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

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

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 $\boldsymbol{x}$ by means of a multi-layer feed-forward neural network. Assume the **conditional probability** is

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

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

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

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

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:

conditional probability

likelihood function

$$p(\boldsymbol{X} | \boldsymbol{\theta})$$

probability of observing the dataset ***X*** given model parameters ***θ***

prior distribution

prior probability

$$p(\boldsymbol{\theta})$$

knowledge of model parameters before we see the data **X**

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:

conditional probability

likelihood function

$$p(\boldsymbol{X}\,|\,\boldsymbol{\theta})$$

probability of observing the dataset ***X*** given model parameters ***θ***

prior distribution

prior probability

$$p(\boldsymbol{\theta})$$

knowledge of model parameters before we see the data **X**

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

posterior probability

$$p(\boldsymbol{\theta}\,|\,\boldsymbol{X}) = \frac{p(\boldsymbol{X}\,|\,\boldsymbol{\theta})p(\boldsymbol{\theta})}{\int d\boldsymbol{\theta}' p(\boldsymbol{X}\,|\,\boldsymbol{\theta}')p(\boldsymbol{\theta}')}$$

probability of the model parameters ***θ*** after observing the dataset ***X***

Bayesian Inference for Neural Nets

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

$$p\left(\mathscr{D}\,|\,\boldsymbol{\theta},\beta\right) = \prod_{n=1}^{N} \mathscr{N}\left(t_n\,|\,y(\boldsymbol{x}_n,\boldsymbol{\theta}),\beta^{-1}\right)$$

dataset

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

$$p\left(\omega\,|\,\mathscr{D},\alpha,\beta\right) \propto p\left(\mathscr{D}\,|\,\boldsymbol{\theta},\beta\right) p\left(\boldsymbol{\omega}\,|\,\alpha\right)$$

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

$$\ln p\left(\omega\,|\,\mathscr{D},\alpha,\beta\right) = -\frac{\alpha}{2}\boldsymbol{\theta}^T\boldsymbol{\theta} - \frac{\beta}{2}\sum_{n=1}^{N}\left(y(\boldsymbol{x}_n,\boldsymbol{\theta}) - t_n\right)^2 + \text{const}$$

log-likelihood

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

Bayesian Inference for Neural Nets

having found a local maximum of the posterior distributions, we can construct its Gaussian approximation by means of the **Hessian matrix** (matrix of second derivatives)

$$\boldsymbol{A} = -\nabla^2 \ln p\left(\omega \,|\, \mathscr{D}, \alpha, \beta\right) = \alpha \boldsymbol{I} + \beta \boldsymbol{H}$$

$$H_{ij} = \frac{\partial^2}{\partial\theta_i \partial\theta_j}\left(\frac{1}{2}\sum_{n=1}^{N}\left(y(\boldsymbol{x}_n, \boldsymbol{\theta}) - t_n\right)^2\right)$$

and thus the **gaussian approximation to the posterior** is

$$p\left(\omega \,|\, \mathscr{D}, \alpha, \beta\right) \simeq q\left(\omega \,|\, \mathscr{D}, \alpha, \beta\right) = \mathscr{N}\left(\boldsymbol{\theta} \,|\, \boldsymbol{\theta}_{\mathrm{MAP}}, \boldsymbol{A}^{-1}\right)$$

full posterior ***gaussian approx*** ***local max of posterior***

finally we are able to evaluate the sought-for **predictive distribution** by marginalising

$$p(t \,|\, \boldsymbol{x}, \mathscr{D}) = \int p(t \,|\, \boldsymbol{x}, \boldsymbol{\theta}) q\left(\omega \,|\, \mathscr{D}, \alpha, \beta\right)$$

probability to observe an output t given 1) a new input vector x and 2) the training dataset D

conditional probability, depends on NN function output

probability dist of the model parameters given training dataset

Bayesian Inference for Neural Nets

now we can evaluate **the full probability distribution** associated to our ML model!

$$p(t\,|\,\boldsymbol{x}, \mathscr{D}) = \int p(t\,|\,\boldsymbol{x}, \boldsymbol{\theta}) q\left(\omega\,|\,\mathscr{D}, \alpha, \beta\right)$$

unfortunately the integral is still very complicated given the non-linear nature of the NN output

we can simply this expression with the assumption that the (Gaussian) **posterior distribution varies slowly** as compared to the NN output

$$p(t\,|\,\boldsymbol{x}, \mathscr{D}, \alpha, \beta) = \mathscr{N}\left(t\,|\,y(\boldsymbol{x}, \boldsymbol{\theta}_{\mathrm{MAP}}), \sigma^2(\boldsymbol{x})\right)$$

where the input dependent variance of this gaussian distribution is given by

$$\sigma^2(\boldsymbol{x}) = \beta^{-1} + \boldsymbol{g}^T \boldsymbol{A}^{-1} \boldsymbol{g}$$

$$y\left(\boldsymbol{x}, \boldsymbol{\theta}\right) \simeq y\left(\boldsymbol{x}, \boldsymbol{\theta}_{\mathrm{MAP}}\right) + \boldsymbol{g}^T(\boldsymbol{\theta} - \boldsymbol{\theta}_{\mathrm{MAP}})$$

finally the hyperparameters of the model can be determined by means of the **evidence framework**

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 shoaled 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 shoaled appear in opposite sides of ``cat"): **rotational invariance***

