Topical Lectures on Machine Learning

Juan Rojo

VU Amsterdam & Theory group, Nikhef

Nikhef Topical Lecture program

Nikhef, Science Park, 9th June 2022

Schedule of these topical lectures

Wednesday 8th June

- Morning session I & II: General introduction to Machine Learning, including supervised (regression + classification) and unsupervised learning and (deep) neural networks (Juan Rojo)
- Afternoon session I & II: hands-on tutorials with Jupyter notebooks (supervised learning regression + classification) (Tanjona Rabemananjara)

Thursday 9th June

- Morning session I: advanced topics in machine learning: convolutional networks, adversarial learning, reinforcement learning (Juan Rojo)
- Morning session II: machine learning for LHCb/flavor physics (Jacco de Vries)
- Afternoon session I: hands-on tutorials with Jupyter notebooks (unsupervised learning) (Tanjona Rabemananjara)
- Afternoon session II: hands-on tutorials with Jupyter notebooks (ML for LHCb/flavor physics)

Friday 10th June

- Morning session I: CNNs in gravitational wave physics (Amit Reza)
- Morning session II: machine learning for LHC high-pT physics (Johnny Raine)
- Afternoon session I: hands-on tutorials with Jupyter notebooks (CNNs in GWs) (Amit Reza)
- Afternoon session II: hands-on tutorials with Jupyter notebooks (ML ATLAS) (Johnny Raine)
- Drinks!

Regression in supervised ML

problems in **Supervised Machine Learning** are defined by the following ingredients:

(1) Input dataset: $\mathscr{D} = (\boldsymbol{X}, \boldsymbol{Y})$

(2) Model: $f\left(\boldsymbol{X}, \boldsymbol{\theta}\right)$

(3) Cost function: $C\left(\boldsymbol{Y}; f(\boldsymbol{X}; \boldsymbol{\theta}\right)$

The cost function measures how well the model (for a specific choice of its parameters) is able to describe the input dataset

Fitting the model means determining the values of its parameters which **minimise the cost function**

$$\left.\frac{\partial C\left(\boldsymbol{Y}; f(\boldsymbol{X}; \boldsymbol{\theta}\right)}{\partial \theta_i}\right|_{\theta=\theta_{\text{opt}}} = 0$$

Regression in supervised ML

What is the best strategy to **determine the model parameters**?

Seems a silly question, surely those are simply **minimum of cost function?**

$$\hat{\theta} = \arg\min_{\theta}\left\{ \sum_{i=1}^{n} \left(y_i - f_\alpha(x_i; \boldsymbol{\theta}_\alpha)\right)^2 \right\}$$

However this is in general **not the case**, because:

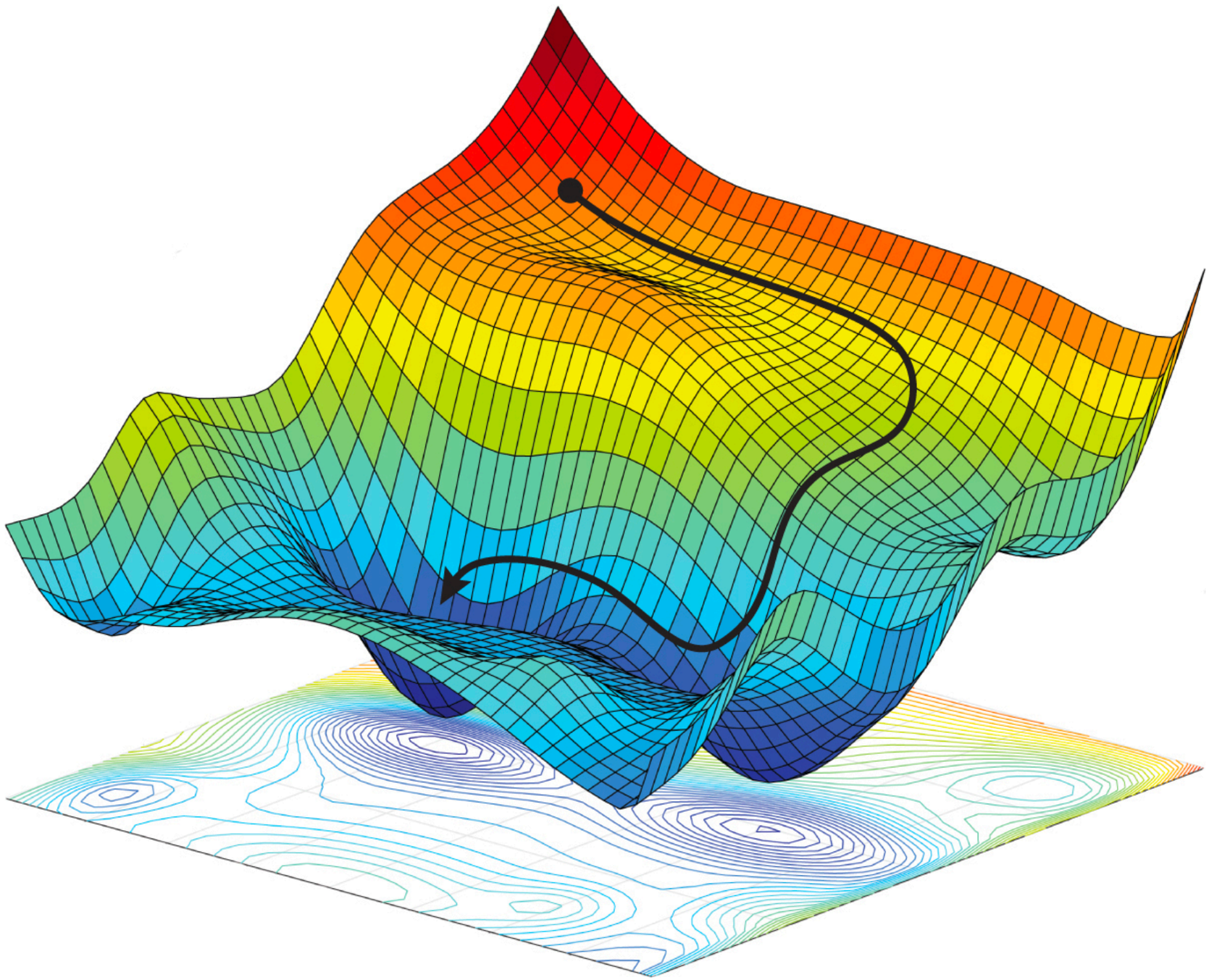
- Real world data is **noisy**: we want to **learn the underlying law**, not the statistical fluctuations

- More than fitting the data, our real goal is to create a model that **predicts future/ different data:** we need figures of merit outside the training dataset!

- To ensure that our model describes the underlying law (and thus one can safely generalise) rather than the noise, a **regularisation procedure** needs to be used

Gradient Descent

basic idea: iteratively adjust the model parameters in the direction where the **gradient of the cost function** is large and negative (**steepest descent direction**)



Adaptative Gradient Descent

Adaptative GD varies the learning rate to reflect the **local curvature** of the parameter space without the need to evaluate the Hessian matrix

RMSprop: keep track also of the **second moment of gradient**, similar as how the momentum term is a running average of previous gradients

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \frac{\boldsymbol{g}_t}{\sqrt{\boldsymbol{s}_t + \epsilon}}$$

regulator to avoid numerical divergences

$$\boldsymbol{g}_t = \nabla_\theta E(\boldsymbol{\theta})$$

gradient at iteration t

$$\boldsymbol{s}_t = \beta \boldsymbol{s}_{t-1} + (1-\beta)\boldsymbol{g}_t^2$$

controls averaging time of 2nd moment of gradient

$$\boldsymbol{s}_t = \mathbb{E}[\boldsymbol{g}_t^2]$$

2nd moment of gradient (averaged over iterations)

$$\beta = 1 \rightarrow \boldsymbol{s}_t = \boldsymbol{s}_{t-1}$$

$$\beta = 0 \rightarrow \boldsymbol{s}_t = \boldsymbol{g}_t^2$$

effective learning rate reduced in directions where gradient is consistently large

Neural Networks

Neural Nets can be defined as **neural-inspired nonlinear models** for supervised learning

- The basic unit of a NN is the **neuron,** a transformation of a set of *d* input features into a scalar output

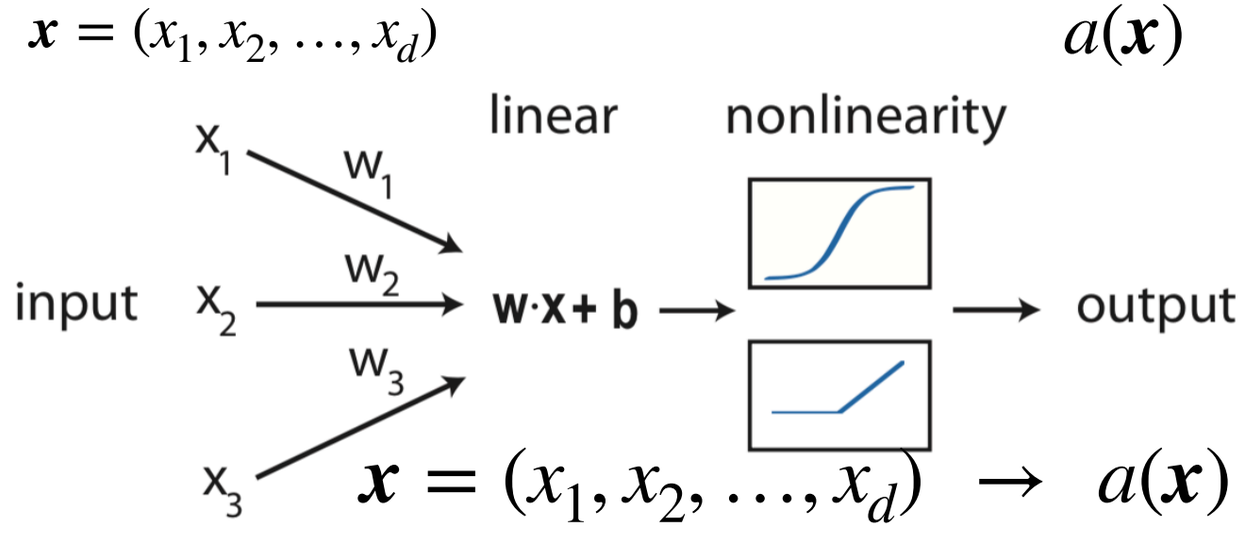
$$\boldsymbol{x} = (x_1, x_2, \ldots, x_d) \rightarrow a(\boldsymbol{x})$$

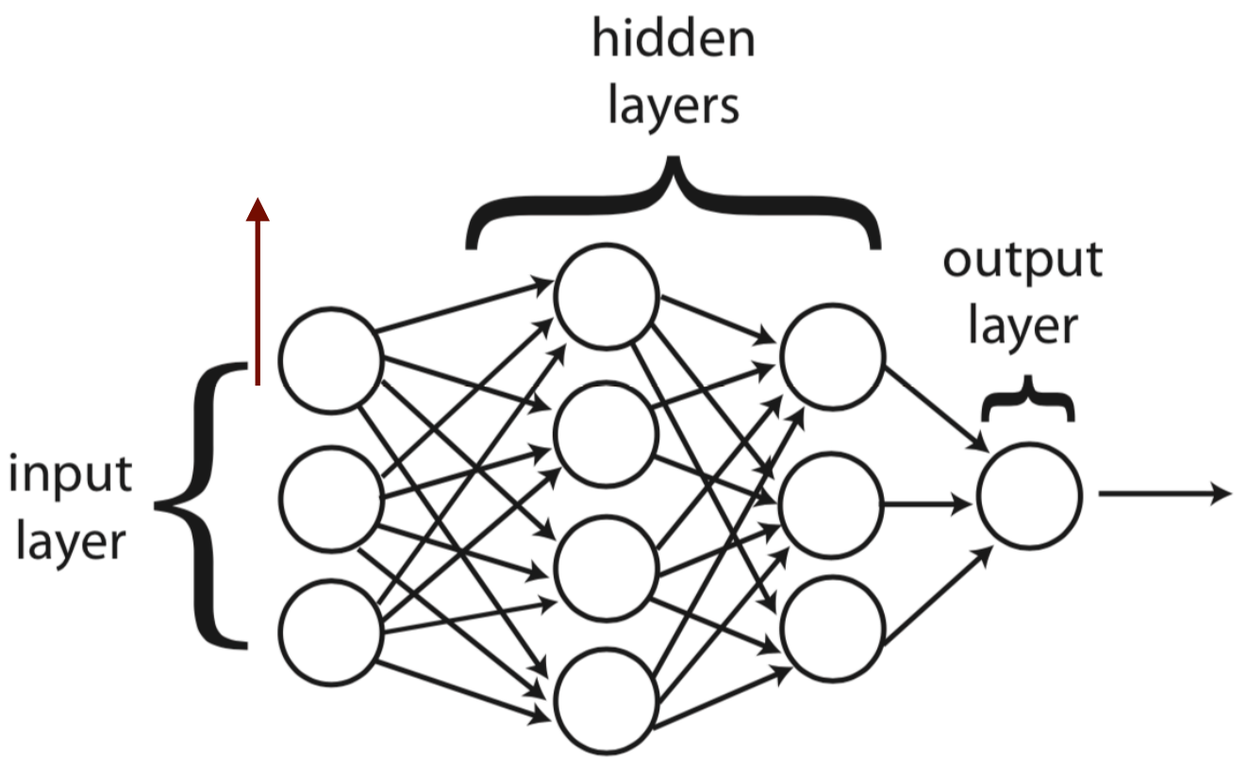
mapping

neuron inputs — *neuron output (activation state)*

- These neutrons are arranged in **layers**, which in turn are stacked on each other. The intermediate layers are called **hidden layers**

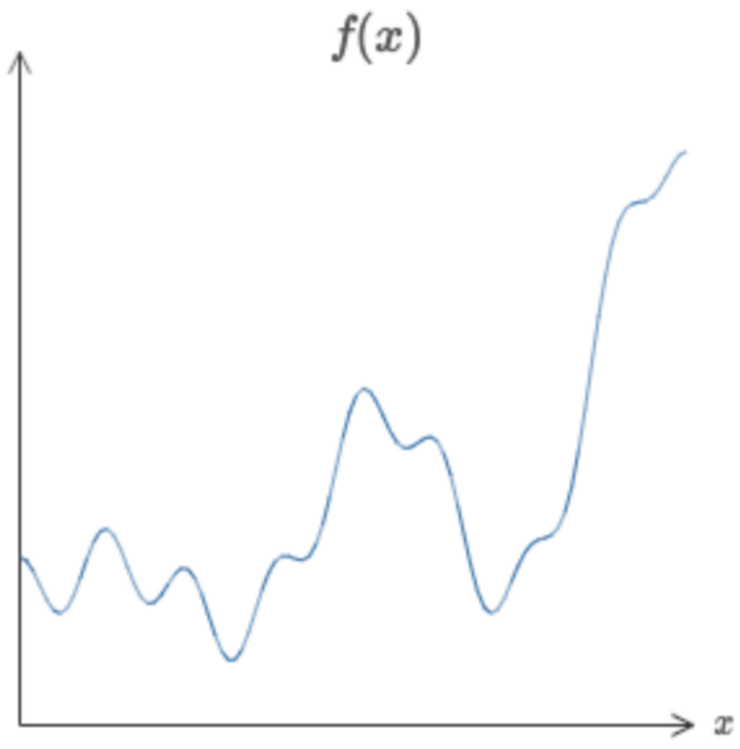
- Here we will focus on **feed-forward NNs,** where the output of the neurons of the previous layer becomes the input of the neurons in the subsequent layer

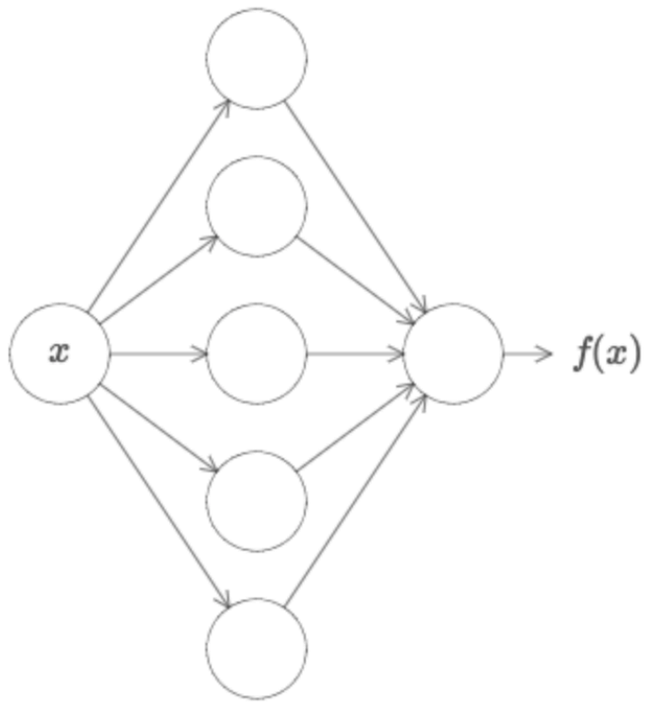




The Universal Approximation Theorem

theorem: a neural network with a single hidden layer and enough neurones can **approximate any continuous, multi-input/multi-output function** with arbitrary accuracy





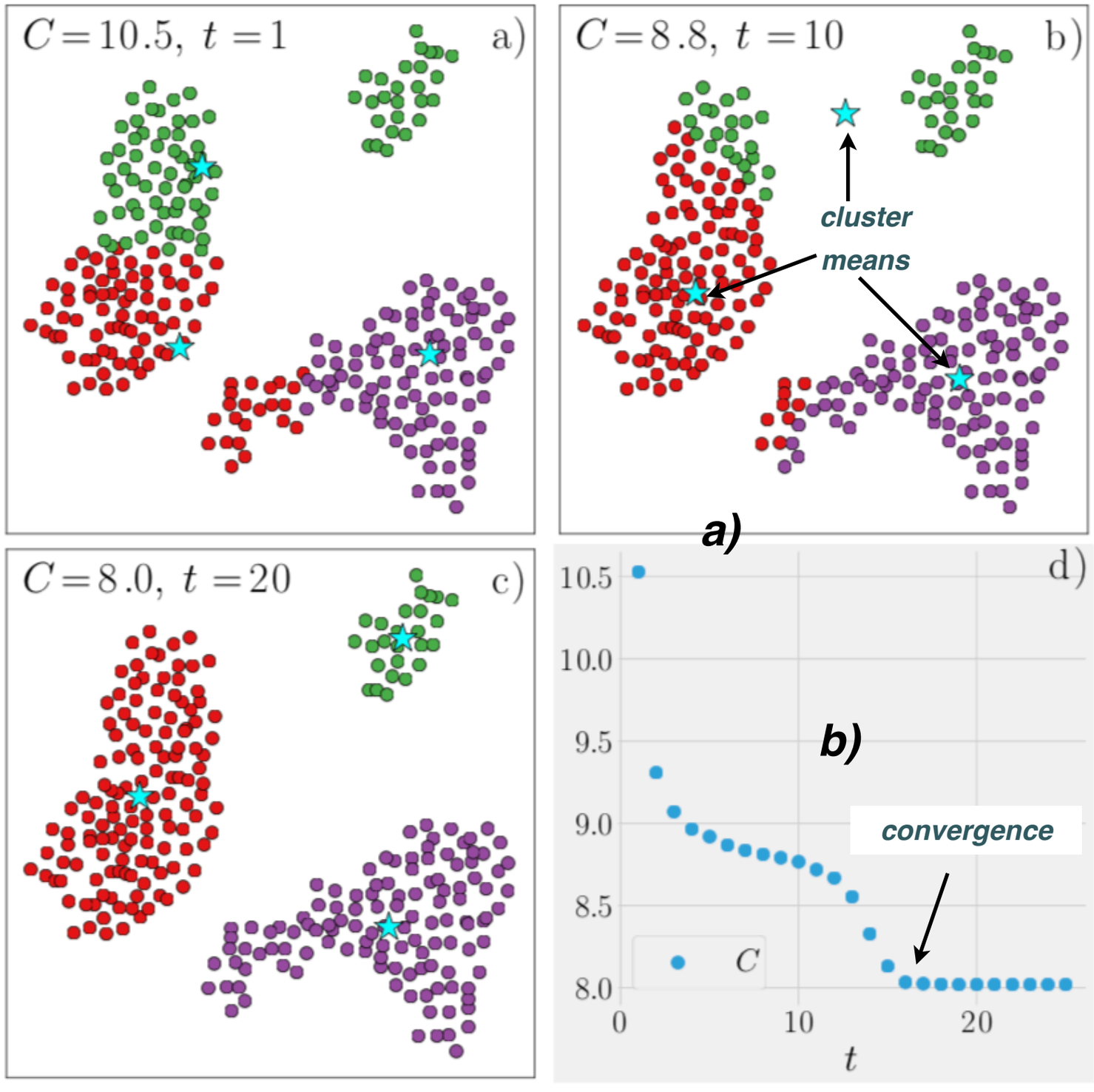
neural networks exhibit *universality properties:* no matter what function we want to compute, we know (theorem!) that there is a neural network which can carry out this task

See M. Nielsen, *Neural Networks and Deep Learning*: http://neuralnetworksanddeeplearning.com/chap4.html

Unsupervised learning: Clustering

these two steps are iterated until some **convergence criterion** is achieved, *e.g.* when the change in the cost function between two iterations is below some threshold

each colour: different cluster



here overfitting not possible: there exists a unique assignment that minimises the cost function

Dimensional reduction

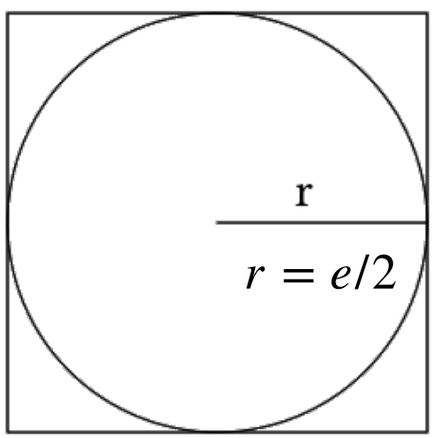
- *High-dimensional data lives near the edge of the sample space*

consider data distributed at random in a *D*-dimensional hypercube $C = [-e/2, e/2]^D$

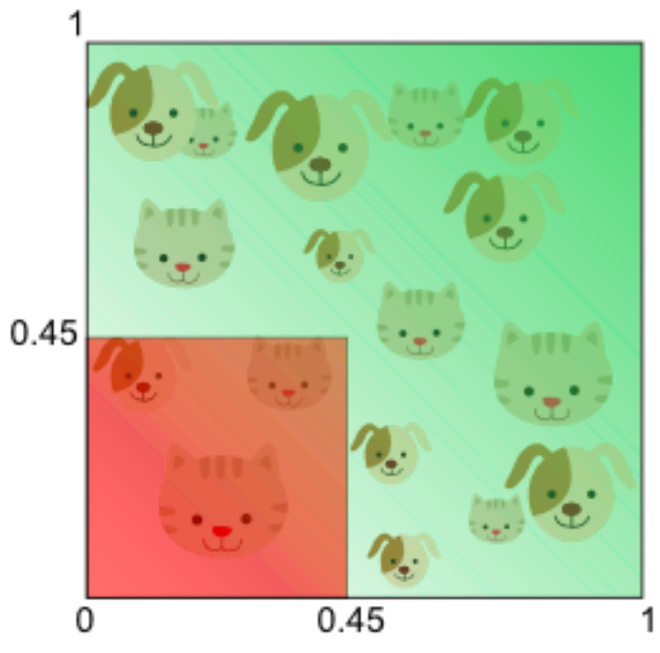
consider a *D*-dimensional sphere *S* of radius *e/2* entered at origin

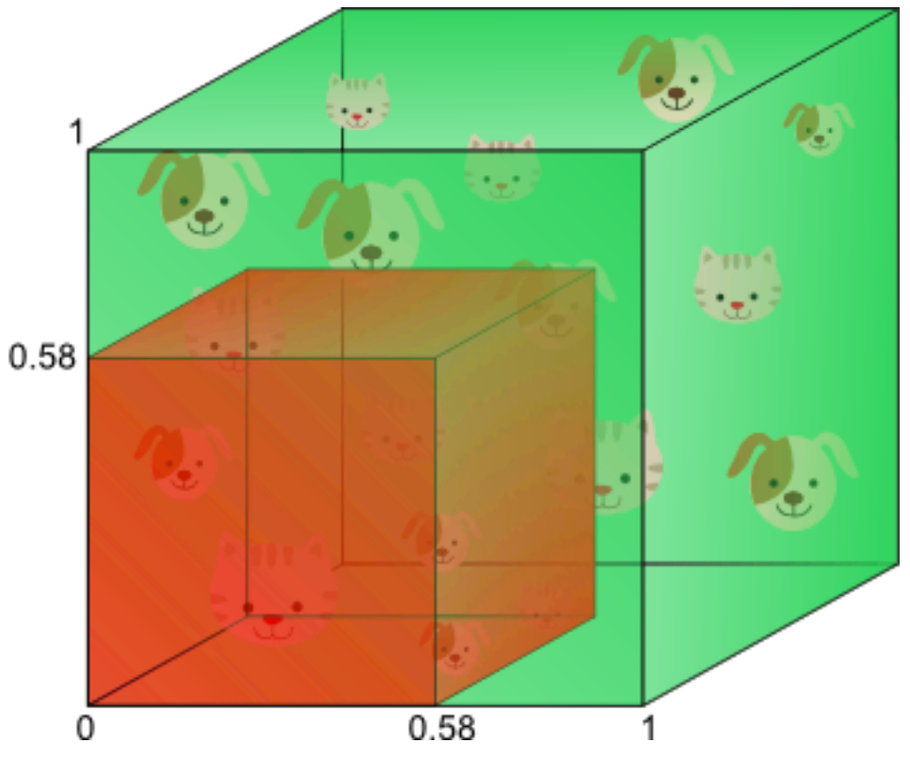
probability that random point from *C* is sampled inside the sphere *S* is

$$P\left(\boldsymbol{x}_i \in \mathcal{S}\right) \simeq \frac{\pi^{D/2}(e/2)^D/\Gamma(D/2+1)}{e^D} = \simeq \frac{\pi^{D/2}}{2^D D^D} \to 0$$



so most of the data lies close to the hypercube **edge**!





