



UNIVERSITY
OF AMSTERDAM

Machine Learning for Physics and Astronomy

Juan Rojo

VU Amsterdam & Theory group, Nikhef

Natuur- en Sterrenkunde BSc (Joint Degree), Honours Track
Lecture 5, 28/09/2020

Today's lecture

- 📌 Pattern recognition and classification with Convolutional Neural Networks
- 📌 Bayesian Neural Networks
- 📌 Reinforcement Learning
- 📌 Guest lecture by **Dr. Christoph Weniger** on applications of ML for astroparticle physics

Convolutional Neural Networks

Learning with symmetry

Like physical systems, many datasets and supervised learning tasks also possess additional **symmetries and structure** what can (and should) be exploited



e.g. we want to train a classifier to identify pictures of cats. What **high-level features** must one learn first?

Learning with symmetry

Like physical systems, many datasets and supervised learning tasks also possess additional **symmetries and structure** what can (and should) be exploited



e.g. we want to train a classifier to identify pictures of cats. What **high-level features** must one learn first?

- 💡 *The features that define ``cat'' are local in the picture: whiskers, tail, paws ...: **locality***

Learning with symmetry

Like physical systems, many datasets and supervised learning tasks also possess additional **symmetries and structure** what can (and should) be exploited



e.g. we want to train a classifier to identify pictures of cats. What **high-level features** must one learn first?

- ✿ *The features that define ``cat'' are local in the picture: whiskers, tail, paws ...: **locality***
- ✿ *Cats can be anywhere in the image: **translational invariance***

Learning with symmetry

Like physical systems, many datasets and supervised learning tasks also possess additional **symmetries and structure** what can (and should) be exploited



e.g. we want to train a classifier to identify pictures of cats. What **high-level features** must one learn first?

- ✿ *The features that define ``cat'' are local in the picture: whiskers, tail, paws ...: **locality***
- ✿ *Cats can be anywhere in the image: **translational invariance***
- ✿ *Relative position of features must be respected (eg whiskers and tail should appear in opposite sides of ``cat''): **rotational invariance***

Learning with symmetry

Like physical systems, many datasets and supervised learning tasks also possess additional **symmetries and structure** what can (and should) be exploited



e.g. we want to train a classifier to identify pictures of cats. What **high-level features** must one learn first?

- ✿ *The features that define ``cat'' are local in the picture: whiskers, tail, paws ...: **locality***
- ✿ *Cats can be anywhere in the image: **translational invariance***
- ✿ *Relative position of features must be respected (eg whiskers and tail should appear in opposite sides of ``cat''): **rotational invariance***

Our classifier should exhibit all these high-level features

Learning with symmetry

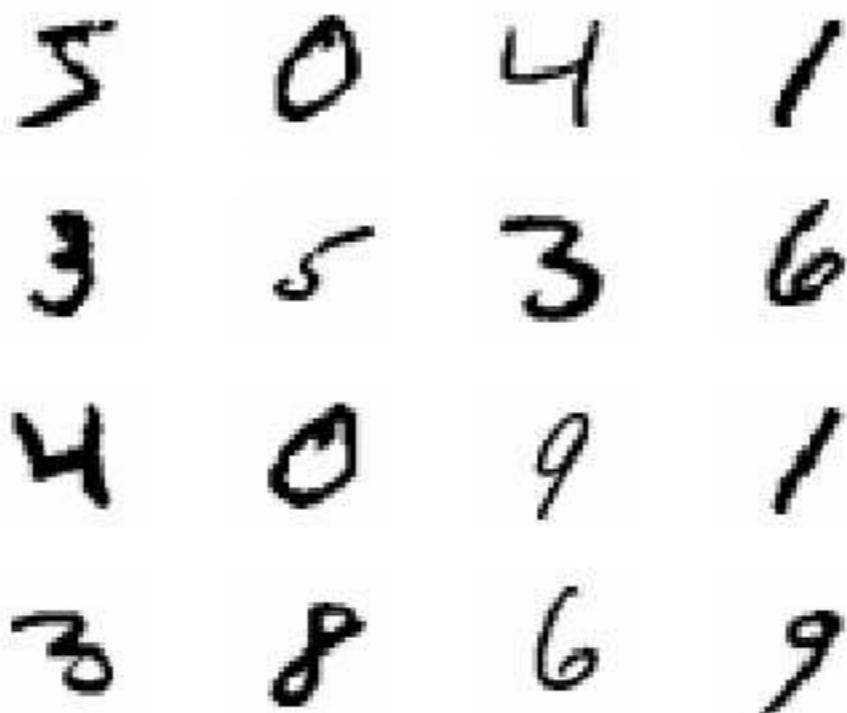
our goal is to create models which are **invariant** wrt certain transformations of the inputs

CNNs hard-code these invariance properties into the **structure of the network**

extensively used for applications in **pattern recognition**

e.g. classify handwritten digits

*Inputs: set of pixel intensity
values of each image*



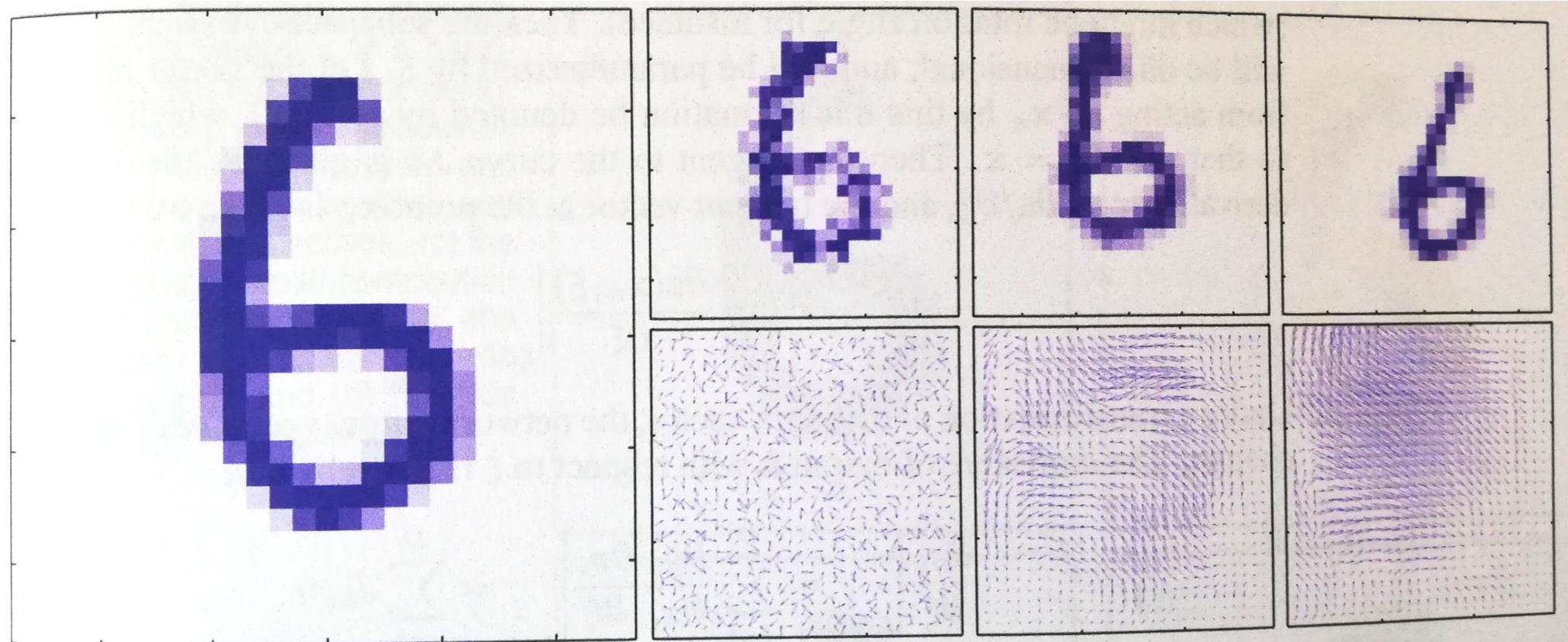
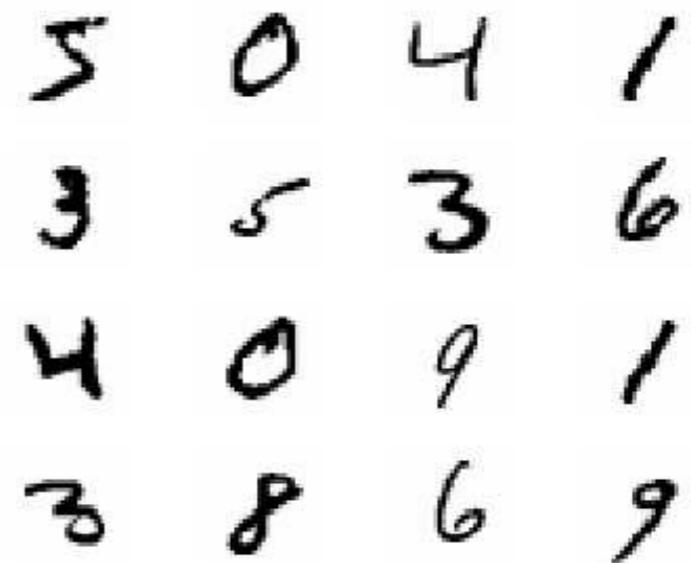
*Output: posterior
probability distribution
over the 10 digit classes*