O***ur classifier should exhibit all these high-level features***

Learning with symmetry

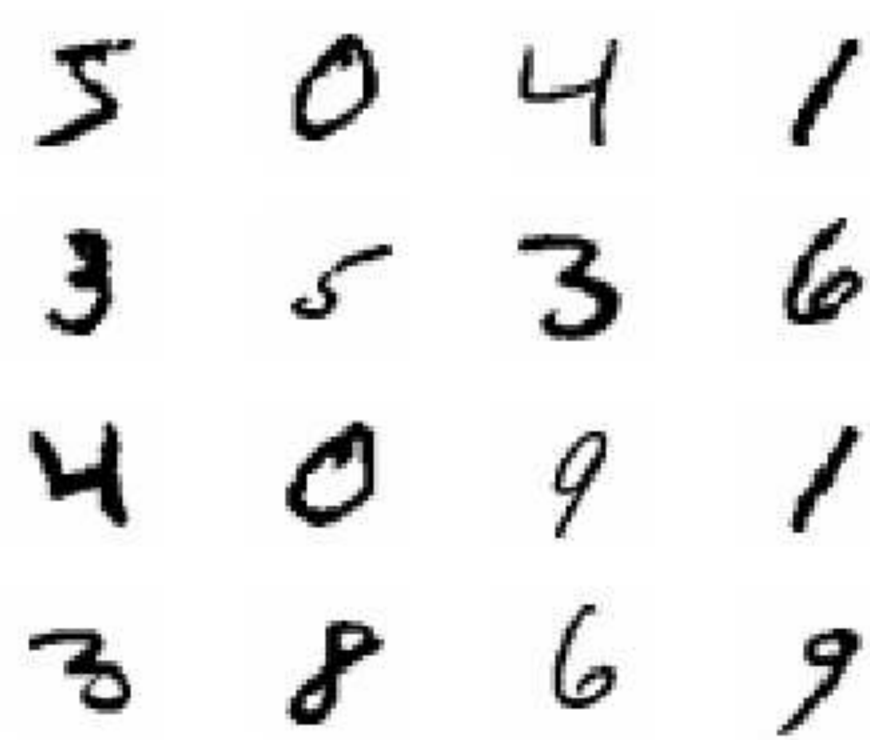
our goal is to create models which are **invariant** wrt certain transformations of the inputs

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

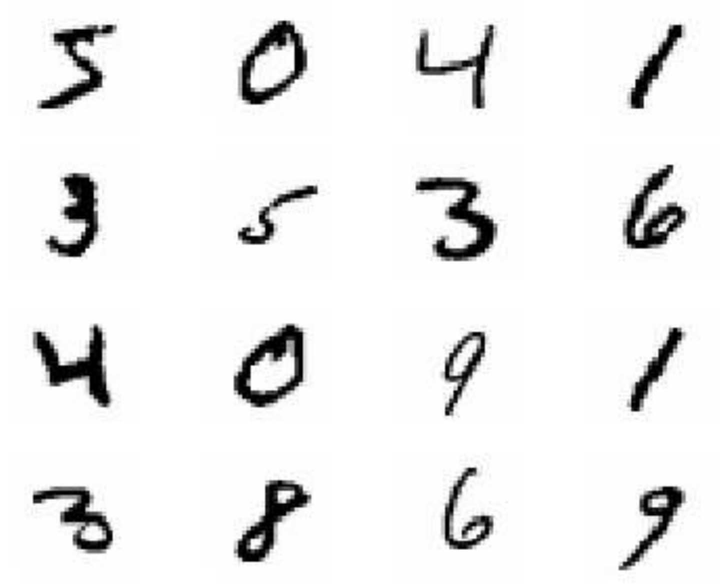


Output: posterior probability distribution over the 10 digit classes

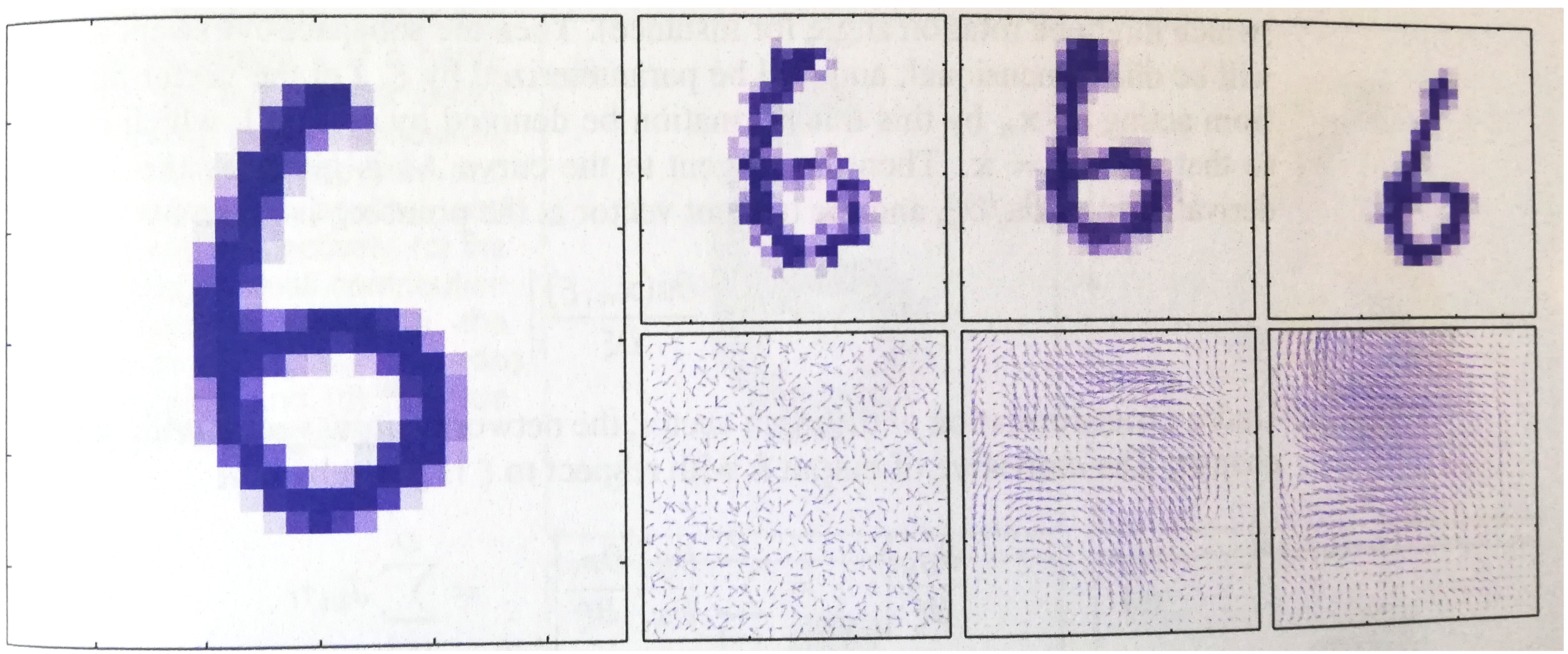
what kind of **symmetries** must we built-in in our ML classifier model?

