



UNIVERSITY  
OF AMSTERDAM

# Machine Learning for Physics and Astronomy

Juan Rojo

VU Amsterdam & Theory group, Nikhef

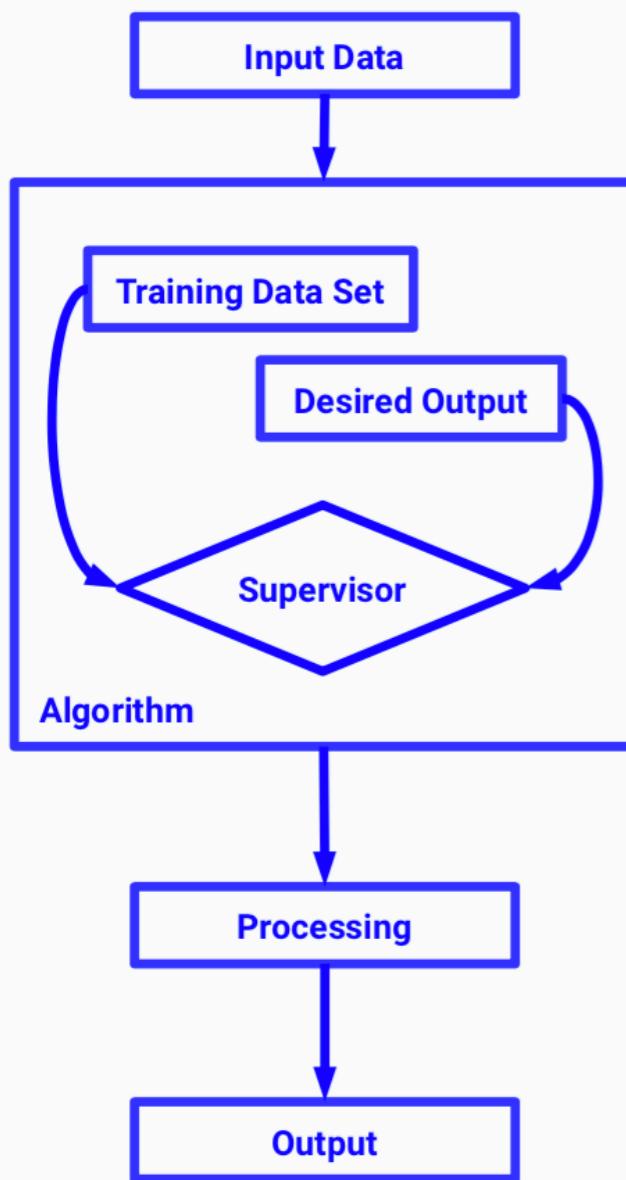
***Natuur- en Sterrenkunde BSc (Joint Degree), Honours Track***  
***Lecture 3, 21/09/2021***

# The path so far

- 💡 Why machine learning is difficult: **fitting vs predicting** in models with a large number of parameters
- 💡 **Supervised Learning** requires *i*) input dataset, *ii*) ML model, and *iii*) definition of a cost function / figure of merit
- 💡 Training of ML models requires **numerical methods** e.g. gradient descent or evolutionary algorithms
- 💡 Each has pros and cons, e.g. genetic algorithms do not require evaluation of gradients, but SGD is more efficient close to the global minimum

# Paradigms in Machine Learning

## Supervised learning



## Unsupervised learning



## Reinforcement learning



This lecture

next week

coming soon!

# Today's lecture

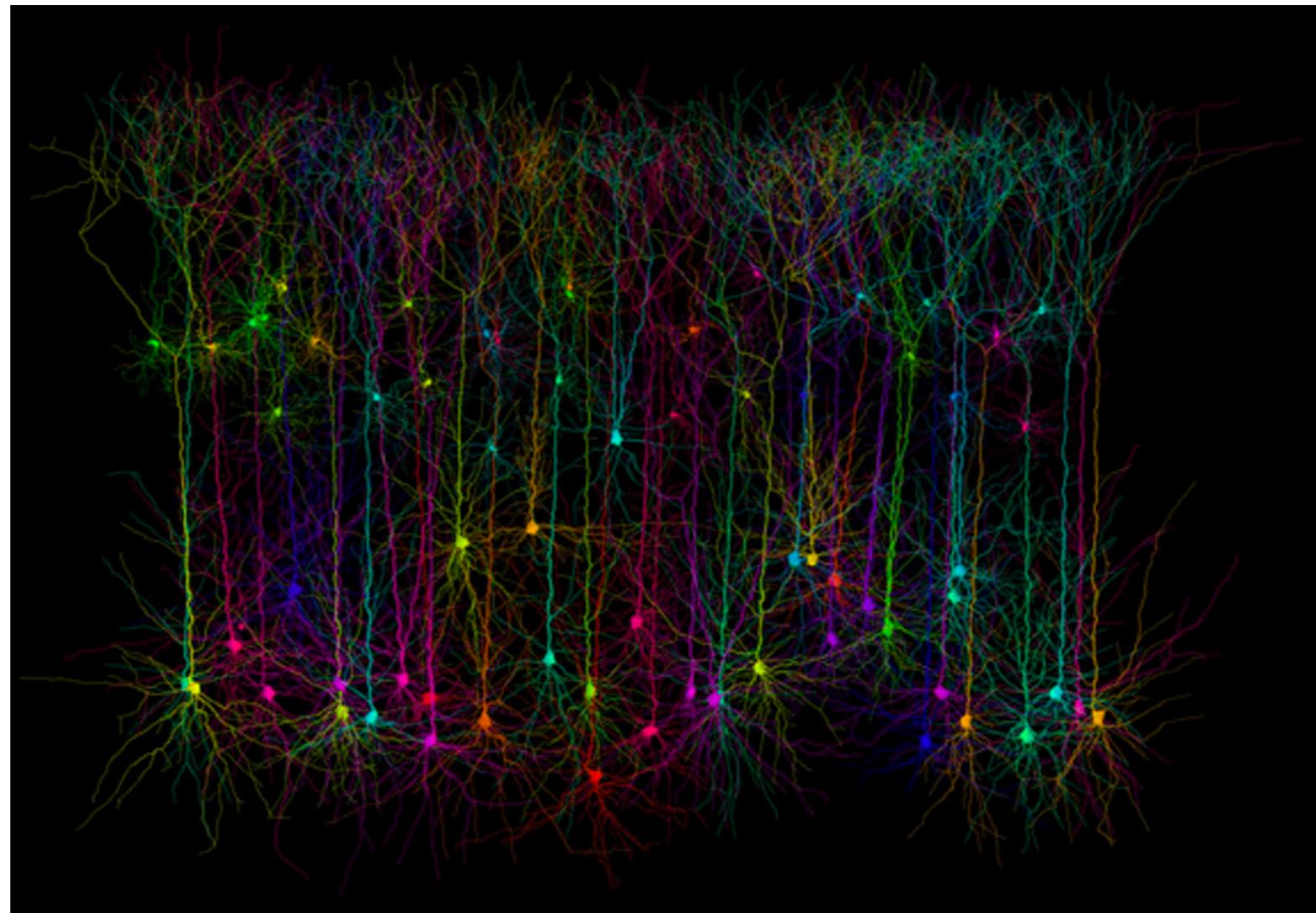
- 📌 Neural Networks and perceptrons
- 📌 Deep Learning
- 📌 Supervised NN learning: **backpropagation**
- 📌 Optimal training and hyperoptimisation

# **(Artificial)**

# **Neural Networks**

# Artificial Neural Networks

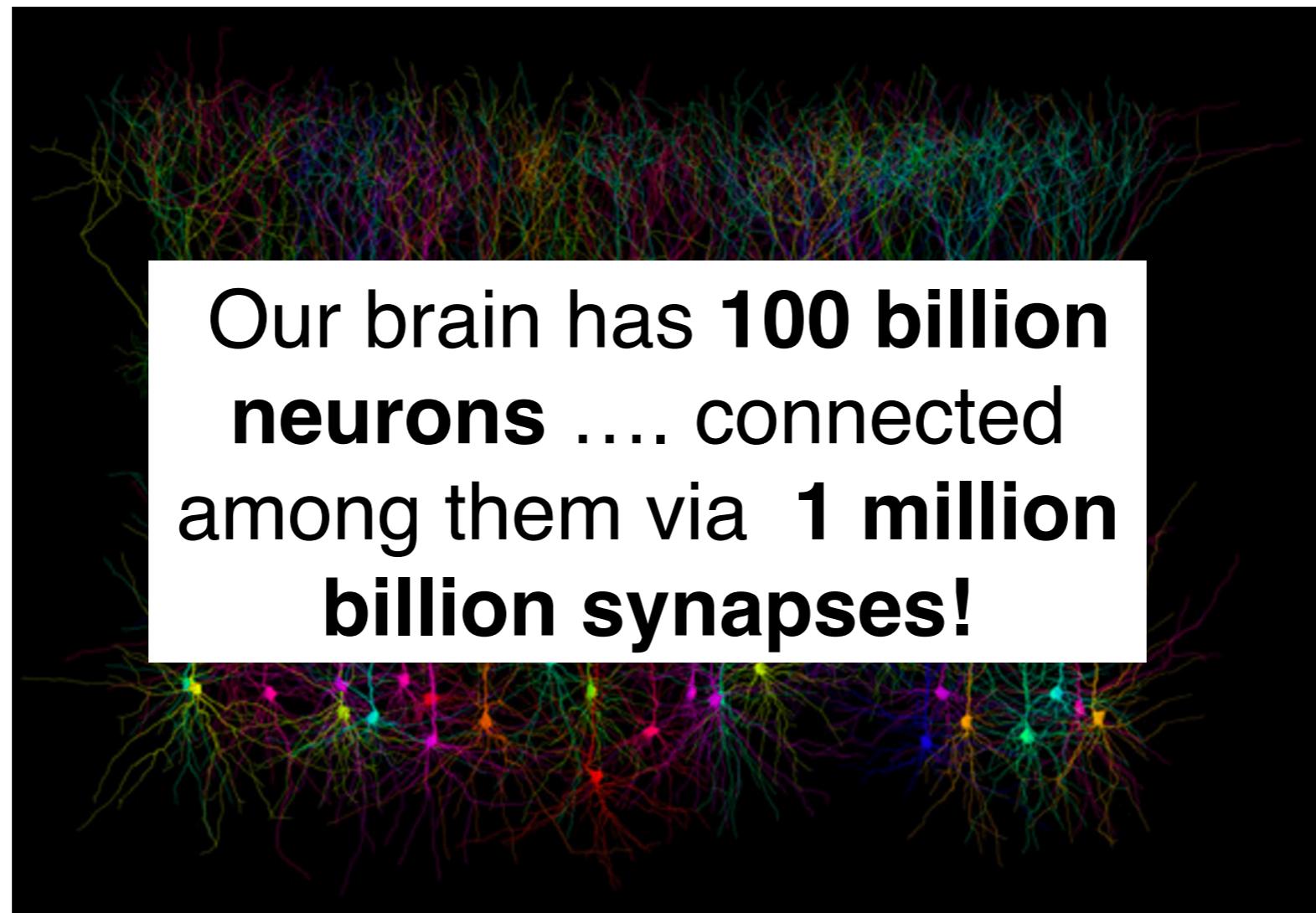
Inspired by **biological brain models**, **Artificial Neural Networks** (ANNs) are mathematical algorithms designed to excel where domains as their evolution-driven counterparts outperforms traditional algorithms in tasks such as **pattern recognition, forecasting, classification, ...**



*what makes our brain so powerful?*

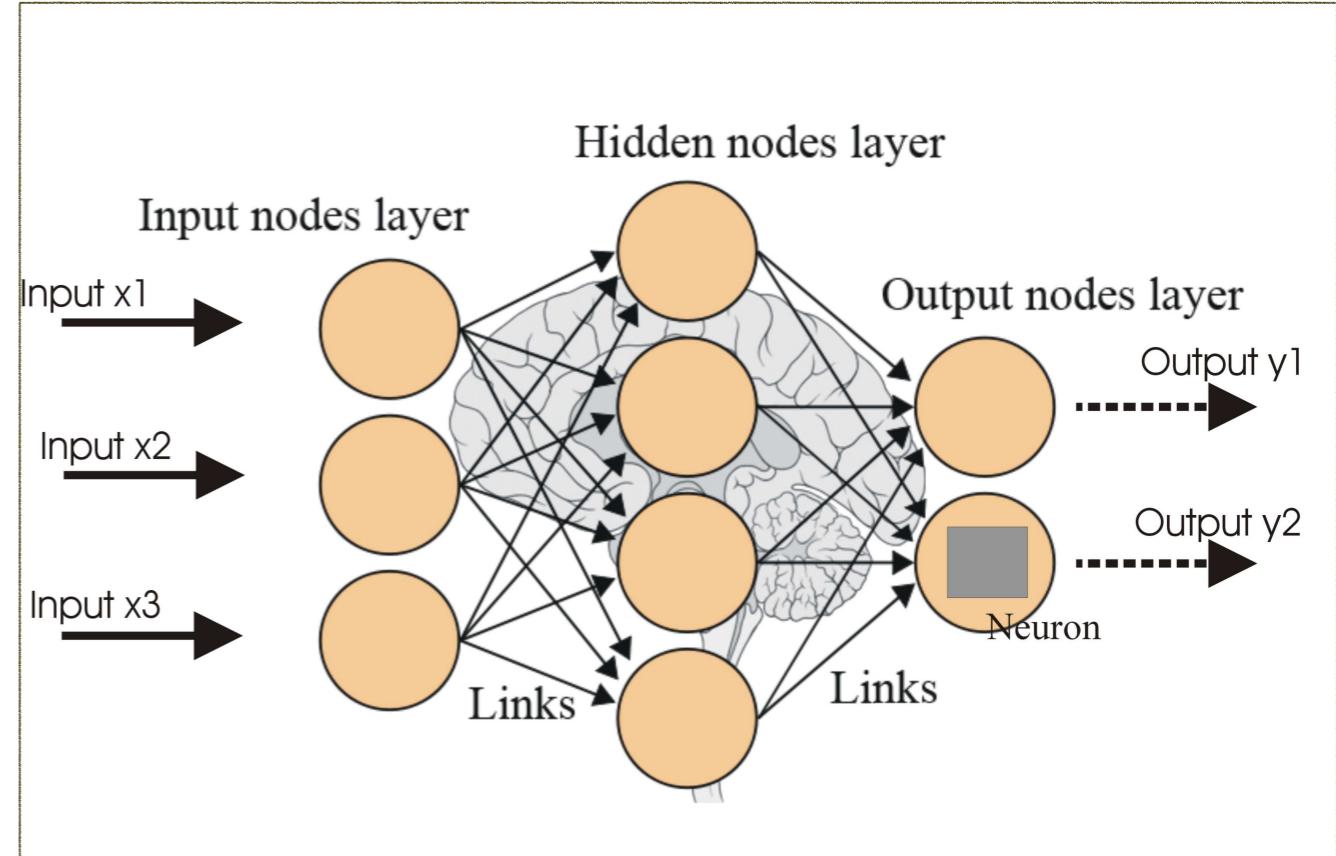
# Artificial Neural Networks

Inspired by **biological brain models**, **Artificial Neural Networks** (ANNs) are mathematical algorithms designed to excel where domains as their evolution-driven counterparts outperforms traditional algorithms in tasks such as **pattern recognition, forecasting, classification, ...**



# Artificial Neural Networks

Inspired by **biological brain models**, **Artificial Neural Networks** (ANNs) are mathematical algorithms designed to excel where domains as their evolution-driven counterparts outperforms traditional algorithms in tasks such as **pattern recognition, forecasting, classification, ...**

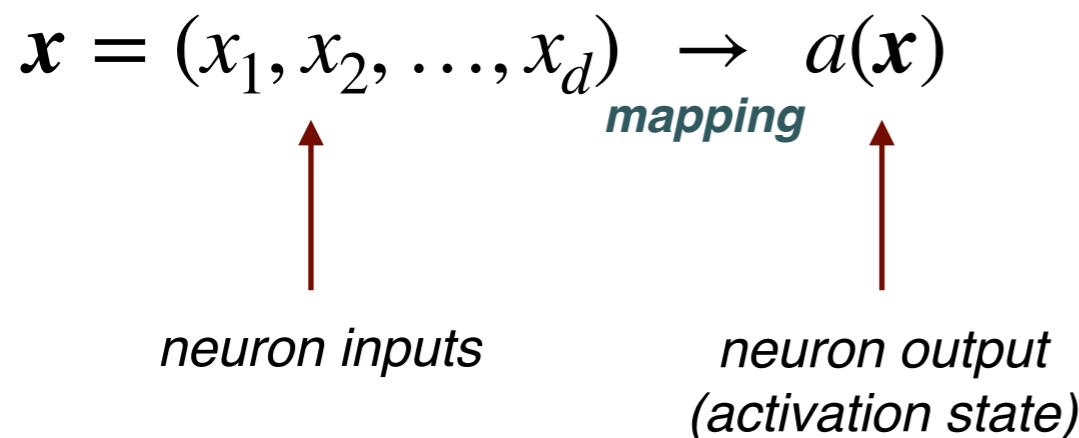


*in ML context, ANN provide a flexible, powerful non-linear model for many problems*