what kind of **symmetries** must we built-in in our ML classifier model?

Learning with symmetry

what kind of **symmetries** must we built-in in our ML classifier model?

- Invariance under **translations**
- Invariance under **scaling**
- Invariance under **small rotations**
- Invariance under **smearing**
- Invariance under **elastic deformations**



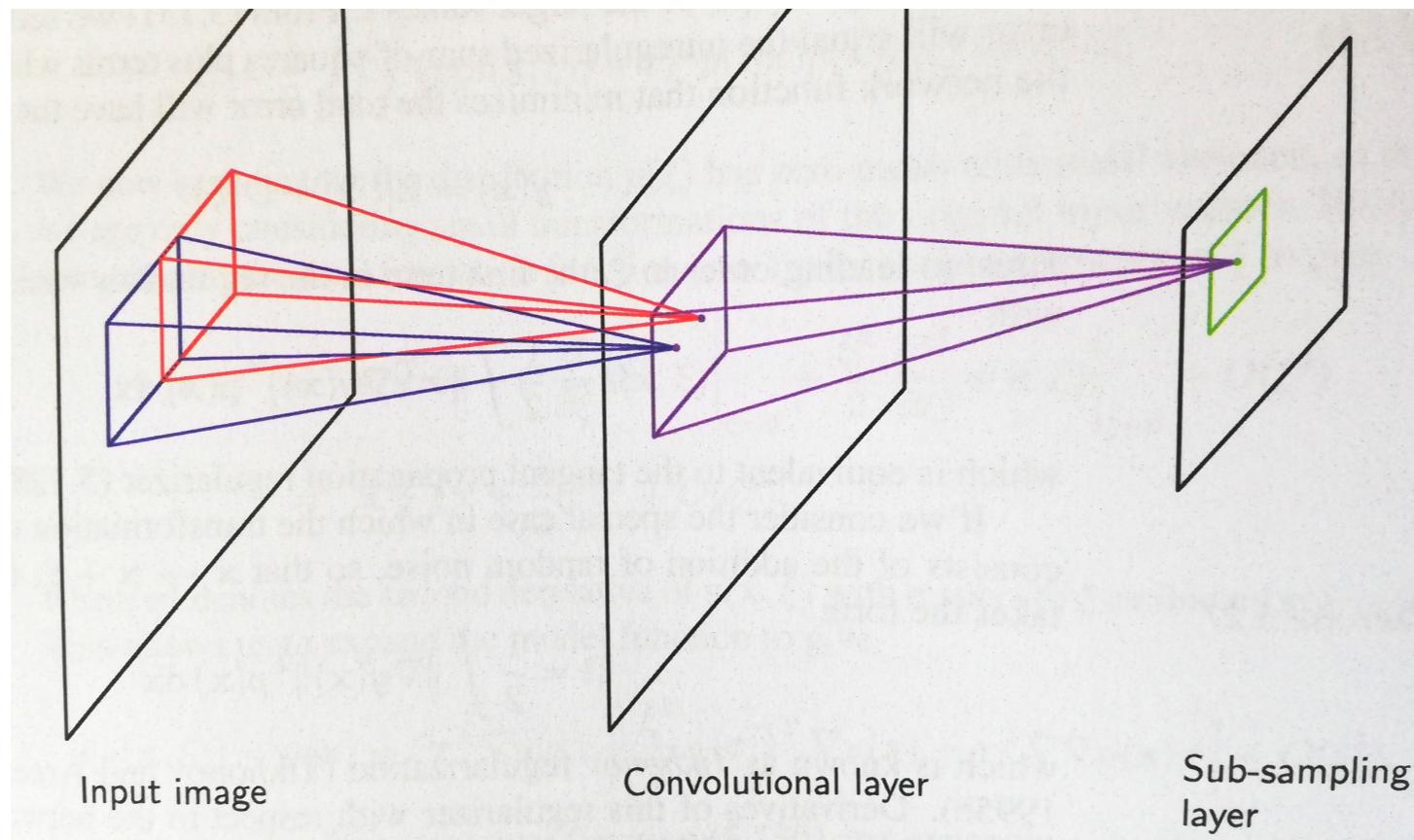
Convolutional Neural Networks

the simplest approach would be to input the images to a **fully connected NN** (see lect 4) which given enough training data (and time) would **learn the symmetries by example**

however this way a crucial property is ignored: **nearby pixels are strongly correlated** we should aim instead first to **identify local features** that depend on small subregions

