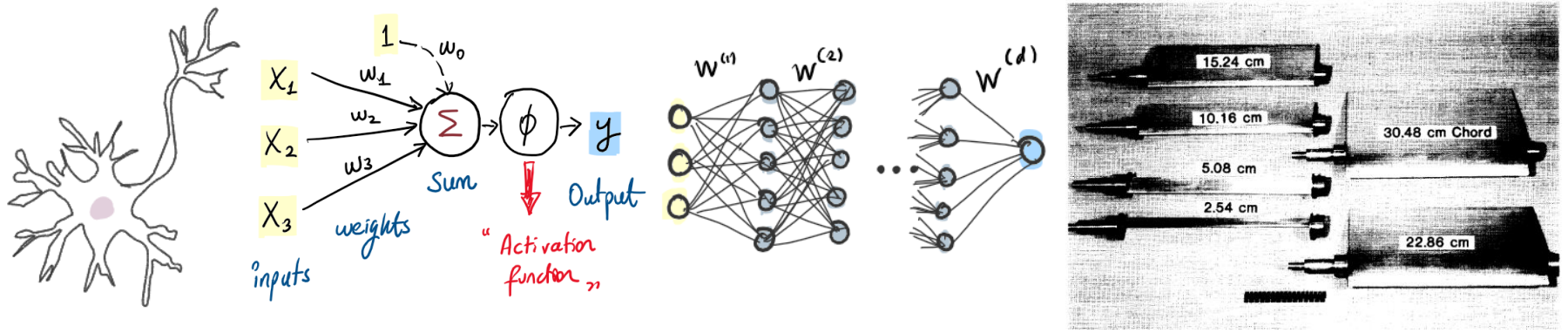
Data Driven Engineering I: Machine Learning for Dynamical Systems

Introduction to Deep Learning

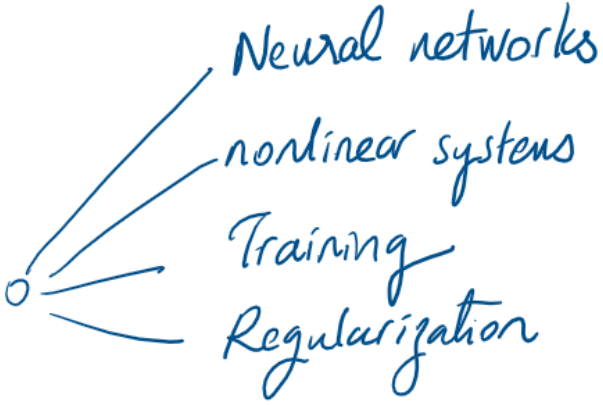
Institute of Thermal Turbomachinery
Prof. Dr.-Ing. Hans-Jörg Bauer



Today's Agenda

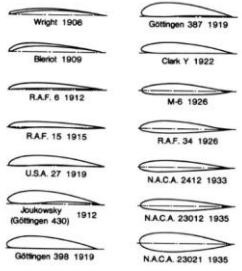
Basic Steps to Follow:

0.) Understand the business/task.
1.) Understand the data.
2.) Explore & prepare the data.
3.) Shortlist candidate models.
4.) Training the model
5.) Evaluate the model predictions.
6.) "Serve" the model!

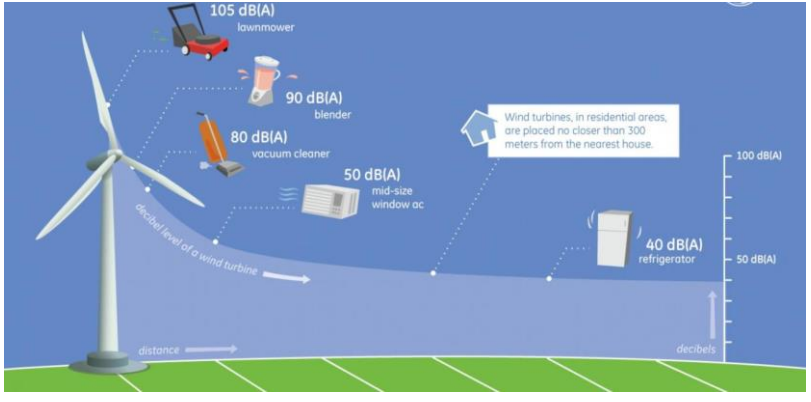


#0 Understanding the task

- **Problem**: NACA 0012 Airfoil Noise Prediction based on Wind Tunnel Testing
- **Noise** generated by an aircraft is an **economic** (efficiency) and **enviromental** issue.
- One component of the noise is **self-noise of the airfoil**: interaction of the airfoil with its own boundary layer



1917, the NACA Technical Report No. 18 titled “Aerofoils and Aerofoil Structural Combinations,” was released.



#0 Understanding the task

- Engineering: semi-empirical models (Brooks)
- Five self-noise mechanisms due to specific boundary-layer phenomena have been identified
- The database is from seven NACA0012 airfoil blade sections of different sizes tested at wind tunnel speeds up to Mach 0.21 and at angles of attack from 0° to 25.2°.
  - ✓ Freq. of noise
  - ✓ Angle of attack
  - ✓ Free stream velocity
  - ✓ Geometry of the airfoil

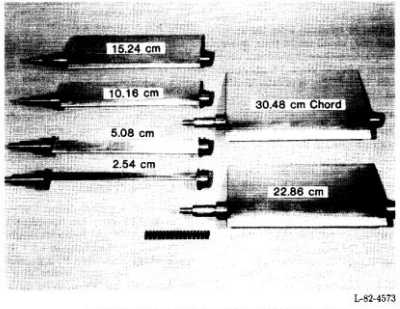
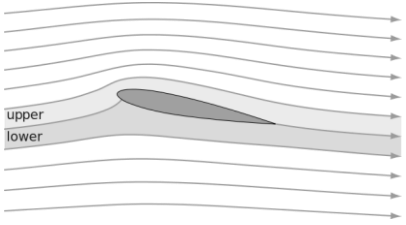
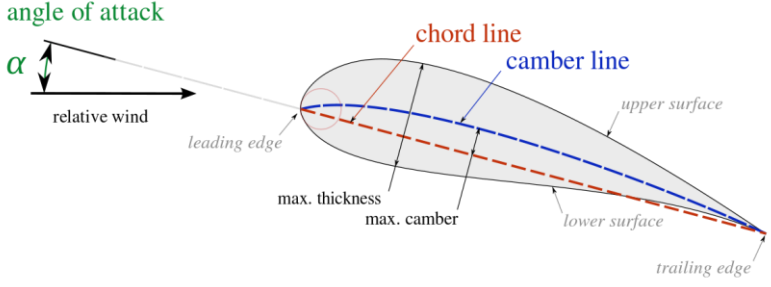


Figure 2. Two-dimensional NACA 0012 airfoil blade models.





#1 Understanding the data

- Check the data source: understand what the data refers to
- Objective: understand the characteristics of the data
- Look at the feature columns:
  - Any missing values?
  - Any features with NaN values?
  - Uniqueness of the dataset? (“cardinality”)

