ESSENTIAL MACHINE LEARNING ALGORITHMS

GRAPHCORE

MACHINE LEARNING

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

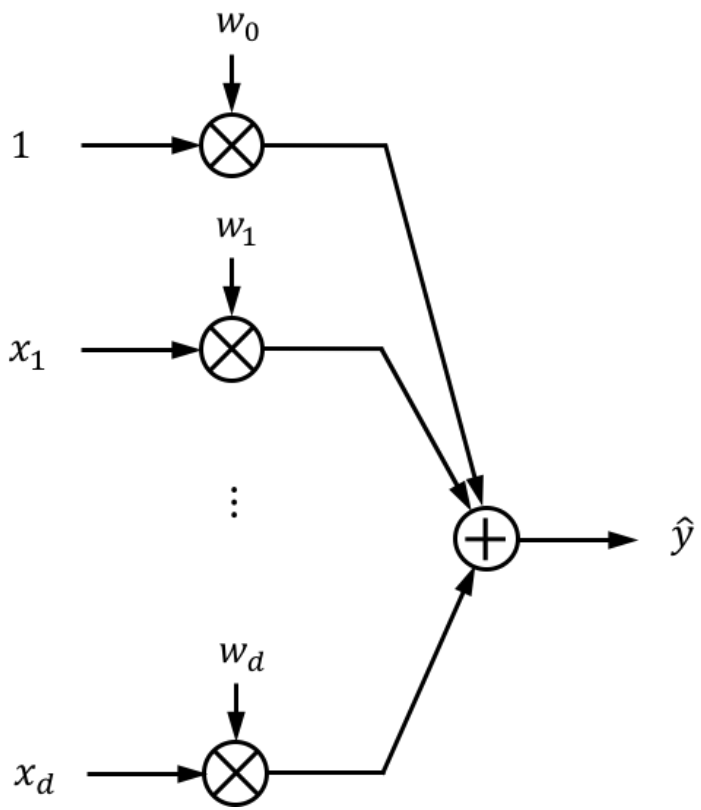
LINEAR REGRESSION

Data set $\{\boldsymbol{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$

Model $y = w_0 + w_1 x_1 + \ldots + w_d x_d + \epsilon$

$= \boldsymbol{w}^T \boldsymbol{x} + \epsilon$

Prediction $\hat{y} = \boldsymbol{w}^T \boldsymbol{x}$



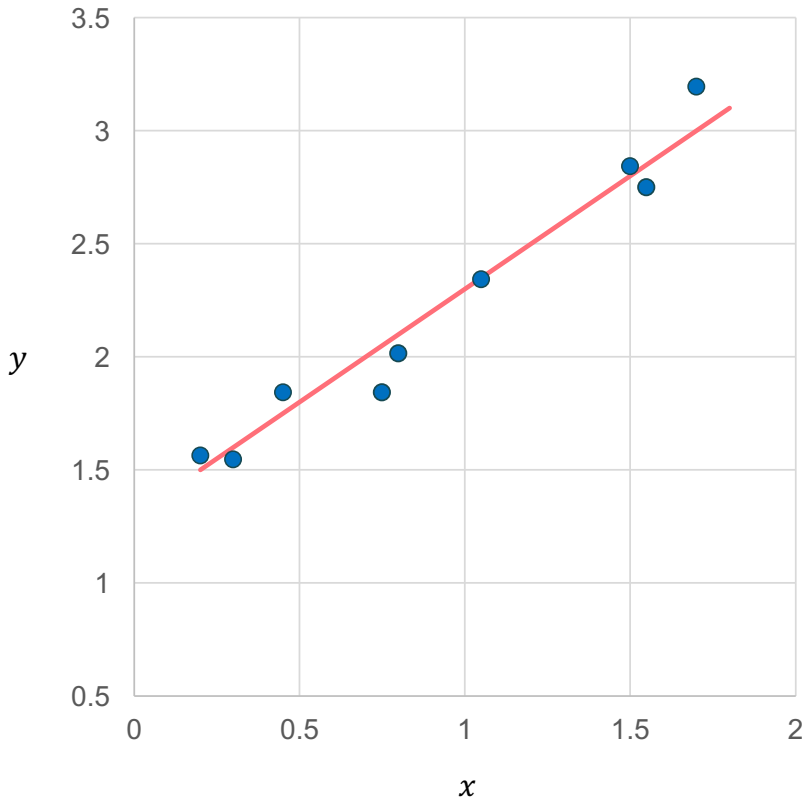
LINEAR REGRESSION

Data set $\{\boldsymbol{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$

Model $y = w_0 + w_1 x_1 + \ldots + w_d x_d + \epsilon$

$= \boldsymbol{w}^T \boldsymbol{x} + \epsilon$

Prediction $\hat{y} = \boldsymbol{w}^T \boldsymbol{x}$



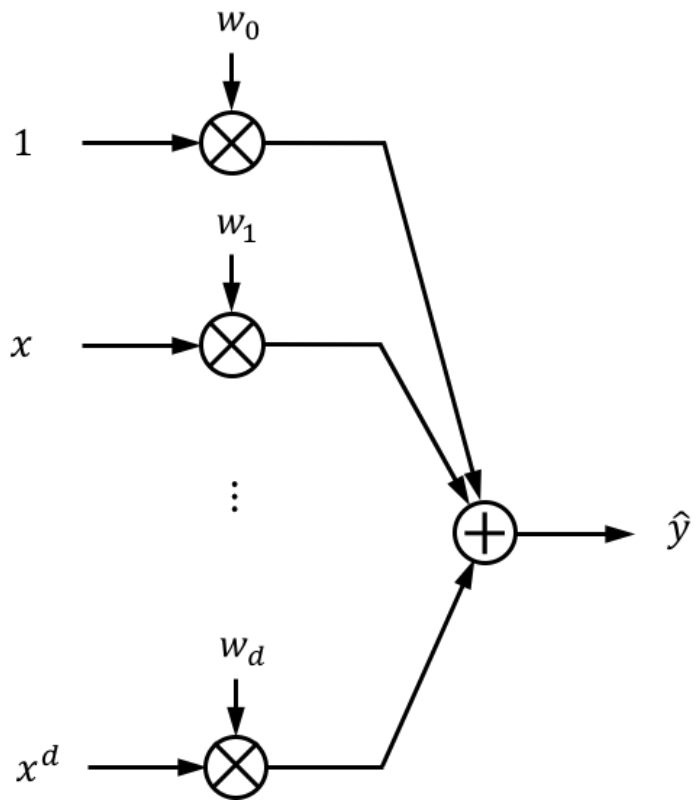
LINEAR REGRESSION

Data set $\{\boldsymbol{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$

Prediction $\hat{y} = \boldsymbol{w}^T \phi(\boldsymbol{x})$

e.g.:

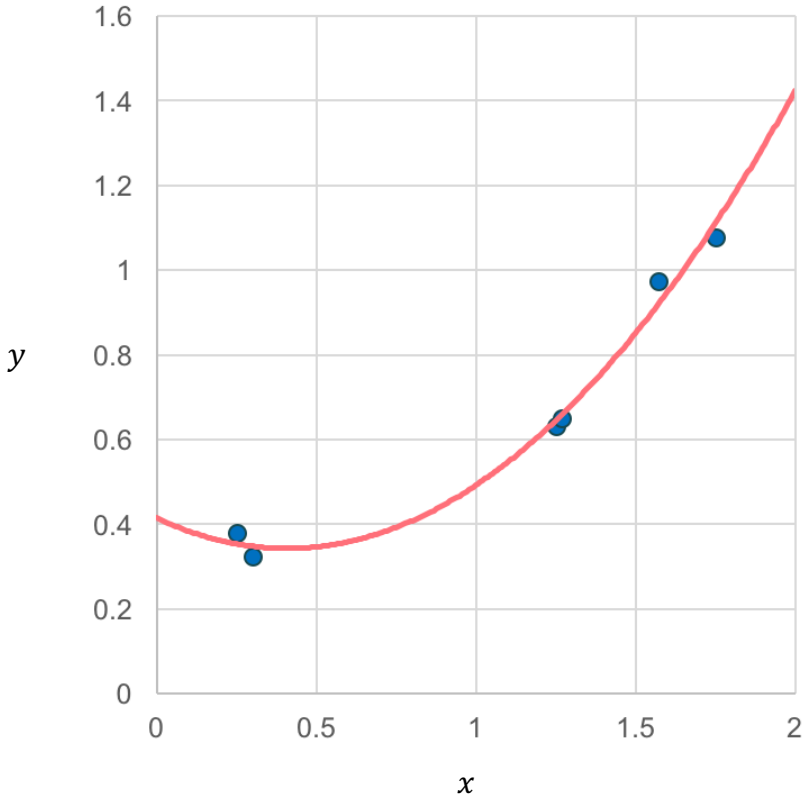
$$\hat{y} = w_0 + w_1 x + w_2 x^2 \ldots + w_d\, x^d$$



OVERFITTING

- *Model capacity*
- *Size of training data set*, N

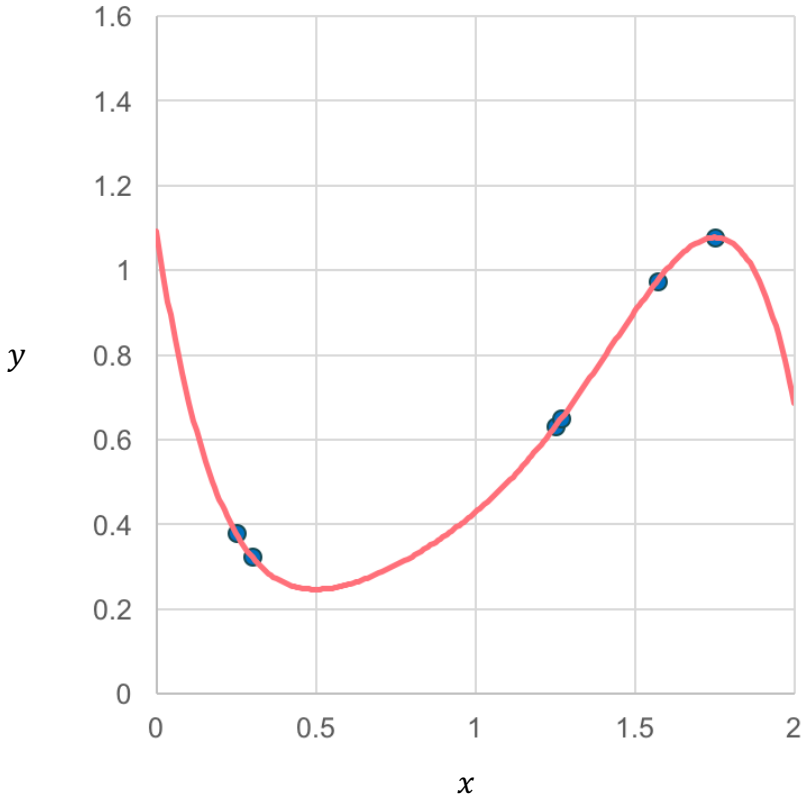
Model capacity and overfitting



OVERFITTING

- *Model capacity*
- *Size of training data set*, N

Model capacity and overfitting



REGULARIZATION

Loss function:

$$J(\boldsymbol{w}) + \textit{norm penalty}$$

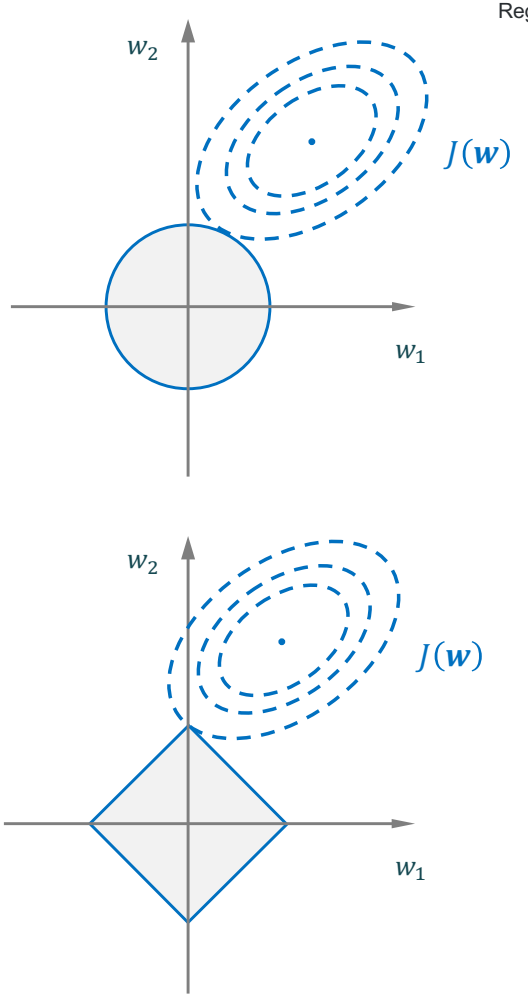
L^2 regularization (*weight decay*)

$$\textit{norm penalty} = \frac{\alpha}{2}\|\boldsymbol{w}\|_2^2$$

L^1 regularization

$$\textit{norm penalty} = \alpha\|\boldsymbol{w}\|_1$$

Regularization



(1) Based on Fig. 3.11 of T. Hastie *et al.*, The Elements of Statistical Learning, 2nd ed., 2009.

