Machine Learning for Physics and Astronomy

Juan Rojo

VU Amsterdam & Theory group, Nikhef

Natuur- en Sterrenkunde BSc (Joint Degree), Honours Track

Lecture 6, 12/10/2021

Today's lecture

- The role of Symmetry in Pattern Recognition
- Image Classification with Convolutional Neural Networks
- Reinforcement Learning
- Bayesian (Probabilistic) Neural Networks

Symmetry in Pattern Recognition

Supervised learning and classification

The goal is to **predict a class label** from a pre-defined list of possibilities

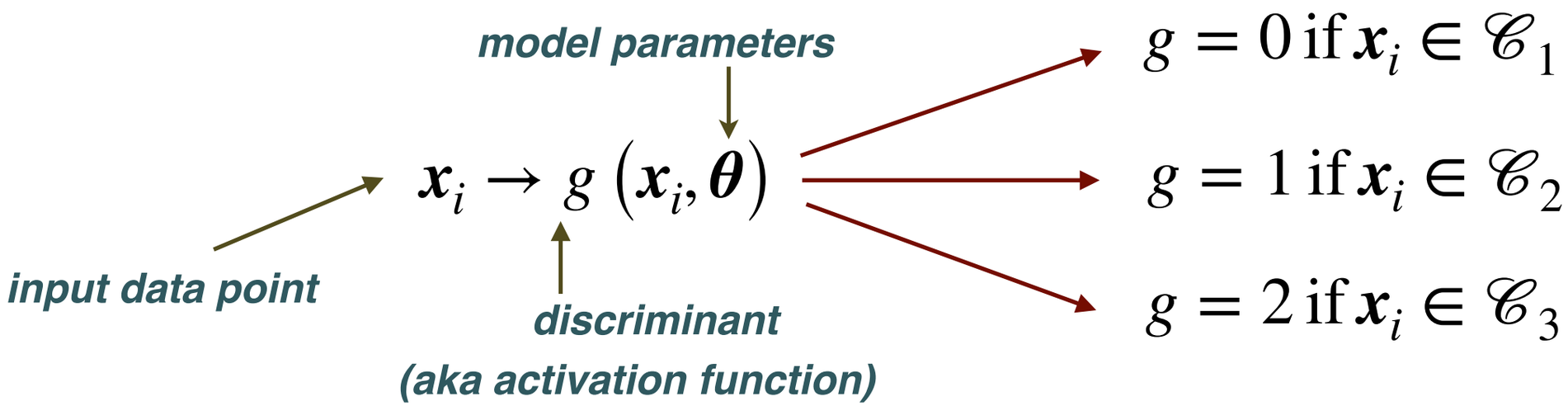
the simplest type of problem is **binary classification** (yes/no problems)

e.g. should I put this email in the spam folder?

but in general one considers **multiclass classification** (> 2 categories)

e.g. which type of bird is the one I just photographed?

In the context of ML applications there exist a large number of approaches to classifications tasks. The most basic one is based on assembling a **discriminant function** that maps each input data point to its specific class



Supervised learning and classification

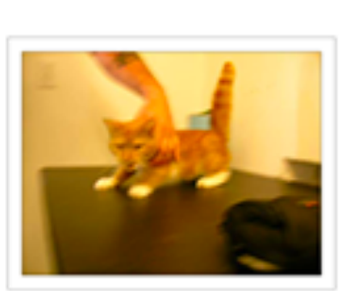
The goal is to **predict a class label** from a pre-defined list of possibilities

the simplest type of problem is **binary classification** (yes/no problems)

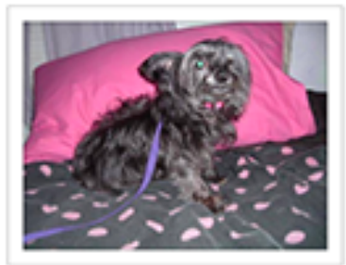
e.g. should I put this email in the spam folder?

but in general one considers **multiclass classification** (> 2 categories)

e.g. which type of bird is the one I just photographed?



cat or dog?



Linear classifiers

In this class of problems, the **dependent variables y_i** are discrete and take values *m=0, …, M-1*, so the index *m* also labels the *M* categories

The goal is to **classify the *n* input samples**, each composed by ***p*** **features,** intro the ***M*** **possible categories** of the problem

The simplest classifier is the **perceptron:** a **linear classifier** that categorises examples from a linear combination of the features

$$\sigma(s_i) = \text{sign}(s_i) = \text{sign}\left(\boldsymbol{x}_i^T \boldsymbol{\theta} + b_0\right)$$

Q: what would the features be in the previous example?

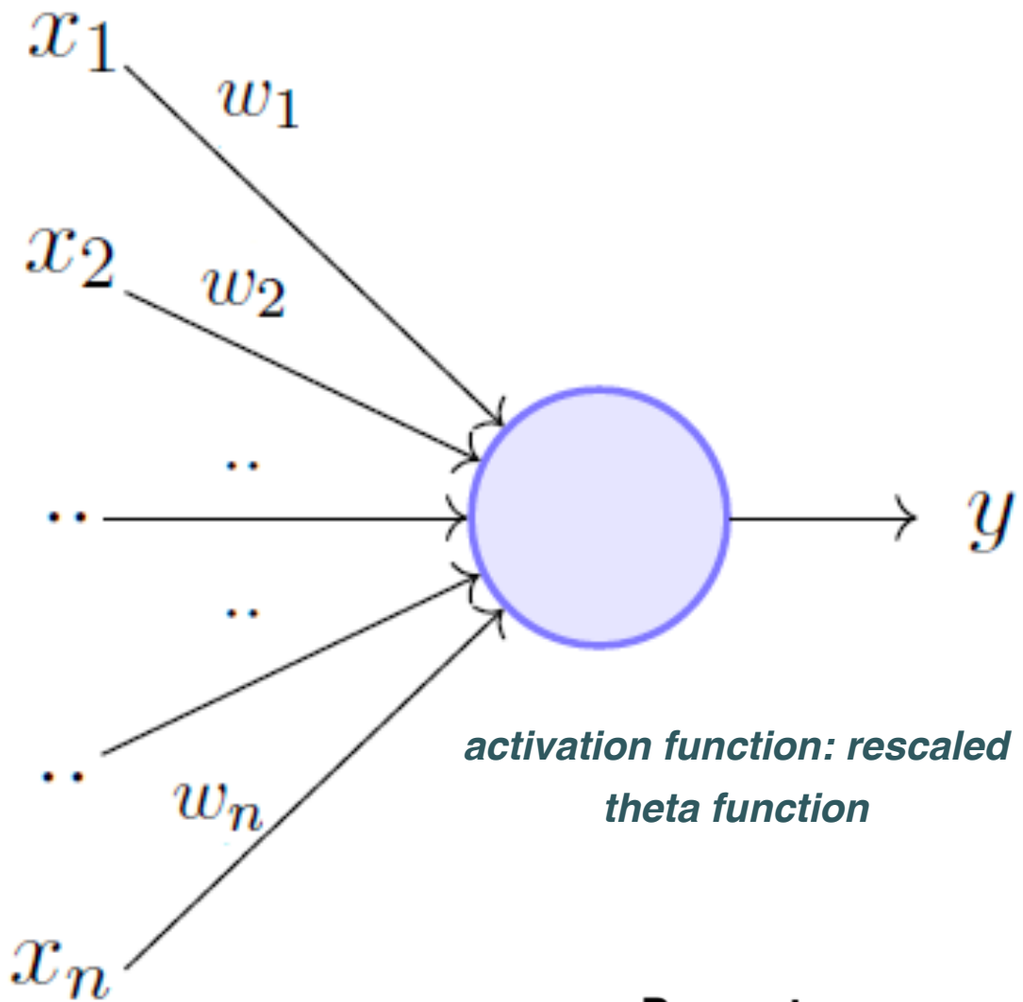
$\boldsymbol{x}_i = \left(x_{i,1}, x_{i,2}, \ldots, x_{i,p}\right)$ *p features of the samples*

model parameters $\boldsymbol{\theta} = \left(\theta_1, \theta_2, \ldots, \theta_p\right)$

a perceptron is a **hard classifier** where each sample is assigned to a category with 100% probability

Perceptrons

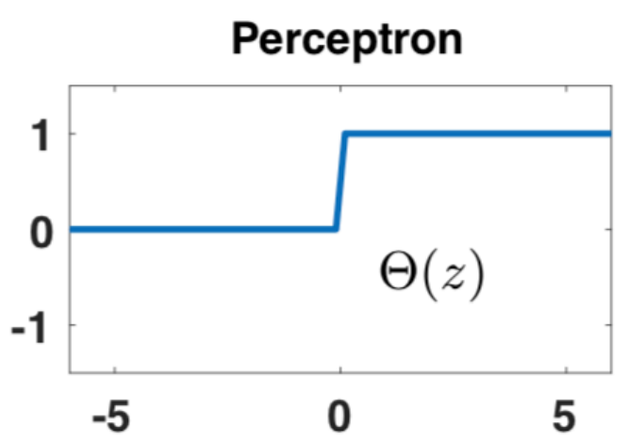
You can think of a perceptron as a **very simple, linear neural network**

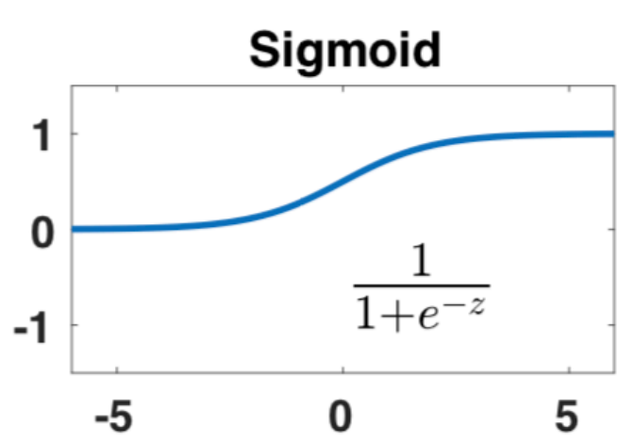


activation function: rescaled theta function

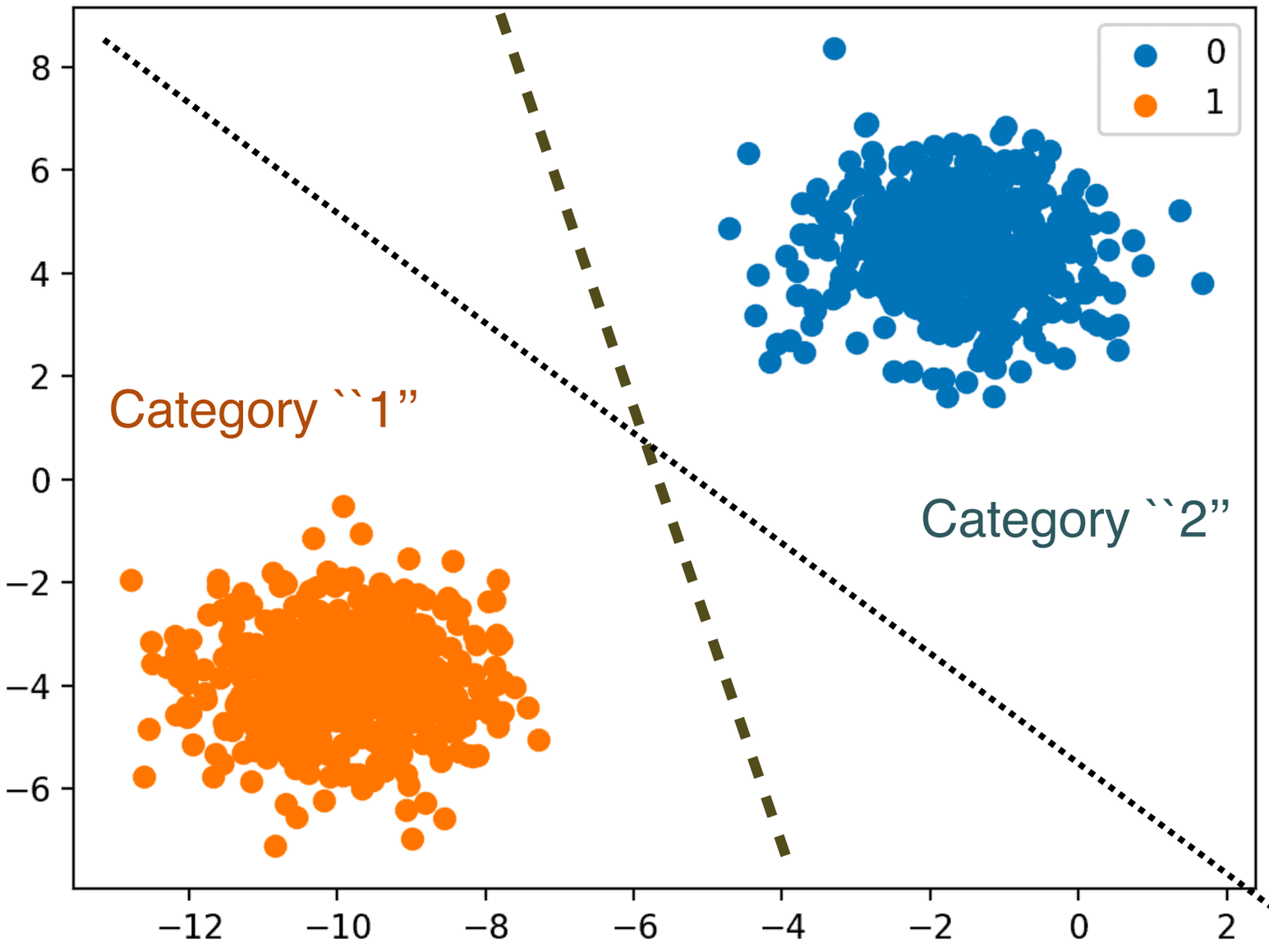
$$y = 1 \text{ if } \left(\boldsymbol{x}_i^T \boldsymbol{\omega} + b_0\right) \geq 0$$

$$y = -1 \text{ if } \left(\boldsymbol{x}_i^T \boldsymbol{\omega} + b_0\right) < 0$$





Linear classification



Only suitable for a limited number of classification problems!

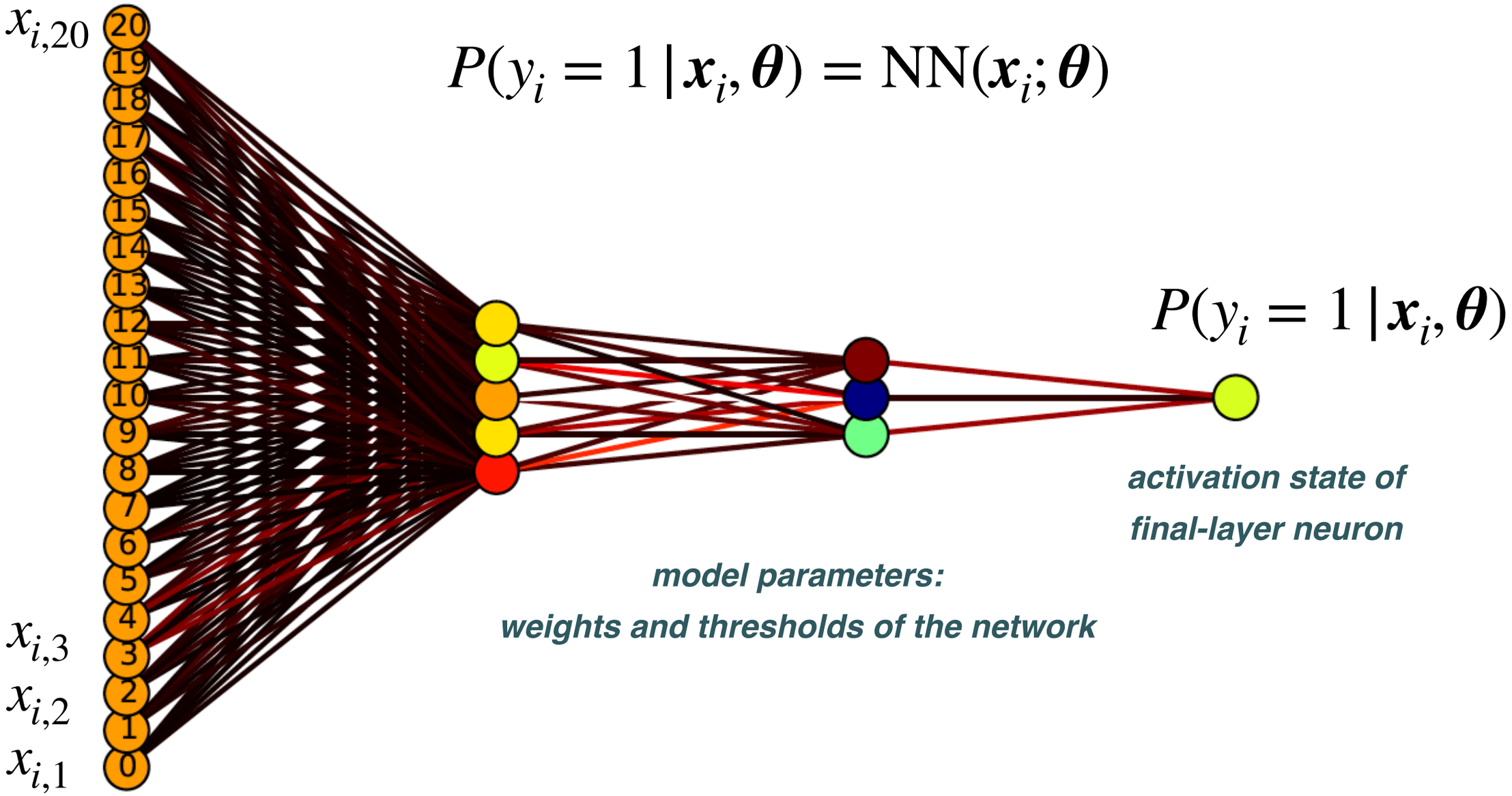
Supervised learning and classification

instead of modelling the classification probability by a **simple logistic function**

$$P(y_i = 1 \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}) = \frac{1}{1 + e^{-\boldsymbol{x}_i^T \boldsymbol{\theta}}}$$

one can adopt more complex models, such as **deep neural networks**

$$P(y_i = 1 \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}) = \mathrm{NN}(\boldsymbol{x}_i; \boldsymbol{\theta})$$



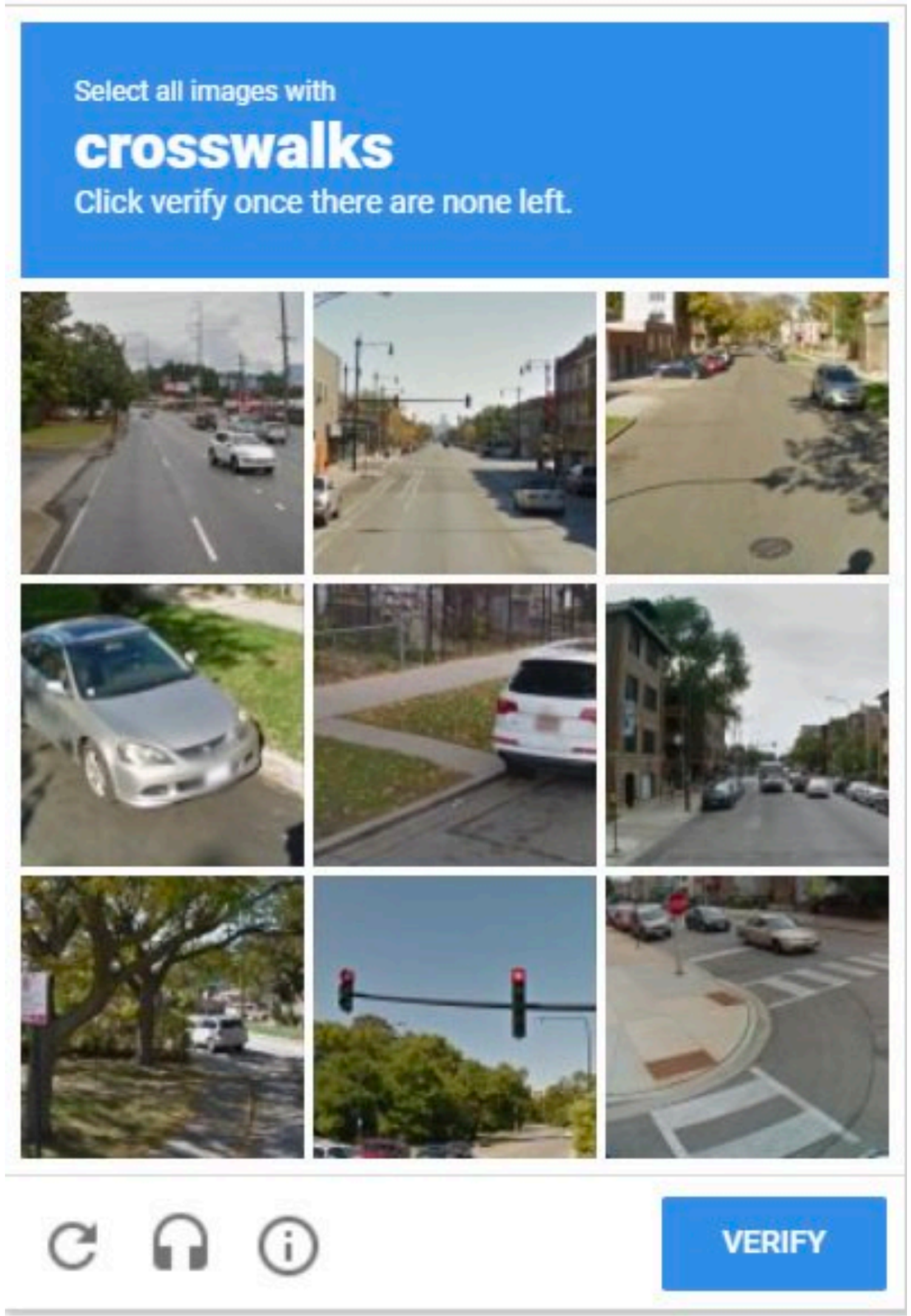
Pattern recognition

Modelling classification with **feed-forward deep networks**

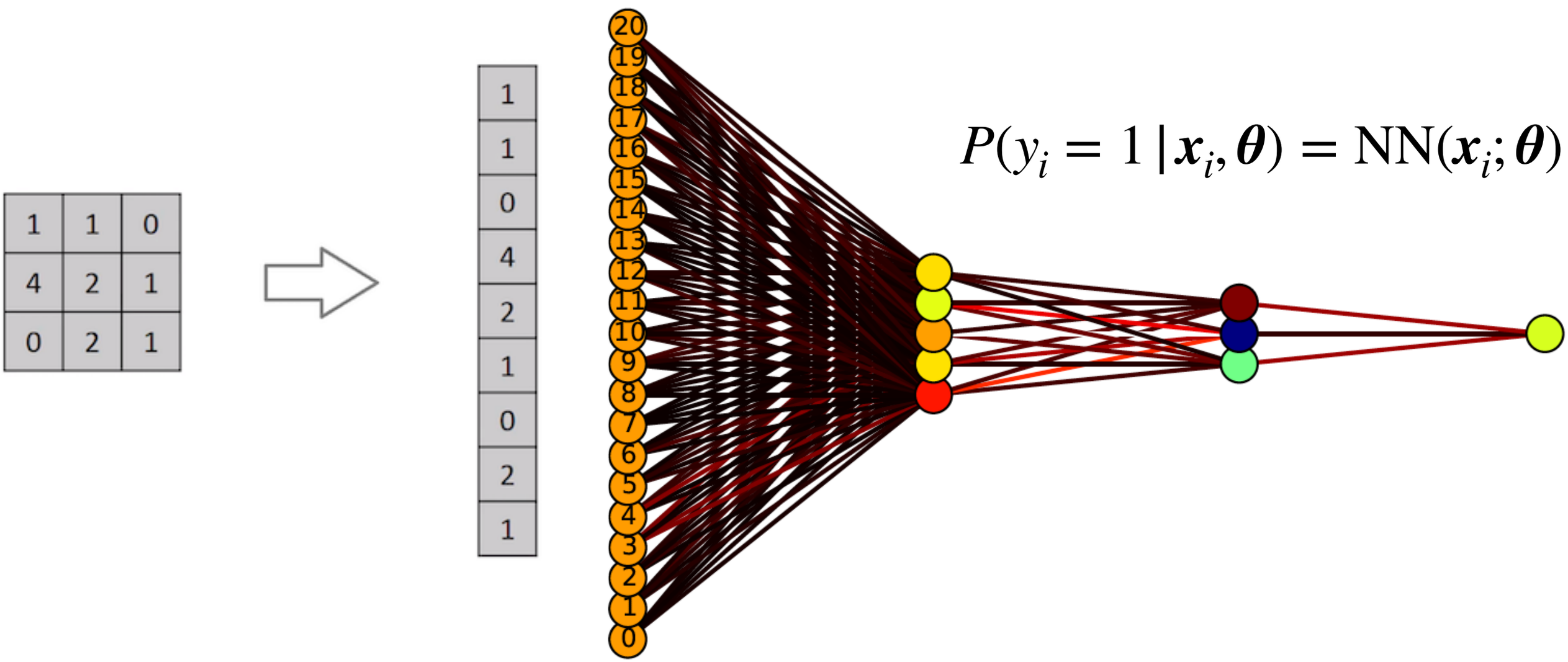
$$P(y_i = 1 \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}) = \mathrm{NN}(\boldsymbol{x}_i; \boldsymbol{\theta})$$

may appear at first glance to work also with **pattern recognition in images**

Q: what do we need to do to ``feed'' such images to a deep feed-forward network for training?