=> Colab

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1503 entries, 0 to 1502
Data columns (total 6 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   frequency               1503 non-null   int64
 1   angle_attack            1503 non-null   float64
 2   chord_length            1503 non-null   float64
 3   Free-stream_velocity    1503 non-null   float64
 4   displacement_thickness  1503 non-null   float64
 5   sound_pressure          1503 non-null   float64
dtypes: float64(5), int64(1)
memory usage: 70.6 KB
```

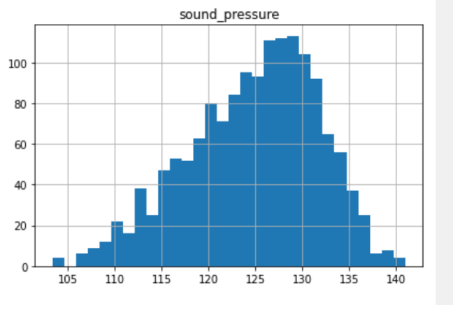
```
data.head(5)
```

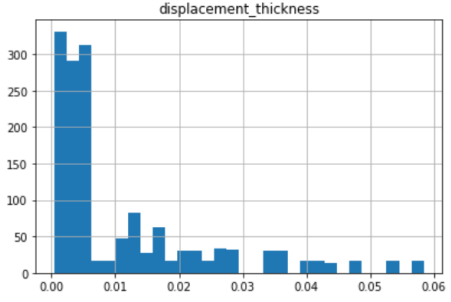
	frequency	angle_attack	chord_length	Free-stream_velocity	displacement_thickness
0	800	0.0	0.3048	71.3	0.002663
1	1000	0.0	0.3048	71.3	0.002663
2	1250	0.0	0.3048	71.3	0.002663
3	1600	0.0	0.3048	71.3	0.002663
4	2000	0.0	0.3048	71.3	0.002663

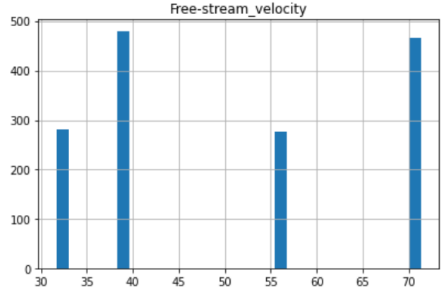
#2 Exploring the data

- **Objective**: generate a data quality report

- Using standard statistical measures of central tendency and variation
  - tabular data and visual plots
  - mean, mode, and median
  - standard deviation and percentiles
  - bars, histograms, box and violin plots







✓ Missing values,
✓ Irregular cardinality problems,
  - 1 or comparably small
✓ Outliers
  - invalid outliers and valid outliers

#2 Exporing the data: Correlation Matrix

- Shows the correlation between each pair of features

$$Cov(a,b) = \frac{1}{n-1} \sum_{i=1}^{n} \left[(a_i - \bar{a}) \times (b_i - \bar{b}) \right]$$

Features; instance; mean; mean

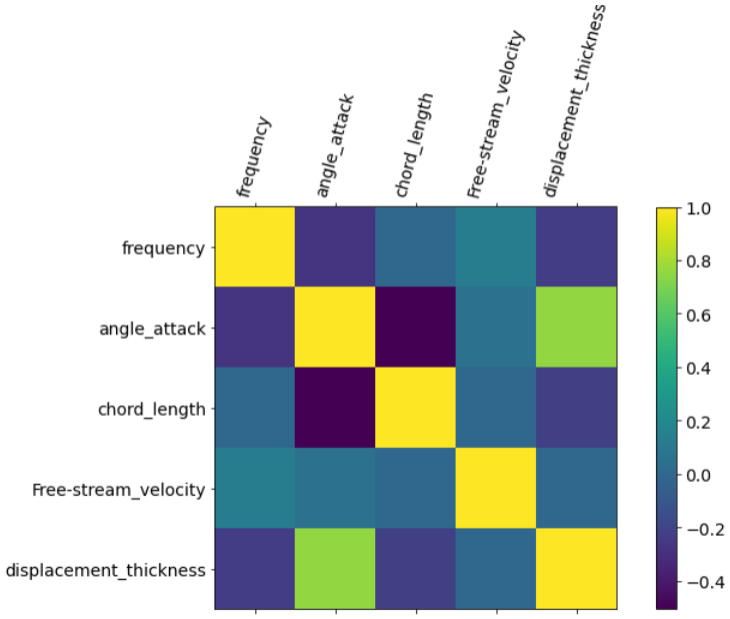
- Normalized form of “covariance”

$$Corr(a,b) = \frac{Cov(a,b)}{SD(a) \times SD(b)}$$

* Normalized
* Dimensionless

Easy to interpret

- Ranges between −1 and +1

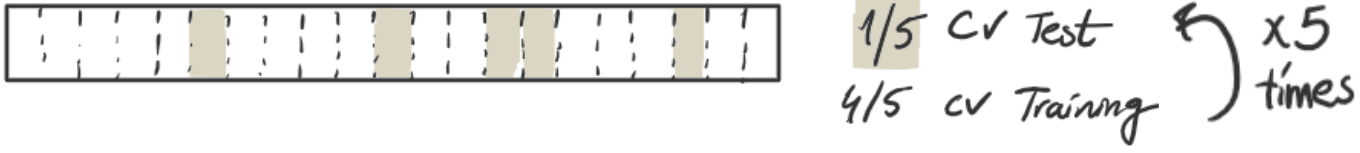


#2 Preparing the Data