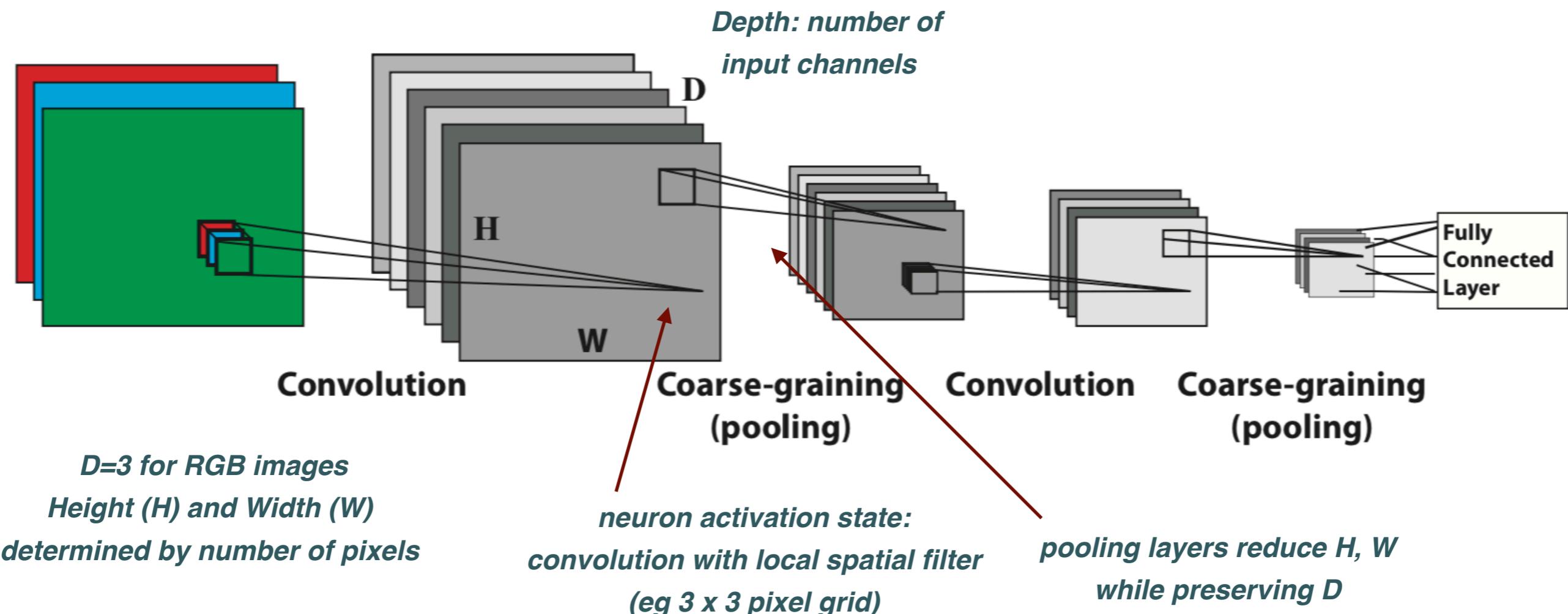
afterwards such local features can be combined into **higher-level ones**

Convolutional Neural Networks (CNNs) are architectures that take **advantage of this additional high-level structures** that all-to-all coupled networks fail to exploit



Convolutional Neural Networks

A CNN is a translationally invariant neural network that respects locality of the input data

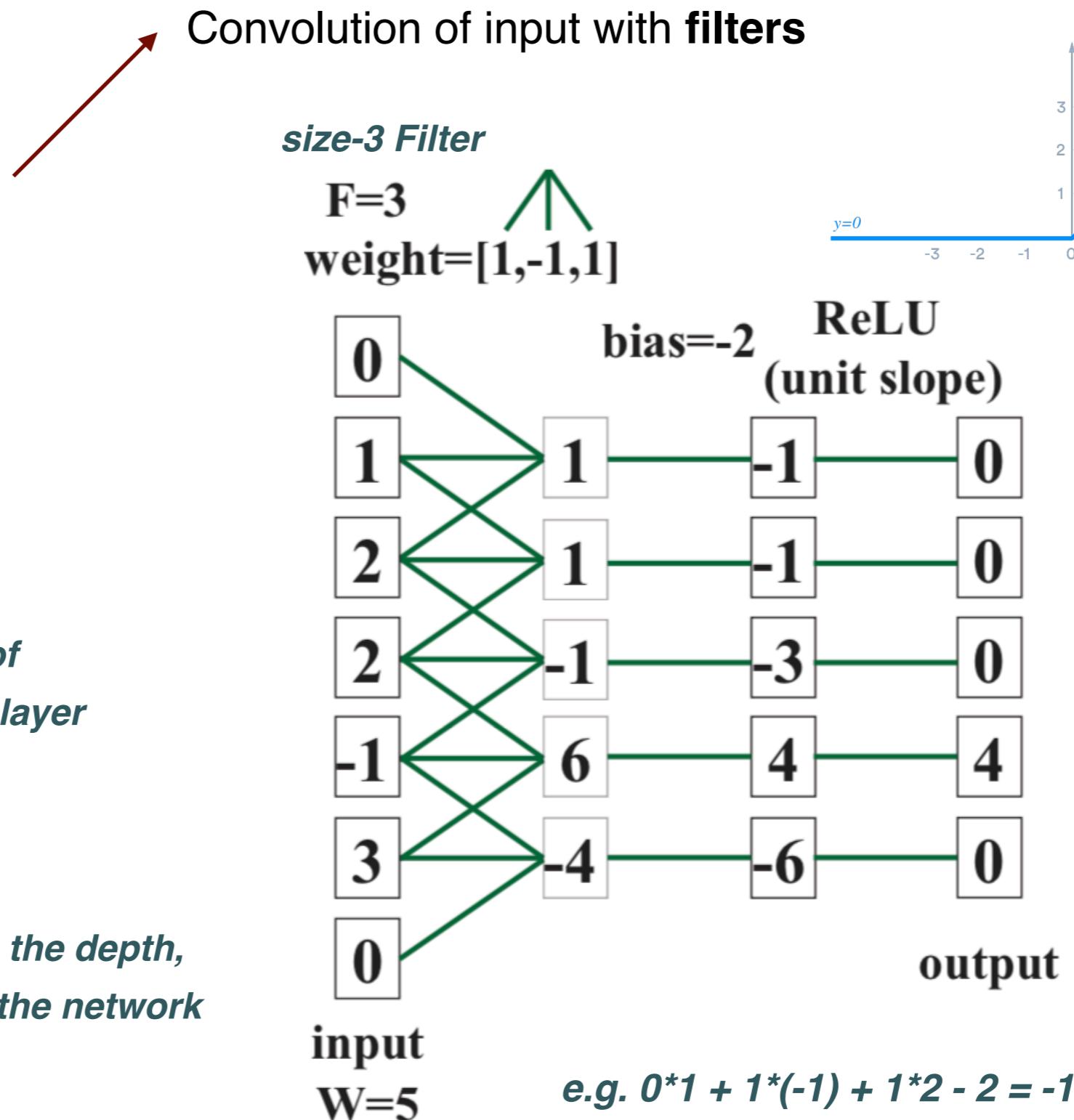


Convolutional Neural Networks

CNNs are composed by
two kinds of layers

*example of
convolutional layer*

*note that convolution changes the depth,
but not the height and width of the network*