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

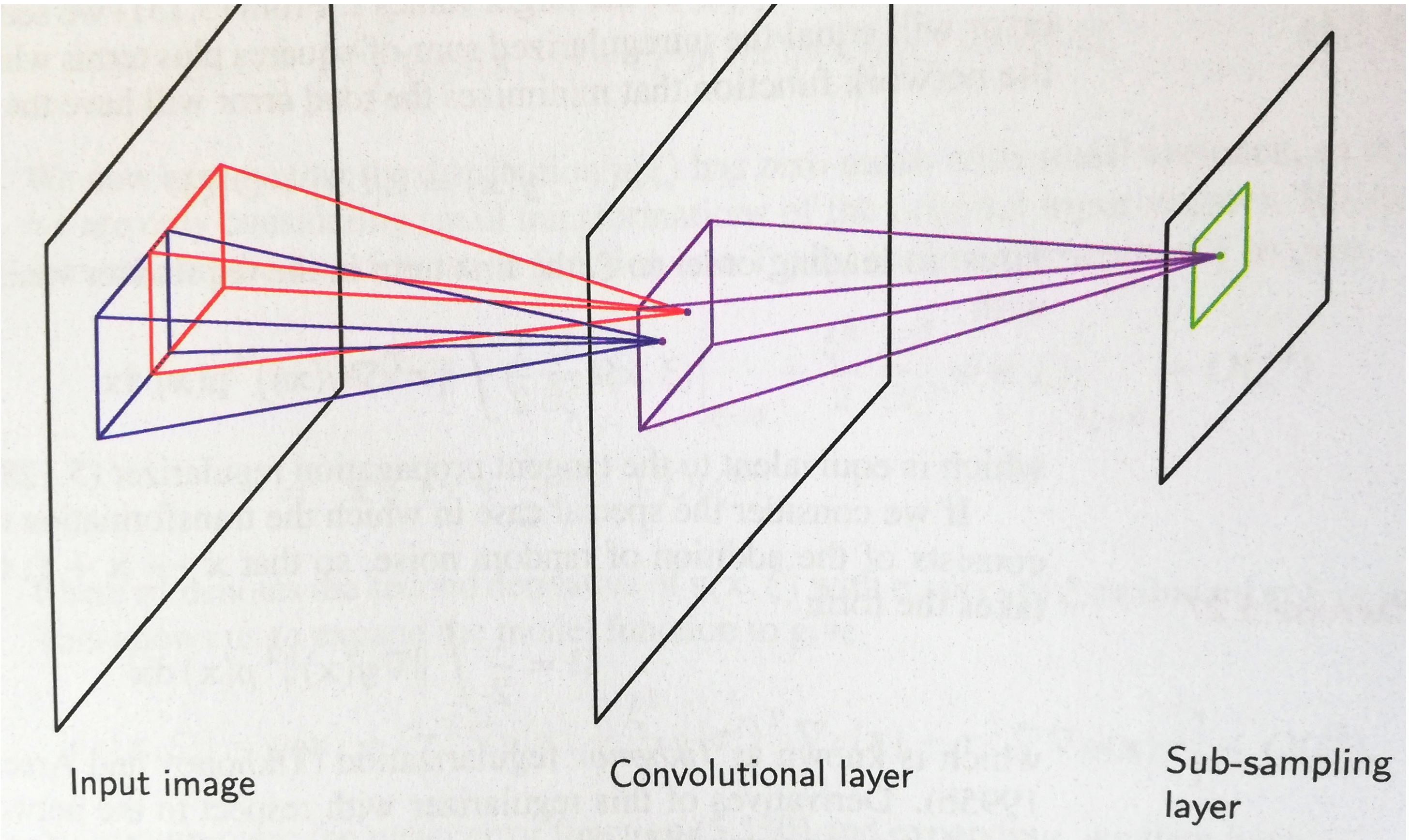
the simplest approach would be to input the images to a **fully connected NN** (see lect 4) which given enough training data (and time) would **learn the symmetries by example**

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**

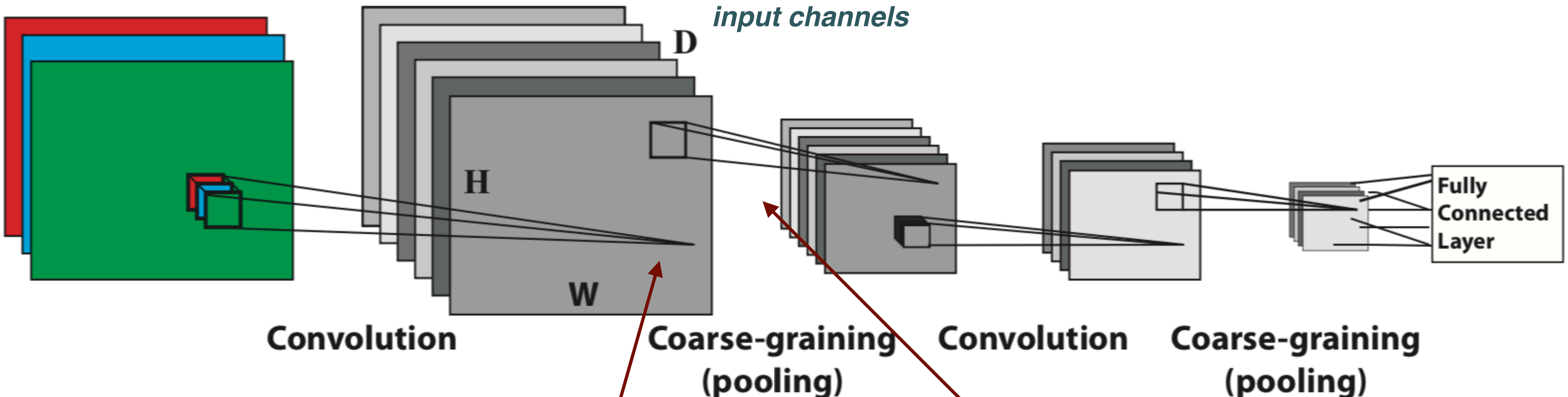
Convolutional Neural Networks (CNNs) are architectures that take **advantage of this additional high-level structures** that all-to-all coupled networks fail to exploit