- Classification >> supervised >> **training & test split**

- Reducing overfitting via **cross-validation**: take **random portions** of the data to build a model

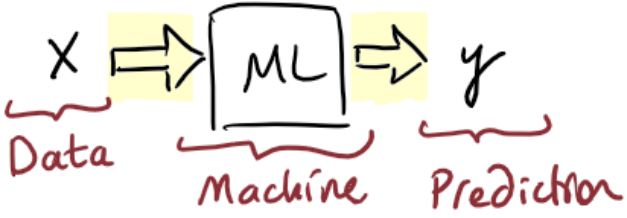
- **k-fold** method: k = 5; (typically 10)



#3 Model Selection:

* Machine learning := Optimization
* Optimization := Predict + Compare + Learn

Prediction:

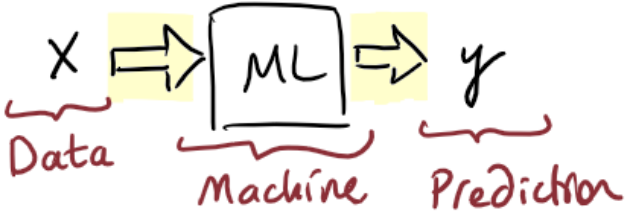


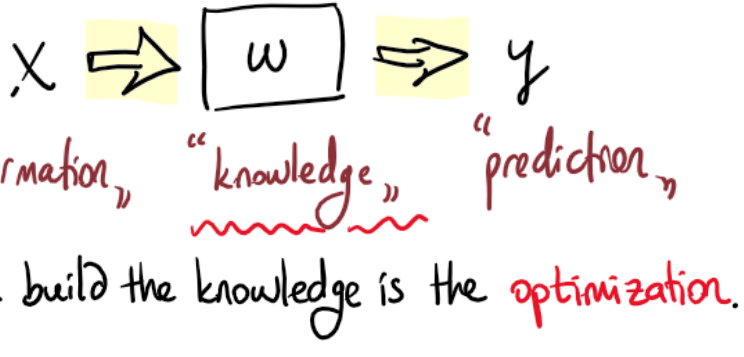
* ML ⇒ Combines multiple input into a prediction
  [AND, OR, NOT]
* Math := Weighted Sum // Dot Product ⇒ $[y = w \cdot x]$
  „we make predictions via weights (w)"

#3 Model Selection:

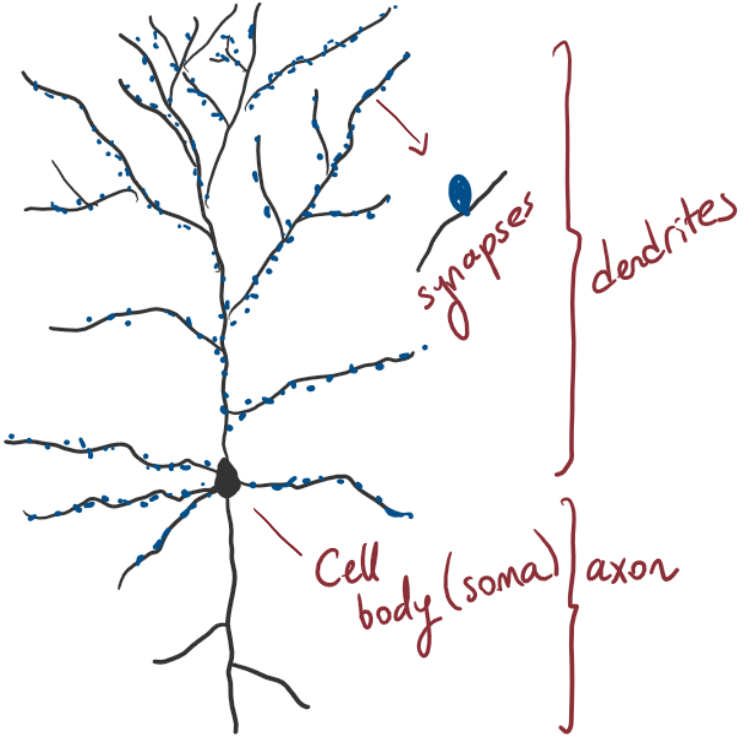
* Machine learning := Optimization
* Optimization := Predict + Compare + Learn

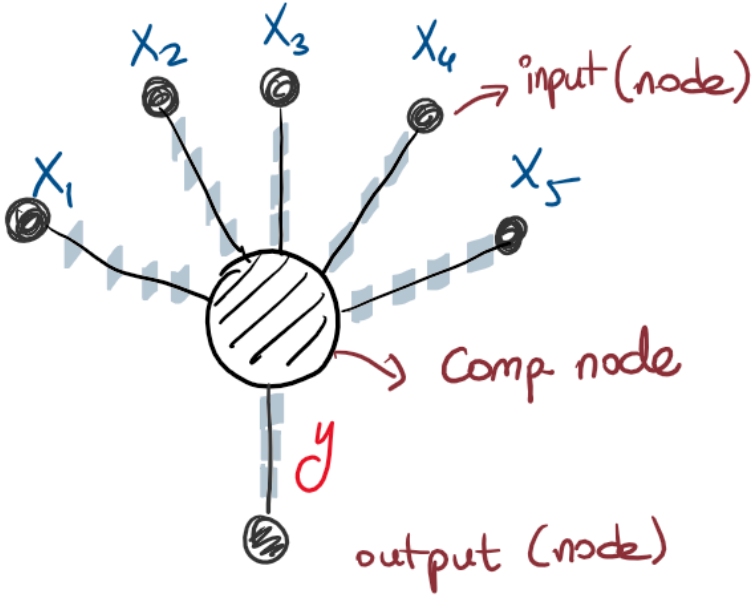
Prediction: "we make predictions via weights (w)"





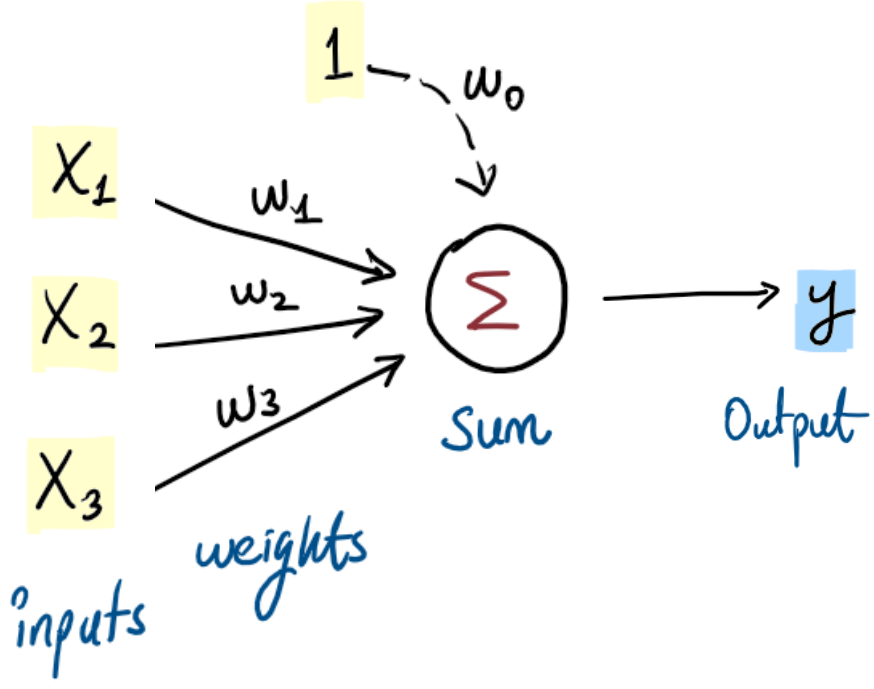
Neural Networks





Neural Networks

Simplest Case: The Perceptron

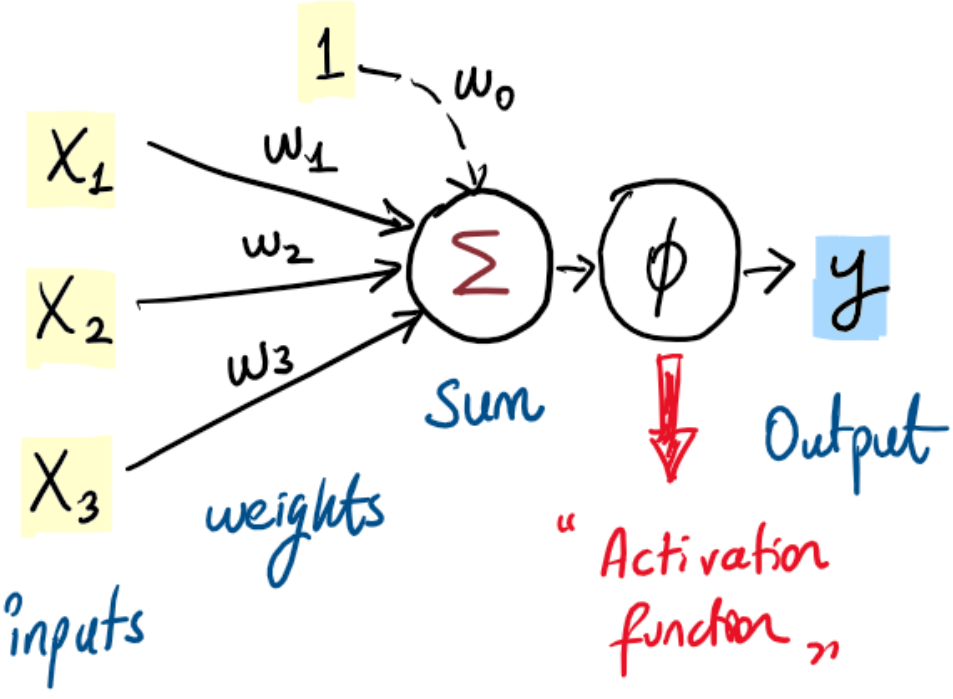


Eq:

$$y = w_0 + \sum_{i=1}^{n} x_i w_i$$

bias

Simplest Case: The Perceptron



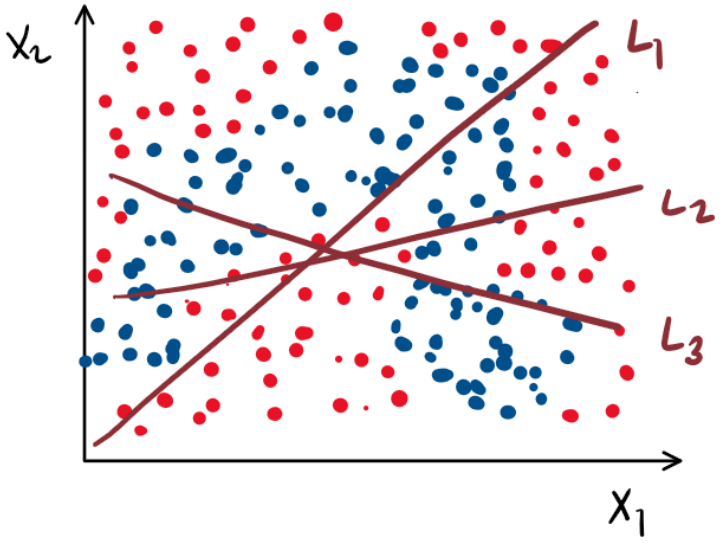
Eg: Binary Classifier.

$$\phi = \begin{cases} 1, & \text{if } w_0 + \sum_{i=1}^{M} x_i w_i > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\phi = f(w, x)$$

Why do we need activation functions ?

* Introduce nonlinearity (remember Kernel trick)



* Linear decision boundaries ($y = w \cdot x$) won't work !
* if we stack neurons ?

i) $y^{(1)} = w^{(1)} \cdot x$

ii) $y^{(2)} = w^{(2)} \cdot y^{(1)}$

iii) $y^{(n)} = w^{(n)} y^{(n-1)}$

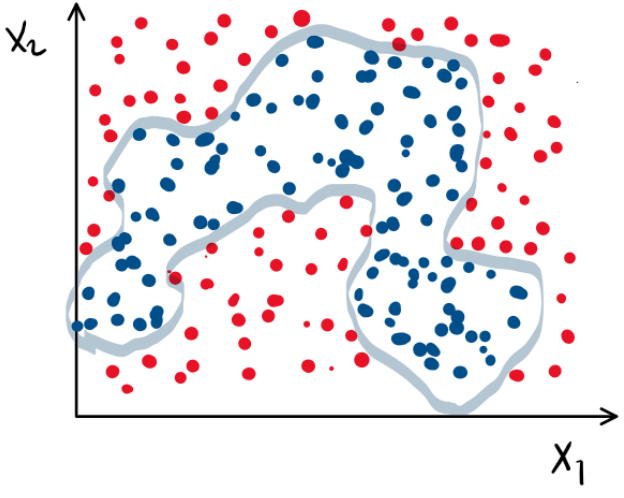
$$y = \underbrace{w^{(n)} \dots w^{(2)} w^{(1)}}_{w^*} \cdot x$$

$$y = w^* \cdot x$$

Just another line

Why do we need activation functions ?

$$y = \phi_n\left(w^{(n)}, \left(\ldots, \phi_2\left(w^{(2)}, \phi_1\left(w^{(1)}, x\right)\right)\ldots\right)\right)$$