Convolutional Neural Networks

CNNs are composed by
two kinds of layers

Convolution layer of input with **filters**

Pooling layer that coarse-grains the input while
maintaining locality and spatial structure

e.g. **MaxPool**, the spatial dimensions are coarse-grained by replacing a small region by single neuron whose output is maximum value of the output in the region

in average pooling, one averages over output in region

12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

max pooling

20	30
112	37

average pooling

13	8
79	20

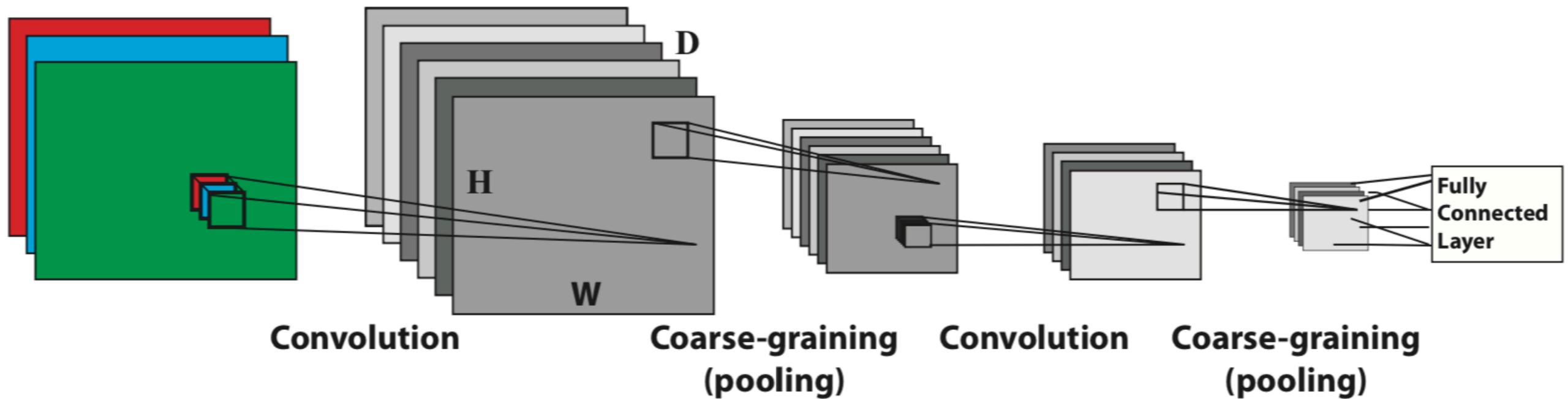
Convolutional Neural Networks

CNNs are composed by
two kinds of layers

Convolution layer of input with **filters**

Pooling layer that coarse-grains the input while
maintaining locality and spatial structure

the convolution and max-pool layers are followed by an **all-to-all connected layer and a high-level classifier**, so that one can train CNNs using the standard backpropagation algorithm



Convolutional Neural Networks

CNNs are composed by two kinds of layers

The diagram illustrates the two types of layers in a CNN. A central text box states: "CNNs are composed by two kinds of layers". Two red arrows point from this text to two separate definitions: "Convolution layer of input with filters" and "Pooling layer that coarse-grains the input while maintaining locality and spatial structure".

Convolution layer of input with **filters**

Pooling layer that coarse-grains the input while maintaining locality and spatial structure

the convolution and max-pool layers are followed by an **all-to-all connected layer and a high-level classifier**, so that one can train CNNs using the standard backpropagation algorithm

note that only problems characterised by a **spatial locality** are amenable to CNNs:
for example the 2D Ising dataset can be studied with CNNs, but not the SUSY dataset

How CNNs work

Bayesian Neural Networks

A probability distribution for NNs

up to here when considering neural networks our goal was to use maximum likelihood to determine the **best values of its model parameters**

for many applications we'd like to know the **full posterior distribution** of the model

consider the problem of predicting a single continuous target variable t from vector of inputs \mathbf{x} by means of a multi-layer feed-forward neural network. Assume the **conditional probability** is

$$p(t | \mathbf{x}, \theta, \beta) = \mathcal{N}(t | y(\mathbf{x}, \theta), \beta^{-1})$$

conditional probability *GaussianDist* *mean: NN output* *variance*

and we also assume a prior distribution over the model parameters to be Gaussian

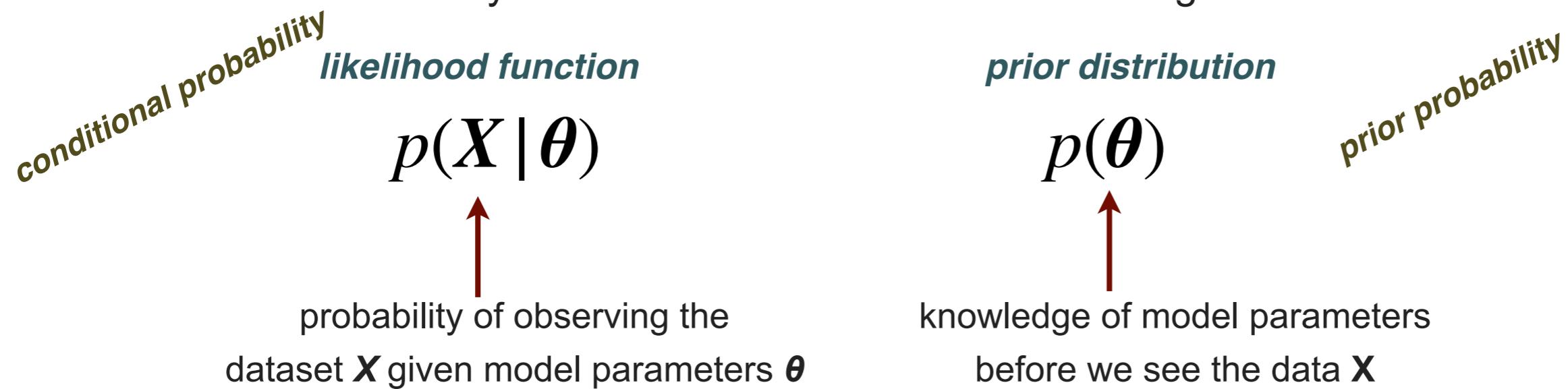
$$\textit{prior probability} \longrightarrow p(\theta) = \mathcal{N}(\theta | \mathbf{0}, \alpha^{-1} \mathbf{I})$$

the key to determine the probability distribution of our ML model is **Bayes' Theorem**

Bayesian Inference

Bayesian inference a method of statistical inference in which **Bayes' theorem** is used to **update the probability for an hypothesis** as more information becomes available

