Postgraduate course
Universitat de Valencia 2020

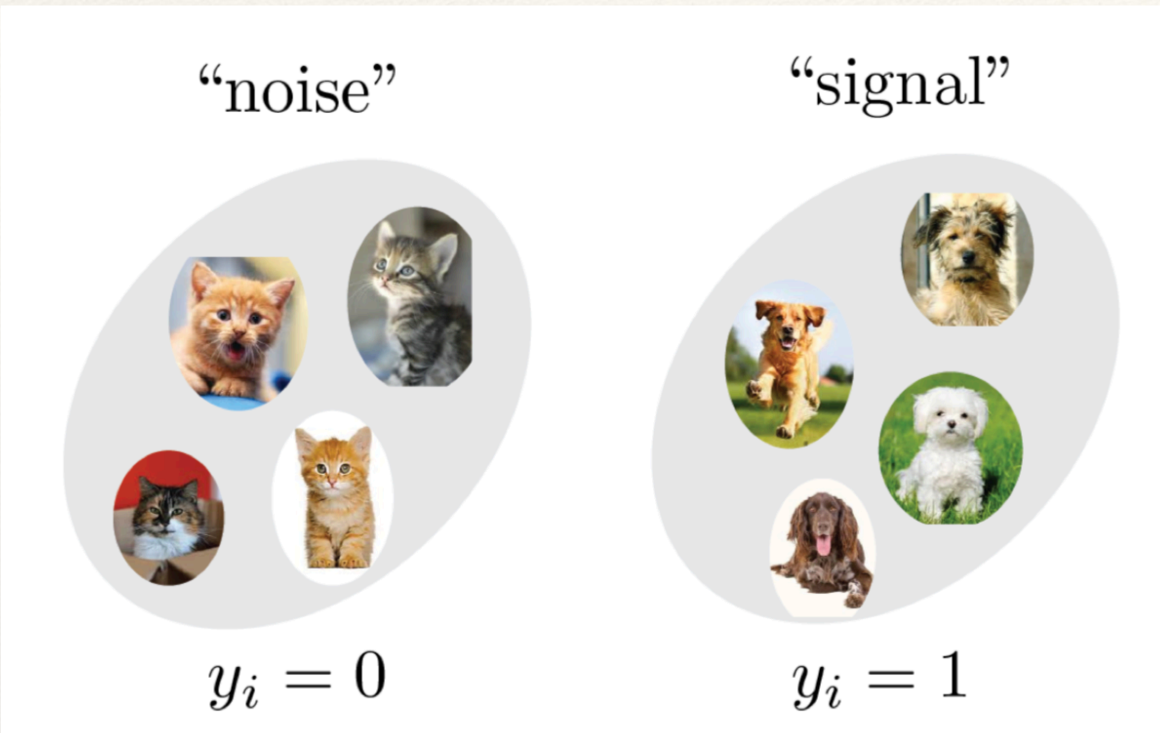
Introduction to Machine Learning for physicists

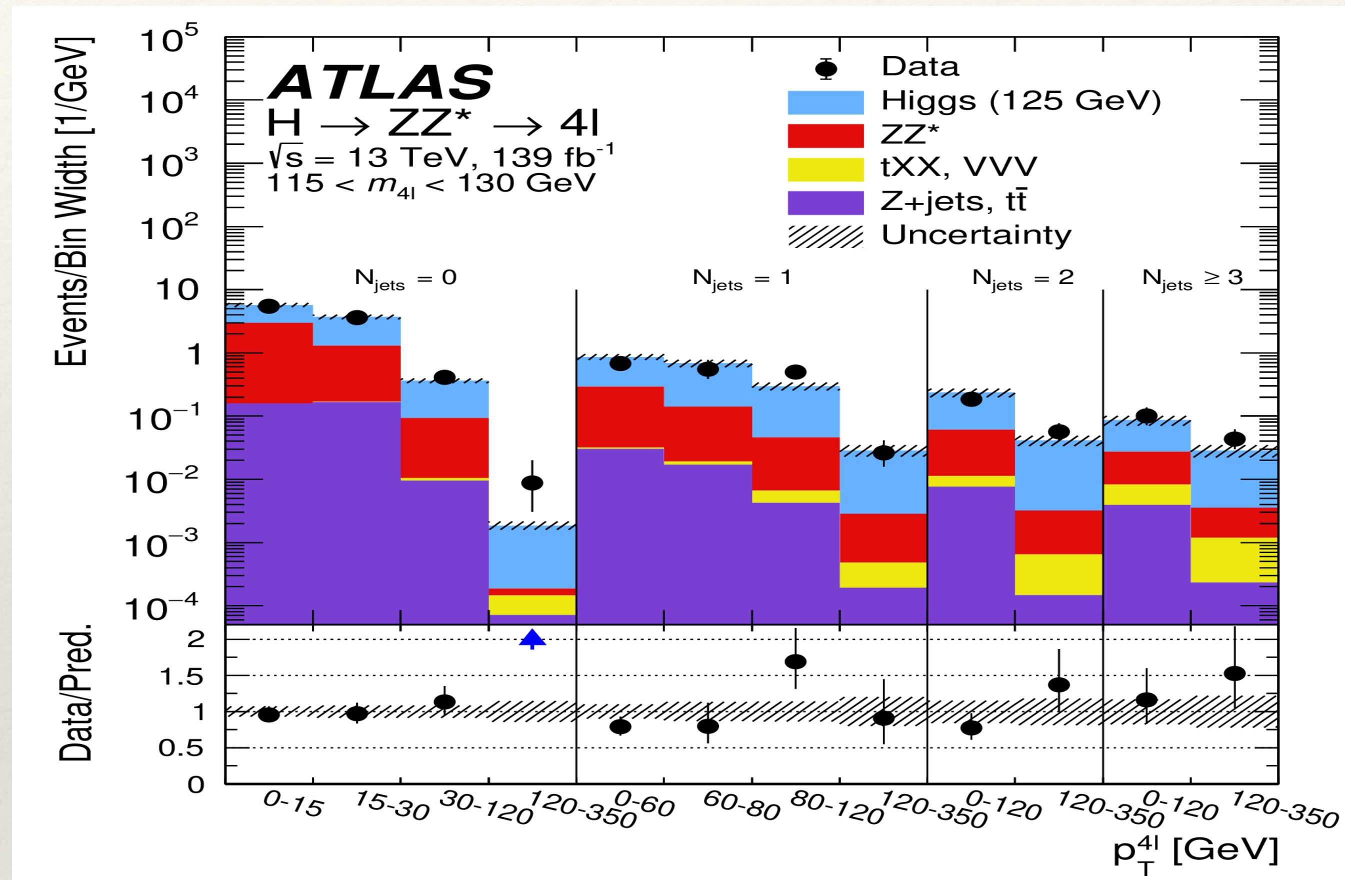
Veronica Sanz (UV/IFIC)