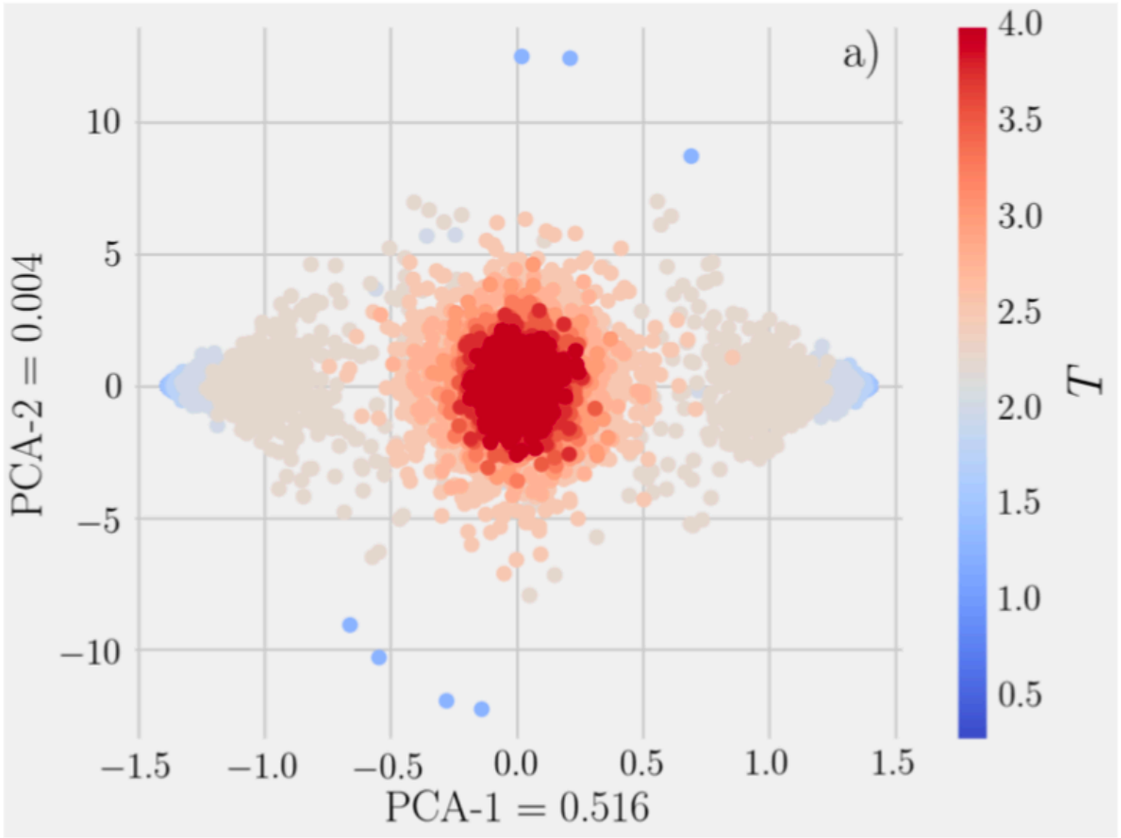
Principal Component Analysis

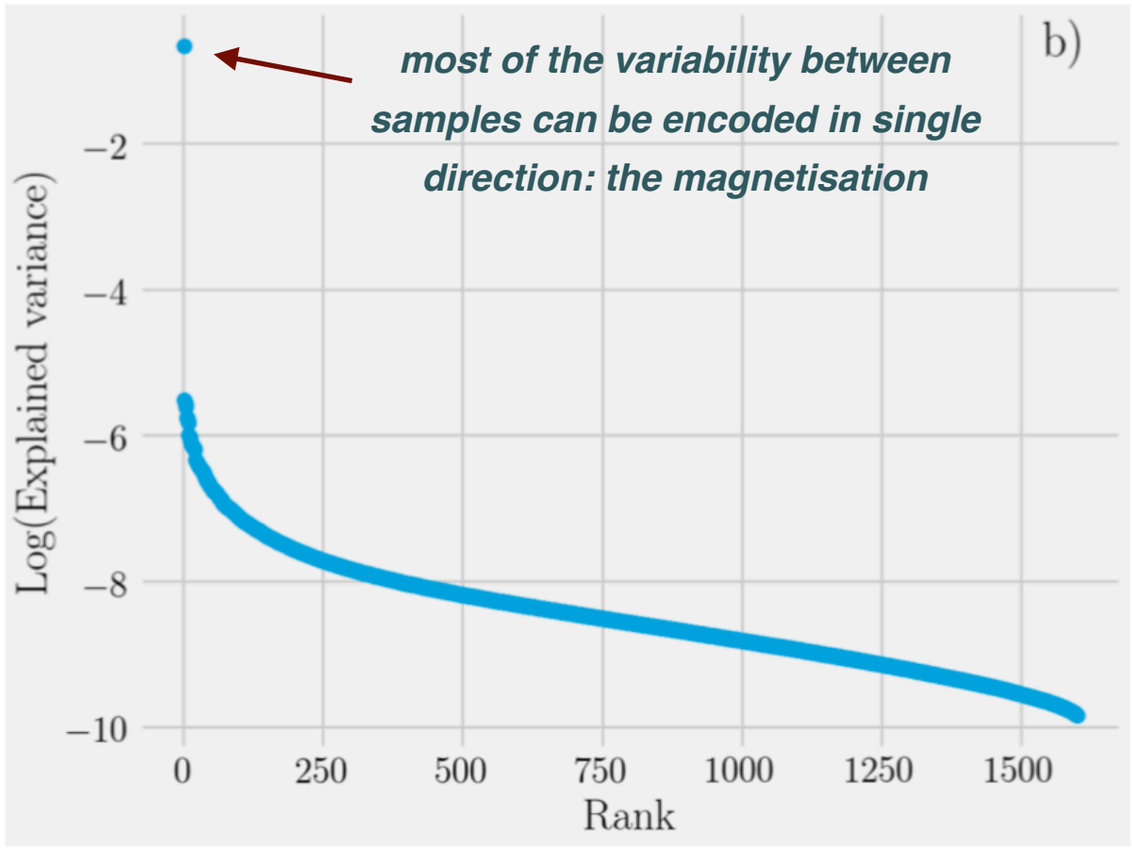
PCA projections often capture the **large-scale structure** of high-dimensional datasets

consider for example the **Ising Model in 2D** with 40 spins: 1600-dimensional space

can we **measure ``order''** with few parameters?

1000 samples for each T





The first principal component accounts for > 50% of total variability!

This first PCA component corresponds to the **magnetisation order parameter**, which we have thus identified without any prior physical knowledge of the system

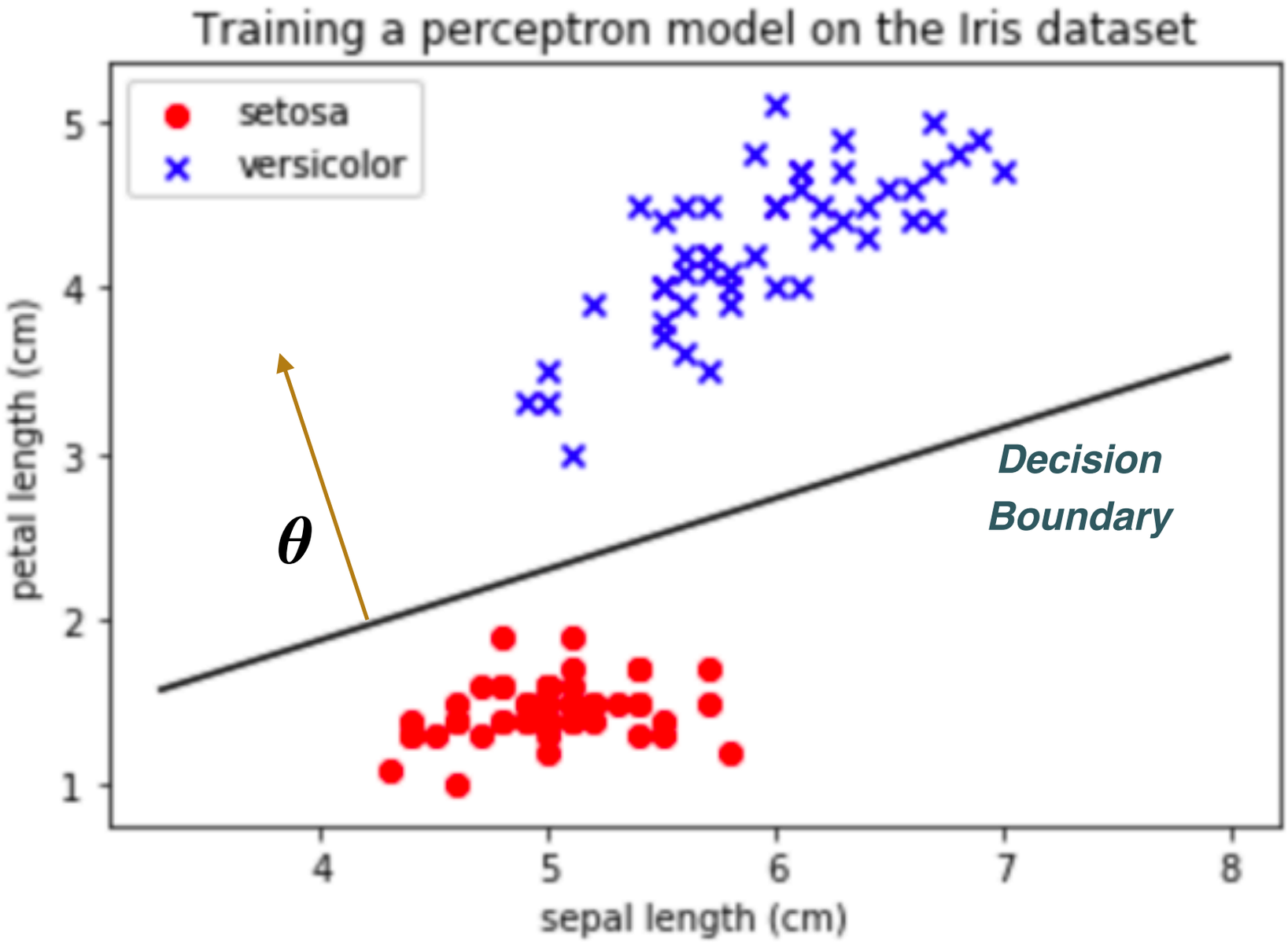
Classification

chihuahua or muffin??

what information do we need a train a ML model to classify a picture as being either a dog or a muffin?

Perceptrons

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



Looking at this example, which feature is typical for the decision boundary of a perceptron?
And for which problems do you expect the perceptron to fail?

Logistic regression

probability distribution of the model parameters θ given the observed data D

$$\mathscr{L}(\boldsymbol{\theta}\,|\,\mathscr{D}) = \prod_{i=1}^{n} P(y_i\,|\,\boldsymbol{x}_i,\boldsymbol{\theta}) = \prod_{i=1}^{n} \left(\sigma(\boldsymbol{x}_i^T\boldsymbol{\theta}\right)^{y_i} \times \left(1-\sigma(\boldsymbol{x}_i^T\boldsymbol{\theta}\right)^{1-y_i}$$

Since the cost function is the **negative log-likelihood**, we find that for logistic regression

$$E(\boldsymbol{\theta}) = -\log\mathscr{L} = \sum_{i=1}^{n} -\left(y_i \log\sigma(\boldsymbol{x}_i^T\boldsymbol{\theta}) + (1-y_i)\log(1-\sigma(\boldsymbol{x}_i^T\boldsymbol{\theta}))\right)$$

which is known as the cross-entropy function

The parameters of the model are determined by minimising the cross-entropy

$$\hat{\theta} = \arg\,\min_{\theta}\left\{ \sum_{i=1}^{n} \left(-y_i \log\sigma(\boldsymbol{x}_i^T\boldsymbol{\theta}) - (1-y_i)\log(1-\sigma(\boldsymbol{x}_i^T\boldsymbol{\theta}))\right) \right\}$$

note that no analytic solution is possible, and numerical methods are required, e.g. Gradient Descent

A: by replacing the sigmoid by a deep learning model

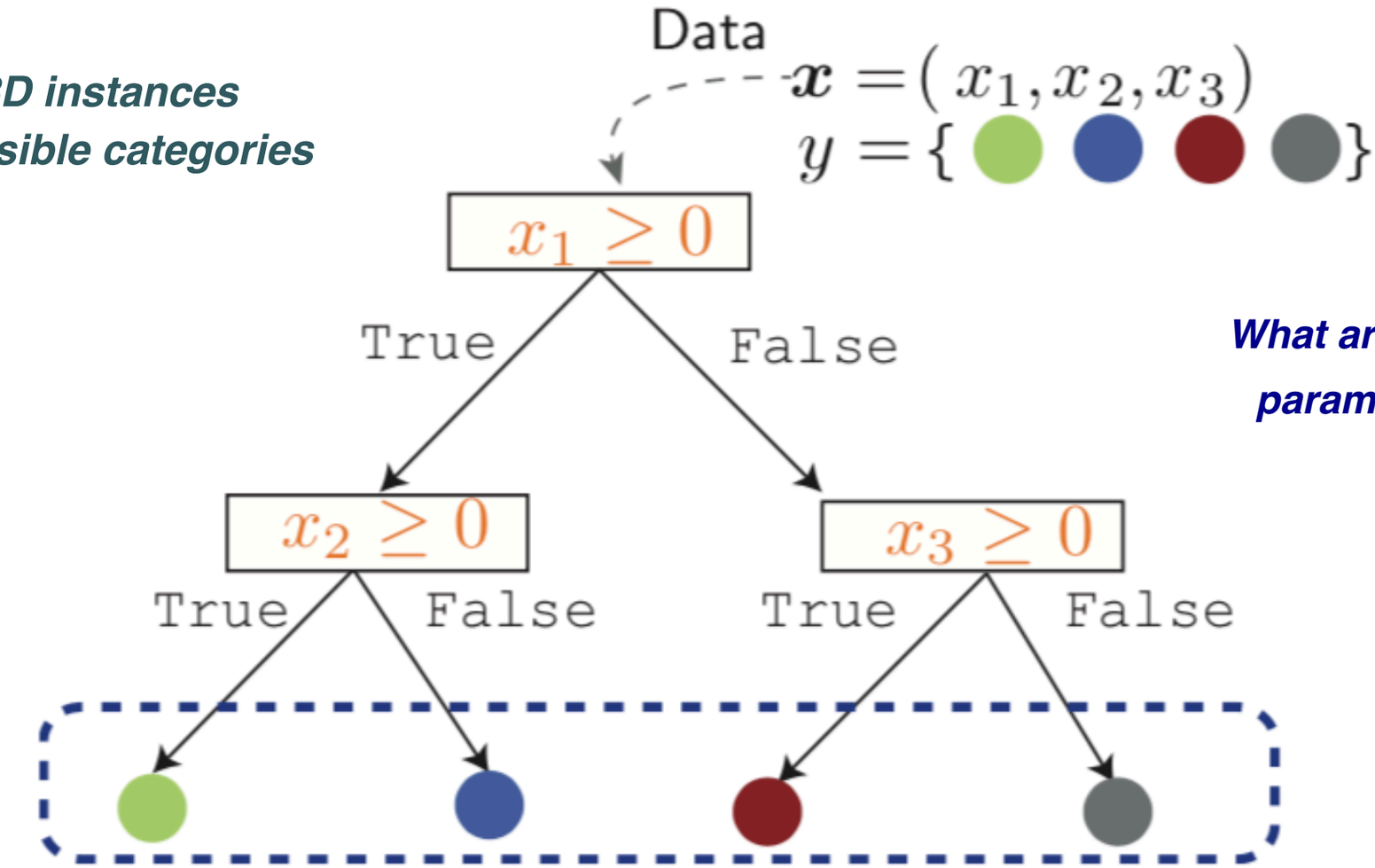
Decision Trees and Random Forests

aka another family of Machine Learning **Classification algorithms**

Decision Trees

Decision Trees are classifiers that work by partitioning the input space into **hypercubes** and then assign a model (e.g. a constant) to each region

classify 3D instances into four possible categories



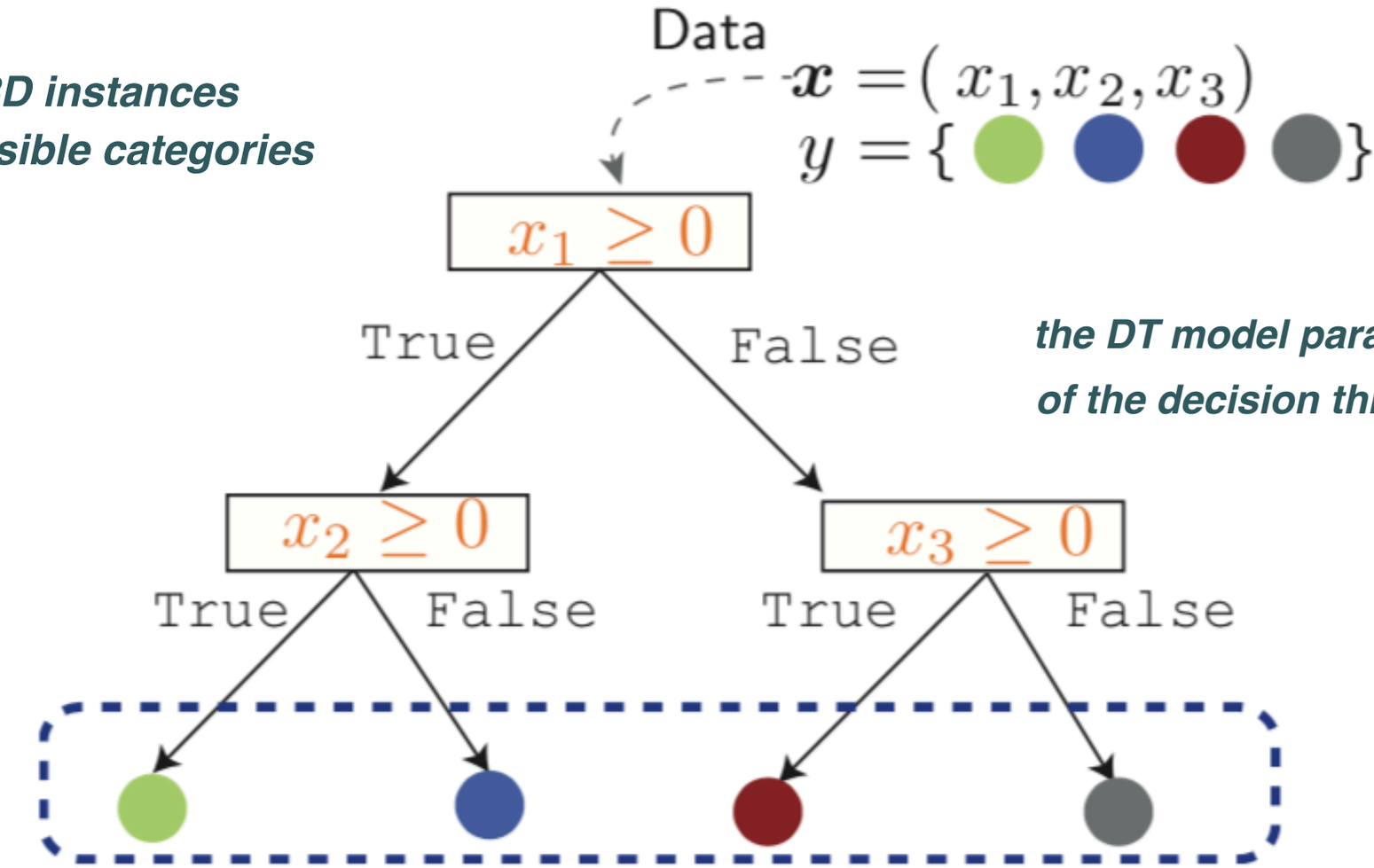
What are the model parameter here?

In ML, a **decision tree** is an algorithm which uses a series of questions to hierarchically partition the data, where each branch of the tree splits the data into smaller subsets

Decision Trees

Decision Trees are classifiers that work by partitioning the input space into **hypercubes** and then assign a model (e.g. a constant) to each region

classify 3D instances into four possible categories

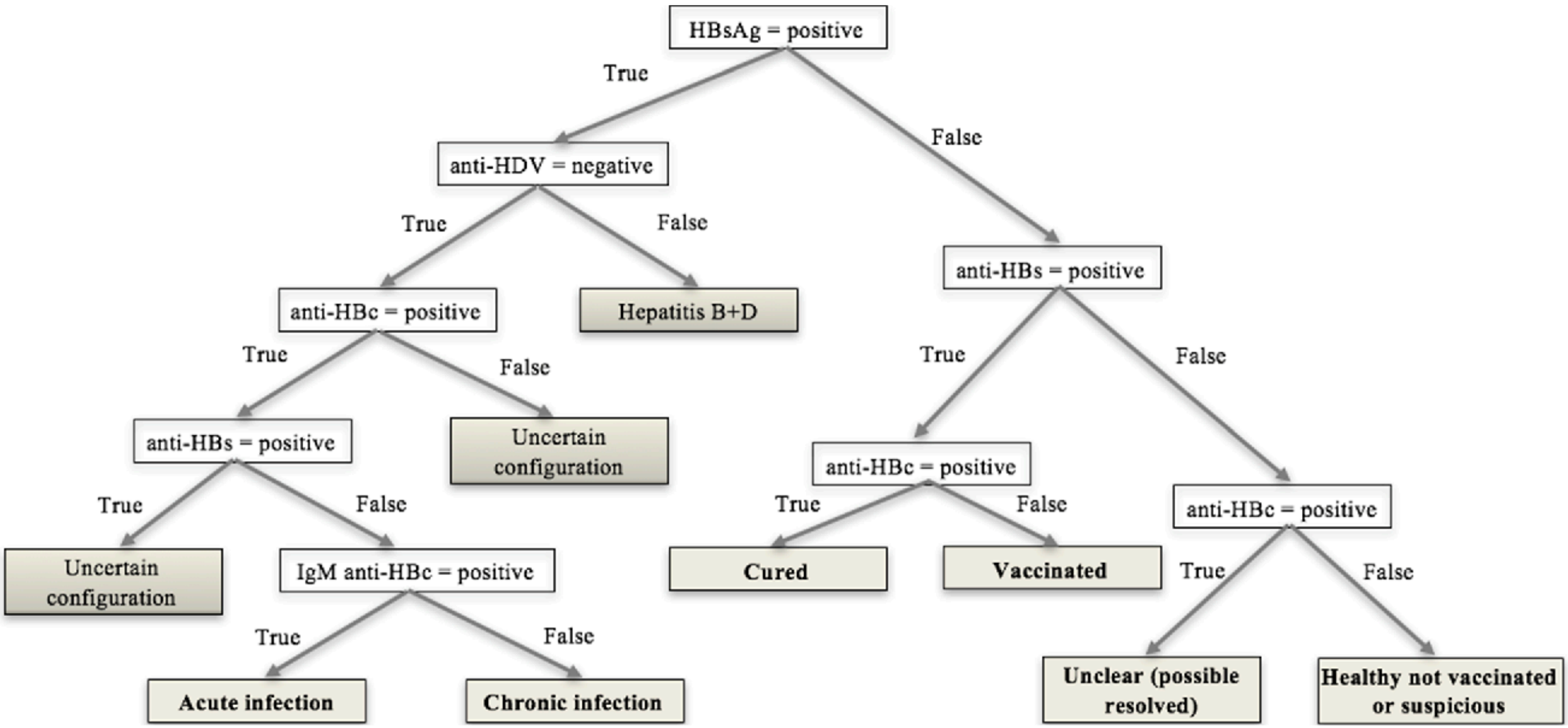


the DT model parameters are the values of the decision thresholds at each node

In ML, a **decision tree** is an algorithm which uses a series of questions to hierarchically partition the data, where each branch of the tree splits the data into smaller subsets

Decision Trees

Decision trees have the key property that their output is **human-interpretable:** sequence of binary decisions applied to individual input variables



Random Forests

individual trees have often **high variance** and are weak classifiers:
we can improve by incorporating them in an ensemble method

→ We need an ensemble of **randomised decision trees** (minimised correlations)

- ***(1)*** train each decision tree on a different bootstrapped dataset: **bagged decision tree**
- ***(2)*** use different random subset of features at each split: **random forest**

reduces correlations between trees that arise when only few features are strongly predictive

as other ML learning classifiers also Random Forests require some **regularisation**, for example a maximum depth of the tree, to control complexity and prevent overfitting

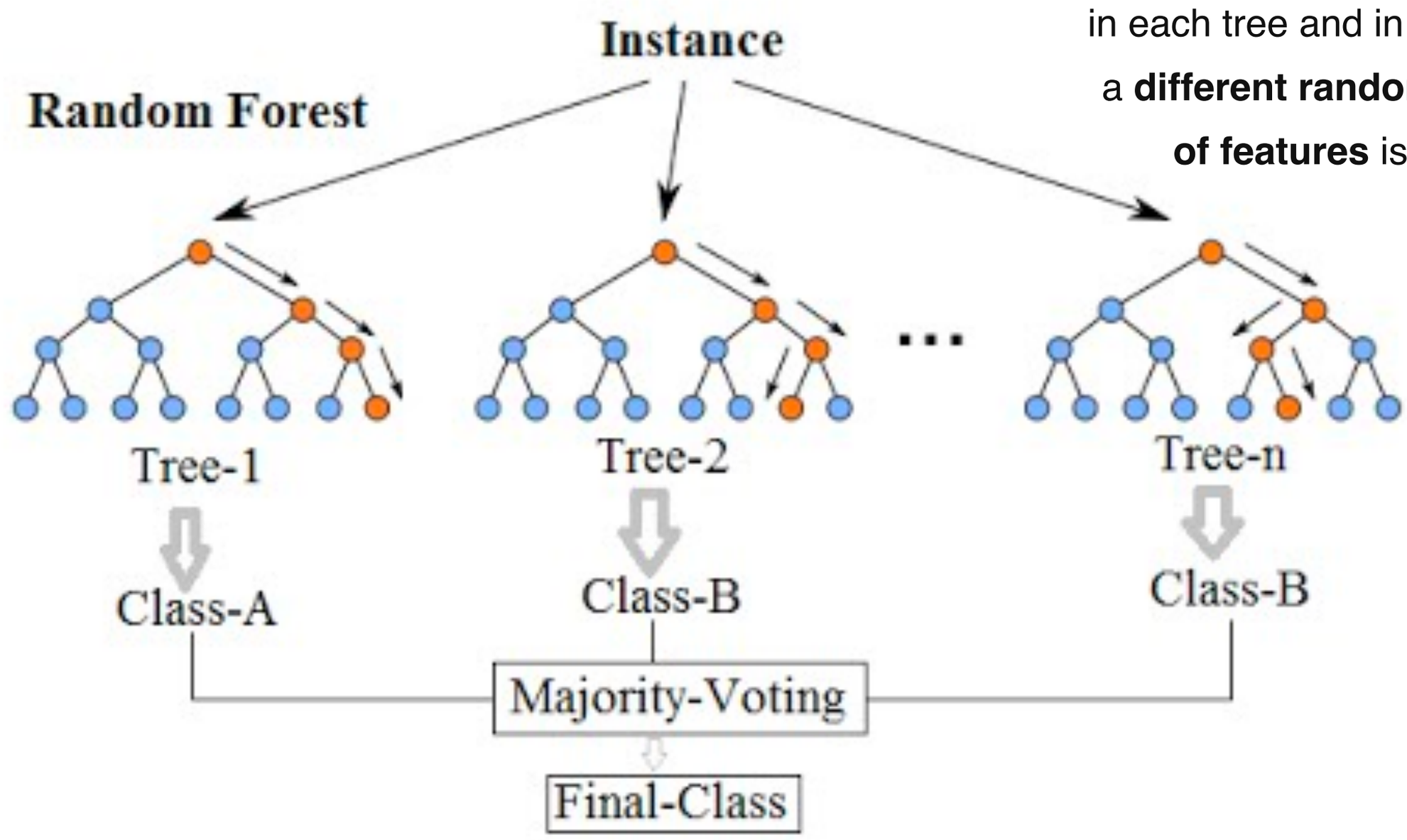
typically, a classification problem with p features, in RFs only $p^{1/2}$ features are used in each split

Random forests have other attractive features, for example, they can be used to **rank the importance of variables** in a regression or classification problem in a natural way

Random Forests

each decision tree in the ensemble is built upon a **random bootstrap sample** of the original data

in each tree and in each split, a **different random subset of features** is used



the final categorisation is assigned *e.g.* from **majority voting**

Generative Models

Generative Models

Q: what is the key property of **generative models** which makes them distinct from **discriminative models?**

Generative Models

Q: what is the key property of **generative models** which makes them distinct from **discriminative models?**

discriminative models: tell part muffins from chihuahuas

why they are conceptually different?

generative models: determine the probability distribution associated to "muffin" …

Generative Models

Q: what is the key property of **generative models** which makes them distinct from **discriminative models?**

discriminative models

$$p\left(\mathscr{C}_k \mid \boldsymbol{x}\right)$$

probability that instance x belongs to class C_k

generative models

$$p\left(\boldsymbol{x} \mid \mathscr{C}_k\right)$$

probability distribution of x associated to the class C_k

What we can do with generative models that is not possible with discriminative models?

