OBSERVATIONAL COSMOLOGY:

INTRODUCTION TO MACHINE LEARNING

Credit: A.Asperti (Unibo); F. Villaescusa (Simon Foundation)

A.A. 2023/2024 | Matteo Costanzi - University of Trieste / INAF

Why Machine Learning?

There are problems that are difficult to address with traditional programming techniques:

- classify a document according to some criteria (e.g. spam, sentiment analysis, ...)
- compute the probability that a credit card transaction is fraudulent
- recognize an object in some image (possibly from an unusual point of view, in new lighting conditions, in a cluttered scene)
- ...

Typically the result depends on a non-linear combination of a large number of parameters, each one contributing to the solution in a small degree

The Machine Learning approach:

Suppose to have a set of input-output pairs (training set):

$$\{\boldsymbol{x}, \boldsymbol{y}\}$$

the problem consists in understanding the map between **x** and **y**

The M.L. approach:

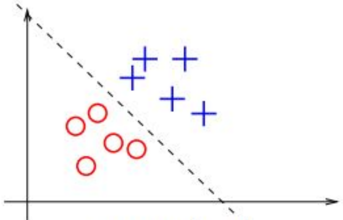
- describe the problem with a model depending on some parameters Θ (i.e. choose a parametric class of functions)
- define a loss function to compare the results of the model with the expected (experimental) values
- optimize (fit) the parameters Θ to reduce the loss to a minimum

The Machine Learning approach:

- Machine Learning problems are in fact optimization problems! So, why talking about learning?
- The point is that the solution to the optimization problem is not given in an analytical form (we don’t have a theoretical/analytical model to explain the data, and often there is no closed form solution).
- So, we use iterative techniques (typically, gradient descent) to progressively approximate the result.
- This form of iteration over data can be understood as a way of progressive learning of the objective function based on the experience of past observations.

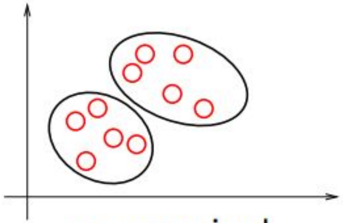
Different types of learning tasks

- **supervised learning**:
  inputs + outputs (labels)
  - classification
  - regression



supervised

- **unsupervised learning**:
  just inputs
  - clustering
  - component analysis
  - autoencoding



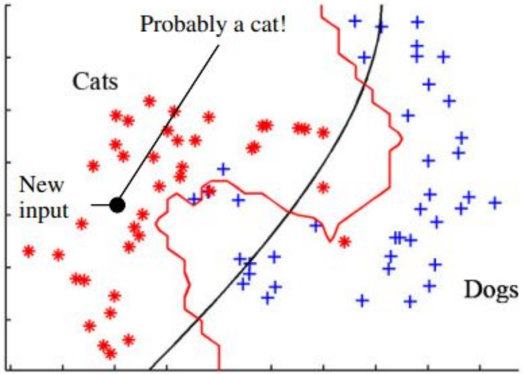
unsupervised

- **reinforcement learning**
  actions and rewards
  - learning long-term gains
  - planning

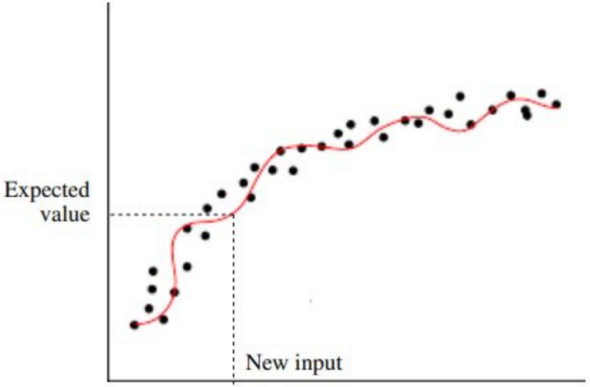
reinforcement

Classification vs. Regression

Two forms of supervised learning: $\{\langle x_i, y_i \rangle\}$



classification



regression

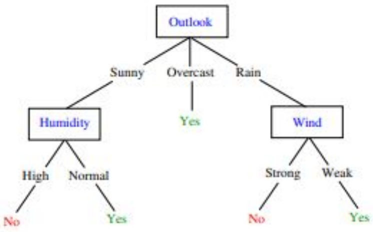
y is discete: $y \in \{\bullet, +\}$

y is (conceptually) continuous

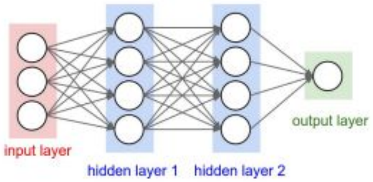
Many different techniques

- **Different ways to define the models**:
  - decision trees
  - linear models
  - neural networks
  - ...

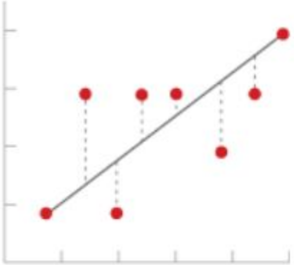
- **Different error (loss) functions**:
  - mean squared errors
  - logistic loss
  - cross entropy
  - cosine distance
  - maximum margin
  - ...