LECTURE 3
NEURAL NETWORKS

The simplest classification problem

dataset $\mathcal{D}(x_i, y_i)$ with $y \in \{0, 1\}$ {no, yes}

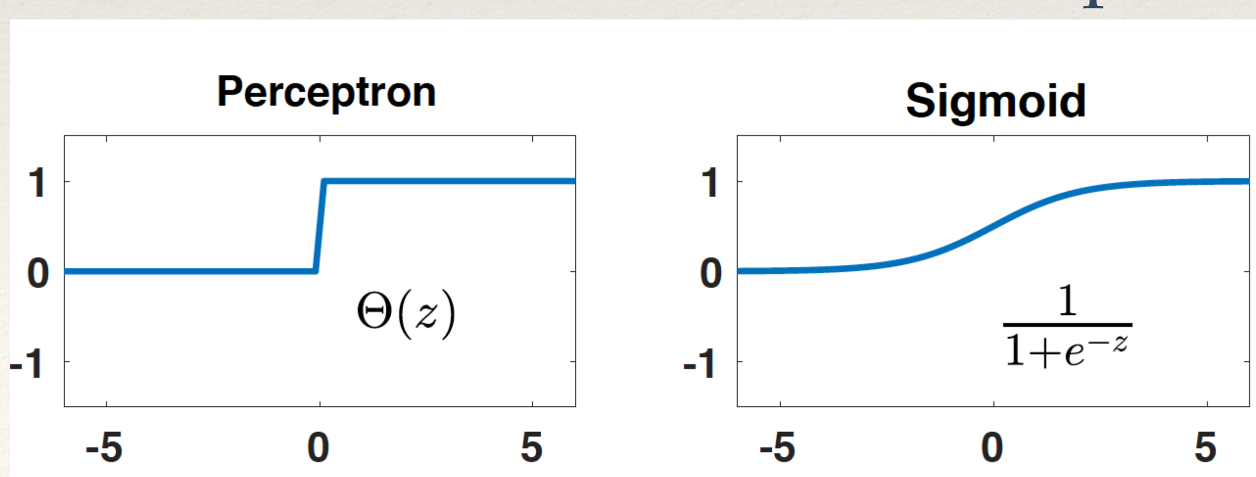




f(w.X) itself can be a function within 0 and 1, for example

X = (pT(4l), Nevents)

w = some numbers



The simplest classification problem

Interpret the output of this transformation as a binomial probability

$$P(y_i = 1) = f(\mathbf{x}_i^T \mathbf{w}) = 1 - P(y_i = 0).$$

e.g. event b-tagged or not, event new physics or not

logistic regression: probability datapoint *x_i* as true or false

We define a cost function for this problem using Maximum Likelihood Estimation (MLE)

$$P(\mathcal{D}|\mathbf{w}) = \prod_{i=1}^{n} \left[f(\mathbf{x}_i^T \mathbf{w})\right]^{y_i} \left[1 - f(\mathbf{x}_i^T \mathbf{w})\right]^{1-y_i}$$

prob dataset $\mathcal{D}$ explained by our *model w*

Binary cost function: Cross-entropy

then log-likelihood is

$$l(\mathbf{w}) = \sum_{i=1}^{n} y_i \log f(\mathbf{x}_i^T \mathbf{w}) + (1 - y_i) \log \left[1 - f(\mathbf{x}_i^T \mathbf{w})\right]$$

best description: parameters *w maximize* the log-likelihood

$$\hat{\mathbf{w}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{n} y_i \log f(\mathbf{x}_i^T \mathbf{w}) + (1 - y_i) \log \left[1 - f(\mathbf{x}_i^T \mathbf{w})\right]$$

Cost function is then chosen to be CROSS-ENTROPY

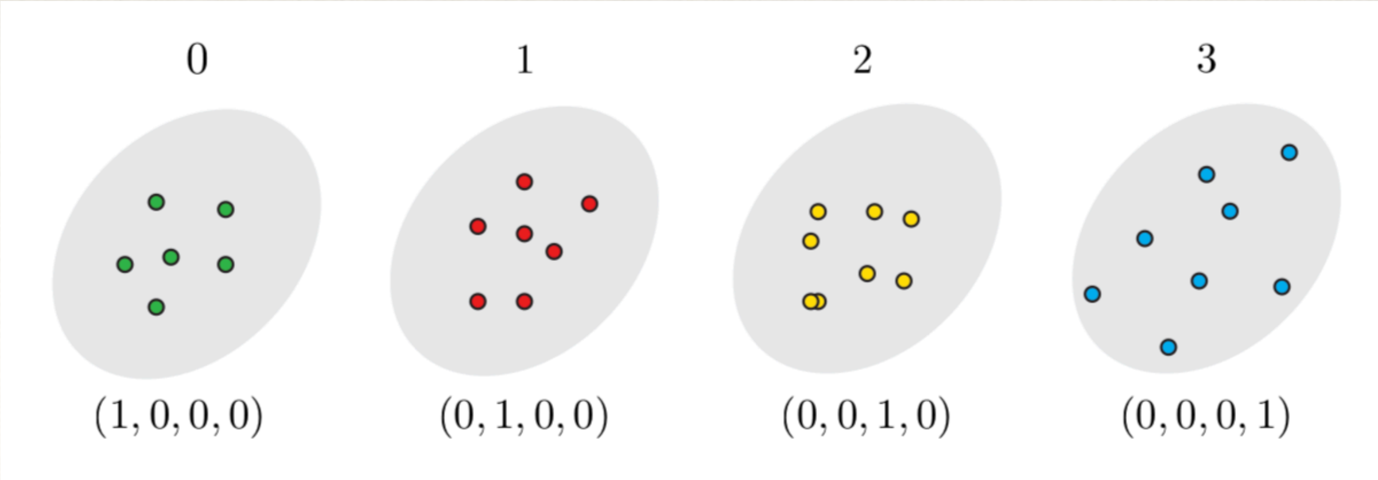
$$\mathcal{C}(\mathbf{w}) = -l(\mathbf{w})$$ +regularization

Logistic regression

As we did yesterday for *linear* regression,
we can do *logistic* regression

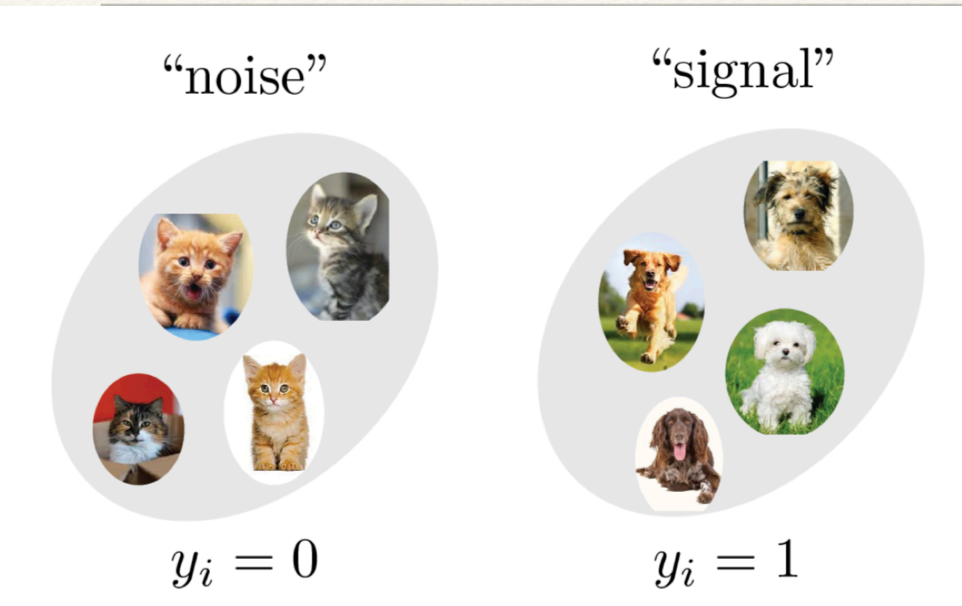
Take a dataset (**X**,y) where y is binary
build a *cost function = cross-entropy*
*m*inimise it and find the parameters **w**
We can use L1 or L2 regularisation

Binary is an example of *categorical* outputs
When we have more than 2 categories, we use a cost function called
SOFTMAX REGRESSION (a generalization of cross-entropy)



often convenient to
describe the categories with
ONE-HOT vectors

Logistic regression: measures of performance



How do we measure performance in a classification task?