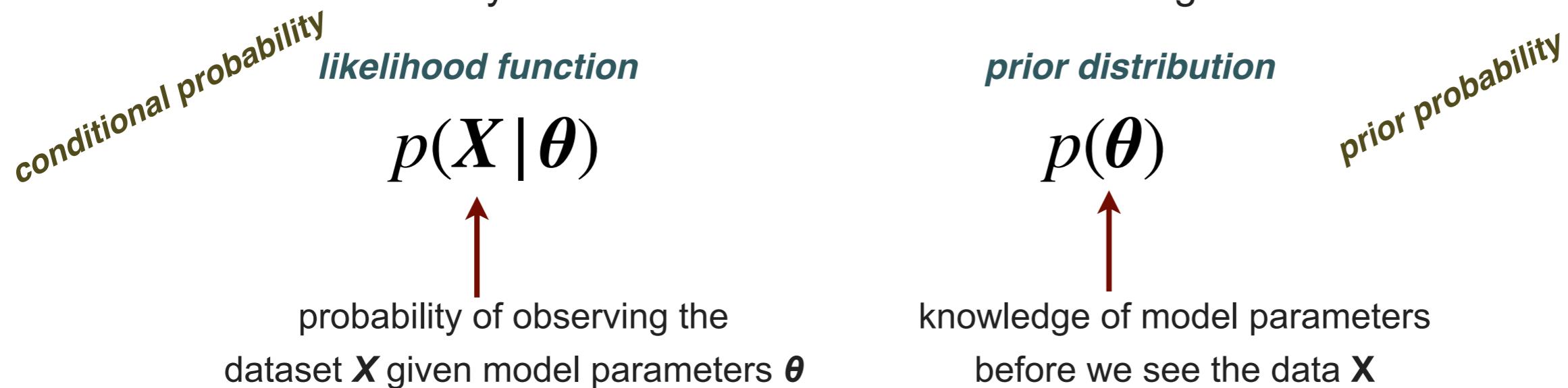
in Bayesian statistics there are two main ingredients:



Bayesian Inference

Bayesian inference a method of statistical inference in which **Bayes' theorem** is used to **update the probability for an hypothesis** as more information becomes available

in Bayesian statistics there are two main ingredients:



which are used to compute the **posterior distribution** using **Bayes' Theorem**

The diagram shows the Bayes' Theorem formula:
$$p(\theta | X) = \frac{p(X | \theta)p(\theta)}{\int d\theta' p(X | \theta')p(\theta')}$$
 A red arrow points up from the text "probability of the model parameters θ after observing the dataset X " to the term $p(\theta | X)$. Another red arrow points up from the text "posterior probability" to the same term. A third red arrow points up from the text "likelihood function" to the term $p(X | \theta)$. A fourth red arrow points up from the text "prior probability" to the term $p(\theta)$.

Bayesian Inference for Neural Nets

back to our model, given N observations, the **likelihood** will just the product of the (independent) conditional probabilities

$$p(\mathcal{D} | \theta, \beta) = \prod_{n=1}^N \mathcal{N}(t_n | y(x_n, \theta), \beta^{-1})$$

dataset 

and using Bayes' Theorem, the **posterior probability for the NN parameters** is

$$p(\omega | \mathcal{D}, \alpha, \beta) \propto p(\mathcal{D} | \theta, \beta) p(\omega | \alpha)$$

which will be non-gaussian since the neural-net output depends non-linearly on its params

one can construct a Gaussian approximation to the posterior based on the **Laplace approximation** once we have found a local maximum

$$\ln p(\omega | \mathcal{D}, \alpha, \beta) = -\frac{\alpha}{2}\theta^T\theta - \frac{\beta}{2} \sum_{n=1}^N (y(x_n, \theta) - t_n)^2 + \text{const}$$

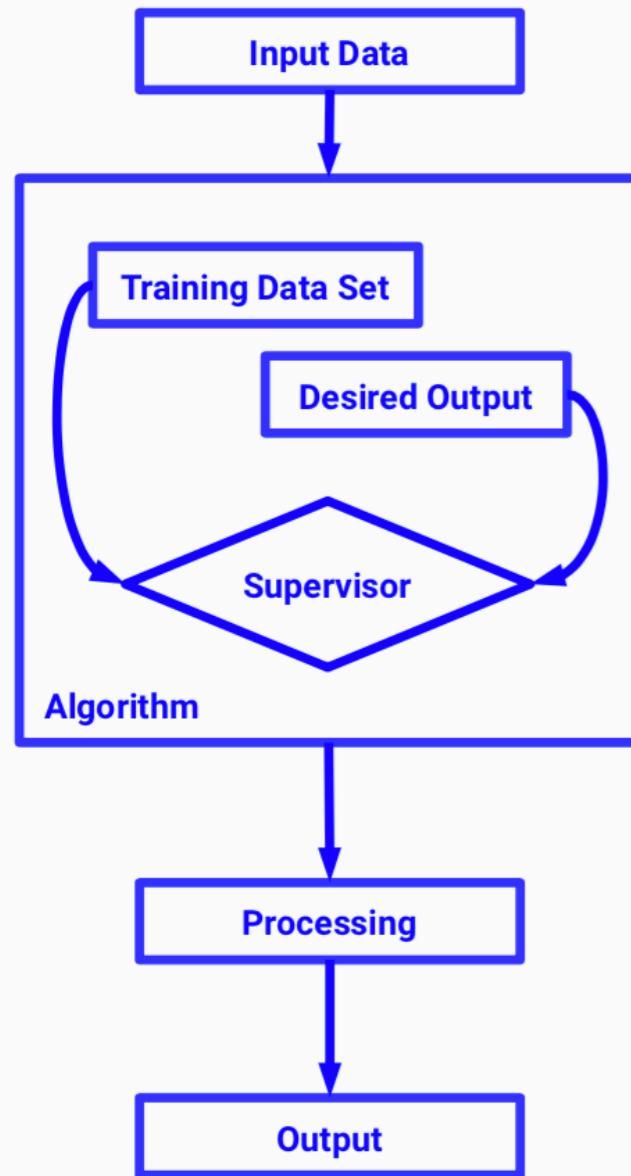

log-likelihood

*for fixed α, β one can find a local minimum
with standard algorithms such as SGD with backpropagation*

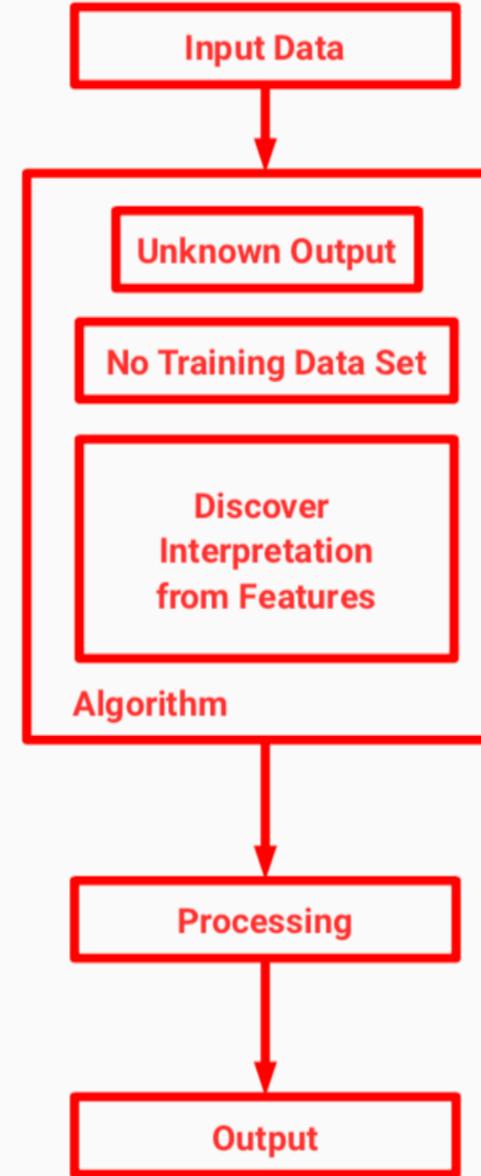
Reinforcement Learning

Supervised vs Unsupervised Learning

Supervised learning



Unsupervised learning



Reinforcement learning



Reinforcement Learning

So far we have considered **two main paradigms** in Machine Learning problems

Supervised Learning: starting from a training dataset with **labelled examples**, $\{x_i, y_i\}_{i=1,N}$, produce a **model $f(x)$** that predicts and generalises the info in the training sample. The labels y_i can be continuous (underlying law is function) or discrete (classification)