Pattern recognition



An image is is just a matrix of values (RGB intensities in each pixel), hence it seems amenable to **standard DNN classification?**

Q: why do you think that this may not be the case?

Pattern recognition

A **brute-force approach (DNN)** approach is no good for pattern recognition since:

- It ignores that patterns exhibits a **number of symmetries** e.g. invariance under translations, reflection, or rotations

- It ignores that a **pattern has intrinsic structure** e.g. if a car has wheels on one side it will also have wheels on the other

- It ignores that a pattern is composed by **spatially ordered features.** e.g. the whiskers of a cat are always close to the eyes

cat

Pattern recognition

A **brute-force approach (DNN)** approach is no good for pattern recognition since:

- It ignores that patterns exhibits a **number of symmetries** e.g. invariance under translations, reflection, or rotations

- It ignores that a **pattern has intrinsic structure** e.g. if a car has wheels on one side it will also have wheels on the other

- It ignores that a pattern is composed by **spatially ordered features.** e.g. the whiskers of a cat are always close to the eyes

cat

still a cat!

Pattern recognition

A **brute-force approach (DNN)** approach is no good for pattern recognition since:

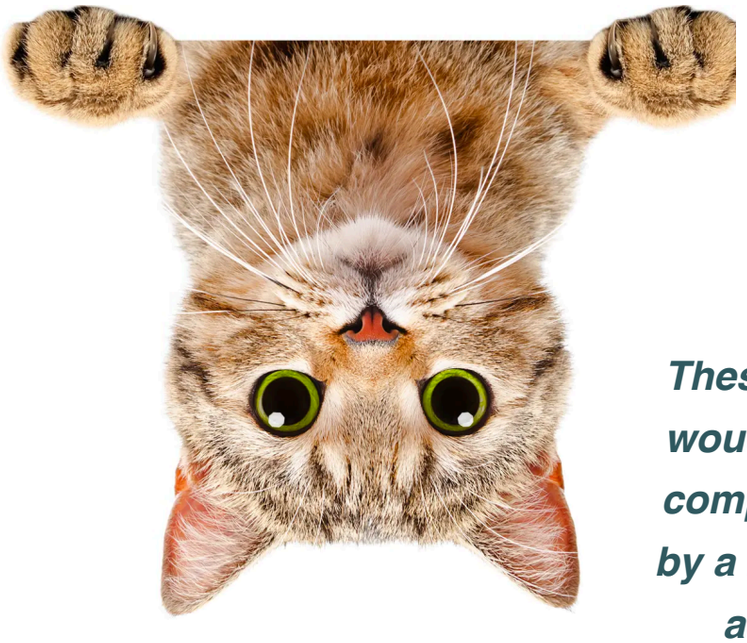
- It ignores that patterns exhibits a **number of symmetries** e.g. invariance under translations, reflection, or rotations

- It ignores that a **pattern has intrinsic structure** e.g. if a car has wheels on one side it will also have wheels on the other

- It ignores that a pattern is composed by **spatially ordered features.** e.g. the whiskers of a cat are always close to the eyes

cat

still a cat!

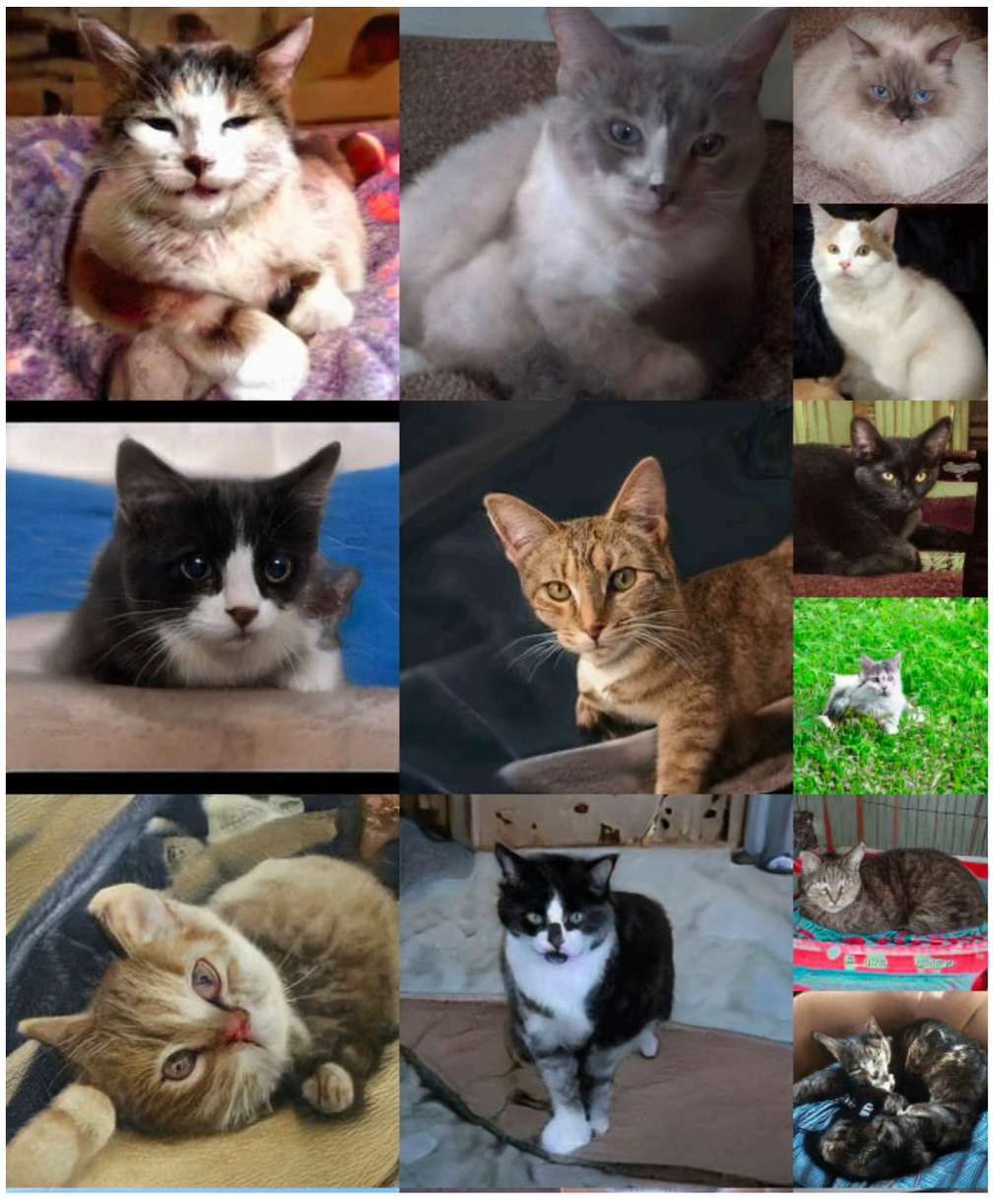


again, still a cat!

These three images would be treated as completely different by a DNN, while they are the same!

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***

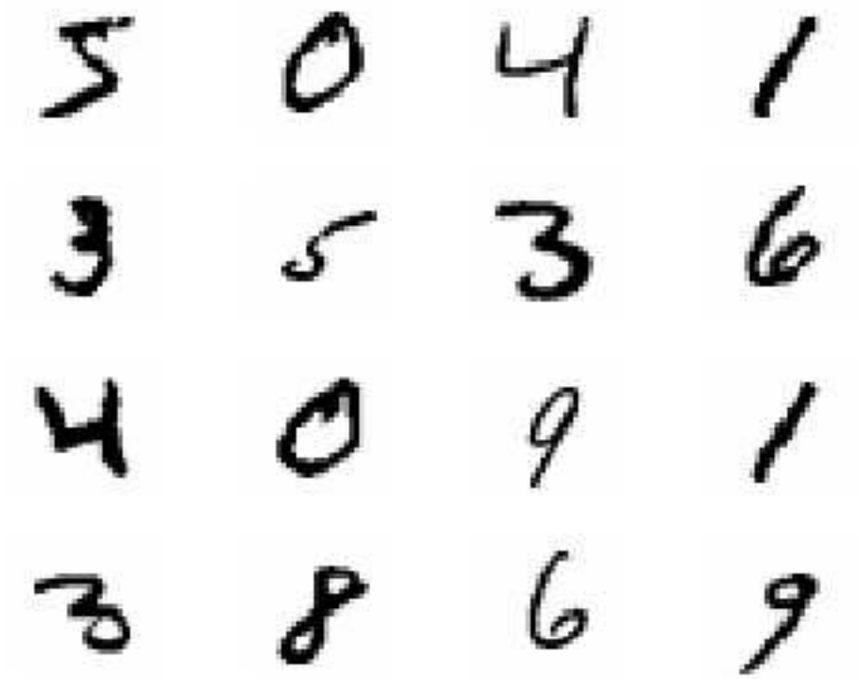
Learning with symmetry

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

Aim: to 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

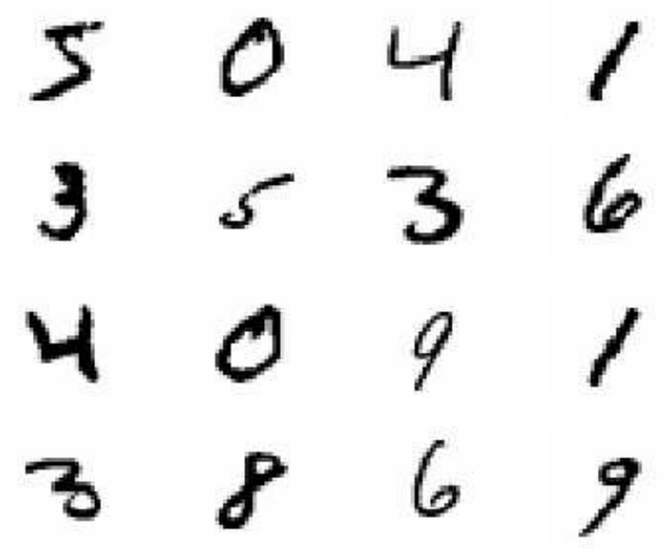
Q: what kind of **symmetries** must we built-in in our ML classifier model?

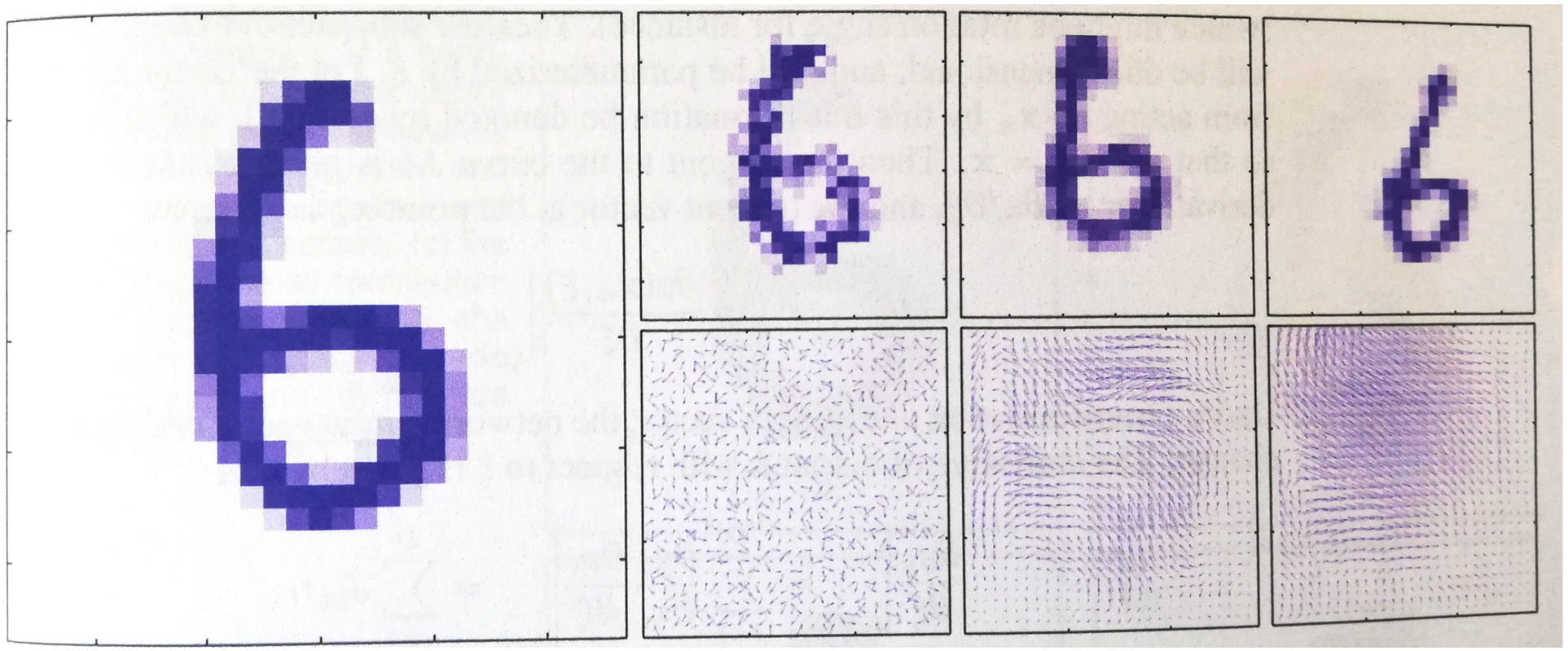
Learning with symmetry

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

some are obvious choices ...

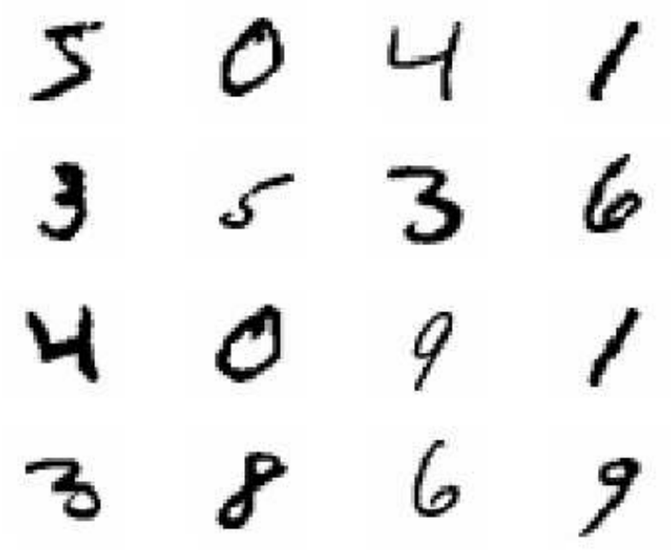
- Invariance under **translations**
- Invariance under **scaling**





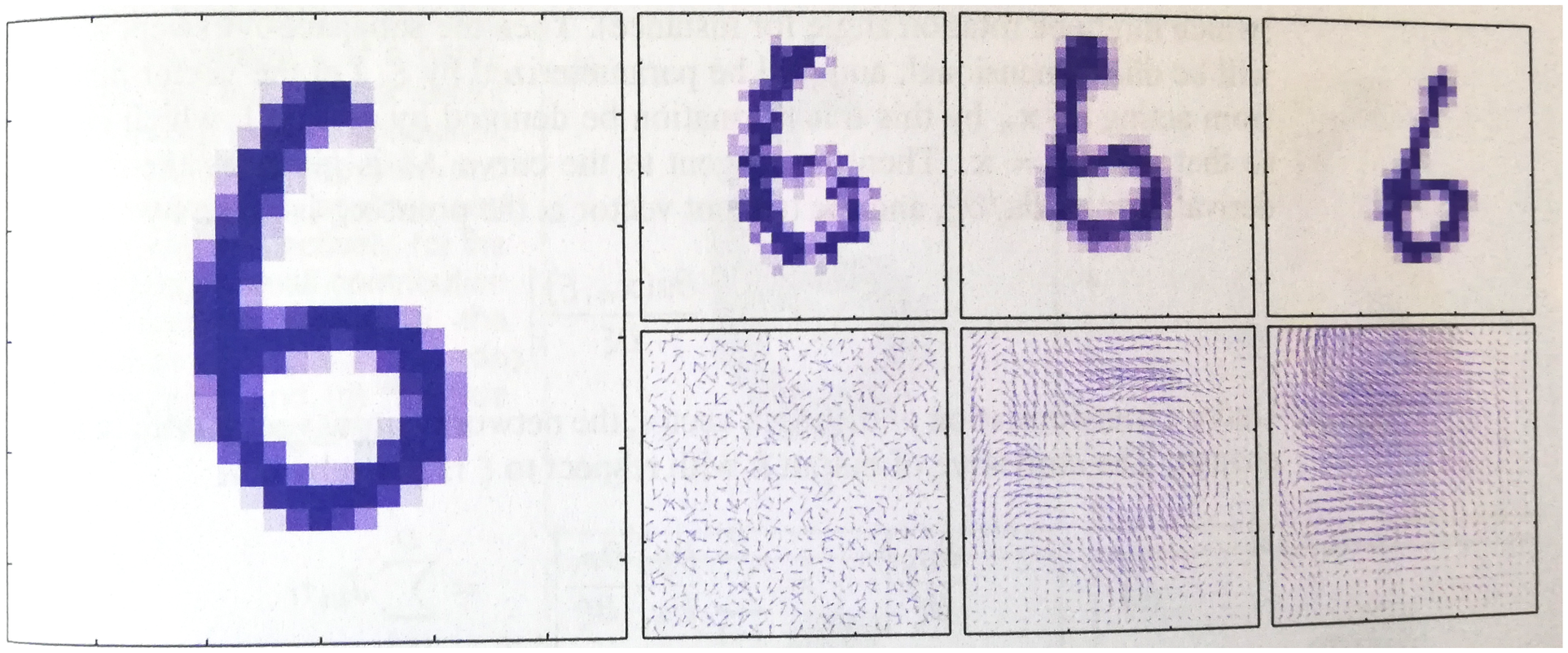
Learning with symmetry

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**

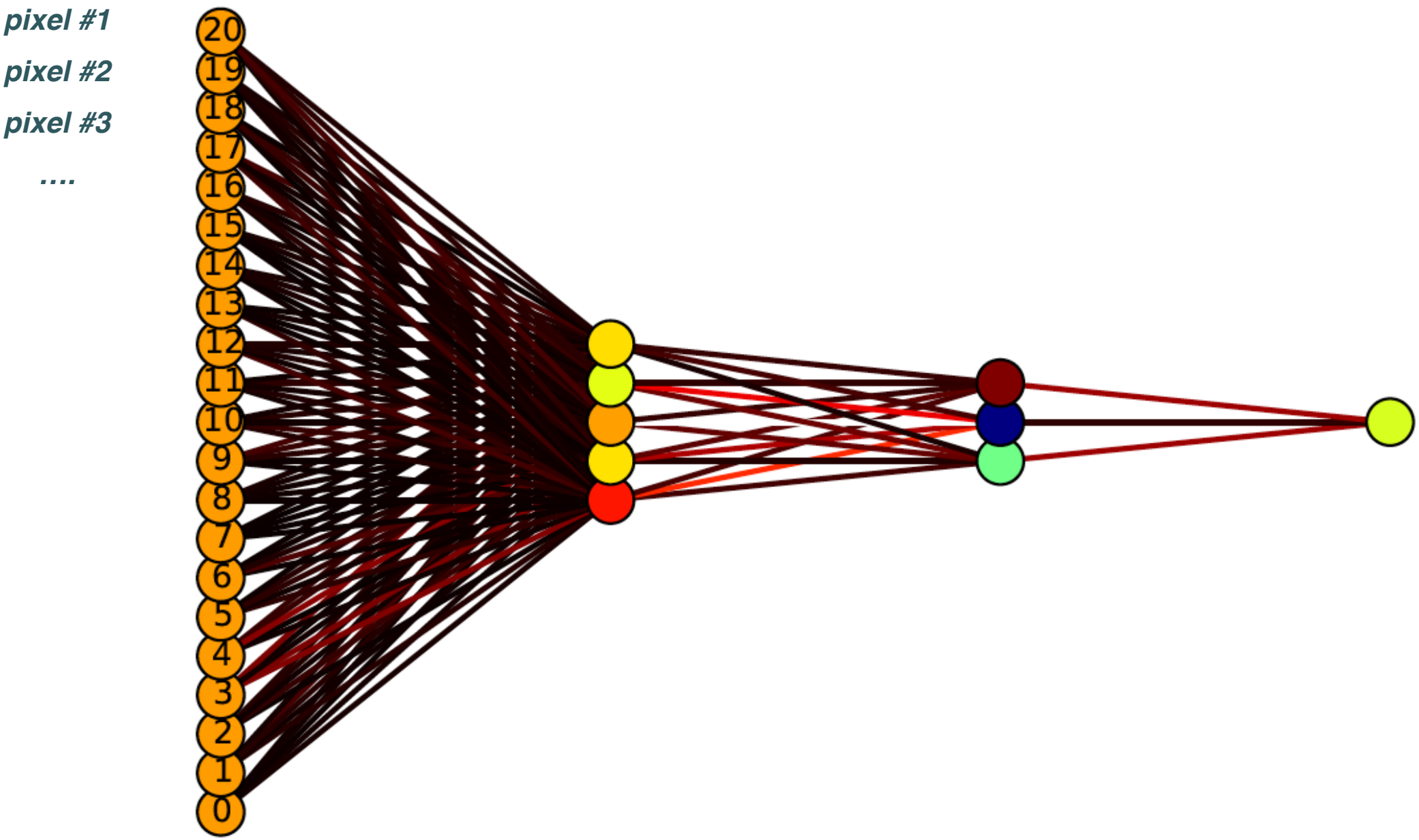
but others are more non-trivial!