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

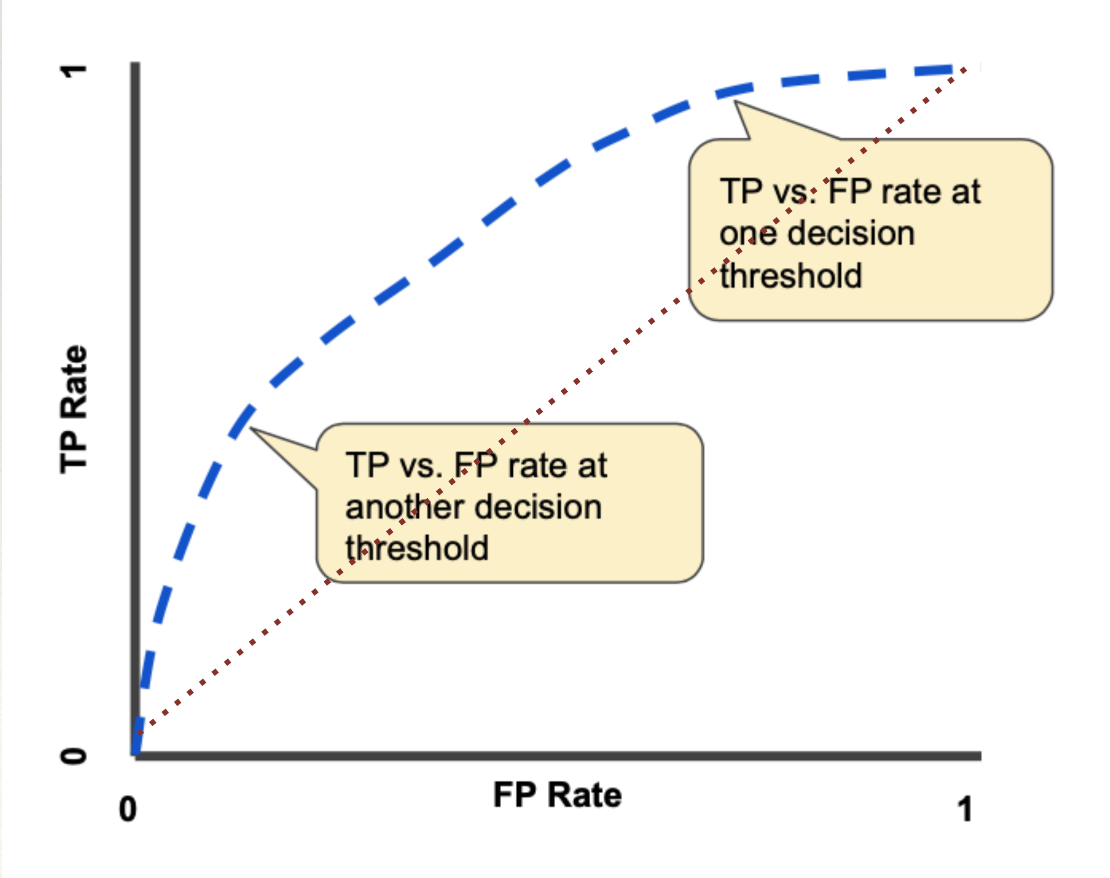
‘We now can distinguish cats and dogs with 95% accuracy’
but maybe identifying cats is easier than dogs
you nearly always get cats right (98%),
but sometimes confuse a little dog for a cat (3%)

Moreover, in your learning you may want to specialise in id’ing dogs because want to make sure you don’t miss any. You raise your threshold for them, paying a price cat performance

Logistic regression: measures of performance

So, let's say your focus is on dogs (you were bitten or had a bad experience)

You would care about
True positives (**TP**) : how many dogs you id
False negatives (**FN**): how many you miss
False positives (**FP**): how many cats you confuse with dogs
True negatives (**TN**): cats you id



$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

AUC = indicates goodness of learning

$$\text{Precision} = \frac{TP}{TP + FP}$$

how often when i say 'dog' is dog?

$$\text{Recall} = \frac{TP}{TP + FN}$$

proportion of dogs I id correctly

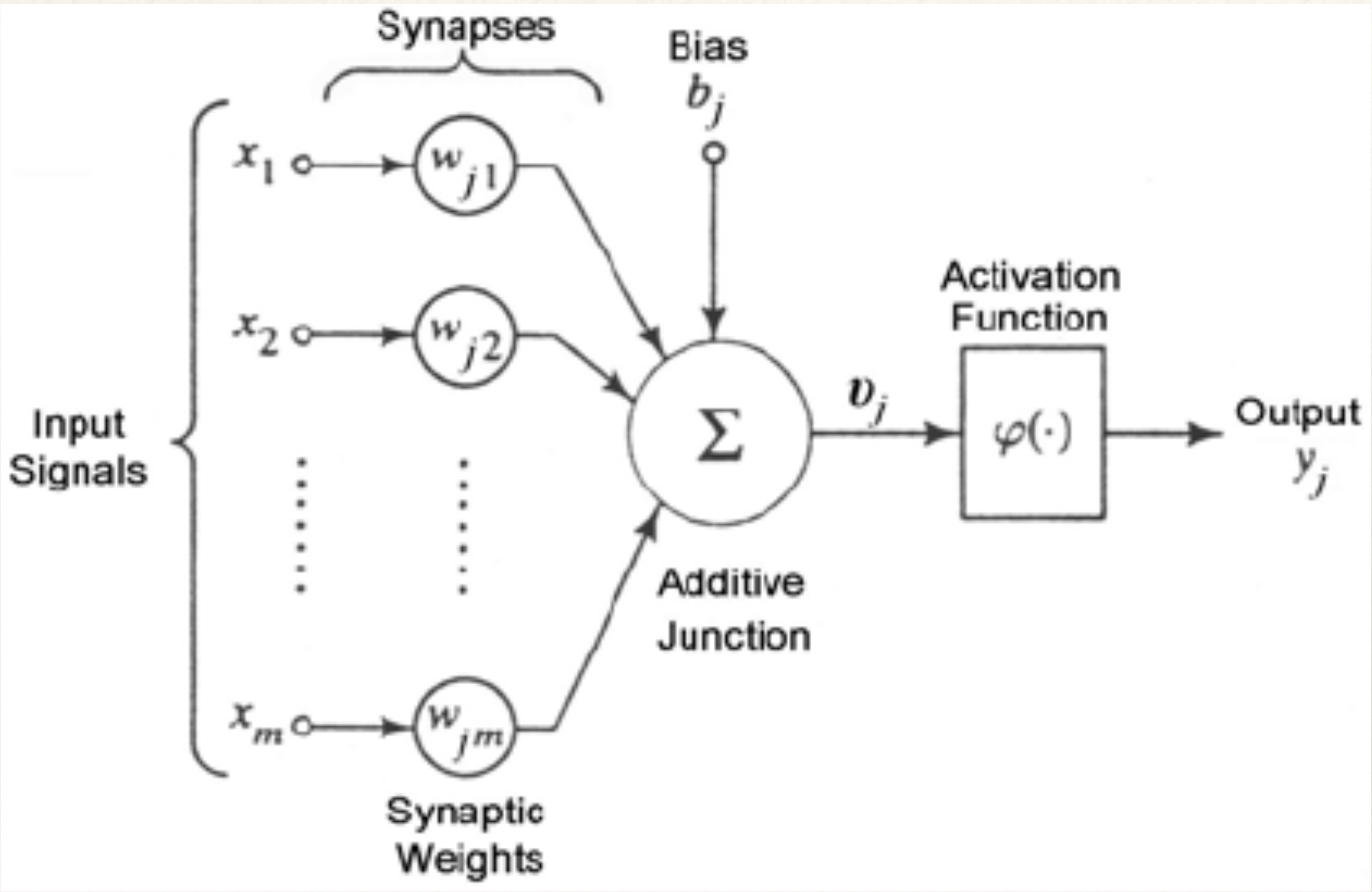
Neural Networks

Learning inspired by biology

Neural Networks (NNs)