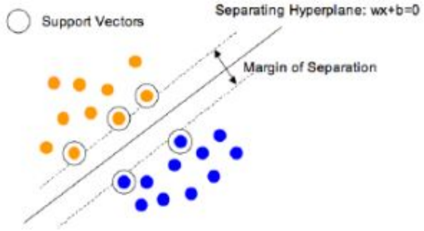
decision tree



neural net

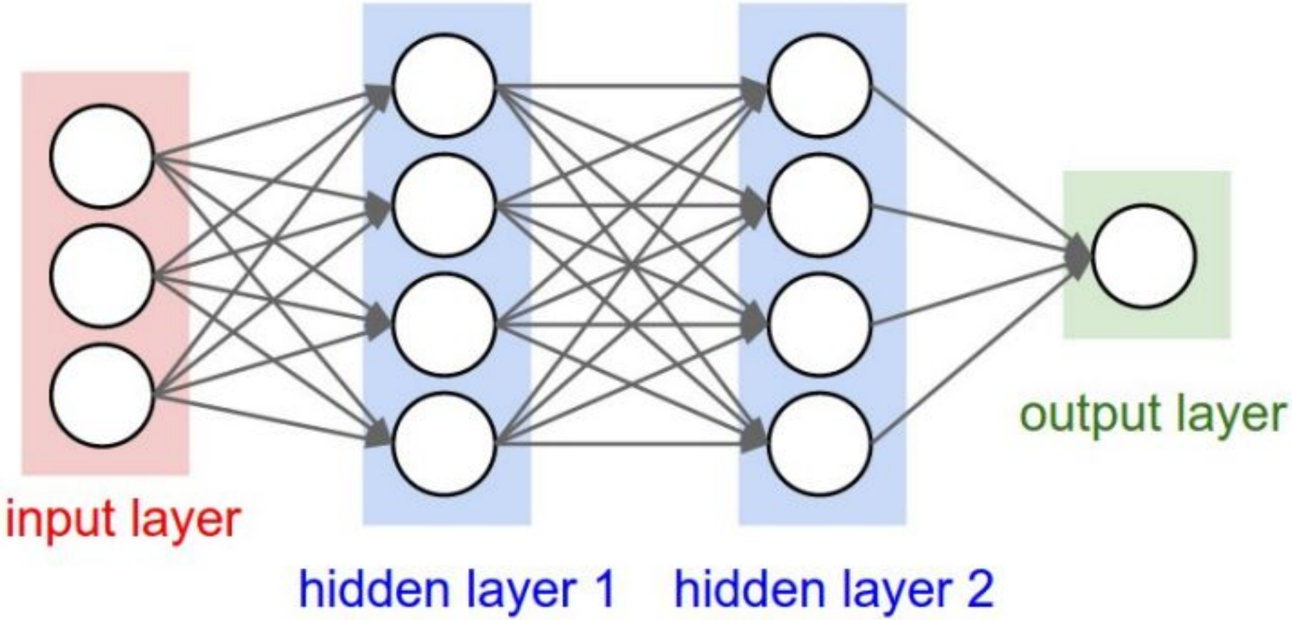


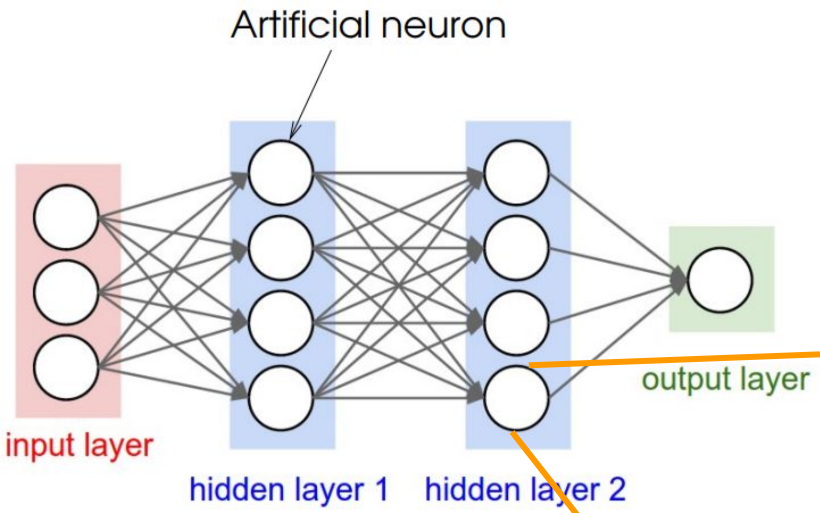
mean squared errors



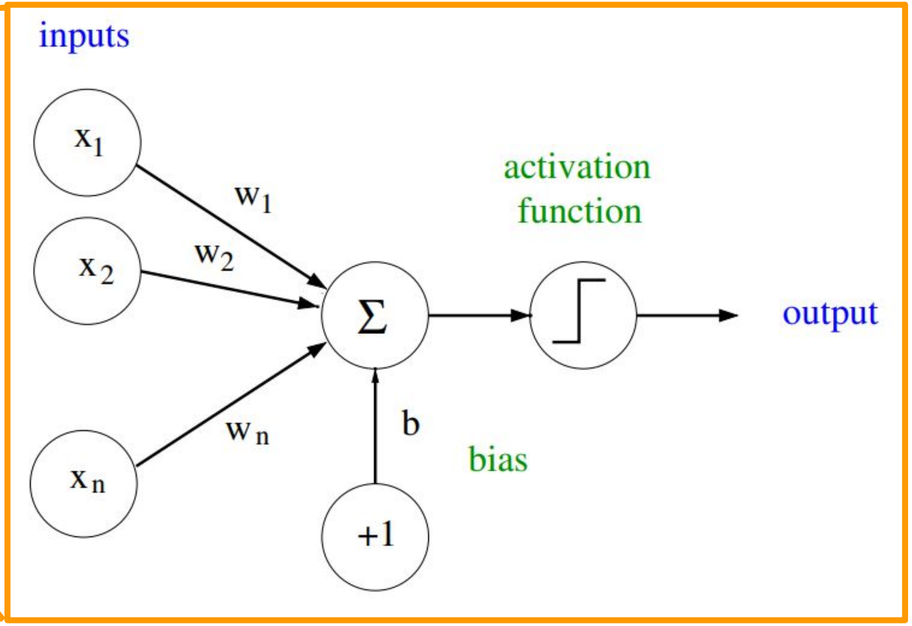
maximum margin

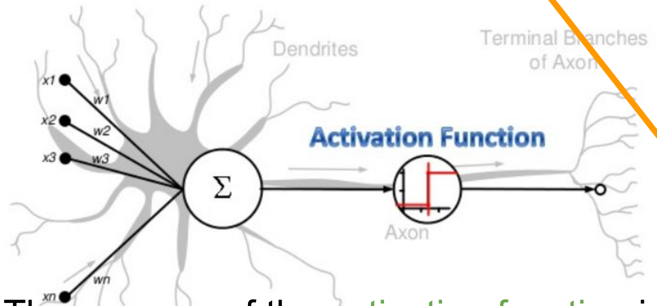
Neural Networks



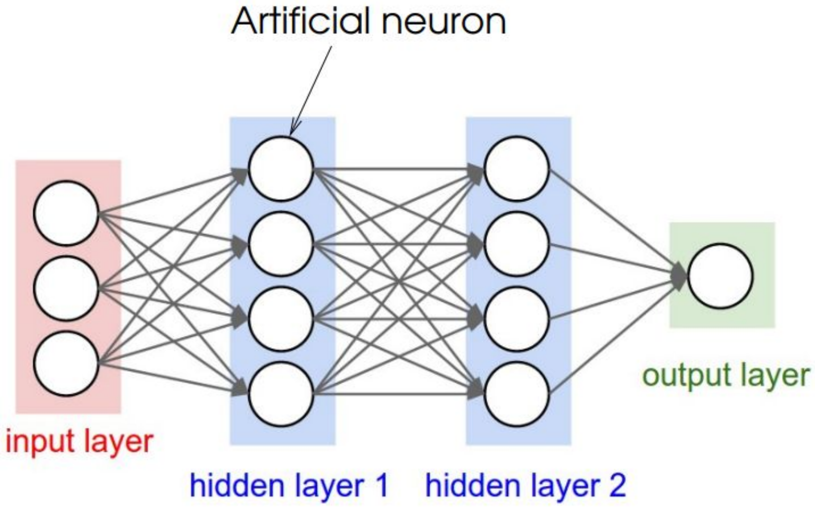


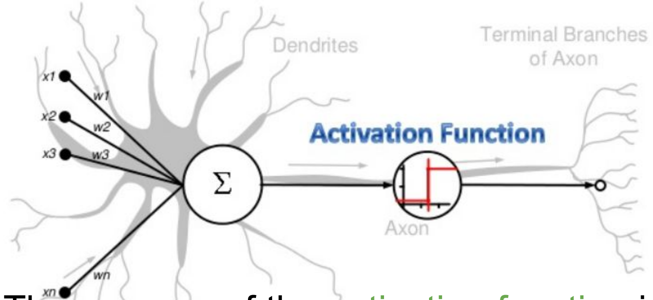
Each neuron takes multiple inputs and produces a single output (that can be passed as input to many other neurons):





The purpose of the activation function is to introduce a thresholding mechanism (similar to the axon-hillock of cortical neurons).





The purpose of the activation function is to introduce a thresholding mechanism (similar to the axon-hillock of cortical neurons).

NOTE: Composing linear transformations makes no sense, since we still get a linear transformation!