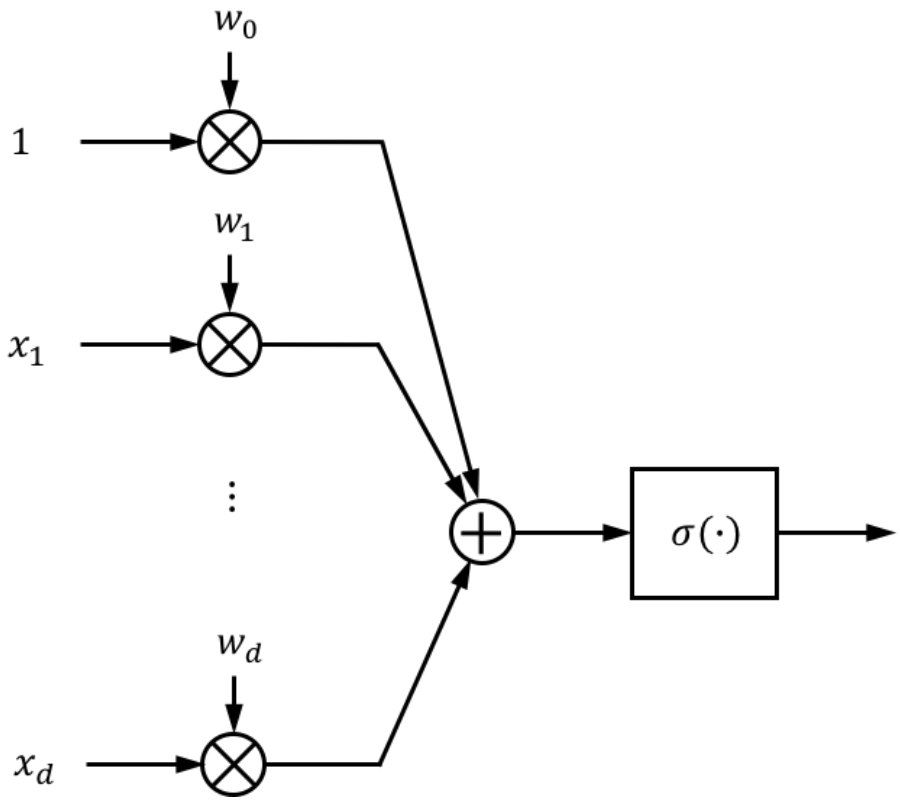
LOGISTIC REGRESSION

Binary Classification

$$\mu(\boldsymbol{x}) = p(y = 1|\boldsymbol{x}, \boldsymbol{w}) = \sigma(\boldsymbol{w}^T \boldsymbol{x})$$

where $\sigma(\cdot)$ is the sigmoid function

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



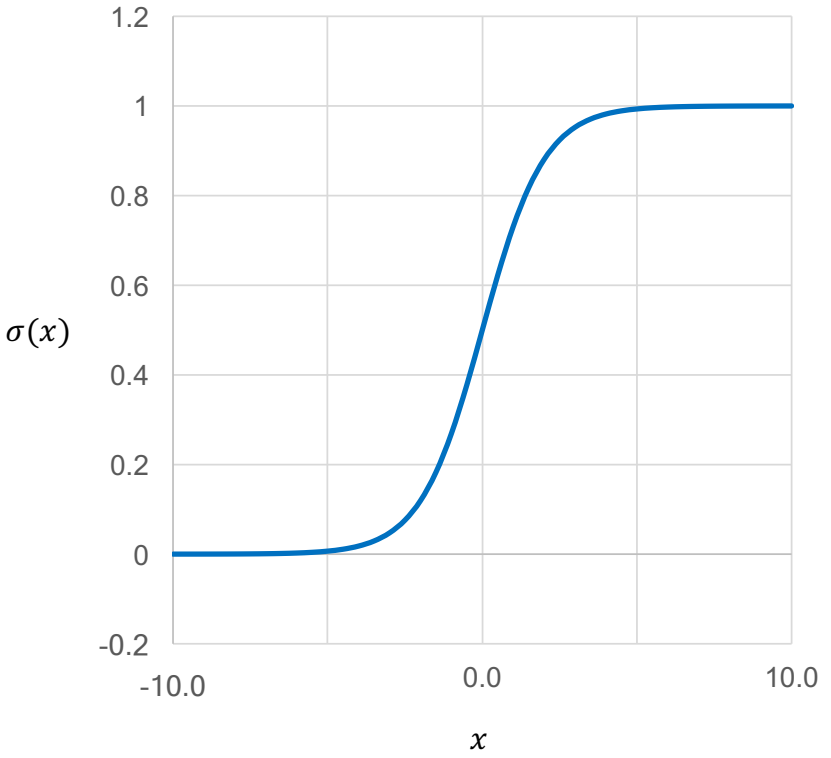
LOGISTIC REGRESSION

Binary Classification

$$\mu(\boldsymbol{x}) = p(y = 1|\boldsymbol{x}, \boldsymbol{w}) = \sigma(\boldsymbol{w}^T \boldsymbol{x})$$

where $\sigma(\cdot)$ is the sigmoid function

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



MAXIMAL MARGIN CLASSIFIER

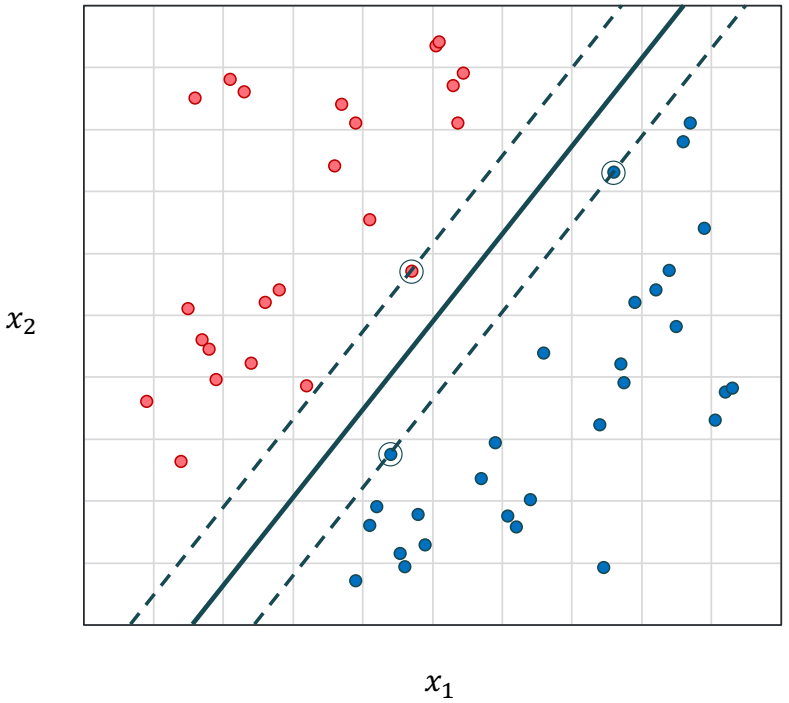
Optimization problem:

$$\max_{\boldsymbol{w}} \; M$$

subject to

$$y^{(i)}\, \boldsymbol{w}^T \boldsymbol{x}^{(i)} > M\,, \quad i = 1, \ldots, N$$

$$\|\boldsymbol{w}\|_2^2 = 1$$



SUPPORT VECTOR CLASSIFIER

Case of not linearly separable classes:

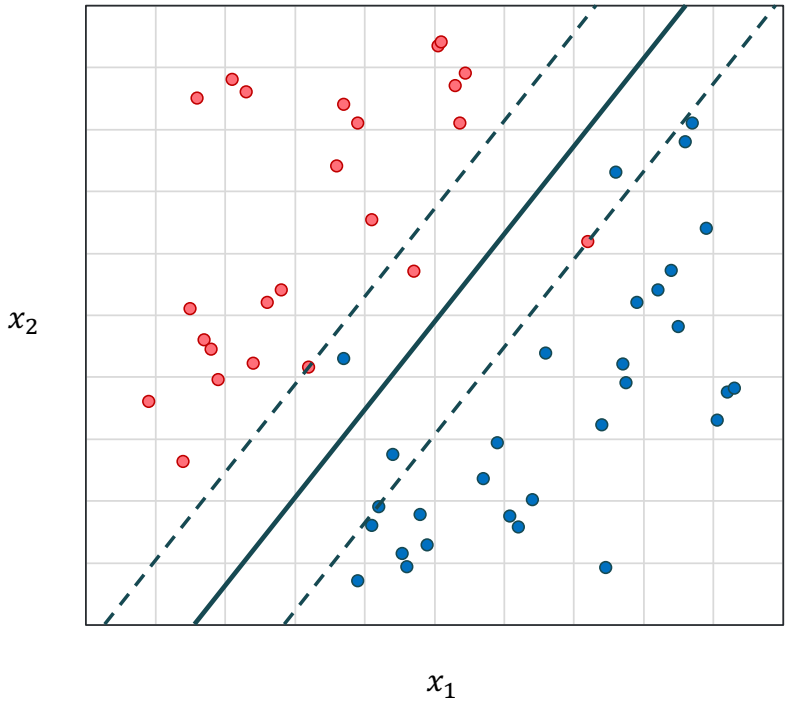
$$\max_{\boldsymbol{w}} M$$

subject to

$$y^{(i)}\, \boldsymbol{w}^T \boldsymbol{x}^{(i)} > M(1-\epsilon^{(i)})\,, \quad i = 1, \dots, N$$

$$\|\boldsymbol{w}\|_2^2 = 1$$

with $\epsilon^{(i)} \geq 0\,, \quad \sum_{i=1}^{N} \epsilon^{(i)} \leq C$



SUPPORT VECTOR MACHINE (SVM)