# 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

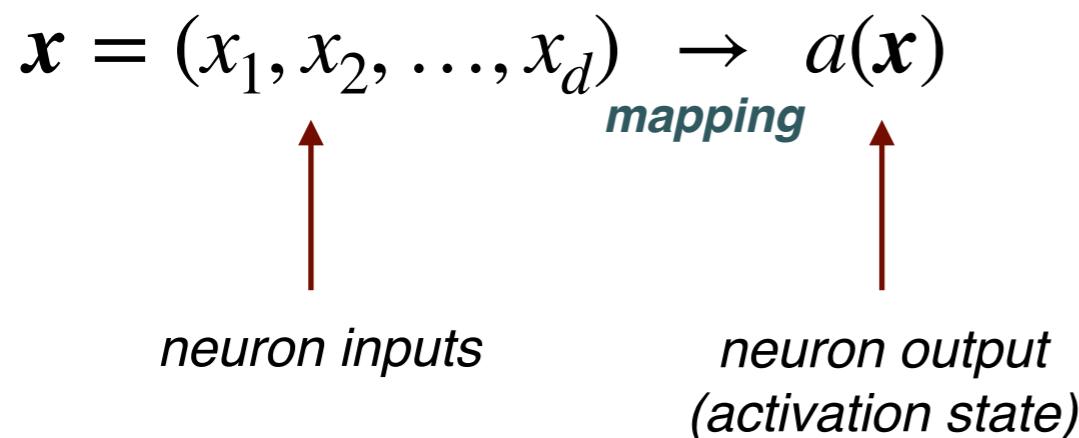


**key feature:** *mapping is (highly) non-linear*

# 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



**key feature: mapping is (highly) non-linear**

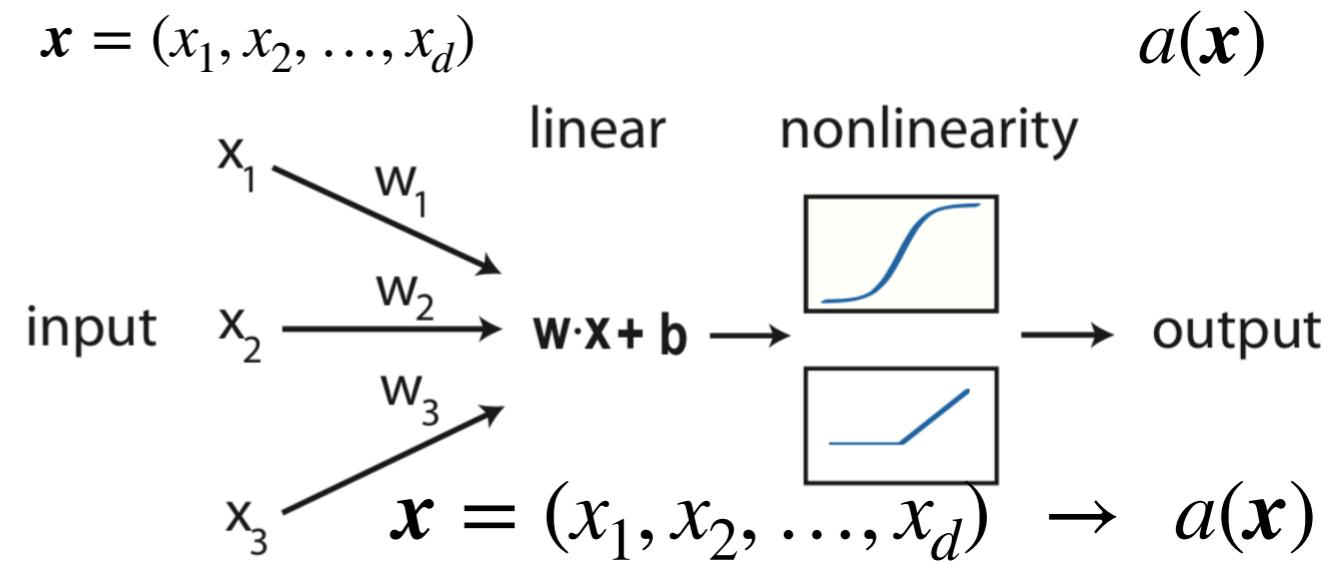
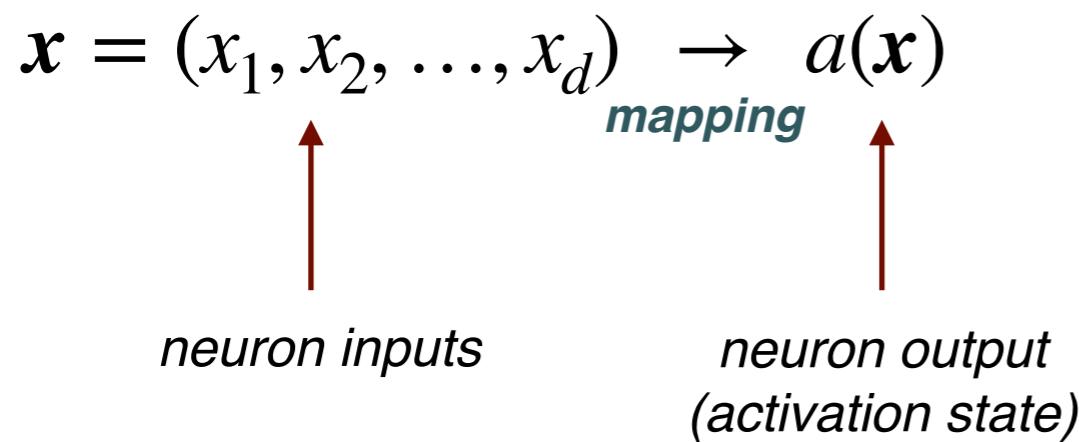
**Q: why a non-linear mapping could be so important for neural networks?**

**(what happens when combining linear transformations?)**

# 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



*biology analog: input are electric pulses, output the activation state of the neuron*



# Neural Networks

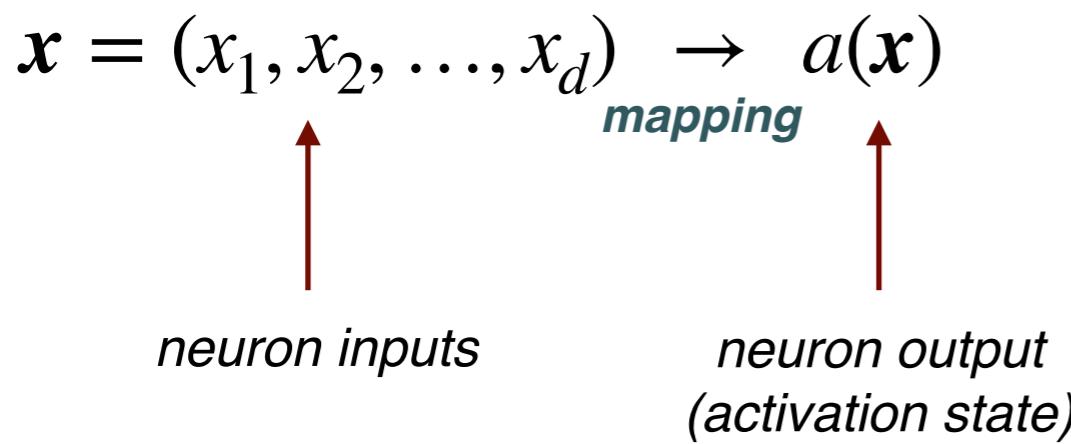
Neural Nets can be defined as **neural-inspired nonlinear models** for supervised learning



# Neural Networks

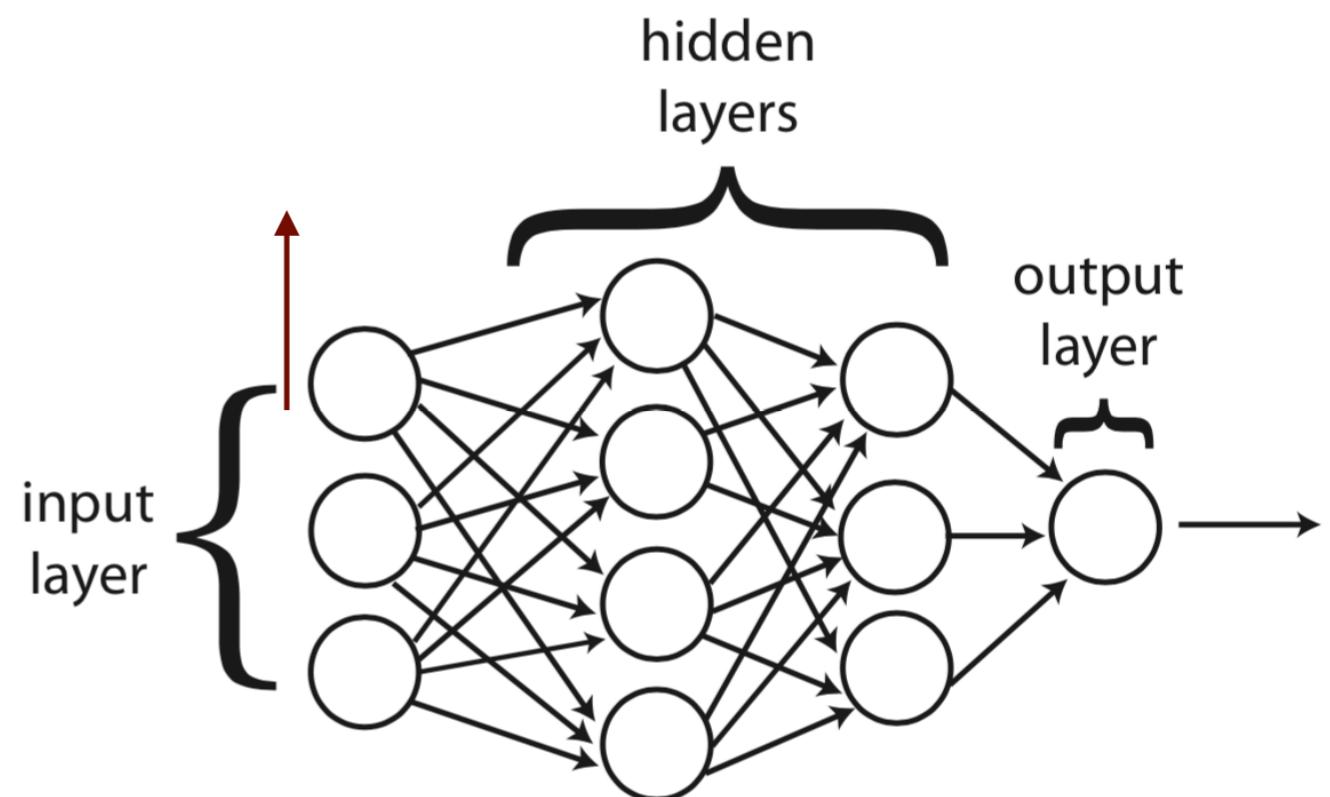
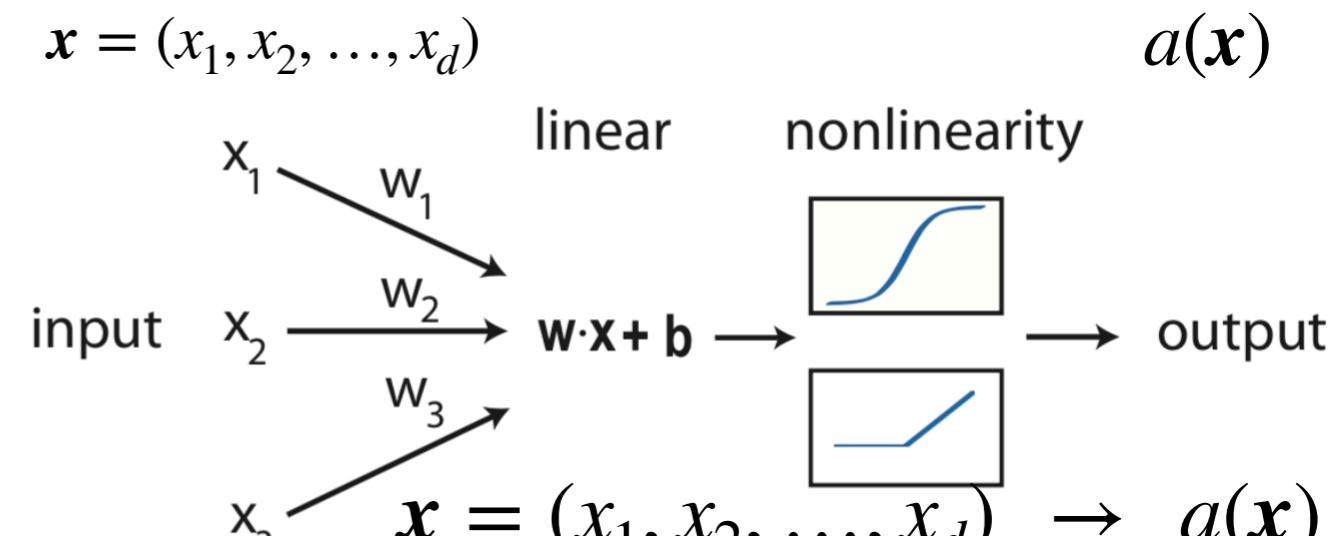
Neural Nets can be defined as **neural-inspired nonlinear models** for supervised learning

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



- These neurons 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



# Neural Networks

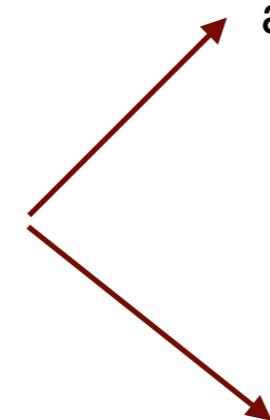
Neural Nets can be defined as **neural-inspired nonlinear models** for supervised learning

the neural network output has  
two components

a **linear operation**, weighting the various inputs

$$z^{(i)} = \mathbf{x}^T \cdot \boldsymbol{\theta}^{(i)} + \boldsymbol{\theta}_0^{(i)}$$

↑      ↑      ↑  
neuron    i-th neuron    i-th neuron  
inputs    parameters    bias



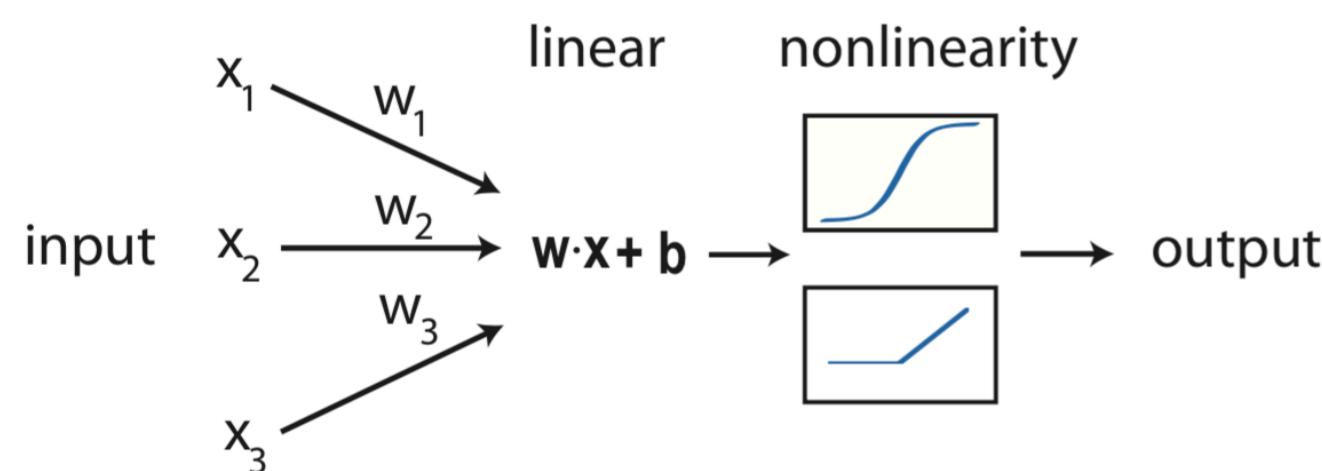
# Neural Networks

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

the neural network output has two components

a linear operation, weighting the various inputs  
$$z^{(i)} = \mathbf{x}^T \cdot \boldsymbol{\theta}^{(i)} + \boldsymbol{\theta}_0^{(i)}$$
  
neuron inputs    i-th neuron parameters    i-th neuron bias

a non-linear transformation  
$$a^{(i)}(\mathbf{x}) = \sigma_i(z^{(i)})$$
  
i-th neuron activation state    i-th neuron activation function (non-linear)



*the parameters of NNs are often called  
“weights” and “thresholds”*

the choice of non-linear activation function affects the **computational and training properties** of the neural nets, since they modify the output gradients required for GD training

*NN nonlinearly only via activation functions!*

# Neural Networks

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

the neural network output has two components

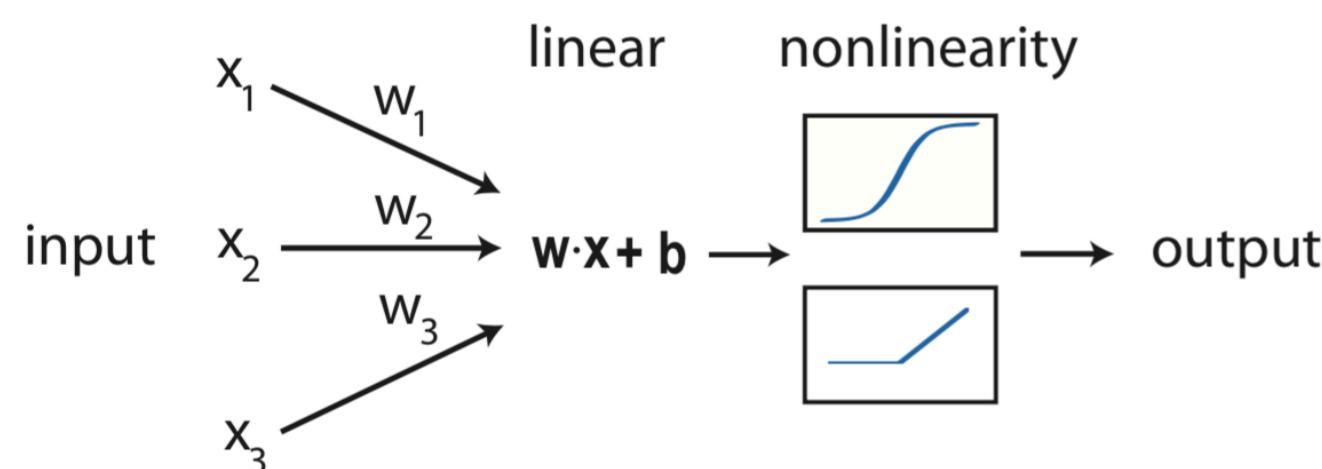
a linear operation, weighting the various inputs  
$$z^{(i)} = \mathbf{x}^T \cdot \boldsymbol{\theta}^{(i)} + \theta_0^{(i)}$$
  
neuron inputs    i-th neuron parameters    i-th neuron bias

a non-linear transformation

$$a^{(i)}(\mathbf{x}) = \sigma_i(z^{(i)})$$

i-th neuron activation state    i-th neuron activation function (non-linear)