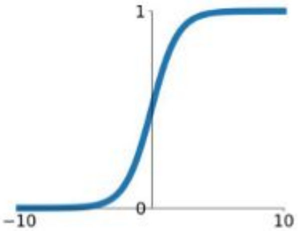
The activation function provides the source of NON LINEARTY in the neural networks

f(x)

Activation Functions

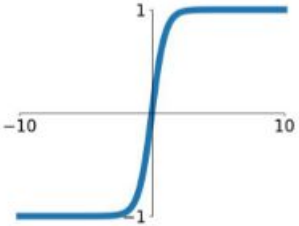
Sigmoid

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



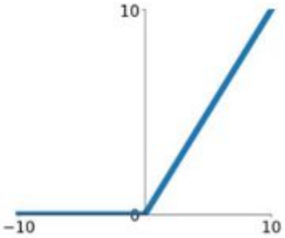
tanh

$\tanh(x)$



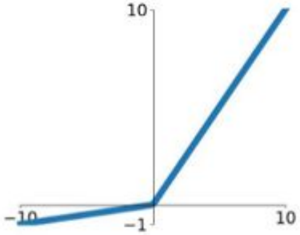
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

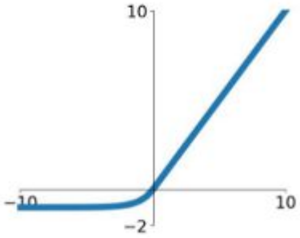


Maxout

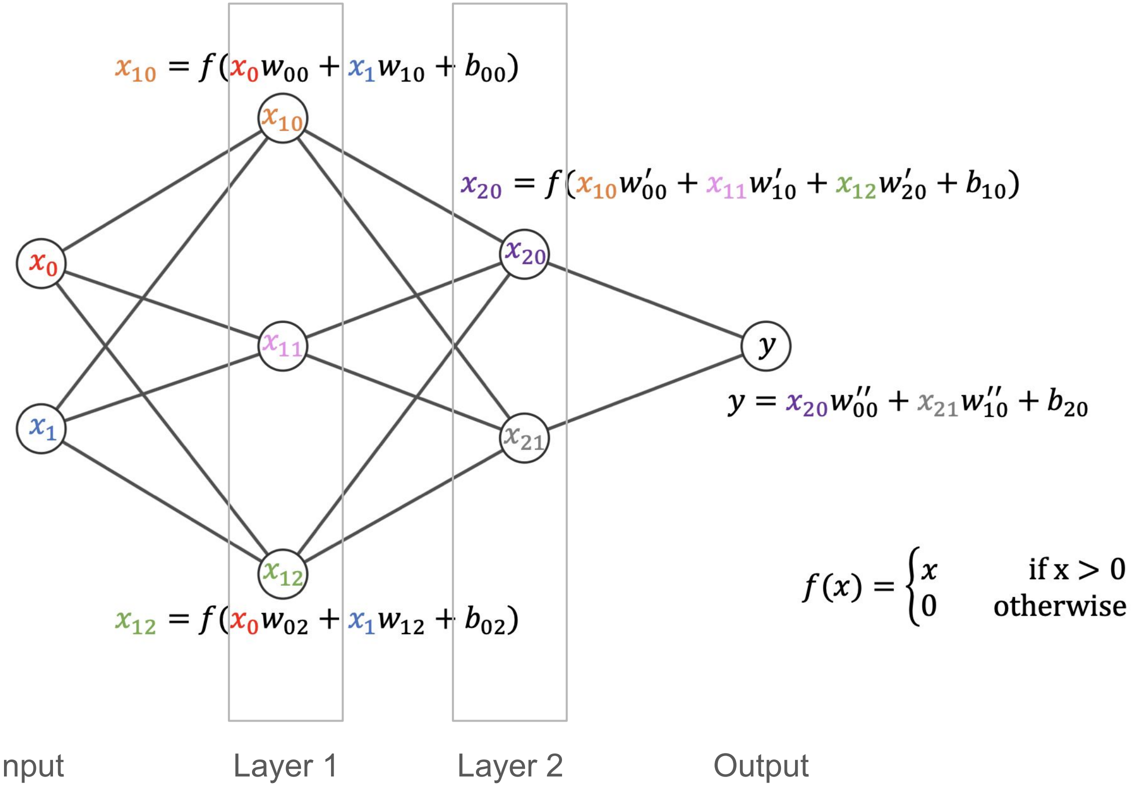
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Dense Feed-Forward NN