Generative Models

let us go back to our discussion of **Decision Theory**. There we saw that a general classification problem can be separated into two distinct steps:

posterior class probabilities

- The **inference stage,** where a set of input examples is used to train a model for $p(\mathscr{C}_k|\boldsymbol{x})$
- The **decision stage,** where the information on these posterior probabilities is used to make **optimal class assignments**

So far we focused on **discriminative models,** where some criterion (*e.g.* minimise misclassification) is used together with the posterior probabilities to assign each new instance to a class

Here discuss **generative models** which aim to model the **distribution in the space of inputs *x*.** The name stems because using this distribution one can **generate synthetic data points** in the input space

one benefit of this approach is that we access the marginal density in the space of input data, *p(**x**),* which is specially useful to detect new data points that have low probability in the model: **outlier, anomaly, or novelty detection**

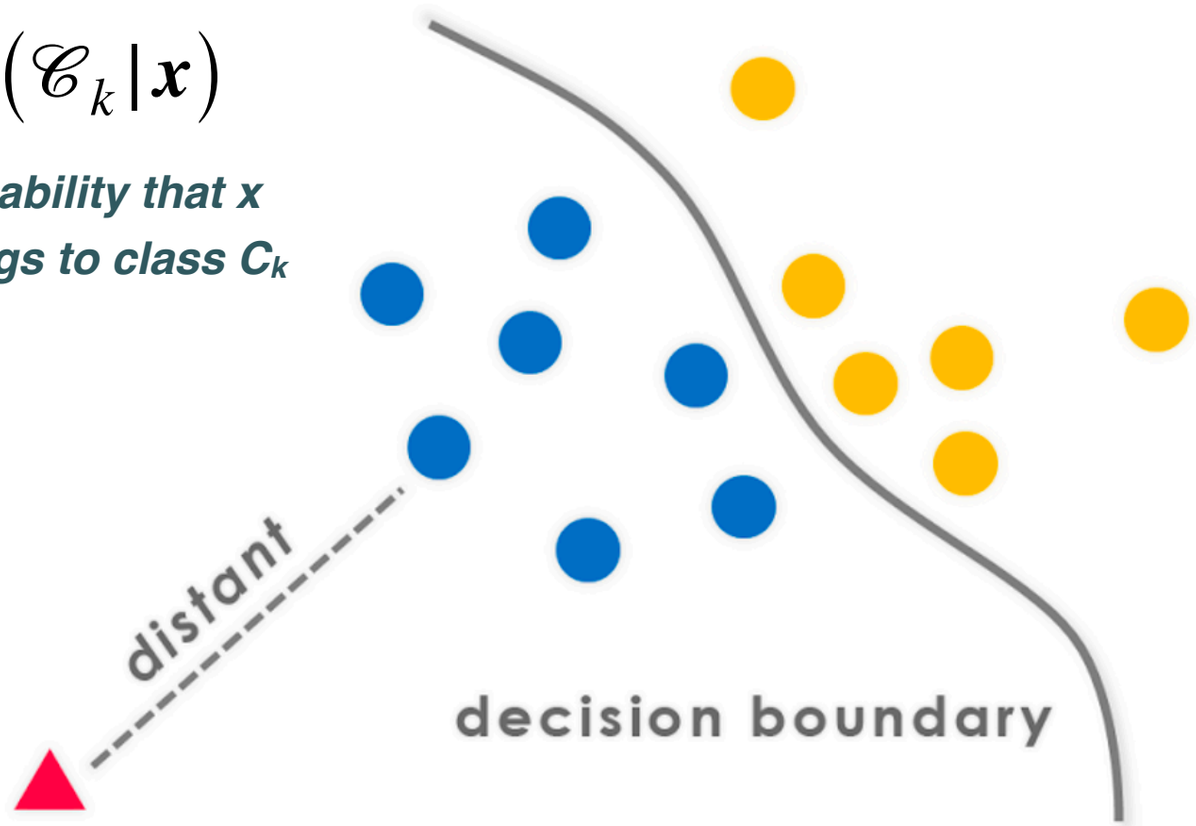
Generative Models

Compare graphically Discriminative and Generative Models for classification

Discriminative

$$p\left(\mathscr{C}_k|\boldsymbol{x}\right)$$

probability that x belongs to class C_k

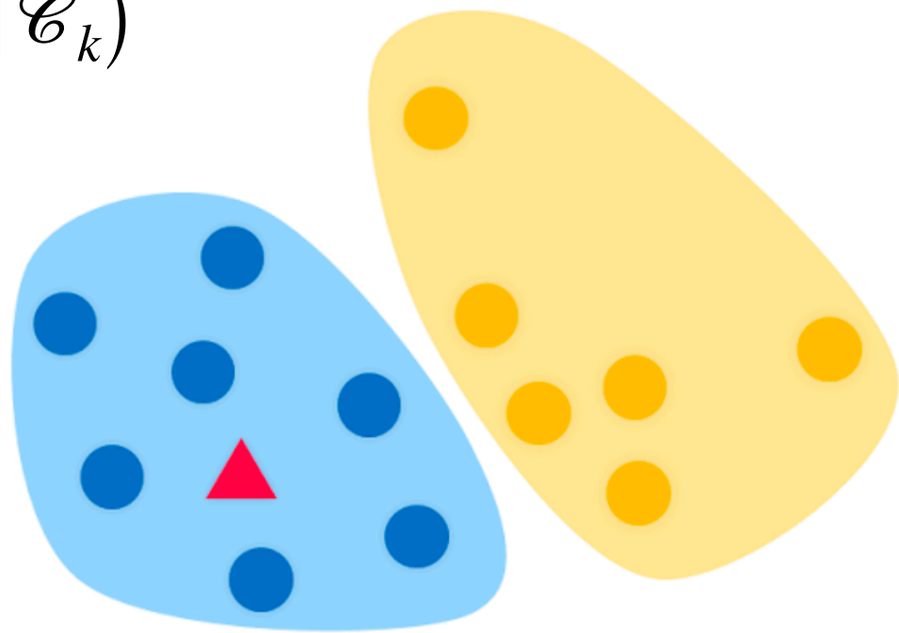


discriminative models works best when new instances are far from the decision boundary

Generative

probability of x in class C_k

$$p\left(x|\mathscr{C}_k\right)$$



in generative models new instances are generated in the bulk of the input space probability distribution

Generative Models

we can summarise the main ideas **underlying generative models** as follows

Most ML models discussed here (Supervised NNs, logistic regression, ensemble models) are **discriminative:** designed to identify **differences between groups of data**

e.g. cats vs dogs discrimination

these models cannot carry some tasks such as **drawing new examples** from an unknown probability distribution: for this we need **generative models**

e.g. learn how to draw new examples of cat and dog images

e.g. generate new samples of a given phase of the Ising model

generative models are Machine Learning techniques that allows to learn **how to generate new examples** similar to those found in a training dataset

Maximising similarity

In **generative models** one deals with two probability distributions (data and model), which we would like to have as similar as possible

$$p_\theta(\boldsymbol{x}) \qquad p_{\text{data}}(\boldsymbol{x})$$

from model — *from training data*

we aim to construct p_θ to model p_{data}

however subtleties about how we define **similarity** have large implications for the model training

maximising the **log-likelihood of the data under the model** is the same as **minimising the KL divergence** between the data distribution and the model distribution

Kullback-Leibler divergence

$$D_{KL}\left(p_{\text{data}}||p_\theta\right) = \int d\boldsymbol{x}\, p_{\text{data}}(\boldsymbol{x}) \log \frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x})}$$

$$= \int d\boldsymbol{x}\, p_{\text{data}}(\boldsymbol{x}) \log p_{\text{data}}(\boldsymbol{x}) - \int d\boldsymbol{x}\, p_{\text{data}}(\boldsymbol{x}) \log p_\theta(\boldsymbol{x})$$

$$= S_p[p_{\text{data}}] - \left\langle \log p_\theta \right\rangle_{\text{data}}$$

Entropy

Expected value of model probability given the data

Adversarial Learning

A subtle point concerns which of the two versions of the KL-divergence to minimise:

(1) $D_{KL}\left(p_{\text{data}}||p_\theta\right)$ → ***(draw values of θ accordingly to p_θ and compare with data)*** → ***Calculable using sampling***

(2) $D_{KL}\left(p_\theta||p_{\text{data}}\right)$ → ***Large when model over-weights low-density regions near real peaks***

→ ***but not calculable since p_{data} unknown ….***

In **Adversarial Learning** we achieve a similar goal as that of minimising ***(2)*** by training a **discriminator** to distinguish between real data points and samples from the model

By punishing the model for generating points that can be easily discriminated from the data, Adversarial Learning decreases the **weight of regions in the model space that are far away from data points,** regions that inevitably arise when maximising the likelihood

Generative adversarial networks

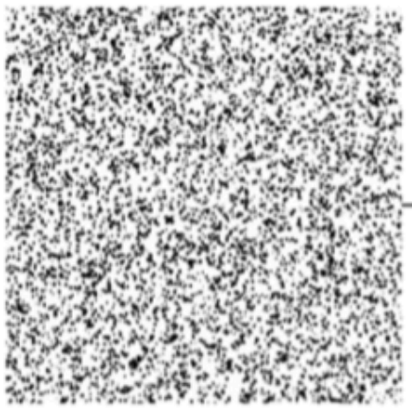
Generative Adversarial Networks (GANs) are deep neural network architectures, composed by two independent NNs which **compete against each other**

- ***(1)*** A **generator G** NN that creates (samples) pseudo-data by inferring the probability distribution associated to the training dataset

- ***(2)*** A **discriminator D** NN which determines the probability of a given sample arises from the actual training data rather than having been produced by **G**

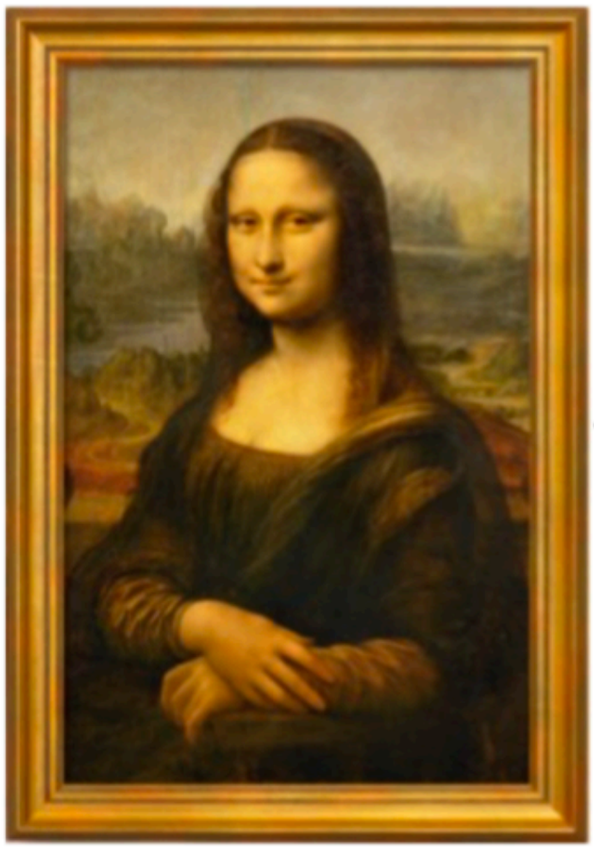
the generator network **G** should be trained to maximise the probability that the discriminator network **D** makes a mistake: that is, **G** should generate pseudo-data samples that are virtually **indistinguishable** from the actual data

Generative adversarial networks



RANDOM NOISE

GENERATOR
THIEF



REAL DATA

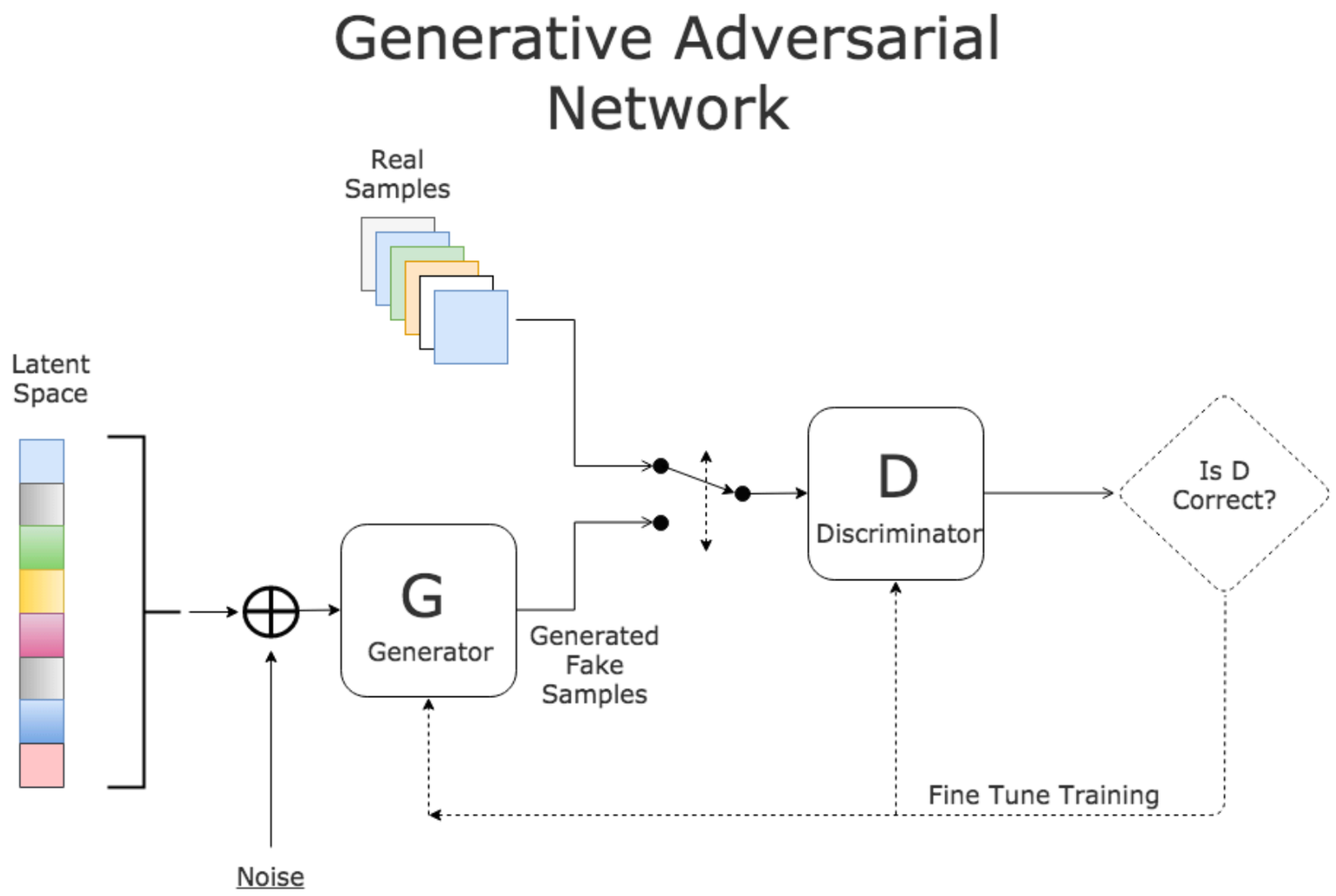
DISCRIMINATOR
DETECTIVE

REAL

FAKE

Generative Adversarial Networks

- Architecture for an **unsupervised neural network training** (unlabelled samples)
- Based on two **independent nets** that work separately and act as **adversaries:**
  - the **Discriminator (D)** undergoes training and plays the role of classifier
  - the **Generator (G)** and is tasked to generate random samples that **resemble real samples** with a twist rendering them as fake samples.



GAN training

As with other NN architectures one uses GD to train GANs, but now one has to **update sequentially** the model parameters of both **G** and **D**

- Take a sample of *N* data points from the training set

$$\{\boldsymbol{x}_n\}_{n=1}^{N} \qquad \boldsymbol{x}_n = (x_{n,1}, x_{n,2}, \ldots, x_{n,p})$$

p = number of features per sample

- Produce a sample of *N* pseudo-data points from generator **G** (at ite$_0$ this is random noise)

$$\{\boldsymbol{z}_n\}_{n=1}^{N} \qquad \boldsymbol{z}_n = (z_{n,1}, z_{n,2}, \ldots, z_{n,p})$$

- Evaluate the cost function: since we are dealing with binary classification (true/false) the appropriate cost function is the **cross-entropy**

$$C\left(\boldsymbol{\theta}_D, \boldsymbol{\theta}_G\right) = \frac{1}{N}\sum_{n=1}^{N}\left(\log D(\boldsymbol{x}_i) + \log(1 - D(G(\boldsymbol{z}_i)))\right)$$

NN params of D ***NN params of G*** ***output of D when input a real data sample*** ***output of D when input a ``fake'' data sample produced by G***

- Train **D** using GD to maximise its discrimination capability

GAN training

- Evaluate the cost function: since we are dealing with binary classification (true/false) the appropriate cost function is the **cross-entropy**

$$C\left(\boldsymbol{\theta}_D, \boldsymbol{\theta}_G\right) = \frac{1}{N}\sum_{n=1}^{N}\left(\log D(\boldsymbol{x}_i) + \log(1 - D(G(\boldsymbol{z}_i)))\right)$$

- Train **D** using GD to maximise its discrimination capability

$$\mathbf{v}_t = \eta_t \nabla_{\theta_D} C(\boldsymbol{\theta}_{D,t}, \boldsymbol{\theta}_{G,t}), \qquad \boldsymbol{\theta}_{D,t+1} = \boldsymbol{\theta}_{D,t} - \mathbf{v}_t$$

- At this point **D** can tell apart data from pseudo-data pretty well, so we need to train **G** to generate better (closer to the training set) pseudo-data samples
- Produce a sample of *N* pseudo-data points from the generator **G** $\{\boldsymbol{z}_n\}_{n=1}^{N}$

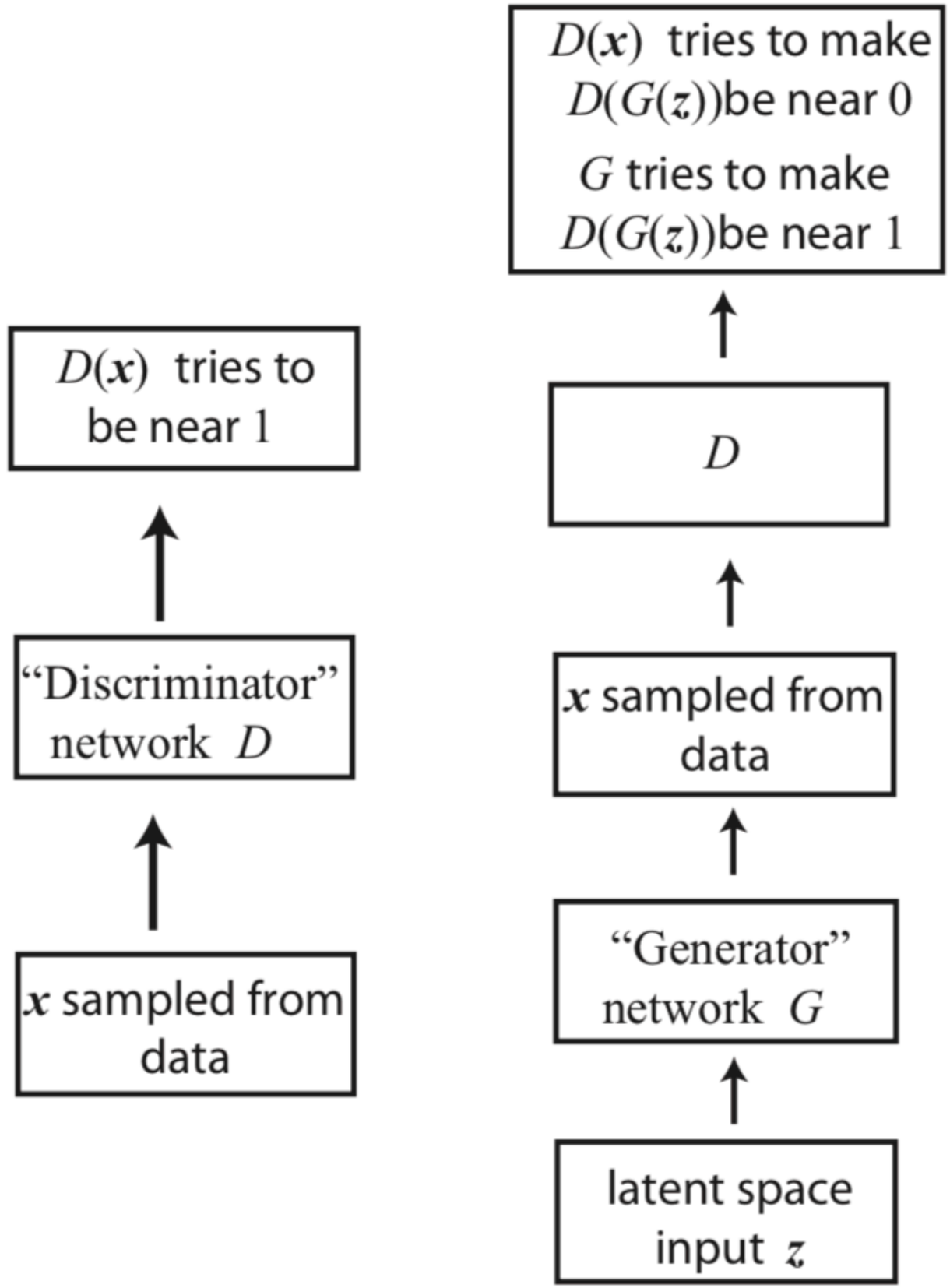
$$C\left(\boldsymbol{\theta}_D, \boldsymbol{\theta}_G\right) = \frac{1}{N}\sum_{n=1}^{N}\log(1 - D(G(\boldsymbol{z}_i)))$$

output of D (now with its parameters fixed)

$$\mathbf{v}_t = \eta_t \nabla_{\theta_G} C(\boldsymbol{\theta}_{D,t}, \boldsymbol{\theta}_G), \qquad \boldsymbol{\theta}_{G,t+1} = \boldsymbol{\theta}_{G,t} - \mathbf{v}_t$$

GAN training

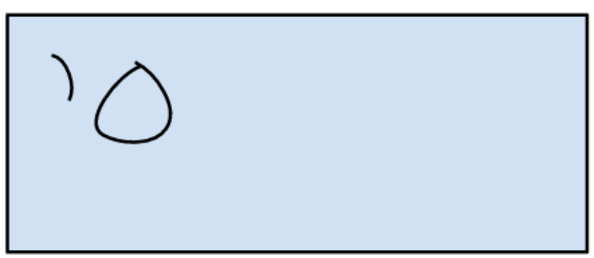
the generator and discriminator are sequentially trained and iterated until convergence is achieved, at this point **D** cannot tell apart the pseudo-data from **G** from the real data



GAN training

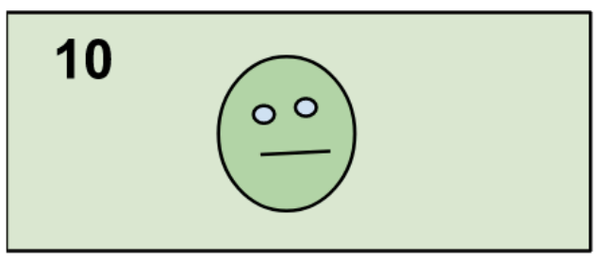
initial training

Generated Data | **Discriminator** | **Real Data**



FAKE REAL

intermediate training



FAKE REAL

long training

REAL REAL

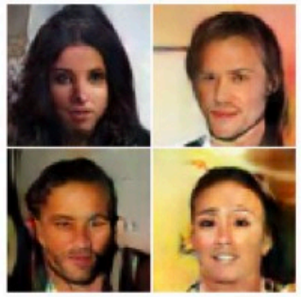
convergence!!