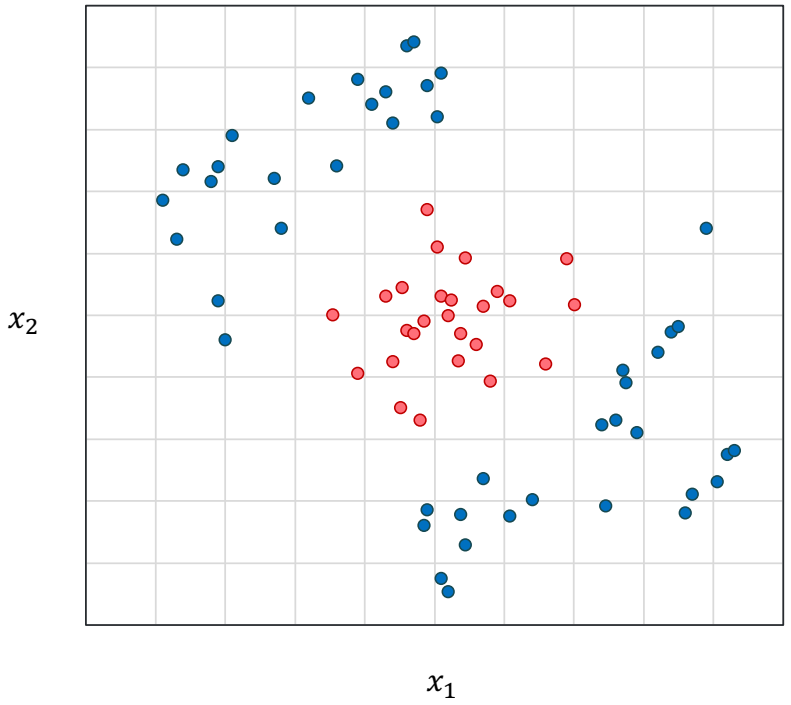
Kernel trick:

$$\hat{y} = \boldsymbol{w}^T \boldsymbol{x} = \sum_{i=1}^{N} \alpha_i \, \boldsymbol{x}^T \boldsymbol{x}^{(i)}$$

replacing $\boldsymbol{x}^T \boldsymbol{x}'$ by $k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}')$

$$\hat{y} = \sum_{i=1}^{N} \alpha_i \, k(\boldsymbol{x}, \boldsymbol{x}^{(i)})$$

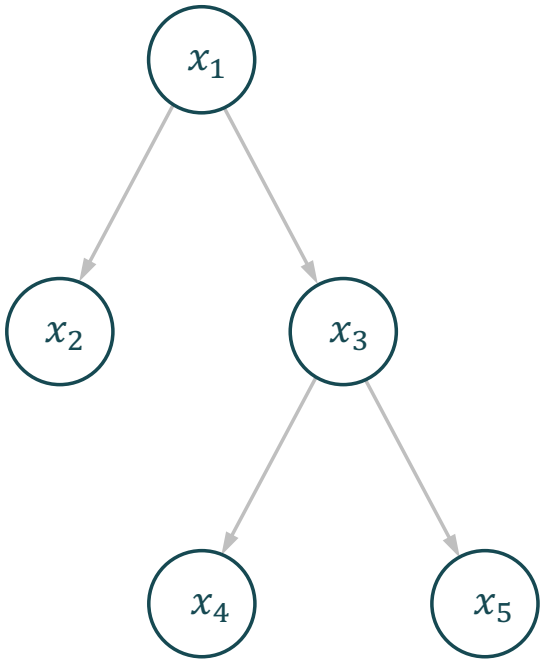
Computation of $k(\boldsymbol{x}, \boldsymbol{x}')$ without working in the enlarged feature space



DIRECTED GRAPHICAL MODELS (BAYES NETS)

Representation of joint distribution by defining *conditional independence* assumptions

$$P(x_1, x_2, \dots, x_n) = P(x_1) \cdot P(x_2 | x_1) \cdot$$
$$\cdot P(x_3 | x_2, x_1) \cdots P(x_n | x_{n-1}, \dots, x_2, x_1)$$



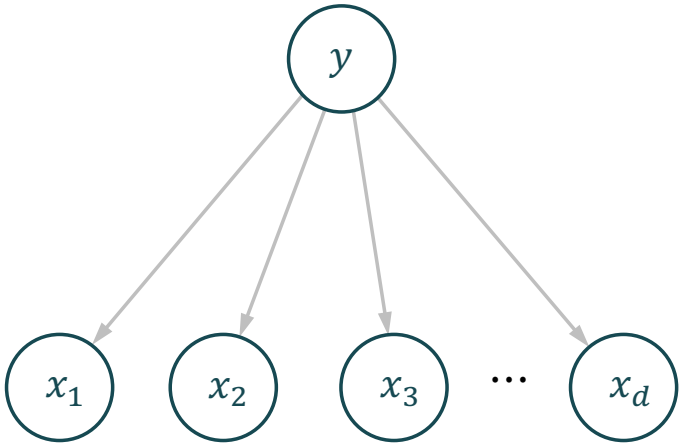
NAÏVE BAYES CLASSIFIER

$$\hat{y} = \arg\max_y P(y|\boldsymbol{x}) = \arg\max_y \frac{P(\boldsymbol{x}|y)P(y)}{P(\boldsymbol{x})}$$

Conditional independence assumption:

$$P(\boldsymbol{x}|y) = P(x_1, x_2, \ldots, x_d|y)$$
$$= P(x_1|y) \cdot P(x_2|y) \cdots P(x_d|y)$$

$$\Rightarrow \quad \hat{y} = \arg\max_y P(y) \prod_{i=1}^{d} P(x_i|y)$$



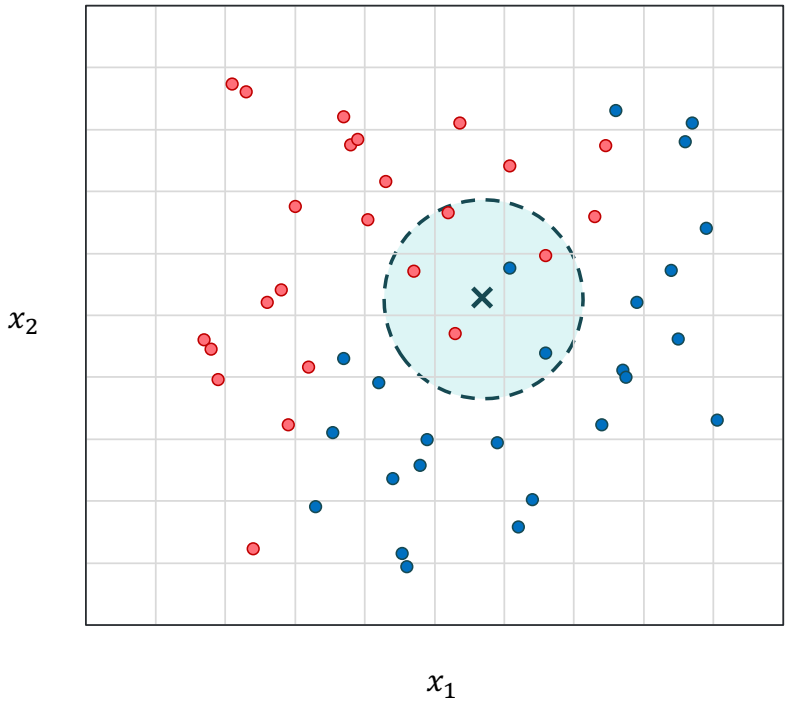
K-NEAREST NEIGHBOURS (*K*-NN)

Data set $\{\boldsymbol{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$

Procedure for a test observation $\boldsymbol{x}$:

- identify the K training points closest to $\boldsymbol{x}$, represented by the set $N_K(\boldsymbol{x})$
- select the class j with largest probability

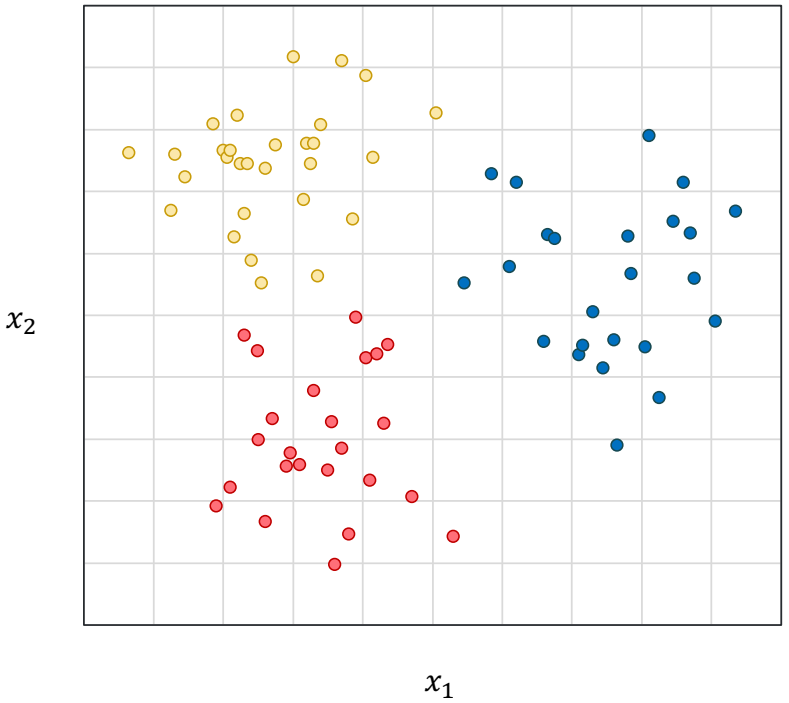
$$P(y = j|\boldsymbol{x}) = \frac{1}{K}\sum_{i\in N_K(\boldsymbol{x})} \mathbb{I}(y^{(i)} = j)$$



K-MEANS CLUSTERING

Procedure after taking K points as the initial *clusters centres* μ_k, repeat the steps:

- assign each data point to the cluster that has the closest centre
- update the position of each cluster centre, by re-computing the mean μ_k

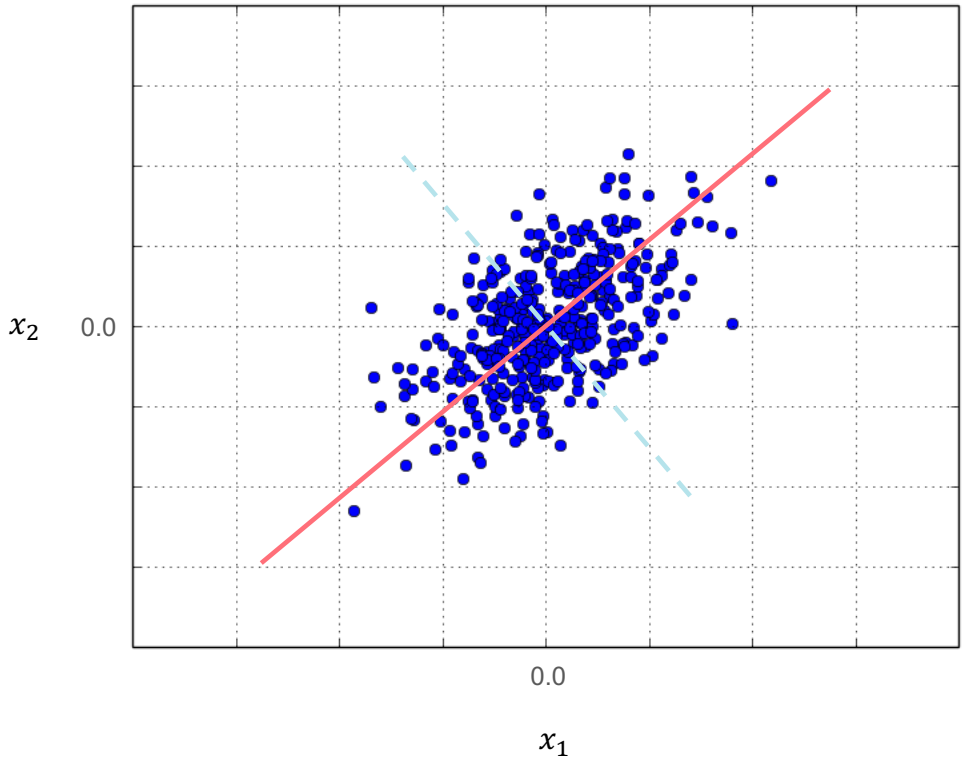


PRINCIPAL COMPONENT ANALYSIS (PCA)

Learns a lower dimensional representation

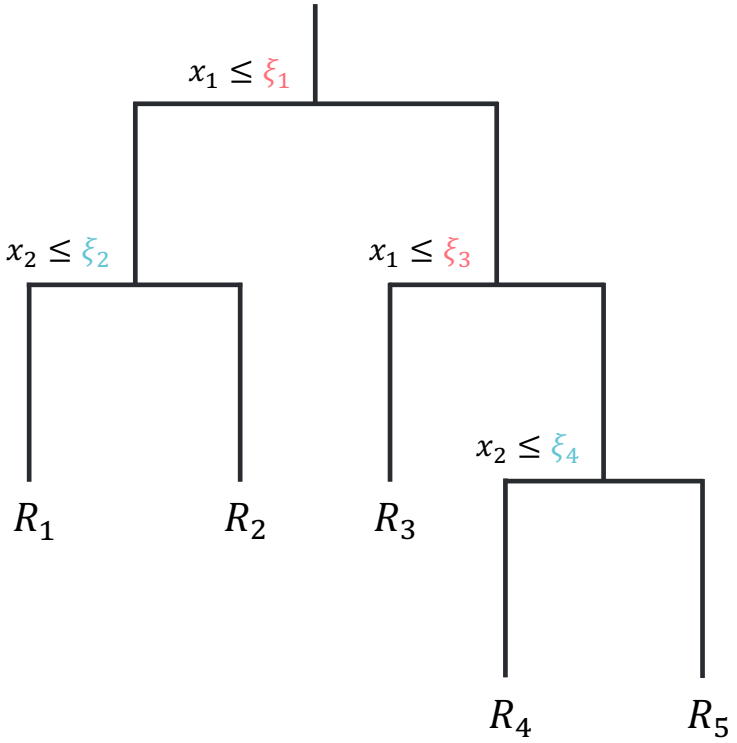
Orthogonal linear transformation $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}$, where $\boldsymbol{W}$ are eigenvectors of $\boldsymbol{X}^T\boldsymbol{X}$ (right singular vectors of $\boldsymbol{X}$):