Convolutional Neural Networks

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

Depth: number of input channels



D=3 for RGB images
Height (H) and Width (W)
determined by number of pixels

neuron activation state:
convolution with local spatial filter
(eg 3 x 3 pixel grid)

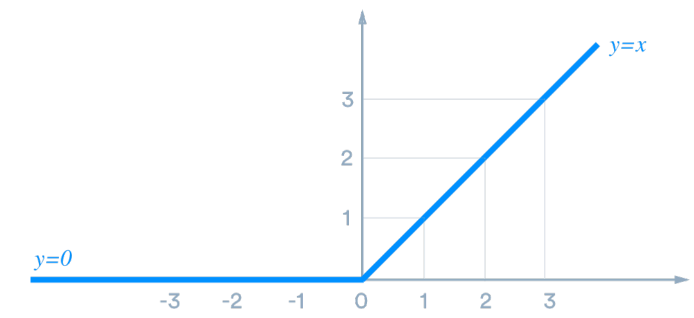
pooling layers reduce H, W
while preserving D

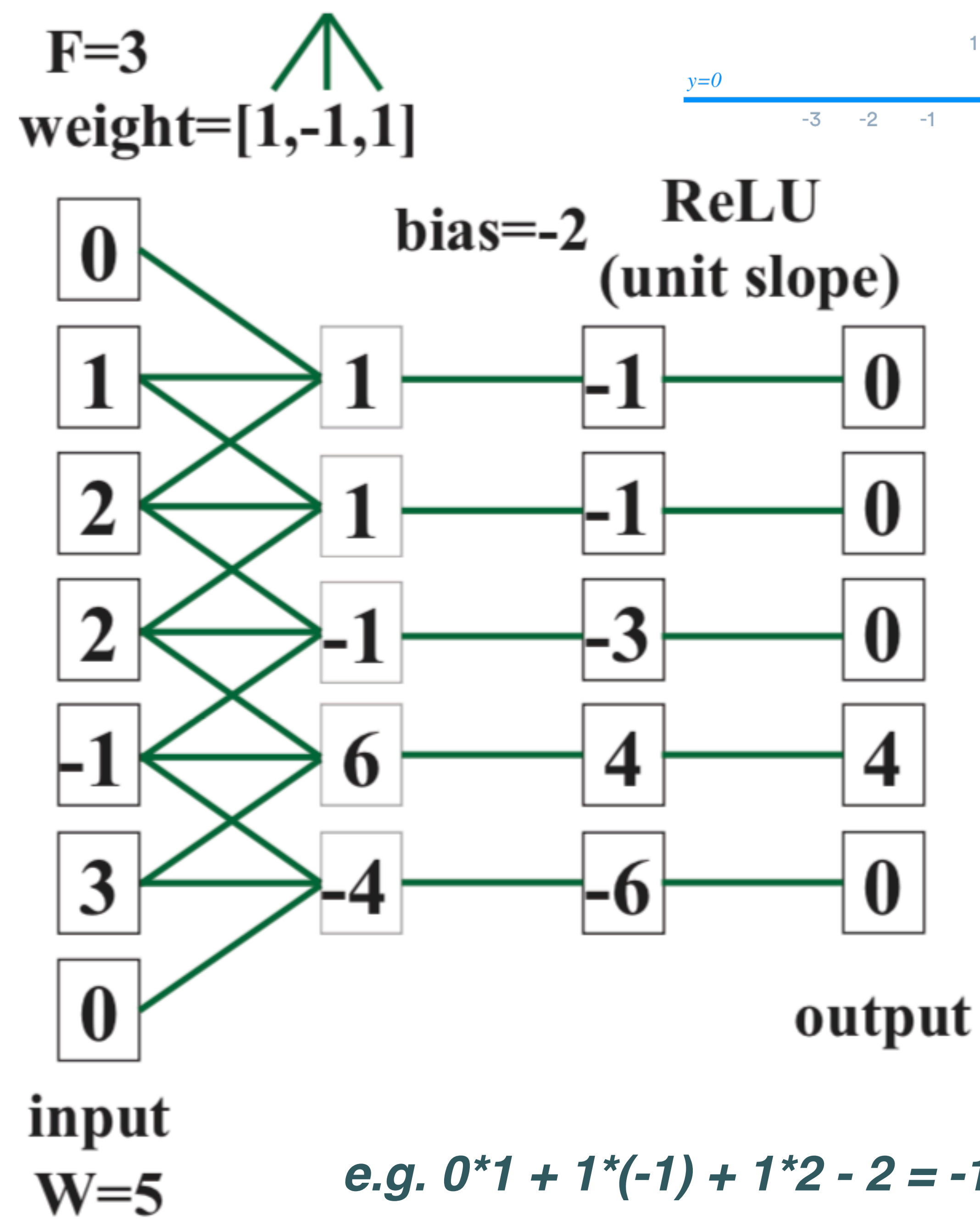
Convolutional Neural Networks

Convolution of input with **filters**

CNNs are composed by
two kinds of layers

size-3 Filter





example of
convolutional layer

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

*e.g. 0*1 + 1*(-1) + 1*2 - 2 = -1*

Convolutional Neural Networks

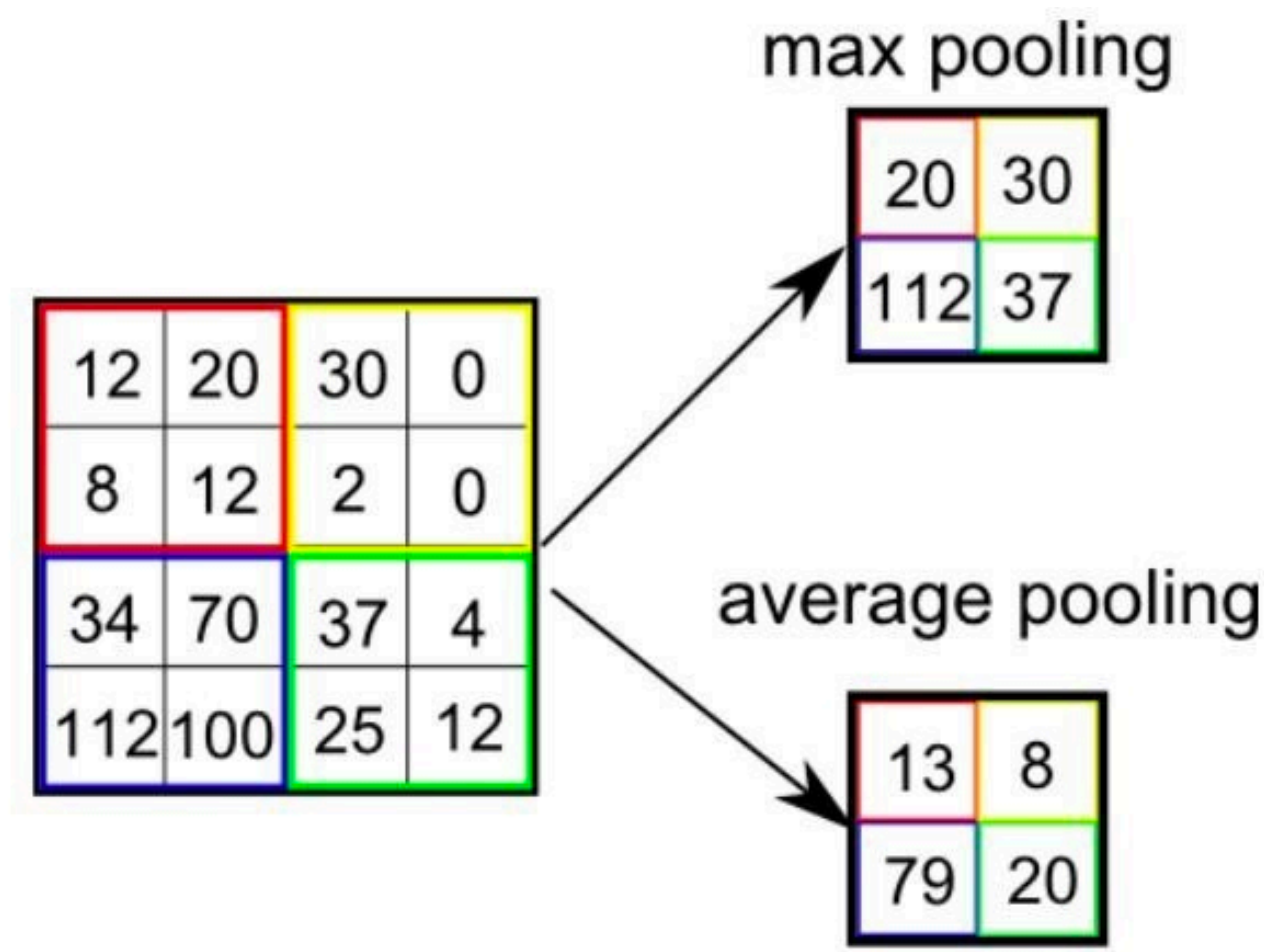
CNNs are composed by two kinds of layers