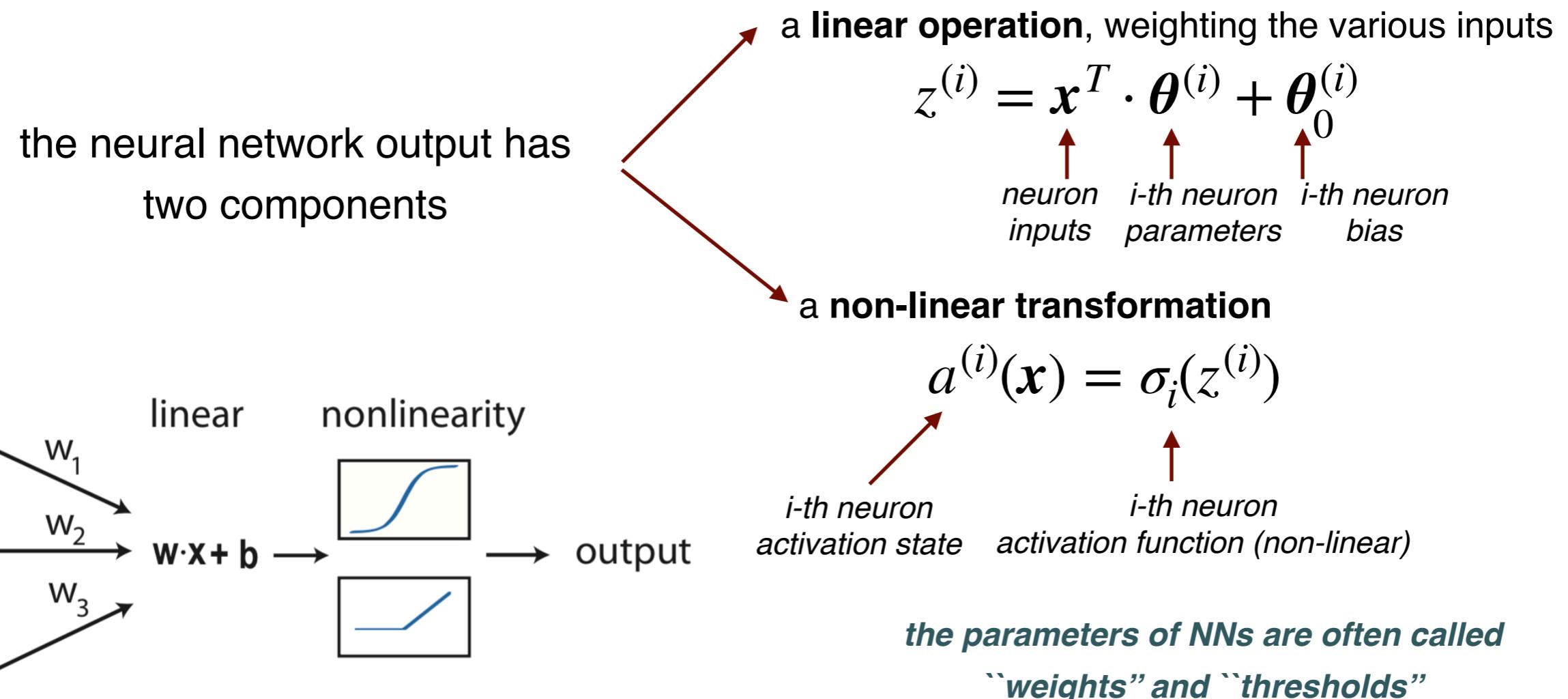
*the parameters of NNs are often called "weights" and "thresholds"*



*What do you think that "neural network training" means at this point?*

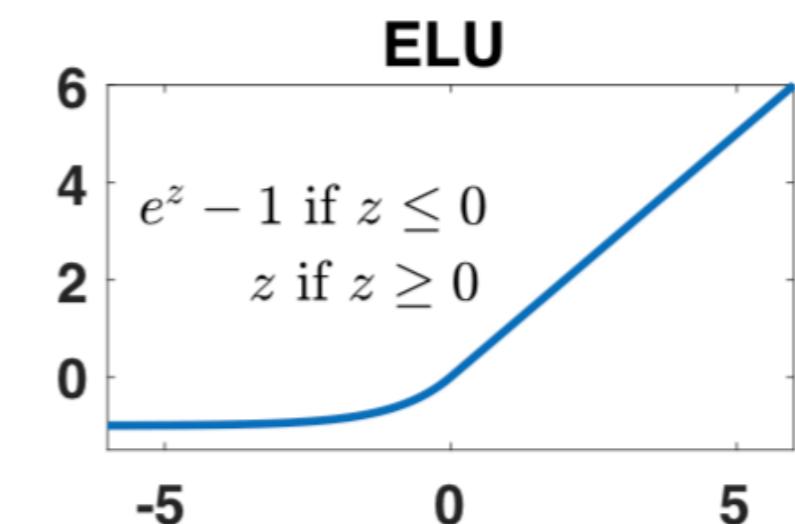
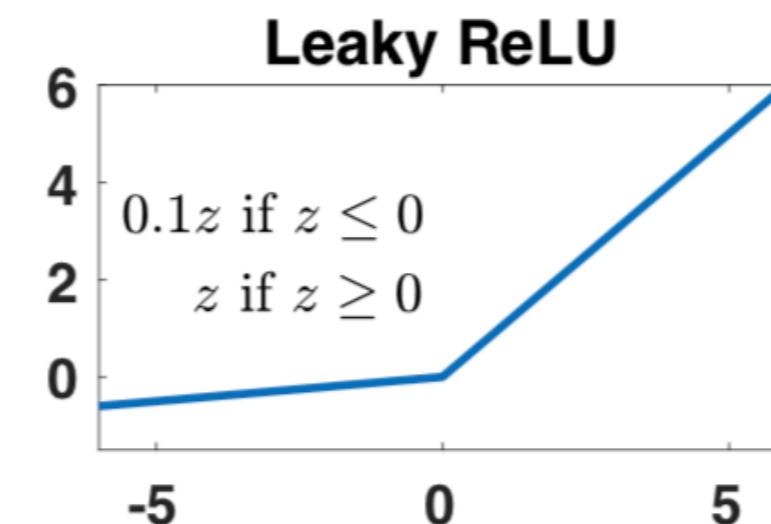
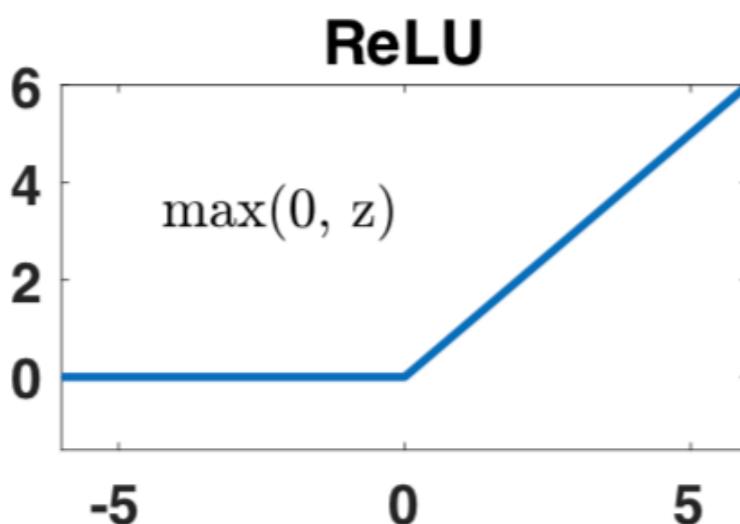
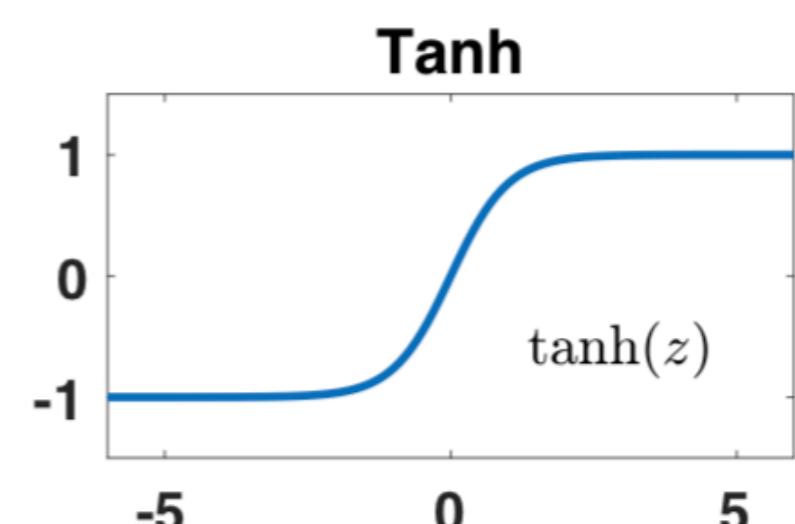
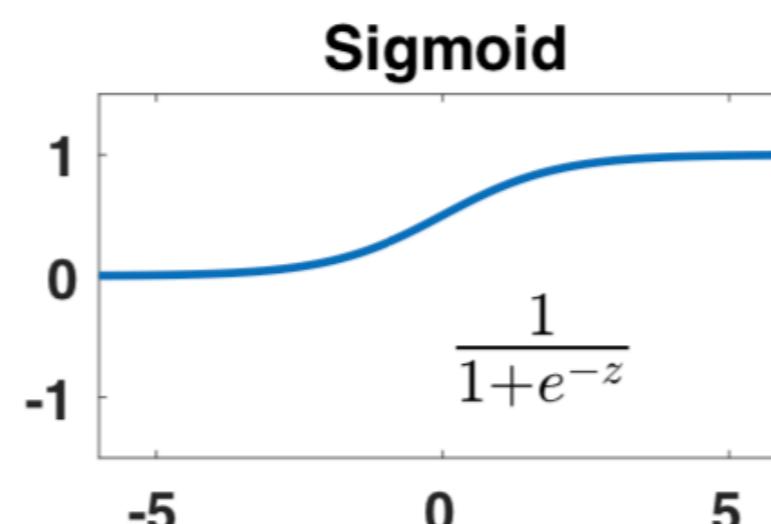
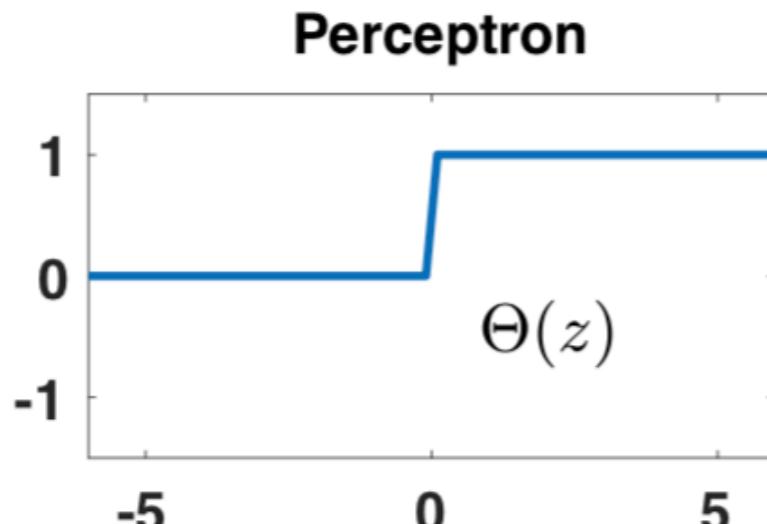
# Neural Networks

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



Neural Network **training** or **learning** means adjusting the model parameters (weights and biases) to **optimise** some figure of merit (cost function)

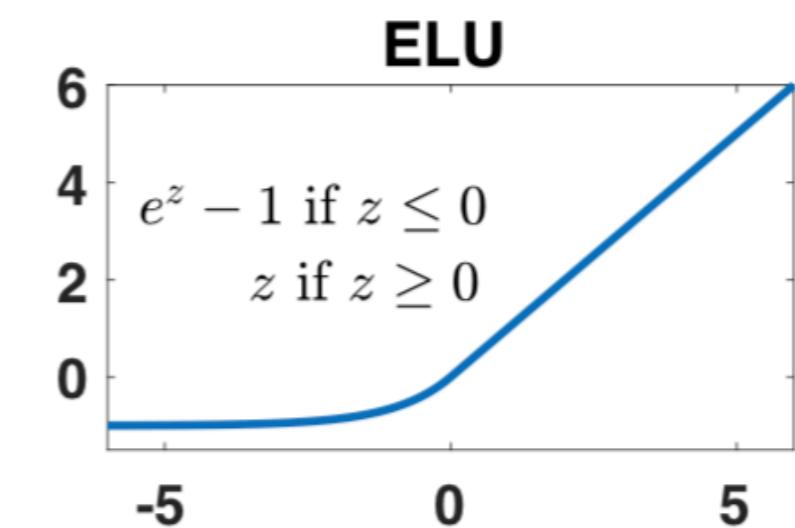
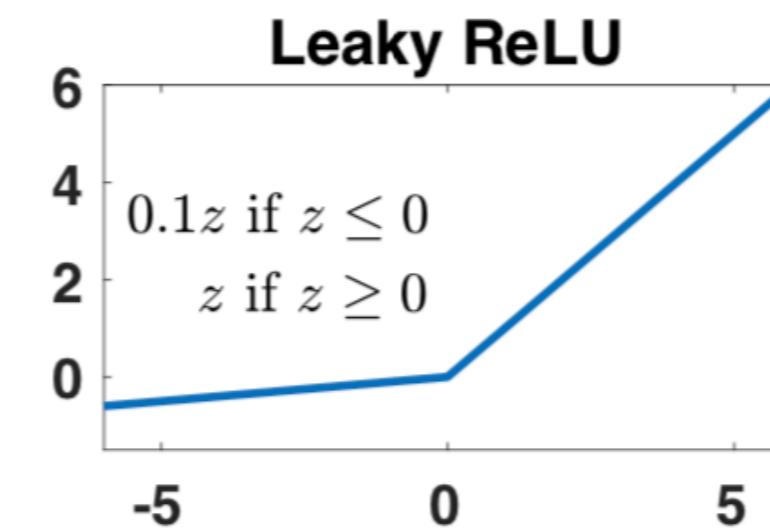
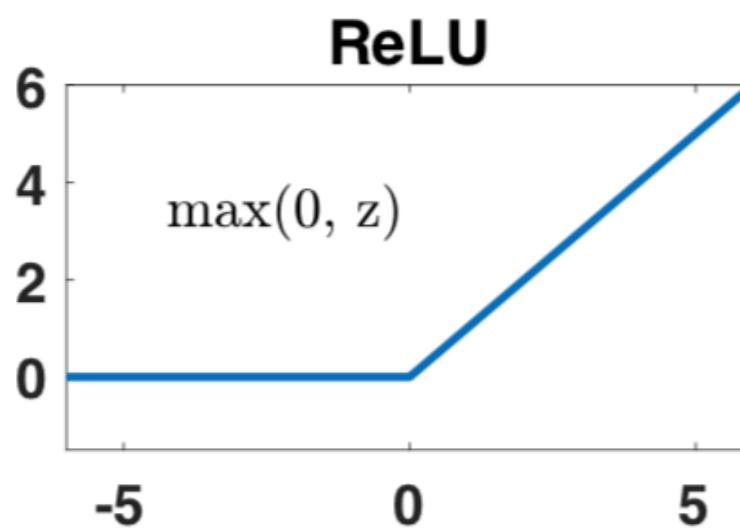
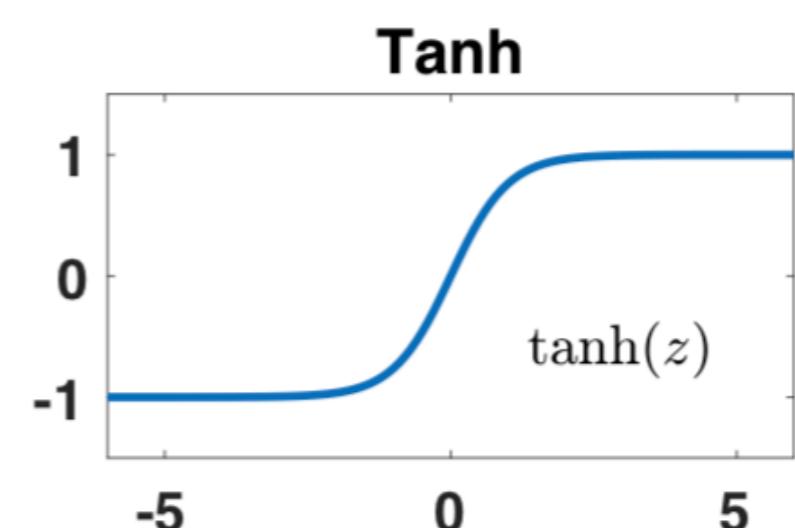
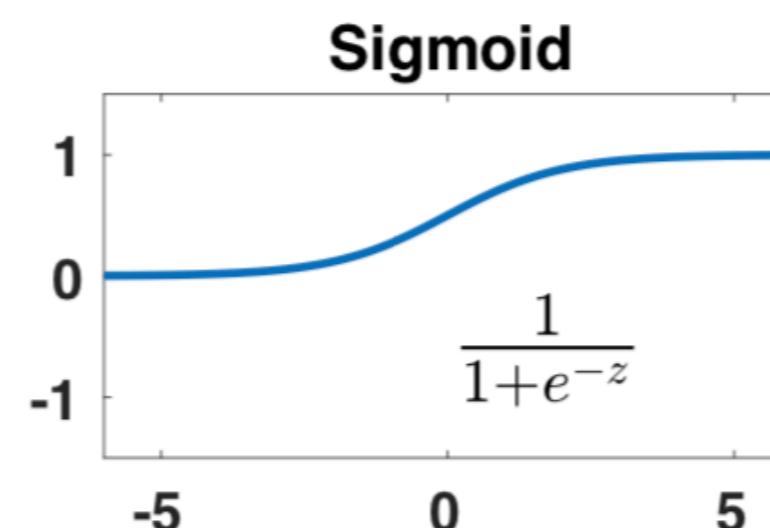
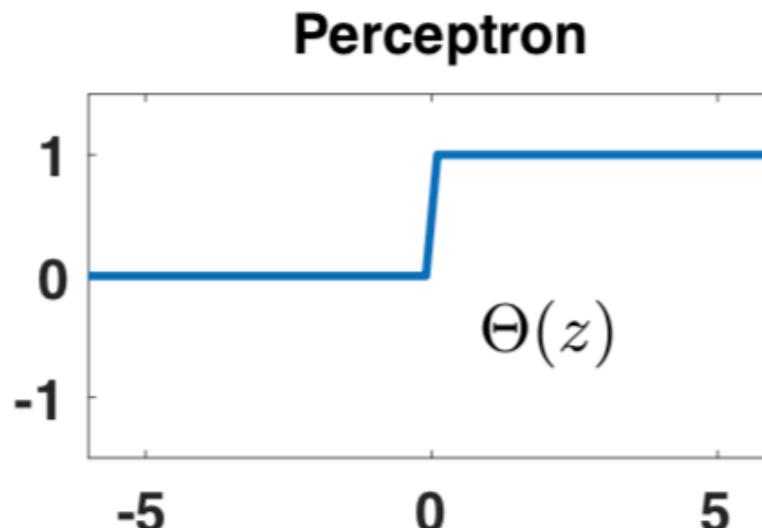
# Activation functions