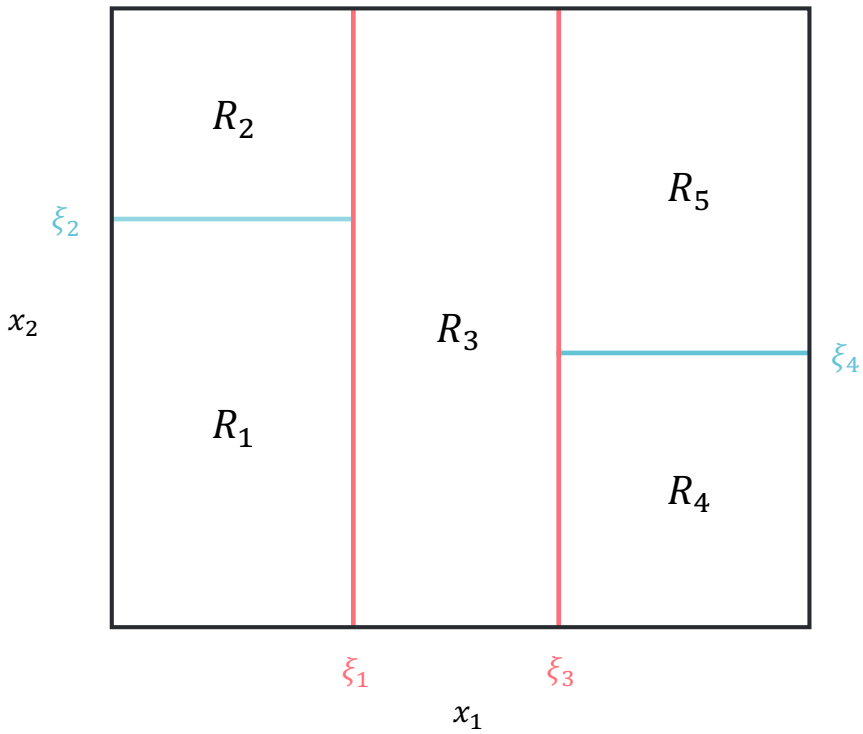
$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{W}^T \quad \Rightarrow \quad \boldsymbol{Z}^T\boldsymbol{Z} = \boldsymbol{\Sigma}^2$$



(1) The matrix $\boldsymbol{X}$ is obtained by stacking the vectors $\boldsymbol{x}^{(i)\,T}$, $i = 1, \ldots, N$

DECISION TREES





(1) Based on Fig. 9.2 of T. Hastie *et al.*, The Elements of Statistical Learning, 2nd ed., 2009.

CLASSIFICATION AND REGRESSION TREES (CART)

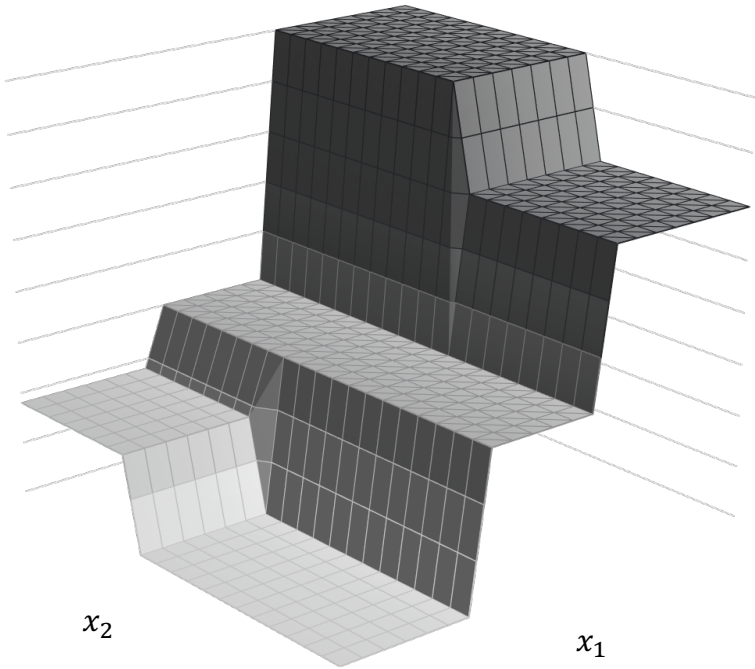
Data set $\{\boldsymbol{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$

Recursive binary splitting:

Regression trees – *RSS cost*

$$\sum_{j=1}^{J} \sum_{i \in R_j} \left(y^{(i)} - \hat{y}_{R_j} \right)^2$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the j-th region

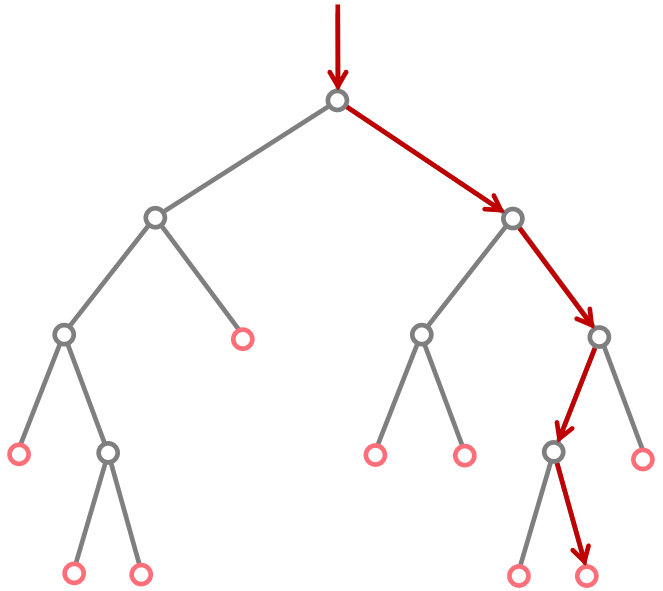


DECISION TREES

Visualization and interpretability

Significantly improved prediction performance using *ensembles of large number of trees*:

- *Bagging*
- *Random Forests*
- *Boosting*



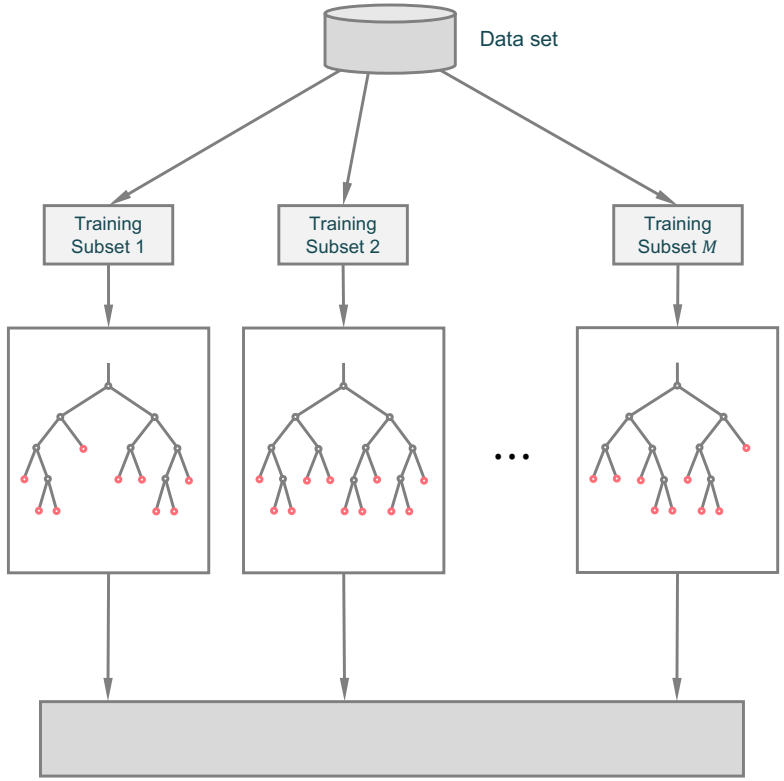
BAGGING (BOOTSTRAP AGGREGATING)

Improves performance by averaging the estimates produced by M different trees on different *subsets of the data*, drawn randomly with replacement (*Bootstrap*):

$$f(\boldsymbol{x}) = \frac{1}{M} \sum_{m=1}^{M} f_m(\boldsymbol{x})$$

where f_m denotes the m-th tree

Bagging

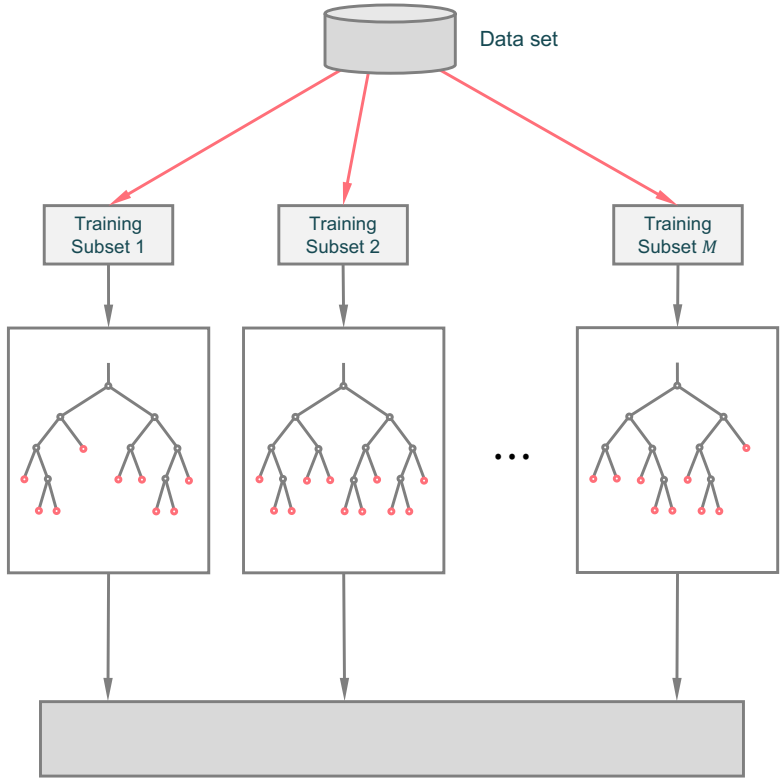


RANDOM FORESTS

Tries to *decorrelate* the base learners, learning the M trees based on both

- a different *subsets of the data*, drawn randomly with replacement (*Bootstrap*), and
- a different, randomly chosen *subset of the predictors* $x_1, \ldots, x_d$ (typically $\sqrt{d}$ predictors) at each split

Reduced variance and reduced test error

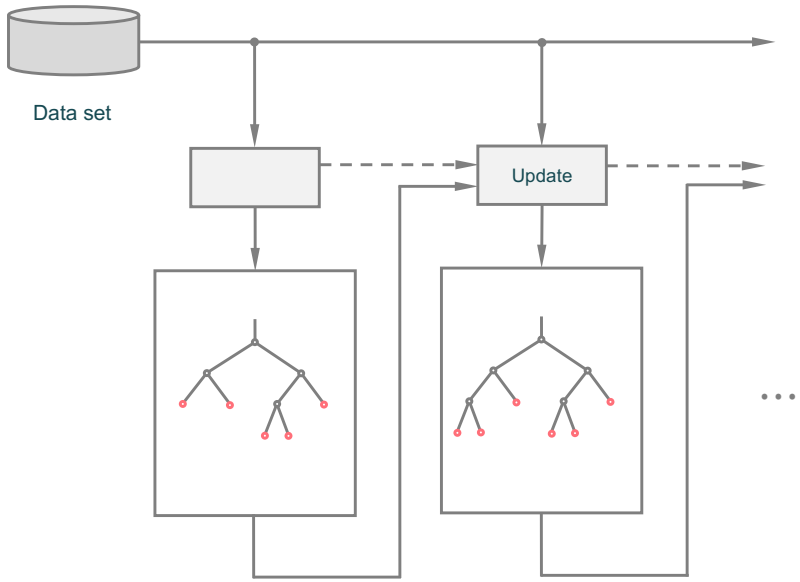


BOOSTING

Builds an ensemble combining *weak learners*

Similar to Bagging, except that

- trees are grown *sequentially*:
  each tree is grown using information from previously grown trees
- does not involve Bootstrap resampling:
  each tree is fit on a modified version of the original data set