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



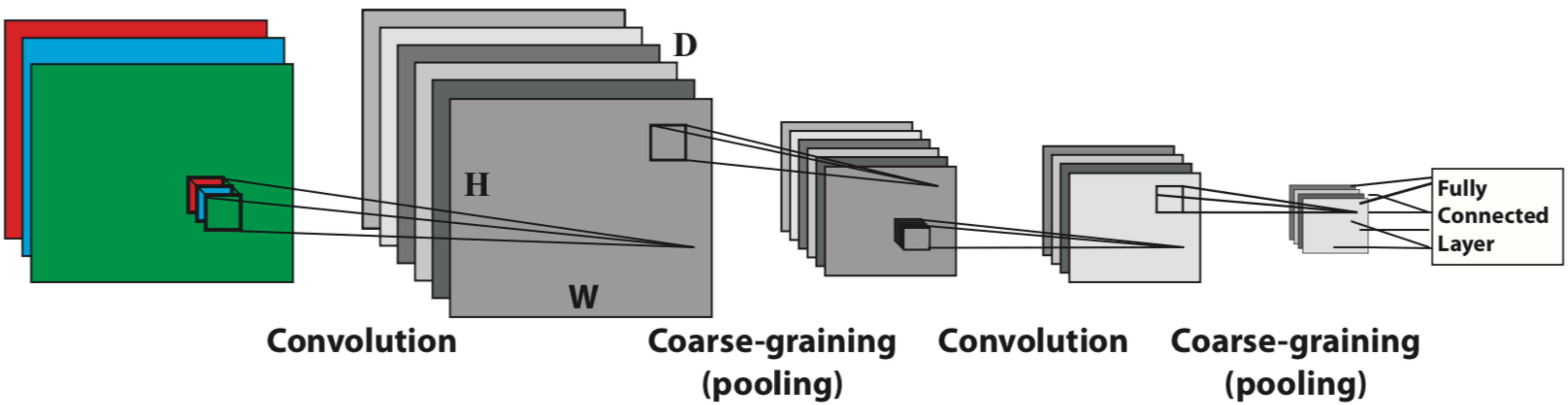
Convolutional Neural Networks

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

Convolution layer of input with **filters**

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

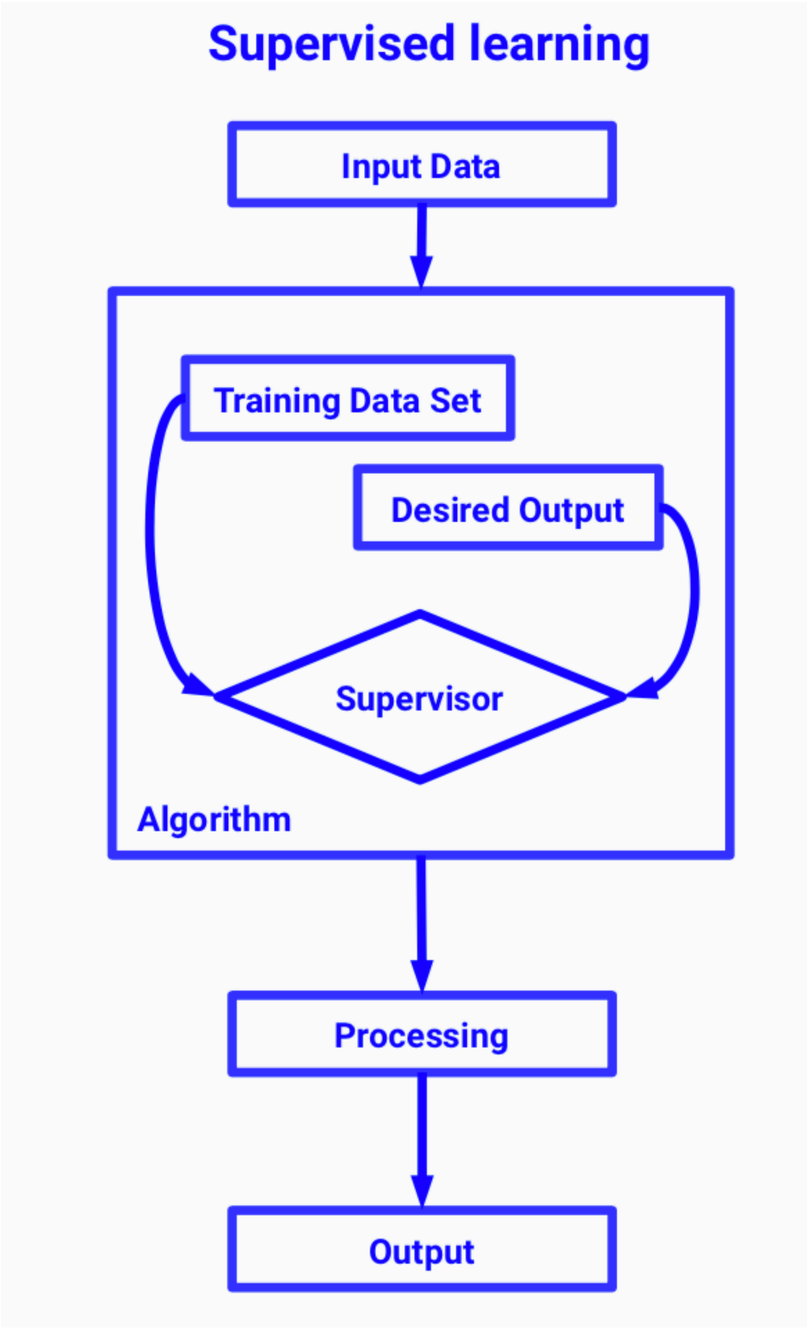
the convolution and max-pool layers are followed by an **all-to-all connected layer and a high-level classifier**, so that one can train CNNs using the standard backpropagation algorithm

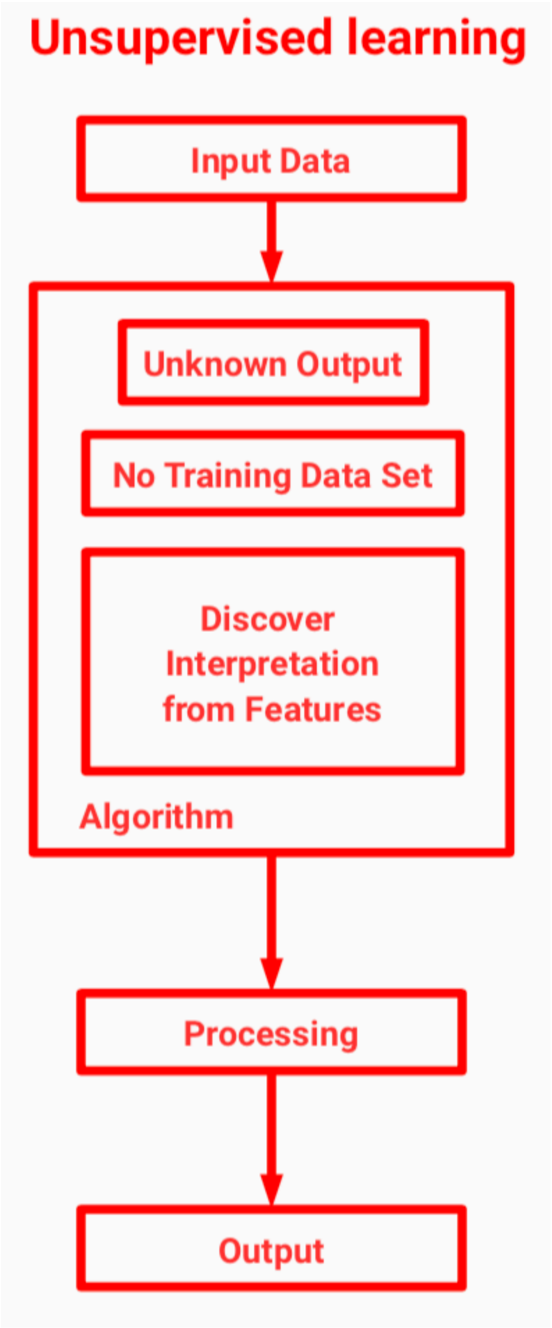
note that only problems characterised by a **spatial locality** are amenable to CNNs:
for example the 2D Ising dataset can be studied with CNNs, but not the SUSY dataset

How CNNs work

Reinforcement Learning

Supervised vs Unsupervised 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)