***what is the main difference between these various activation functions?***

***and how it could affect NN training?***

# Activation functions



- Activation functions can be classified between those that **saturate at large inputs** (e.g. sigmoid) and those that **do not saturate at large inputs** (e.g. Rectified Linear Units ReLU)
- The choice of non-linearities has important implications for **GD NN training methods**:

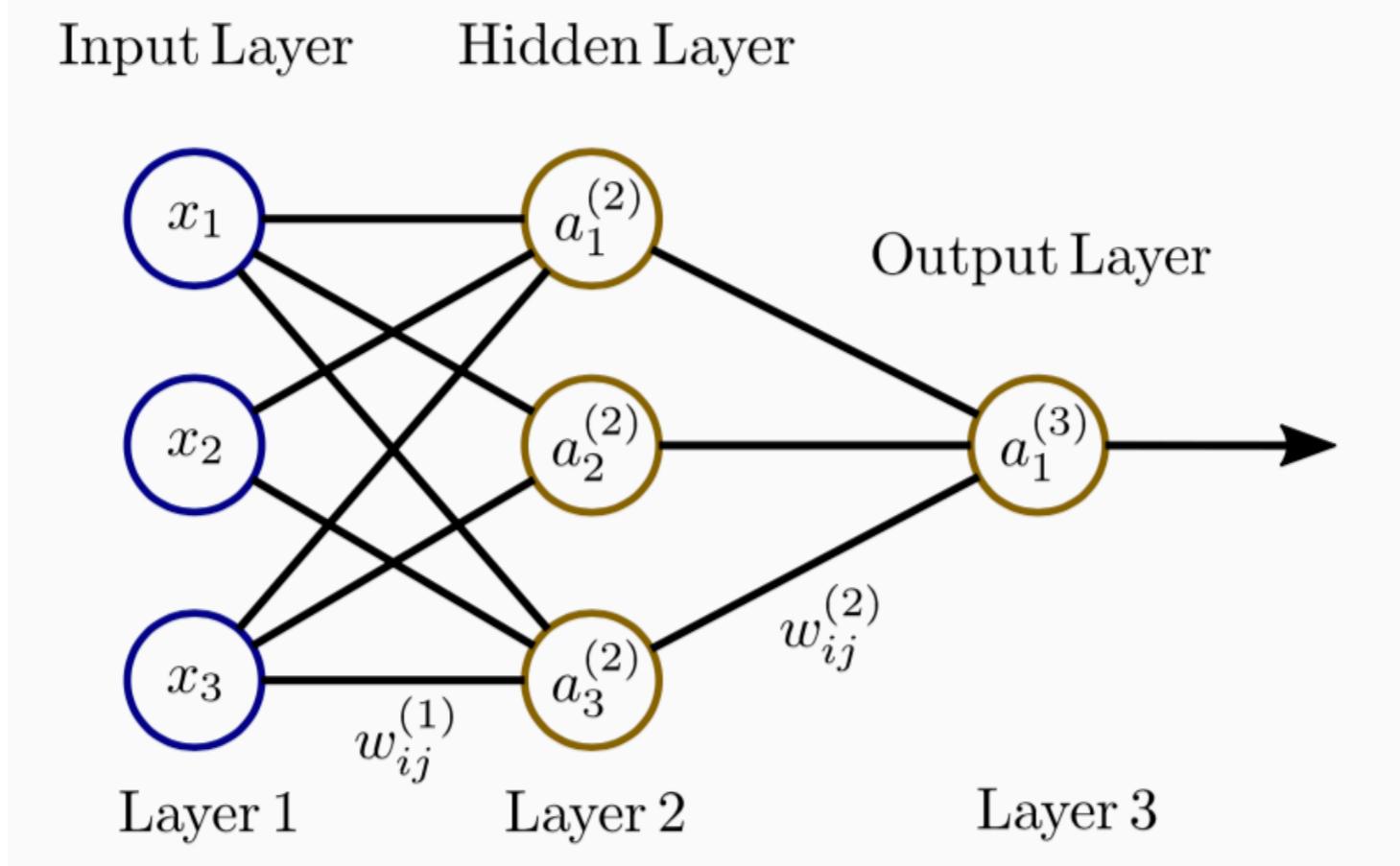
$$\text{sigmoid} \rightarrow \left. \frac{d\sigma}{dz} \right|_{z \gg 1} = 0$$

*vanishing gradients are problematic for deep networks: loss of sensitivity*

$$\text{ReLU} \rightarrow \left. \frac{d\sigma}{dz} \right|_{z \gg 1} \neq 0$$

# A simple network

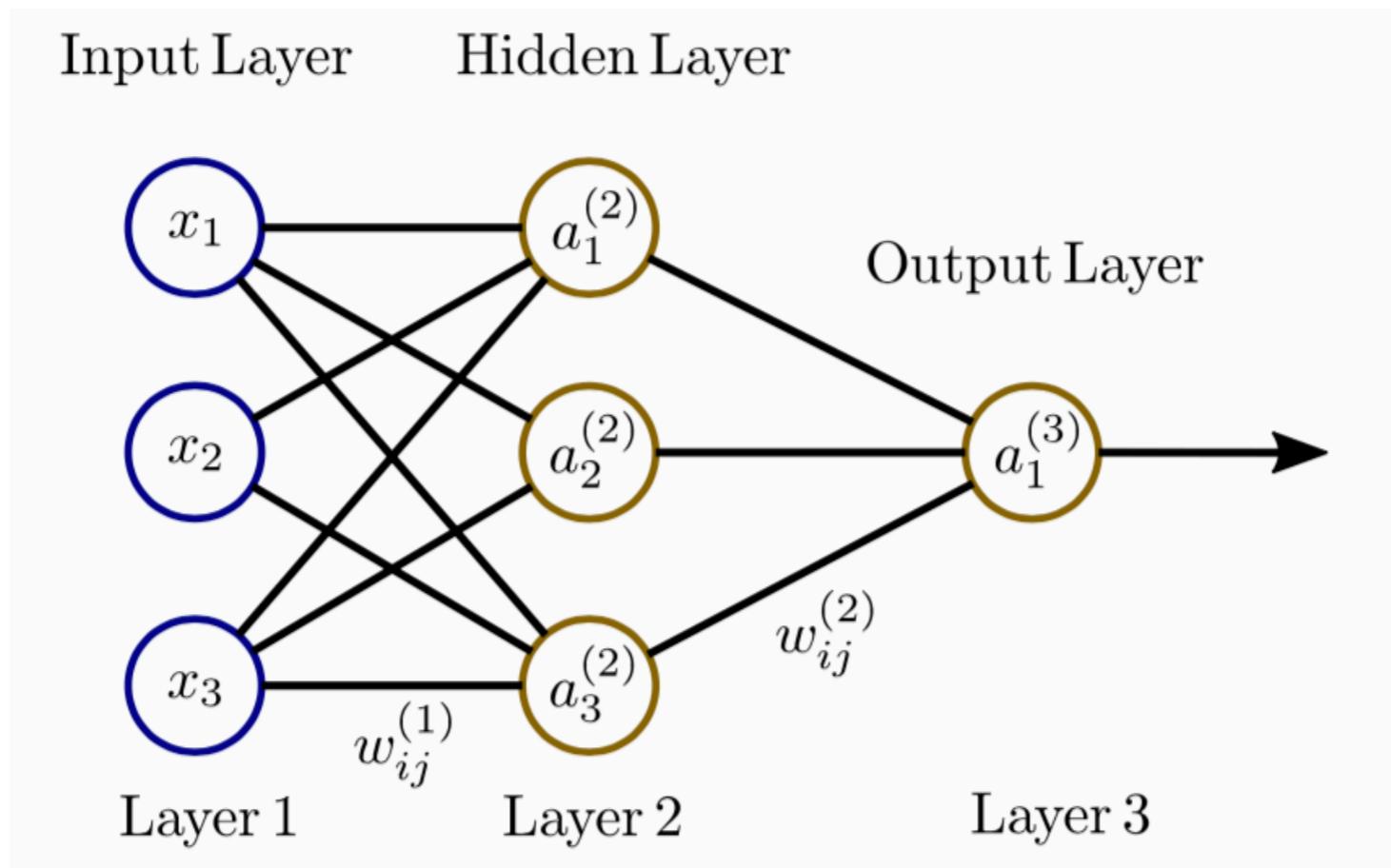
to realise that NNs are nothing mysterious but just a **complex non-linear mapping** between inputs and outputs, consider the explicit calculation of the output of a simple network



*can one evaluate analytically  $a_1^{(3)}$  as function of  $x_1$ ,  $x_2$ ,  $x_3$ ?*

# A simple network

to realise that NNs are nothing mysterious but just a **complex non-linear mapping** between inputs and outputs, consider the explicit calculation of the output of a simple network



using the three inputs (Layer 1), compute activation states of neurons in Layer 2

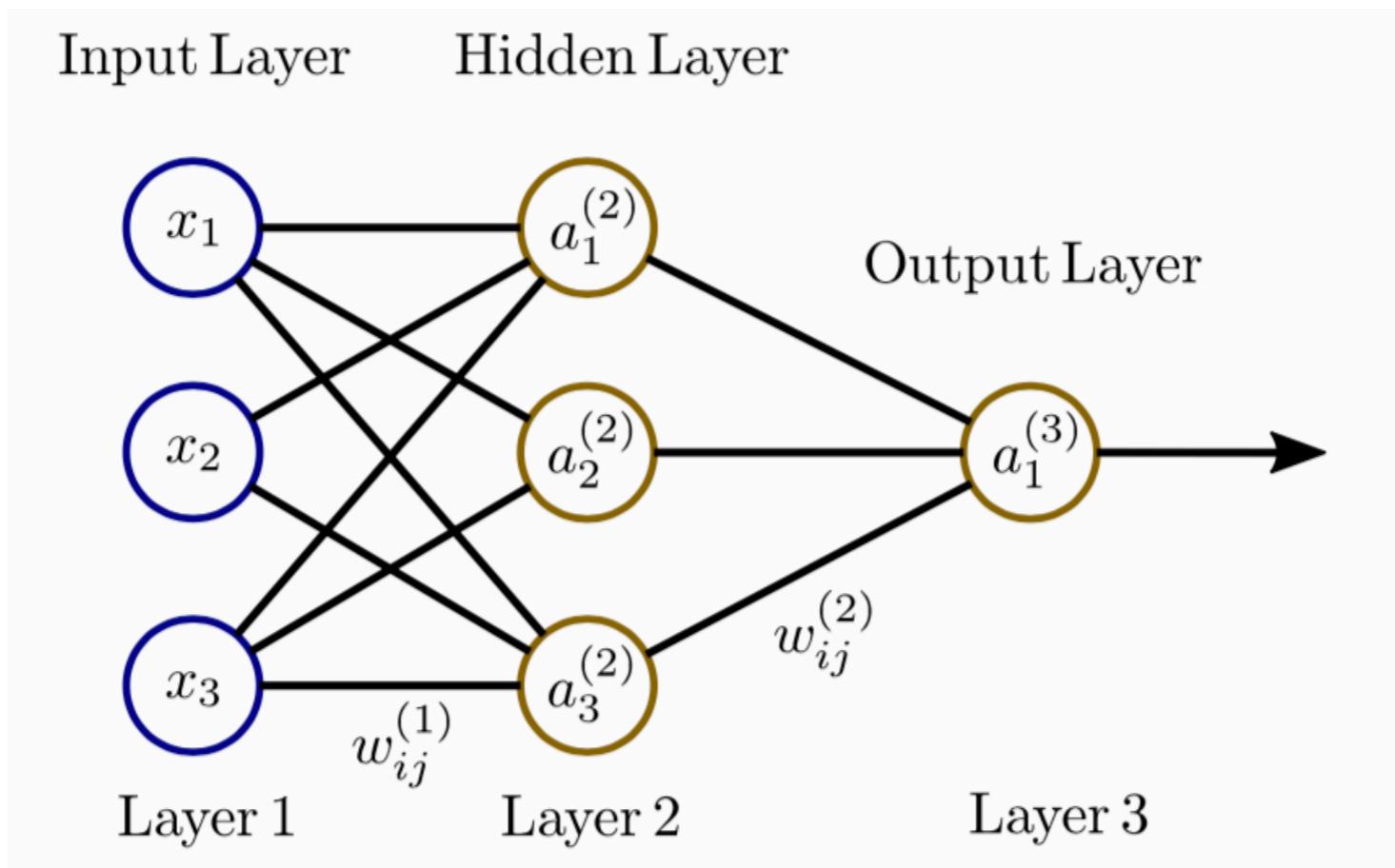
$$a_1^{(2)} = \sigma \left( x_1 \theta_{11}^{(1)} + x_2 \theta_{12}^{(1)} + x_3 \theta_{13}^{(1)} + \theta_{10}^{(1)} \right)$$

$$a_2^{(2)} = \sigma \left( x_1 \theta_{21}^{(1)} + x_2 \theta_{22}^{(1)} + x_3 \theta_{23}^{(1)} + \theta_{20}^{(1)} \right)$$

$$a_3^{(2)} = \sigma \left( x_1 \theta_{31}^{(1)} + x_2 \theta_{32}^{(1)} + x_3 \theta_{33}^{(1)} + \theta_{30}^{(1)} \right)$$

# A simple network

to realise that NNs are nothing mysterious but just a **complex non-linear mapping** between inputs and outputs, consider the explicit calculation of the output of a simple network



using the activation states of neurons in Layer 2, compute activation state of output neuron

$$a_1^{(3)} = \sigma \left( a_1^{(2)}\theta_{11}^{(2)} + a_2^{(2)}\theta_{12}^{(2)} + a_3^{(2)}\theta_{13}^{(2)} + \theta_{10}^{(2)} \right)$$