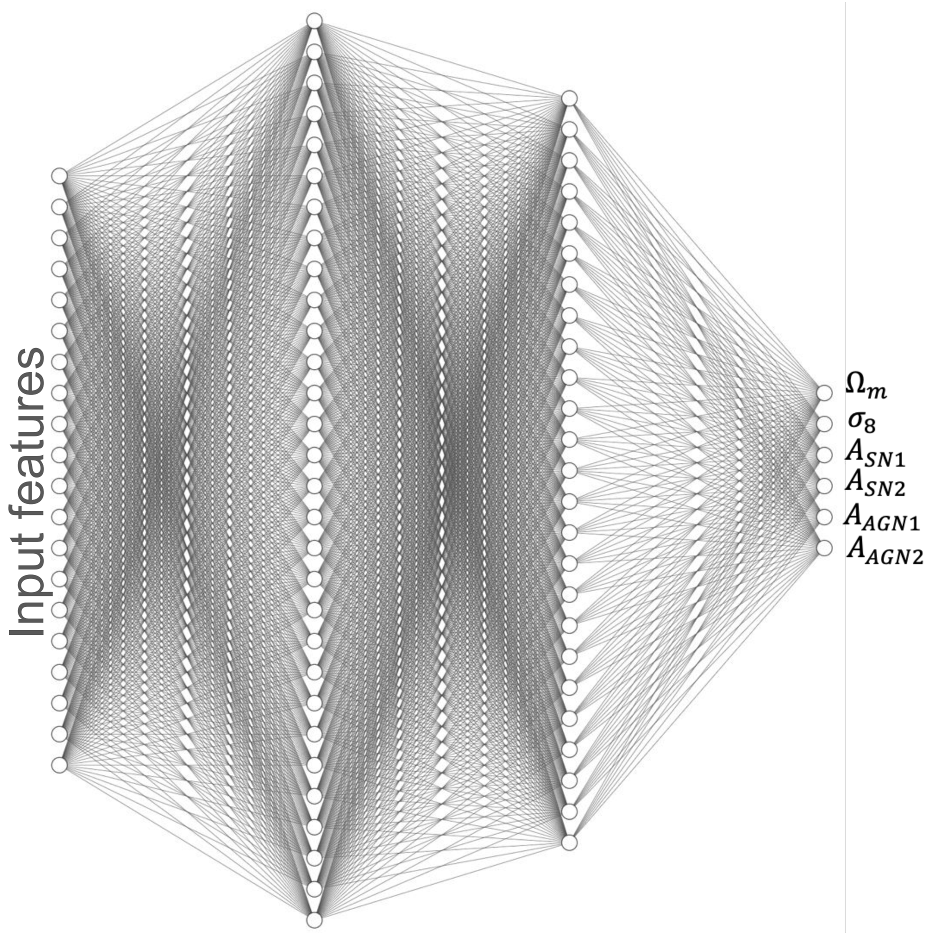


The most typical feed-forward network is a dense (i.e. w/ more than 1 hidden layer) network where each neuron at layer k − 1 is connected to each neuron at layer k.

The network is defined by a matrix of parameters (weights) w^k for each layer (+ biases). The matrix w^k has dimension $L_k \times L_{k+1}$ where L_k is the number of neurons at layer k.

The weights w^k and biases are the parameters of the model: they are learned during the training phase.

Training the NN



Goal: tune the value of the network parameters to get the most accurate predictions on the parameters.

Accuracy defined in the *loss function*

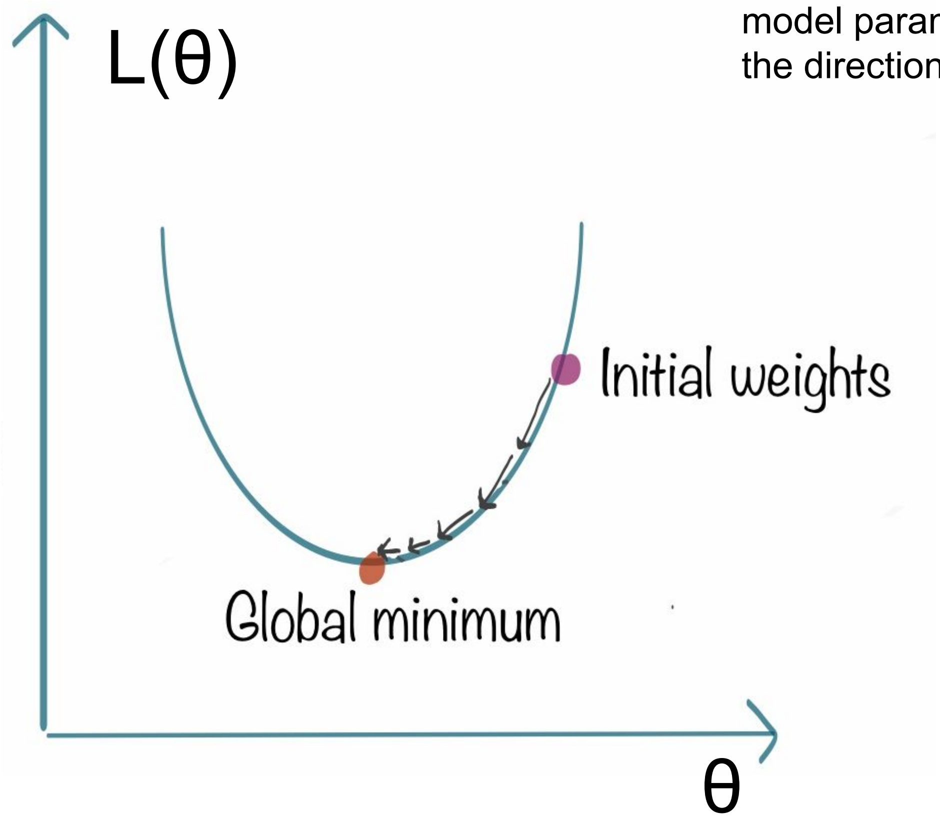
$$L = \frac{1}{N}\sum_{i=1}^{N}(\theta_{NN} - \theta_{True})^2$$

In other words we want to "learn" the parameters which minimize the loss function (optimization problem!)

Gradient Descent

The objective is to minimize the loss function over (fixed) training samples by suitably adjusting the parameters ϑ_i .

To do so we compute the gradient of the loss function w.r.t. the model parameters ϑ_i , $\nabla_\vartheta L$. The gradient is the vector pointing in the direction of steepest ascent.