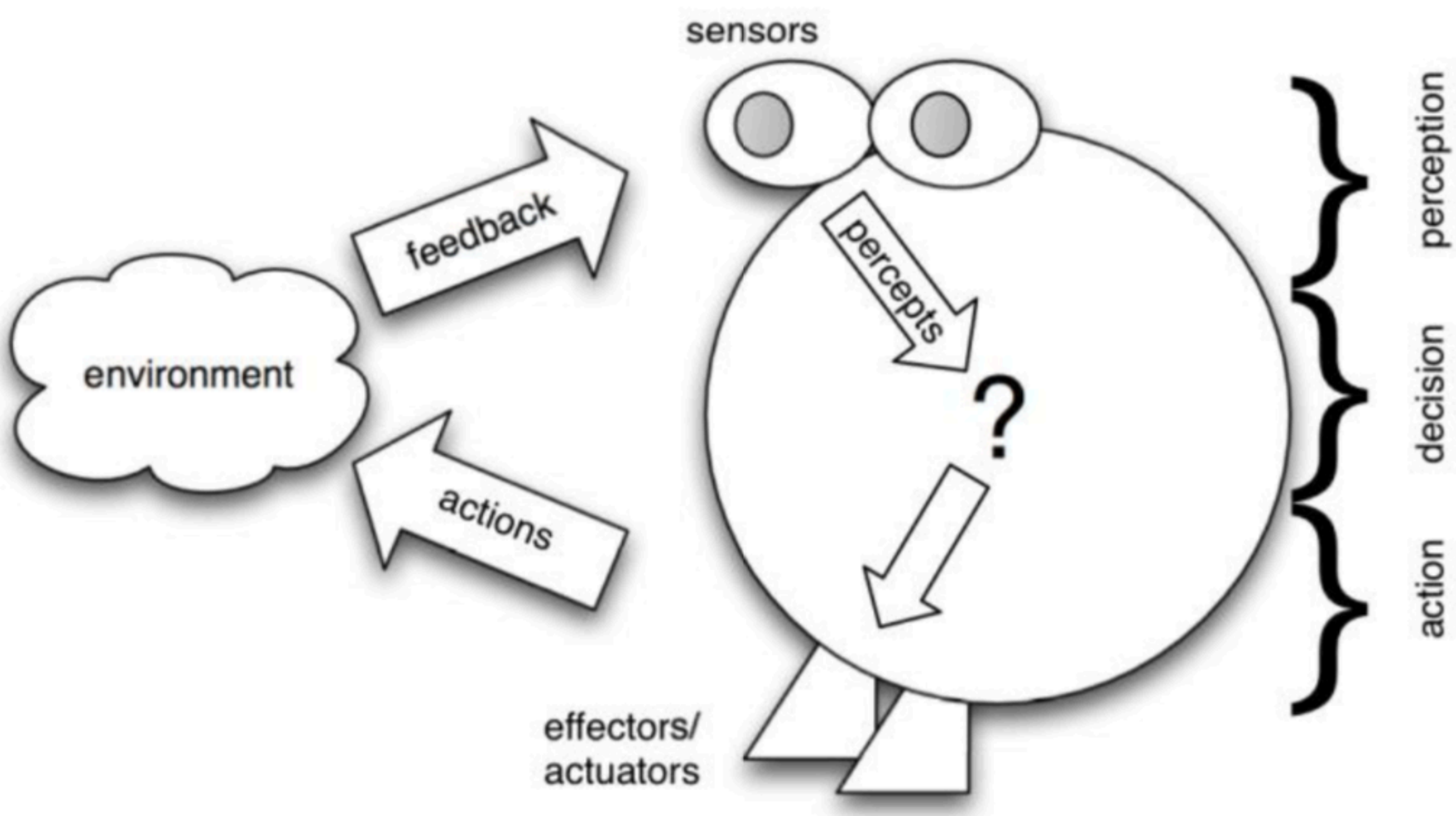
Unsupervised Learning: starting from a training dataset with **unlabelled examples, $\{x_i\}_{i=1,N}$** , produce a **model** that takes a sample as input and as output produces the solution of a practical problem, such as **clustering**, **dimensional reduction**, or **outlier detection**

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:

- ***Accessible or Inaccessible***: can the agent obtain updated and accurate information about the state of the environment?
- ***Deterministic or non-deterministic*** : has each action that the agent perform always associated the same effect?
- ***Static vs dynamics***: is the environment stable expect for the action of the agent?
- ***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

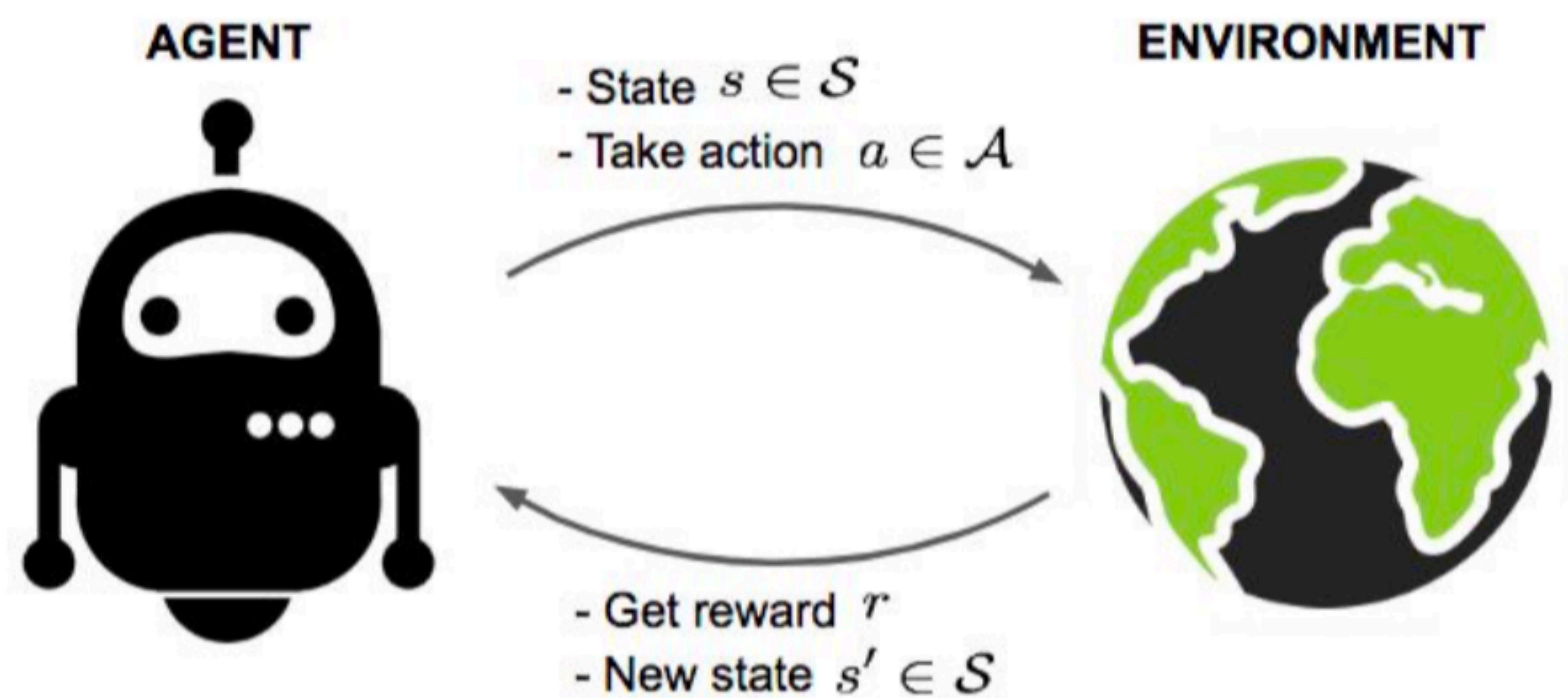
design an agent that **performs complex tasks** and **takes autonomous action** to fulfil its design goals, in an environment that is: partly inaccessible, non-deterministic, non-episodic, dynamic and continuous (*i.e.* the real world!).