Unsupervised Learning: starting from a training dataset with **unlabelled examples**, $\{x_i\}_{i=1,N}$, produce a **model** that takes a sample as input and as output produces the solution of a practical problem, such as **clustering**, **dimensional reduction**, or **outlier detection**

Reinforcement Learning

So far we have considered **two main paradigms** in Machine Learning problems

Supervised Learning: starting from a training dataset with **labelled examples**, $\{x_i, y_i\}_{i=1,N}$, produce a **model $f(x)$** that predicts and generalises the info in the training sample. The labels y_i can be continuous (underlying law is function) or discrete (classification)

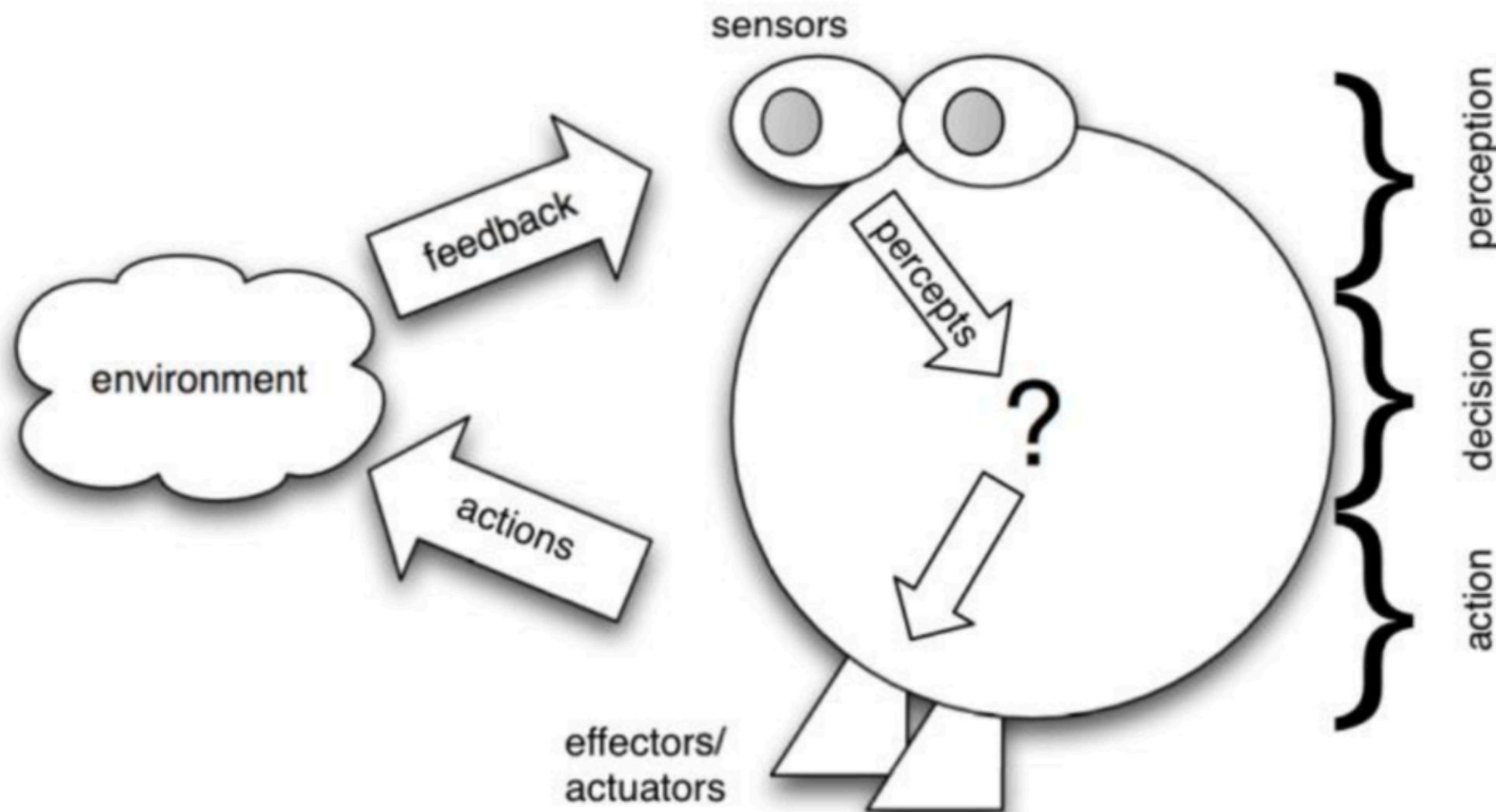
Unsupervised Learning: starting from a training dataset with **unlabelled examples**, $\{x_i\}_{i=1,N}$, produce a **model** that takes a sample as input and as output produces the solution of a practical problem, such as **clustering**, **dimensional reduction**, or **outlier detection**

now we want to discuss a **third ML paradigm**

Reinforcement Learning: given a complex task in a complex environment (dynamic, non deterministic, only partly accessible) train an **agent** that carry out **autonomous action** in this environment and complete the requested task

Agents in Reinforcement Learning

In the context of **Reinforcement Learning**, an **agent** is a computer system capable **autonomous action** in some environment, in order to achieve its delegated goals



Agents in Reinforcement Learning

In the context of **Reinforcement Learning**, an **agent** is a computer system capable **autonomous action** in some environment, in order to achieve its delegated goals

Can you think of trivial ``agents''?

Agents in Reinforcement Learning

In the context of **Reinforcement Learning**, an **agent** is a computer system capable **autonomous action** in some environment, in order to achieve its delegated goals



trivial agents: thermostat, e-mail daemons, alarms,

Agents in Reinforcement Learning

In the context of **Reinforcement Learning**, an **agent** is a computer system capable **autonomous action** in some environment, in order to achieve its delegated goals

non-trivial agents should exhibit the following properties:

- ✿ **Reactive:** interact with environment and react its changes
- ✿ **Proactive:** recognise opportunities and take initiative
- ✿ **Social:** cooperate with other agents (and humans!) via cooperation, negotiation, coordination
- ✿ **Rational:** the agent will always act to fulfil its goals
- ✿ **Adaptability:** the agent is able to improve its performance over time

Agents in Reinforcement Learning

Environments in RL can exhibit the following features:

- ➊ **Accessible or Inaccessible:** can the agent obtain updated and accurate information about the state of the environment?
- ➋ **Deterministic or non-deterministic :** has each action that the agent perform always associated the same effect?
- ➌ **Static vs dynamics:** is the environment stable expect for the action of the agent?
- ➍ **Discrete vs continuous:** are there a finite or infinite number of actions possible?