A framework to develop AI, based on an architecture of *neurons*

ONE NEURON = BUILDING BLOCKS OF NNs



First, a linear transformation

$$z = w.x + b$$

Second, a non-linear function

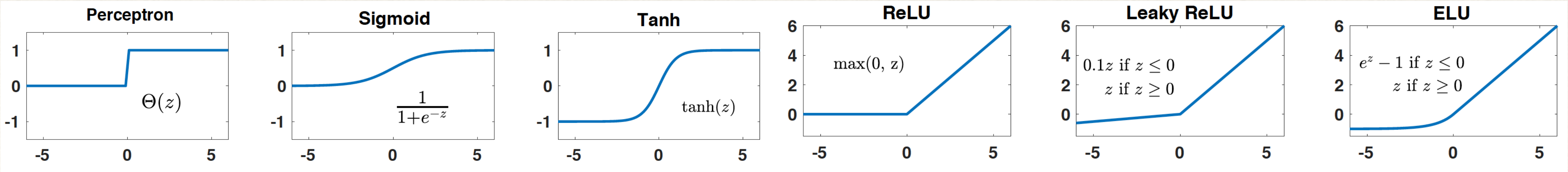
$$y = f(z)$$

y: **output, scalar**

(passes information, or not)

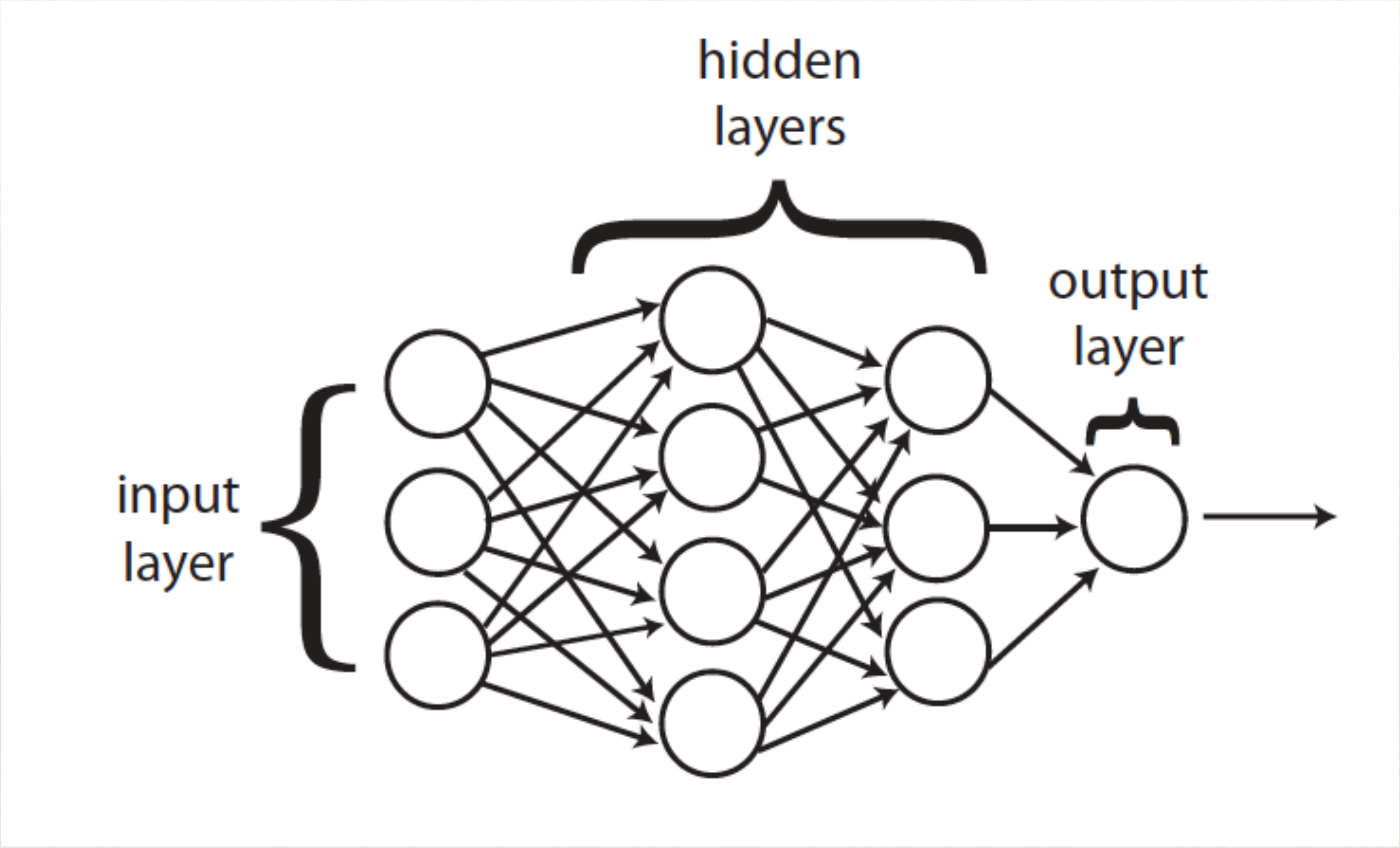
f: **activation function**

Examples of activation functions



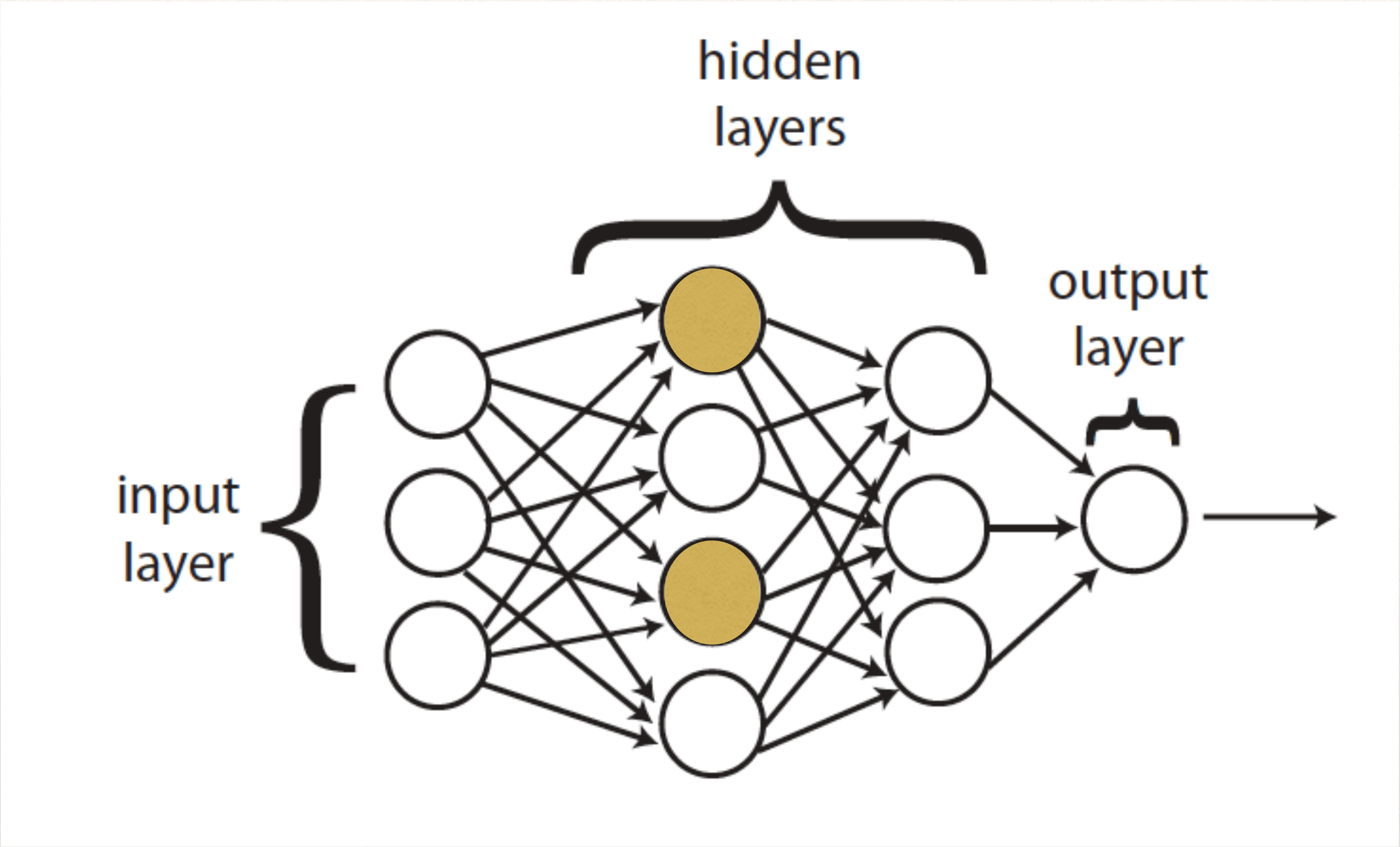
NN Architecture

Taking many neurons together, we can build an *architecture*