Image generation with GANs

https://thispersondoesnotexist.com/

Image generation with GANs

DCGAN
11/2015



EBGAN-PT
9/2016

BEGAN
3/2017
128 × 128

Progressive GAN
10/2017
1024 x 1024

horse → zebra

Image generation with GANs

https://deepdreamgenerator.com

Symmetry in Pattern Recognition

Supervised learning and classification

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

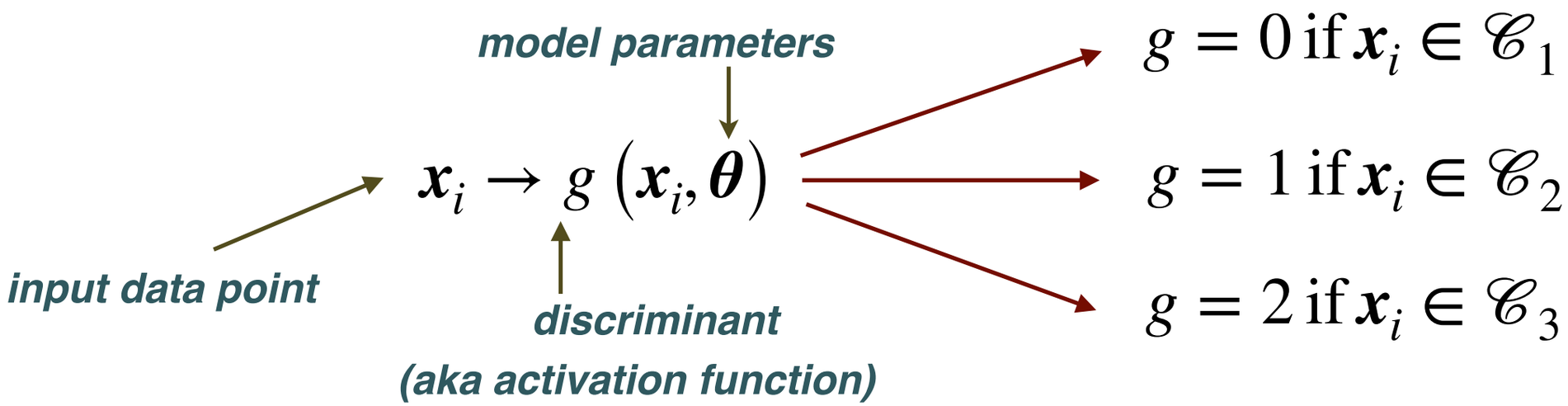
the simplest type of problem is **binary classification** (yes/no problems)

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

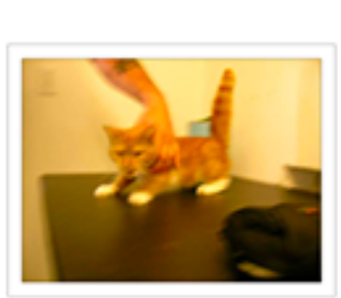
The goal is to **predict a class label** from a pre-defined list of possibilities

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

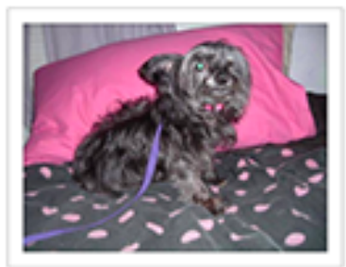
e.g. should I put this email in the spam folder?

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?

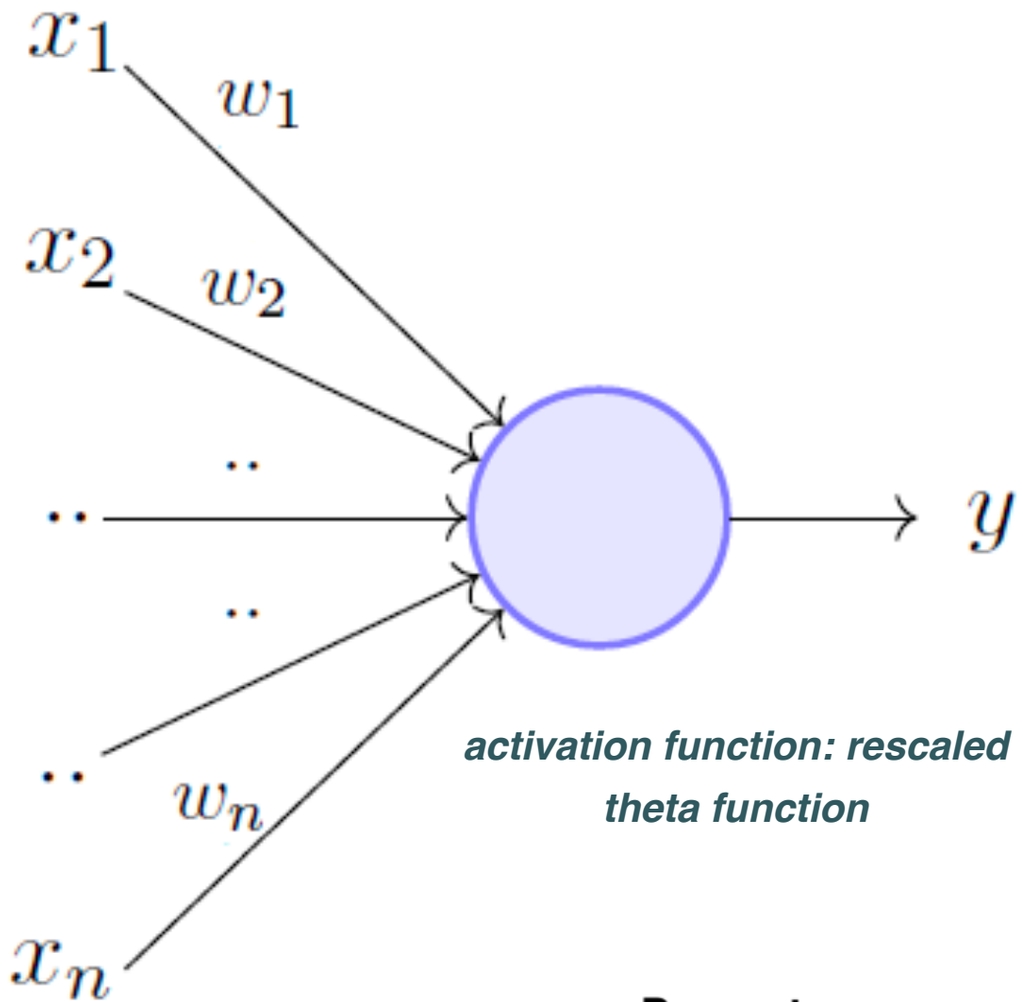
$\boldsymbol{x}_i = \left(x_{i,1}, x_{i,2}, \ldots, x_{i,p}\right)$ *p features of the samples*

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

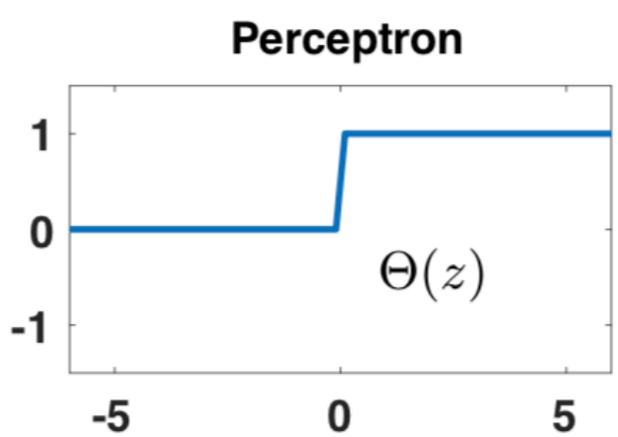
You can think of a perceptron as a **very simple, linear neural network**

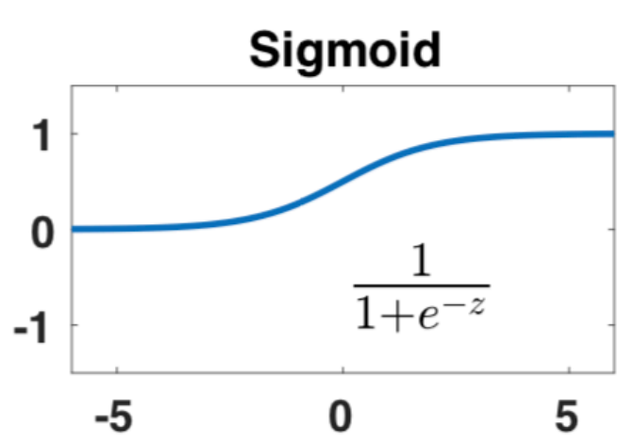


activation function: rescaled theta function

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

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





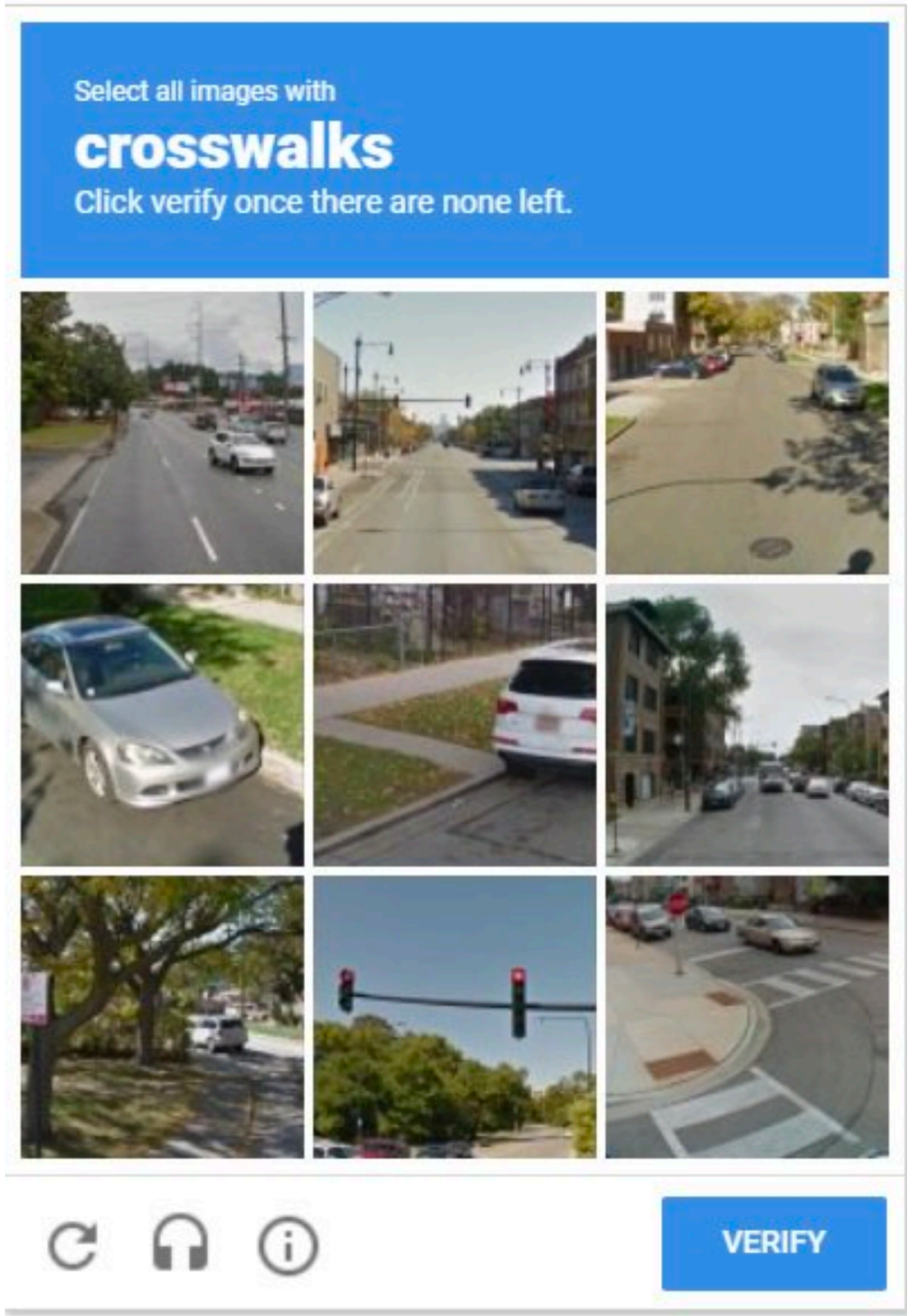
Pattern recognition

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

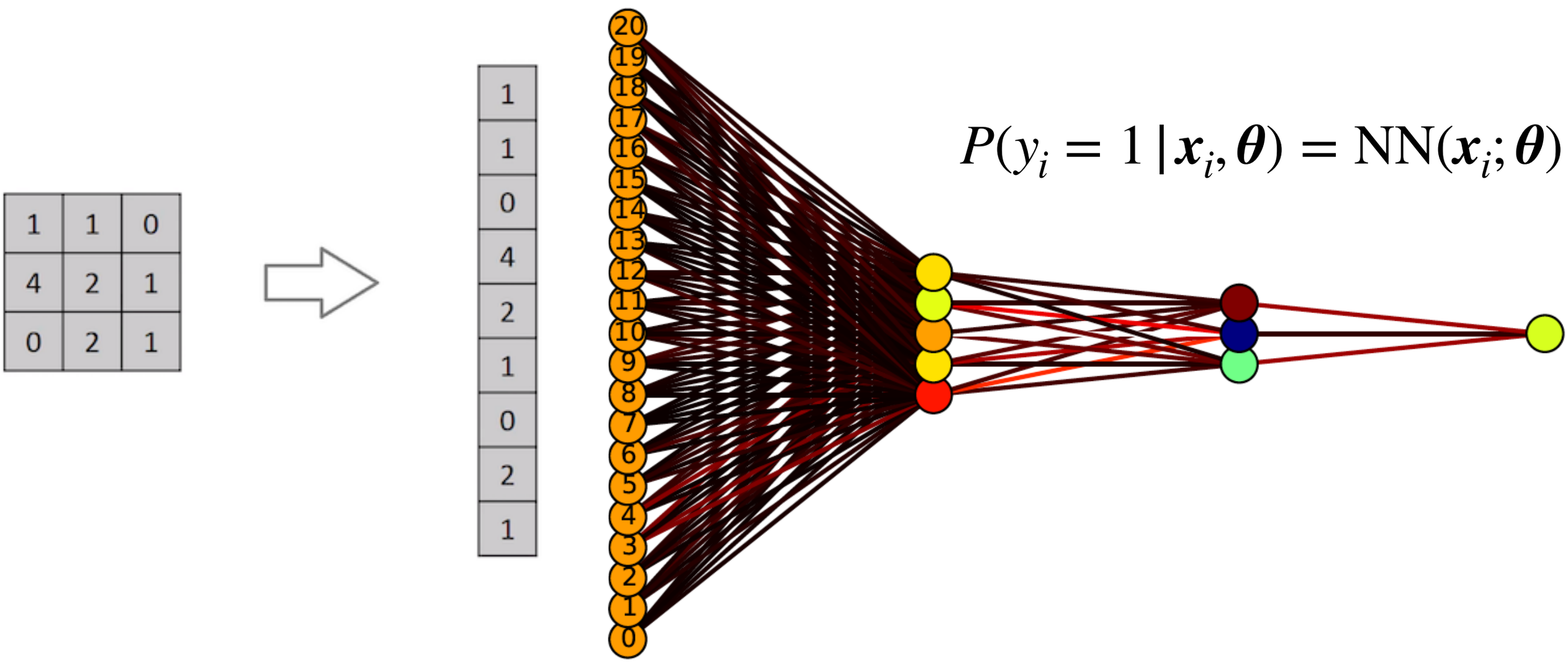
$$P(y_i = 1 \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}) = \mathrm{NN}(\boldsymbol{x}_i; \boldsymbol{\theta})$$

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!

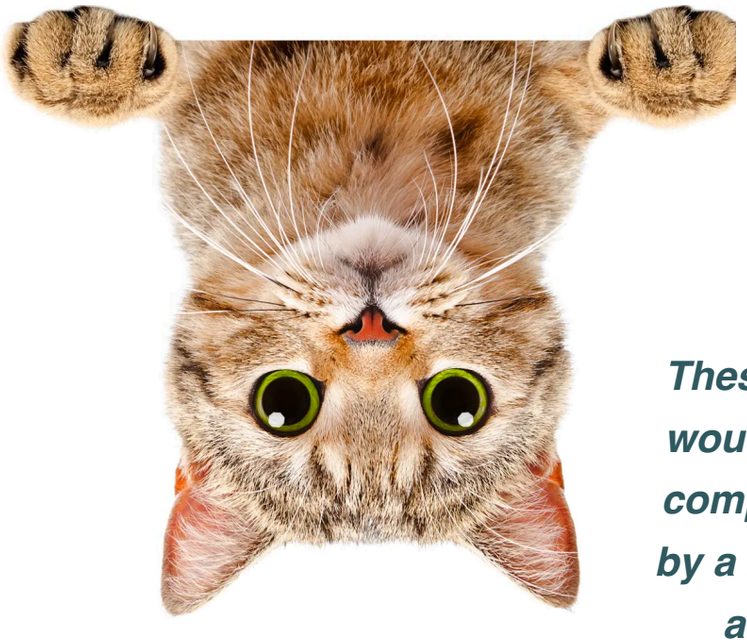
Pattern recognition

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!

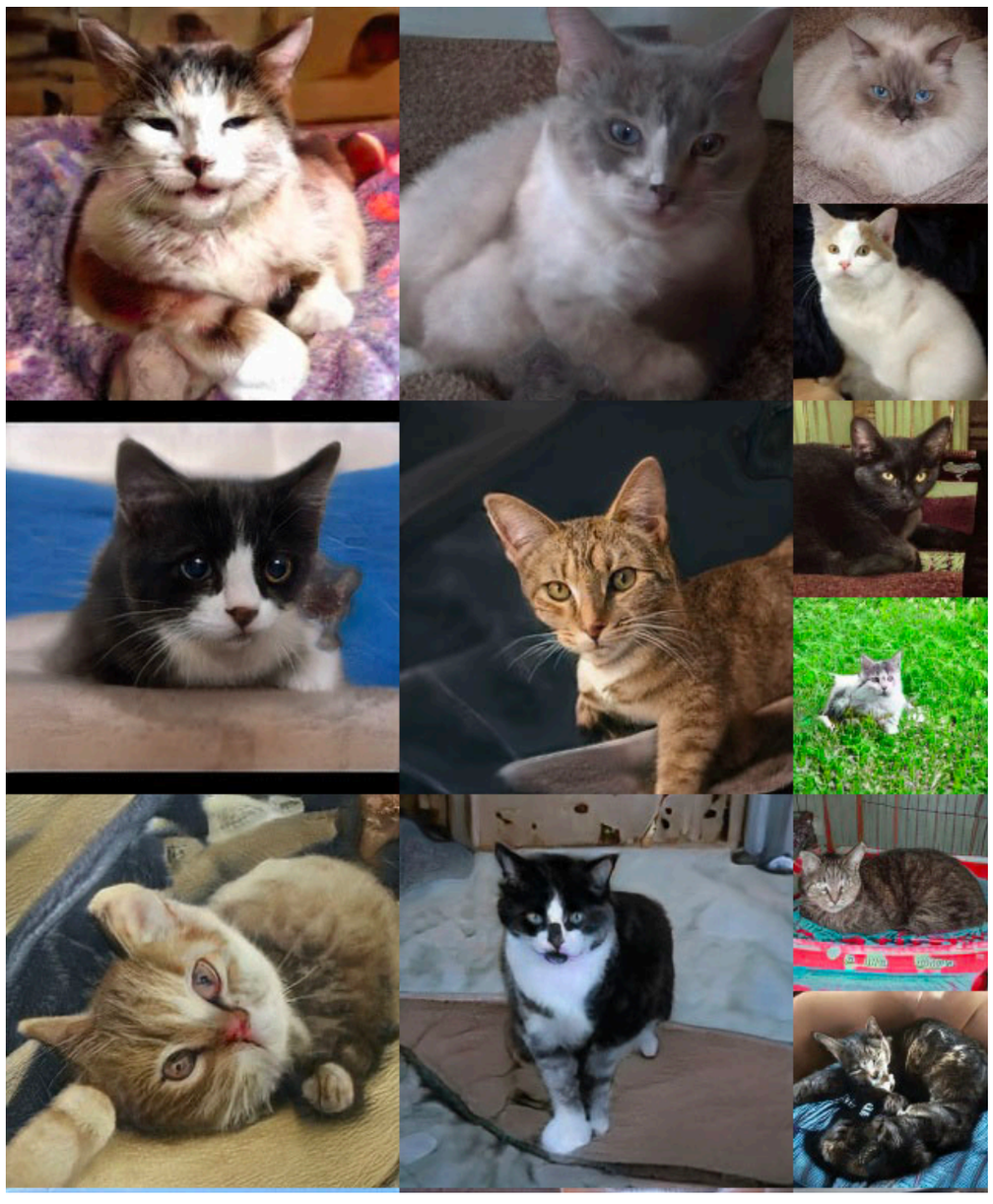


again, still a cat!

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***

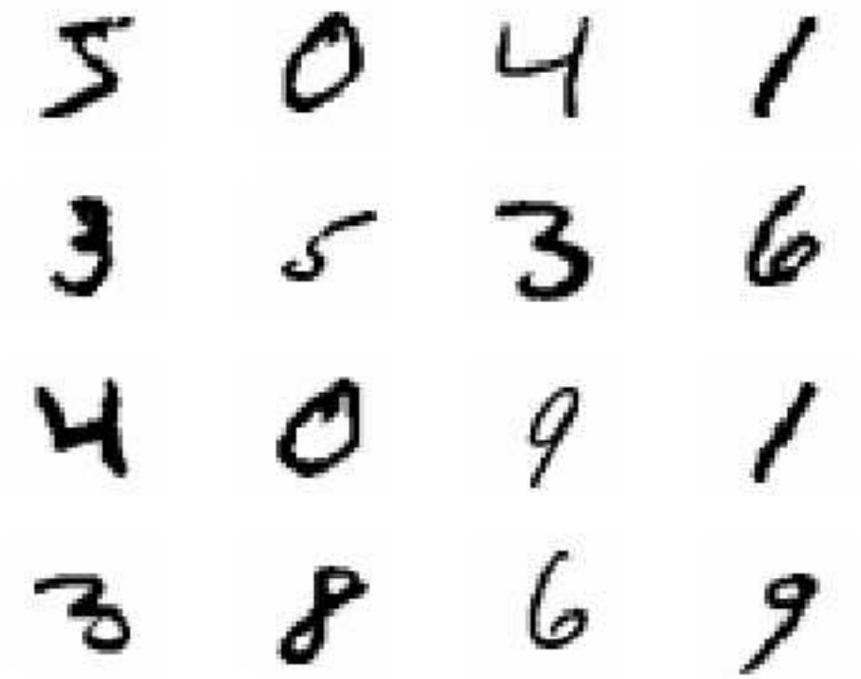
Learning with symmetry

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

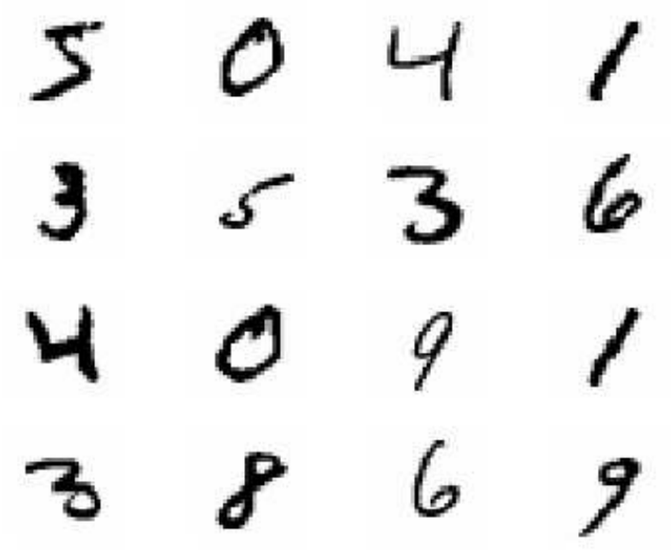
Q: what kind of **symmetries** must we built-in in our ML classifier model?