We can reach a minimal configuration for L(ϑ) by iteratively taking small steps in the direction opposite to the gradient (gradient descent).

$$\theta_{i+1} = \theta_i - \lambda \nabla_\theta L$$

learning rate

Stochastic Gradient Descent

$$\theta_{i+1} = \theta_i - \lambda \nabla_\theta L$$

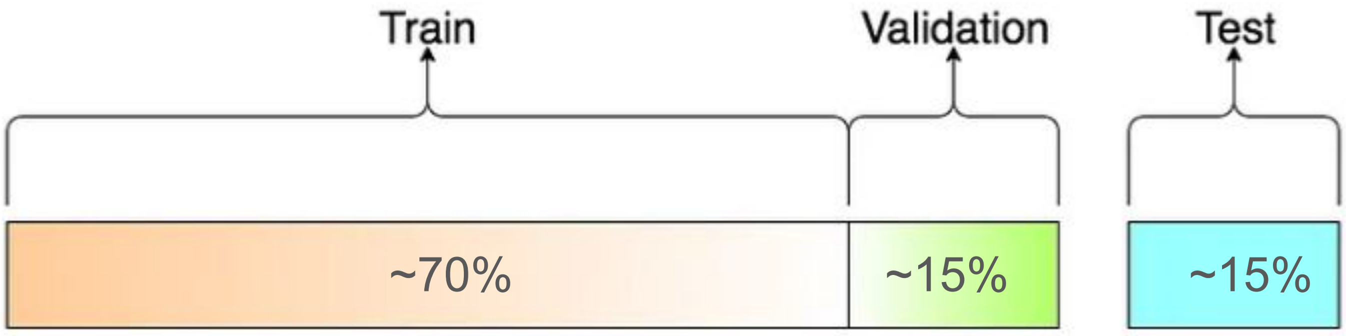
- Compute the derivative using all available data?
  Derivative will be smooth. Fast convergence but you may end up in a local minima

- Compute the derivative using a single data point?
  Derivative will be noisy. Will help escaping local minima, but hard to get convergence

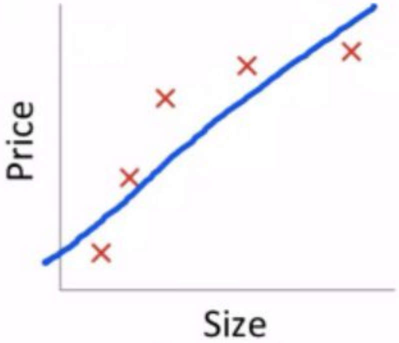
- Compute the derivative using a batch of point?
  Good trade between fast convergence and escape saddle points; also efficient for memory usage

Training, validation and test data:

- **Training Dataset:** The actual dataset that we use to train the model (weights and biases in the case of a Neural Network). The model sees and learns from this data.
- **Validation Dataset:** The sample of data used to provide an unbiased evaluation of a model fit on the training dataset. The model see this data but doesn't learn from it.
- **Test Dataset:** The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset. The model doesn't see or learn from this data.



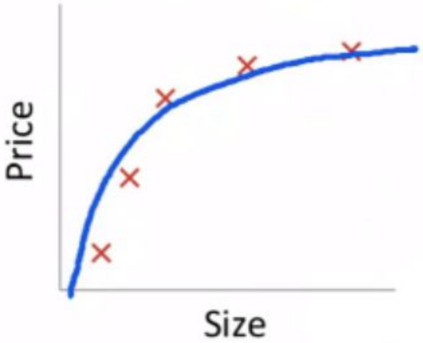
Regularization



$\theta_0 + \theta_1 x$

High bias
(underfit)

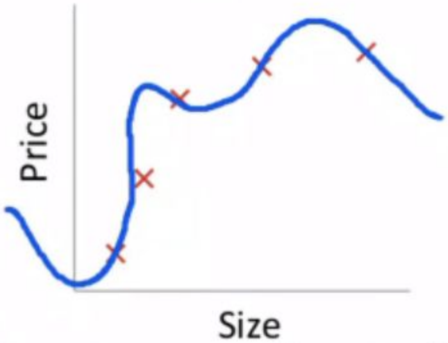
$d=1$



$\theta_0 + \theta_1 x + \theta_2 x^2$

"Just right"