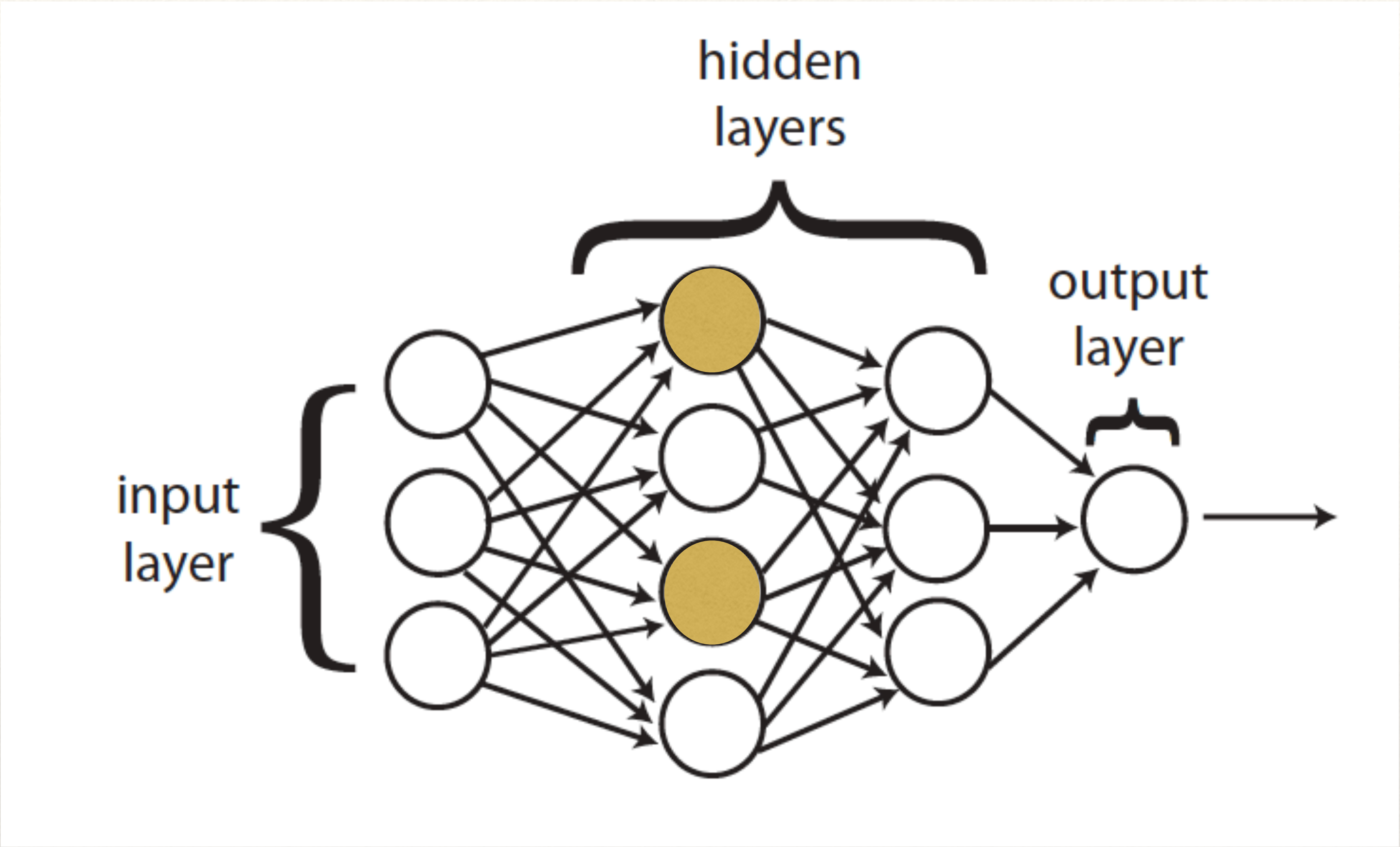
each circle is a neuron,
where the inputs (in-arrows) are transformed into output (out-arrows)
the outputs of each layer serve as input for the next

Why are we doing this?



This NN transforms
inputs (at the input layer) into an output (output layer)
by passing via the hidden layers
non-linear transformations of many non-linear transformations=
highly non-linear transformation of input into output

Why are we doing this?