⇩

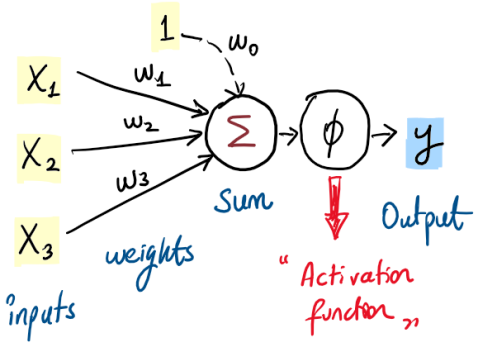
highly underdetermined system

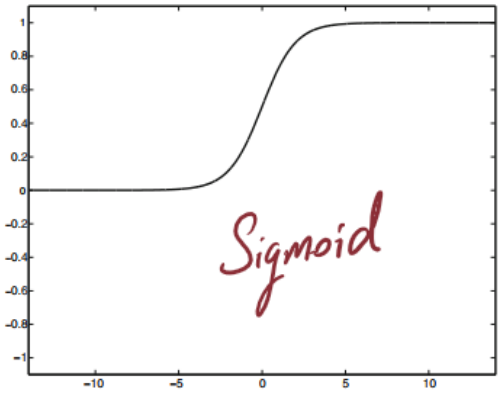
⇦ Requires constraints to obtain unique solutions !

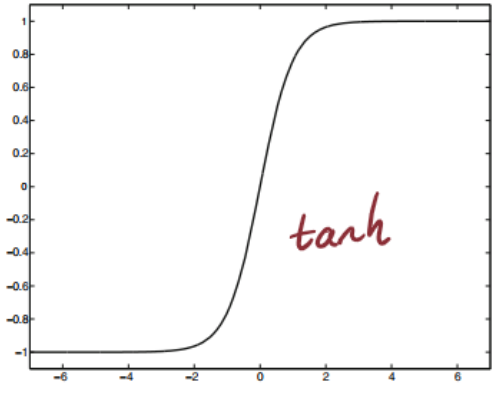
Why do we need activation functions ?

Common Activation Functions

- Linear ⇒ $f(x) = x$
- Sigmoid ⇒ $f(x) = 1/[1 + \exp(-x)]$
- tanh ⇒ $f(x) = \tanh(x)$



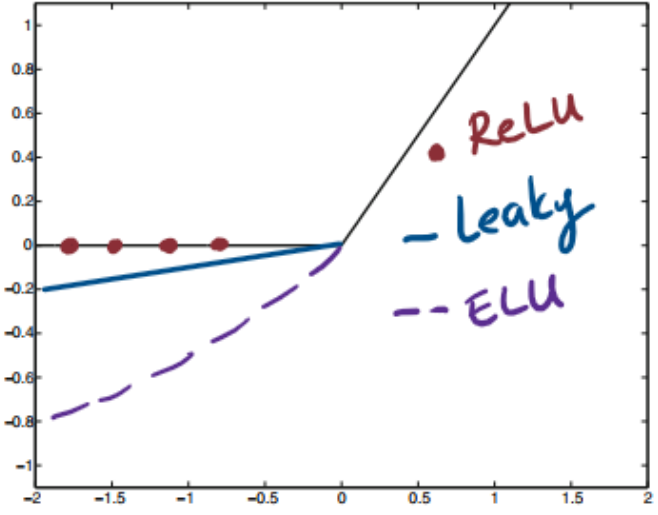




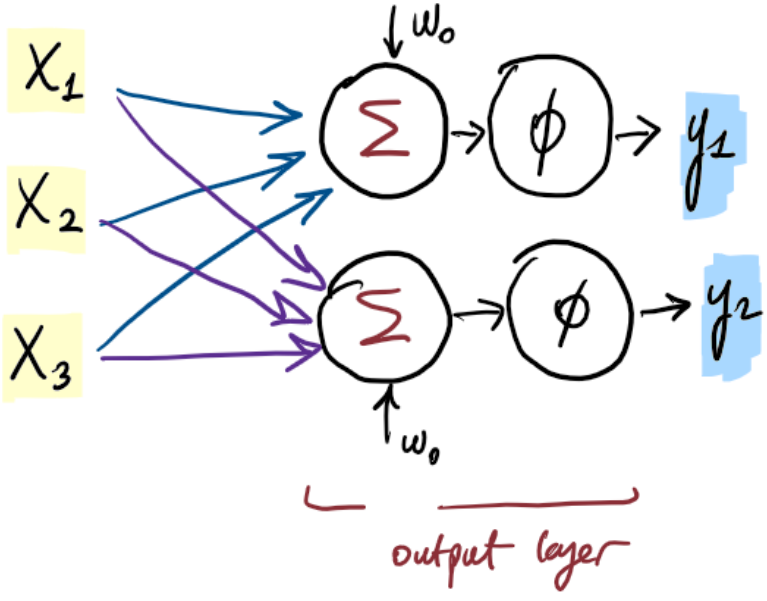
Why do we need activation functions?

Common Activation Functions

- Rectified Linear Unit (ReLU) $\Rightarrow f(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases} \Rightarrow \underbrace{\alpha \cdot x, \; x \leq 0}_{\text{"Leaky"}}$
- ELU $\Rightarrow f(x) = \begin{cases} x, & x > 0 \\ a(\exp(x) - 1), & x \leq 0 \end{cases}$
- SELU $\Rightarrow f(x) = \begin{cases} \beta \cdot x, & x > 0 \\ \alpha\beta(\exp(x) - 1), & x \leq 0 \end{cases}$



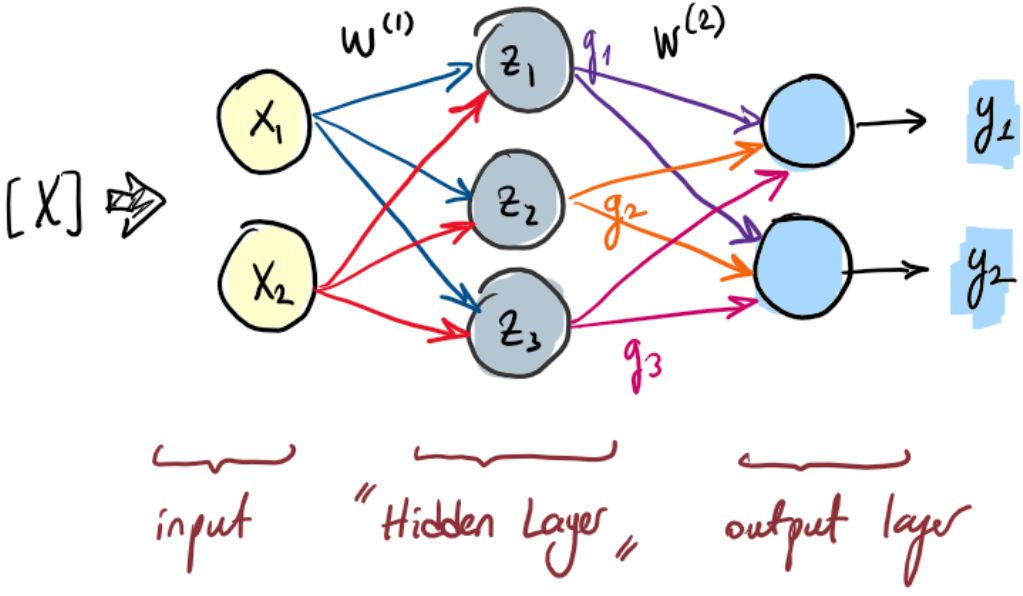
Multi-output Perceptron



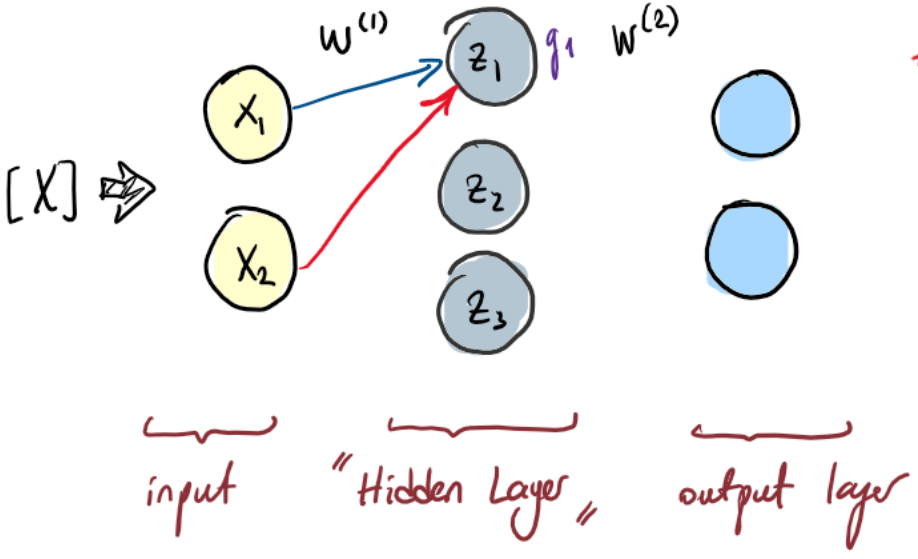
$$\Rightarrow y_1 = \phi\left[w_{0_1} + \sum_{j=1}^{n} x_j w_{j_1}\right]$$

$$\Rightarrow y_2 = \phi\left[w_{0_2} + \sum_{j=1}^{n} x_j w_{j_2}\right]$$

Multi-layer Perceptron:



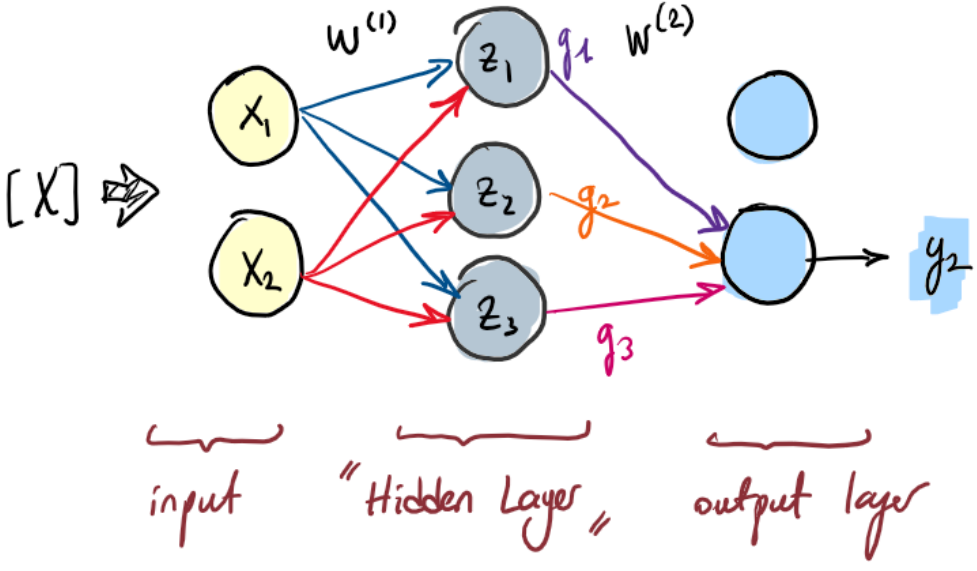
Multi-layer Perceptron:



Eq $$z_1 = w_{0_1}^{(1)} + \sum_{j=1}^{2} w_j^{(1)} x_j \qquad (1)$$

$$g_1 = \phi(z_1) \qquad (2)$$

Multi-layer Perceptron:



Eq.

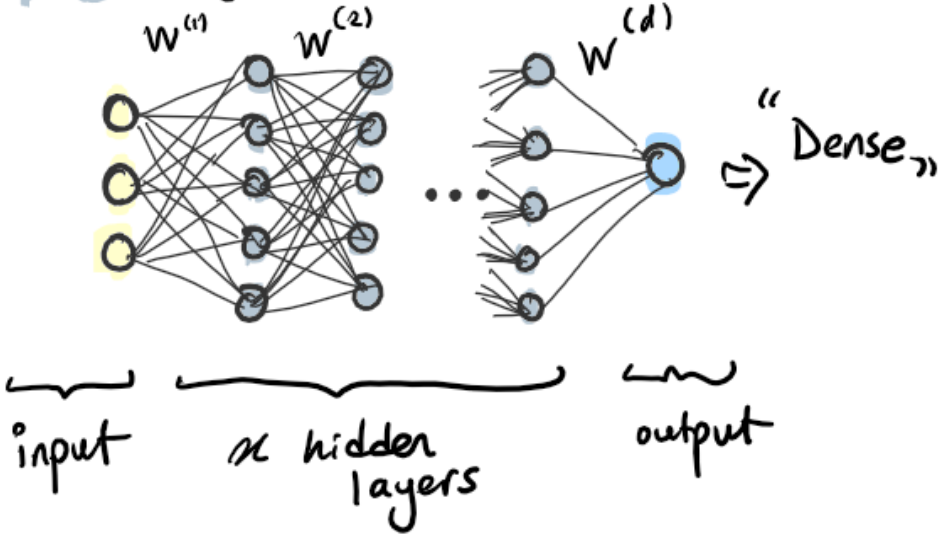
$$z_1 = w_{0_1}^{(1)} + \sum_{j=1}^{2} w_j^{(1)} x_j \qquad (1)$$

$$g_1 = \phi(z_1) \qquad (2)$$

$$y_2 = \phi\left[w_{0_2}^{(2)} + \sum_{j=1}^{3} g_j w_{j_2}^{(2)} \right] \qquad (3)$$

Multi-layer Perceptron:

Multi-layer:



⇒ "Dense"

* How can we find the optimum $W^{(i)}$?

⇓

"Learning Step"

Ateliers & Saveurs in Montreal

colab