NN output is **analytical function** of the inputs and the model parameters

# A simple network

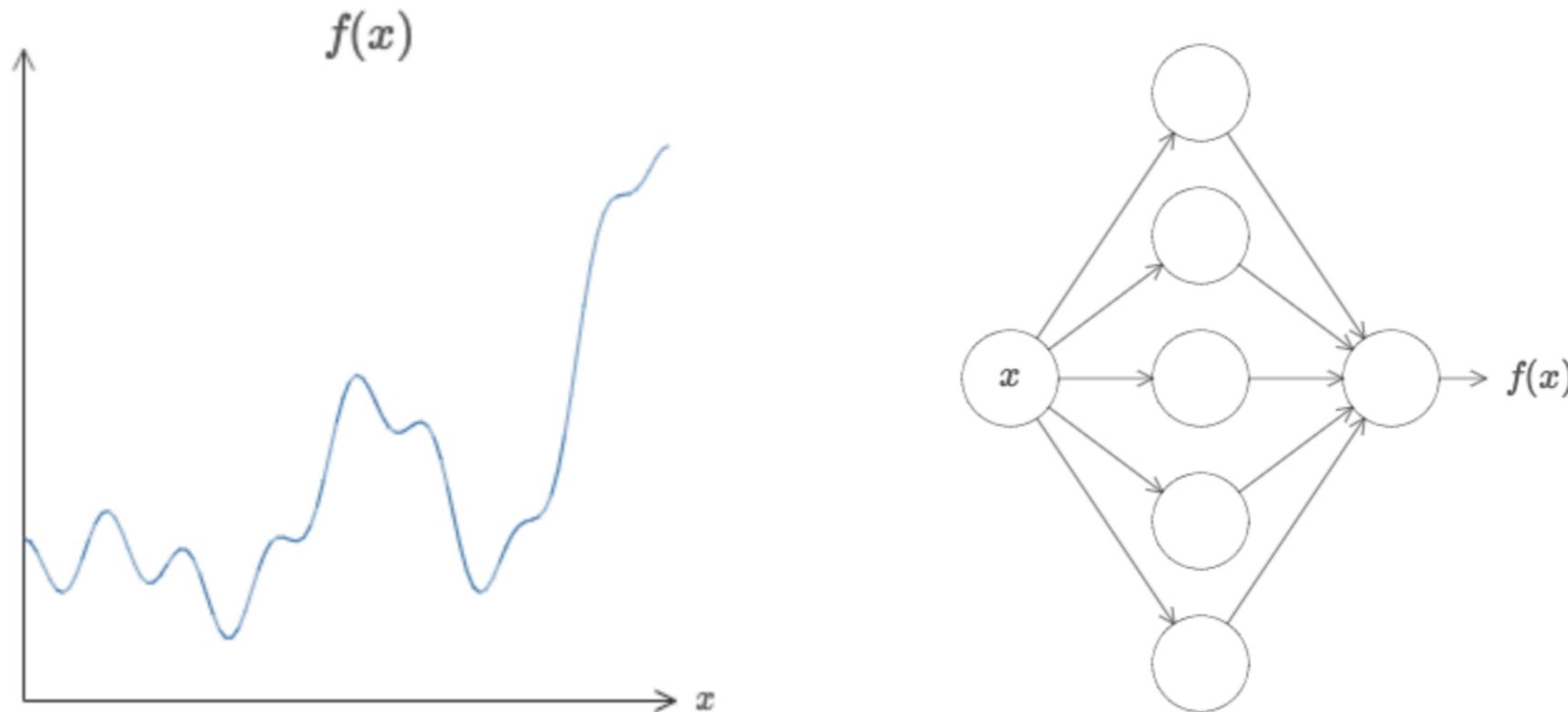
to realise that NNs are nothing mysterious but just a **complex non-linear mapping** between inputs and outputs, consider the explicit calculation of the output of a simple network

$$a_1^{(3)} = \sigma \left( \sigma \left( x_1 \theta_{11}^{(1)} + x_2 \theta_{12}^{(1)} + x_3 \theta_{13}^{(1)} + \theta_{10}^{(1)} \right) \theta_{11}^{(2)} + \right.$$
$$\sigma \left( x_1 \theta_{21}^{(1)} + x_2 \theta_{22}^{(1)} + x_3 \theta_{23}^{(1)} + \theta_{20}^{(1)} \right) \theta_{12}^{(2)} +$$
$$\left. \sigma \left( x_1 \theta_{31}^{(1)} + x_2 \theta_{32}^{(1)} + x_3 \theta_{33}^{(1)} + \theta_{30}^{(1)} \right) \theta_{13}^{(2)} + \theta_{10}^{(2)} \right)$$

substitute the activation function e.g. for the sigmoid and you have the **analytic output** of a simple NN

# 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

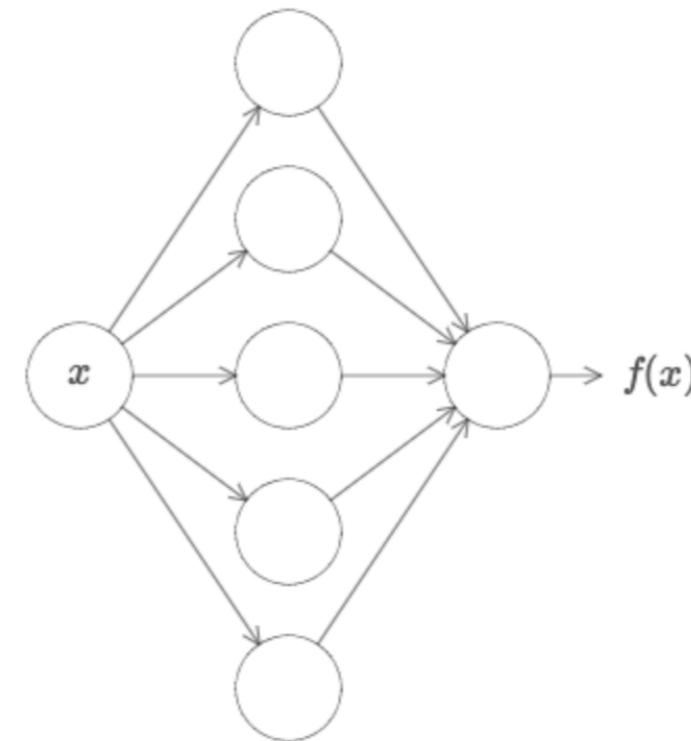
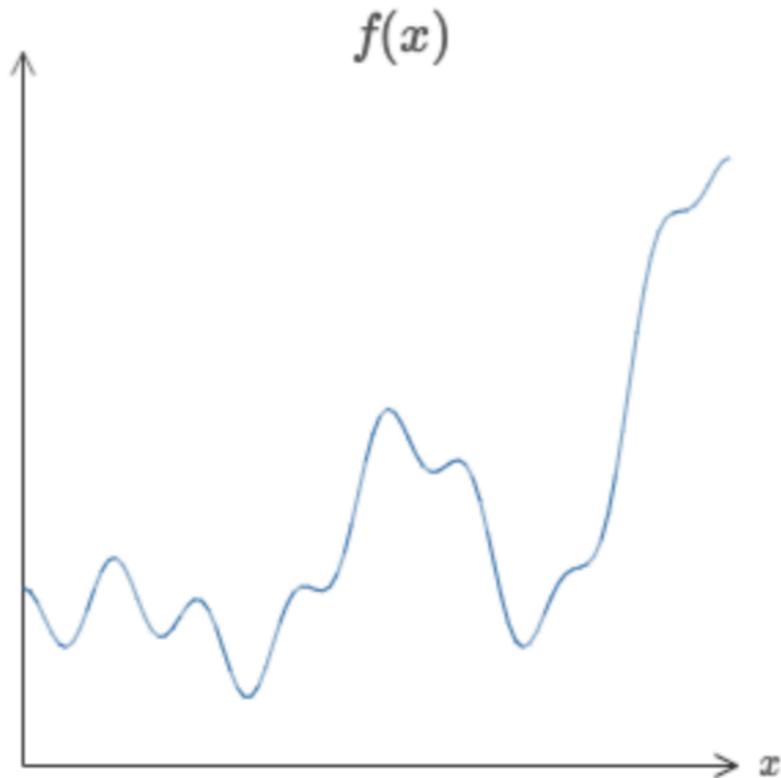


neural networks exhibit *universality properties*: no matter what function we want to compute, we know (theorem!) that there is a neural network which can carry out this task

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

# 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

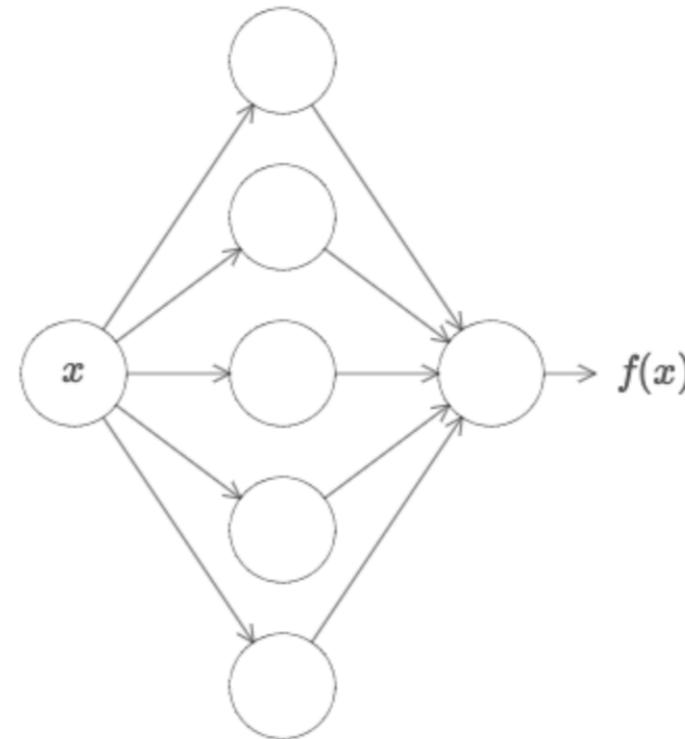
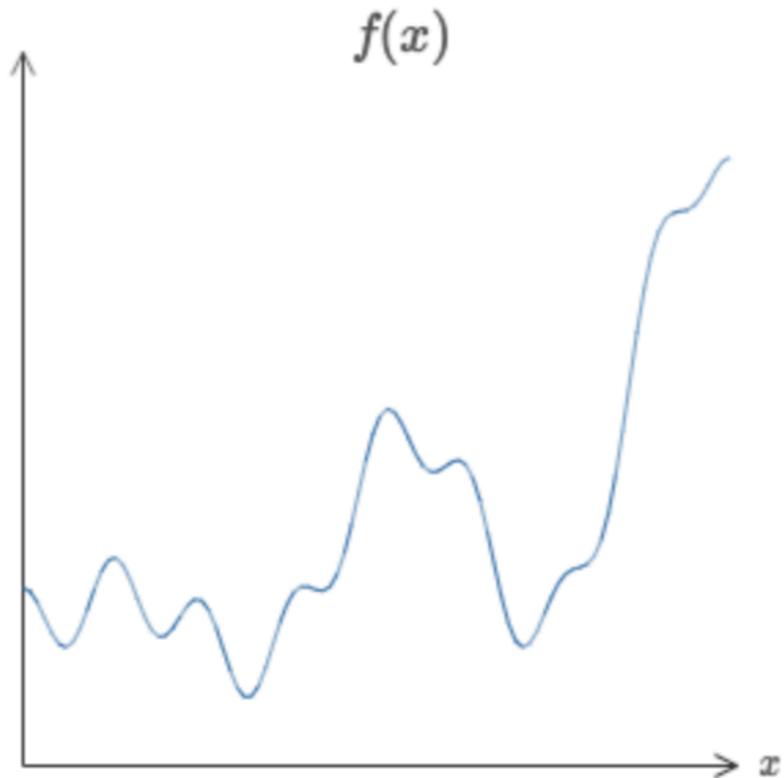


*what do you think is the advantage as compared to e.g. Chebyshev polynomials?*

$$f(x) = \sum_{i=1}^n a_i T_i(x)$$

# 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



*what do you think is the advantage as compared to e.g. Chebyshev polynomials?*

$$f(x) = \sum_{i=1}^n a_i T_i(x)$$

- i) *interpolating polynomials unstable for large n*
- ii) *difficult to generalise to multiple inputs*

# **Deep Learning & Backpropagation**

# Deep Learning

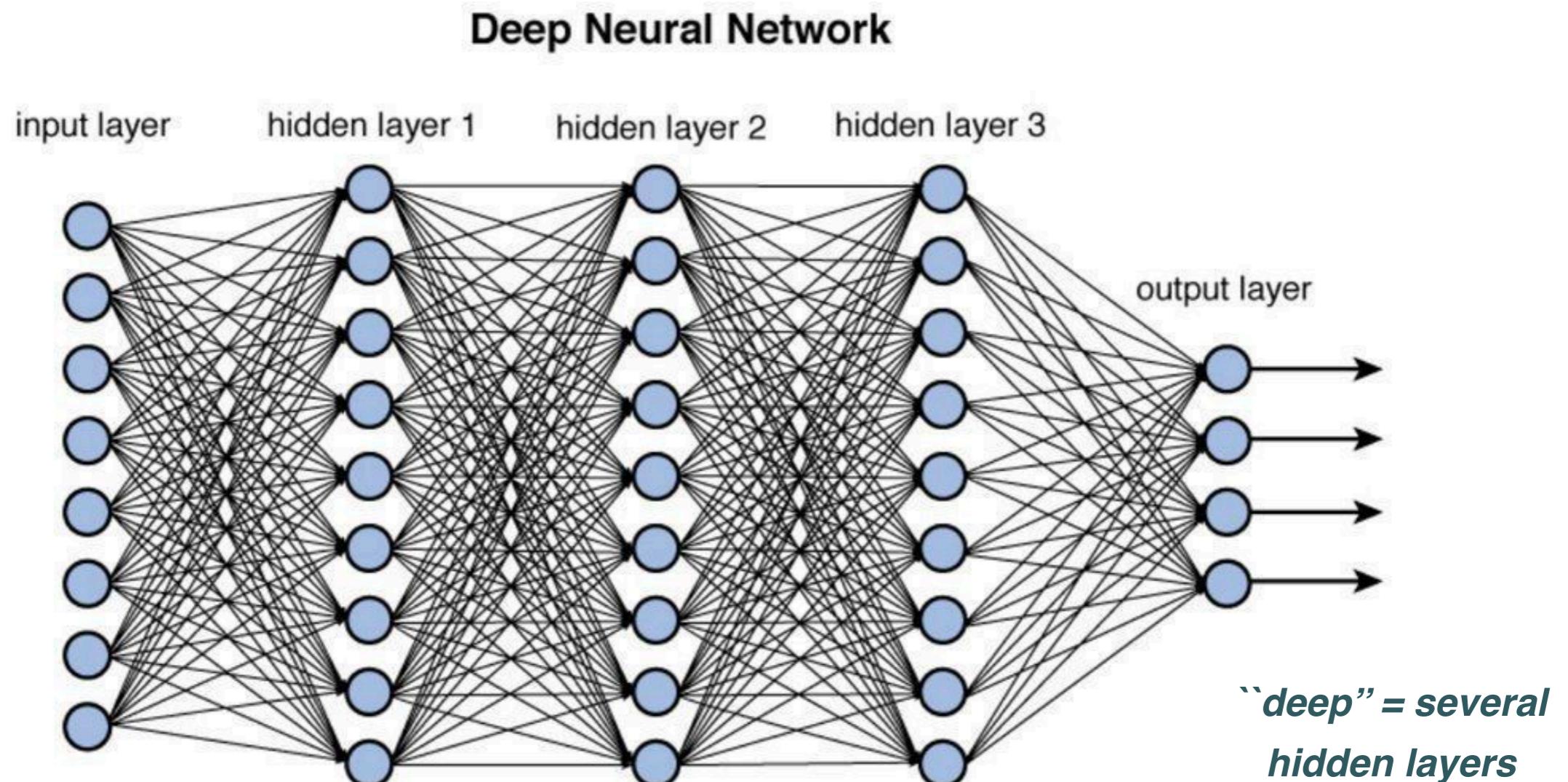
*What do you think the term ``deep learning'' means?*

*Is there a qualitative difference with vanilla artificial neural networks?*

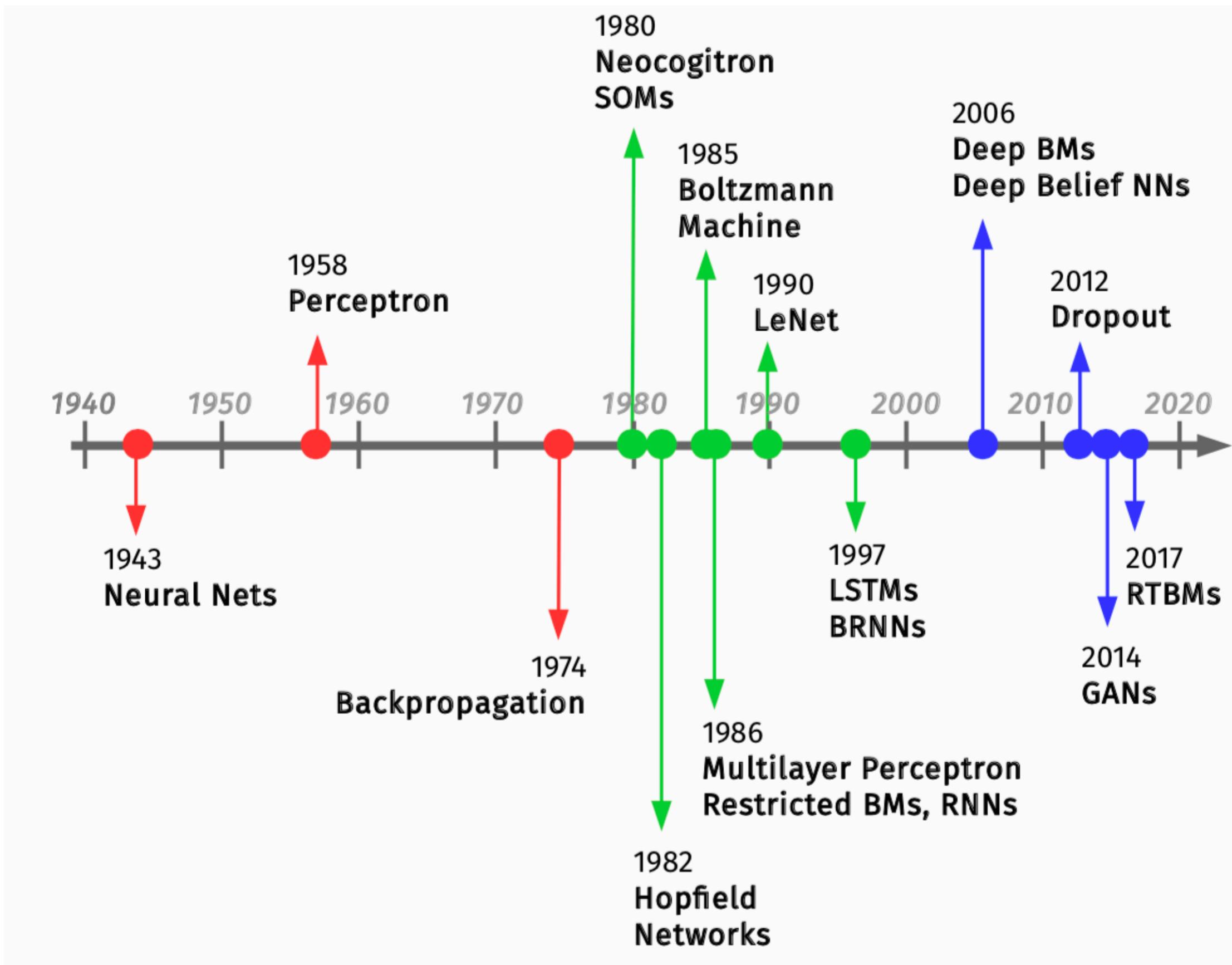
# Deep Networks

A neural network can thus be thought of a **complicated non-linear mapping** between the inputs and the outputs that depends on the parameters (weights and bias) of each neuron