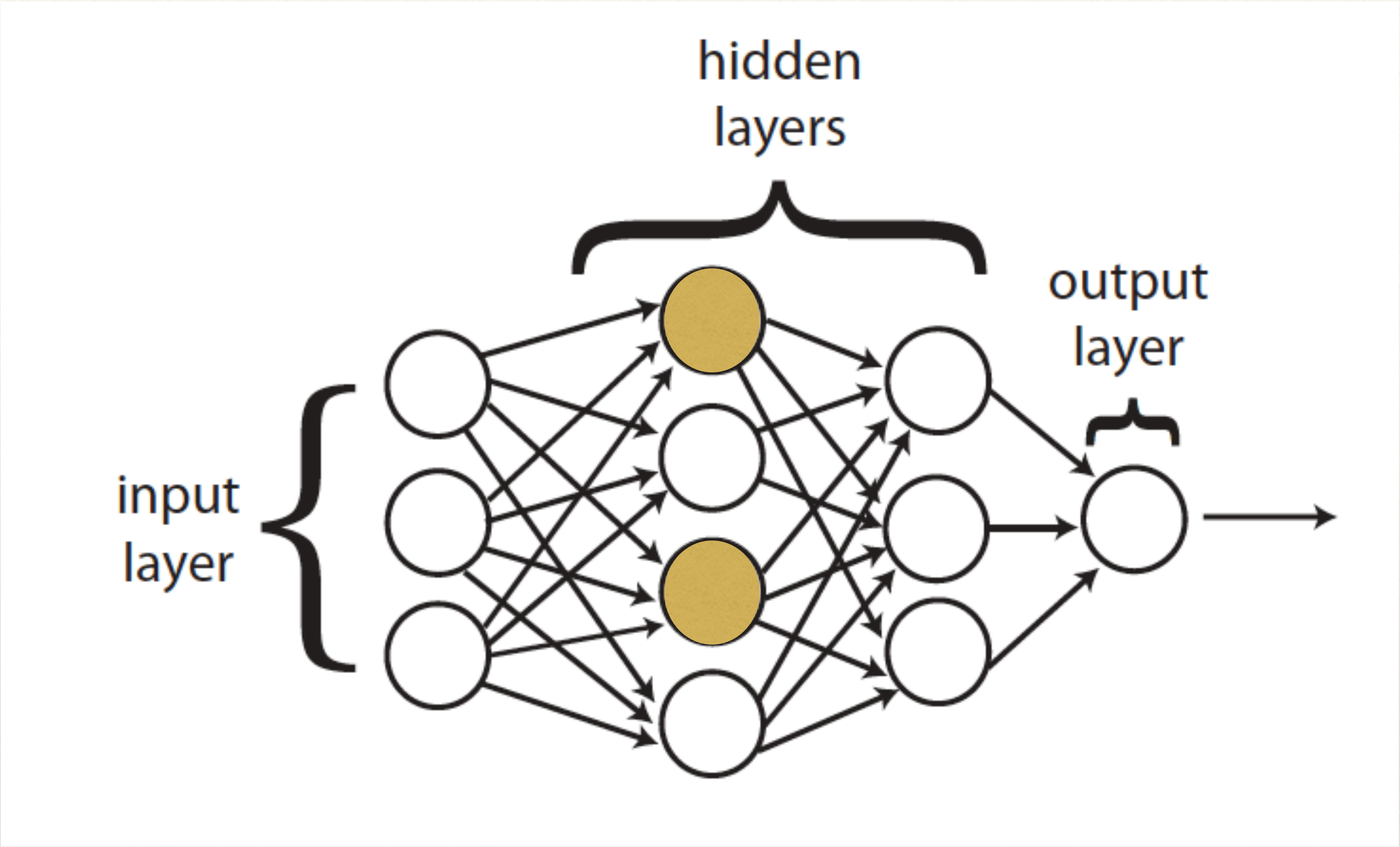


This NN transforms
inputs (at the input layer) into an output (output layer)

$$y(x)$$

which couldn't be captured by simple functional forms

Why are we doing this?