Convolutional Neural Networks

Convolutional Neural Networks

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



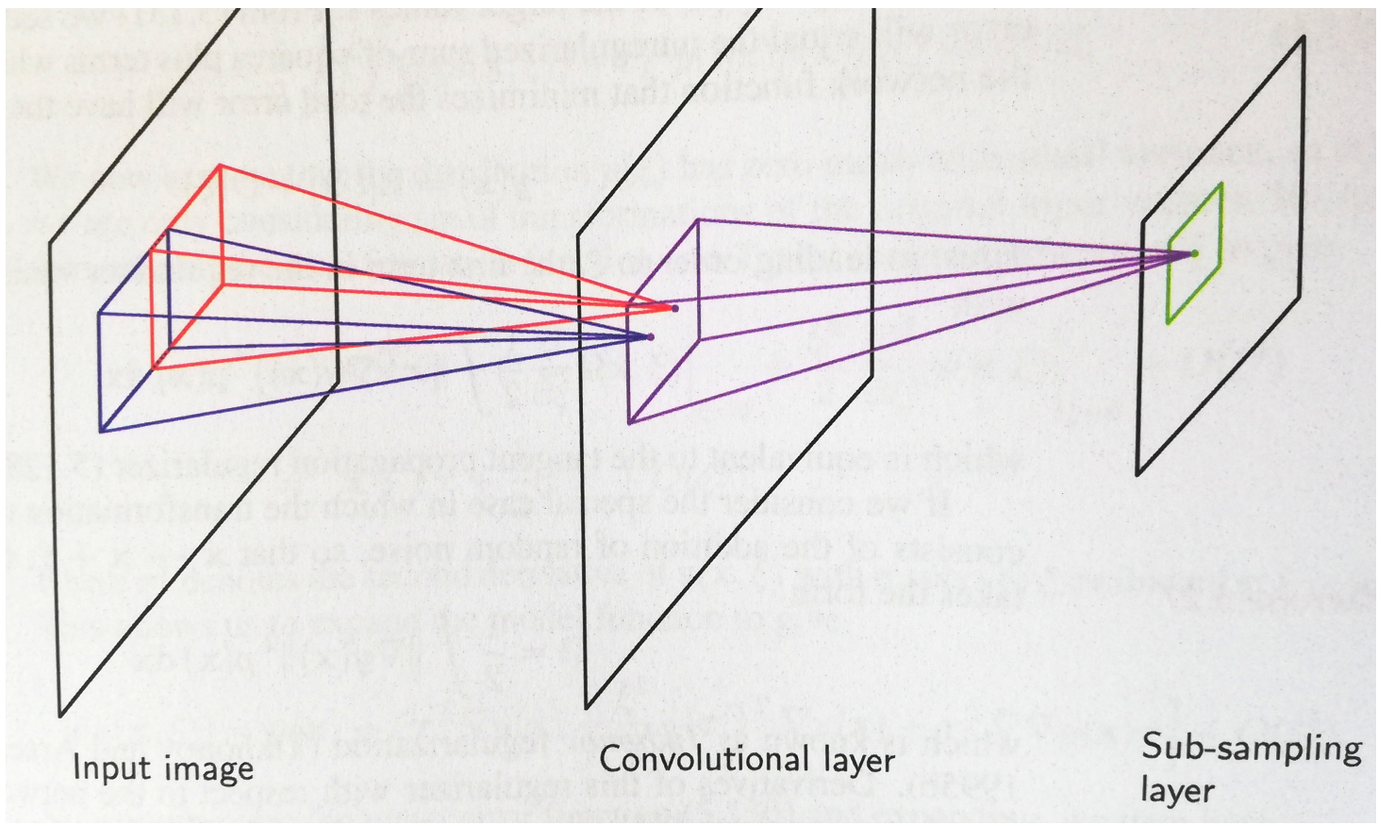
Convolutional Neural Networks

the simplest approach would be to input the images to a **fully connected NN** 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

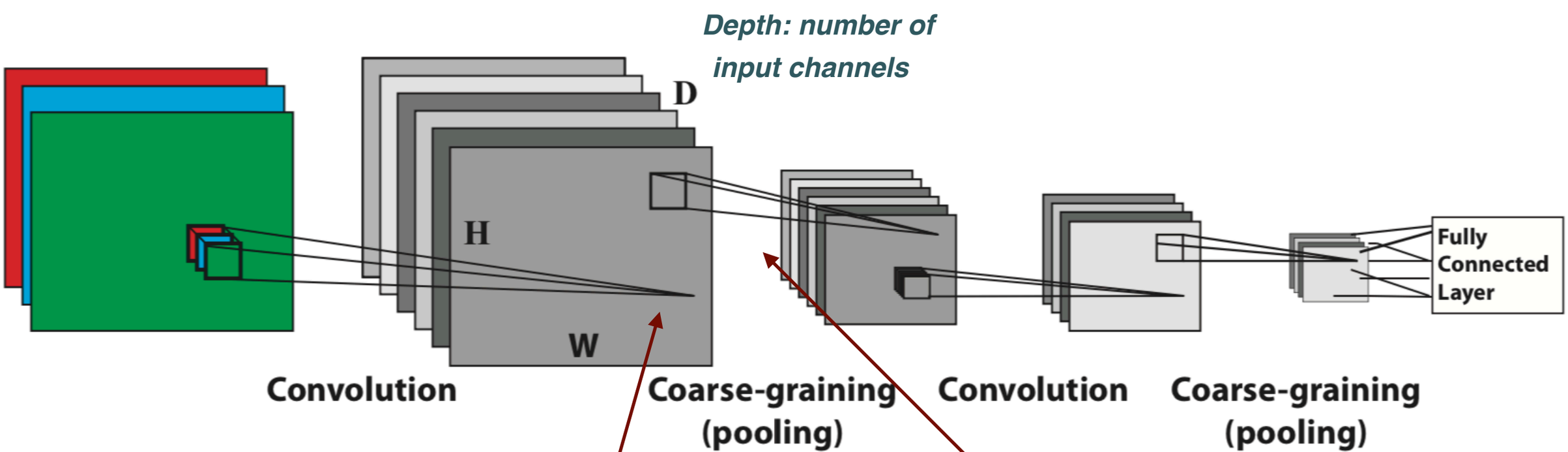
afterwards such local features can be combined into **higher-level ones**

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

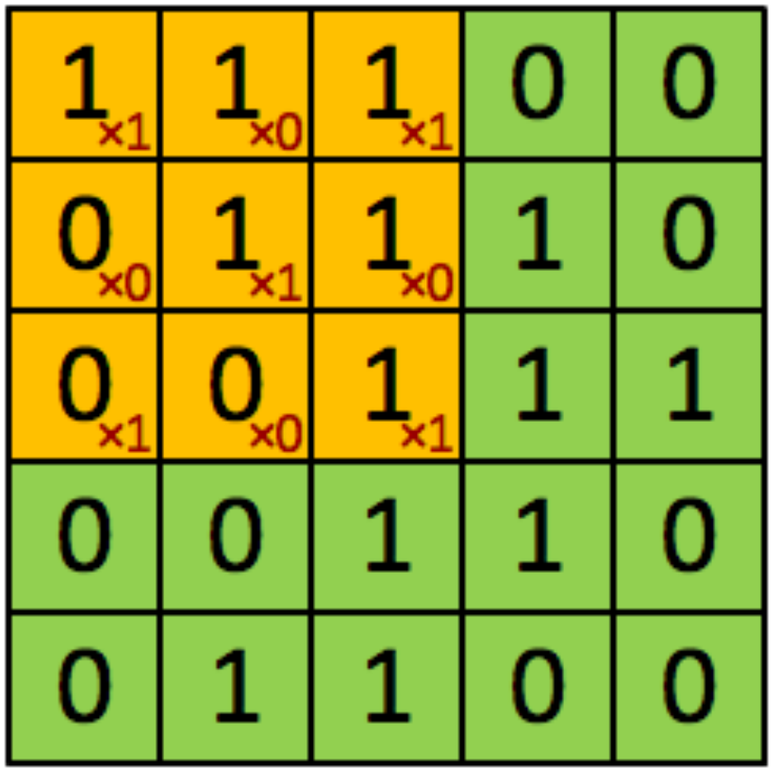


D=3 for RGB images
Height (H) and Width (W) determined by number of pixels

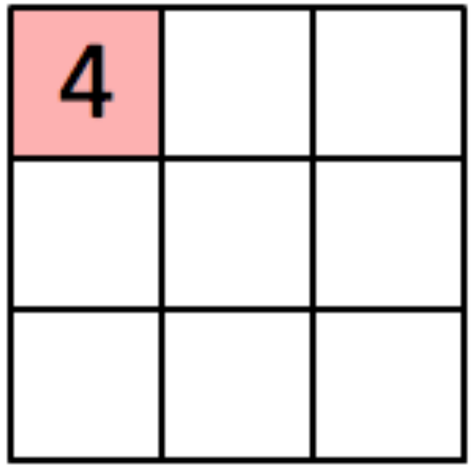
neuron activation state: convolution with local spatial filter (eg 3 x 3 pixel grid)

pooling layers reduce H, W while preserving D

Convolutional Layer



Image



Convolved Feature

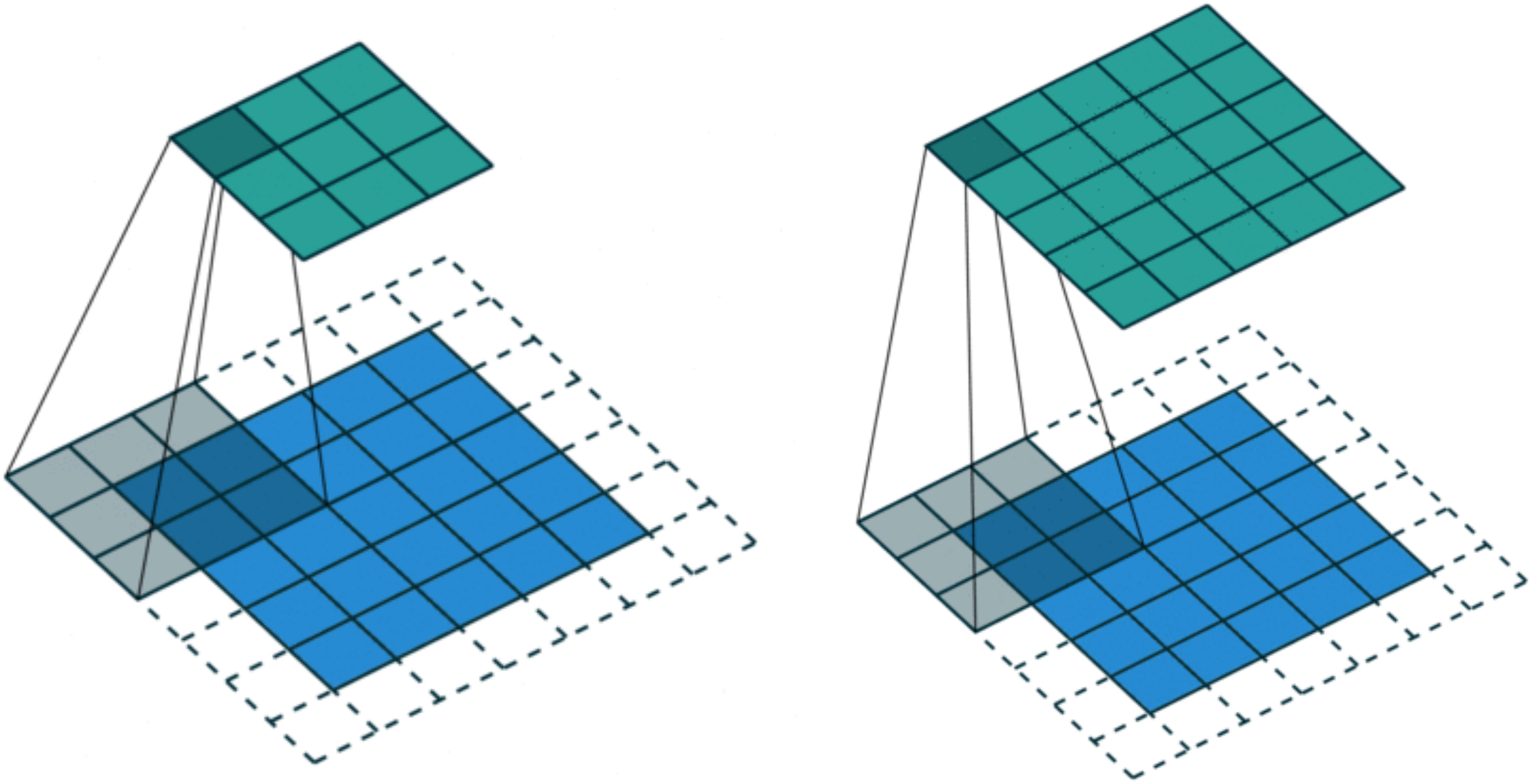
$$\text{Kernel} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

Sweep a **kernel** along the input image (*5 x 5 pixels*). Here the Kernel is (3 *x 3)* hence the **convolved feature is** *(3 x 3)*

e.g. first row & first layer: 1x1 + 1x0 + 1x1 + 0x0 + 1x1 + 1x0 + 0x1 + 0x0 + 1x1 =4

The goal of the **Convolutional Layers** to to identify **low-level** features in images such as edges

Convolutional Layer



Depending on the choice of **kernel, padding,** and **stride length** we will get different results for a convolved layer

Q: which parameters of the CNN can then be tuned during the network training?

Convolutional Neural Networks

CNNs are composed by two kinds of layers

- Convolution layer of input with **filters/kernels**
- **Pooling** layer that coarse-grains the input while maintaining locality and spatial structure

Q: Assume you have a 4x4 image: how you can coarse-grain information?

$$\begin{pmatrix} 3 & 2 & 5 & 7 \\ 0 & 1 & 0 & 8 \\ 1 & 6 & 9 & 23 \\ 14 & 8 & 6 & 3 \end{pmatrix}$$

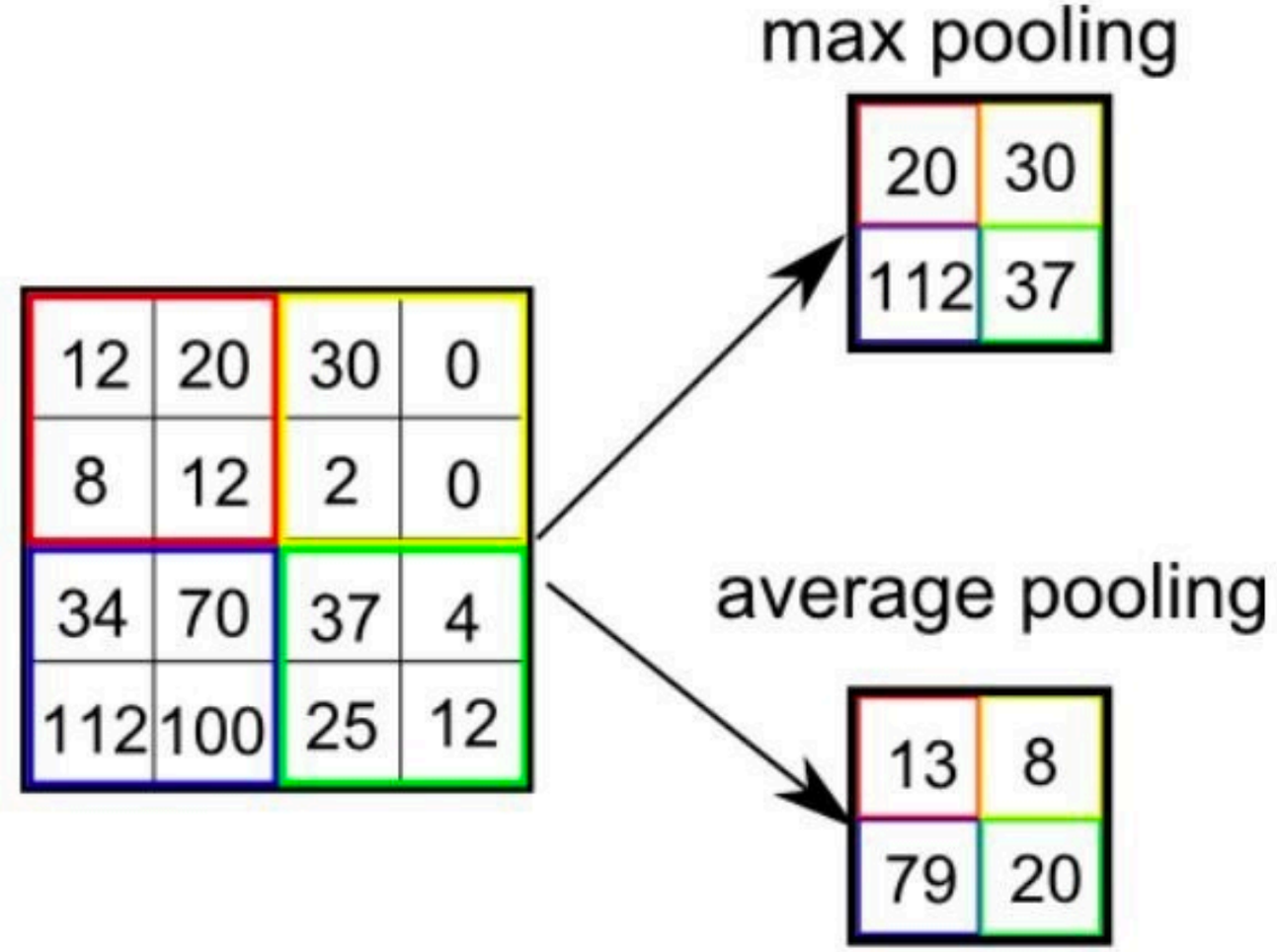
Convolutional Neural Networks

CNNs are composed by two kinds of layers

- Convolution layer of input with **filters/kernels**
- **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/kernels**
- **Pooling** layer that coarse-grains the input while maintaining locality and spatial structure

original image image

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

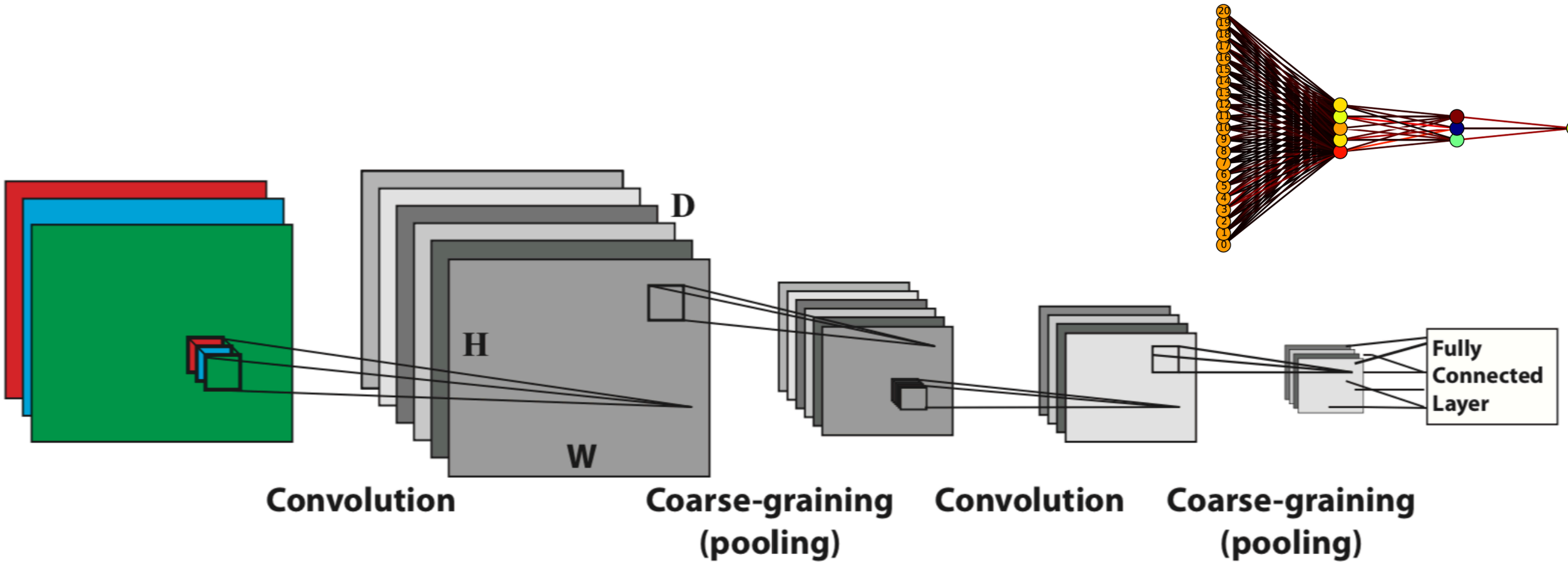
Average Pooling

pooled image

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

Convolutional Neural Networks

the convolution and pooling 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

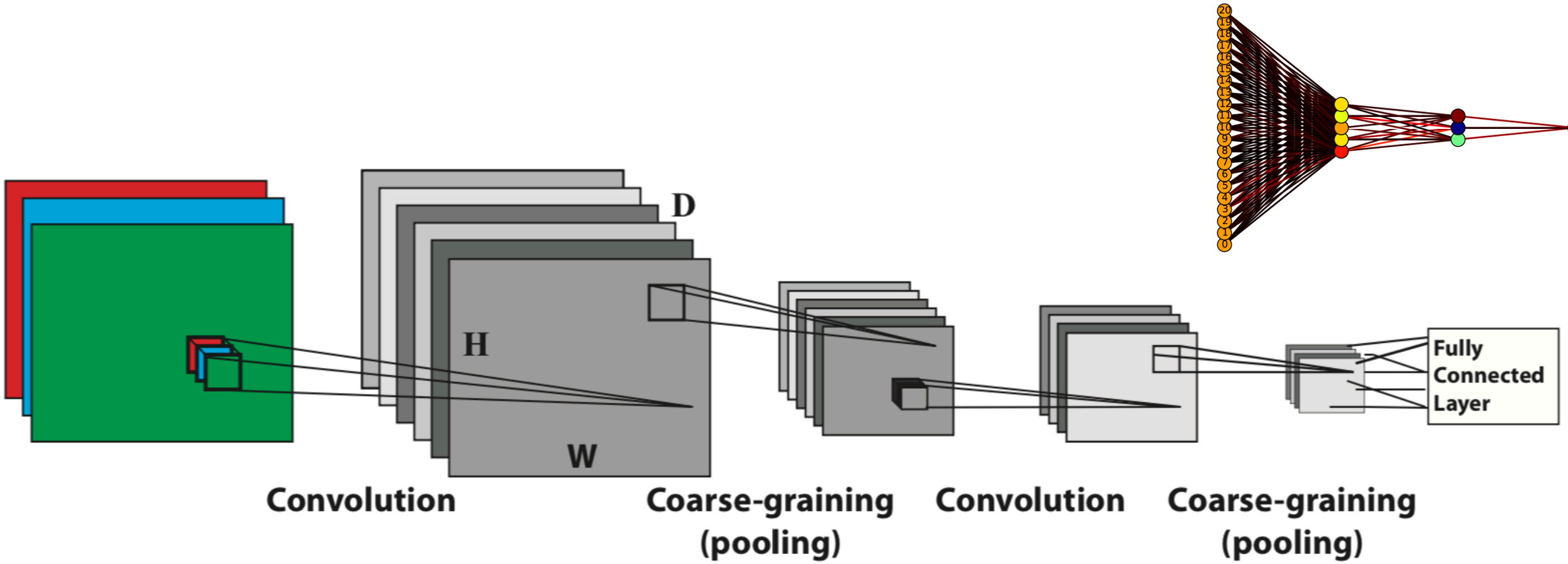


After convolution + pooling layers, we can flatten our images and input them to a regular DNN!