We can make a NN **deep** by adding hidden layers, which greatly expands their **representational power**, also known as **expressivity**



# A timeline of neural networks



# NN training

As standard in **Supervised Learning**, the first step to train a NN is to specify a **cost function**

$$(x_i, y_i) , \quad i = 1, \dots, n \quad \longrightarrow \quad \hat{y}_i(\theta) , \quad i = 1, \dots, n$$

*for each of the  $n$  data points ...*

*... the output of the NN provides the model prediction*

the loss function depends on whether the NN should provide **continuous or categorical** (discrete) predictions. For continuous data we can have the **mean square error**

$$E(\theta) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i(\theta))^2$$

for categorical data we use the **cross-entropy**, which for binary (true/false) classification is

$$E(\theta) = \frac{1}{n} \sum_{i=1}^n (y_i \ln \hat{y}_i(\theta) + (1 - y_i) \ln(1 - \hat{y}_i(\theta)))$$

where the true labels satisfy  $y_i \in \{0,1\}$

NN training are based on a specific version of GD methods: **backpropagation**

# Backpropagation

For deep NNs the brute-force evaluation of the **gradients of the cost function** is impractical

*GD method:*

$$\mathbf{v}_t = \eta_t \nabla_{\theta} E(\theta_t), \quad \theta_{t+1} = \theta_t - \mathbf{v}_t$$

*learning rate*      *gradient of cost function*      *update of model parameters between iterations  $t$  and  $t+1$*



**Why? What prevents us from using simple GD to train NNs?**

# Backpropagation

For deep NNs the brute-force evaluation of the **gradients of the cost function** is impractical

*GD method:*

$$\mathbf{v}_t = \eta_t \nabla_{\theta} E(\theta_t), \quad \theta_{t+1} = \theta_t - \mathbf{v}_t$$

*learning rate*      *gradient of cost function*      *update of model parameters between iterations  $t$  and  $t+1$*

***Why? What prevents us from using simple GD to train NNs?***

- i) *In general cost function flat in many parameters: flat directions*
- ii) *Numerically instability of nested gradients*
- iii) *Requires evaluating network activation states multiple times for every single iteration*

# Backpropagation

For deep NNs the brute-force evaluation of the **gradients of the cost function** is impractical

**GD method:**

$$\mathbf{v}_t = \eta_t \nabla_{\theta} E(\theta_t), \quad \theta_{t+1} = \theta_t - \mathbf{v}_t$$

*learning rate*                   *gradient of cost function*                   *update of model parameters between iterations  $t$  and  $t+1$*

instead one uses **backpropagation**, which cleverly exploits the **layered structure** of NNs

we will use the following notation:

- A neural network with  **$L$  layers**, labelled as  $l=1,\dots,L$
- $\omega_{jk}^{(l)}$ : **weights** connecting  $k$ -th neuron in the  $(l-1)$ -th layer to  $j$ -th neuron in the  $l$ -th layer
- $b_j^{(l)}$ : **bias** of the  $j$ -th neuron in the  $l$ -th layer
- $a_j^{(l)}$ : **activation state** of the  $j$ -th neuron in the  $l$ -th layer

# Backpropagation

**feed-forward NNs:** the activation state of the  $j$ -th neuron in the  $l$ -th layer is a (nonlinear) function of the activation states of all the neurons in the  $(l-1)$ -th layer

$$a_j^{(l)} = \sigma \left( \sum_{k=1}^{n_{l-1}} \omega_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)} \right) \equiv \sigma(z_j^{(l)}), \quad j = 1, \dots, n_l$$

the cost function of the NN therefore depends on:

- $a_j^{(L)}$ : **activation states** of the neurons on the **output layer  $L$**  (directly)
- $a_j^{(l)}$ : **activation states** of the neurons on the hidden layers  $l < L$  (indirectly)

we define the **error** associated to the  $j$ -th neuron in the output layer and hidden layers as

$$\Delta_j^{(L)} \equiv \frac{\partial E(\theta)}{\partial z_j^{(L)}} \quad \Delta_j^{(l)} \equiv \frac{\partial E(\theta)}{\partial z_j^{(l)}} = \frac{\partial E(\theta)}{\partial a_j^{(l)}} \frac{da_j^{(l)}}{dz_j^{(l)}}$$

*here assume that output layer is linear: ensures that model predictions are unbounded*

# Backpropagation

$$\Delta_j^{(L)} = \frac{\partial E(\theta)}{\partial z_j^{(L)}} \quad \Delta_j^{(l)} = \frac{\partial E(\theta)}{\partial z_j^{(l)}} = \frac{\partial E(\theta)}{\partial a_j^{(l)}} \sigma' \left( z_j^{(l)} \right)$$

to derive the backpropagation equations, we will use the chain rule & NN layer structure

$$\Delta_j^{(l)} = \frac{\partial E(\theta)}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \frac{\partial E(\theta)}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \Delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}$$

where here we exploit the **NN layered structure**: activation states of neurons in  $(l+1)$ -th layer depends only on activation states of neurons in  $l$ -th layer

$$\Delta_j^{(l)} = \frac{\partial E(\theta)}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \Delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \left( \sum_k \Delta_k^{(l+1)} \omega_{kj}^{(l+1)} \right) \sigma' \left( z_j^{(l)} \right)$$

# Backpropagation

$$\Delta_j^{(L)} = \frac{\partial E(\theta)}{\partial z_j^{(L)}} \quad \Delta_j^{(l)} = \frac{\partial E(\theta)}{\partial z_j^{(l)}} = \frac{\partial E(\theta)}{\partial a_j^{(l)}} \sigma' \left( z_j^{(l)} \right)$$

to derive the backpropagation equations, we will use the chain rule & NN layer structure

$$\Delta_j^{(l)} = \frac{\partial E(\theta)}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \frac{\partial E(\theta)}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \Delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}$$

where here we exploit the **NN layered structure**: activation states of neurons in  $(l+1)$ -th layer depends only on activation states of neurons in  $l$ -th layer

$$\Delta_j^{(l)} = \frac{\partial E(\theta)}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \Delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \left( \sum_k \Delta_k^{(l+1)} \omega_{kj}^{(l+1)} \right) \sigma' \left( z_j^{(l)} \right)$$

$$z_k^{(l+1)} = \sum_{k'=1}^{n_l} \omega_{jk'}^{(l+1)} a_{k'}^{(l)} + b_j^{(l+1)} = \sum_{k'=1}^{n_l} \omega_{jk'} \sigma(z_{k'}^{(l)}) + b_j^{(l+1)}$$

# Backpropagation

$$\Delta_j^{(L)} = \frac{\partial E(\theta)}{\partial z_j^{(L)}} \quad \Delta_j^{(l)} = \frac{\partial E(\theta)}{\partial z_j^{(l)}} = \frac{\partial E(\theta)}{\partial a_j^{(l)}} \sigma' \left( z_j^{(l)} \right)$$

to derive the backpropagation equations, we will use the chain rule & NN layer structure

$$\Delta_j^{(l)} = \frac{\partial E(\theta)}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \frac{\partial E(\theta)}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \Delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}$$

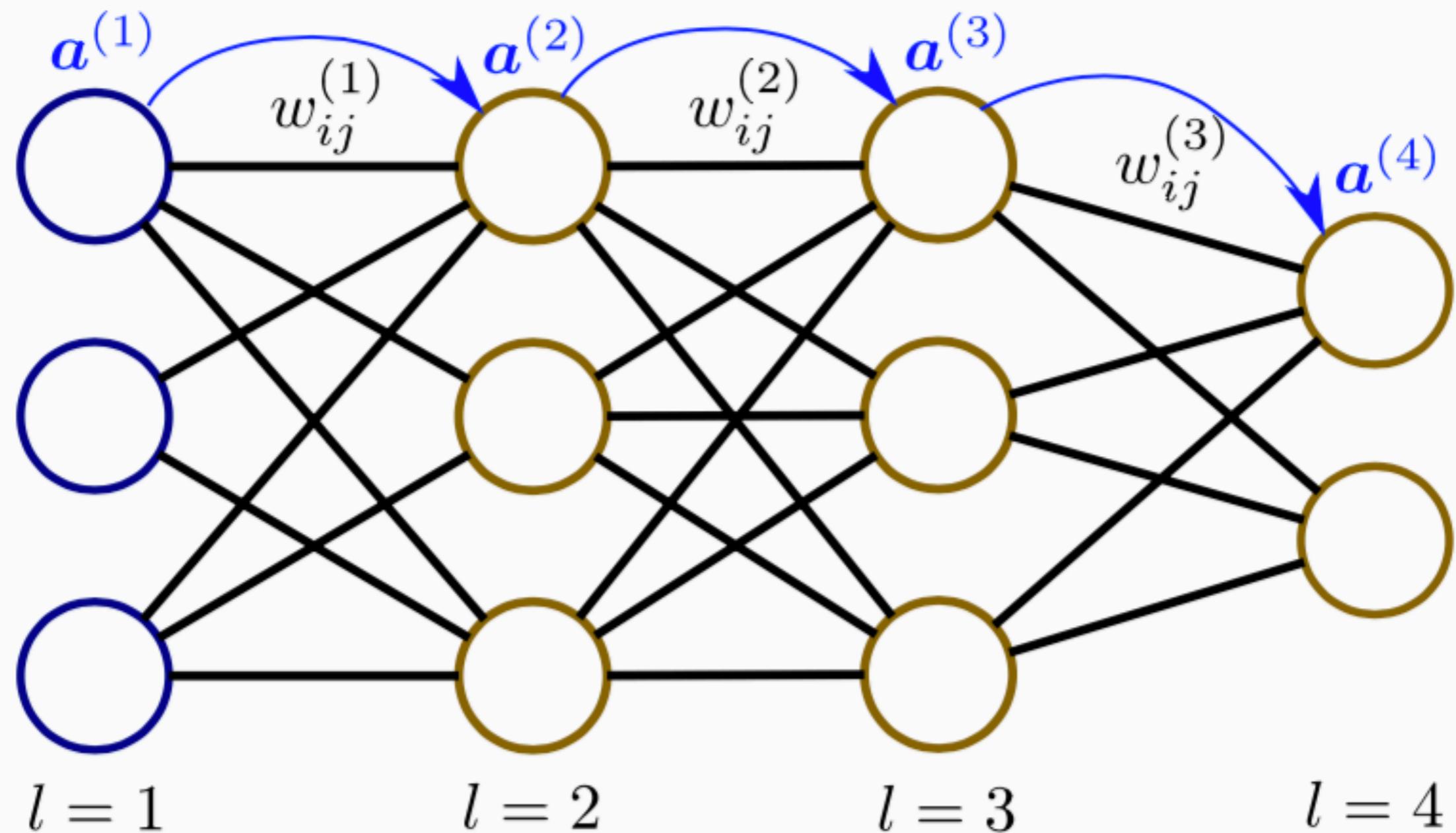
where here we exploit the **NN layered structure**: activation states of neurons in  $(l+1)$ -th layer depends only on activation states of neurons in  $l$ -th layer

$$\Delta_j^{(l)} = \frac{\partial E(\theta)}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \Delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \left( \sum_k \Delta_k^{(l+1)} \omega_{kj}^{(l+1)} \right) \sigma' \left( z_j^{(l)} \right)$$

$$\frac{\partial E}{\partial \omega_{jk}^{(l)}} = \frac{\partial E}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial \omega_{jk}^{(l)}} = \Delta_j^{(l)} a_k^{(l-1)}$$

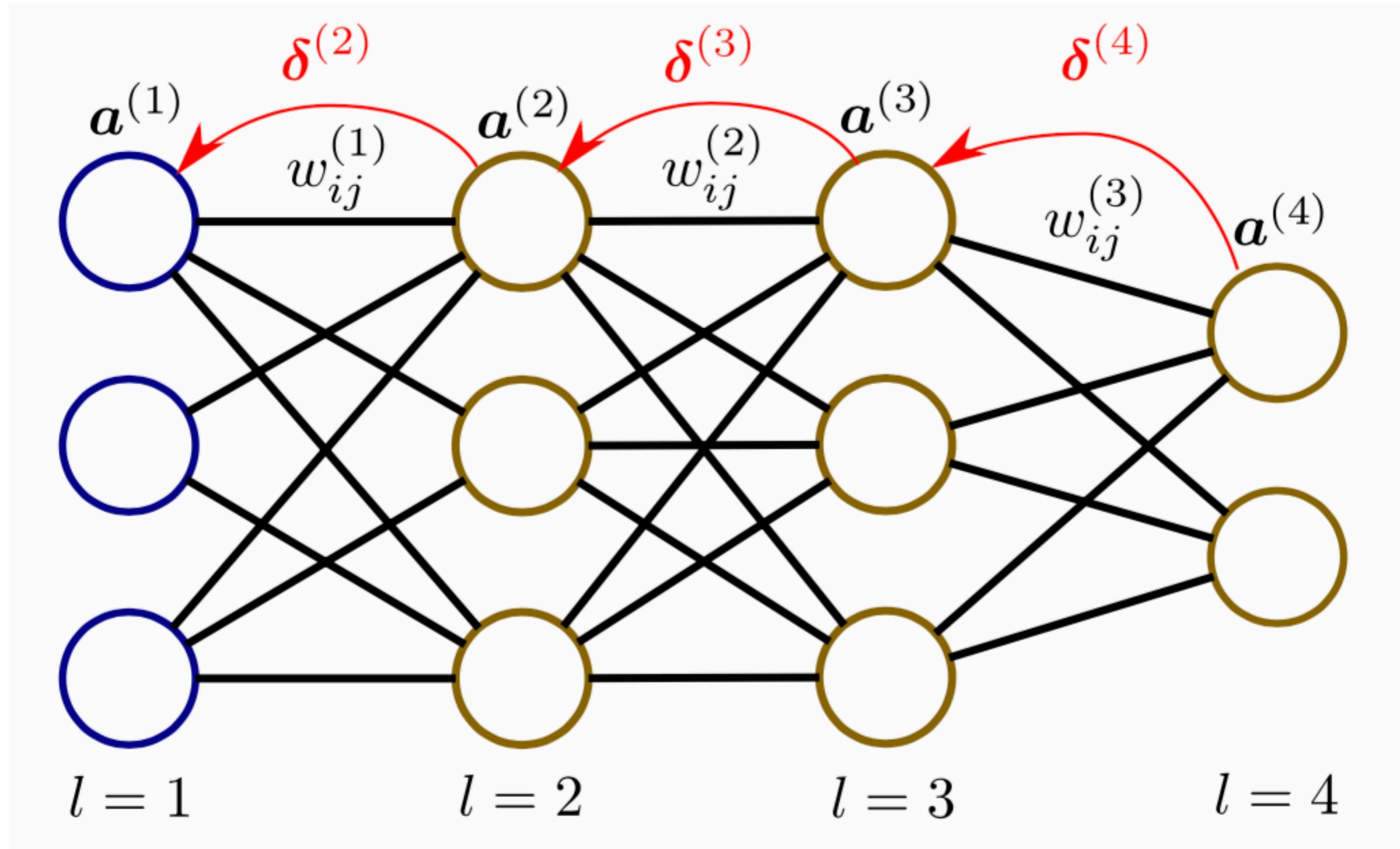
# Backpropagation

first evaluate the activation states of all neurons using **forward propagation** ...