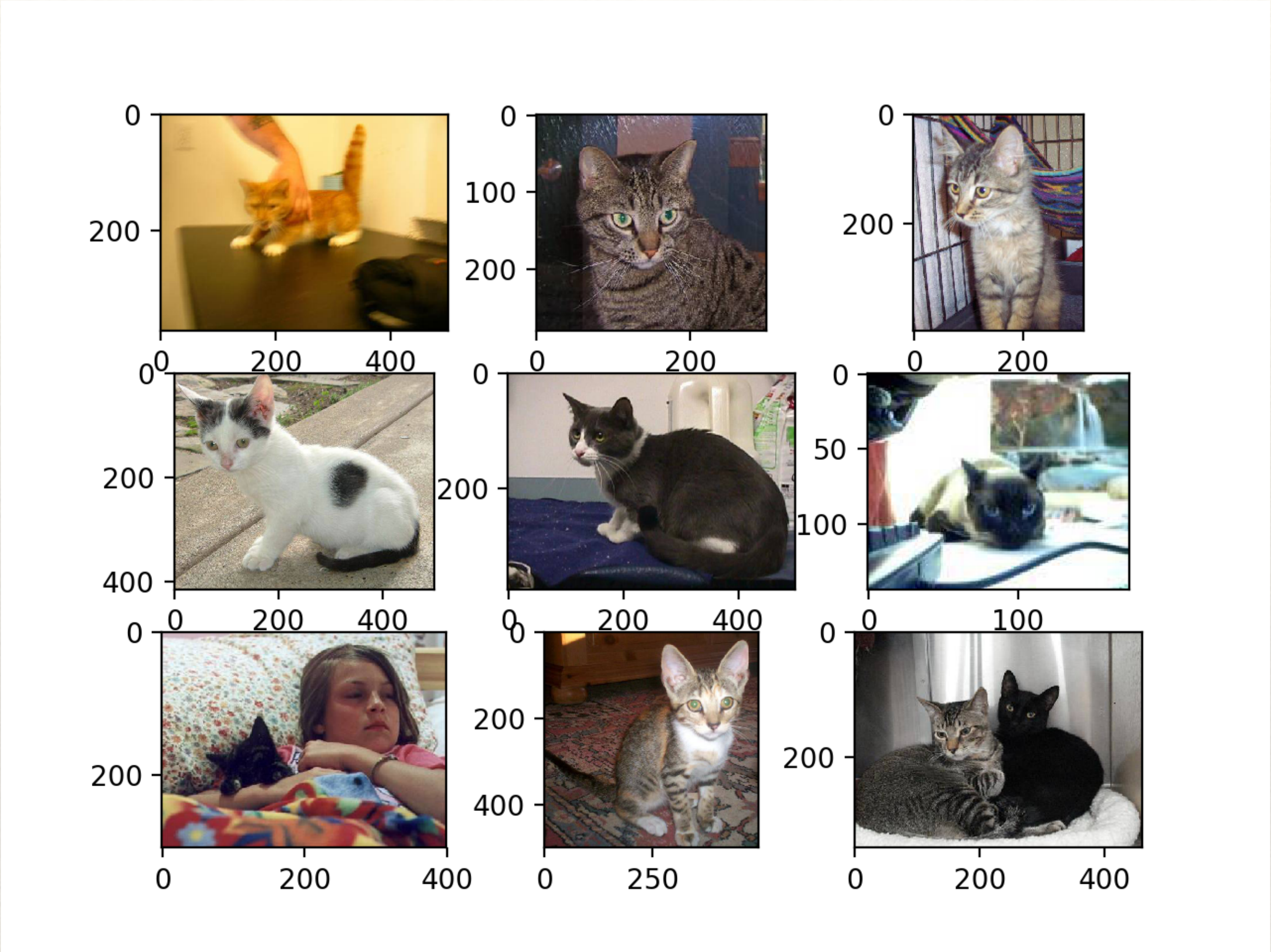


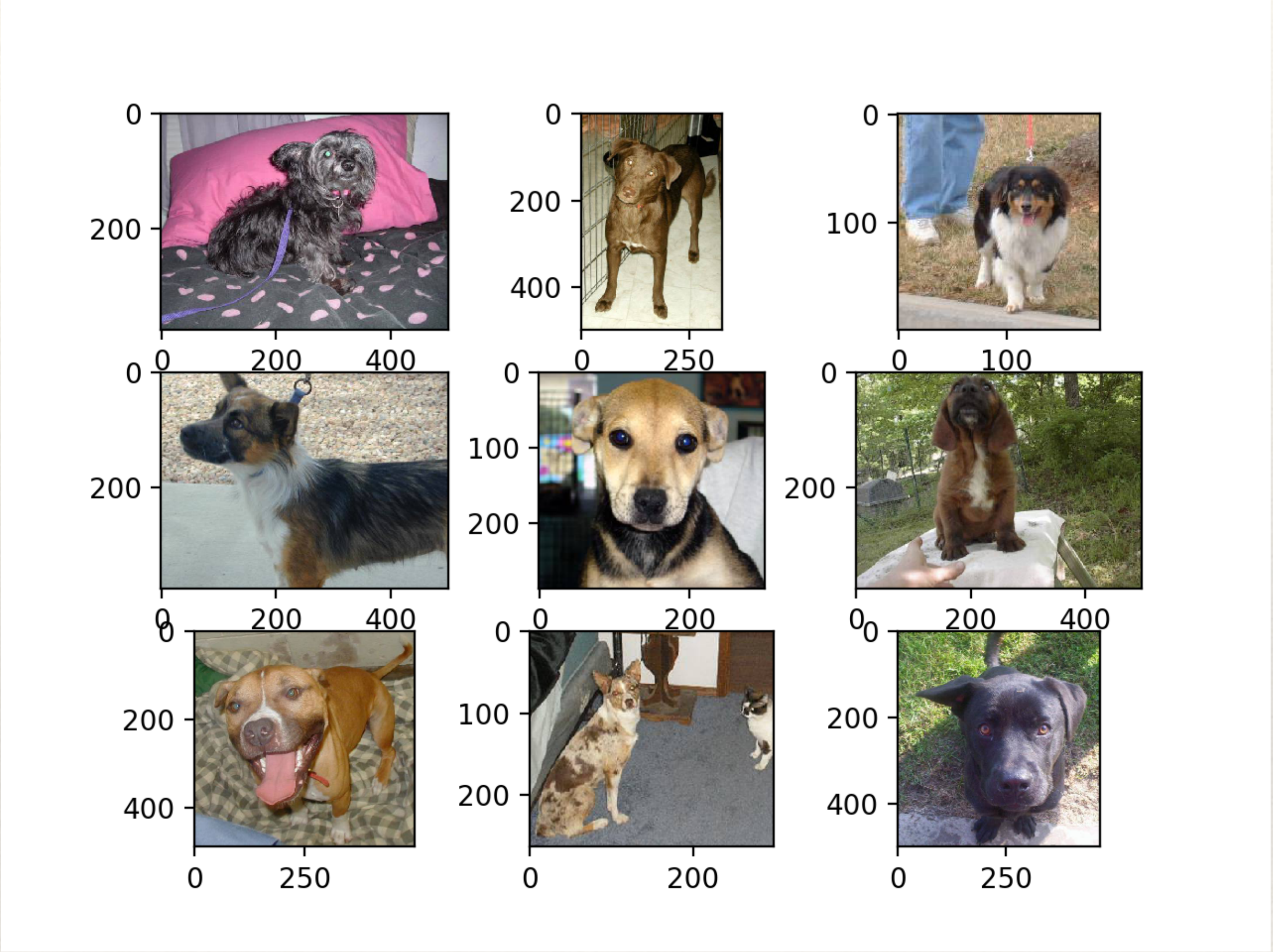
Neural Networks can model **complexity**
They have a high degree of expressivity
/exhibit high representational power
More hidden layers=> more complex features
Deep learning, deep NN

Complex features

images, speech : are complex

For example: cats / dogs



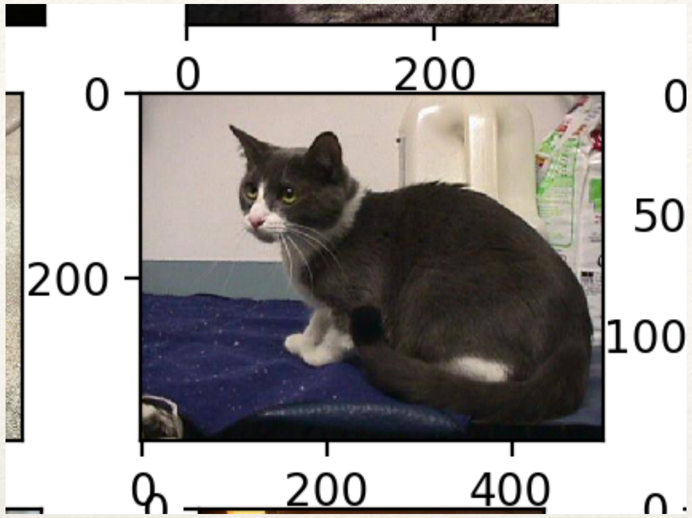


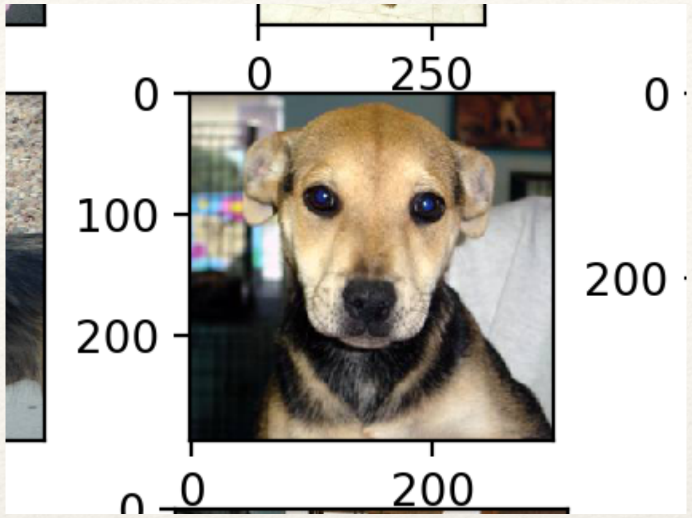
you can distinguish these cats and dogs, right? but how?
would you be able to write a code which classifies them with ~ 100% accuracy? well, a NN can learn to do this!

Convolutional Neural Networks

CNNs

Complex features are often *local*



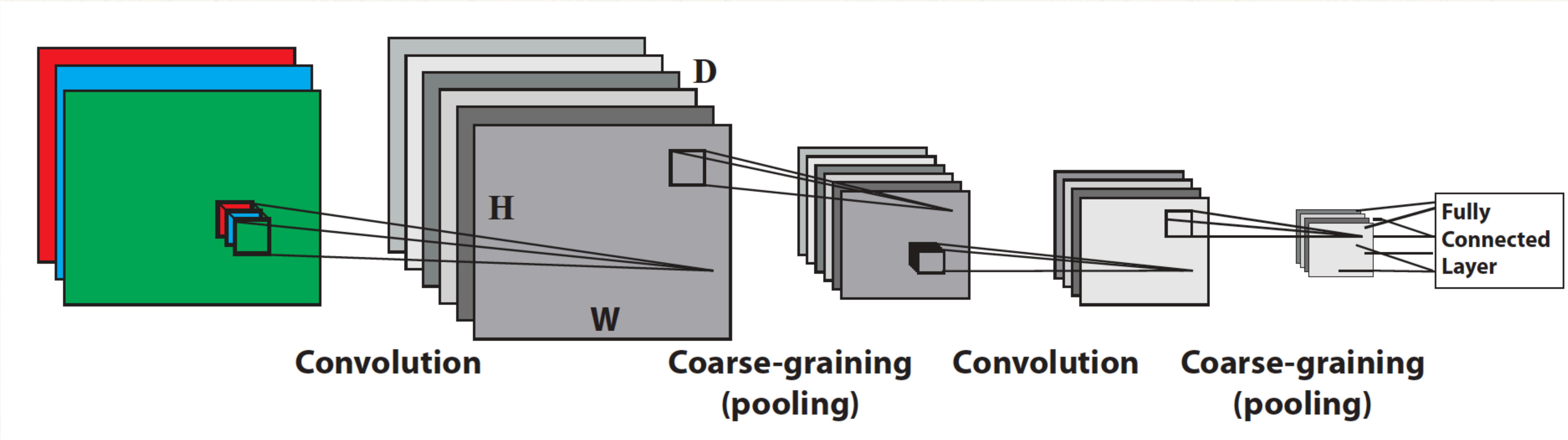


Apart from shape and color,
we know a cat is a cat because there are relations among their features, e.g. the position of the eyes/ ears respect to the head centre, independently of where in the image the cat is