Comparison: Choice of Loss Function

* Machine learning := Optimization := Predict + Compare + Learn

* $y_T := y_P$ + Error } How different is y_T from y_P
* How to compare y_T // y_P ?
* How to measure error ?

} Nature of the problem

Comparison: Choice of Loss Functions

* Machine learning := Optimization := Predict + Compare + Learn

- Regression → MAE, MSE, R^2, ...
- Classification → probabilistic → Cross entropy
  - → Relative entropy
- Clustering → Purity Score, ...
- Dim. Reduction → Reconstruction error, Precision
- Anomaly Detection → Anomaly Score

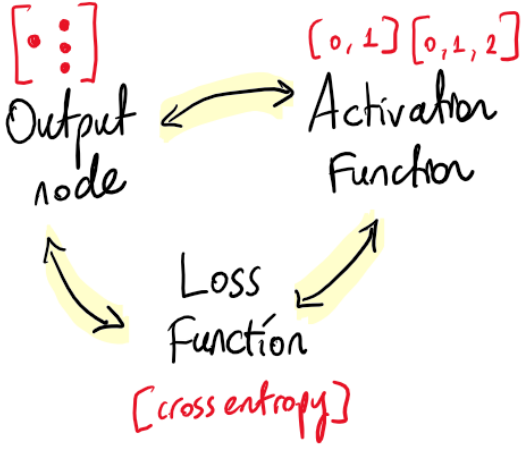
Eg: Impact of Error Interpretation

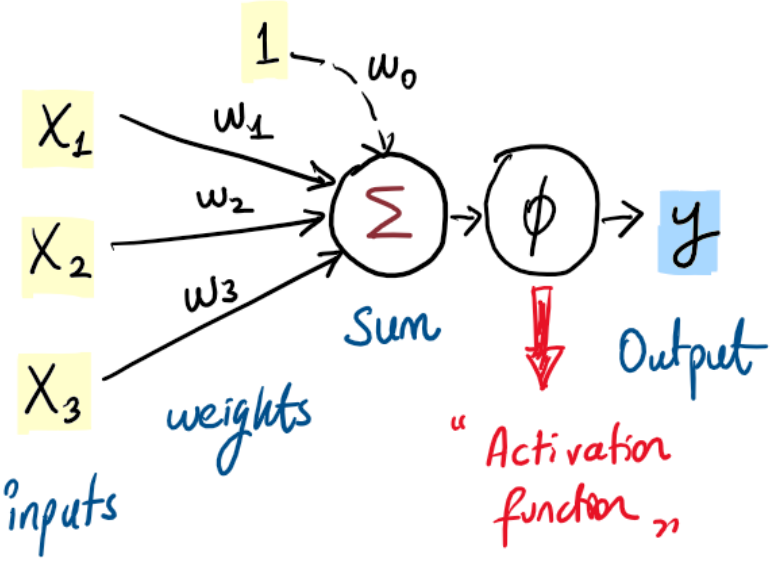
* $MSE = (y_T - y_P)^2 \propto \epsilon^2$: (2) 10 → (4) 100; 0.1 → 0.01
* $MAE = |y_T - y_P| \propto \epsilon$: (2) 10 → (2) 10; 0.1 → 0.1
* Which error sources are more important for me?

Comparison: Choice of Loss Functions

* Machine learning := Optimization := Predict + Compare + Learn

Impact on the Architecture:





Learning : Backpropagation

* Machine learning := Optimization := Predict + Compare + Learn

* How can we sets the weights so that NN makes accurate predictions ?

* Obj: Find W^* that gives the min. loss $J_{mn}(y_T - y_P)$

$W \leftarrow W^{(1)}, W^{(2)}, W^{(3)}, \ldots W^{(d)}$ (too many $w_j^{(i)}$)

Learning: Backpropagation

Karlsruhe Institute of Technology

Algorithm:

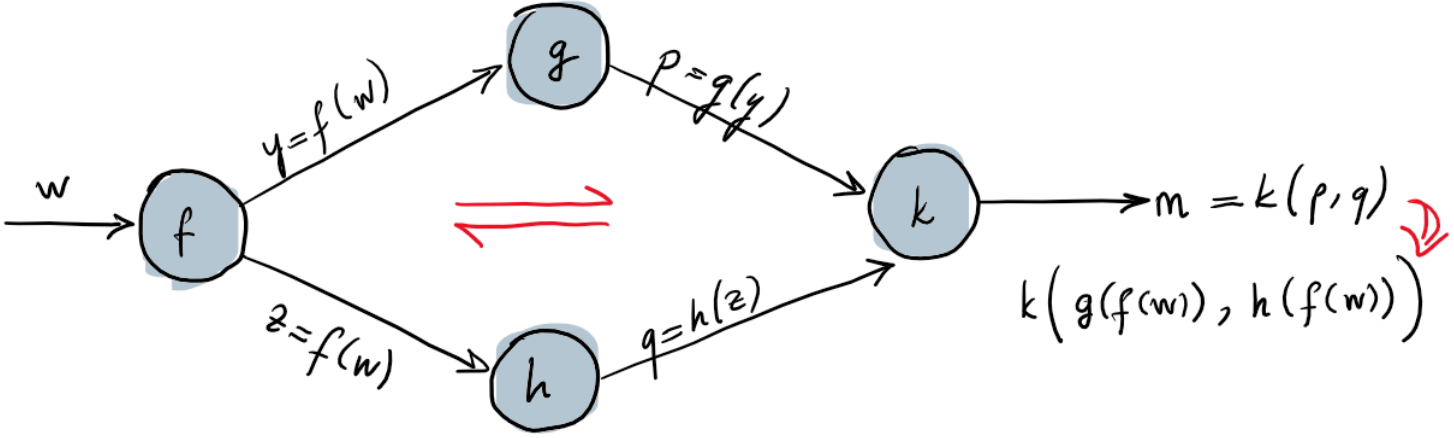
(i) Initialize the weights (randomly)

(ii) Calculate | y_p given W.
| local derivates at the nodes
| (influence of $w_{ij}^{(l)}$ on the information prop.)

(iii) Learn the gradient of loss function wrt. different weights. $\left(\frac{\partial J(w)}{\partial w} \right)$

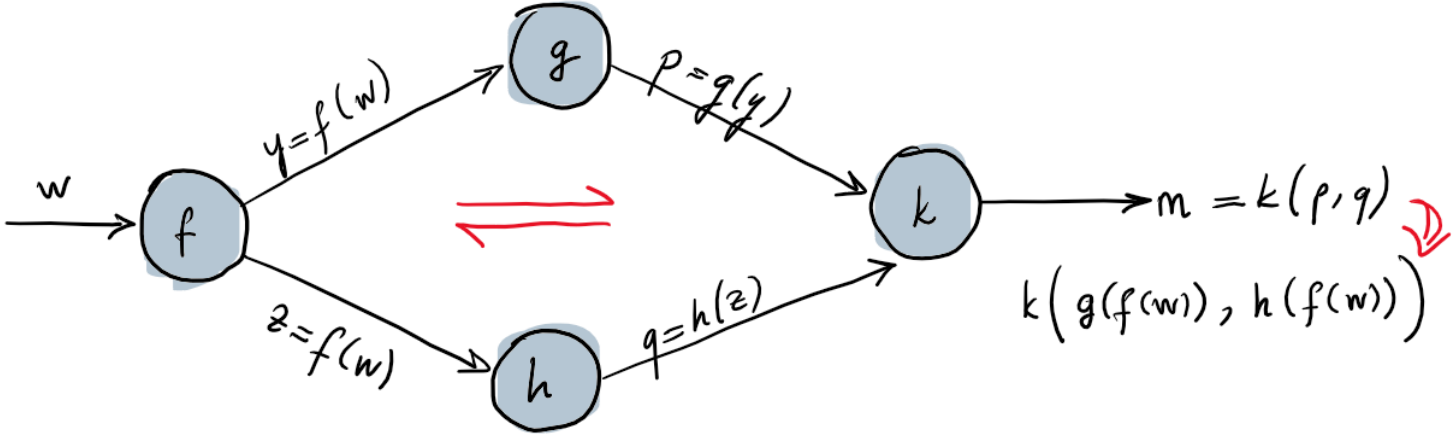
(iv) Update the weights (as a function of $\partial J / \partial w$)

Learning: Backpropagation