$d=2$

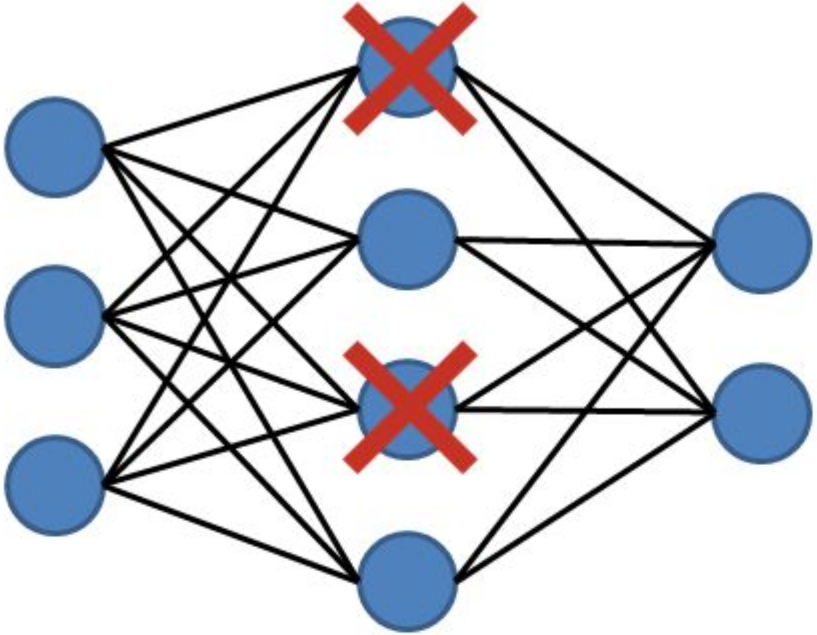


$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

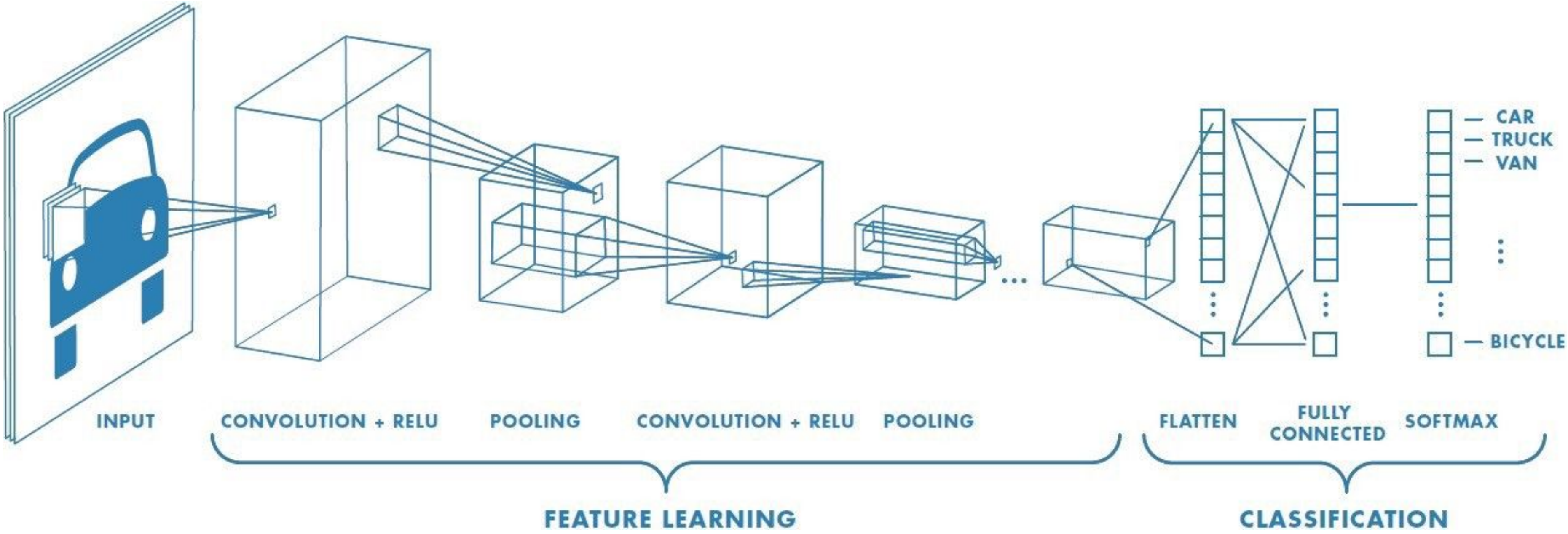
High variance
(overfit)

$d=4$

Regularization

Weight decay	Dropout
$L = \frac{1}{N}\sum_{i=1}^{N}(\theta_{NN} - \theta_{True})^2 + \eta \sum w_i^2$	

Convolutional Neural Networks (CNN)



CNN layers

5 x 5 – Image Matrix

3 x 3 – Filter Matrix

[1*GcI7G-JLAQiEoCON7xFbhg.gif]

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix}0&0&0\\0&1&0\\0&0&0\end{bmatrix}$	
Edge detection	$\begin{bmatrix}1&0&-1\\0&0&0\\-1&0&1\end{bmatrix}$	
	$\begin{bmatrix}0&1&0\\1&-4&1\\0&1&0\end{bmatrix}$	
	$\begin{bmatrix}-1&-1&-1\\-1&8&-1\\-1&-1&-1\end{bmatrix}$	
Sharpen	$\begin{bmatrix}0&-1&0\\-1&5&-1\\0&-1&0\end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}$	

Padding

0	0	0	0	0	0	0	0
0	3	3	4	4	7	0	0
0	9	7	6	5	8	2	0
0	6	5	5	6	9	2	0
0	7	1	3	2	7	8	0
0	0	3	7	1	8	3	0
0	4	0	4	3	2	2	0
0	0	0	0	0	0	0	0