Q1: what we have gained? Why this is better than just flattening out the original images?

Convolutional Neural Networks

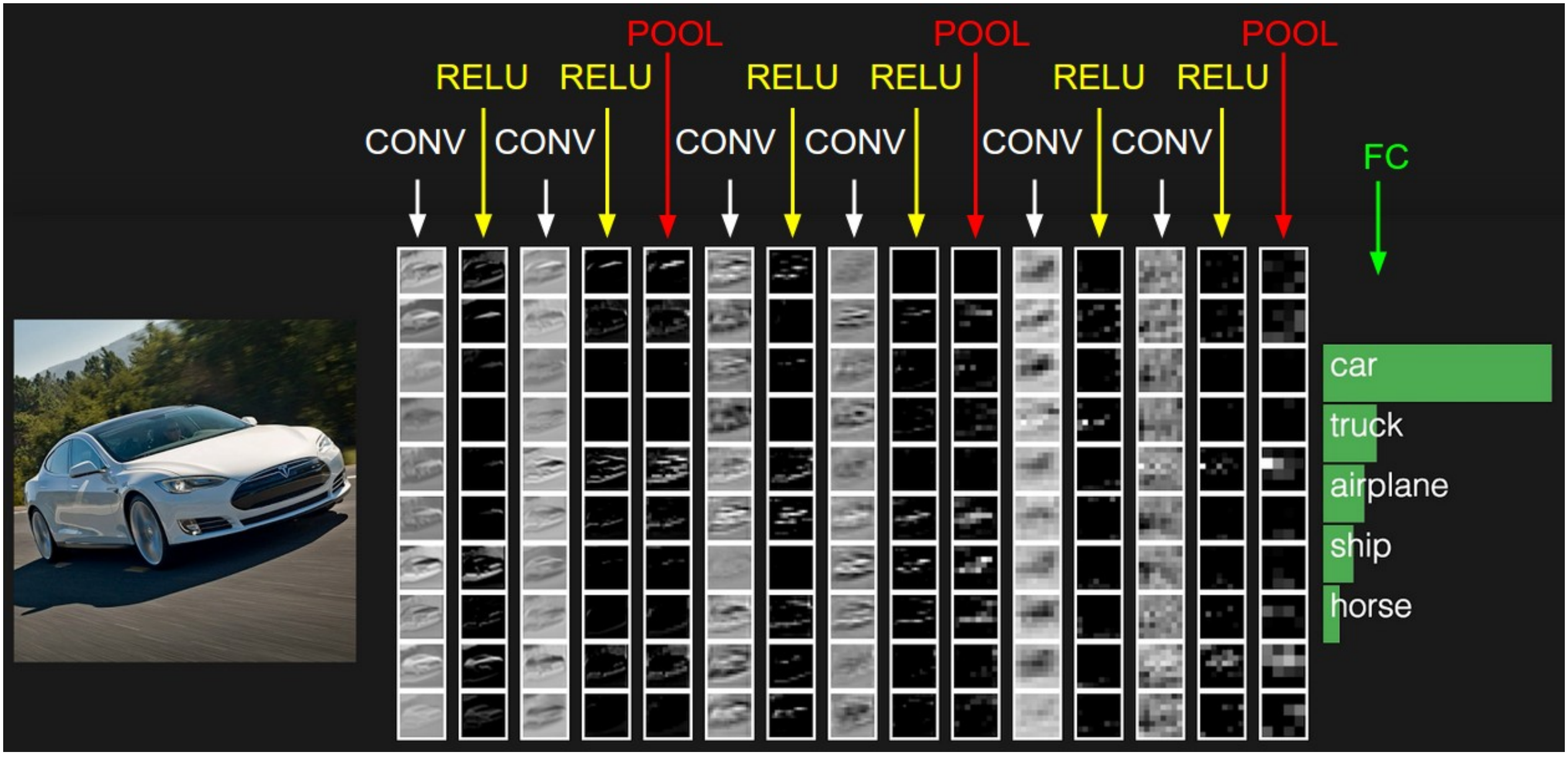
the convolution and pooling layers are followed by an **all-to-all connected layer and a high-level classifier**, so that one can train CNNs using the standard backpropagation algorithm



After convolution + pooling layers, we can flatten our images and input them to a regular DNN!

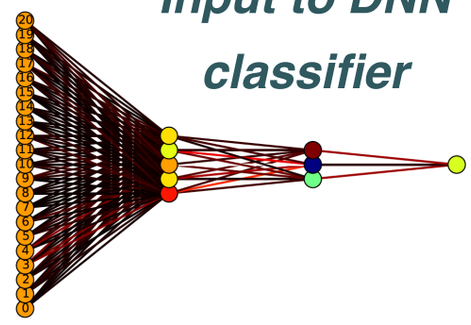
Q2: Do you think we can tailor kernels/filter to identify specific features of a pattern?

Convolutional Neural Networks

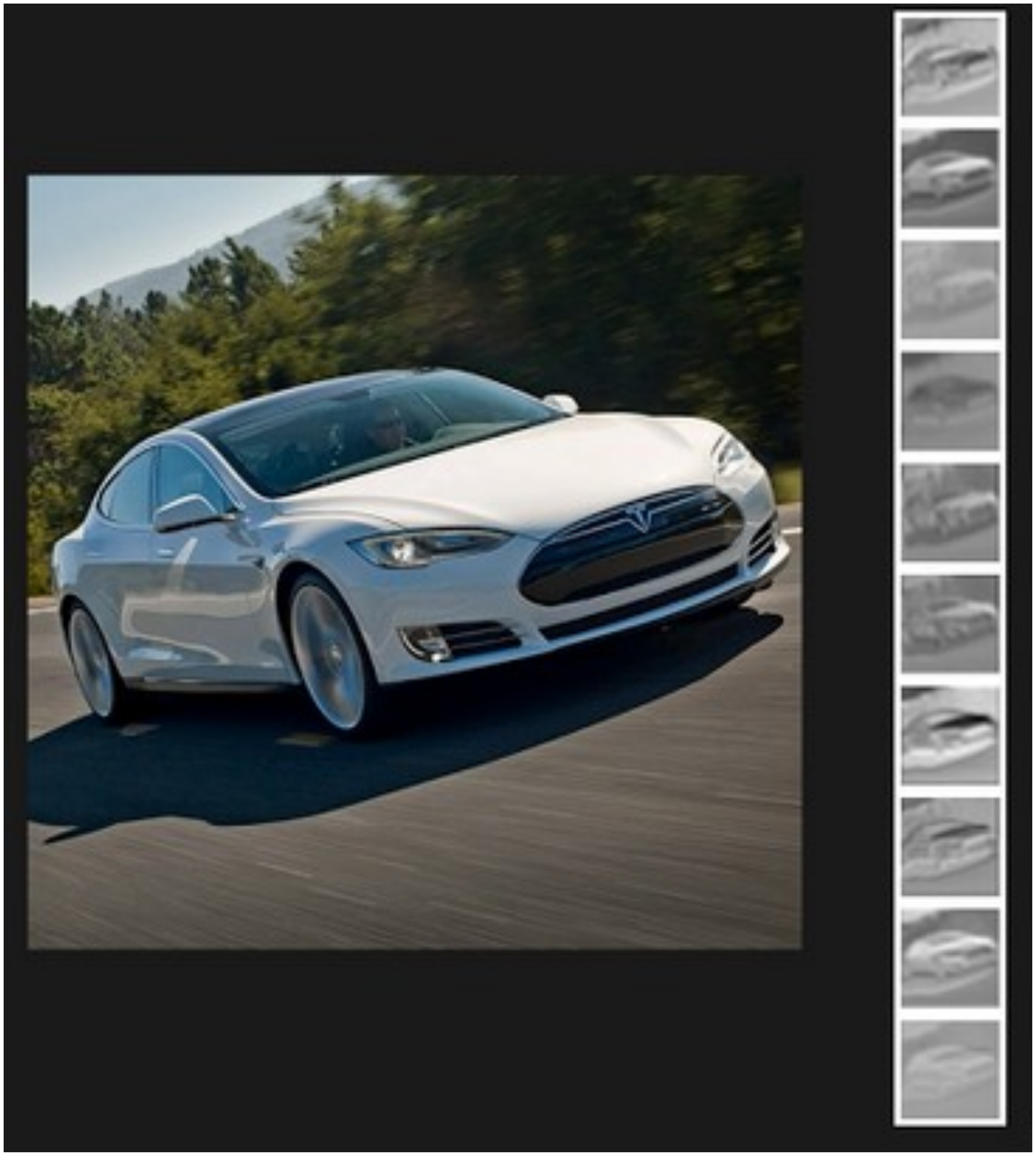


original image

Input to DNN classifier

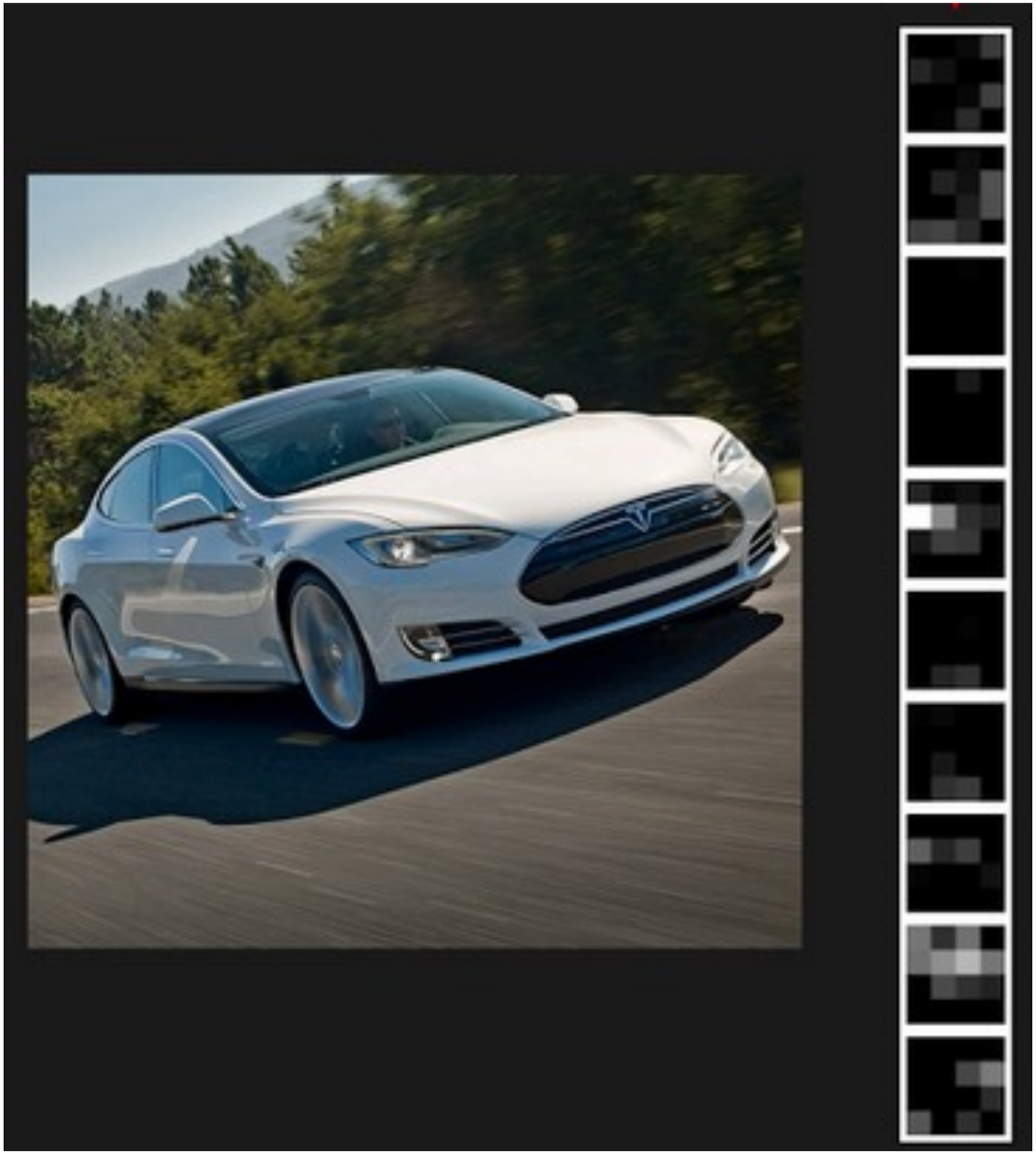


Convolutional Neural Networks



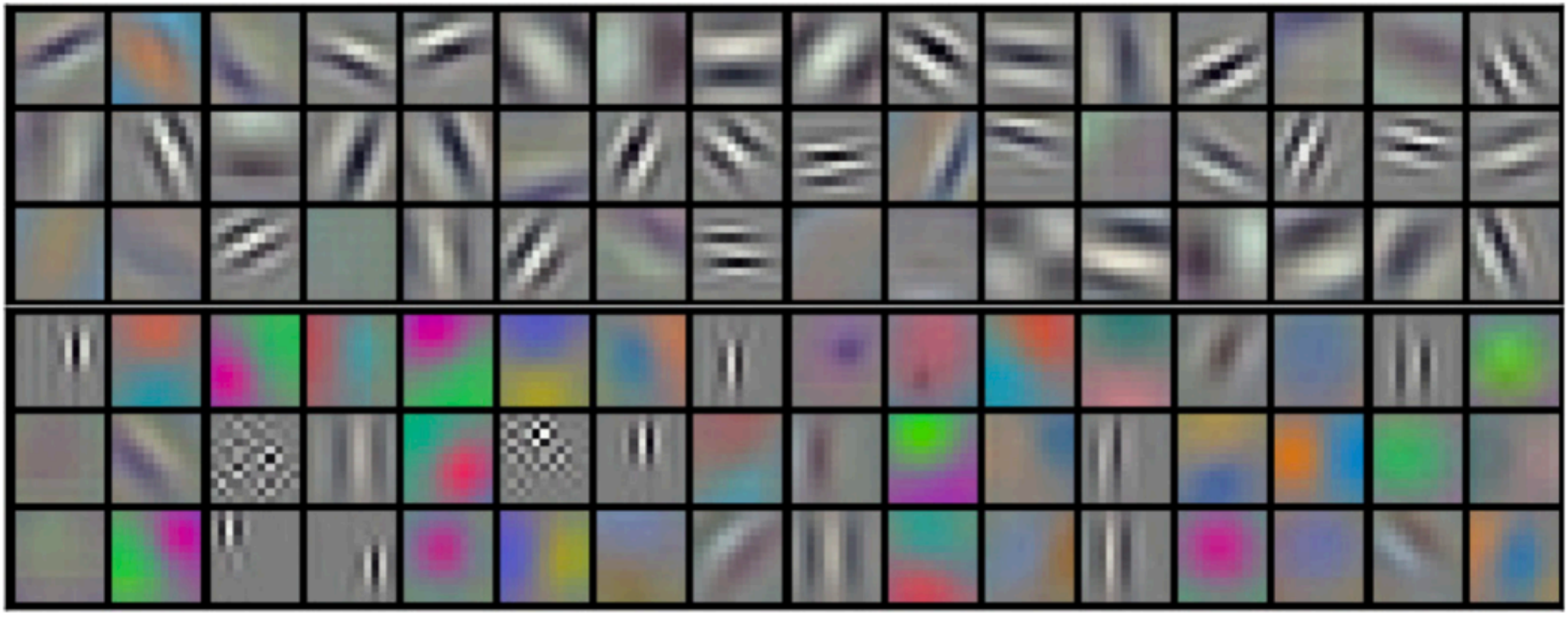
first convolutional layer: identify edges, surfaces, contrast differences, kinks …

Convolutional Neural Networks



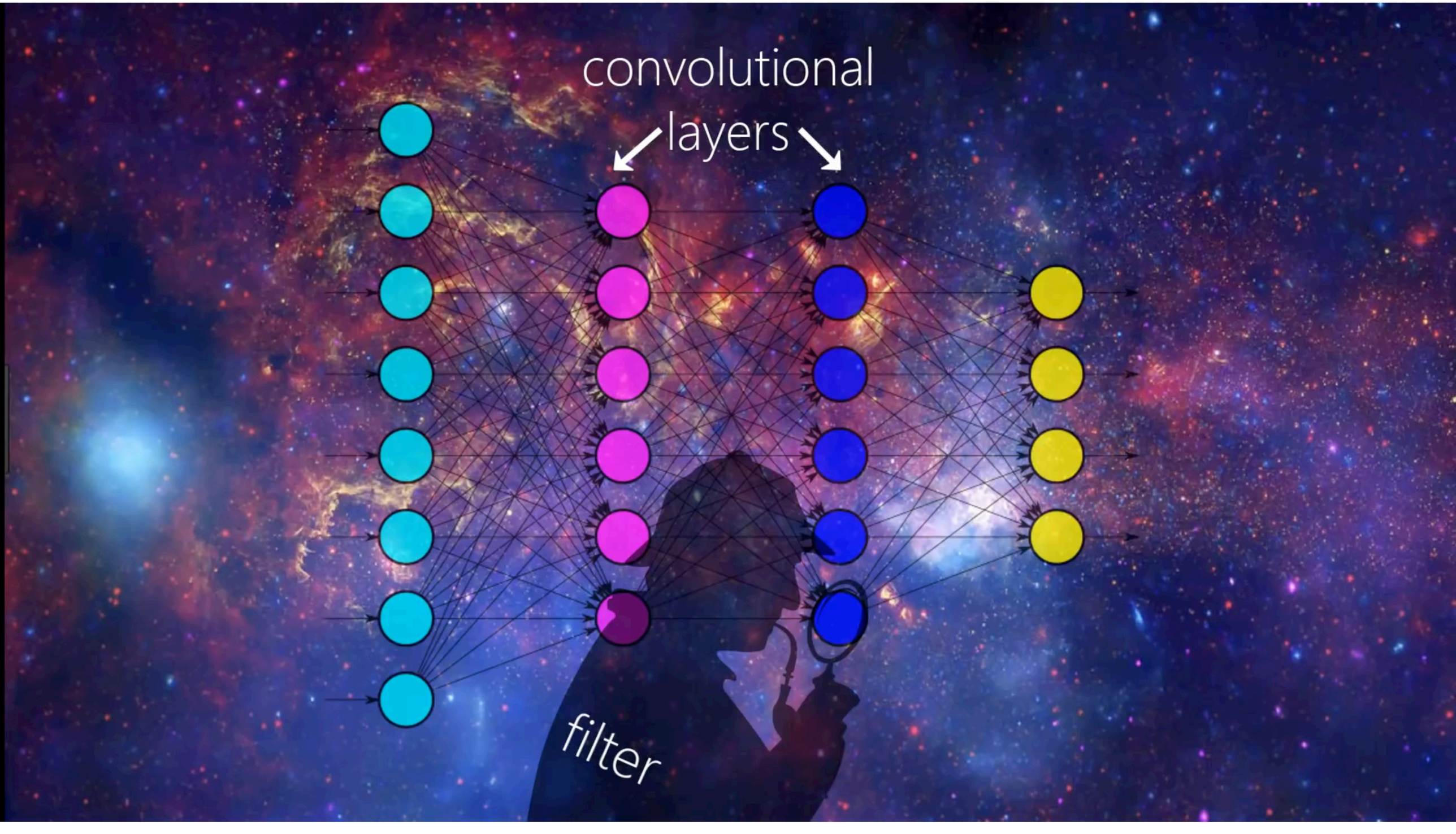
final convolutional layer: high-level features of ``car'' to be feed to the DNN classifier

Convolutional Neural Networks

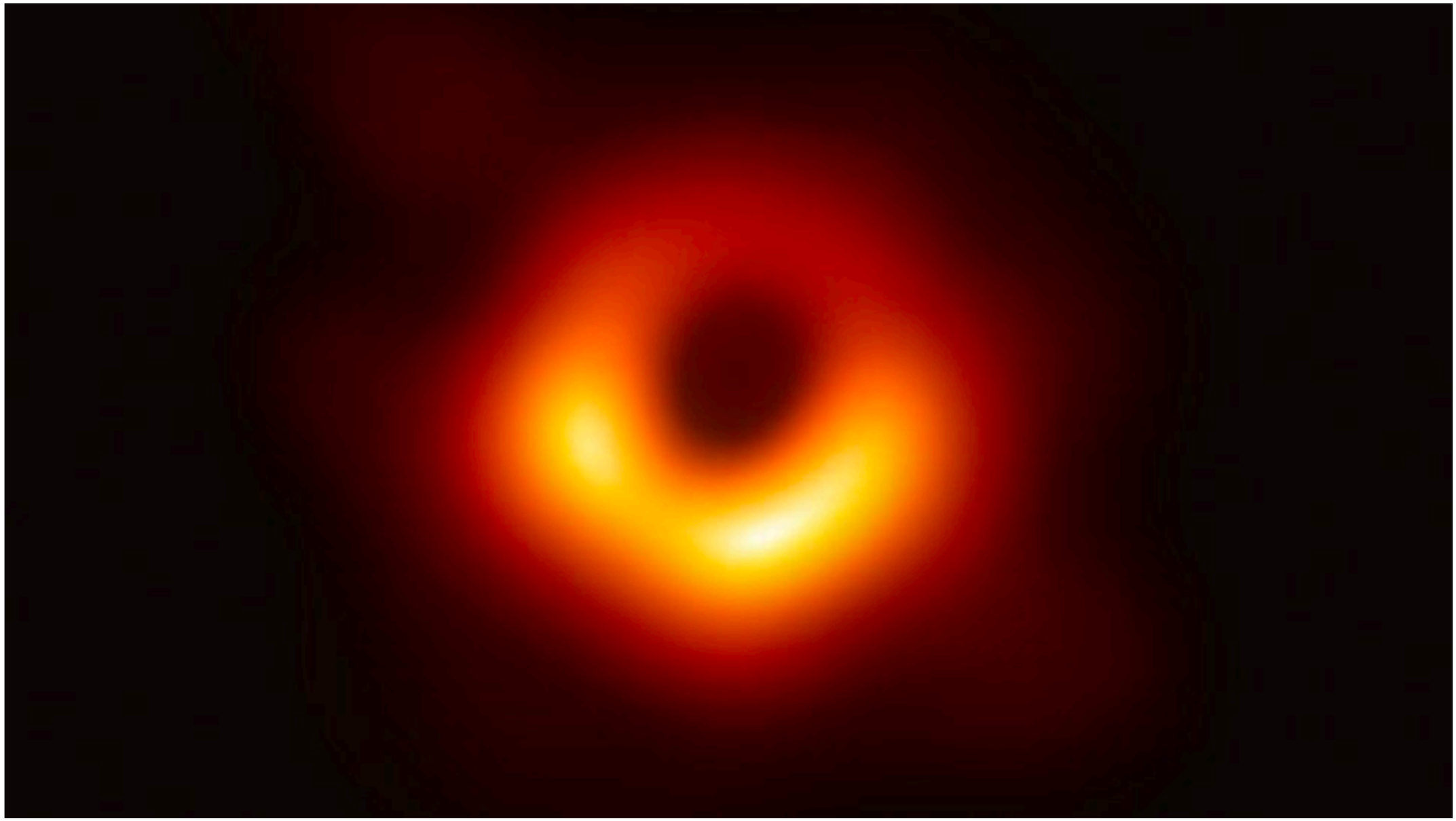


We can choose filters of a CNN to highlight specific colour combinations, lines in specific directions, edges or angles of a given size ….