- ☑ 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!).

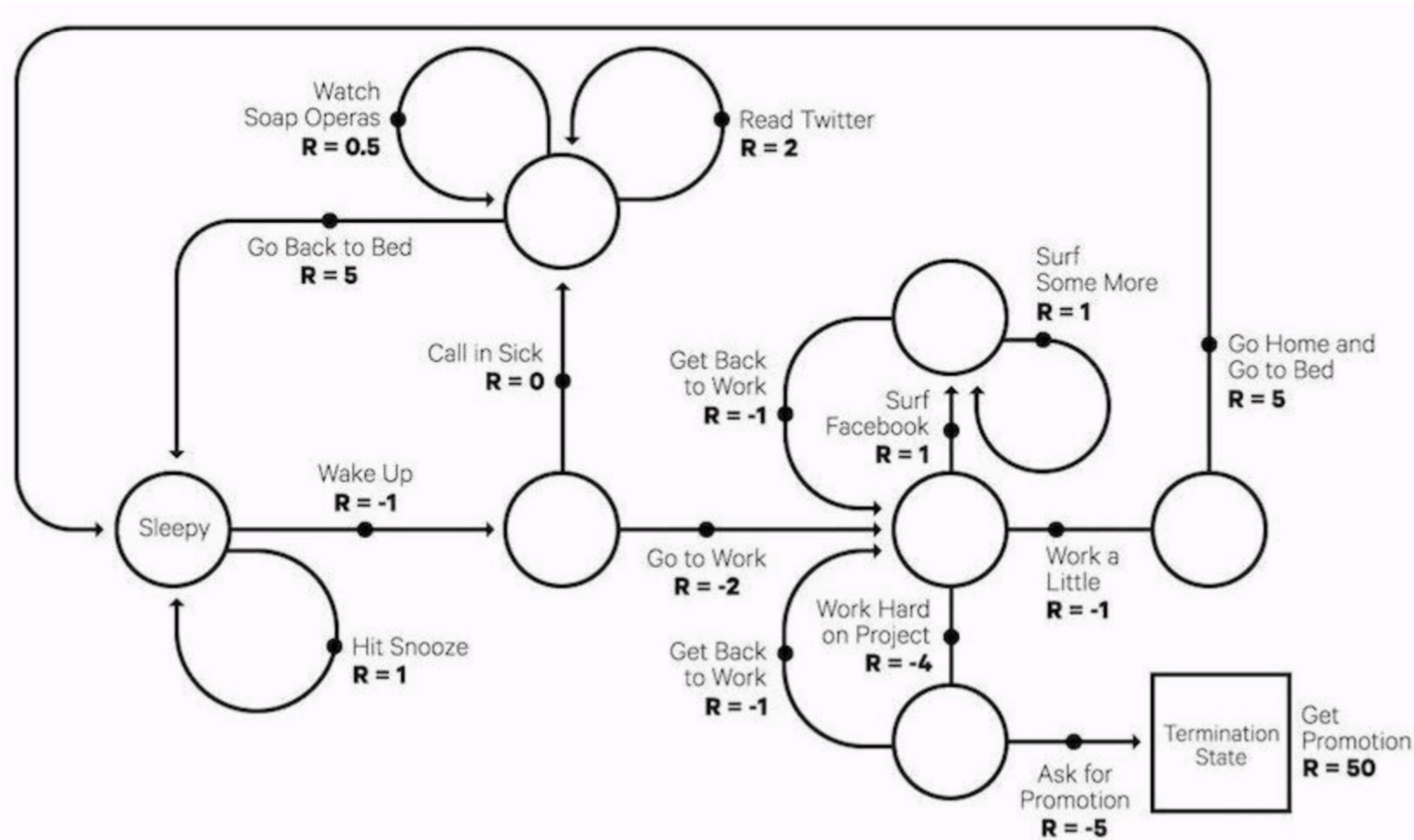


A Reinforcement Learning system

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



th

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

BLOG POST
RESEARCH
06 DEC 2018

AlphaZero: Shedding new light on chess, shogi, and 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.

BLOG POST
RESEARCH
24 JAN 2019

AlphaStar: Mastering the Real-Time Strategy Game StarCraft II

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

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**

$$Q\ : S \times A \rightarrow \mathbb{R}$$

quality function

state

action

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 then enters into a **new state s_t**
- The quality function (cost function) ***Q*** is updated

reward

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

discount factor

(estimate of)

future optimal value

learning rate

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