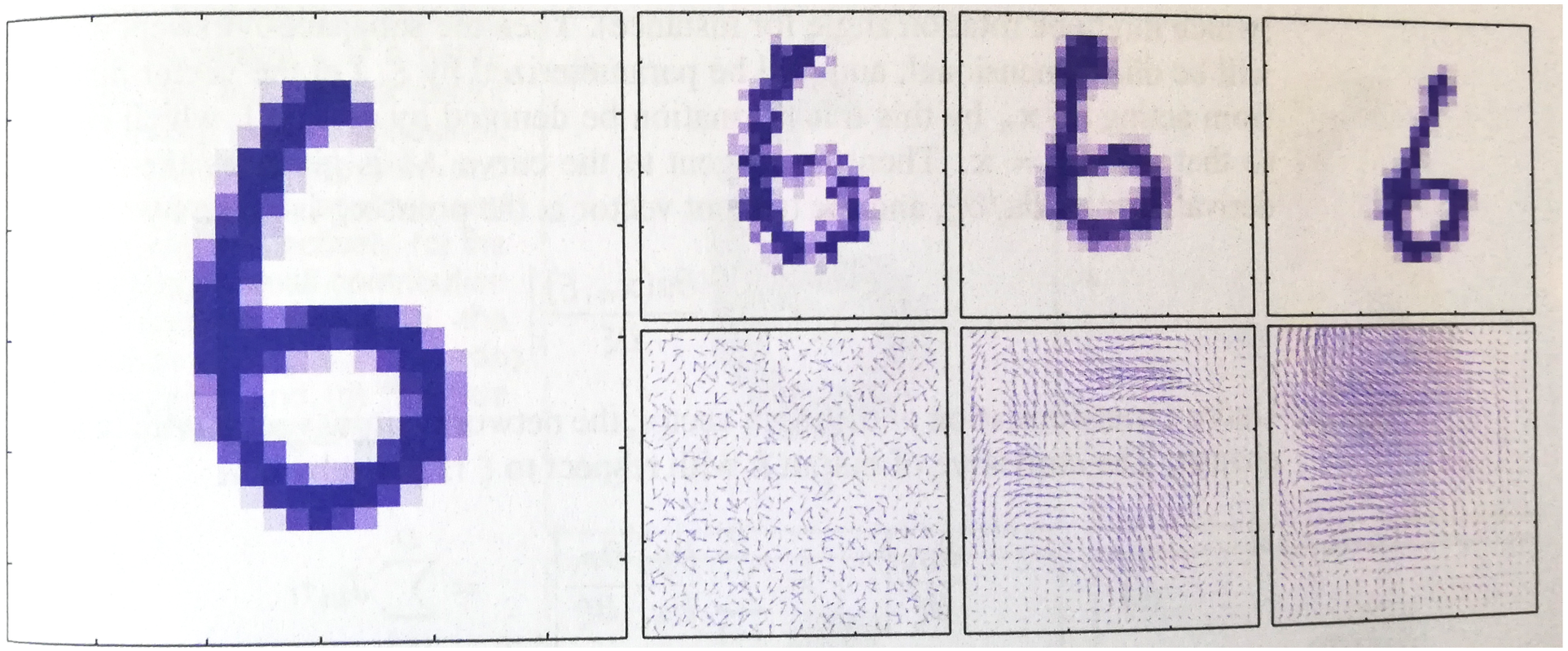
Learning with symmetry

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

some are obvious choices ...

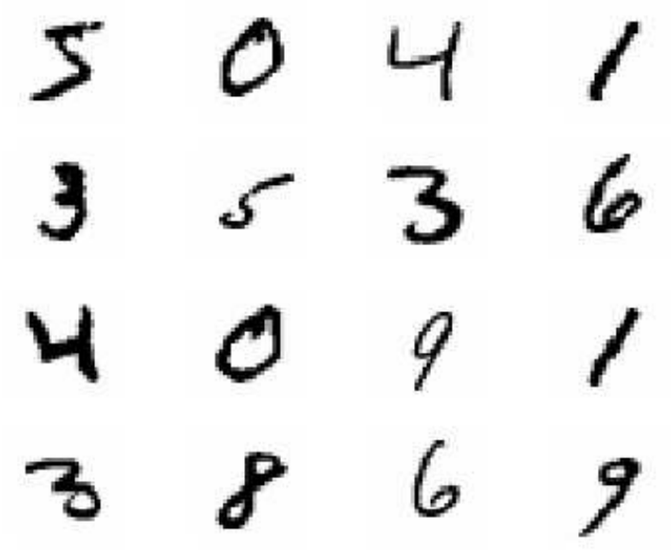
- Invariance under **translations**
- Invariance under **scaling**





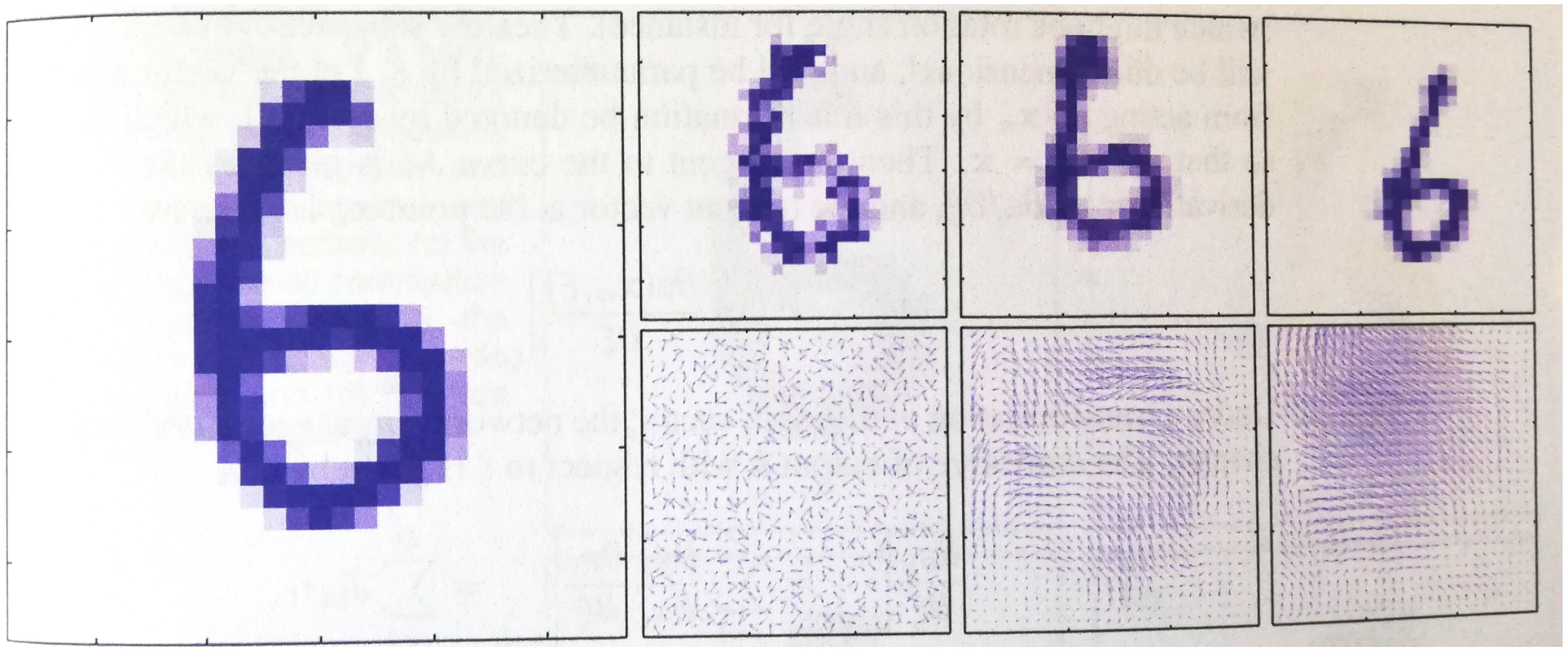
Learning with symmetry

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**

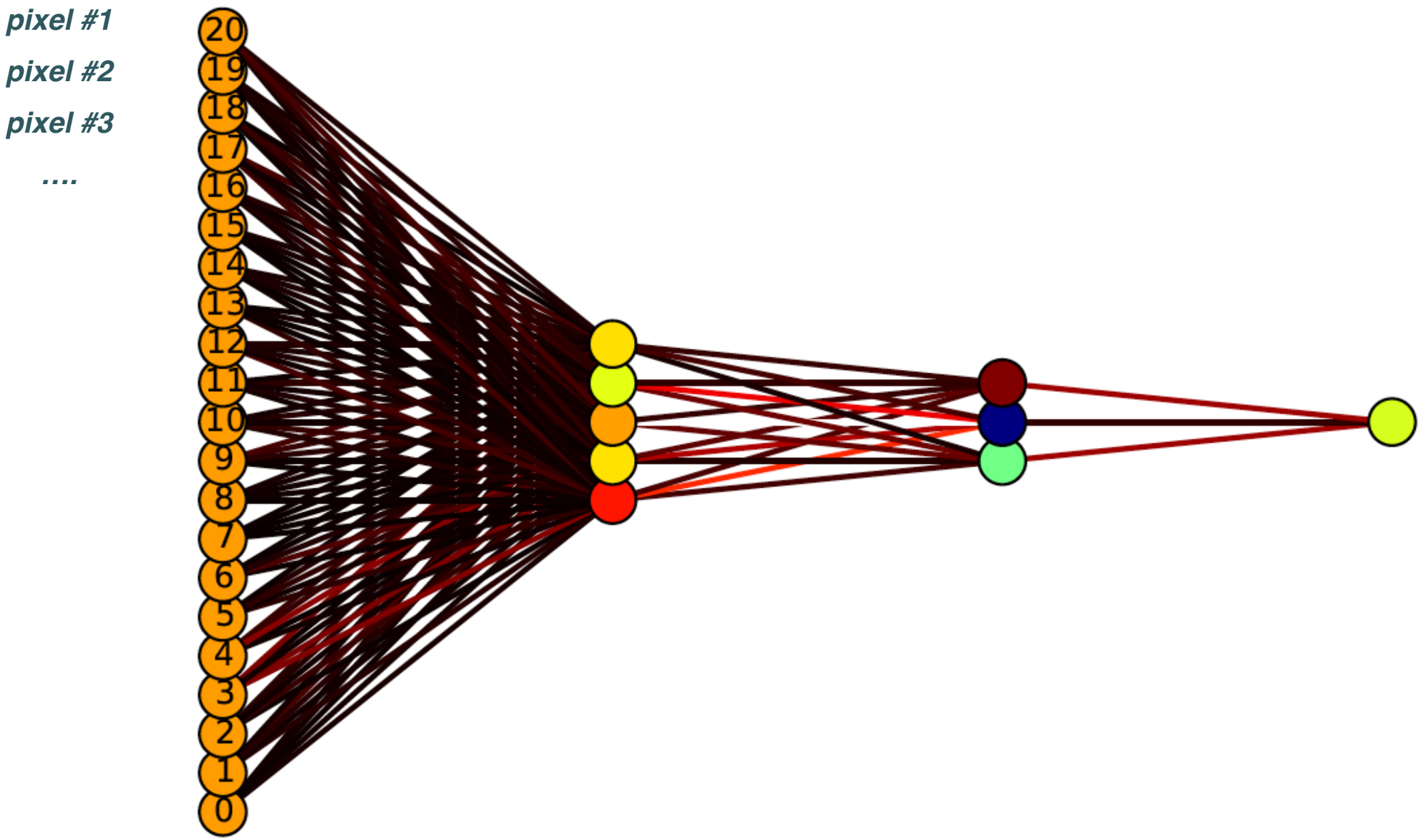
but others are more non-trivial!



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

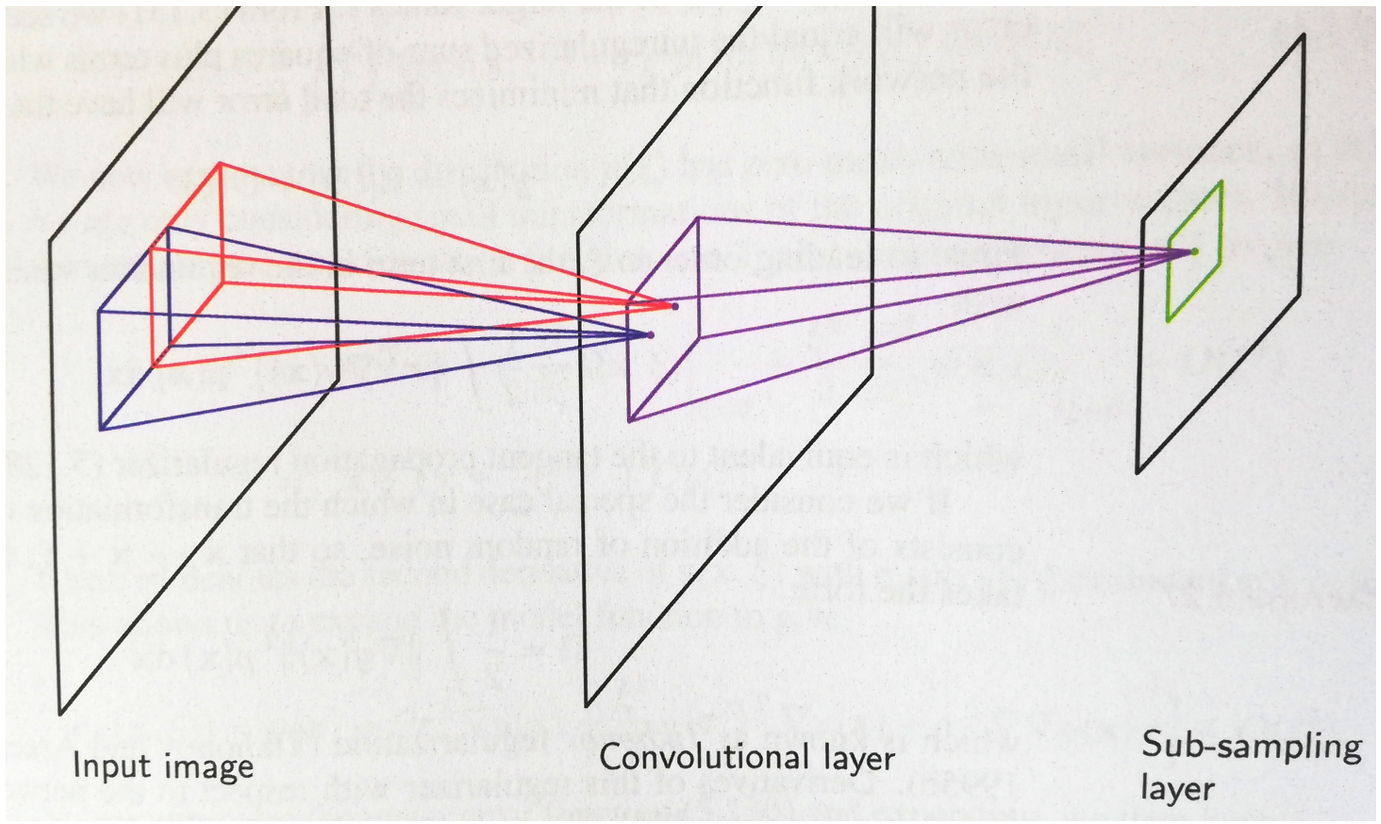
the simplest approach would be to input the images to a **fully connected NN** which given enough training data (and time) would **learn the symmetries by example**

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

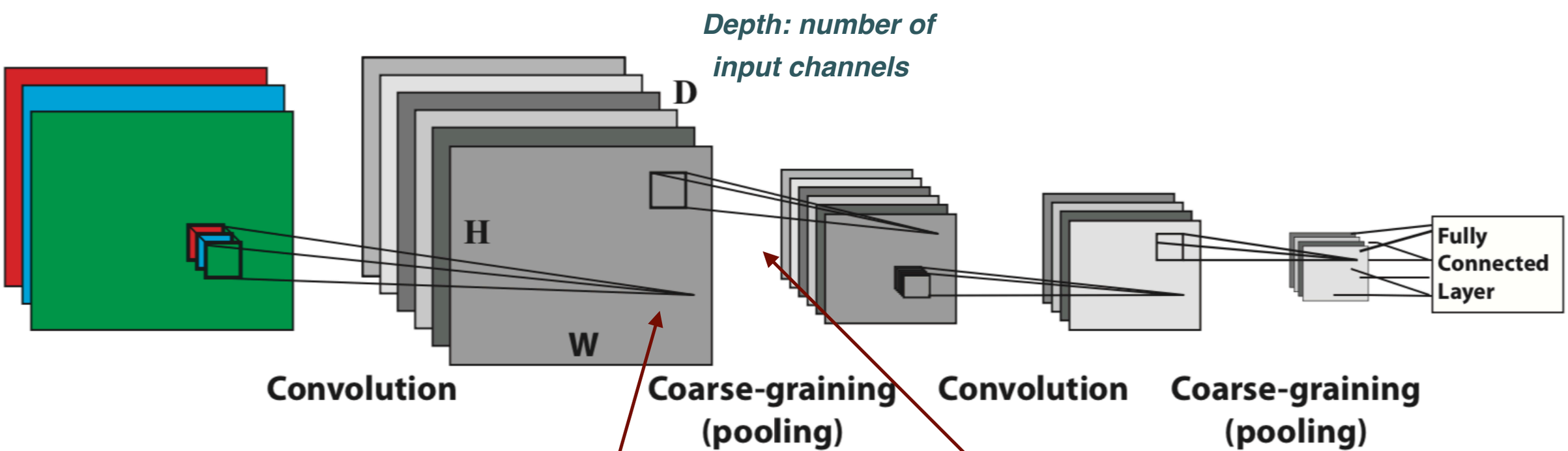
afterwards such local features can be combined into **higher-level ones**

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

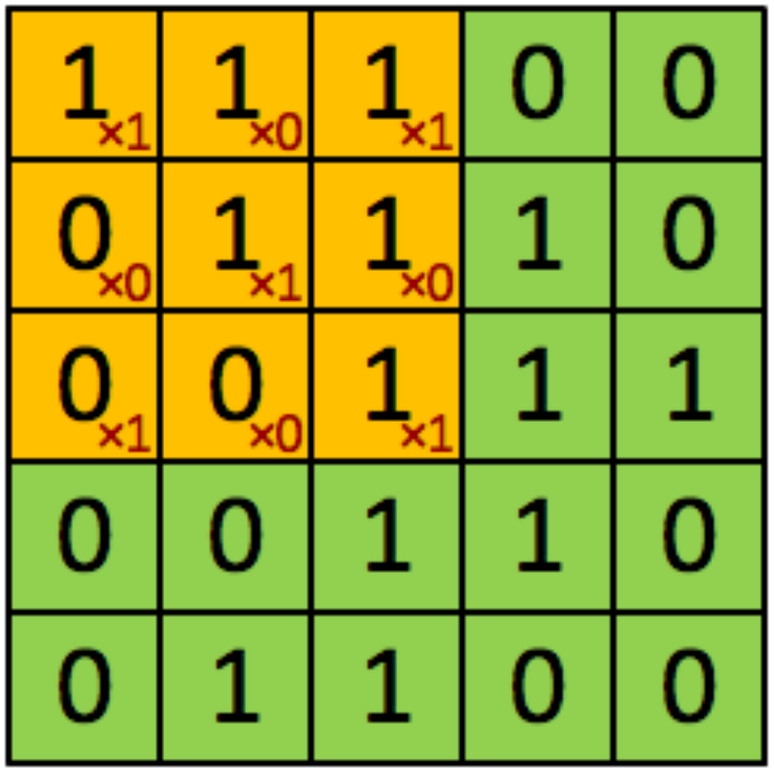


D=3 for RGB images
Height (H) and Width (W) determined by number of pixels

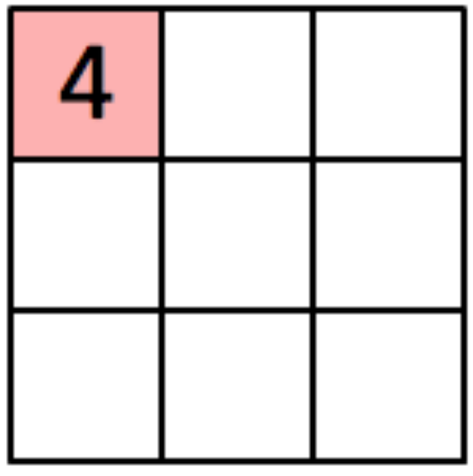
neuron activation state: convolution with local spatial filter (eg 3 x 3 pixel grid)

pooling layers reduce H, W while preserving D

Convolutional Layer



Image



Convolved Feature

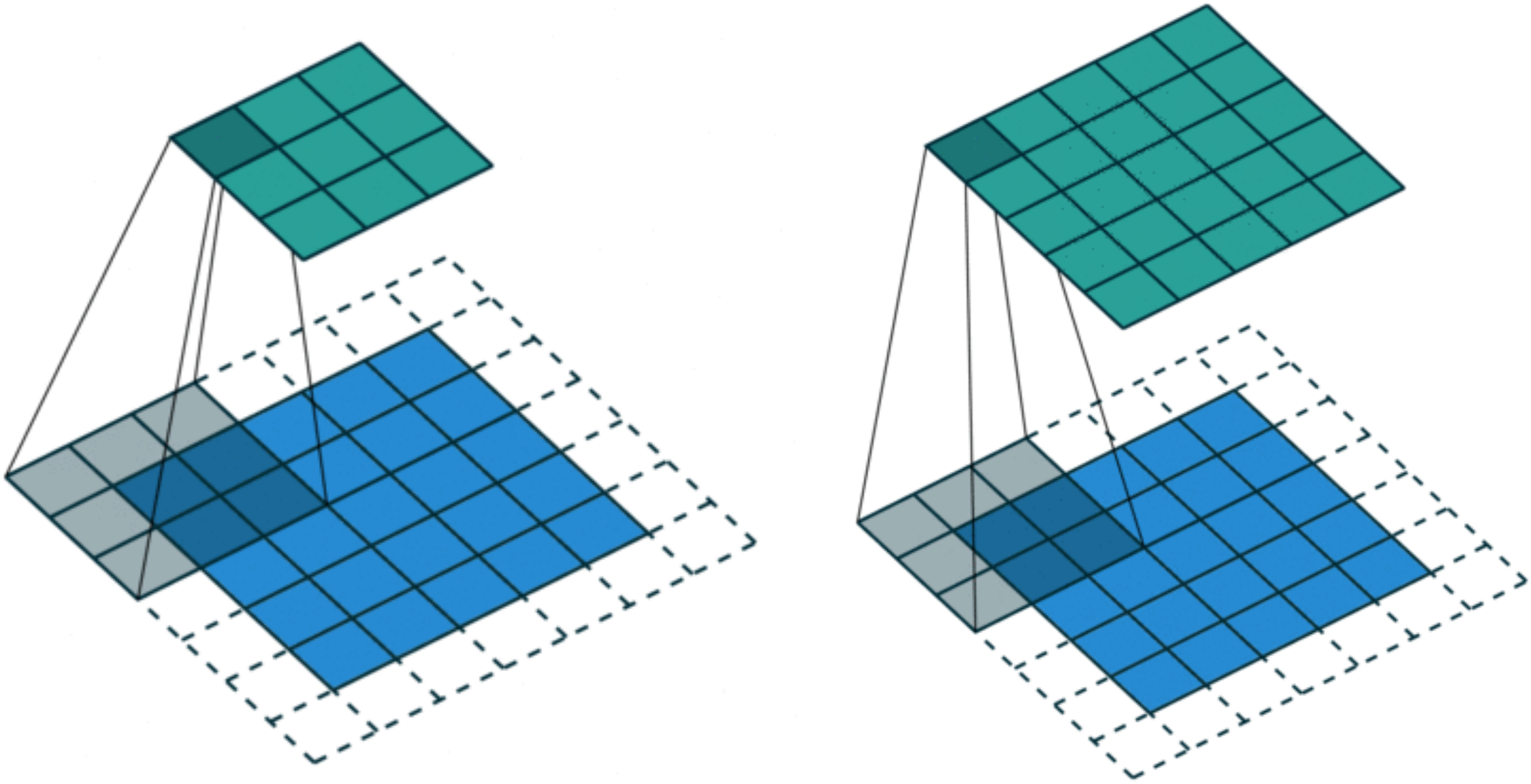
$$\text{Kernel} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

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?