A Reinforcement Learning system

The ultimate goal of **Reinforcement Learning** is to

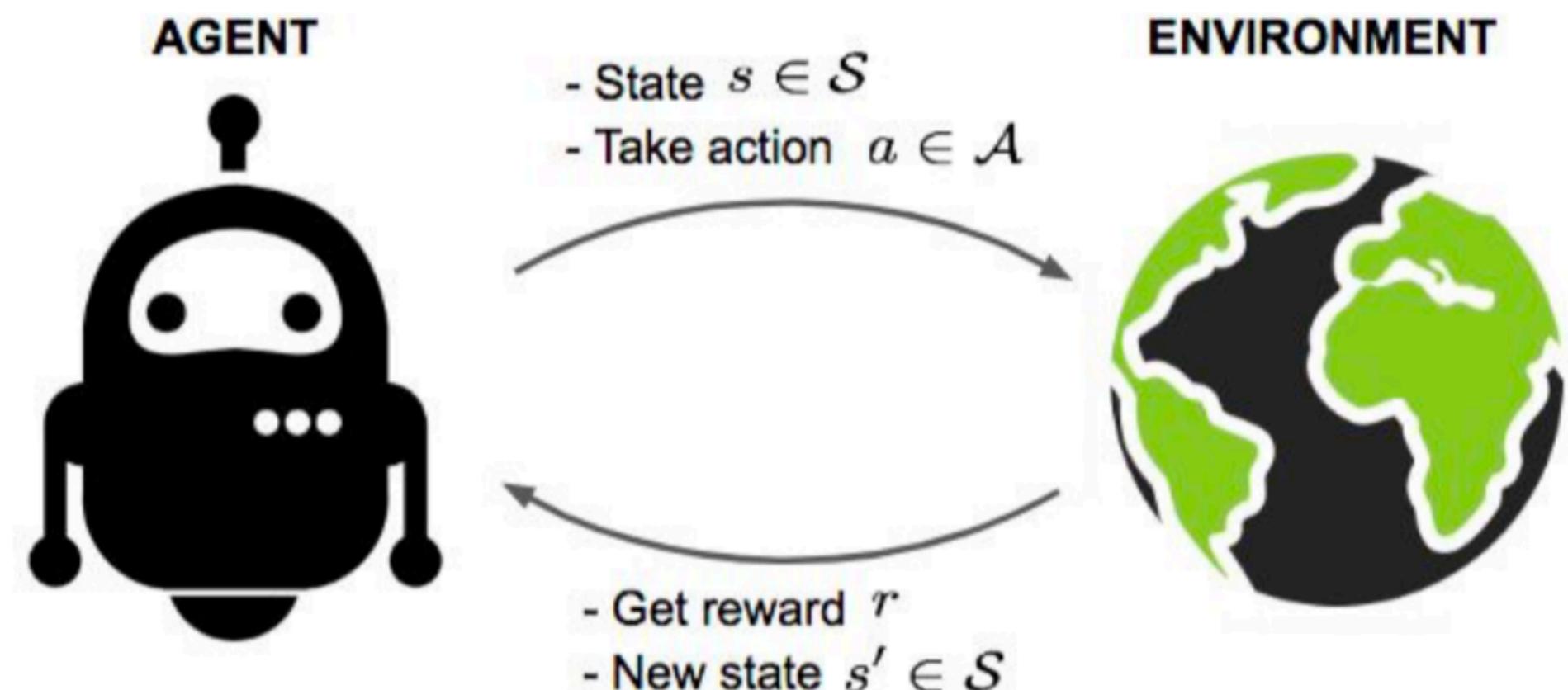
design an agent that **performs complex tasks** and **takes autonomous action** to fulfil its design goals, in an environment that is: partly inaccessible, non-deterministic, non-episodic, dynamic and continuous (*i.e.* the real world!).

- The agent receives the state of the environment as a **vector of features** (inputs)
- The agent can execute actions in every state, with different actions bringing **different rewards**
- Goal: **learn a policy**, *i.e.* a function that maps the features of an state vector to an optimal action to be taken in that stage
- An action is optimal if it **maximizes the expected average reward**
- In RL **decision making is sequential and the goal is long-term** (*i.e.* game playing, robotics, resource management, ...)

Agents in Reinforcement Learning

The ultimate goal of **Reinforcement Learning** is to

design an agent that **performs complex tasks** and **takes autonomous action** to fulfil its design goals, in an environment that is: partly inaccessible, non-deterministic, non-episodic, dynamic and continuous (*i.e.* the real world!).