How CNNs work

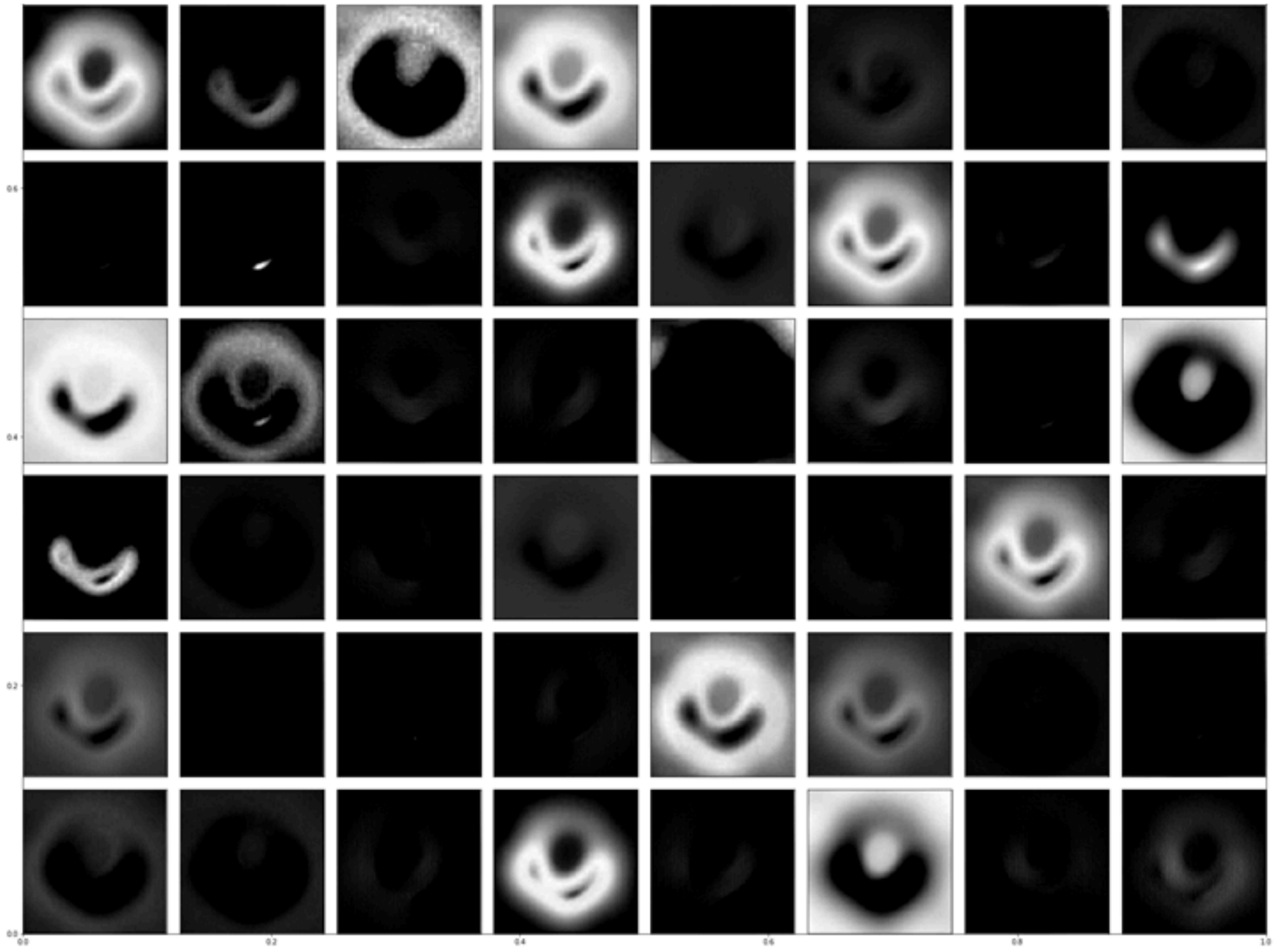


CNNs in Physics and Astronomy



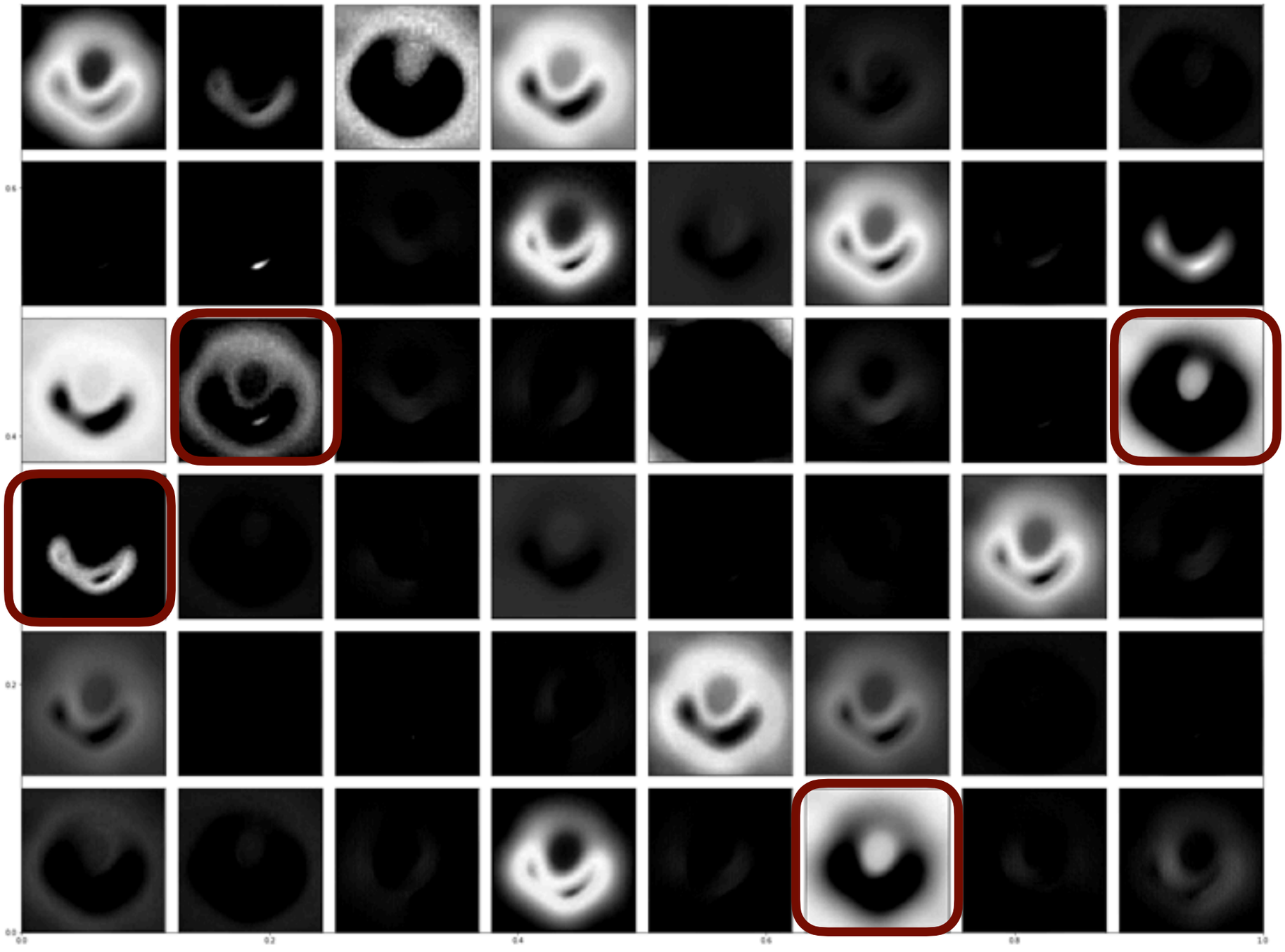
What is this image?

CNNs in Physics and Astronomy



This is how the black hole image is processed by the intermediate convolutional layers of a CNN

CNNs in Physics and Astronomy

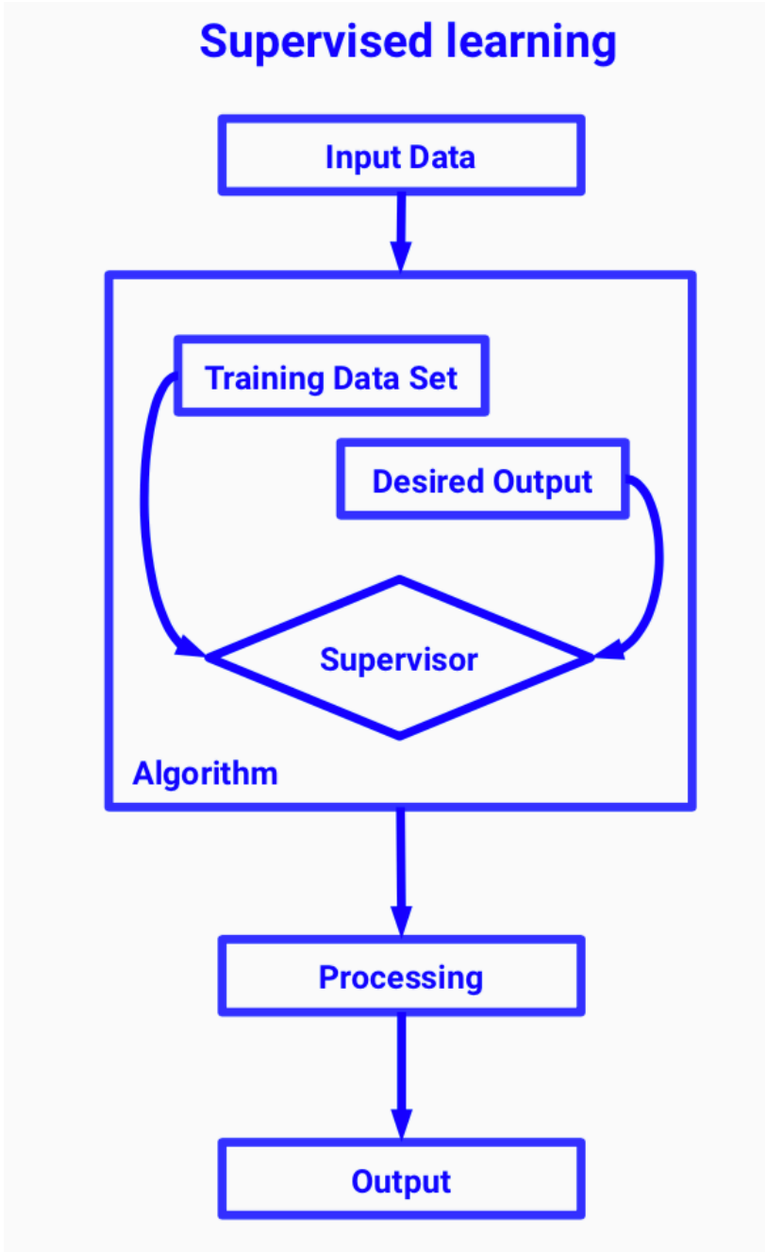


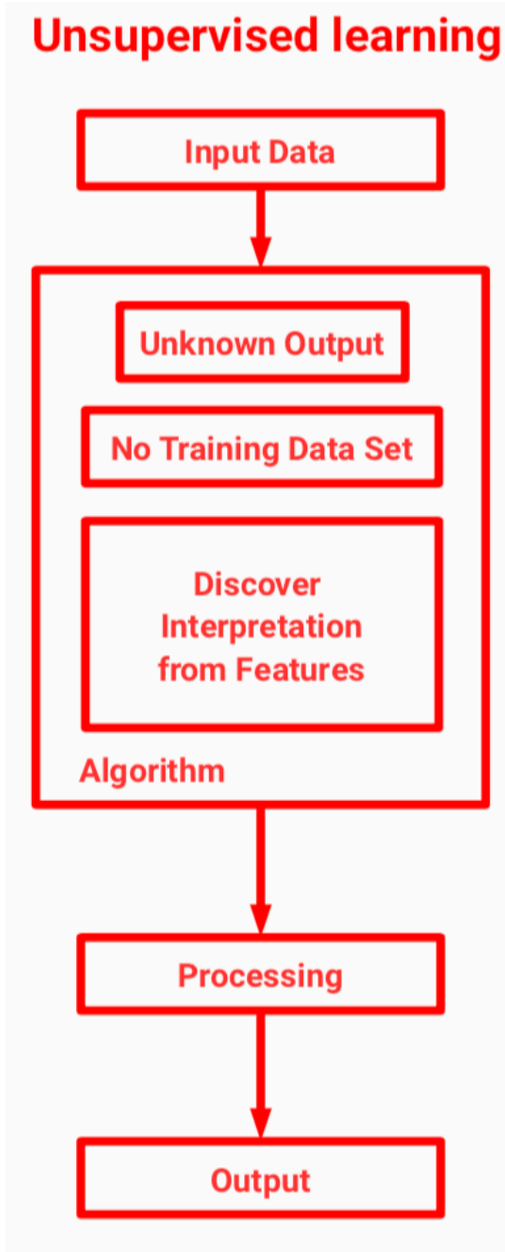
This is how the black hole image is processed by the intermediate convolutional layers of a CNN

Q: can you identify what is the task of each filter?

Reinforcement Learning

Supervised vs Unsupervised Learning





``Learn from examples”

``Find patterns in unlabelled data”

??

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)

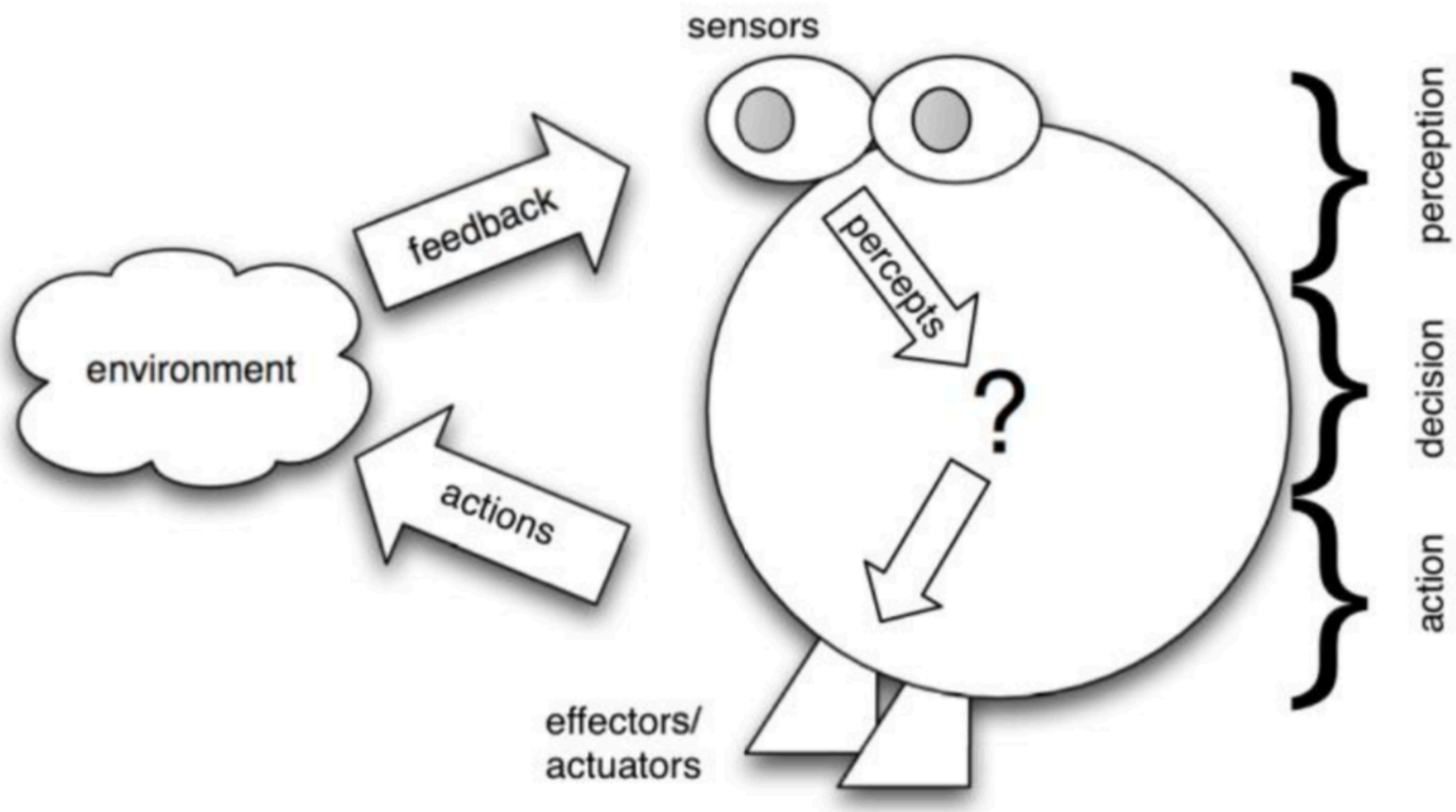
Unsupervised Learning: starting from a training dataset with **unlabelled examples,** $\{x_i\}_{i=1,N}$, produce a **model** that takes a sample as input and as output produces the solution of a practical problem, such as **clustering**, **dimensional reduction**, or **outlier detection**

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

Q: 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

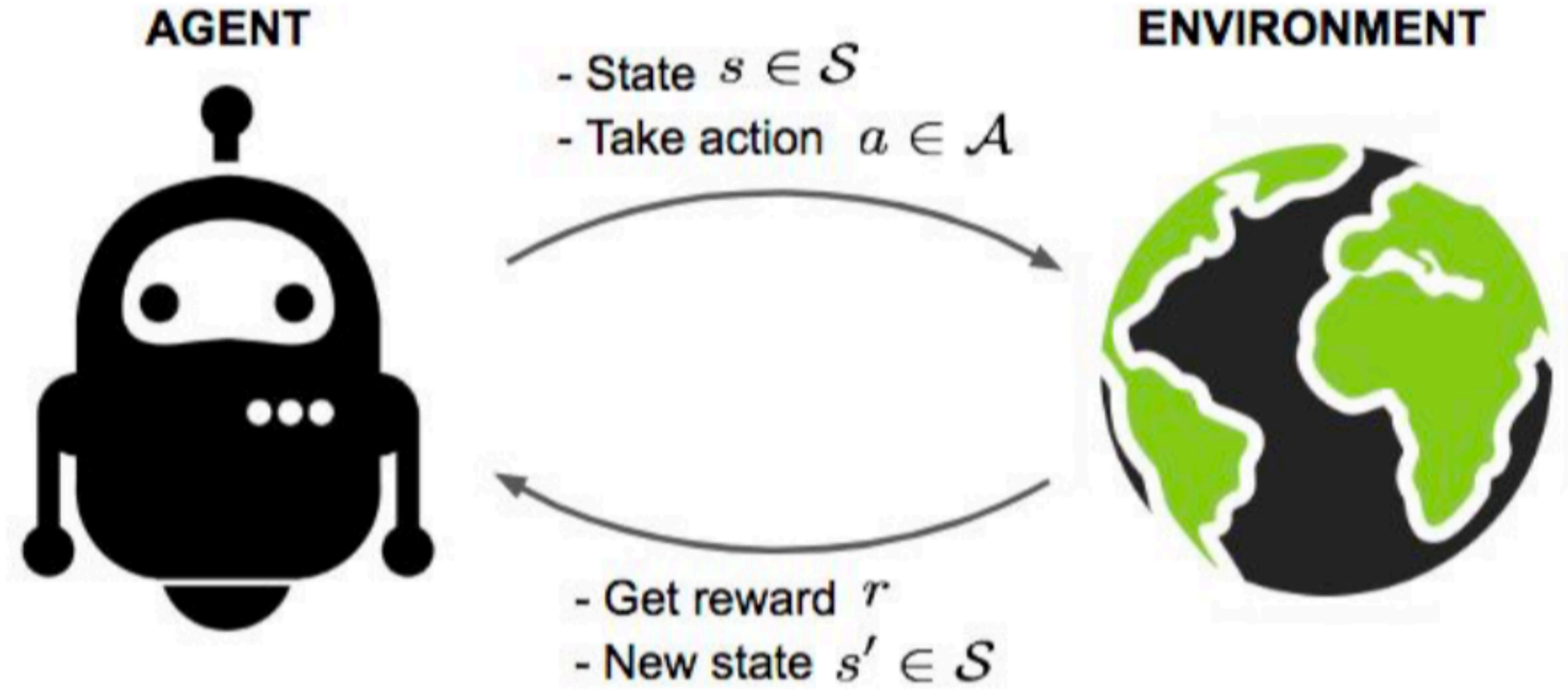
☑ An action is optimal if it **maximizes the expected average reward**

☑ 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!).