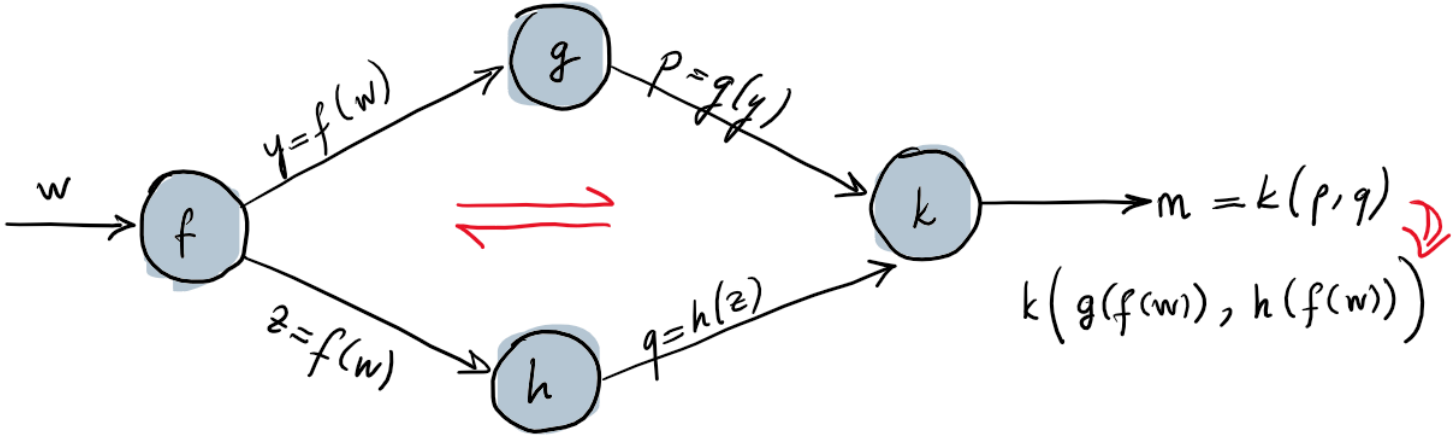
$$\textcircled{1} \quad \frac{\partial m}{\partial w} = \frac{\partial m}{\partial p}\frac{\partial p}{\partial w} + \frac{\partial m}{\partial q}\frac{\partial q}{\partial w} \quad \left[\text{multivariable CR}\right]$$

Learning : Backpropagation



(2) $$\frac{\partial m}{\partial w} = \frac{\partial m}{\partial p}\frac{\partial p}{\partial y}\frac{\partial y}{\partial w} + \frac{\partial m}{\partial q}\frac{\partial q}{\partial z}\frac{\partial z}{\partial w}$$ [Univariate CR]

Learning: Backpropagation



(3) $$\frac{\partial m}{\partial w} = \frac{\partial}{\partial p}\left(k(p,q)\right) g'(y)\, f'(w) + \frac{\partial}{\partial q}\left(k(p,q)\right) h'(z)\, f'(w)$$ [Activation func.]

path #1 path #2

Repeat this for every weight in the network !

Learning : Backpropagation

(4) Error $= E = \frac{(m_t - m)^2}{2} \Rightarrow \frac{\partial E}{\partial w} = -(m_t - m)\frac{\partial m}{\partial w}$

(m_t: True Value, m: Pred.)

Error $= E = (m_t - m) \Rightarrow \frac{\partial E}{\partial w} = -\frac{\partial m}{\partial w}$

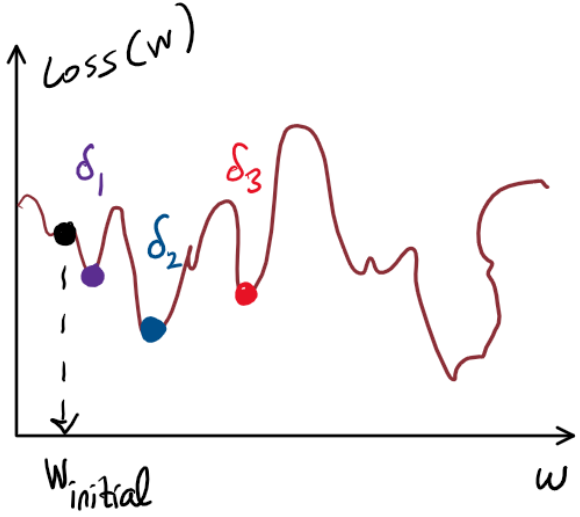
(5) Gradient Decent; $w_{t+1} = w_t + \delta \frac{\partial E}{\partial w_t}$ (δ: Learning Rate)

Ateliers & Saveurs in Montreal

colab

GD Algorithms: Improvements

* $w_{t+1} = w_t + \delta \frac{\partial E}{\partial w_t} \Rightarrow$ Learning rate is a hyperparameter



Option 1

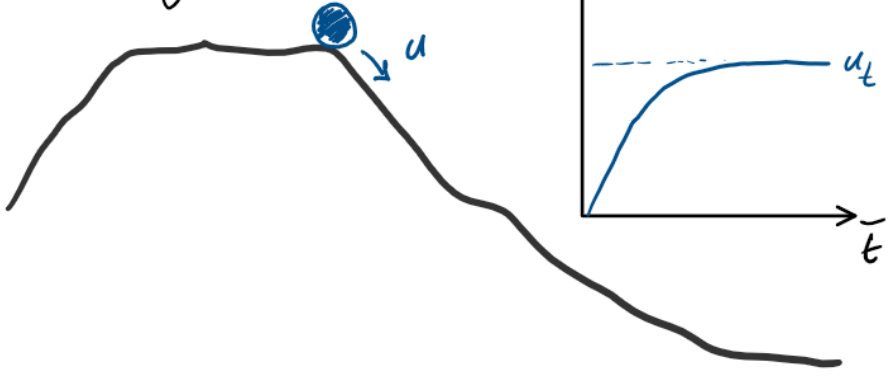
* Try different δ → Grid Search
  → Random Search
  → ...

Option 2

* Use an adaptive method ⇒ " $\frac{\partial E}{\partial w_t}$ "

GD Algorithms: Improvements

* GD with momentum optimizer :
* Regular GD ⇒ regular steps down the slope.
* "Building momentum up;"



accel.

$$\text{momentum} = \mu \cdot \text{momentum} + \delta \nabla \epsilon$$

$$w_{t+1} = w_t + \text{momentum}$$