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

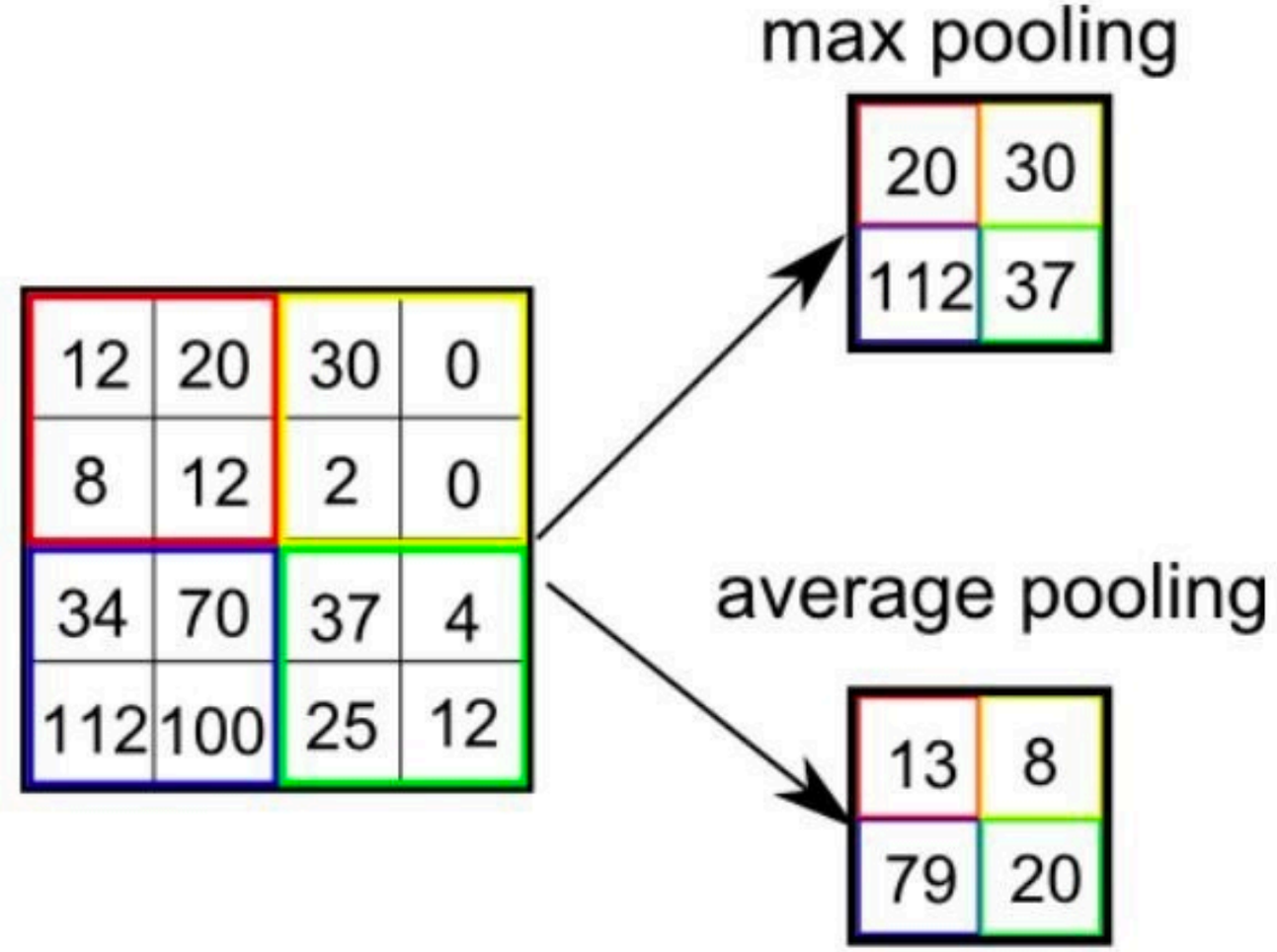
Convolutional Neural Networks

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

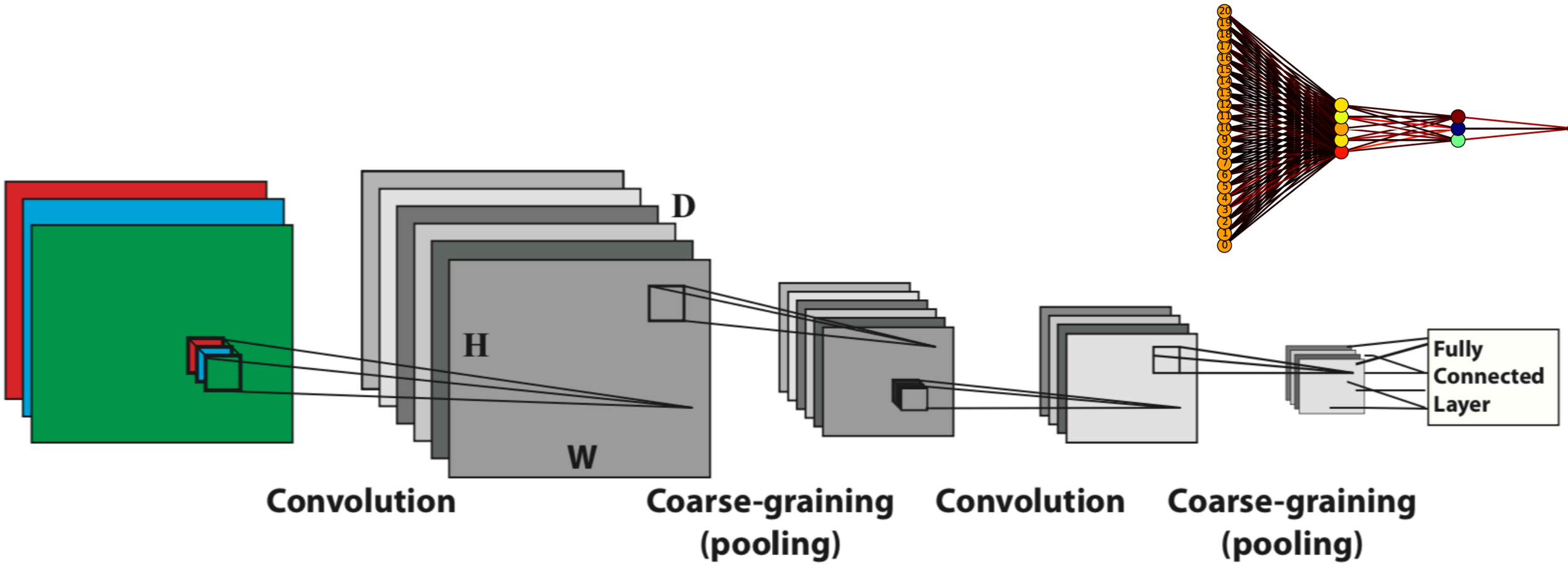
Average Pooling

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

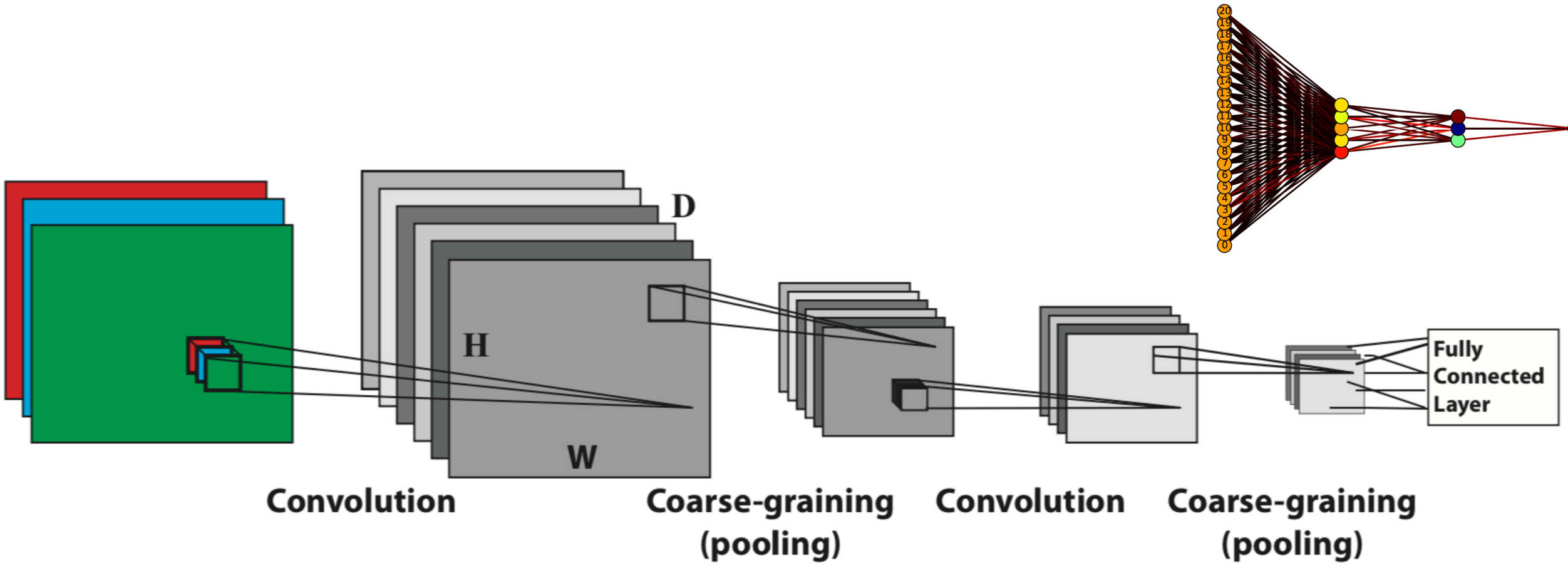


After convolution + pooling layers, we can flatten our images and input them to a regular DNN!

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

Convolutional Neural Networks

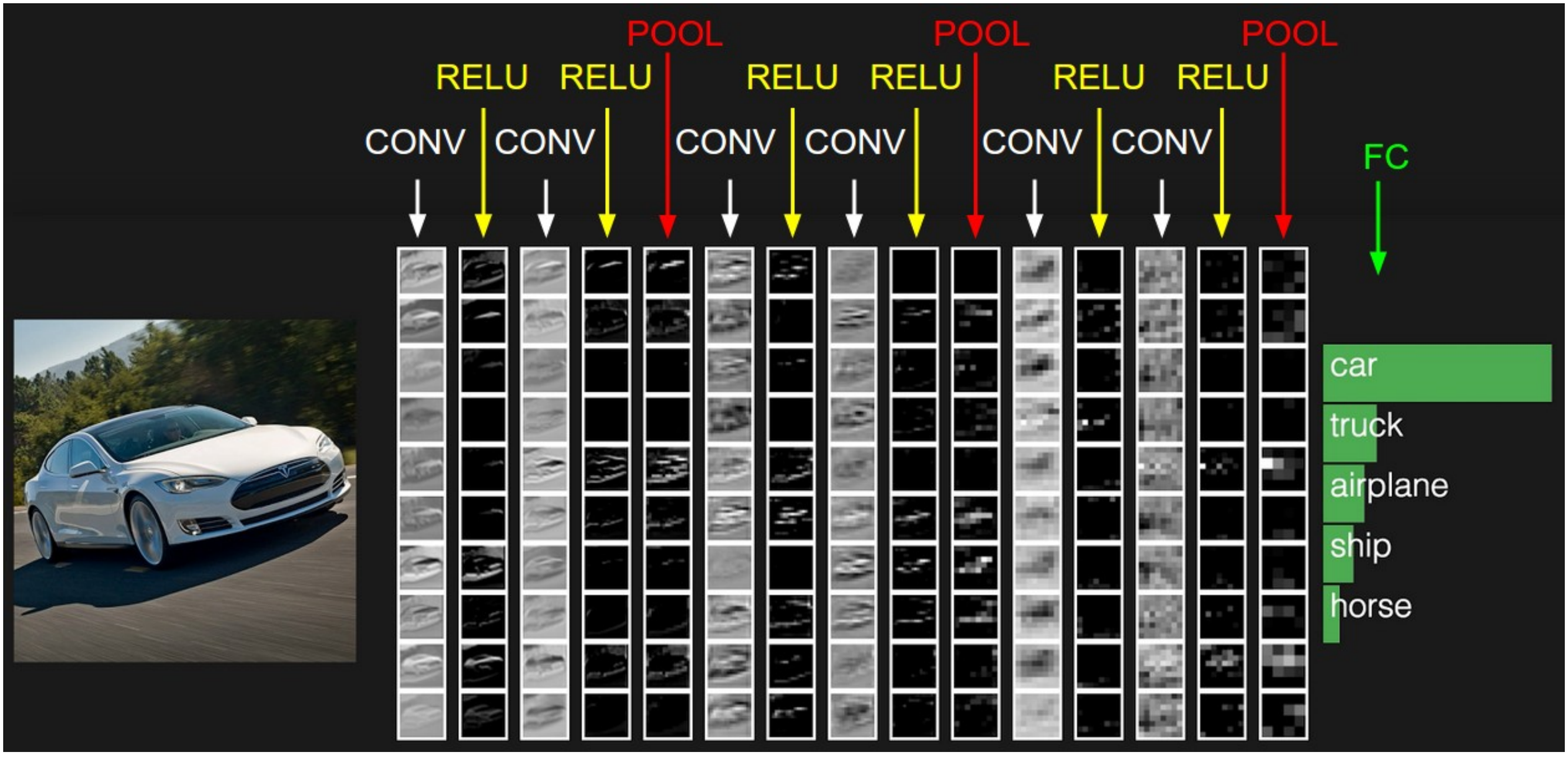
the convolution and pooling layers are followed by an **all-to-all connected layer and a high-level classifier**, so that one can train CNNs using the standard backpropagation algorithm



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

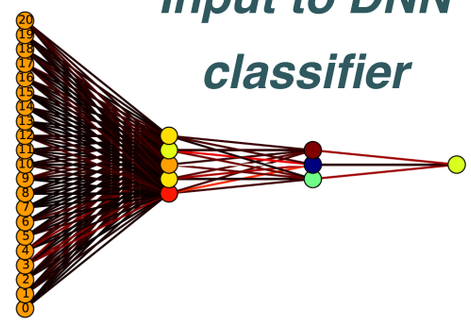
Q2: Do you think we can tailor kernels/filter to identify specific features of a pattern?

Convolutional Neural Networks

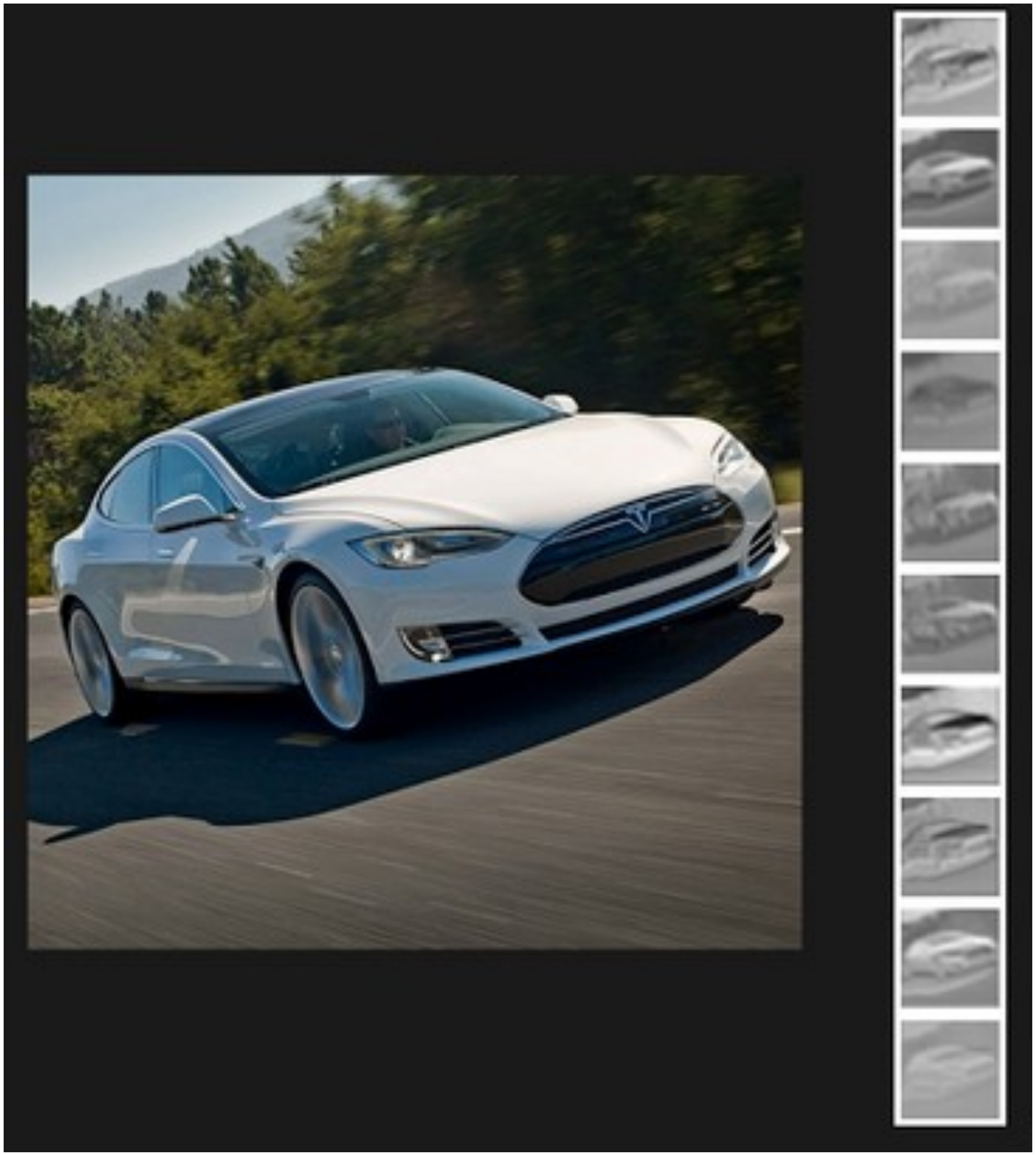


original image

Input to DNN classifier

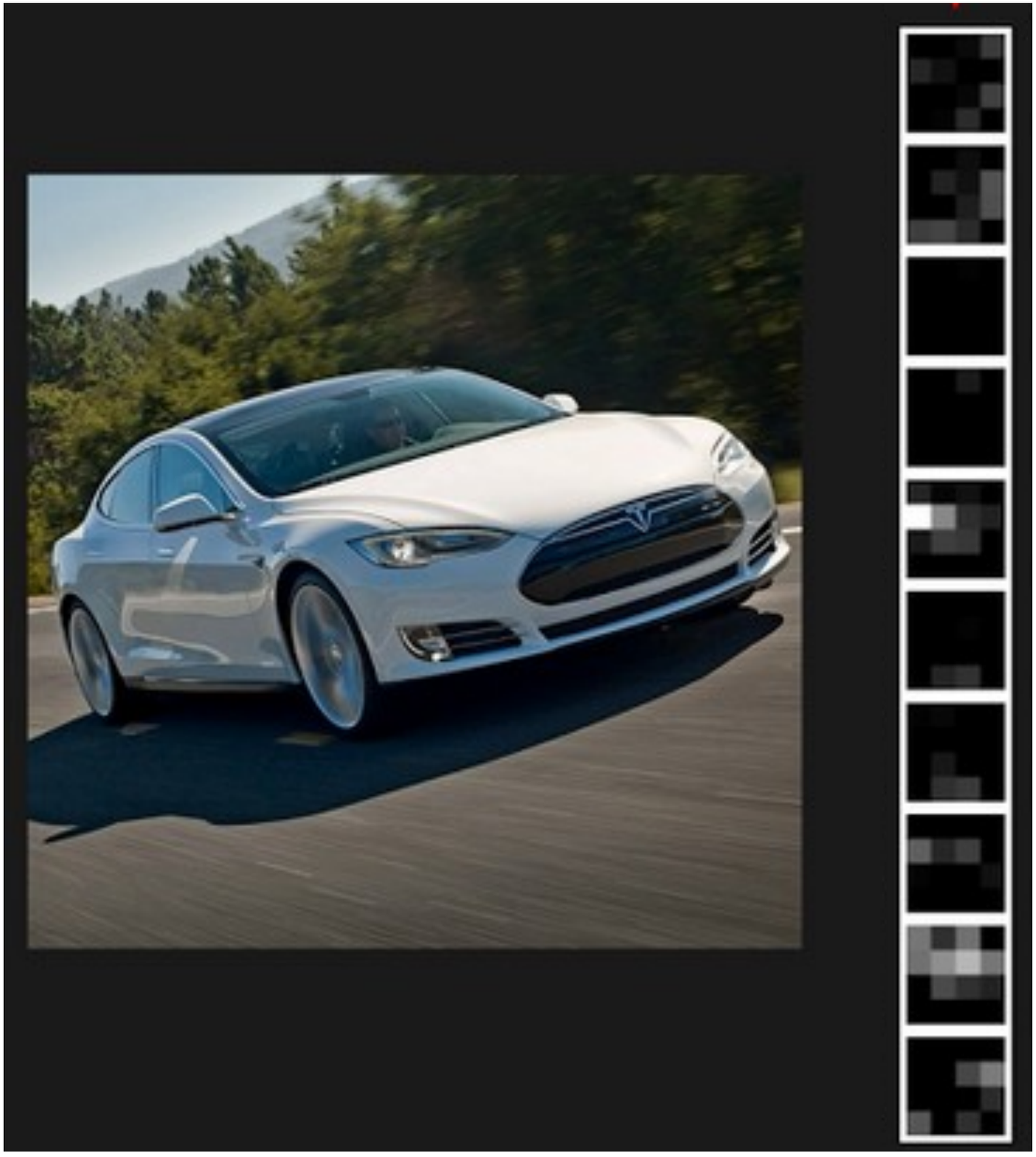


Convolutional Neural Networks



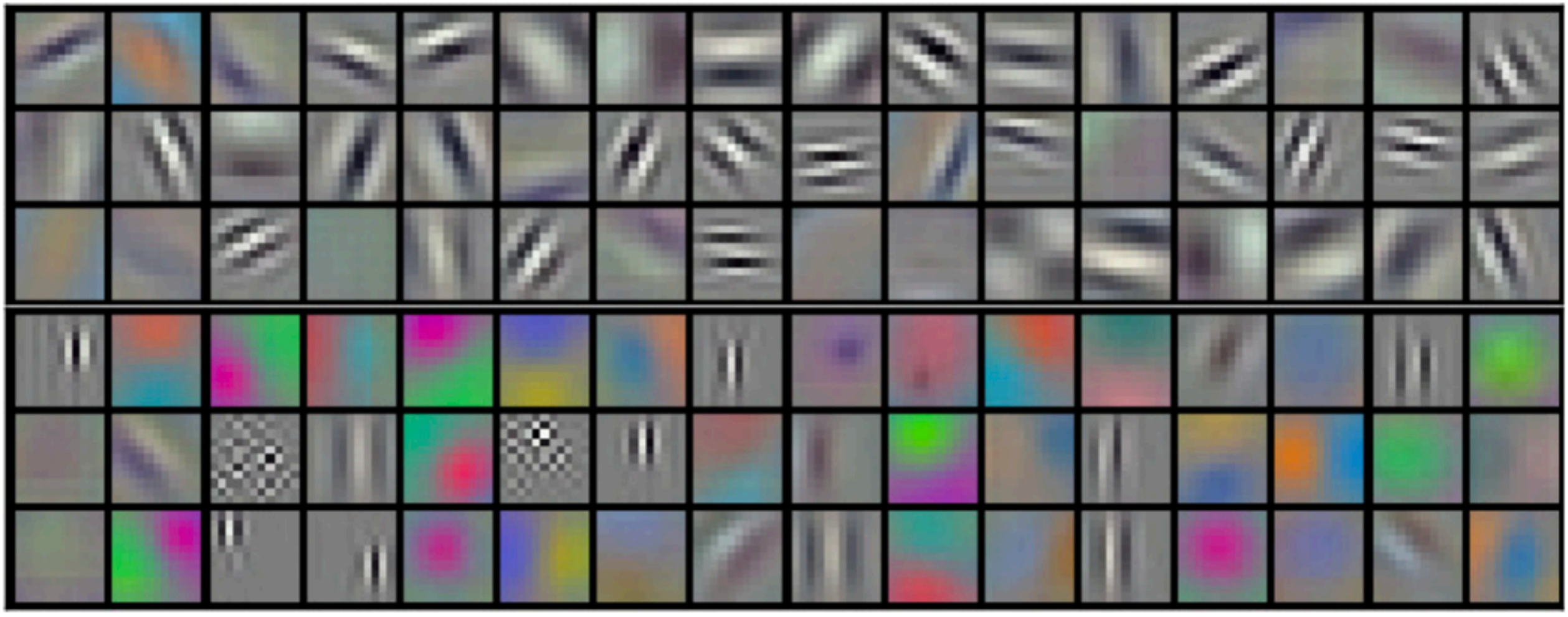
first convolutional layer: identify edges, surfaces, contrast differences, kinks …

Convolutional Neural Networks



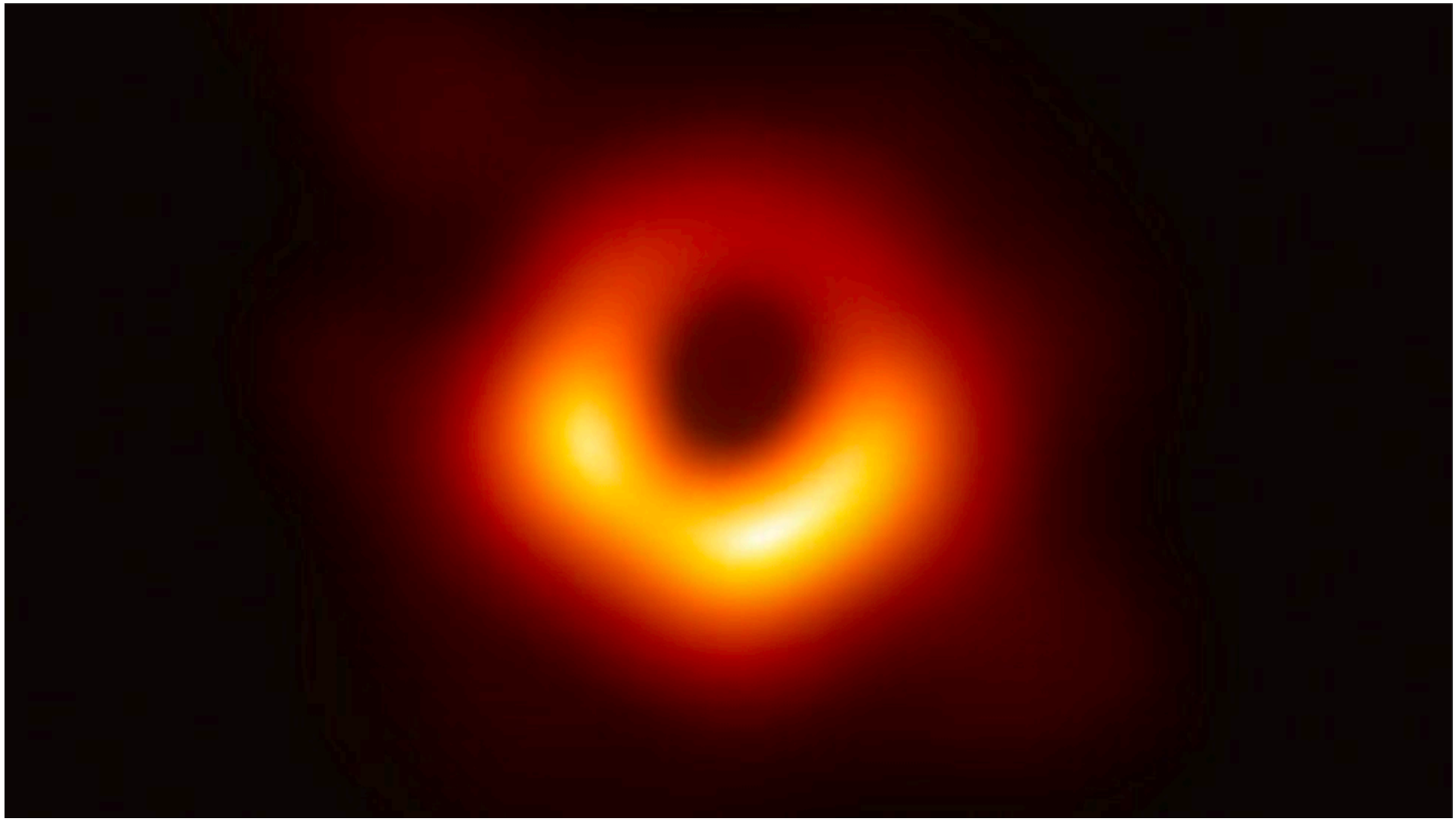
final convolutional layer: high-level features of ``car'' to be feed to the DNN classifier

Convolutional Neural Networks



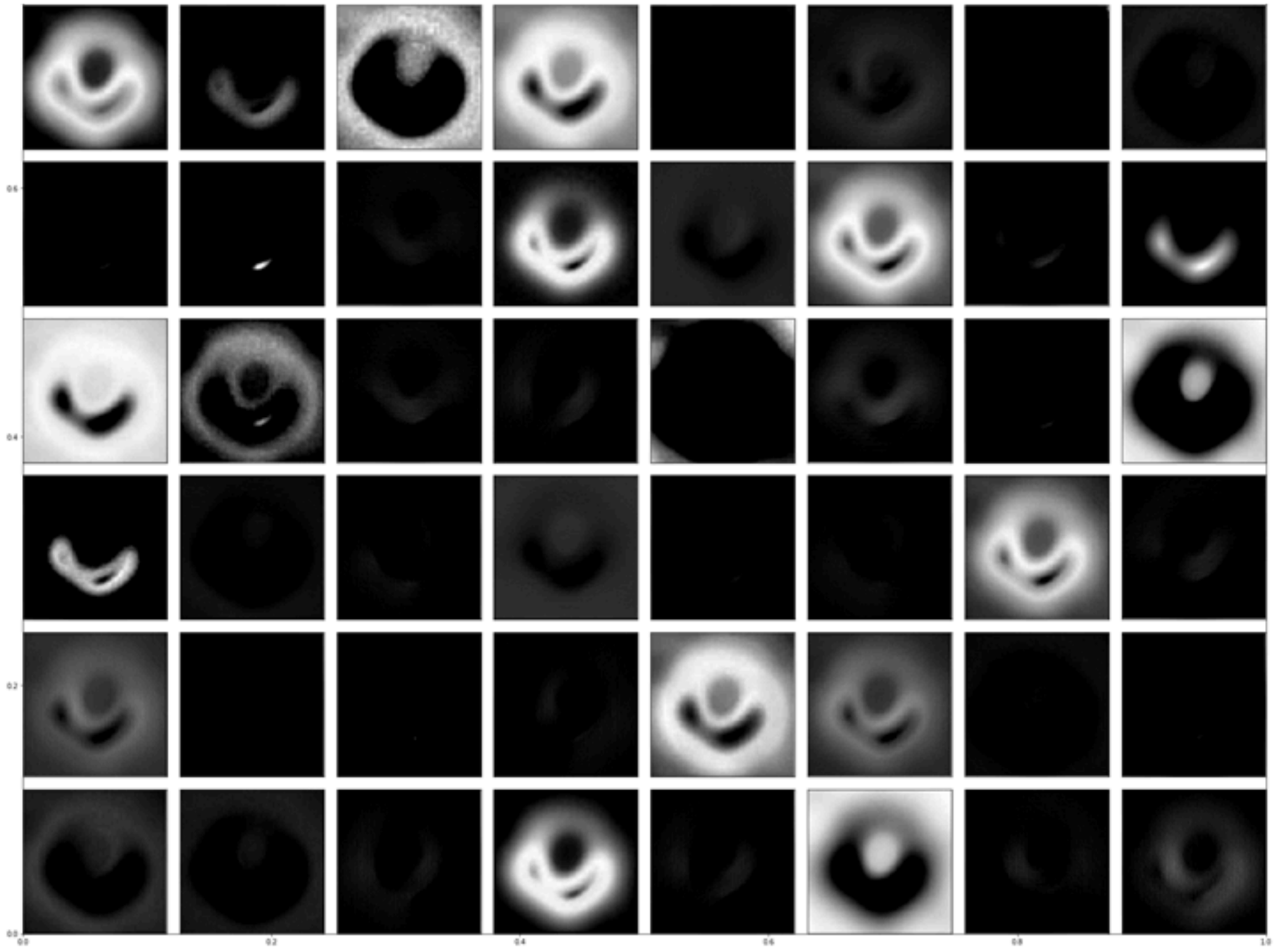
We can choose filters of a CNN to highlight specific colour combinations, lines in specific directions, edges or angles of a given size ….