$6 \times 6 \rightarrow 8 \times 8$

*

1	0	-1
1	0	-1
1	0	-1

3×3

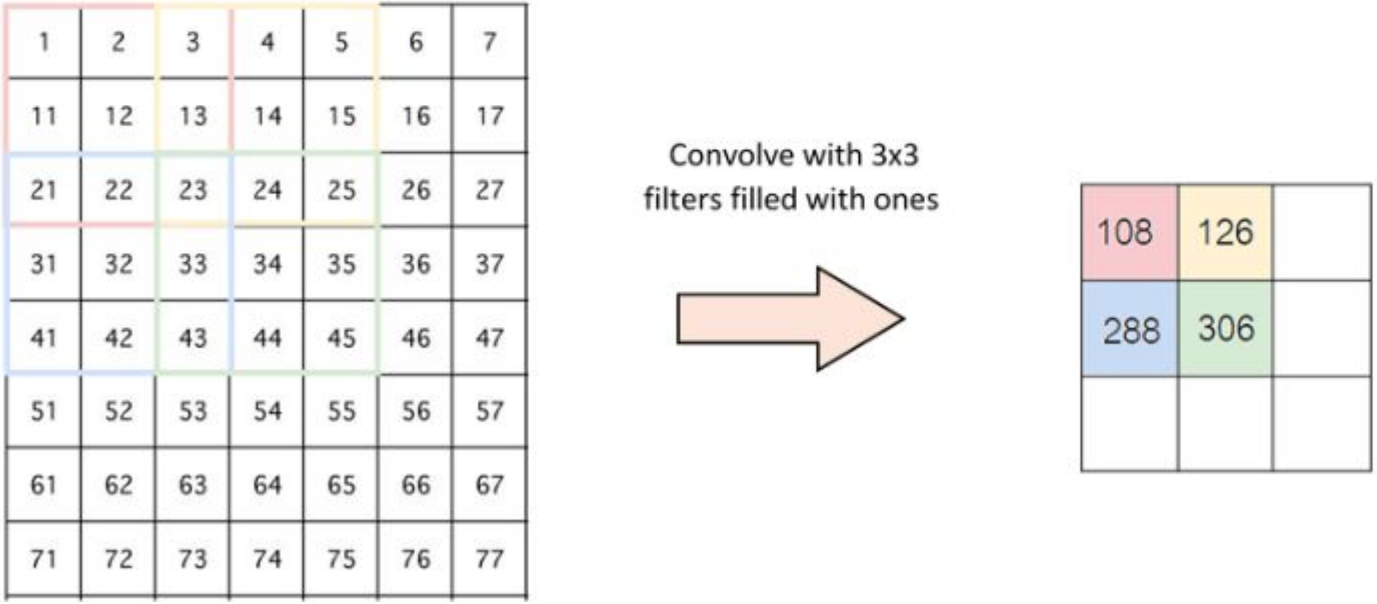
=

-10	-13	1			
-9	3	0			

6×6

1*1VJDP6qDY9-ExTuQVEOlVg.gif

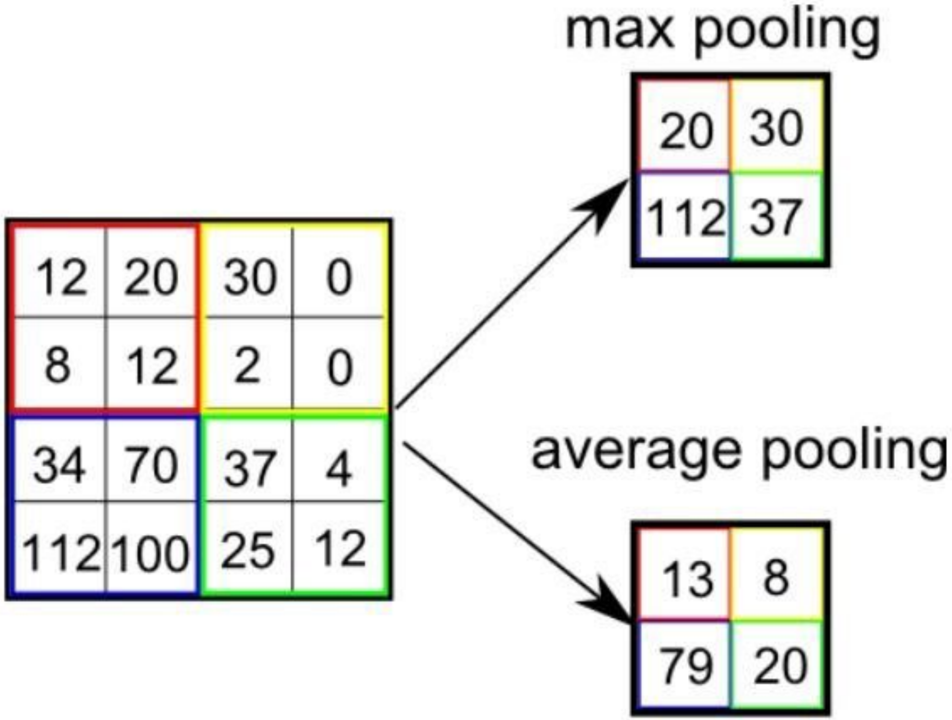
Strides



$$S_{\text{out}} = \frac{S_{\text{in}} + 2\text{Padding} \; - \; \text{Kernel\_size} \; - 2}{\text{Stride}} + 1$$

Pooling

1*uoWYsCV5vBU8SHFPAPao-w.gif



BatchNorm

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$