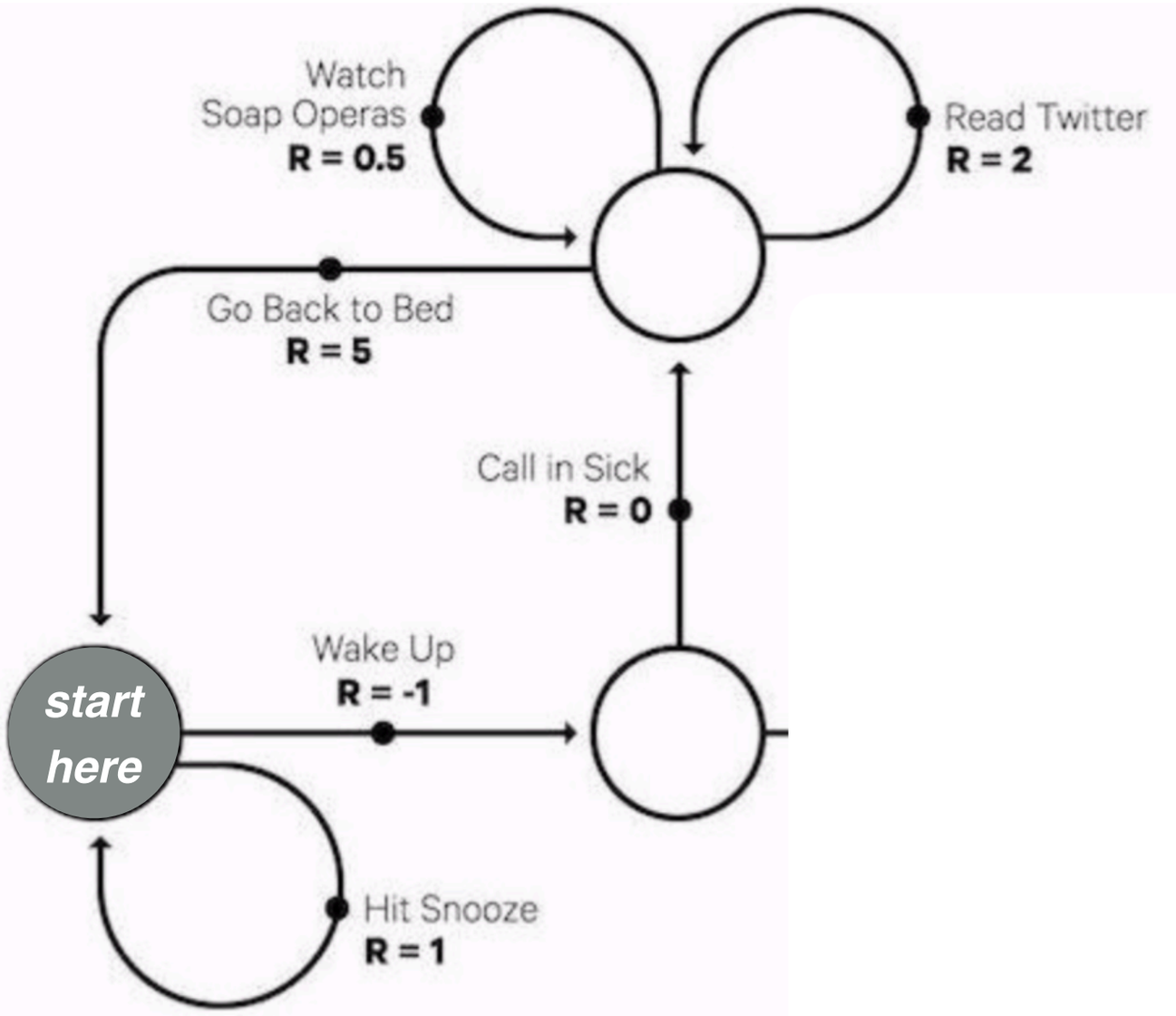


A Reinforcement Learning system

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

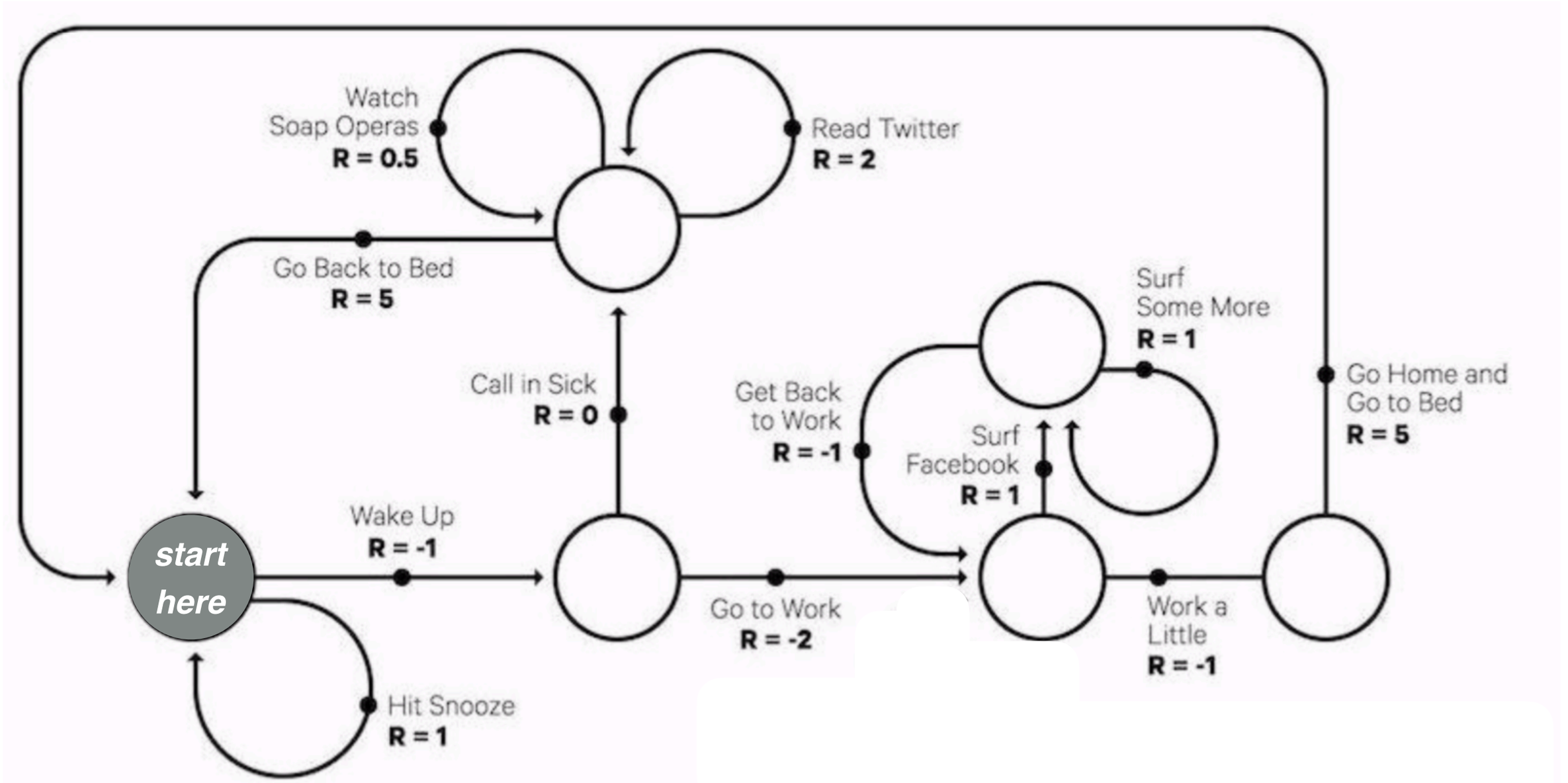
A Reinforcement Learning system



first explore a few possible ``actions'' that the ``agent'' takes and identify path to maximal reward

Q: what is the optimal outcome here?

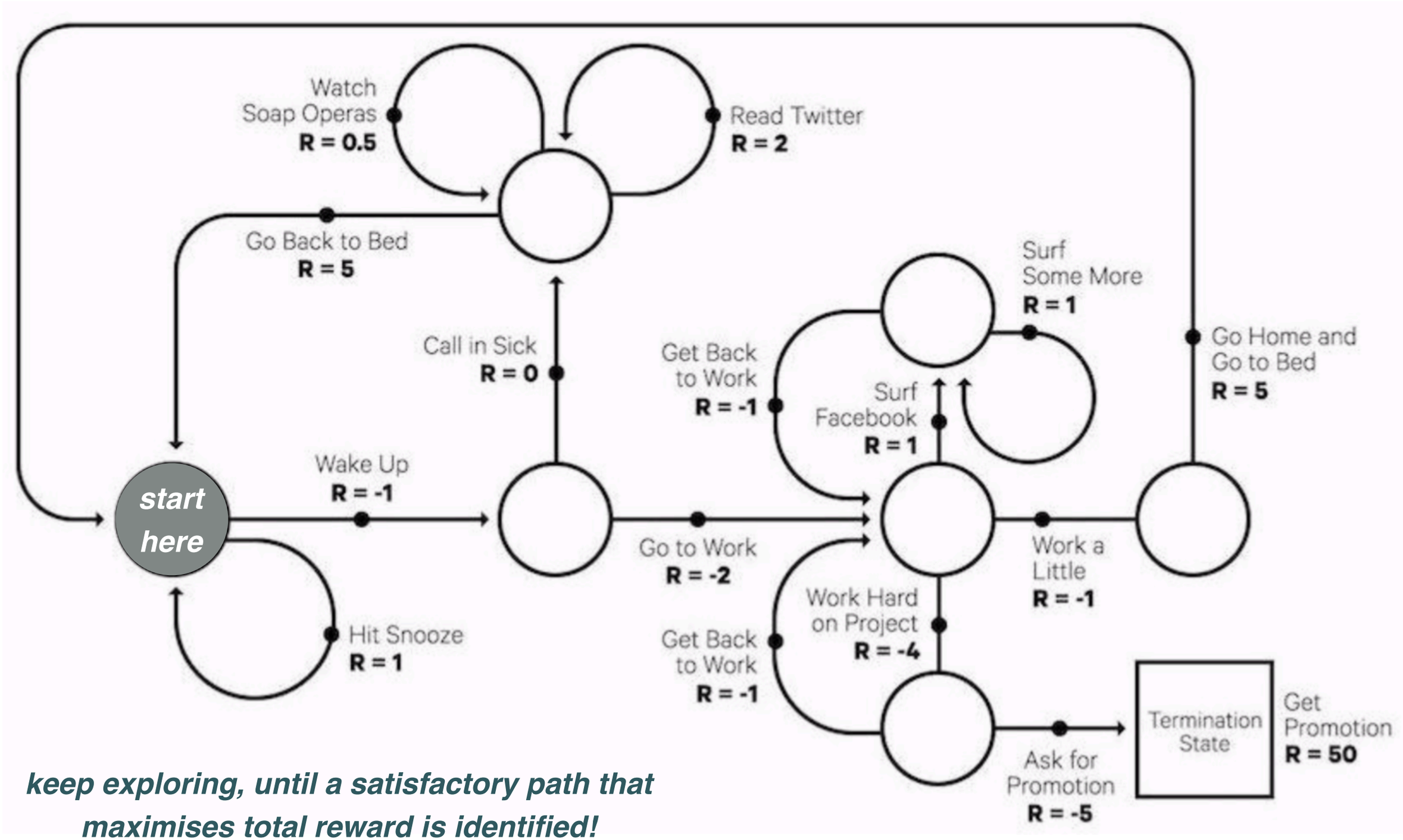
A Reinforcement Learning system



now explore some more possible action paths ….

Q: what is now the optimal outcome here?

A Reinforcement Learning system



keep exploring, until a satisfactory path that maximises total reward is identified!

Q: what is now the optimal outcome here?

Crucial to explore all possible outcomes

Reinforcement Learning for Gaming

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

BLOG POST
RESEARCH

04 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 Gaming

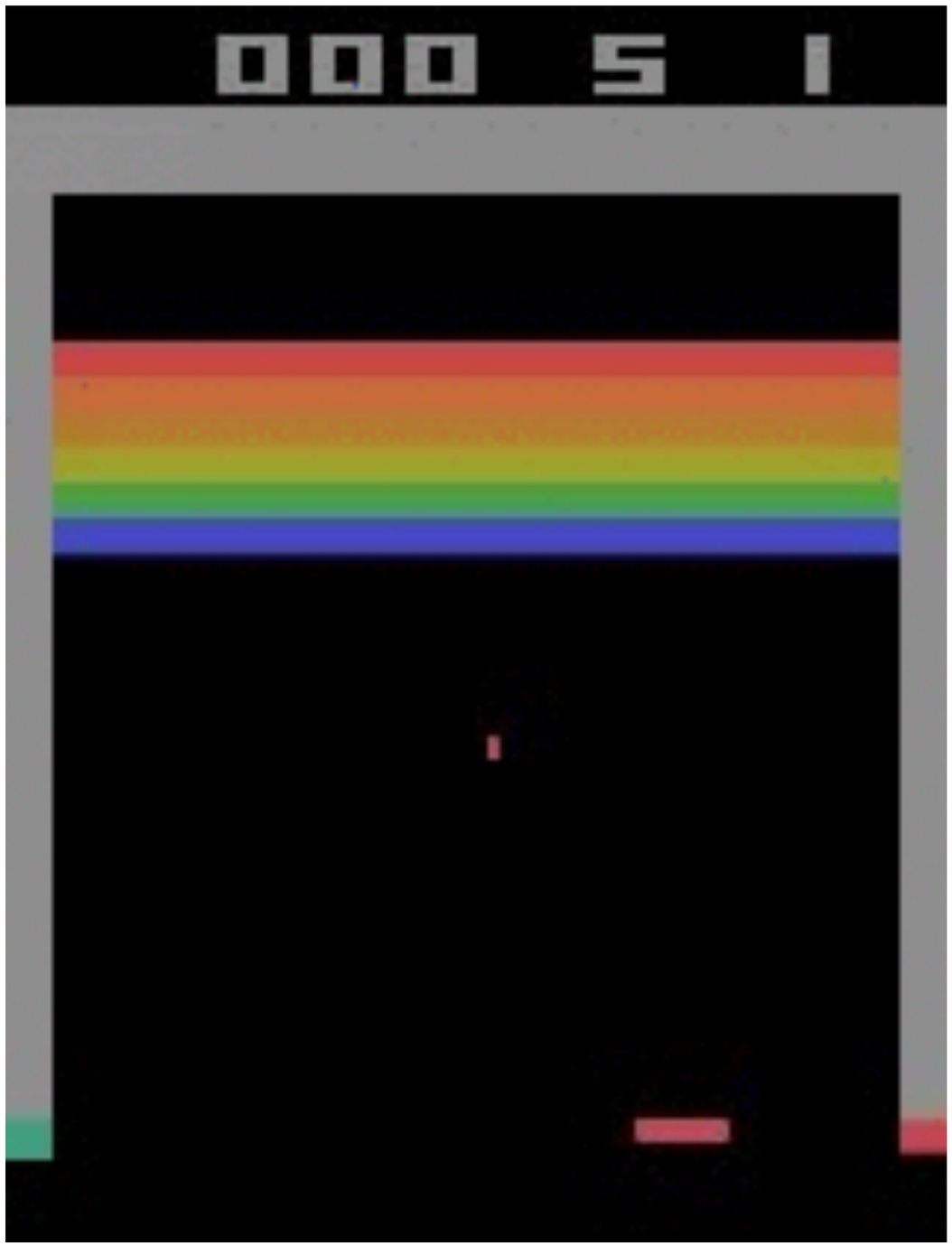
DeepMind AlphaStar: AI breakthrough or pushing the limits of reinforcement learning?