NEURAL NETWORKS

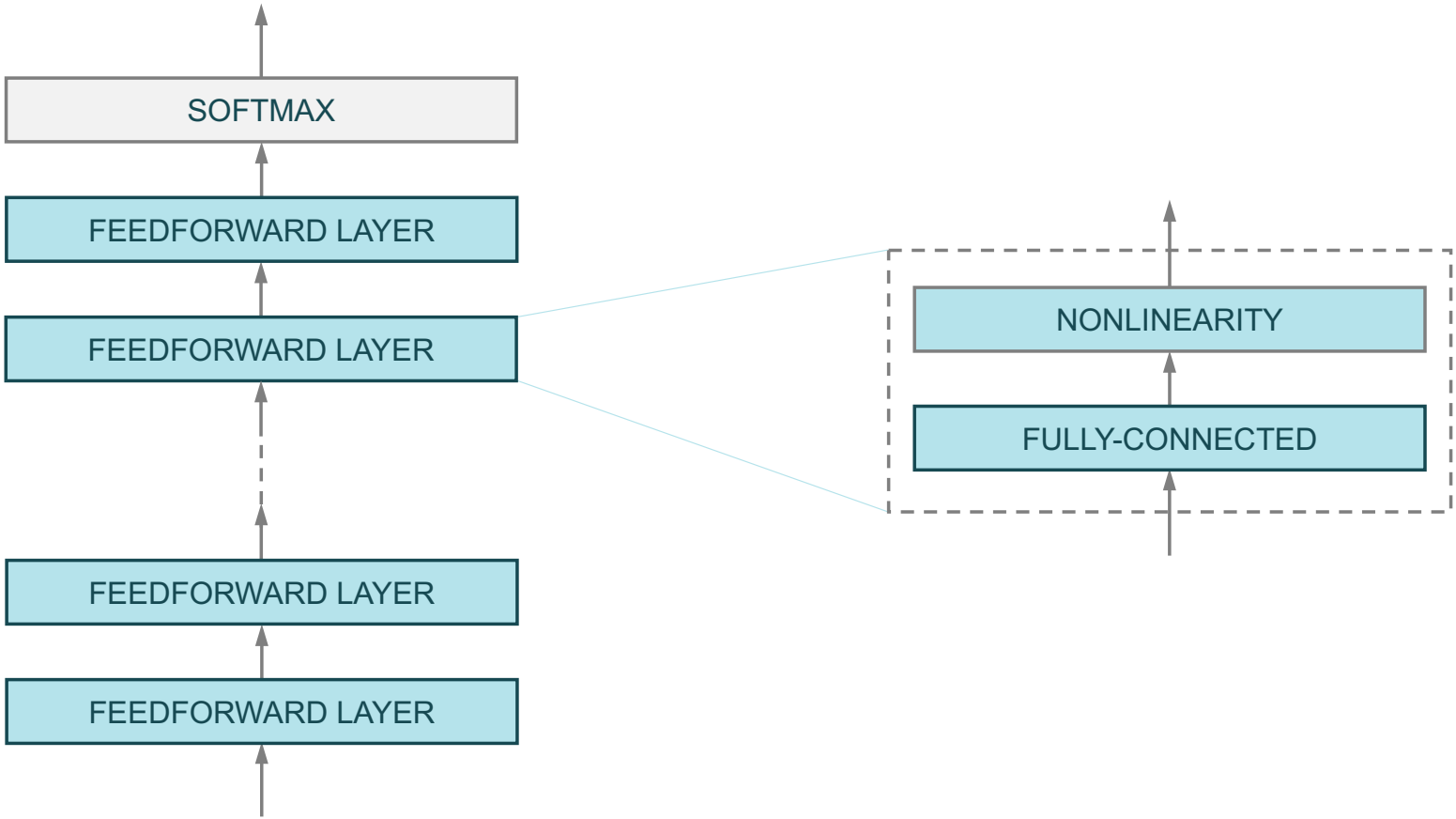
- Feedforward Neural Networks (FNN)
- Convolutional Neural Networks (CNN)

  image processing applications (e.g., face recognition, sign and pedestrian detection for autonomous cars)

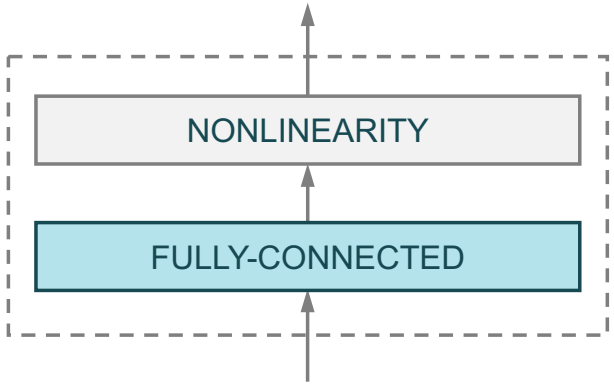
- Recurrent Neural Networks (RNN)

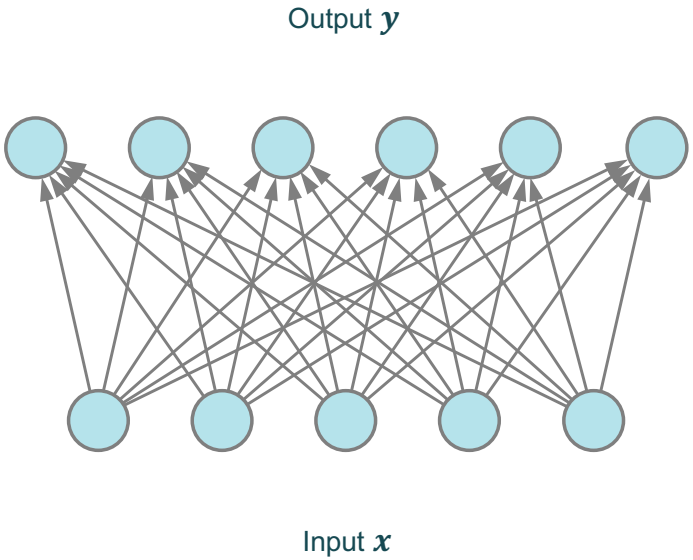
  natural language processing applications (e.g., speech transcription, machine translation)

FEEDFORWARD NEURAL NETWORKS

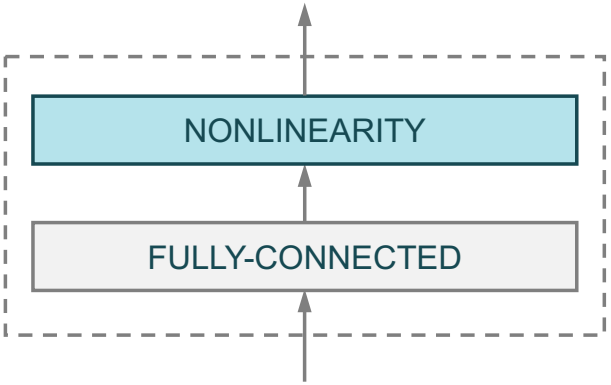


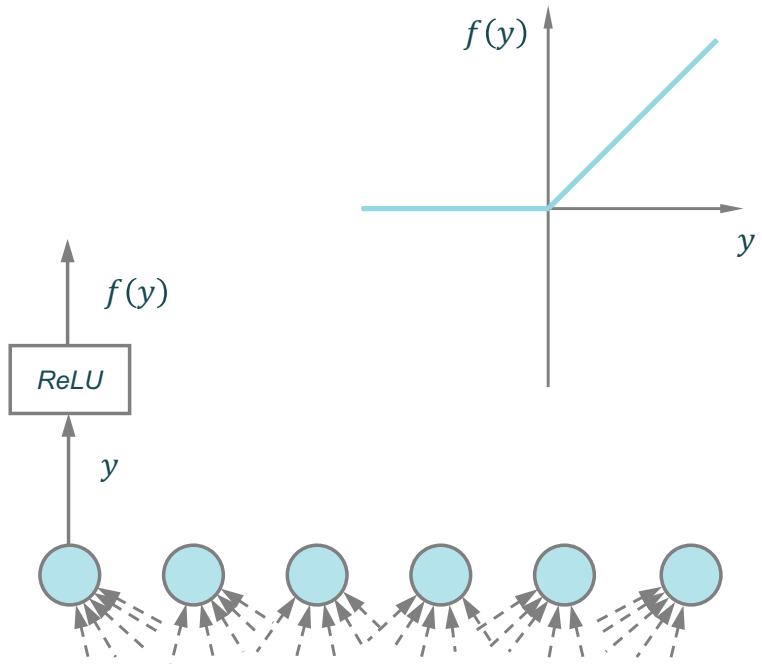
FEEDFORWARD NEURAL NETWORKS





FEEDFORWARD NEURAL NETWORKS





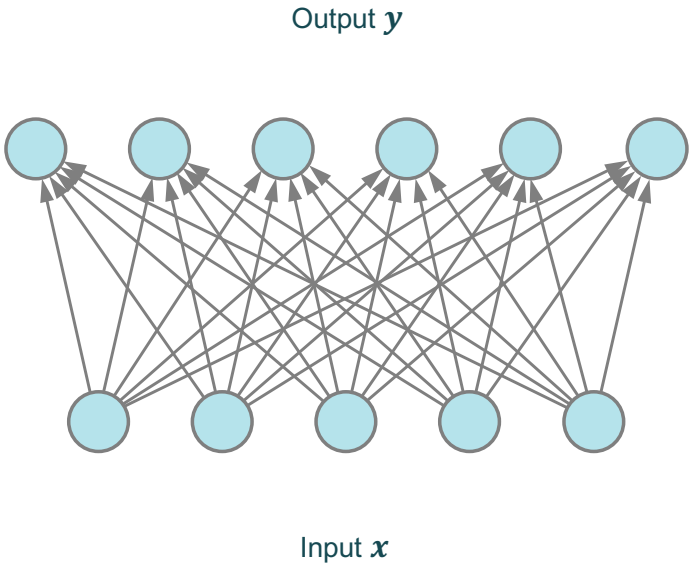
FEEDFORWARD NEURAL NETWORKS

Fully-connected *linear layer*:

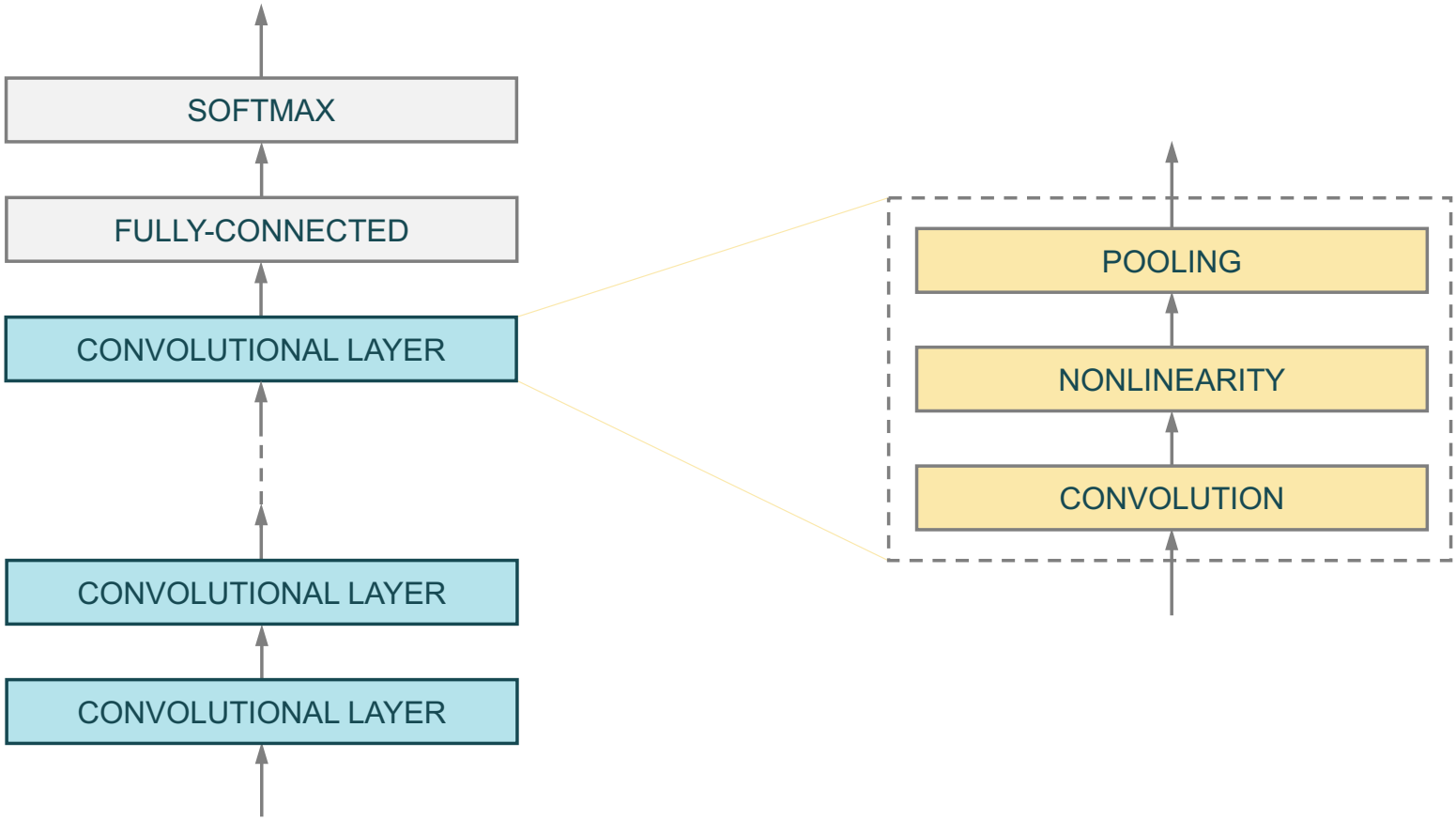
$$\boldsymbol{y} = \boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}$$

Nonlinear *activation function*:

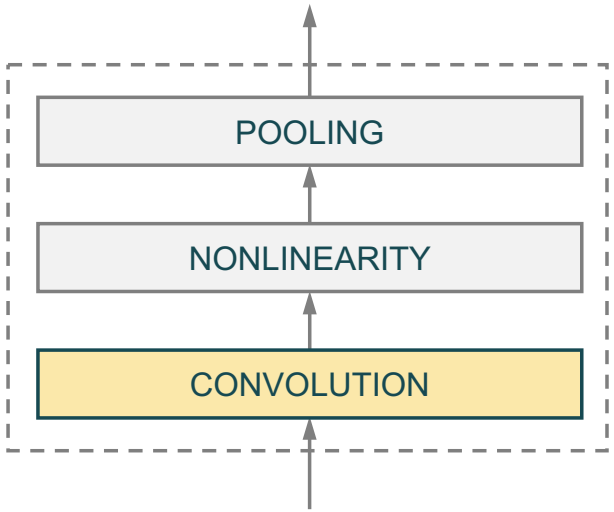
$$\boldsymbol{z} = f(\boldsymbol{y})$$

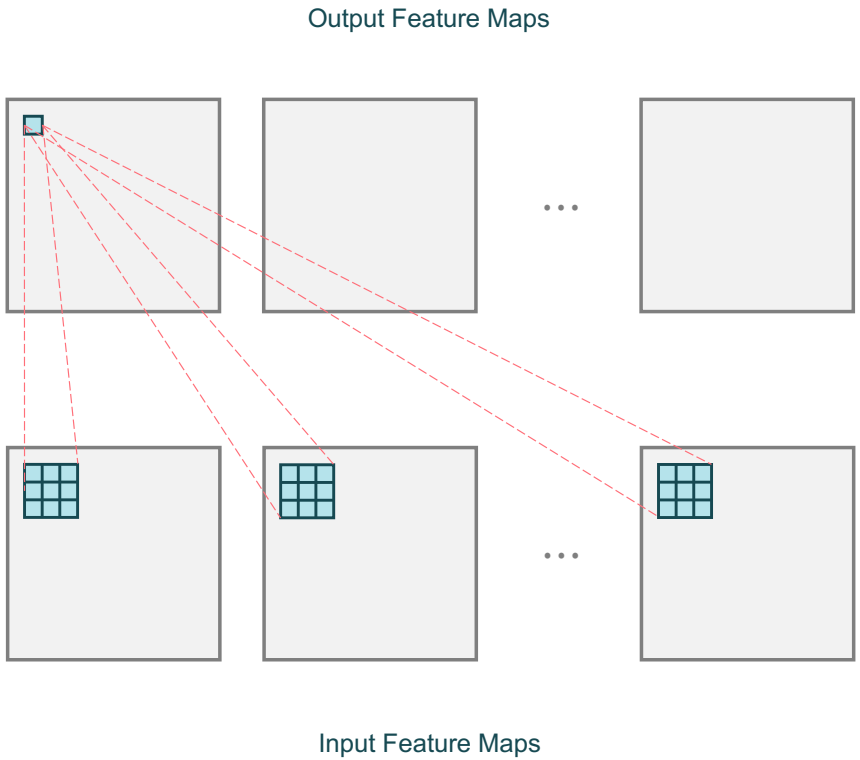


CONVOLUTIONAL NEURAL NETWORKS

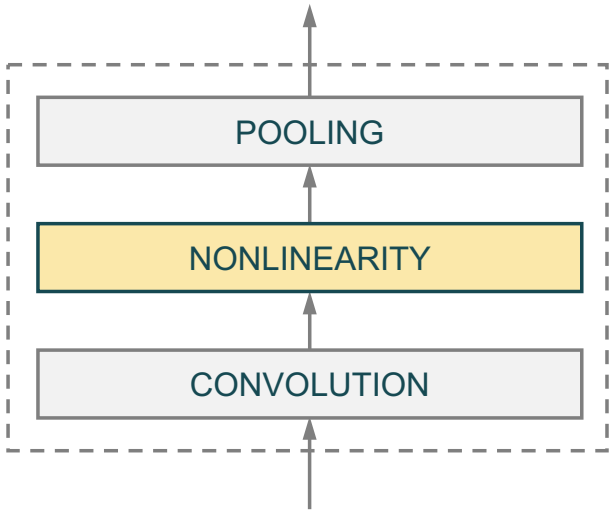


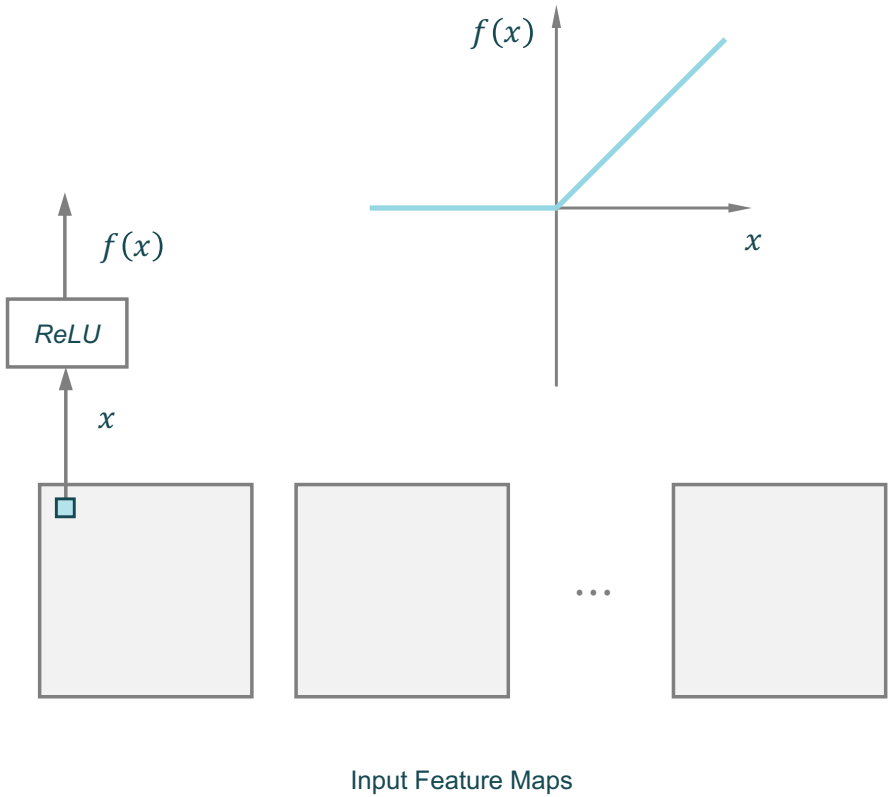
CONVOLUTIONAL NEURAL NETWORKS



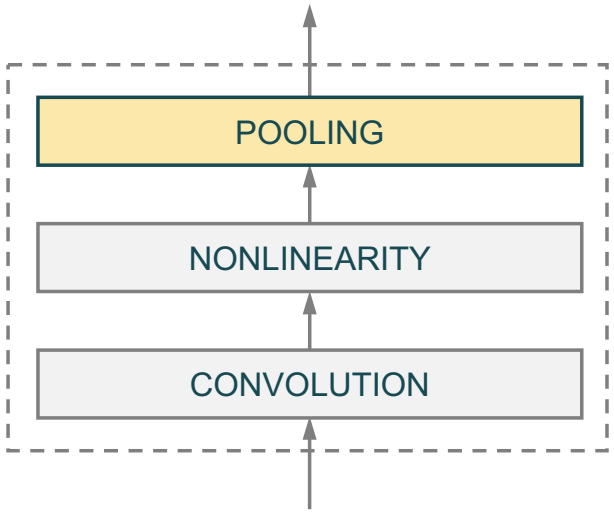


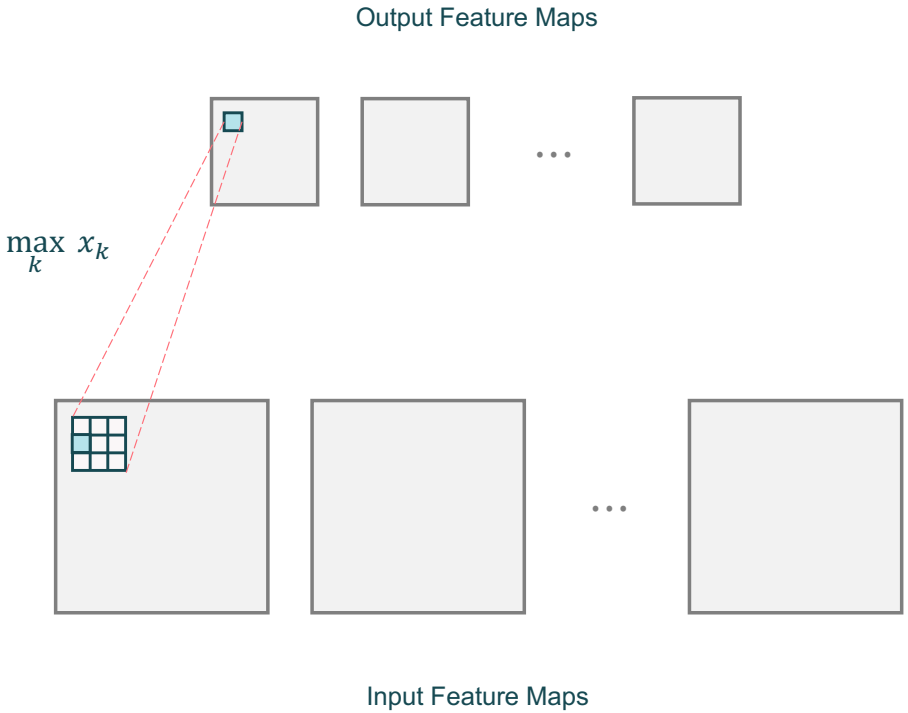
CONVOLUTIONAL NEURAL NETWORKS



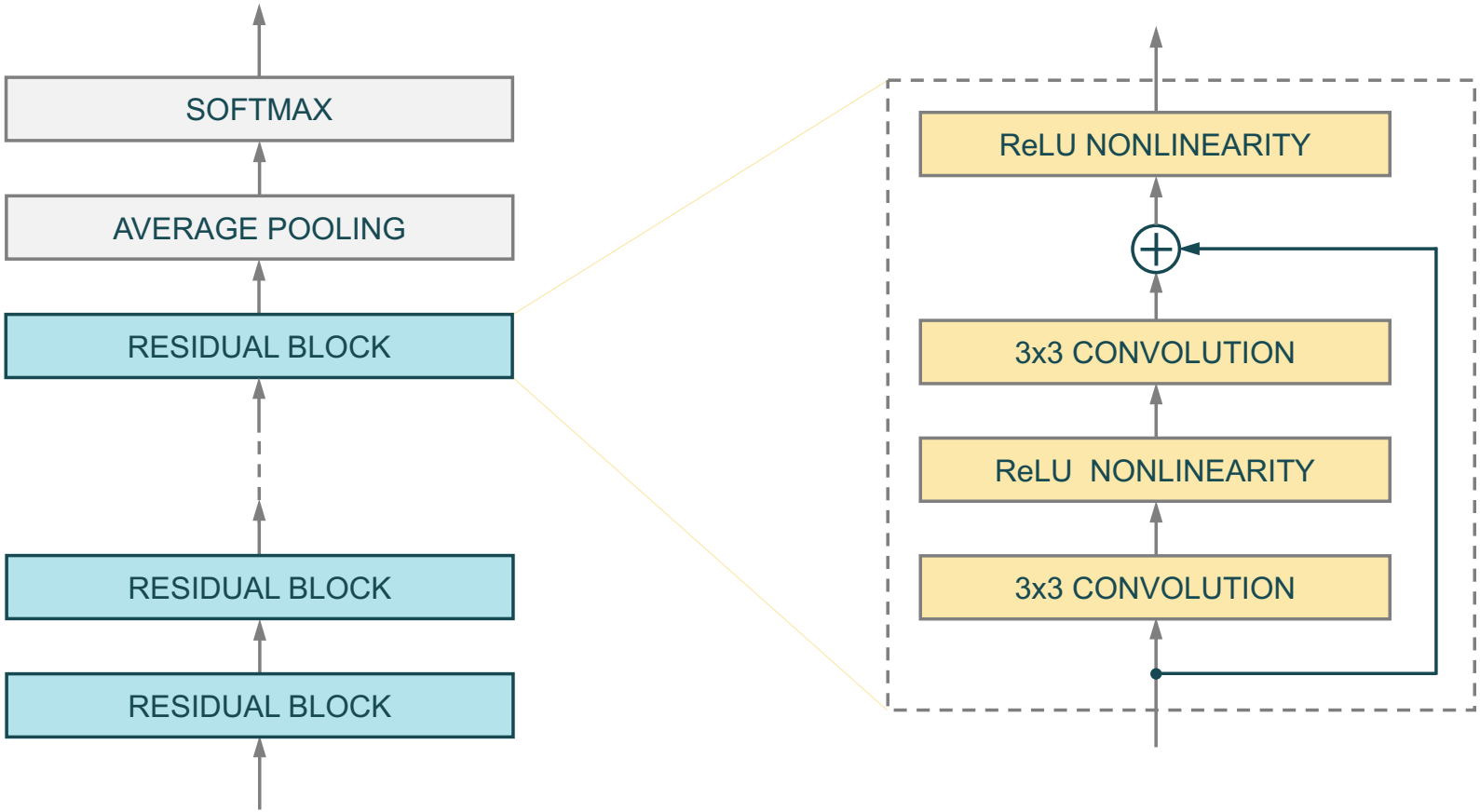


CONVOLUTIONAL NEURAL NETWORKS

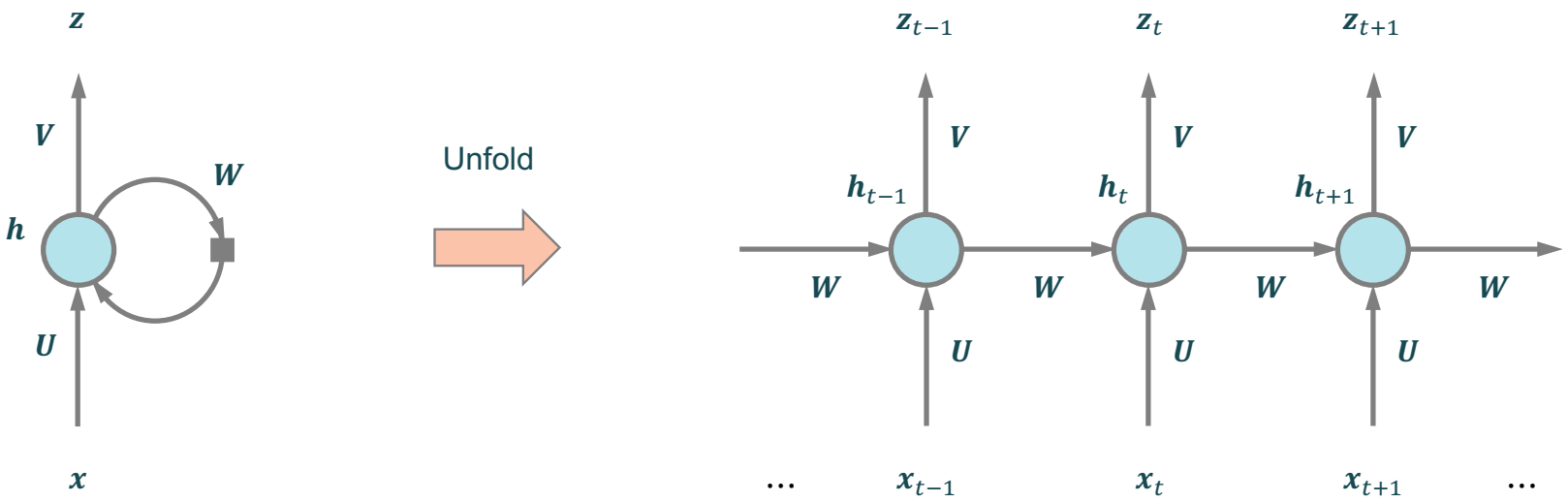




RESNET ARCHITECTURE



RECURRENT NEURAL NETWORKS



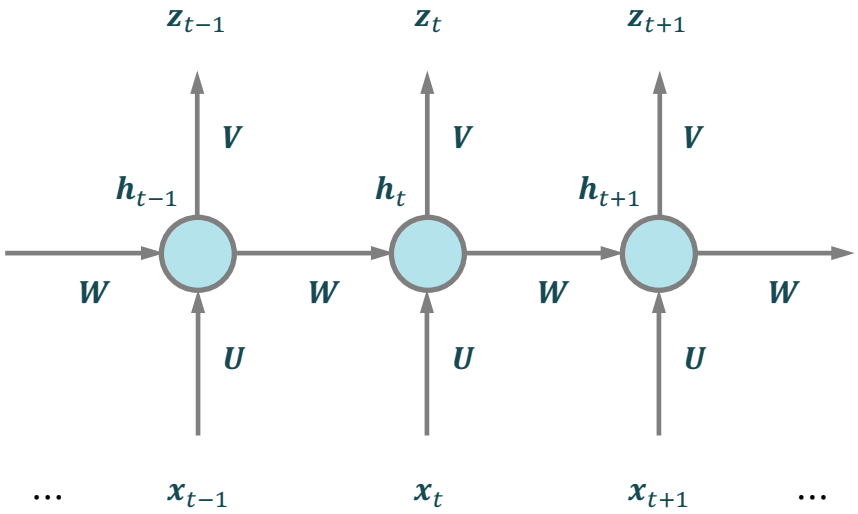
RECURRENT NEURAL NETWORKS

Efficient parametrization: recurrent application of the same function

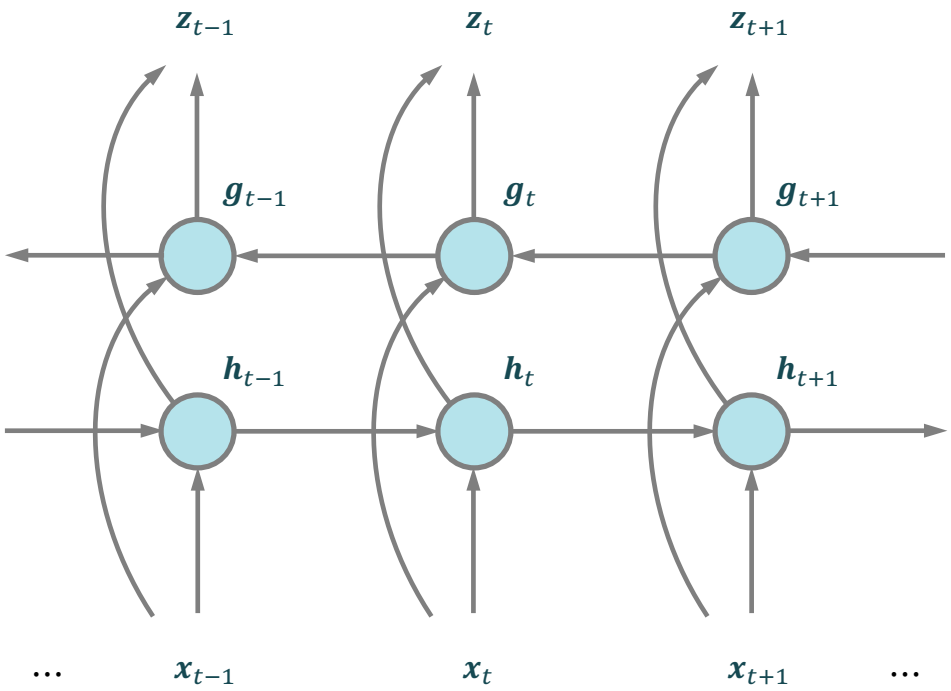
$$\boldsymbol{y}_t = \boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{W}\boldsymbol{h}_t + \boldsymbol{b}$$

$$\boldsymbol{h}_t = \tanh(\boldsymbol{y}_t)$$

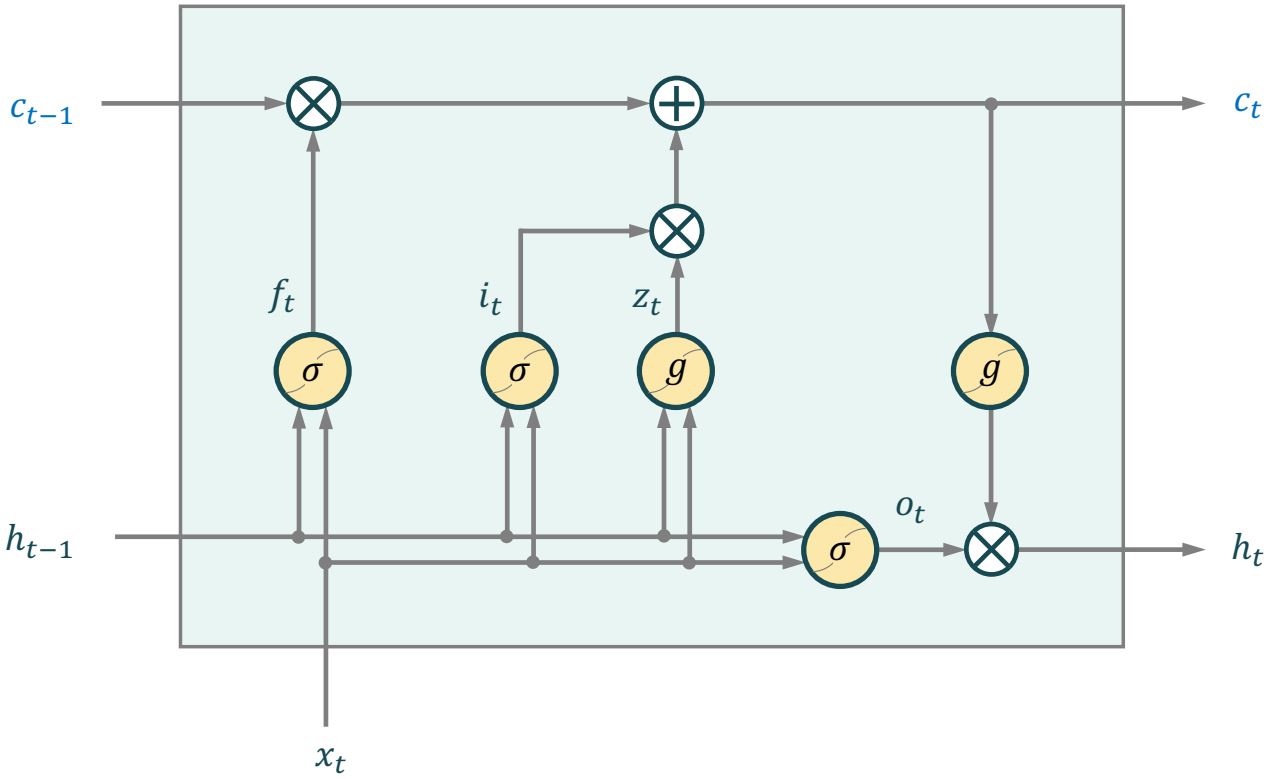
$$\boldsymbol{z}_t = \boldsymbol{V}\boldsymbol{h}_t + \boldsymbol{c}$$