$\mu \rightarrow [0, 1]$

high friction — no friction

GD Algorithms: Improvements

* **Nesterov Accelerated Gradient (NAG)**
  - $momentum = \mu \cdot momentum + \delta \nabla E(w_t + \mu \cdot momentum)$
  - $w_{t+1} = w_t + momentum$

  get "future" acc.

  "Decay rate"

  $\ell \Rightarrow 0.9 \quad (\ell - 1) \Rightarrow 0.1$

* **RMSProp:** accumulates the most recent iterations
  - "Divide the gradient by a running average of its recent magnitude.
  - $MS_t = 0.9\, MS_{t-1} + 0.1\, \nabla E^2$
  - $w_{t+1} = w_t + \delta \nabla E / \sqrt{MS_t + \epsilon}$

* **ADAM: Adaptive Moment** $\approx$ Momentum + RMSProp

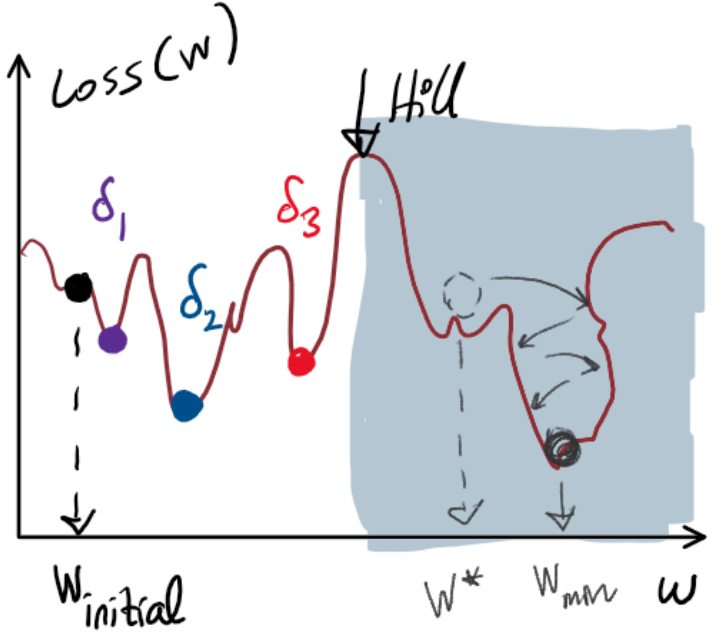
Ateliers & Saveurs in Montreal

colab

GD Algorithms: Initialization

* $w_{t+1} = w_t + \delta \frac{\partial E}{\partial w_t}$

- δ (basic parameter)
- $\frac{\partial E}{\partial w_t} \Rightarrow$ adaptive schemes
- $w_t \Rightarrow$ Smart initialization



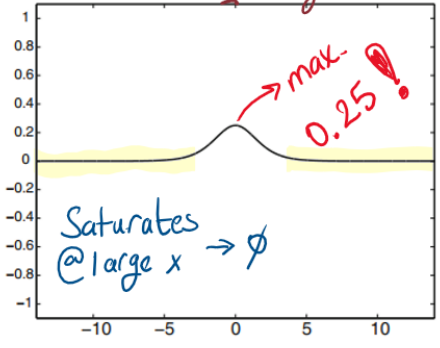
GD Algorithms: Initialization

$$w_{t+1} = w_t + \delta \frac{\partial E}{\partial w_t}$$

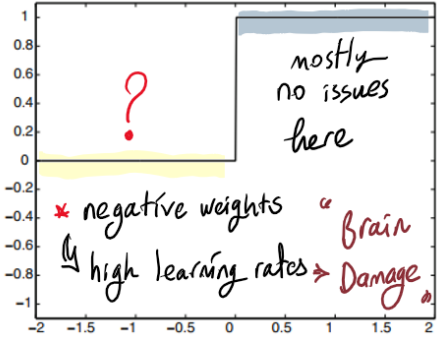
↓ vanishing grad. ↑↑ Exploding grad.

(–) Activation functions
(–) initialization

Derivative of Sigmoid



Derivative of ReLU



Solution:

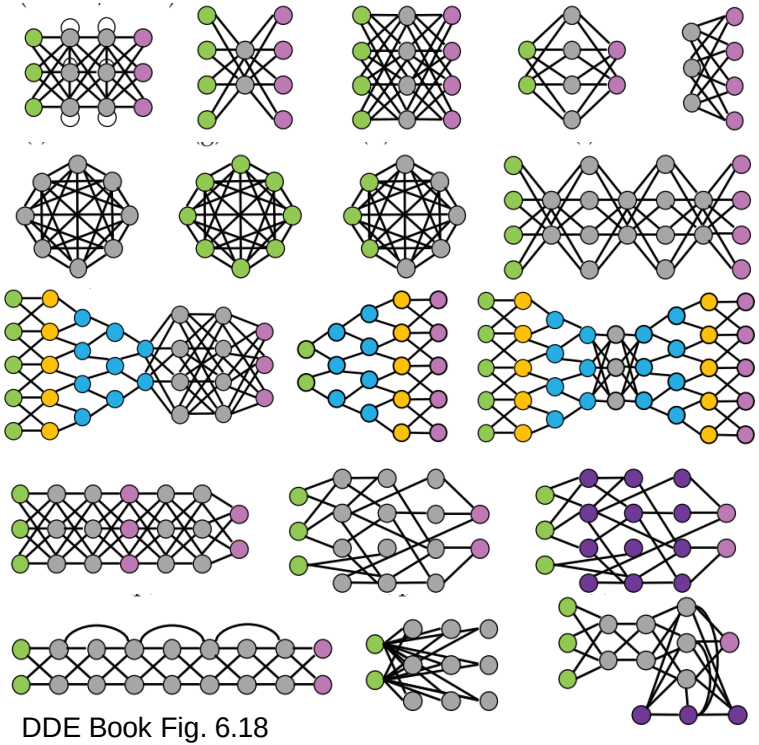
leaky ReLU
Small LR

colab

Regularization

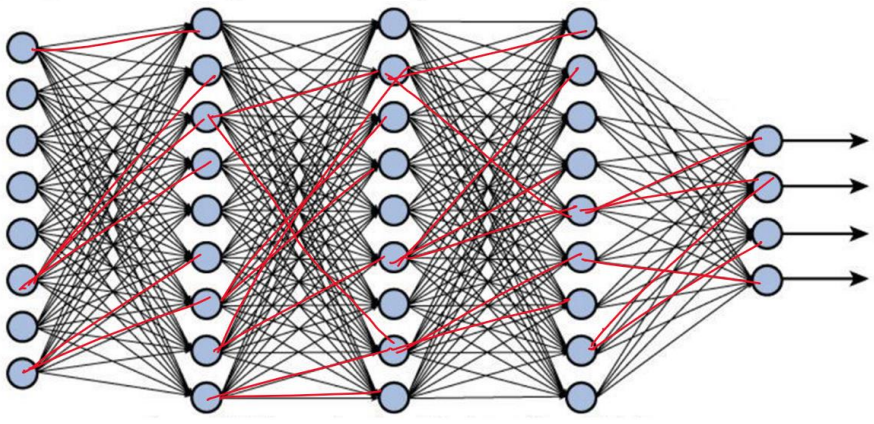
* Architecture (#layers, #neurons, connections)
* Dropout
* Early stopping
* penalty based (l_1, l_2)

Architecture Regulation:



DDE Book Fig. 6.18

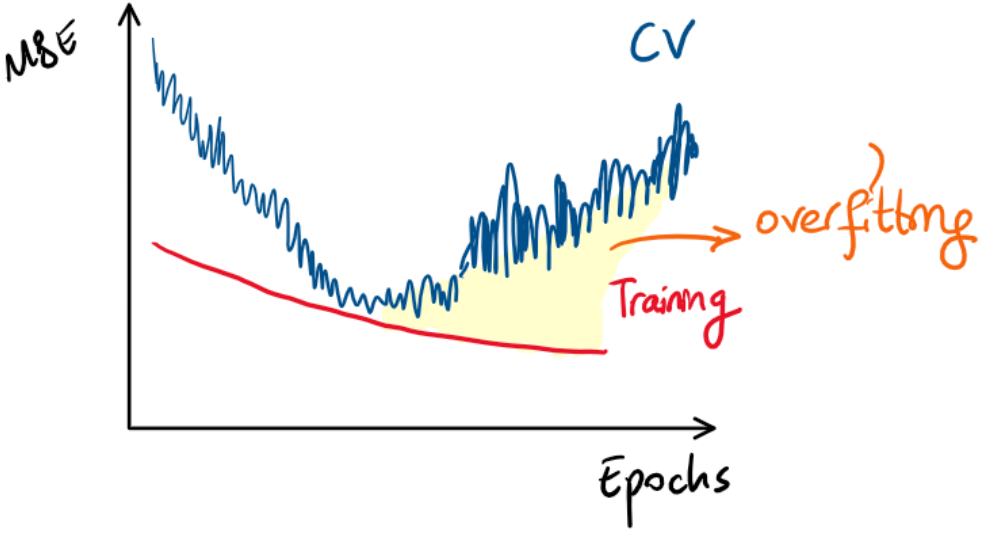
- # layers
- # neurons / layer
- Connections
- Dropouts (~20 – 50 %)



Ateliers & Saveurs in Montreal

colab

Early Stopping:



* As training (epochs) continues, model would overfit the training data.

* A natural solution:
  - ✓ Keep track of best CV scores