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

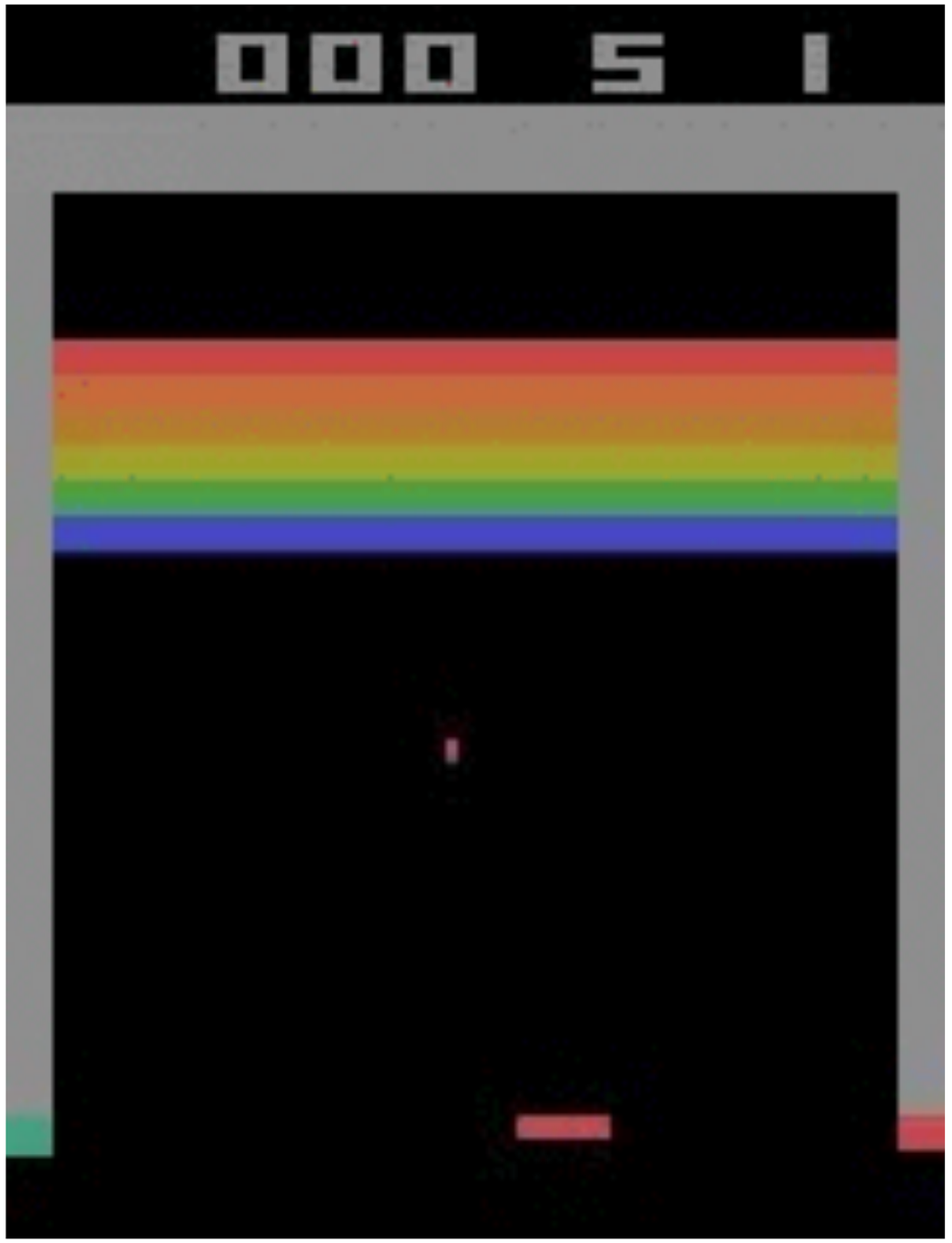
4 min read

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

Reinforcement Learning for Gaming



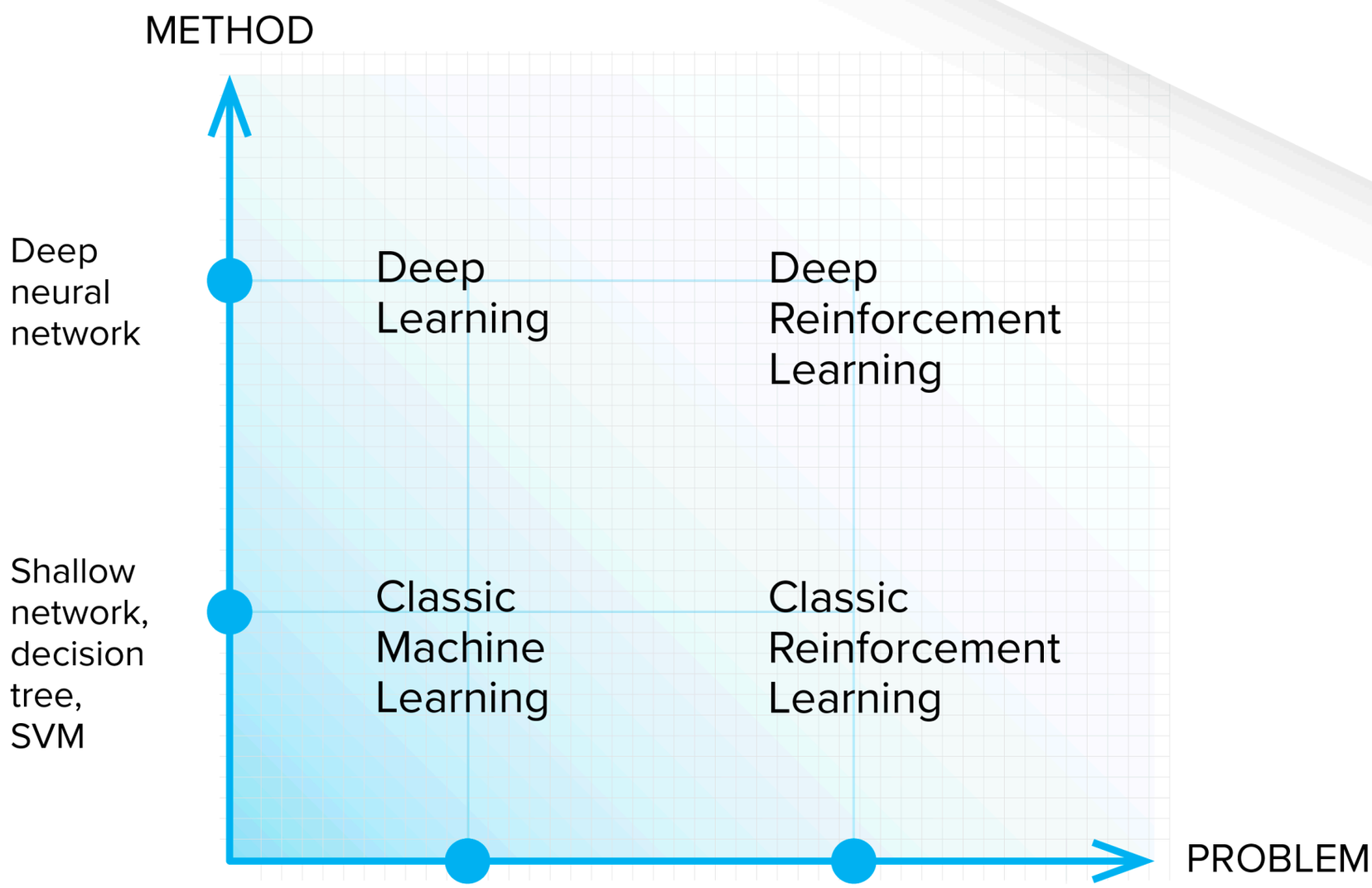
Initial Performance



After (RL) Training

Reinforcement Learning for Gaming

Reinforcement ML



Supervised Learning: find a predictive model that reproduces known answers

Reinforcement Learning: find a model that implements a successful action policy

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

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

reward

learning rate

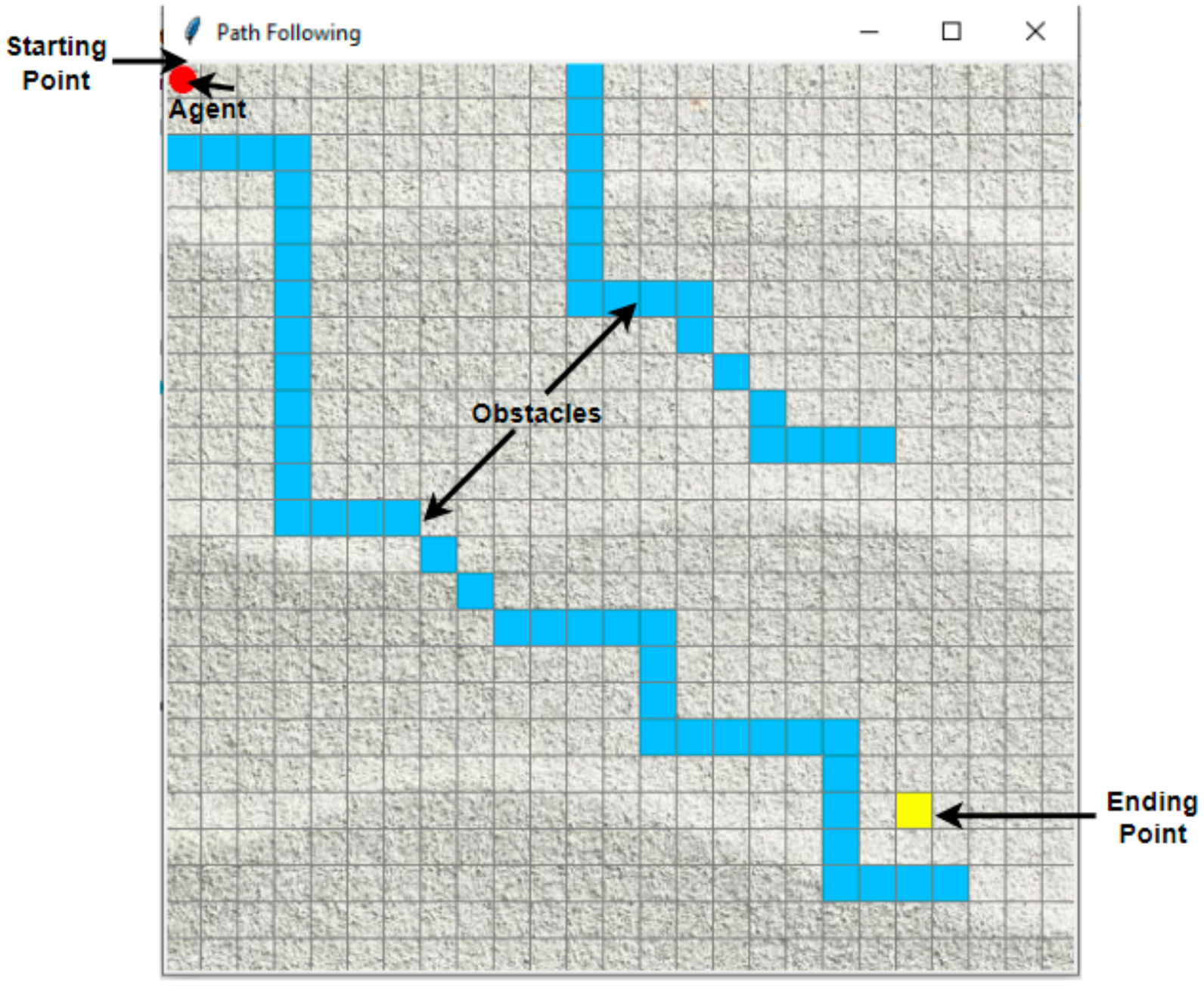
discount factor

(estimate of) future optimal value

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

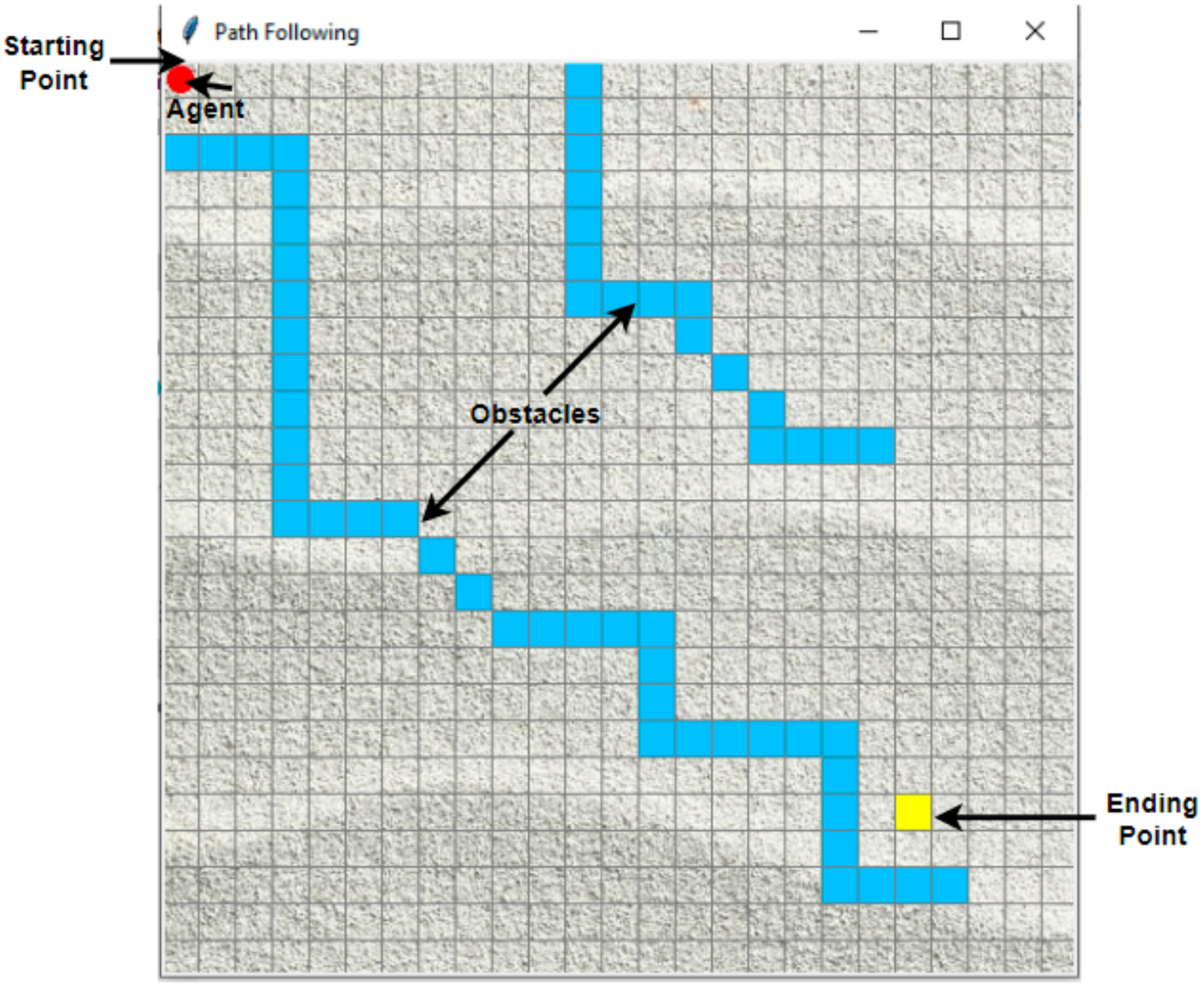
Q-learning

Let’s explain how Q-learning works with an specific example: train an agent to reach the **ending point of a maze** while avoiding the obstacles



Q-learning

First of all we need to initialise the **Q-table** $Q : S \times A \to \mathbb{R}$



Q1: how many possible actions does the agent have?

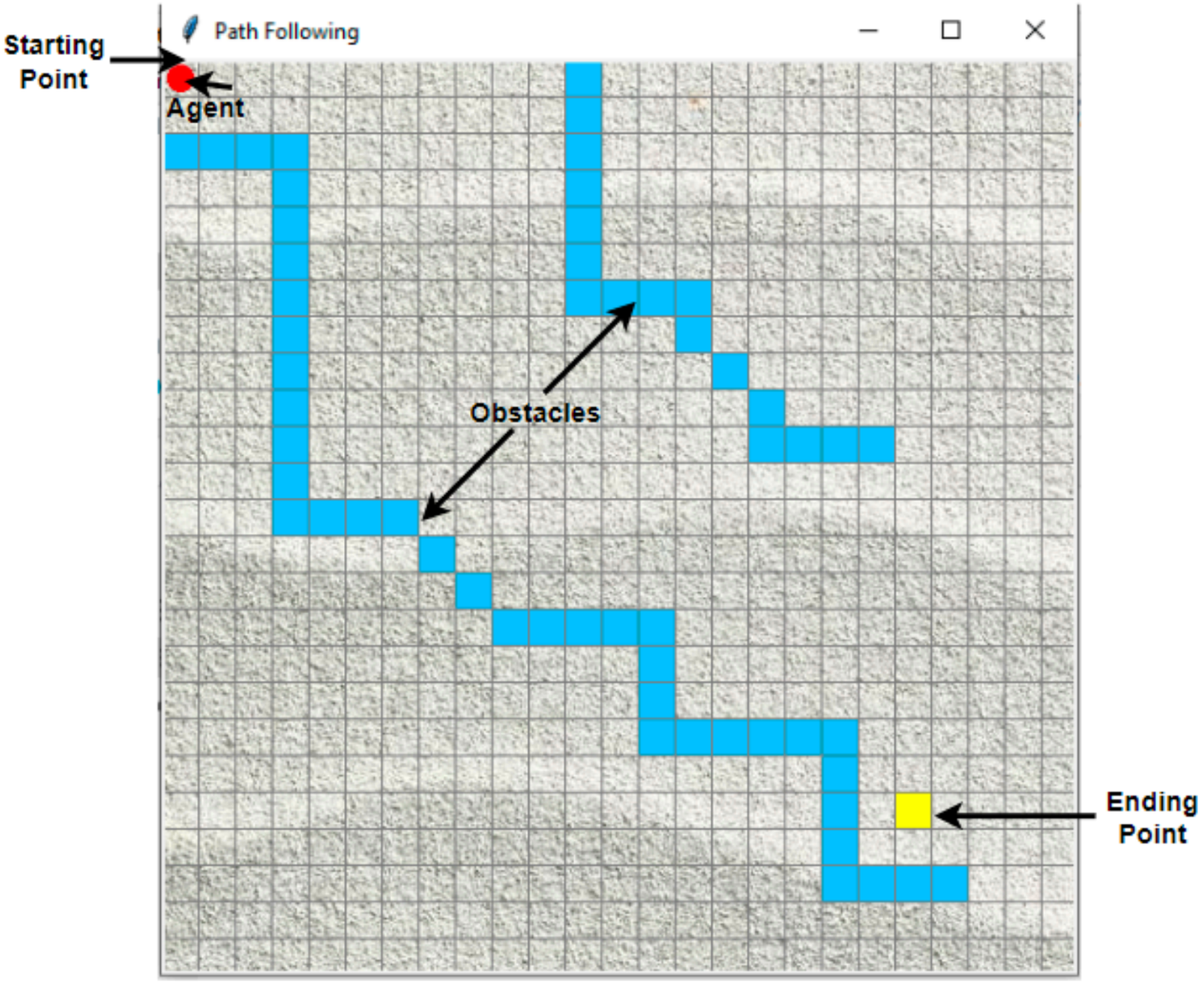
Q2: How do we characterise the state of the system?

recall: we don't have access to the environment!

Q3: what kind of rewards should we assign if we want to end up with an effective policy for the agent?

Q-learning

First of all we need to initialise the **Q-table** $Q : S \times A \rightarrow \mathbb{R}$



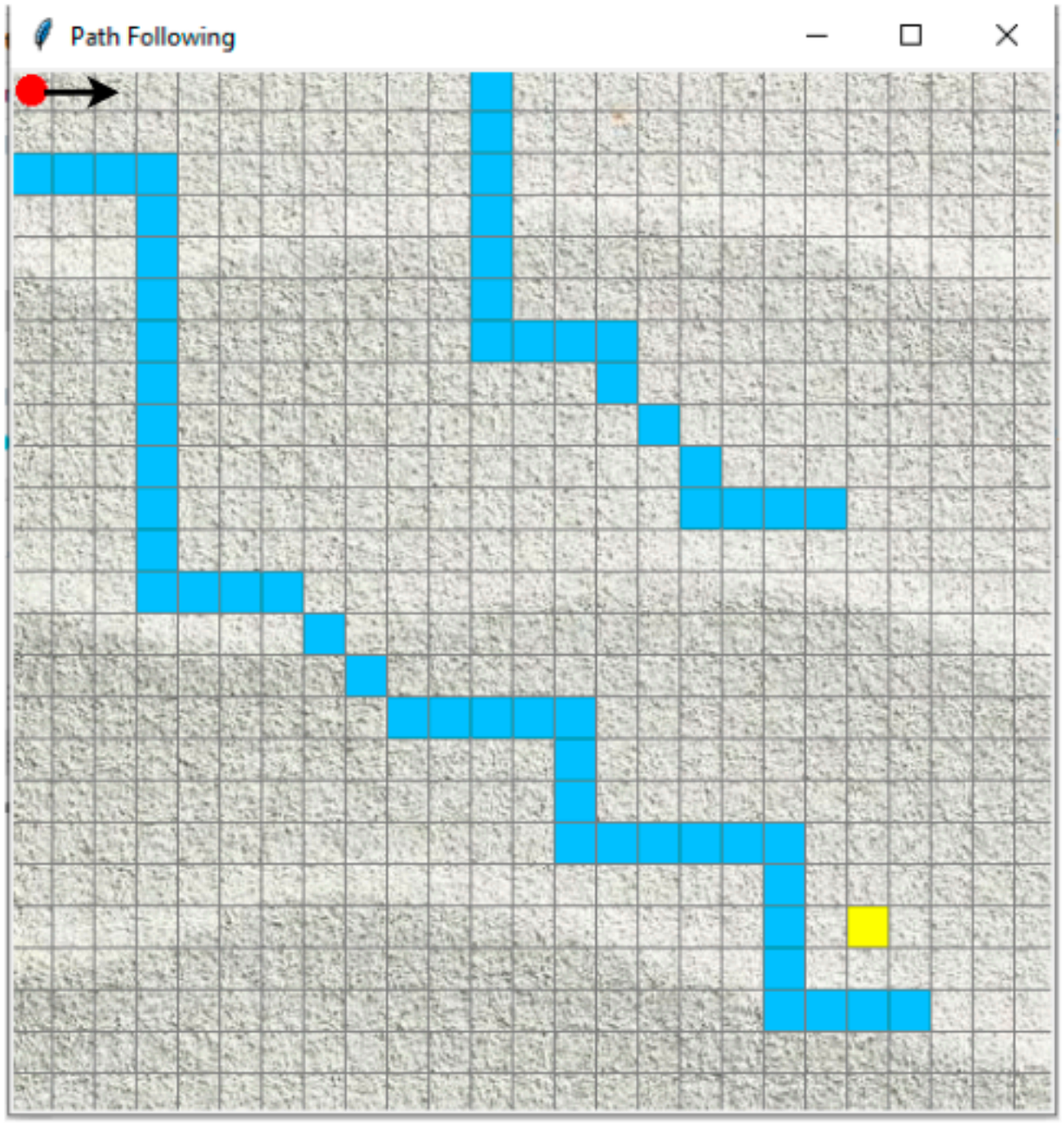
	up	down	left	right
start	0	0	0	0
idle	0	0	0	0
correct path	0	0	0	0
wrong path	0	0	0	0
end	0	0	0	0

note that ``absolute position in maze" is not part of the system state: why not?

The Q-table will become our policy once we explore actions and see what rewards we get

Q-learning

Take action and **update Q-table** $Q^{\pi}(s_t, a_t) = E\left[R_{t+1} + \gamma R_{t+2} + \ldots\right]\left(s_t, a_t\right)$



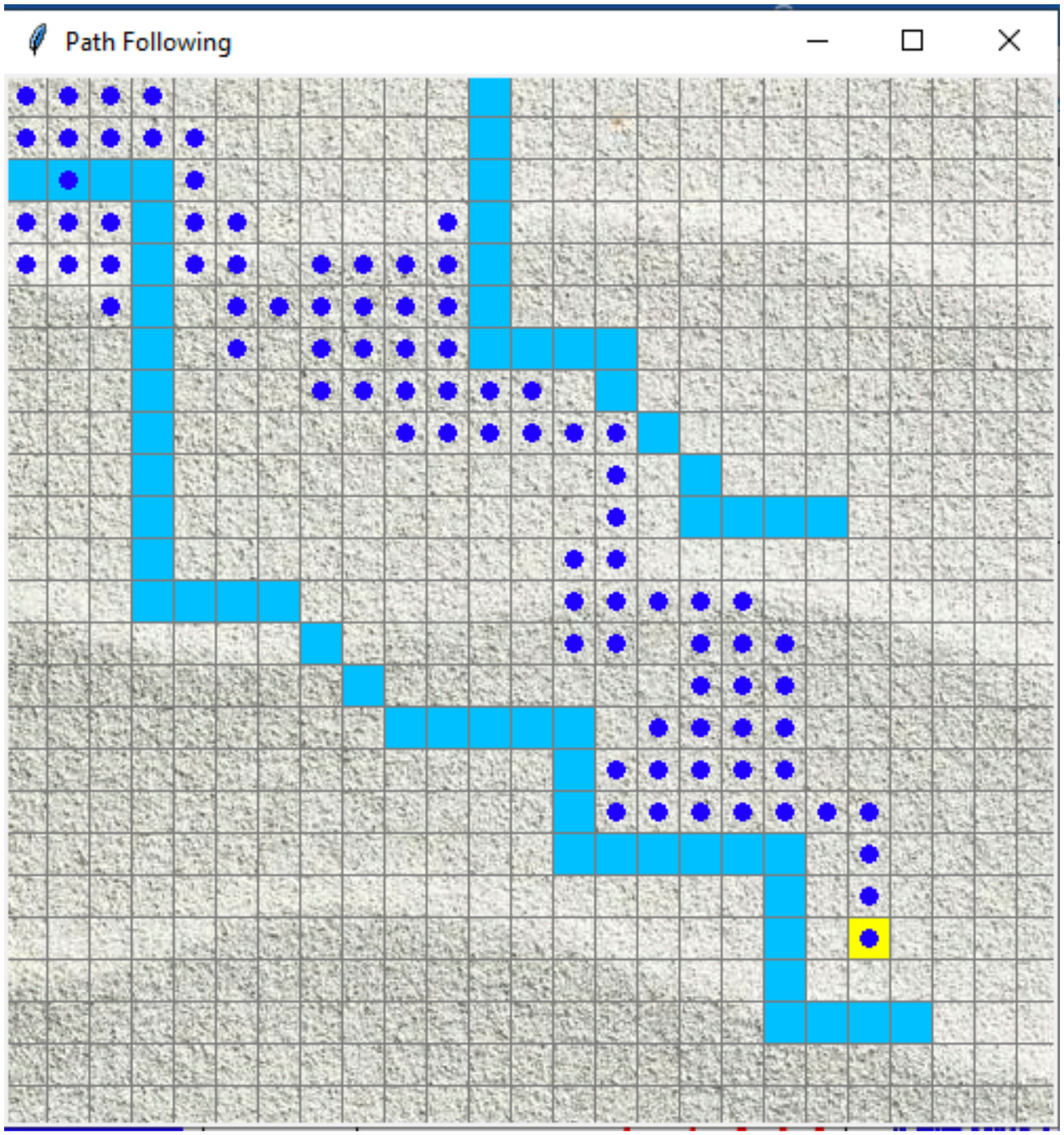
	up	down	left	right
start	0	0	0	**1**
idle	0	0	0	0
correct path	0	0	0	0
wrong path	0	0	0	0
end	0	0	0	0

when should we assign a negative reward?

positive reward, since we have not found any obstacles

Q-learning

Explore the environment and update the Q-table. Iterate until an **effective policy is found**

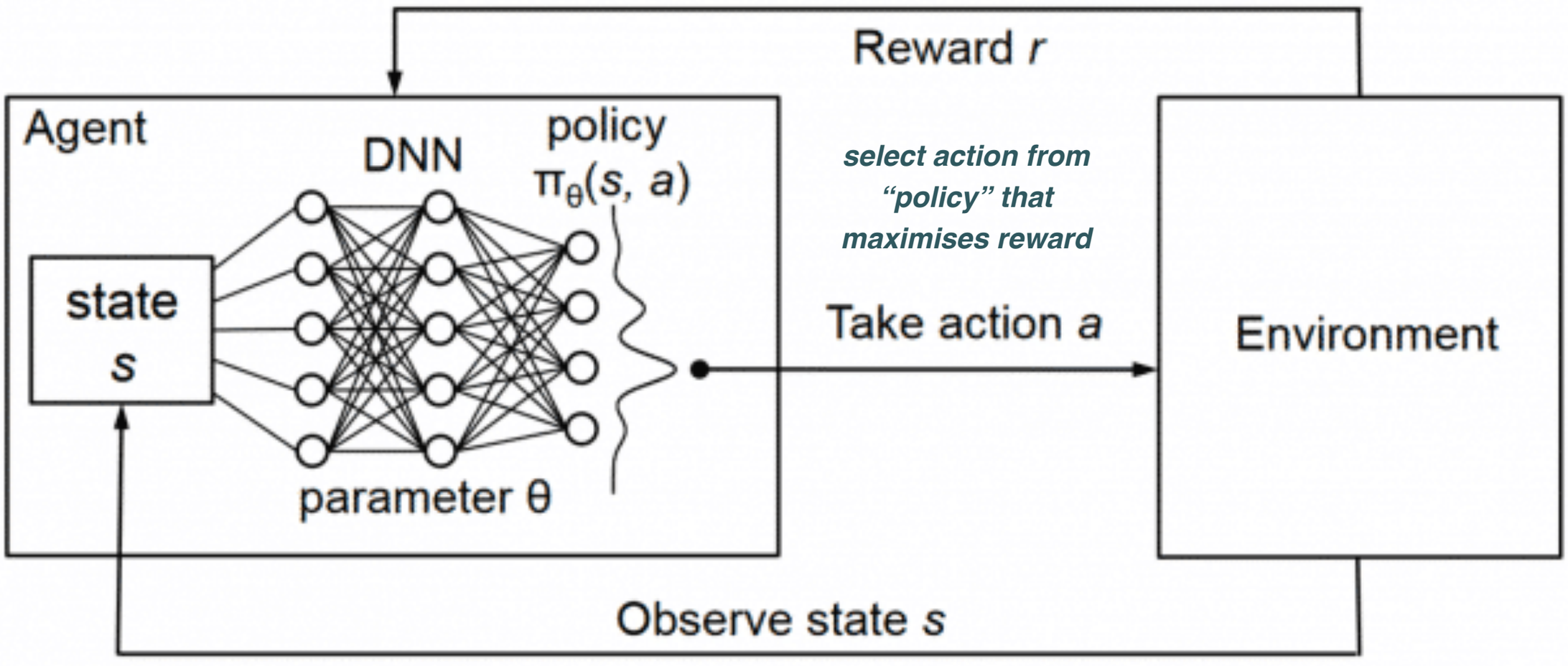


	up	down	left	right
start	0	0	0	**1**
idle	0	0	0	0
correct path	0	**50**	0	**22**
wrong path	**15**	0	0	**18**
end	0	0	0	**1**

after exploring the environment, the agent finds that a good policy is based on going only "right" and "down"