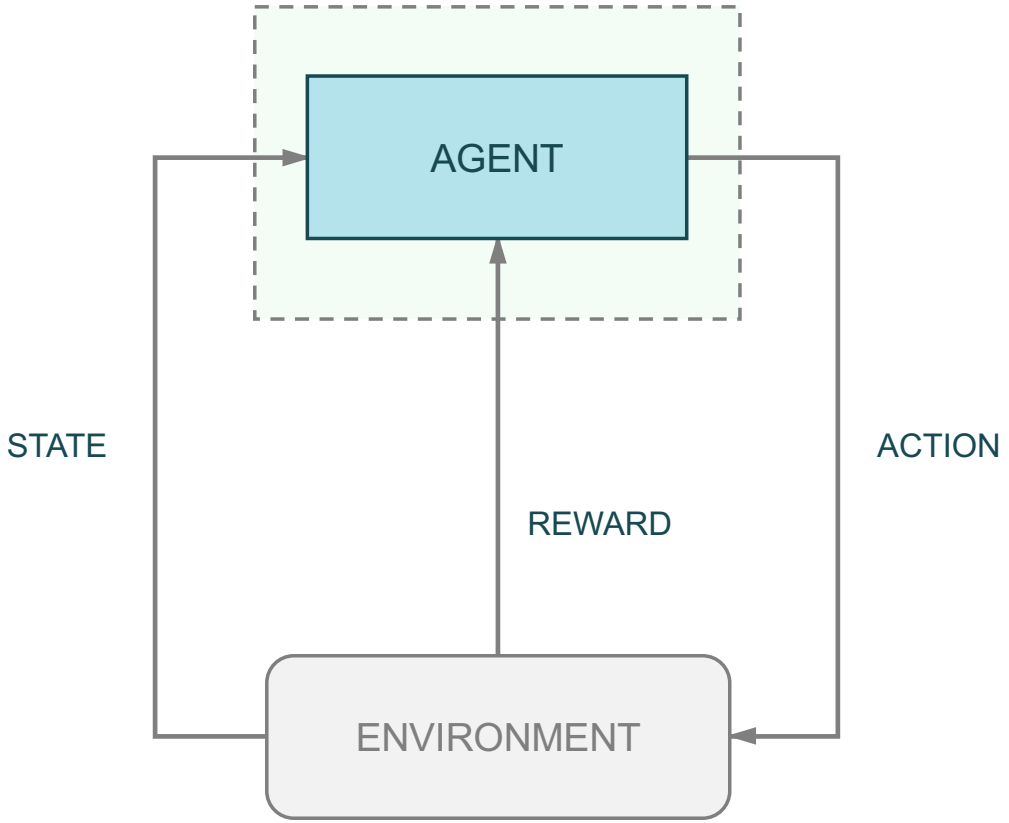
RECURRENT NEURAL NETWORKS



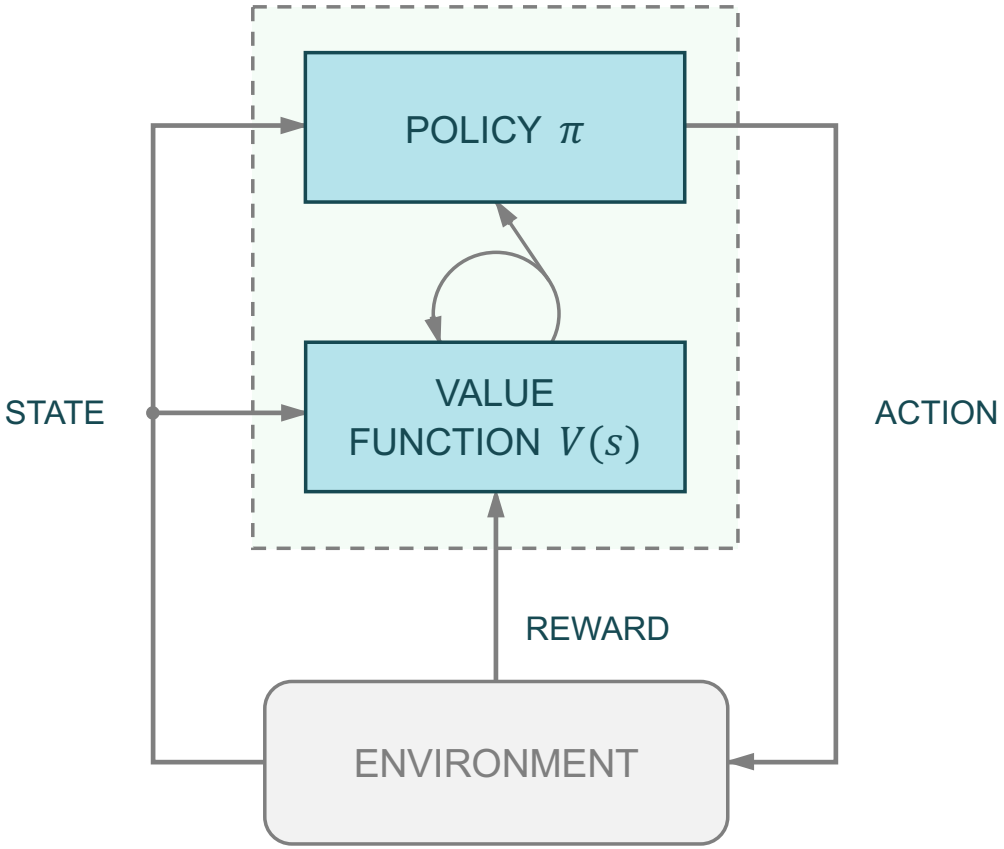
LONG SHORT-TERM MEMORY (LSTM) CELL



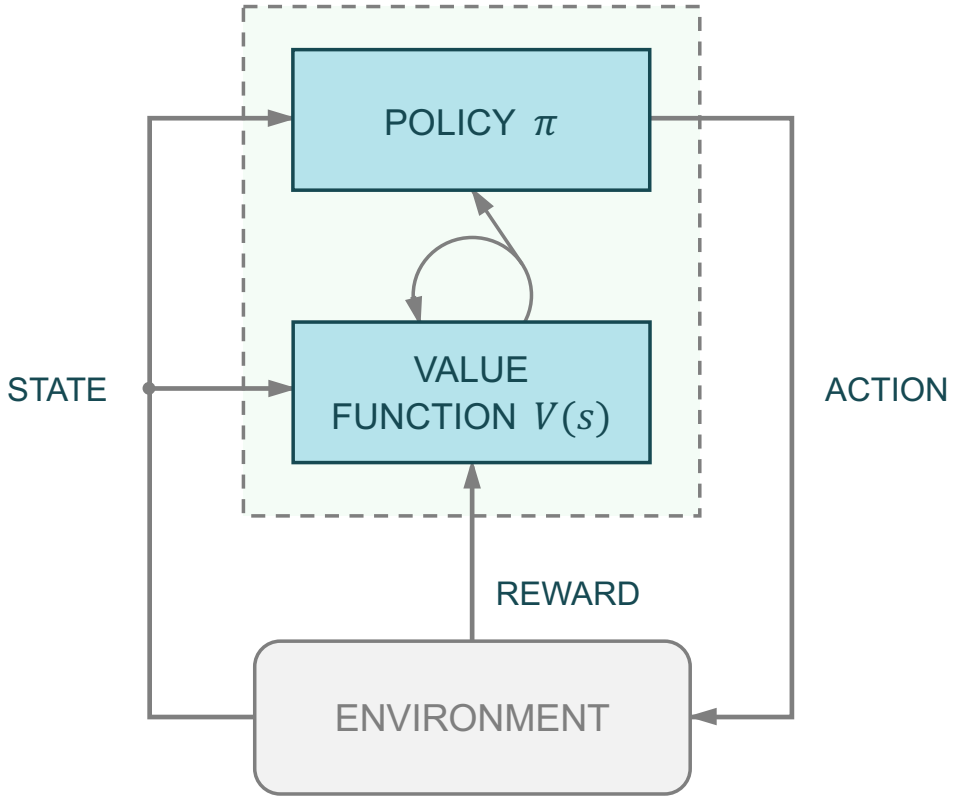
REINFORCEMENT LEARNING



REINFORCEMENT LEARNING



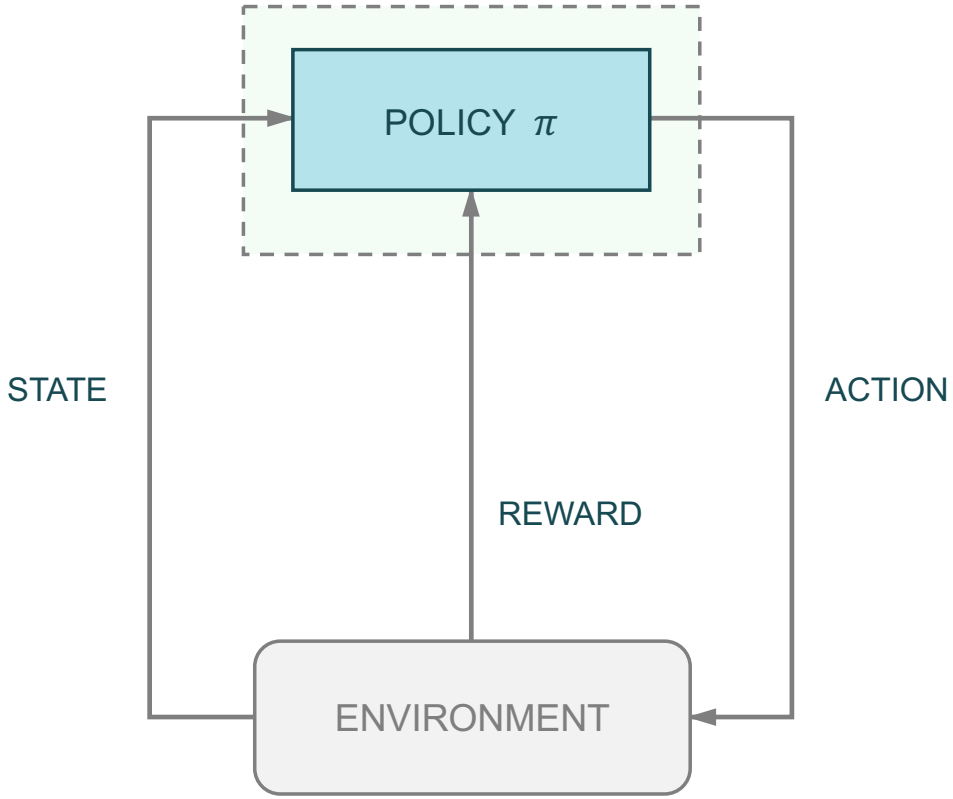
REINFORCEMENT LEARNING TASKS



Reinforcement Learning (RL):

- Maximization of *expected reward*
- Gradient based optimization by *backpropagation*
- Exploration by *trial-and-error* on *actions*

REINFORCEMENT LEARNING TASKS



Evolution Strategies (ES):

- Maximization of *expected reward*
- *Black-box optimization* using *finite differences*
- Exploration by *trial-and-error* on *parameters*

THANK YOU

Carlo Luschi
carlo@graphcore.ai