# Backpropagation

then **backpropagate** to evaluate the **errors**  $\Delta$  and thus the gradients over model parameters



# Backpropagation

we now have all the ingredients to deploy the **backpropagation algorithm for NN training**

• (1) Evaluate activation state of neurons in input layer  $a_j^{(1)}$ ,  $j = 1, \dots, n_1$

• (2) **Feed-forward:** evaluate activation states for each subsequent layer

$$a_j^{(l)} = \sigma \left( \sum_{k=1}^{n_l} \omega_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)} \right) \equiv \sigma(z_j^{(l)}) , l = 2, \dots, L$$

• (3) With this information evaluate error on network's output layer

$$\Delta_j^{(L)} = \frac{\partial E(\theta)}{\partial z_j^{(L)}} \quad \text{here enters dependence on cost function}$$

• (4) **Backpropagate** this error to evaluate the errors on all hidden layers

$$\Delta_j^{(l)} = \left( \sum_k \Delta_k^{(l+1)} \omega_{kj}^{(l+1)} \right) \sigma' \left( z_j^{(l)} \right)$$

*error in j-th neuron in (l)-th layer*      *error in k-th neuron in (l+1)-th layer*

# Backpropagation

we now have all the ingredients to deploy the **backpropagation algorithm for NN training**

- (5) Evaluate **gradients of the model parameters** associated to the gradient of cost function

$$\frac{\partial E}{\partial \omega_{jk}^{(l)}} = \Delta_j^{(l)} a_k^{(l-1)} \quad \frac{\partial E}{\partial b_j^{(l)}} = \Delta_j^{(l)}$$

extremely efficient way of calculating the gradients of the model parameters,  
requiring only a **single forward and backward pass** of the neural network

$$\Delta_j^{(L)} = \frac{\partial E(\theta)}{\partial z_j^{(L)}}$$

*at most  $j$  numerical derivatives need to be evaluated, even in NNs with millions of parameters*

# Backpropagation

we now have all the ingredients to deploy the **backpropagation algorithm for NN training**

• (5) Evaluate **gradients of the model parameters** associated to the gradient of cost function

$$\frac{\partial E}{\partial \omega_{jk}^{(l)}} = \Delta_j^{(l)} a_k^{(l-1)} \quad \frac{\partial E}{\partial b_j^{(l)}} = \Delta_j^{(l)}$$

extremely efficient way of calculating the gradients of the model parameters,  
requiring only a **single forward and backward pass** of the neural network

with this info one can carry out with Gradient Descent and **update the model parameters**

$$\mathbf{v}_t = \eta_t \nabla_{\theta} E(\boldsymbol{\theta}_t), \quad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t$$

note that even with BP training of large (deep) networks can be **computationally intensive!**

*Note that we are hiding under the carpet an important problem:  
what happens if the NN output enters the cost function in a non-trivial manner? Where is this issue relevant in BP?*

# Backpropagation

we now have all the ingredients to deploy the **backpropagation algorithm for NN training**

• (5) Evaluate **gradients of the model parameters** associated to the gradient of cost function

$$\frac{\partial E}{\partial \omega_{jk}^{(l)}} = \Delta_j^{(l)} a_k^{(l-1)} \quad \frac{\partial E}{\partial b_j^{(l)}} = \Delta_j^{(l)}$$

extremely efficient way of calculating the gradients of the model parameters,  
requiring only a **single forward and backward pass** of the neural network

with this info one can carry out with Gradient Descent and **update the model parameters**

$$\mathbf{v}_t = \eta_t \nabla_{\theta} E(\boldsymbol{\theta}_t), \quad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t$$

note that even with BP training of large (deep) networks can be **computationally intensive!**

Further complication if cost function **depends non-trivially on NN output**

$$E(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n \left( \mathcal{F}_i^{(\text{dat})} - \mathcal{F}_i^{(\text{model})}(\hat{\mathbf{y}}; \boldsymbol{\theta}) \right)^2$$

*e.g. model requires  
integral of NN output*

# **Optimal training & Hyperoptimisation**

# NN initialisation

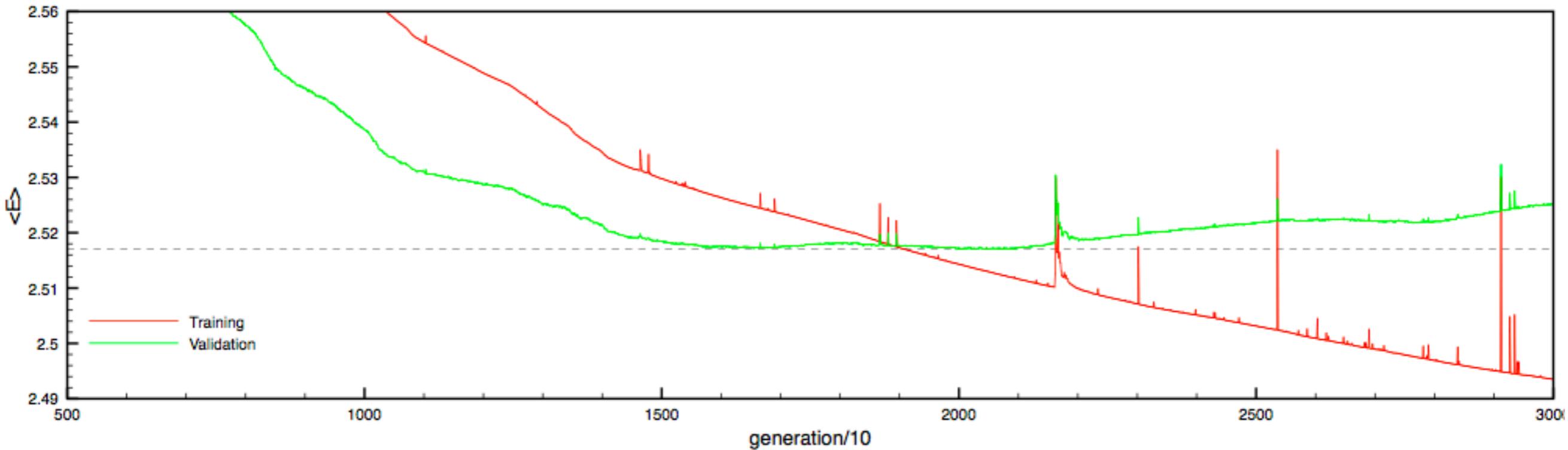
A further subtlety concerning NN training is that sometimes **a clever initialisation of the model parameters** helps in the learning process

- ✿ **zero:** all weights set to zero (initial complexity equivalent to single neuron)
- ✿ **random:** breaks parameter symmetry
- ✿ **glorot/xavier:** weights distributed randomly with Gaussian with variance based on in/out size of the neuron
- ✿ **he:** random initialisation avoiding the saturation region of the activation function

in many cases trial-and-error required to assess what works best

# NN regularisation: cross-validation

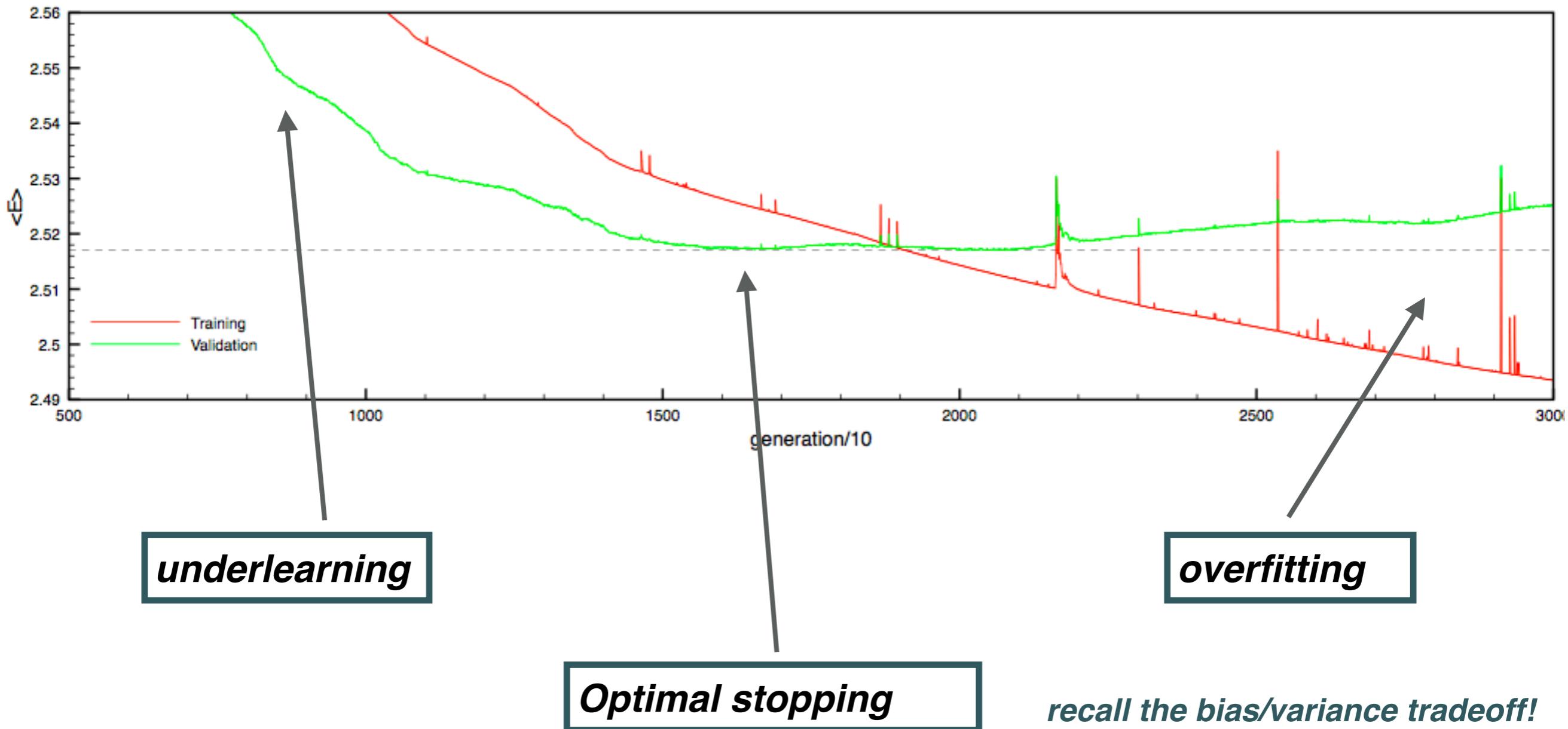
Neural networks provide extremely flexible models to describe complex datasets, but one should **avoid overfitting**, else the model will be **unable to generalise**



*what is the optimal training length for this model?*

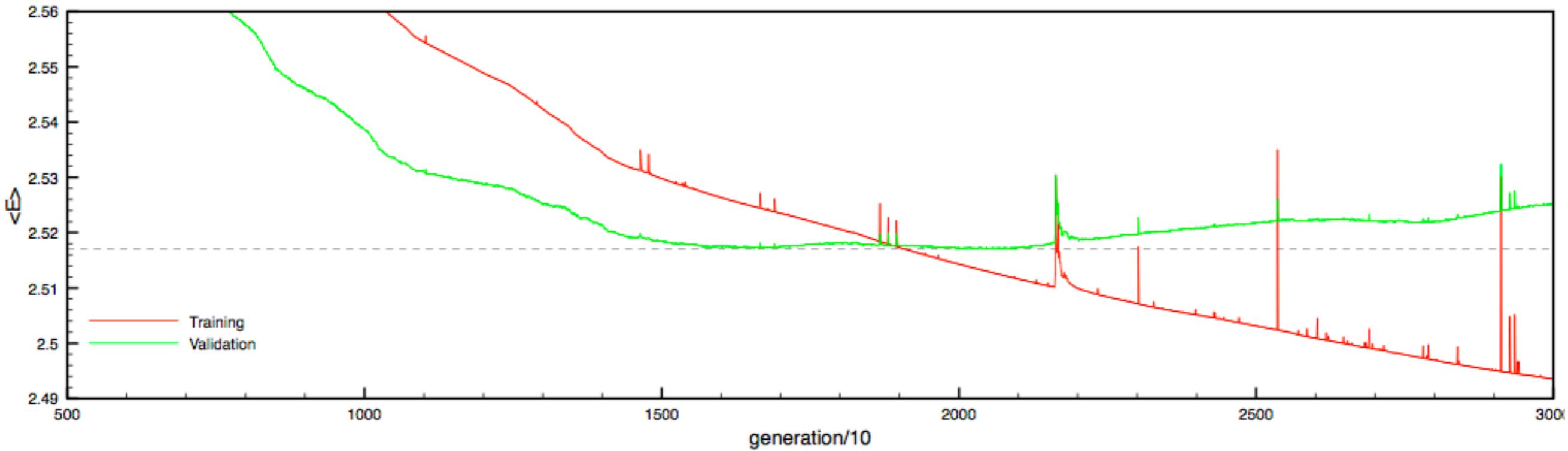
# NN regularisation: cross-validation

Neural networks provide extremely flexible models to describe complex datasets, but one should **avoid overfitting**, else the model will be **unable to generalise**



# NN regularisation: cross-validation

Neural networks provide extremely flexible models to describe complex datasets, but one should **avoid overfitting**, else the model will be **unable to generalise**

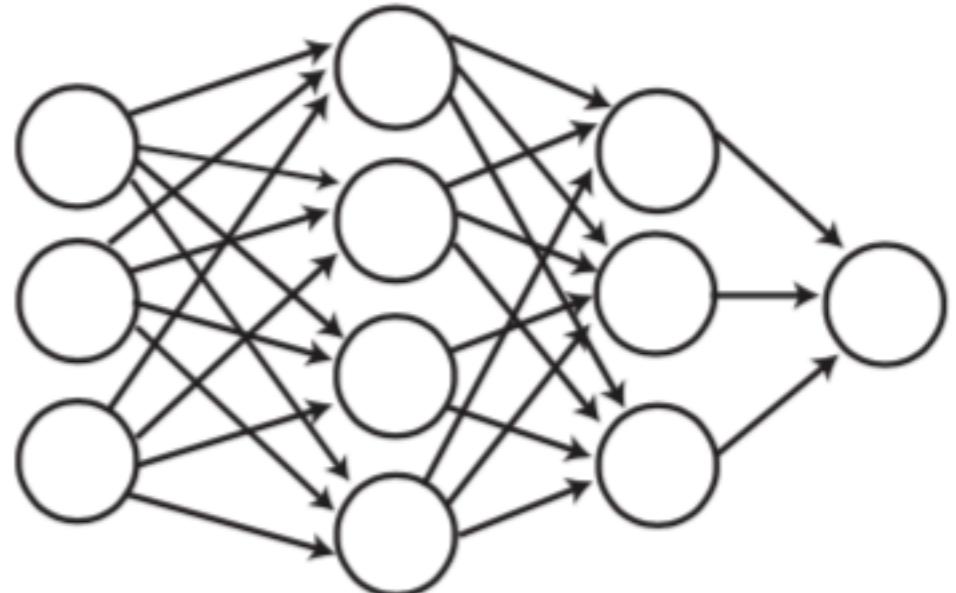


***Cross-validation look-back stopping:*** train the NN for a large fixed number of iterations. Then look back and determine the point at the training where the out-of-sample error was the smallest. Take as best-fit NN parameters those corresponding to that point of the training

# NN regularisation: Dropout

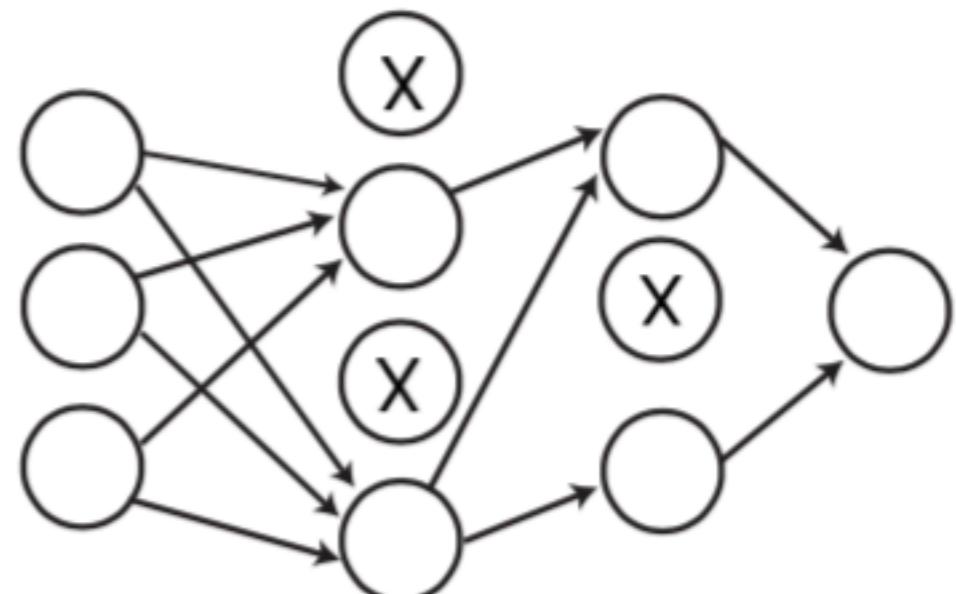
- Dropout is one of the standard **regularisation procedures** that aim to avoid overfitting when training deep NNs

Standard Neural Net



- During the training procedure, neurons are randomly “**dropped out**” of the neural network with some probability  $p$  giving rise to a thinned network, and the GD gradients are computed only there

After applying Dropout



- Dropout prevents overfitting by **reducing correlations among neurons** and reducing the variance

*effective reduction of model complexity!*

# Hyperoptimisation

In most Machine Learning applications, the model has several parameters which are typically **adjusted by hand** (trial and error) rather than algorithmically:

- Network architecture: number of layers of neurons per layer, activation functions, ...
- Choice of minimiser (which of the GD variants?)
- Learning rate, momentum, memory, size of mini-batches, ....
- Regulariation parameters, stopping, dropout rate, patience, ...

one can avoid the need of subjective choice by means of **an hyperoptimisation procedure**, where all model and training/stopping parameters are determined algorithmically

Such hyperoptimisation requires introducing a **reward function** to grade the model.  
Note that this is different from the **cost function**: the latter is optimised separately model by model (e.g. for each NN architecture) while the former compares between all optimised models

*e.g. cost function*     $C = E_{\text{tr}}$

$R = E_{\text{val}}$     *reward function*

# Hyperoptimisation

In most Machine Learning applications, the model has several **hyperparameters** which are typically **adjusted by hand** (trial and error) rather than algorithmically:

- ✿ Network architecture: number of layers of neurons per layer, activation functions, ...
- ✿ Choice of minimiser (which of the GD variants?)
- ✿ Learning rate, momentum, memory, size of mini-batches, ....
- ✿ Regularisation parameters, stopping, dropout rate, patience, ...

*how to avoid subjective choices (practically)?  
Is there such a thing as optimal hyperparameters*

# Hyperoptimisation

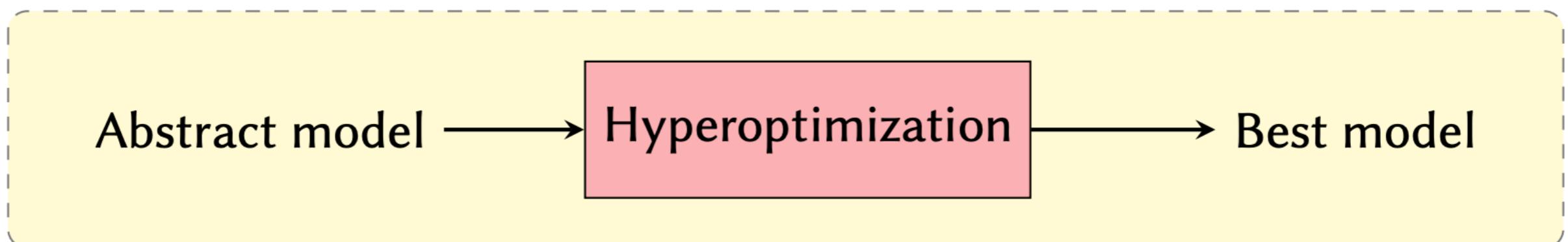
## How are hyperparameters chosen?