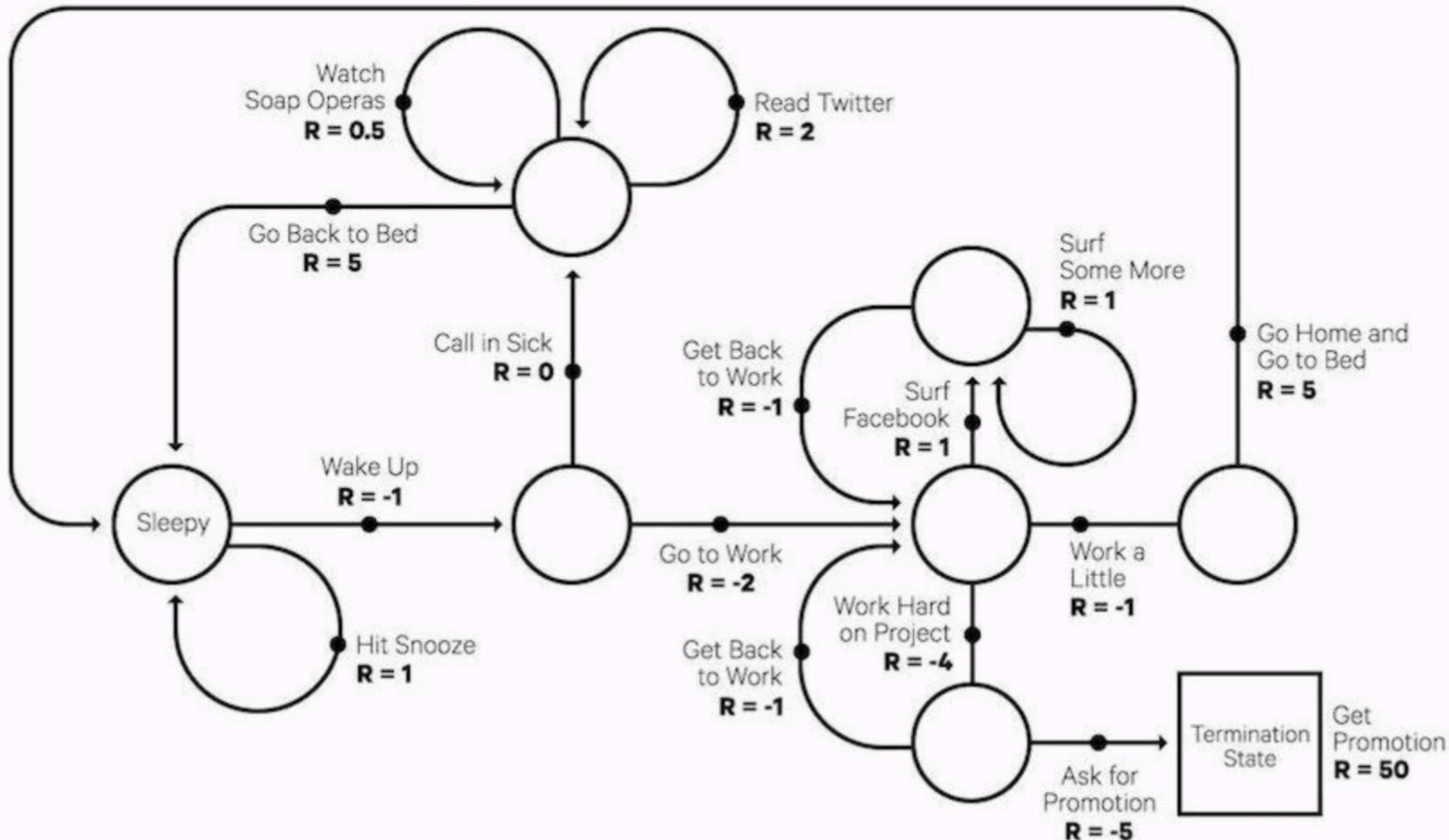


A Reinforcement Learning system

use **Reinforcement Learning** is to determine the actions that will get us a promotion at work!

the goal of RL is **maximise the total reward**: need to explore all possible options to determine the best policy for each action that it might need to carry

A Reinforcement Learning system

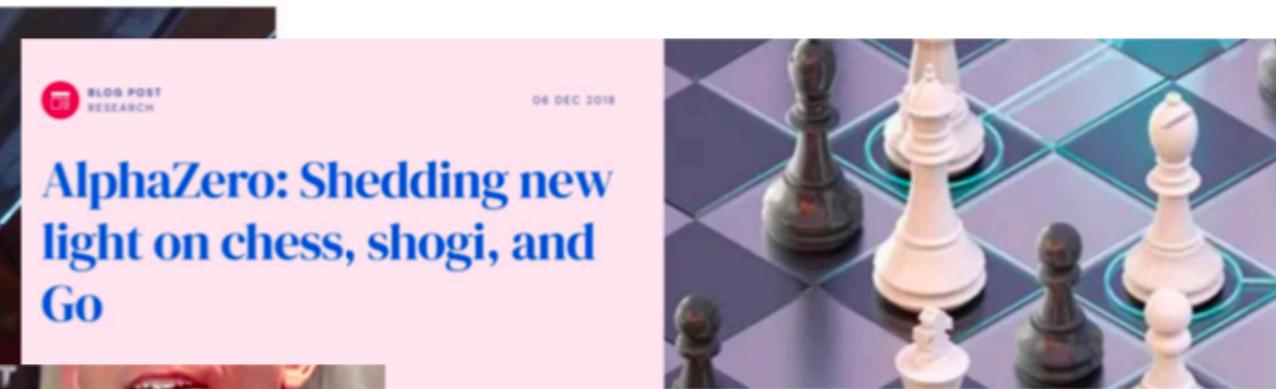


tl

explore all possible options to determine the best policy
for each action that it might need to carry

Reinforcement Learning for Games

AlphaGo: using machine learning to master the ancient game of Go



In late 2017 we [introduced AlphaZero](#), a single system that taught itself from scratch how to master the games of chess, [shogi](#) (Japanese chess), and [Go](#), beating a world-champion program in each case. We were excited by the preliminary results and thrilled to see the response from members of the chess community, who saw in AlphaZero's games a ground-breaking, highly dynamic and "[unconventional](#)" style of play that differed from any chess playing engine that came before it.

Reinforcement Learning for Games

DeepMind AlphaStar: AI breakthrough or pushing the limits of reinforcement learning?

By **Ben Dickson** - November 4, 2019

 Me gusta 52

 Facebook

 Twitter

 Reddit

 LinkedIn

4 min read



DeepMind's AI program AlphaStar managed to defeat 99.8 percent of StarCraft II players.



Reinforcement Learning for Games



REINFORCEMENT LEARNING DEMO

Q-learning

Q-learning is a **model-free reinforcement learning algorithm**, which aims to learn a **policy** about what actions should the agent carry out for different circumstances

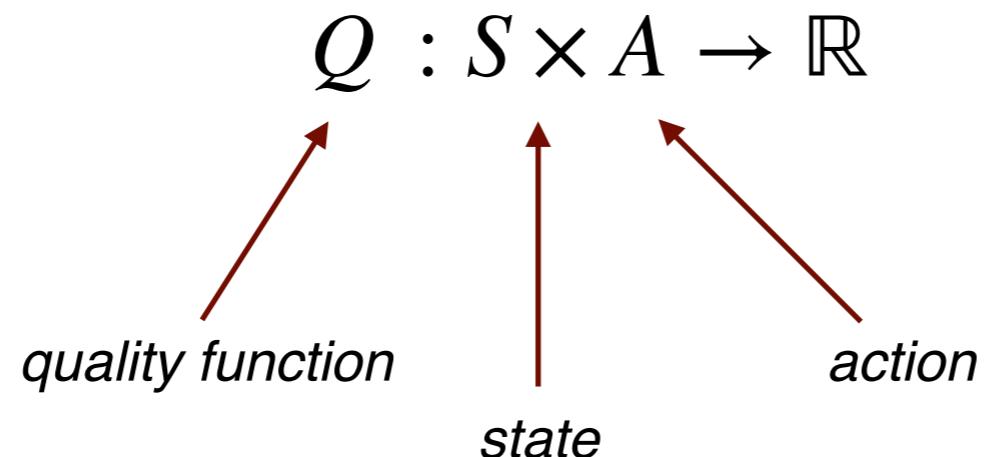
no model of the environment need: the agent learns to maximise its future reward by repeatedly interacting with the environment

In Q-learning, the weight for a step into the future is calculated by the **discount factor**

$$0 \leq \gamma^{\Delta t} \leq 1$$

earlier rewards valued higher than later ones

analog of cost function in Supervised Learning is the **quality of state-action combination**



the goal of Q-learning is to determine the actions that the agent should take for each state in order to **maximise the total reward**

Q-learning

Schematically, at **each iteration of the Q-learning algorithm** the following steps take place:

- The agent selects an **action a_t**
- As a consequence of this action, the agent observes a **reward r_t**
- The agent then enters into a **new state s_t**
- The quality function (cost function) **Q** is updated

$$Q^{\text{new}}(s_t, a_t) \leftarrow (1 - \alpha) \times Q^{\text{old}}(s_t, a_t) + \alpha (r_t + \gamma \times \max_a Q(s_{t+1}, a))$$

The diagram illustrates the Q-learning update rule. It shows the formula $Q^{\text{new}}(s_t, a_t) \leftarrow (1 - \alpha) \times Q^{\text{old}}(s_t, a_t) + \alpha (r_t + \gamma \times \max_a Q(s_{t+1}, a))$. Four red arrows point to specific terms in the formula: one to $(1 - \alpha)$ labeled "learning rate", one to γ labeled "discount factor", two to $\max_a Q(s_{t+1}, a)$ labeled "(estimate of) future optimal value", and one to r_t labeled "reward".

After training, the agent has a policy **Q** which tells it how to act for each circumstance

Deep Q-Networks

the model for the **policy function** (which action to take as function of the state) can be parametrised using Deep Neural Networks, in this case called Q-Networks