CNNs in Physics and Astronomy



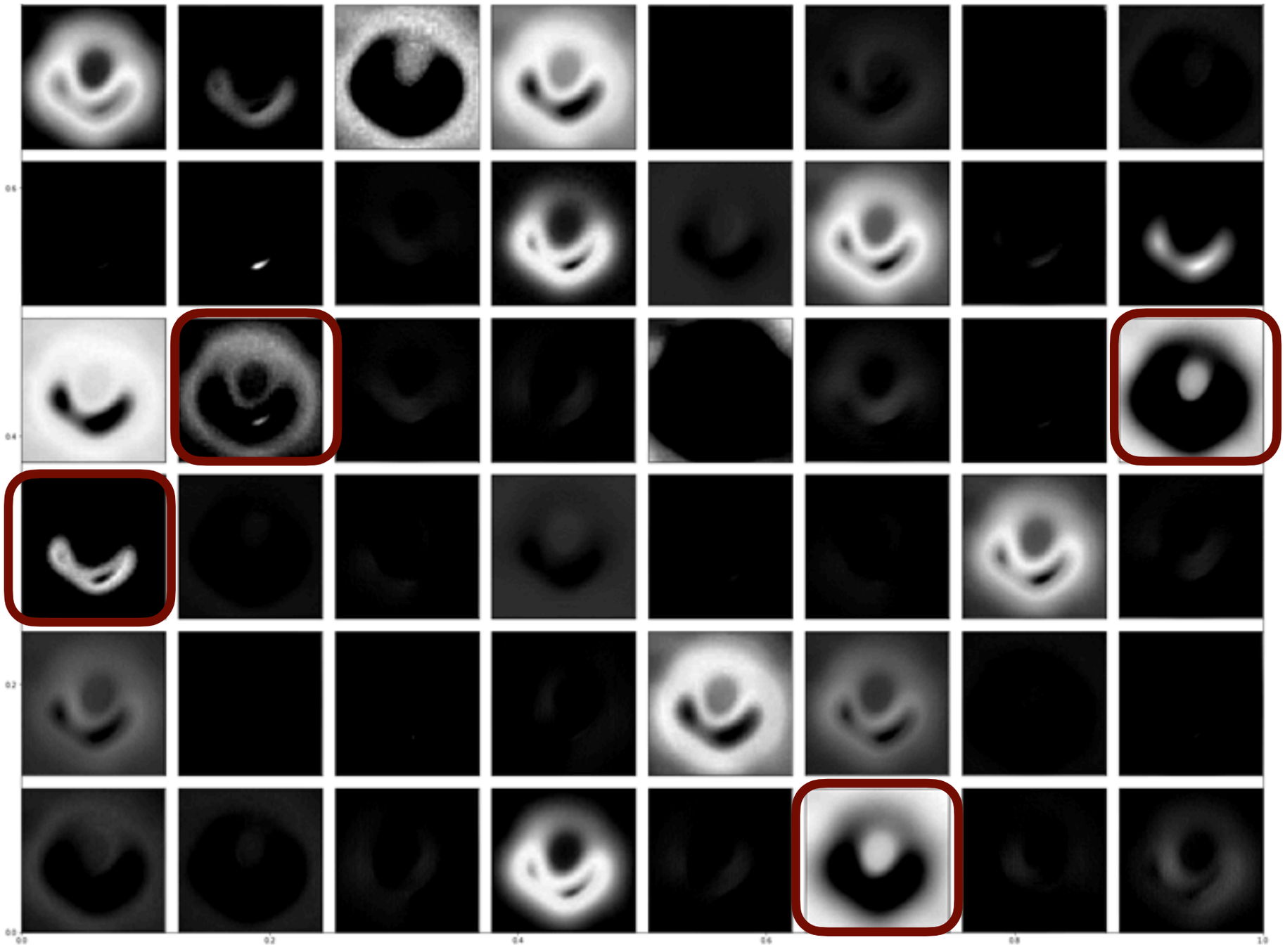
What is this image?

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

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)

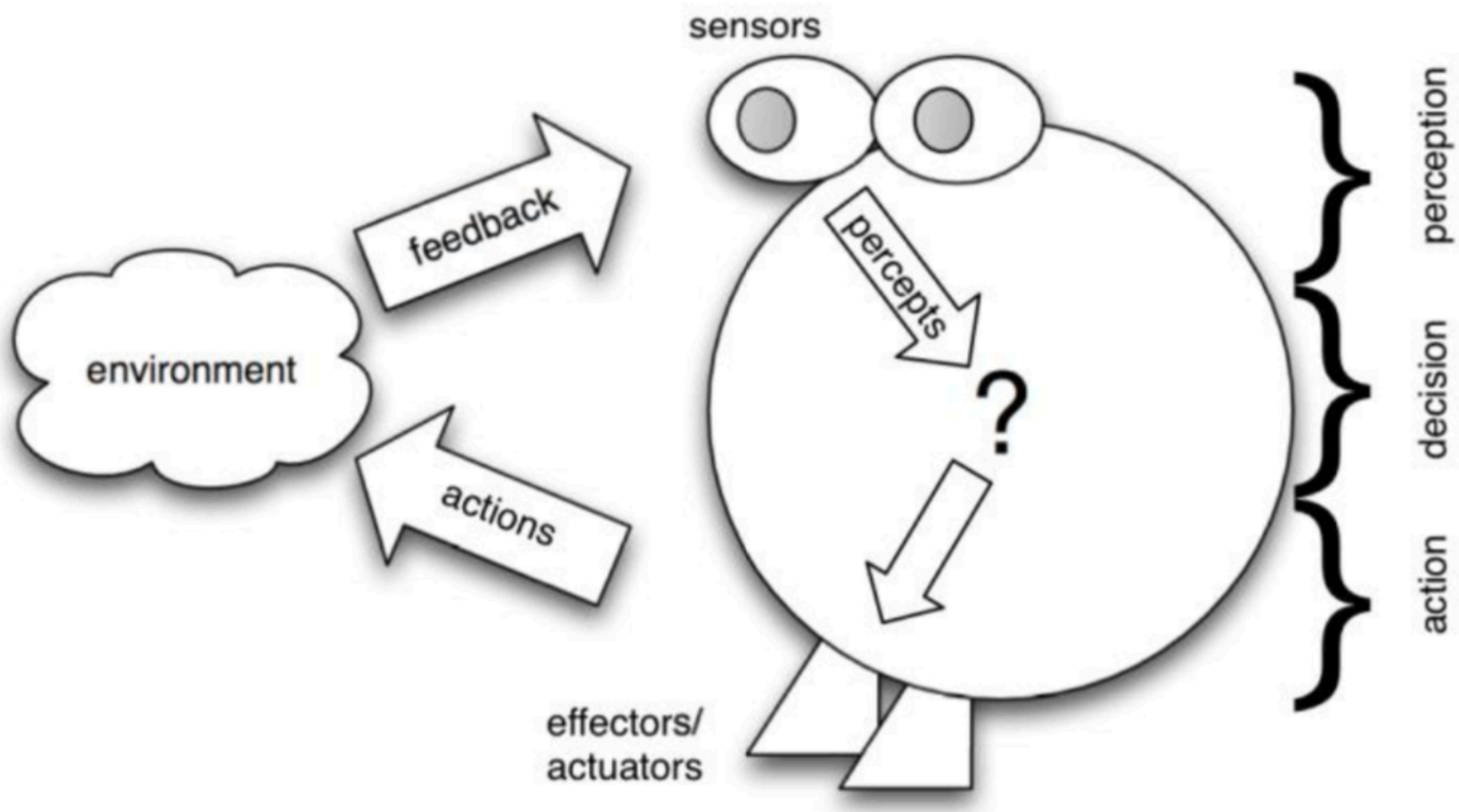
Unsupervised Learning: starting from a training dataset with **unlabelled examples,** $\{x_i\}_{i=1,N}$, produce a **model** that takes a sample as input and as output produces the solution of a practical problem, such as **clustering**, **dimensional reduction**, or **outlier detection**

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

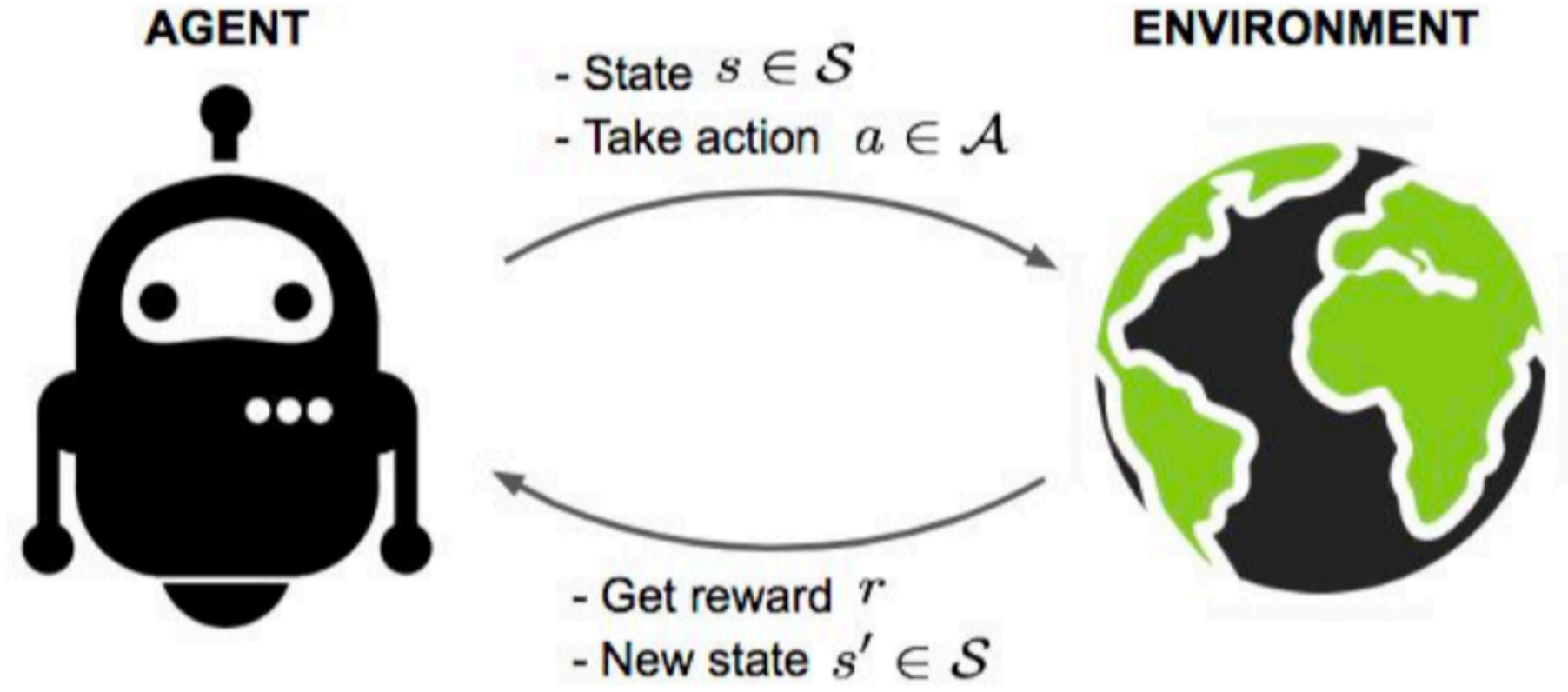
non-trivial agents should exhibit the following properties:

- ***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

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!).

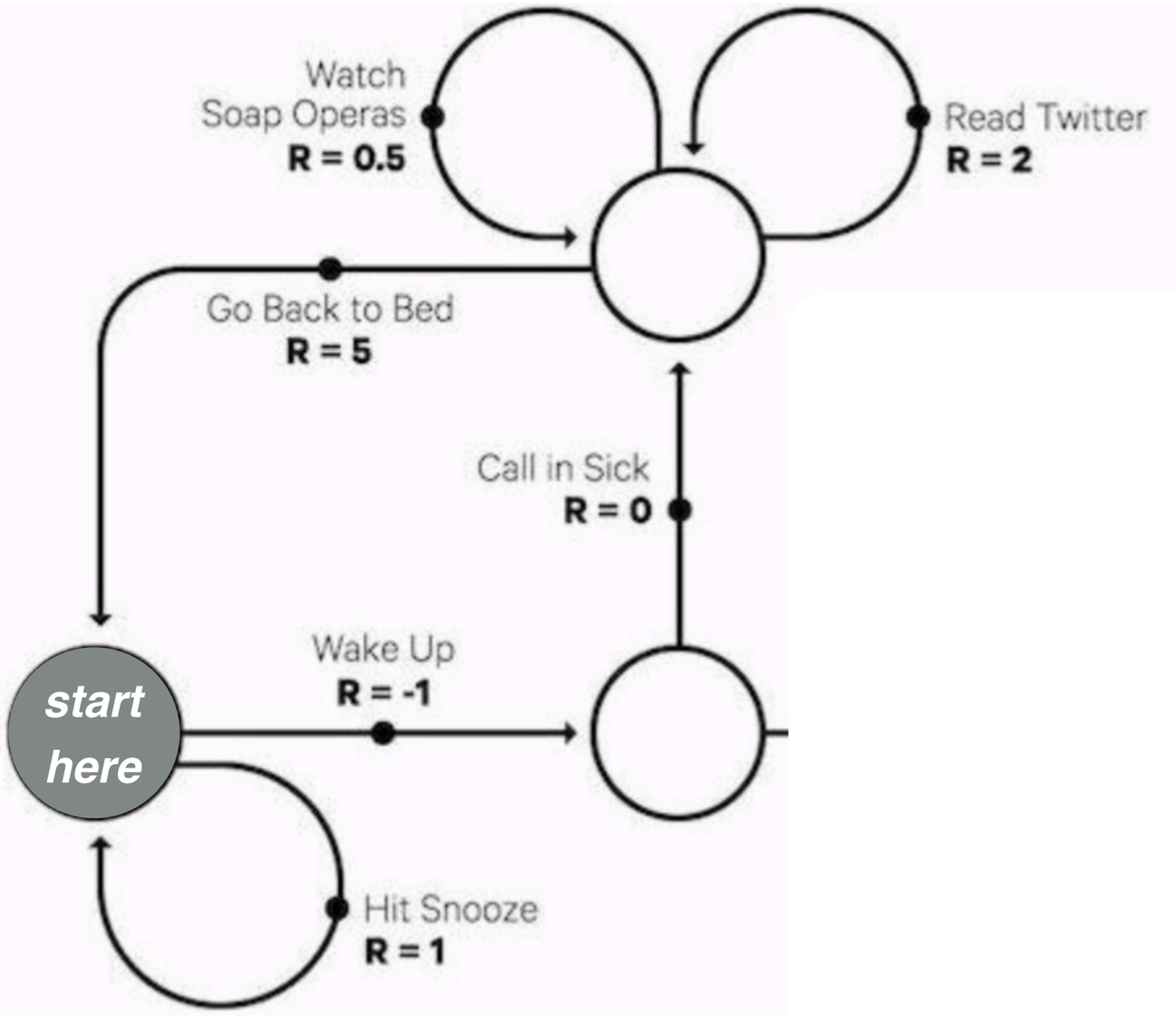


A Reinforcement Learning system

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

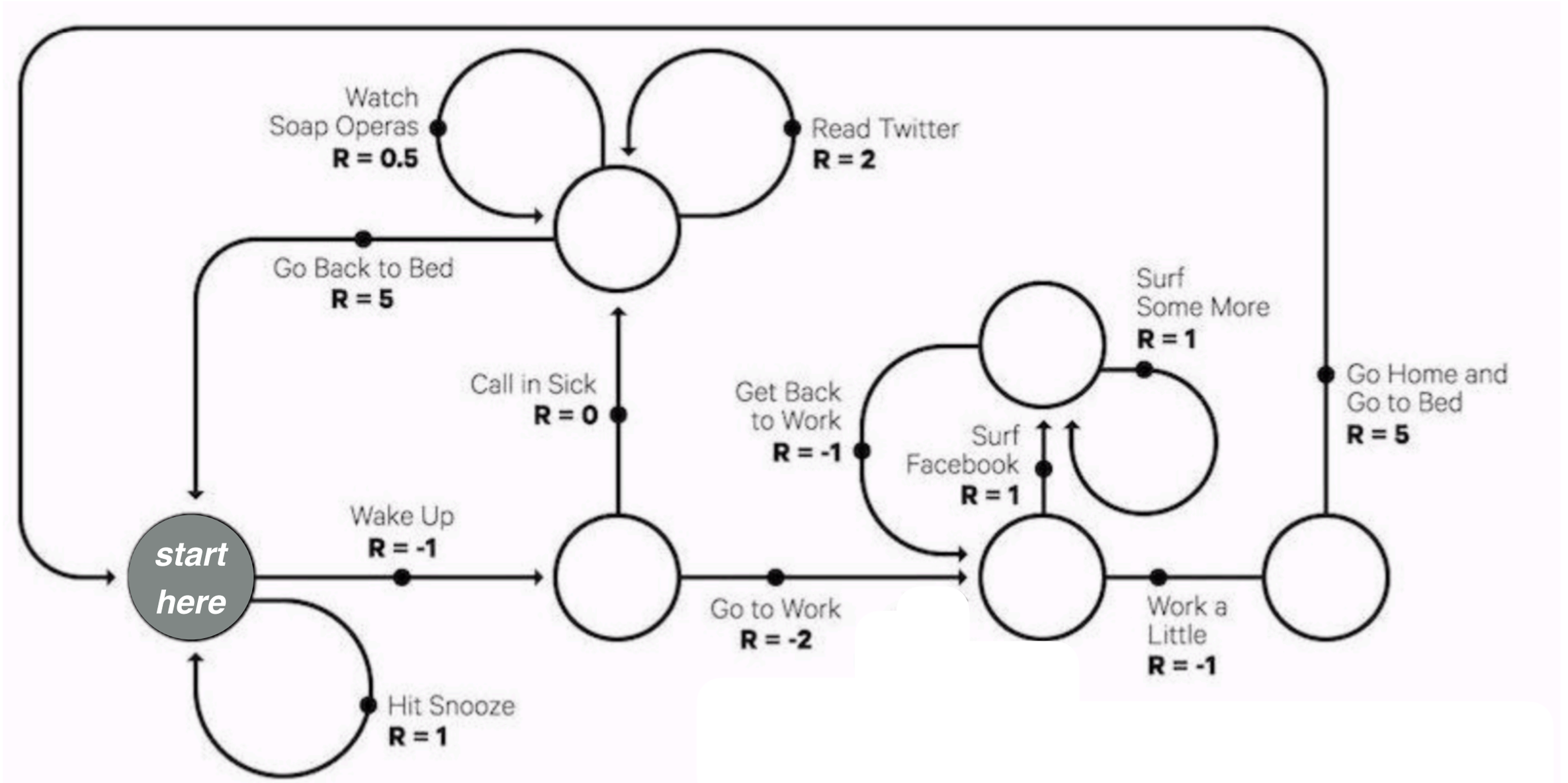
A Reinforcement Learning system



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

Q: what is the optimal outcome here?

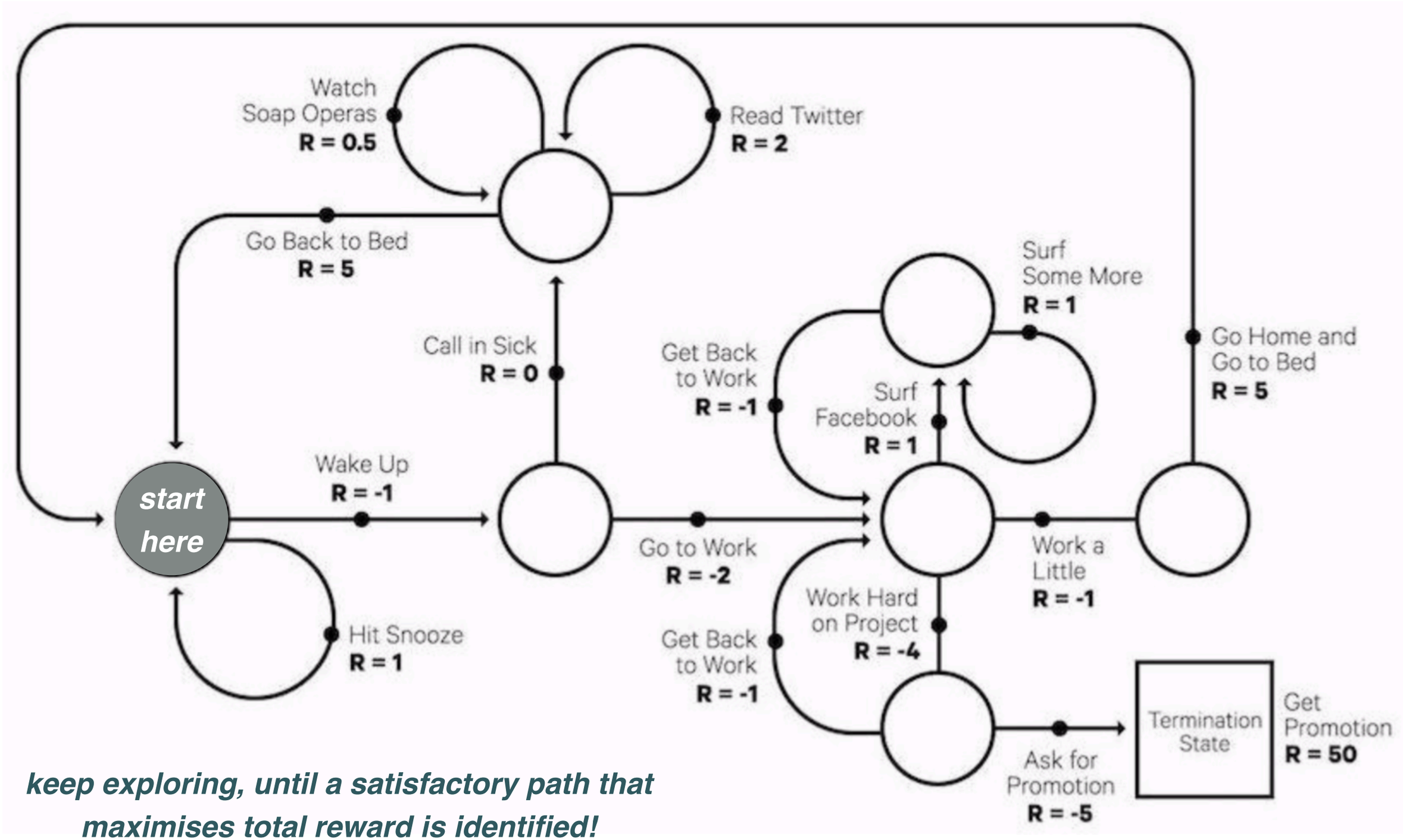
A Reinforcement Learning system



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

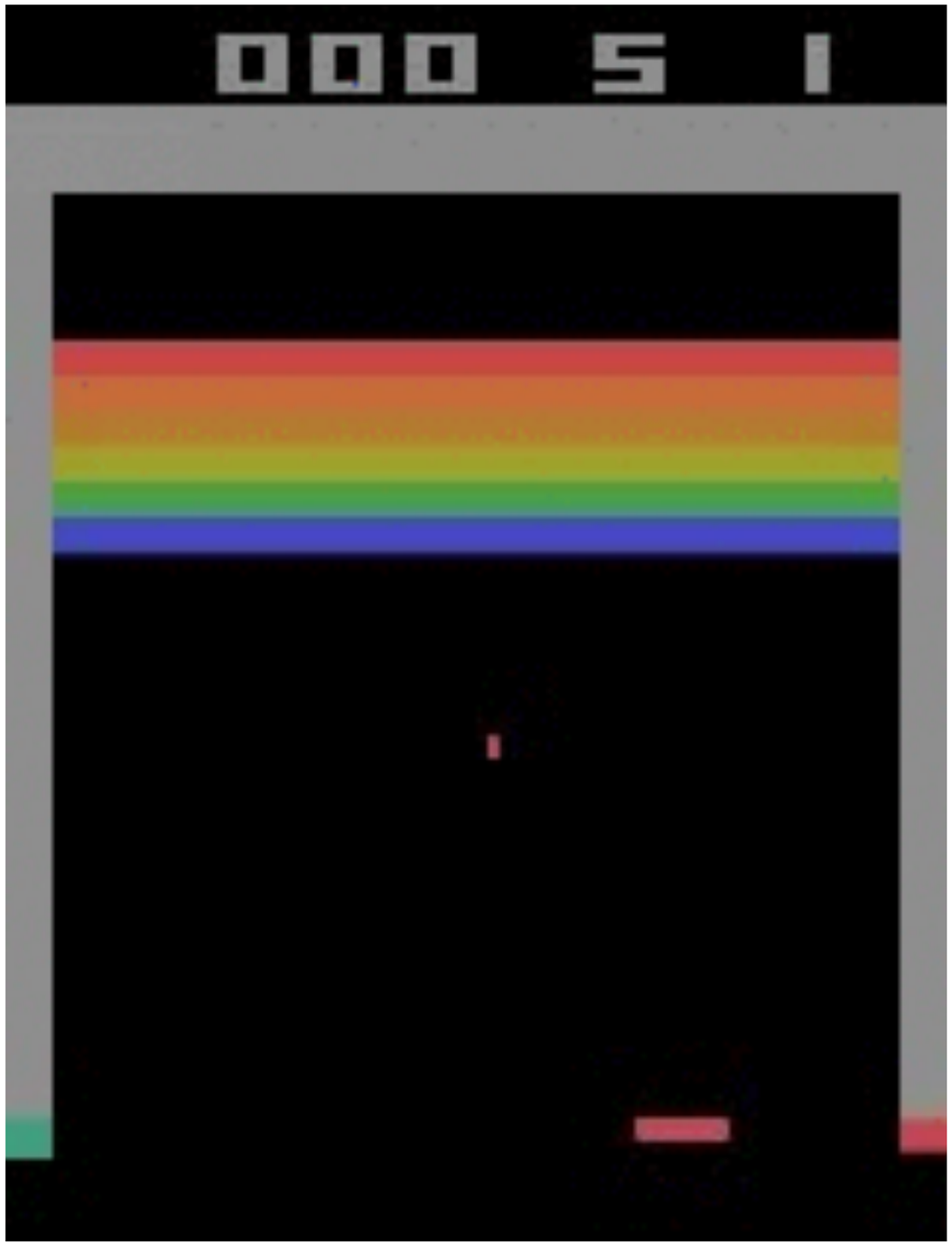
DeepMind AlphaStar: AI breakthrough or pushing the limits of reinforcement learning?