  - ✓ Stop if there is no more improvement.

Tensor Flow:

* callbacks.ModelCheckpoint
* callbacks.EarlyStopping

Ateliers & Saveurs in Montreal

colab

Penalty-based Regularization

Karlsruhe Institute of Technology

* The common approach in ML Algorithms
* $E_{total} = E_{data} + E_{Reg}$
* Obj: Force the weights take smaller values
* TF → Keras → l_1, l_2, l_1-l_2 Reg.

Example:

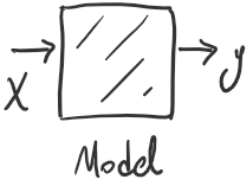
* $Loss = \sum (y_t - y_p)^2 + \lambda \sum w_i^2$ (hyperparameter)
* $w_{t+1} = w_t + \delta \frac{\partial E}{\partial w_t}$
* $w_{t+1} = w_t (1 - \delta\lambda)$ Forgetting mechanism

Ateliers & Saveurs in Montreal

colab

Additional Notes

Model Selection: Error Norms



(2) $y_{True_i} = y_{p_i} + Error_i$ } Error Metric (norm) := Goodness of a fit

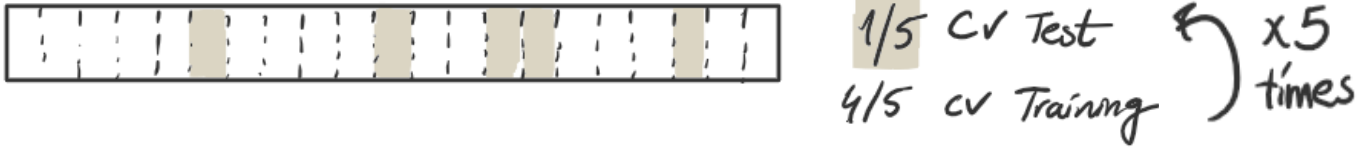
- Maximum error (ℓ_∞) $\max_{i \in [1,n]} |y_{true_i} - y_{p_i}|$
- Mean absolute error (ℓ_1) $1/n \sum_{i=1}^{n} |y_{true_i} - y_{p_i}|$
- Least Squares error (ℓ_2) $\left(1/n \sum_{i=1}^{n} |y_{true_i} - y_{p_i}|^2\right)^{1/2}$

#4 Training the model

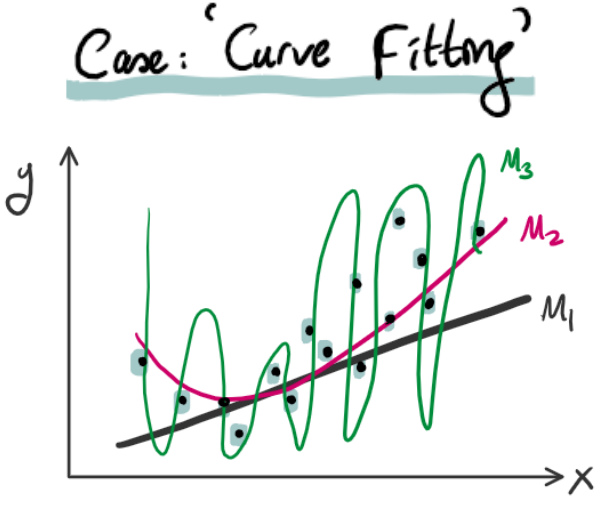
- Classification >> supervised >> **training & test split**

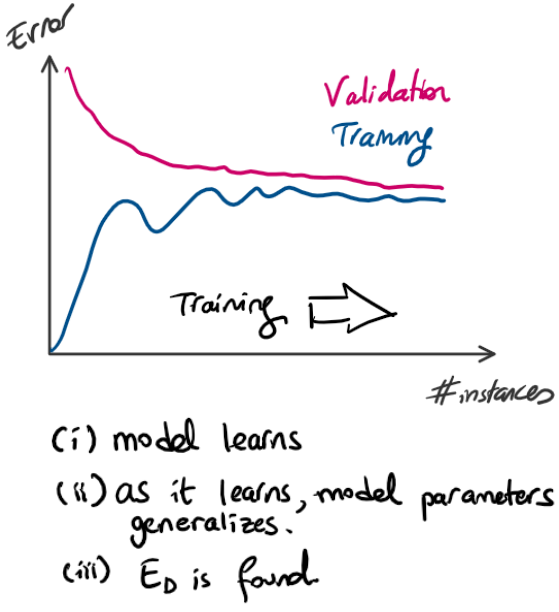
- Reducing overfitting via **cross-validation**: take **random portions** of the data to build a model

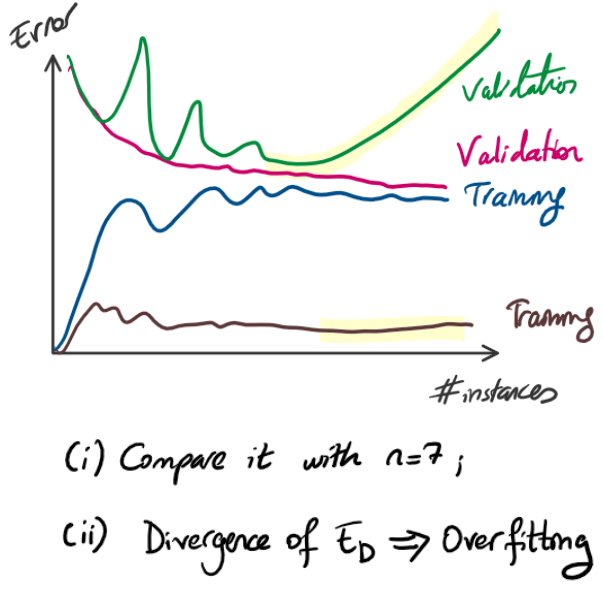
- **k-fold** method: k = 5; (typically 10)



#5 Evaluation of the results: Learning Curves







Last Layer Configuration Examples

Problem type	Last-layer activation	Loss function
Binary classification	sigmoid	Binary cross-entropy
Multiclass, single-label classification	softmax	Categorical cross-entropy
Multiclass, multilabel classification	sigmoid	Binary cross-entropy
Regression to arbitrary values	None	MSE
Regression to values between 0 and 1	sigmoid	MSE or Binary cross-entropy