Locality and **translational invariance** must end up playing a role in the identification task

Convolutional Neural Network (CNN)
a type of NN architecture designed to exploit these two characteristics

CNNs

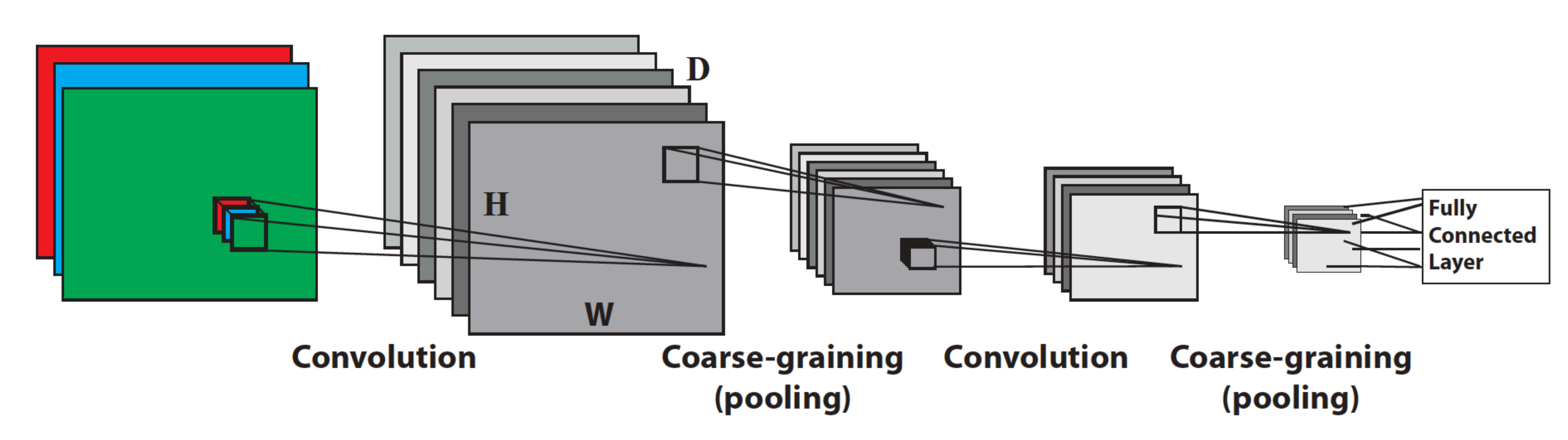


Two types of basic layers

Convolution layer: Height, Width and Depth (e.g. RGB channels) *Convolution*= operation to reduce information while maintaining spatial relations (locality and translation properties)

Pooling: Take areas of the image and reduce them. Example, max-pooling would take 2X2 neurons and replace by a single neuron with input the max of the 4

CNNs



Why do we do this?

Too much superfluous information in an image

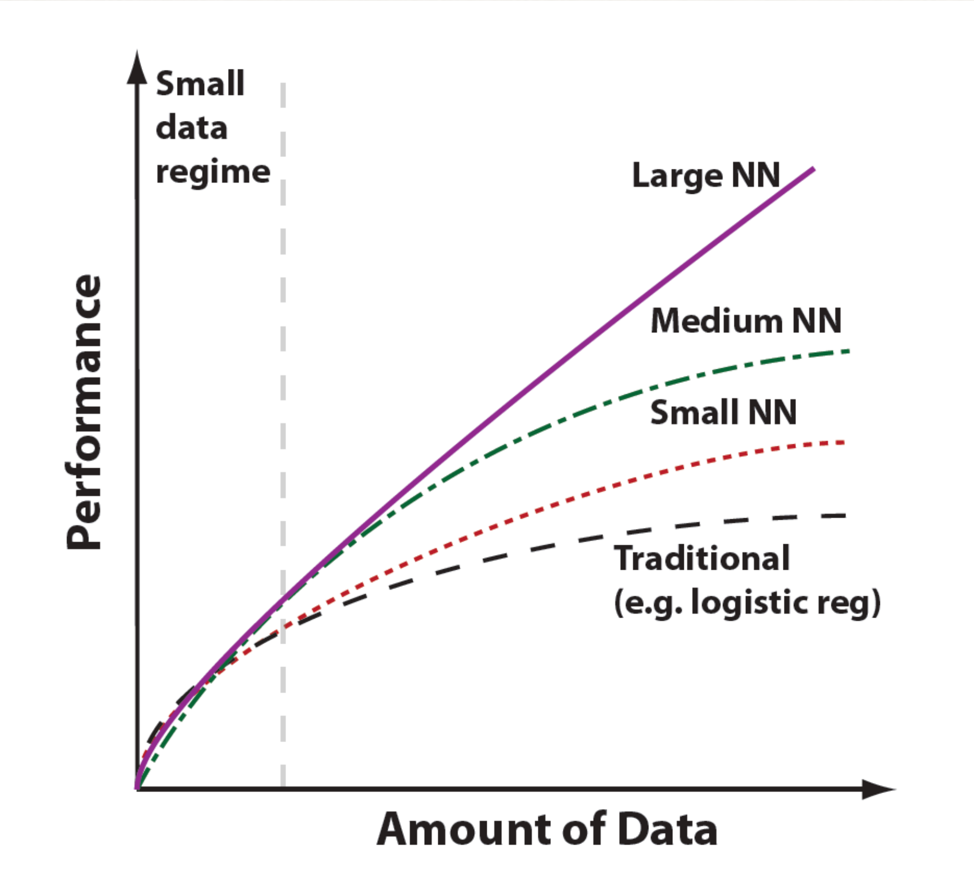
Need to transform the image and capture the essentials
while maintaining spatial relations

As we advance in the layers, the CNN is transforming the original image into something more and more abstract

In physics, translationally invariant systems can be parametrised by wave number and functional form (sin, cos)
whereas an arbitrary system would be *much more complex*

Why are NNs so good at learning?

Good at learning: ability to learn with little *domain knowledge*
That's something physicists (as humans) are good at
(Physics -> other things)
DNNs are good at this too, they are able to take large streams of data and learn features with little guidance, work like *black boxes*



Good at handling large amounts of data: needle in a haystack
The NN structure (layers, 0/1 gates) allows a high representation power with moderate computational demands, e.g. allows parallelisation, use of GPUs…
It scales better than other learning methods (like SVMs)

Today: NNs and CNNs

We will take a standard dataset, MNIST

Build and train a NN
to become better at recognising hand-written numbers
This is a *supervised* ML problem
(we know the true labels)
we train on a large sample (60K) images

We will build a fully connected NN,
a Convolutional Neural Network,
and use Data Augmentation
Our precision will go from 96% till 99%
Link to Google Colab notebook on Neural Networks

TOMORROW: *Unsupervised*