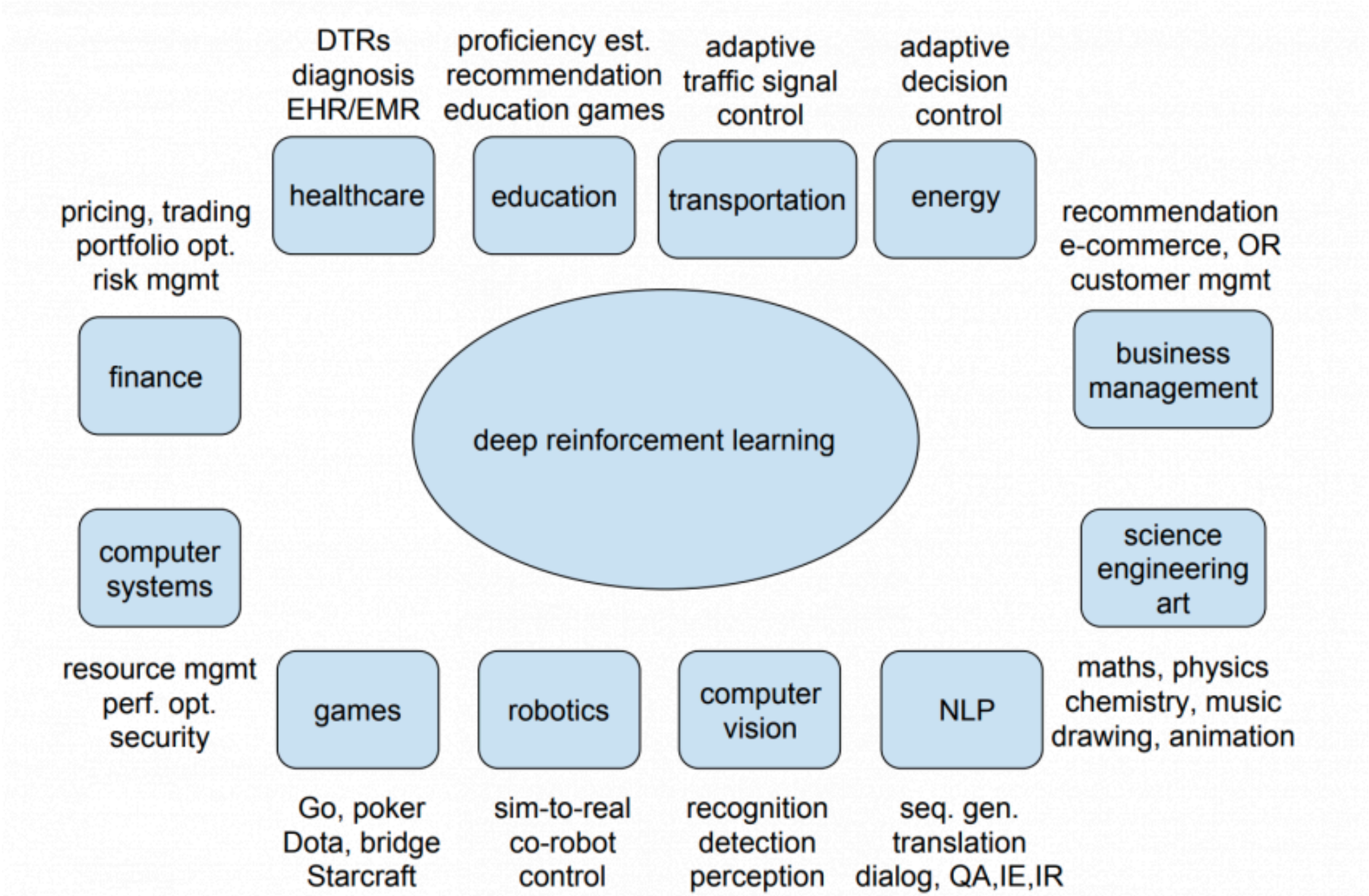
Deep Q-Networks

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**

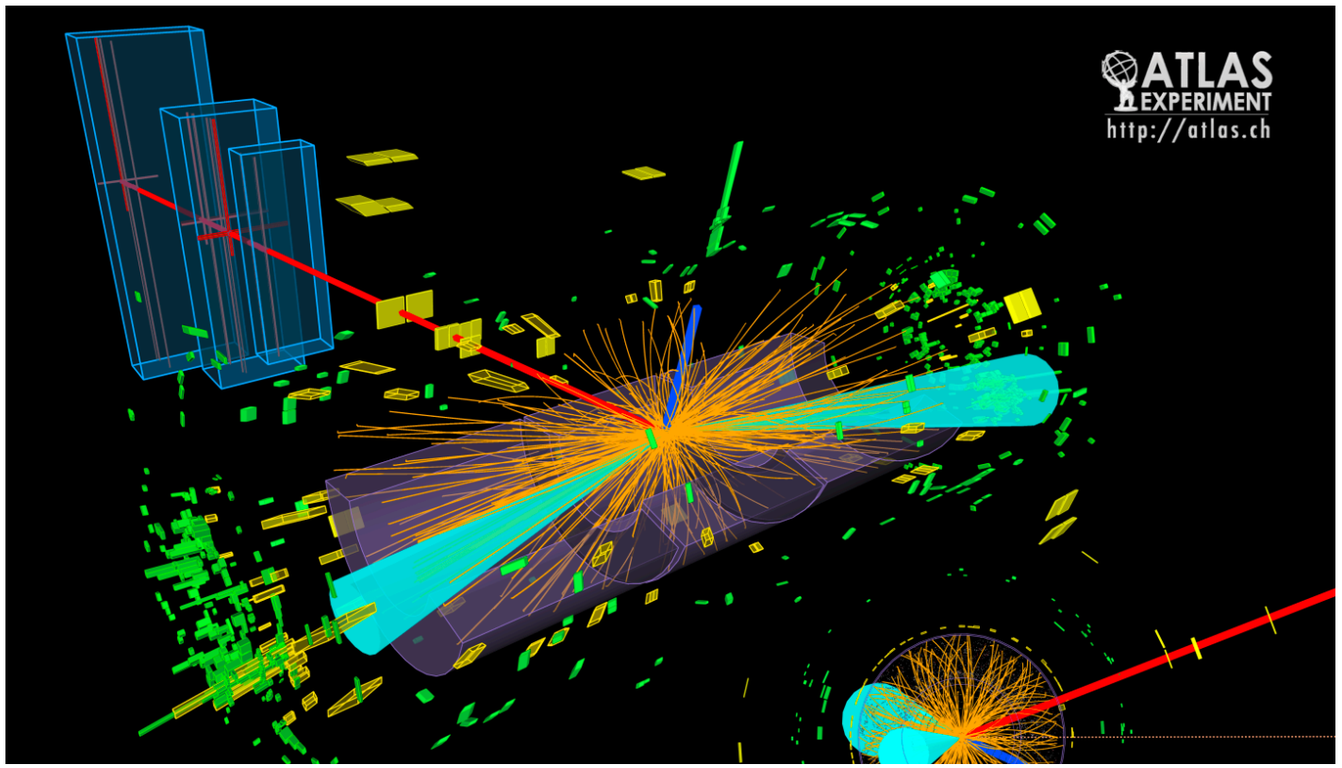


The DNN model parametrises the probability distribution associated to each possible action, given a state of the system

Deep RL applications

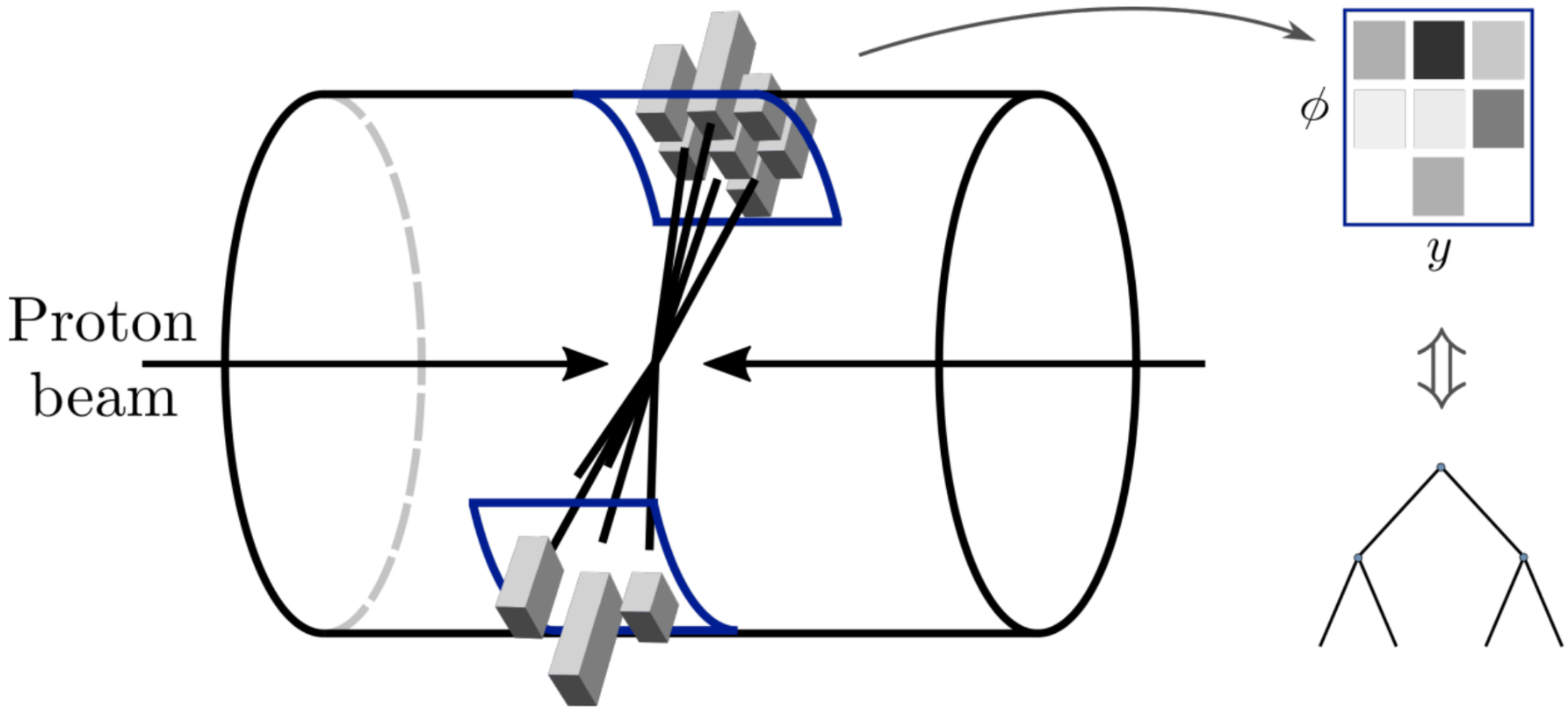


Reinforcement Learning in Physics



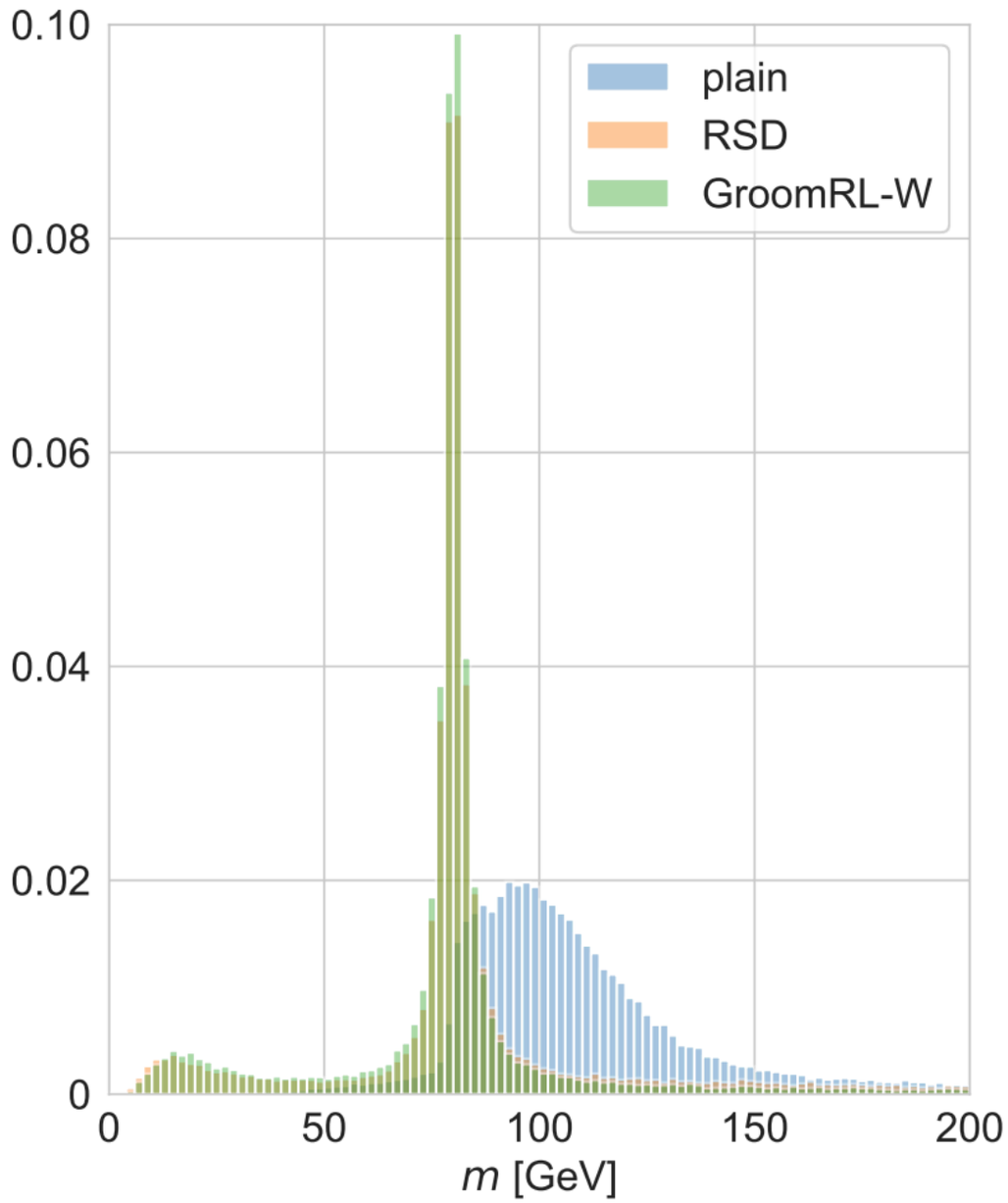
collisions at the LHC often result into sprays of collimated particles: jets

Jets can be mapped into images and processed with reinforcement learning

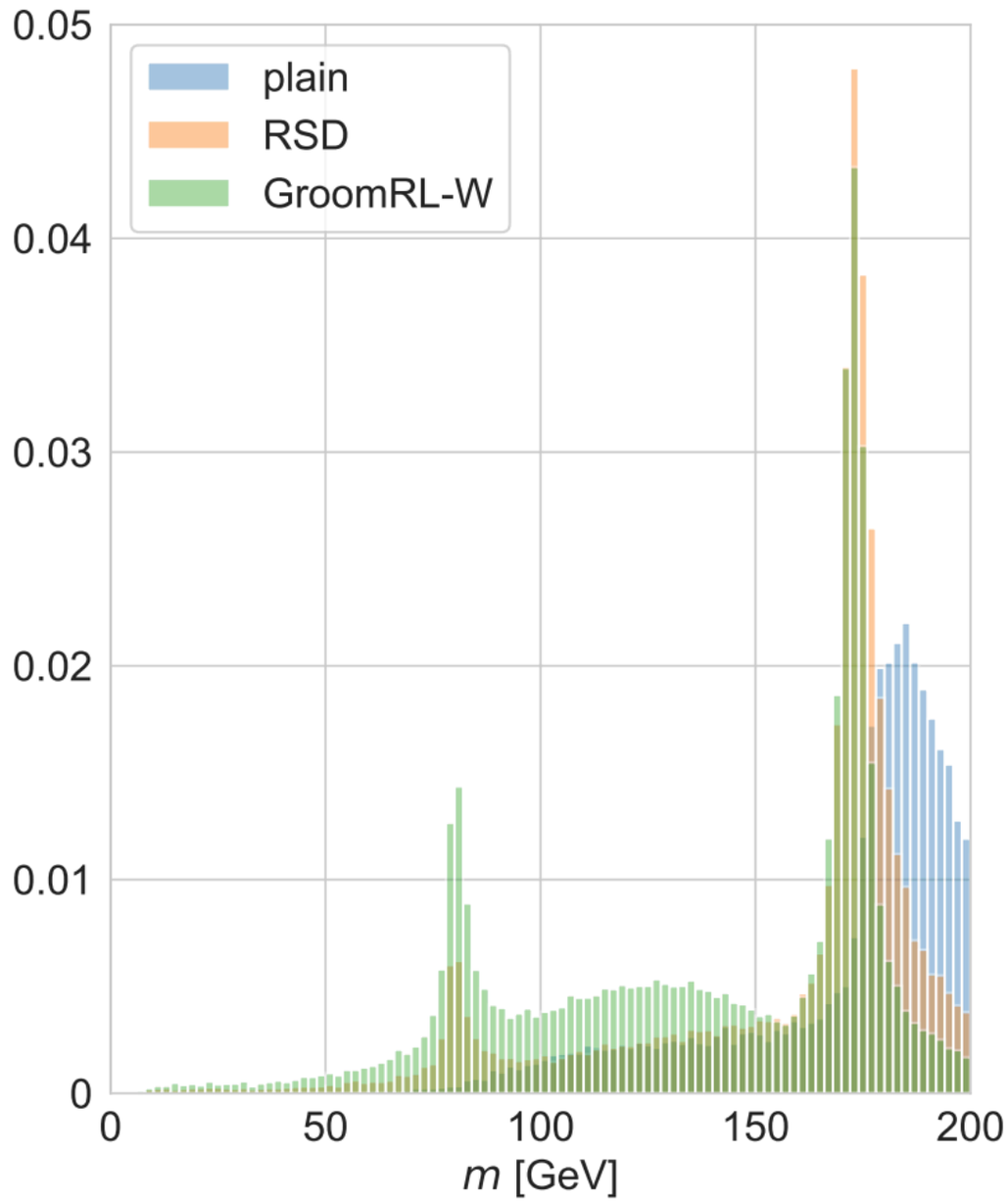


Goal: learn a policy that removes ``background'' particles from the jet

Reinforcement Learning in Physics



(b) W



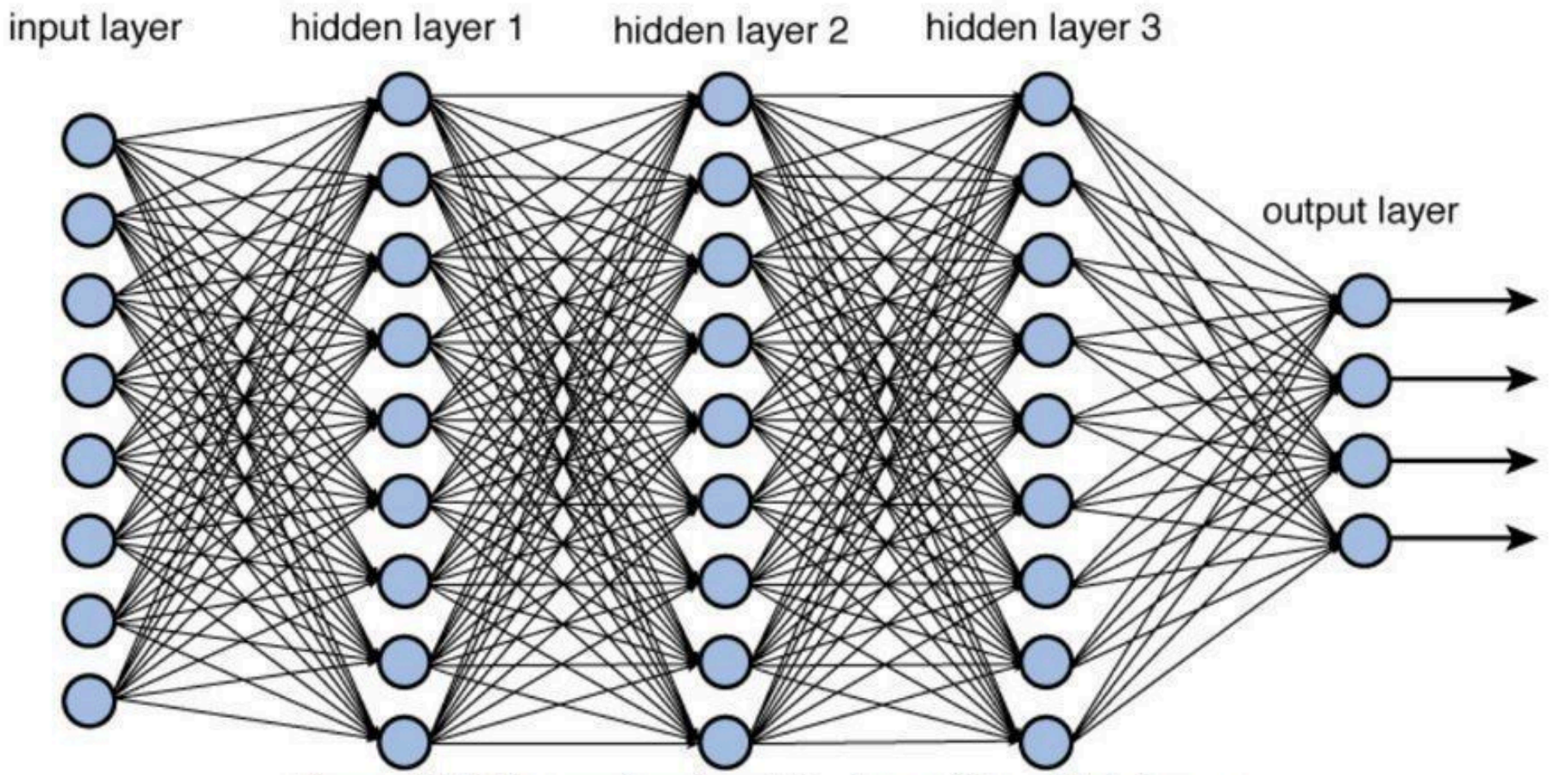
(c) top

Instead of defining a traditional ``jet grooming'' algorithm, construct a policy by defining rewards about **what are desirable traits of a well-groomed jet** (e.g. narrower peaks)

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**



$$E(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\left(\mathscr{F}_i^{(\mathrm{dat})} - \mathscr{F}_i^{(\mathrm{model})}(\hat{\boldsymbol{y}};\boldsymbol{\theta})\right)^2 \qquad \frac{\partial E(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = 0 \;\rightarrow \boldsymbol{\theta}_{\mathrm{opt}}$$

plus cross-validation etc

However for many applications we'd like a probabilistic interpretation of the NN output!

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

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

α, β are the model hyperparameters

conditional probability — *Gaussian Distribution* — *mean: NN output* — *variance*

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

prior probability →

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

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

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

probability of observing the dataset $\boldsymbol{X}$ given model parameters $\boldsymbol{\theta}$

prior distribution

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

knowledge of model parameters before we see the data $\mathbf{X}$

prior probability

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

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

prior probability

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

conditional probability

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

dataset

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

posterior probability

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

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

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

log-likelihood

from prior

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)

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

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

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

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

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

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

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

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!

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

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

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

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**

Summary

- Convolutional Neural Networks offer many advantages in **pattern recognition and image classification**

- Key advantage over fully-connected DNN is that they **respect the symmetries and locality** of the patterns we want to identify

- Reinforcement learning aims to **construct a policy** such that an agent maximises the total reward when operating in a partially unknown environment.

- **RL can be combined with DNN** to construct complex models of the probability distributions associated to the various possible actions of its policy