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

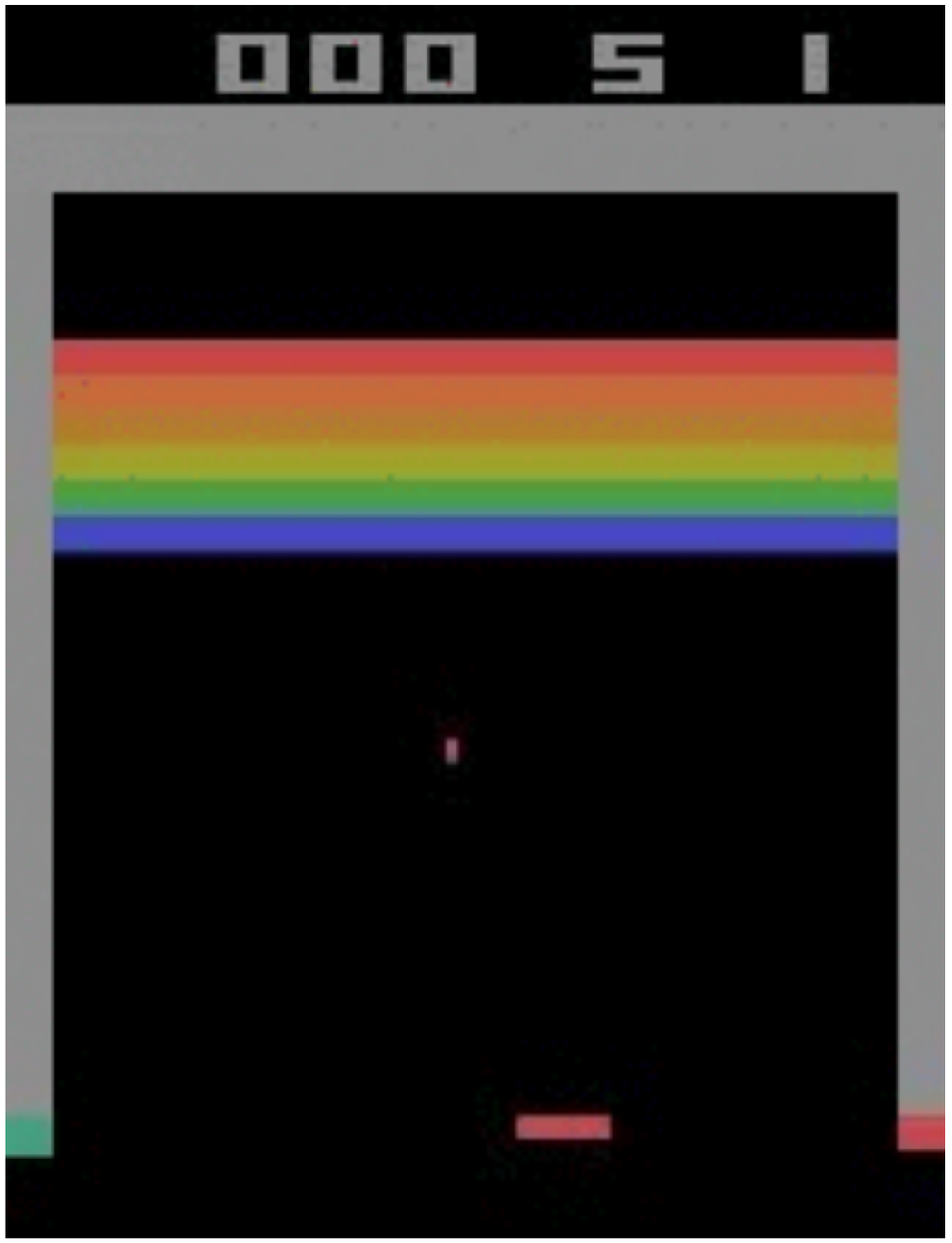
4 min read

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

Reinforcement Learning for Gaming



Initial Performance

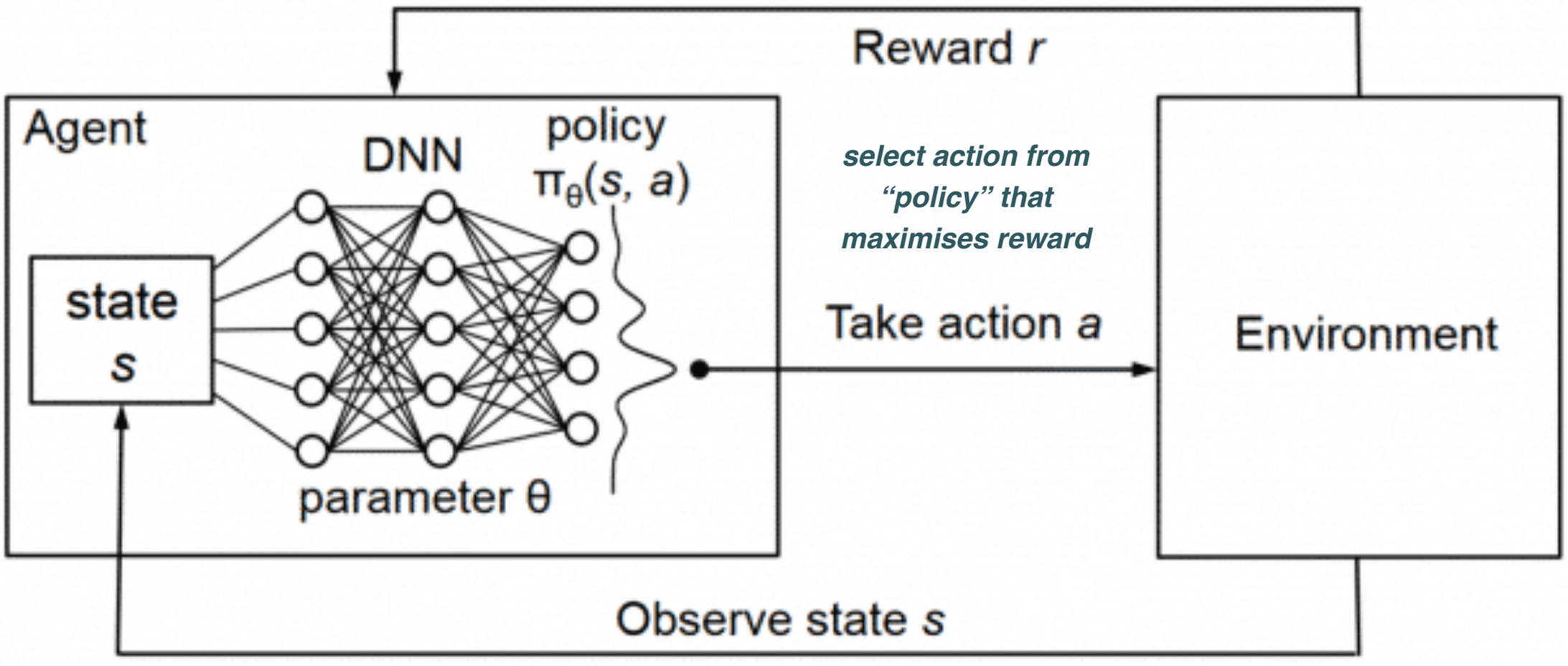


After (RL) Training

Reinforcement Learning for Gaming

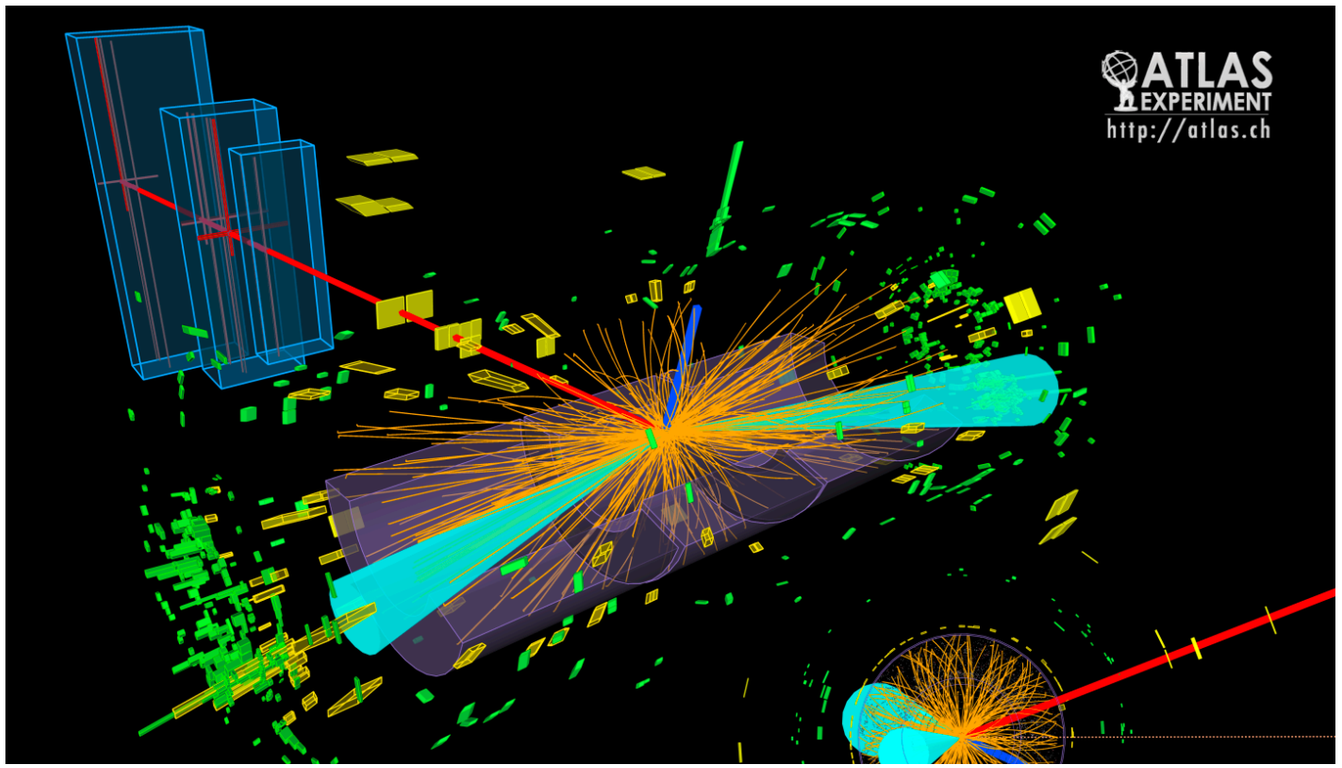
Deep Q-Networks

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**



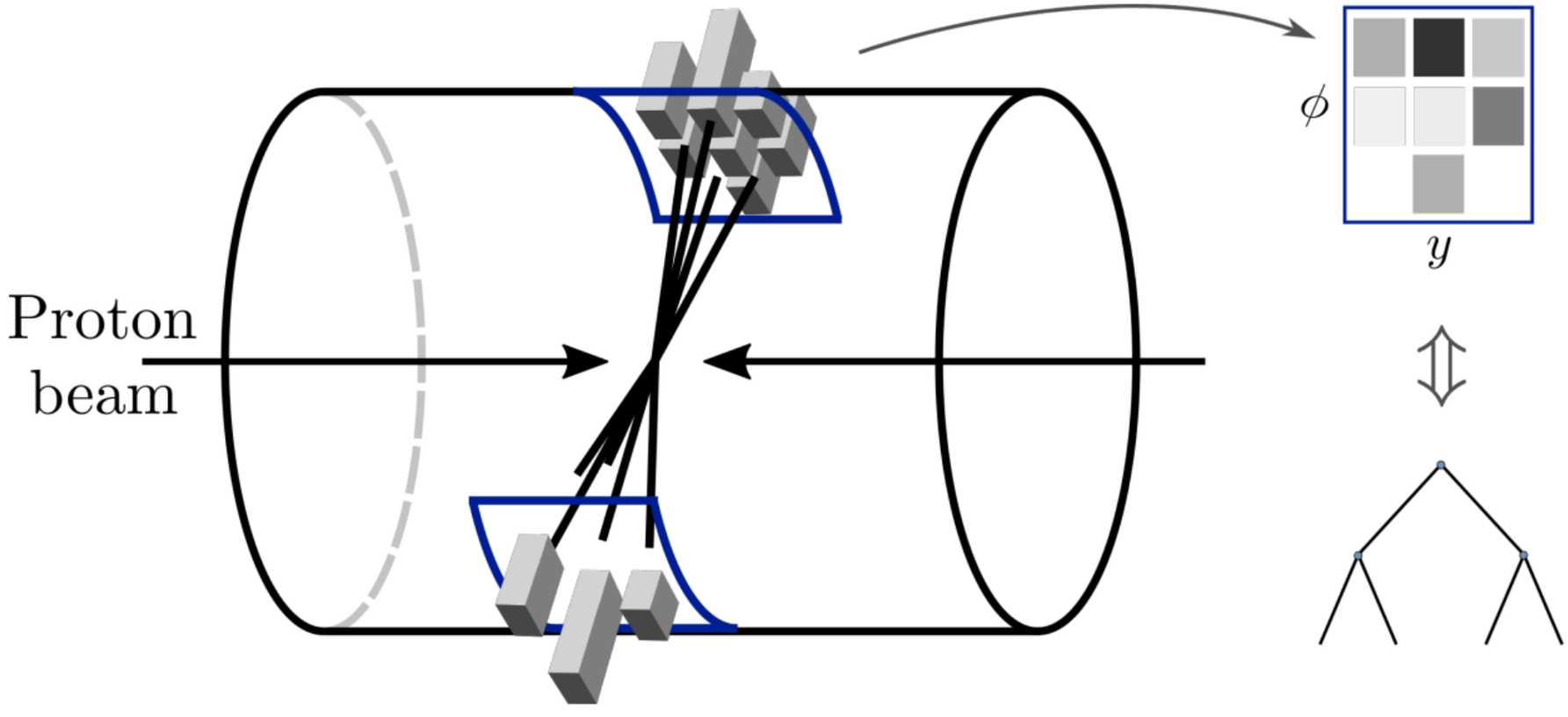
The DNN model parametrises the probability distribution associated to each possible action, given a state of the system

Reinforcement Learning in Physics



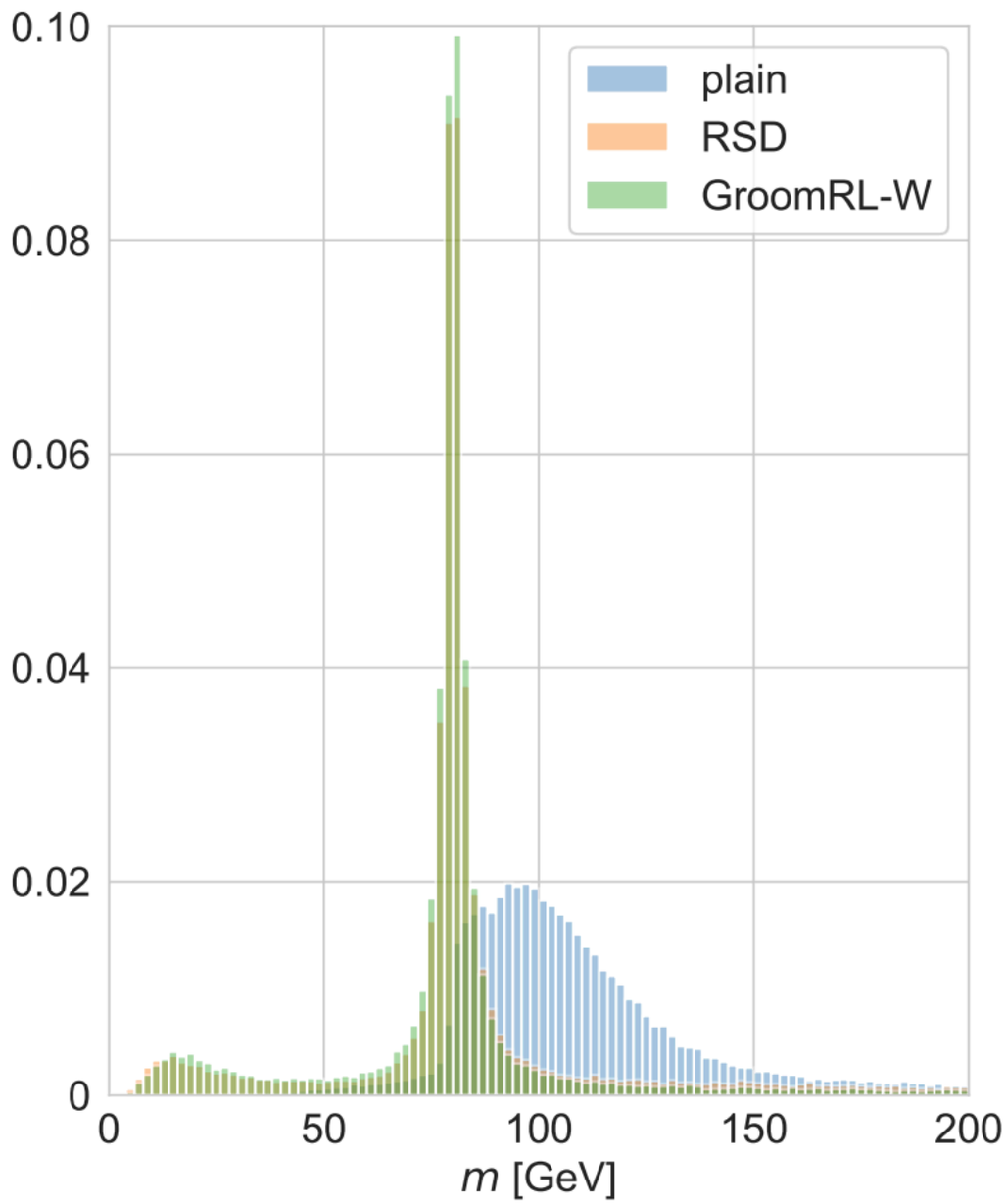
collisions at the LHC often result into sprays of collimated particles: jets

Jets can be mapped into images and processed with reinforcement learning

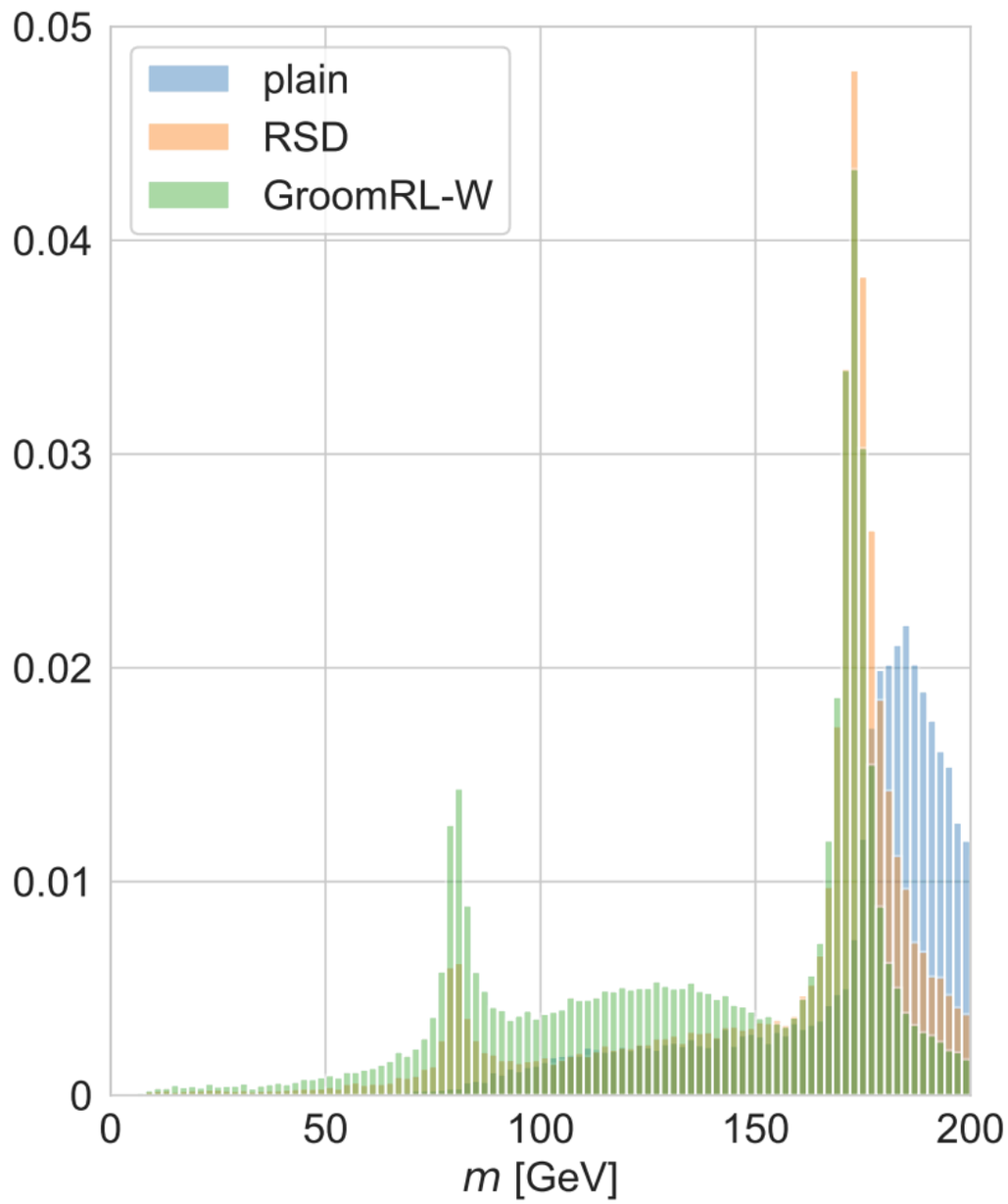


Goal: learn a policy that removes ``background'' particles from the jet

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)

Summary

- Machine learning techniques are by now **ubiquitous** in (astro-)particle physics and in gravitational wave astronomy
- Not only they make possible a **performance speedup** and **efficiency gain** for existing tasks, their availability realises **new scientific goals** thought to be unattainable
- Up to here we have presented the **basics of ML algorithms**: in the following, specific **applications to Nikhef research** in ATLAS, LHCb, and GWs!

Thursday 9th June

- Morning session II: machine learning for **LHCb/flavor physics** (Jacco de Vries)
- Afternoon session I: hands-on tutorials with Jupyter notebooks (unsupervised learning) (Tanjona Rabemananjara)
- Afternoon session II: hands-on tutorials with Jupyter notebooks (ML for LHCb/flavor physics)

Friday 10th June

- Morning session I: CNNs in **gravitational wave physics** (Amit Reza)
- Morning session II: machine learning for **LHC high-pT physics** (Johnny Raine)
- Afternoon session I: hands-on tutorials with Jupyter notebooks (CNNs in GWs) (Amit Reza)
- Afternoon session II: hands-on tutorials with Jupyter notebooks (ML ATLAS) (Johnny Raine)
- Drinks!