Humans are very good at recognising patterns → we can choose a reasonably good model by looking at a few trials, but two problems arise:

- ✓ There is no guarantee that it is the best model.
- ✓ It requires -by definition- human intervention.

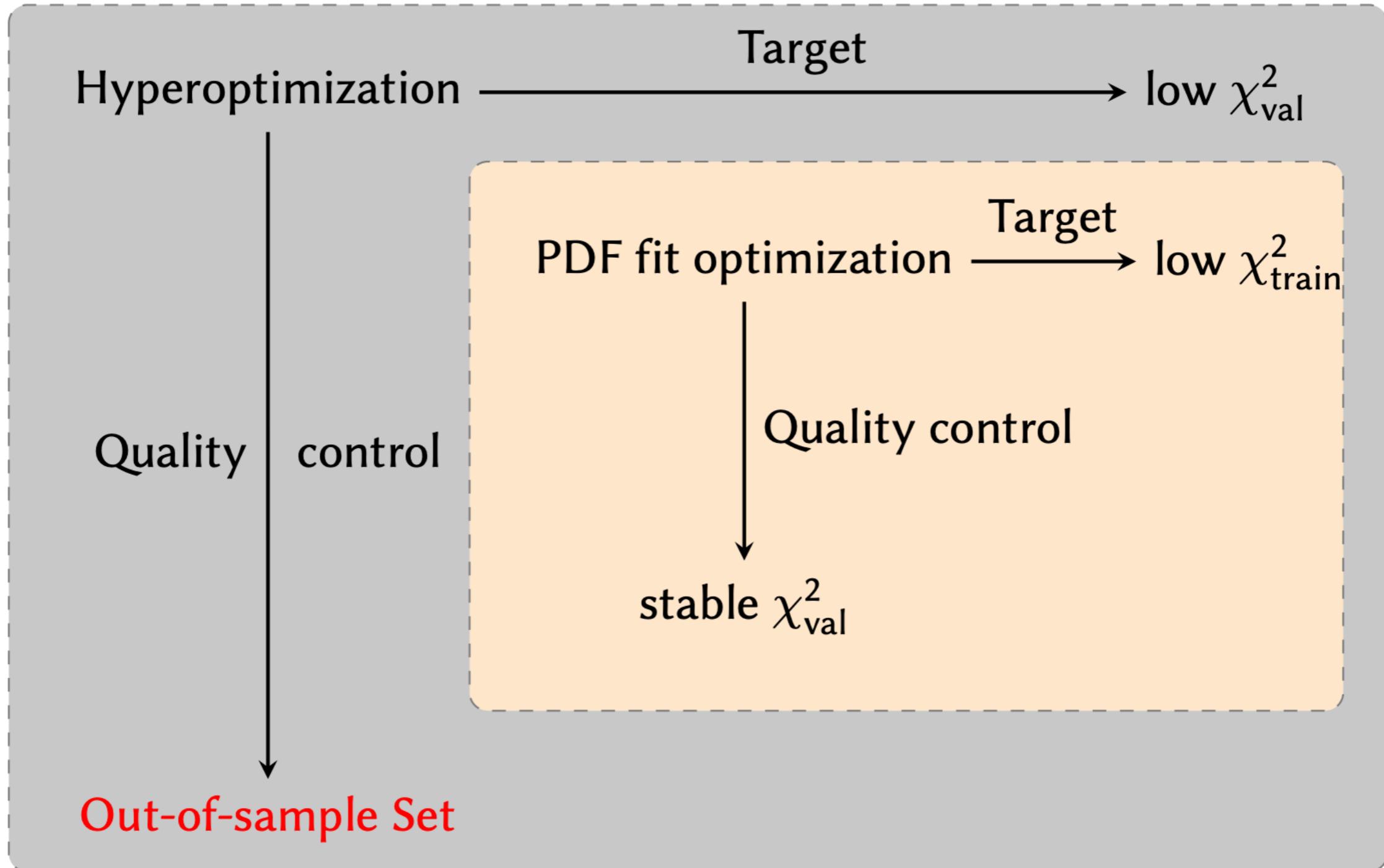
### The goal:

Run thousands of examples (with different parameters) and define a robust algorithm that will select the best model automatically.



# Hyperoptimisation

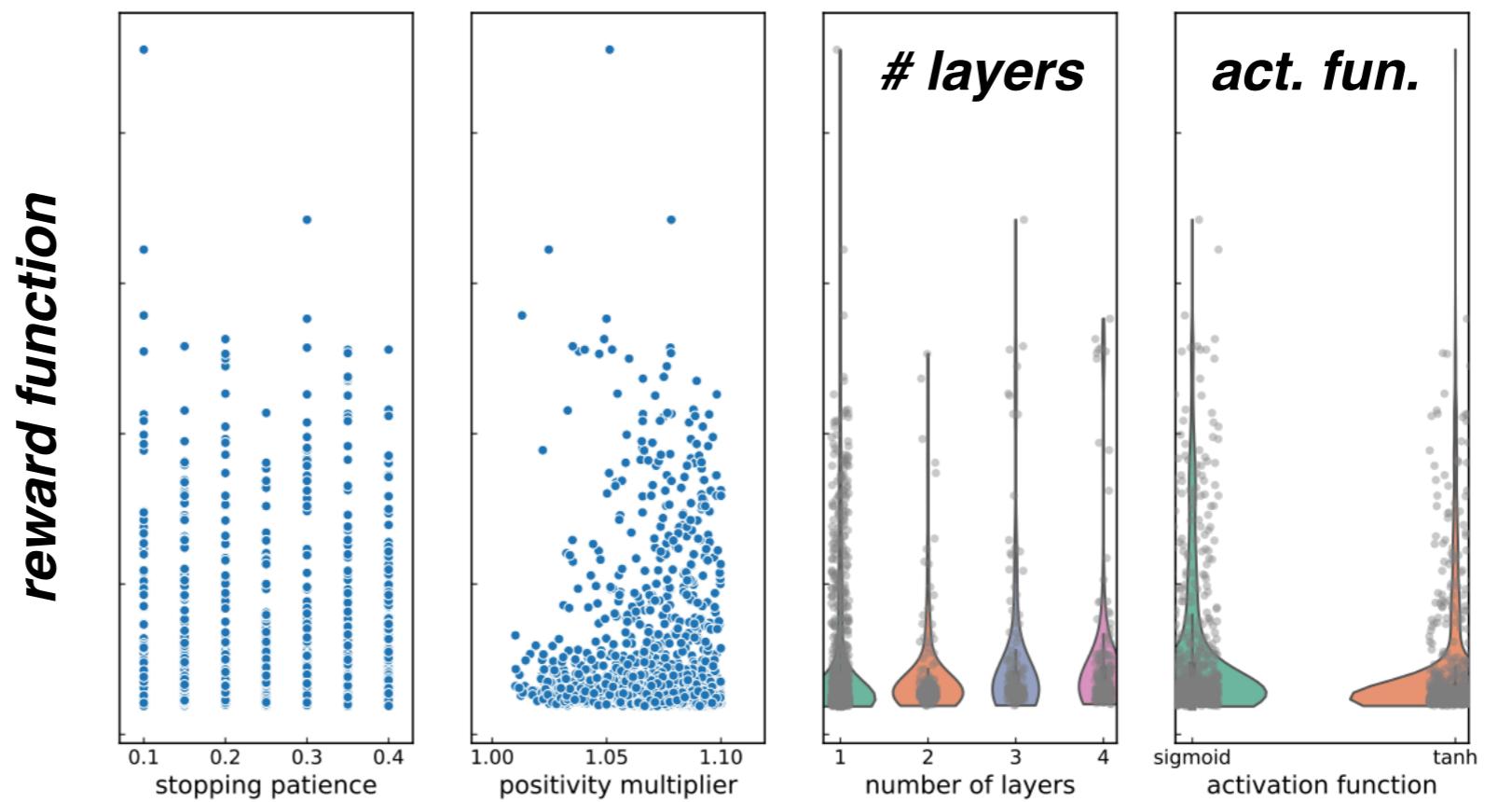
From optimization to hyperoptimization: quality control



# Hyperoptimisation

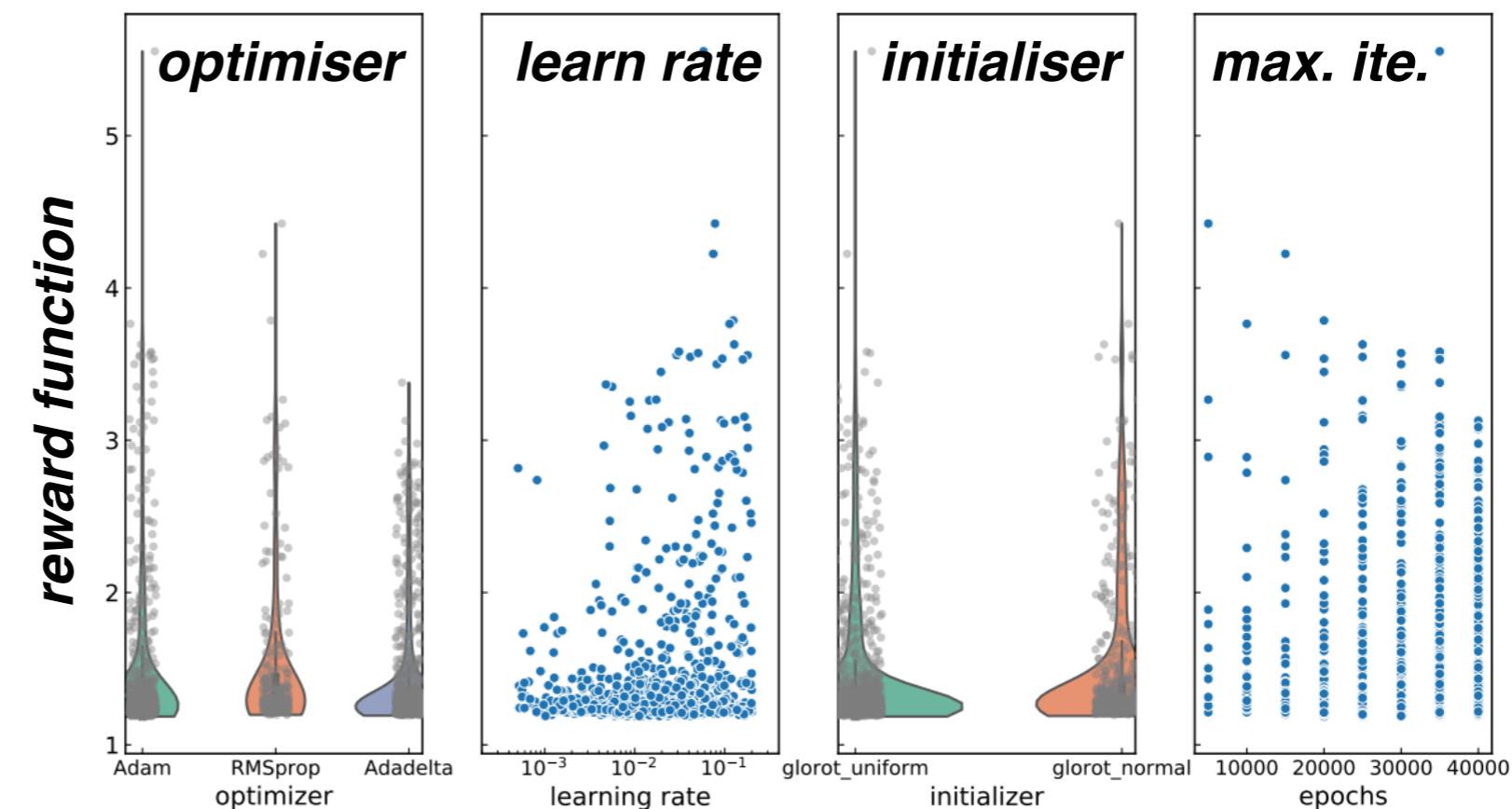
- In a hyperparameter scan one can compare the performance of **hundreds or thousands** of parameter combinations

- Some choices are **discrete** (type of minimiser, # of layers) others are **continuous** (learning rate)



- One can also **visualise** which choices are more crucial and which ones less important

- The violin plots are the **KDE-reconstructed probability distributions** for the hyperparameters



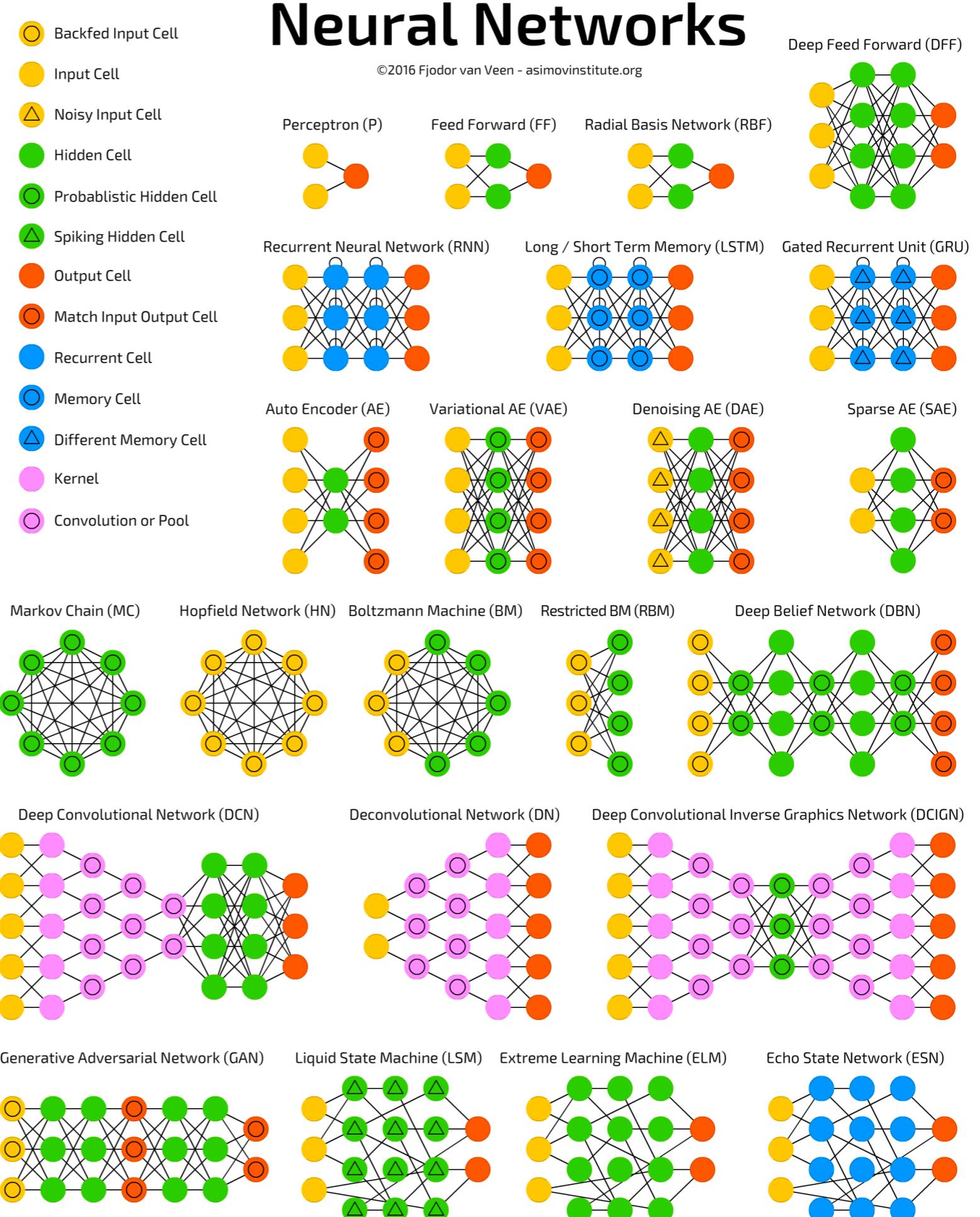
# Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

- A large variety of neural network architectures have been proposed: we will only study some of them in this course

- They differ in: number of layers and neurons, role of the neurons, connections between neurons and layers, ....

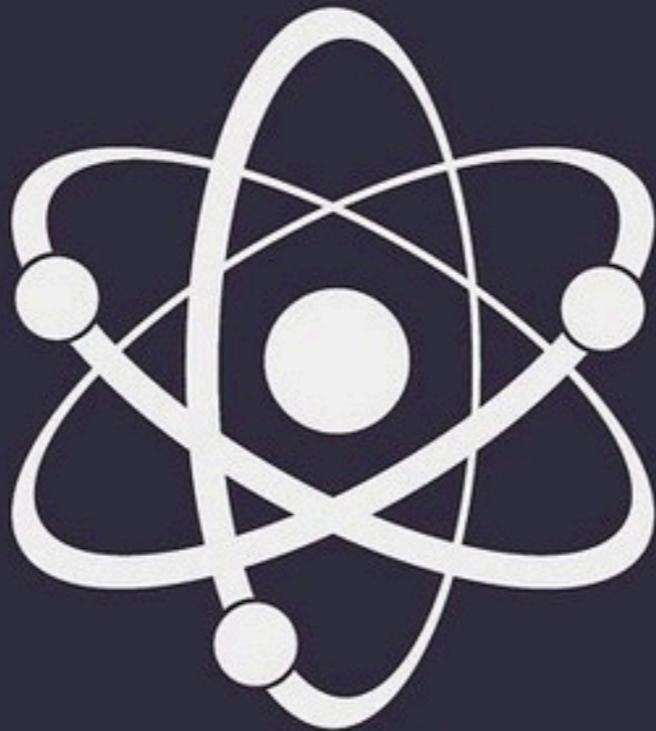
- Each architecture in general has associated **different training and regularisation strategies**: no fit-for-all methods available!



# **Tutorial #3:**

# **regression with NNs**

# NEVER TRUST AN ATOM

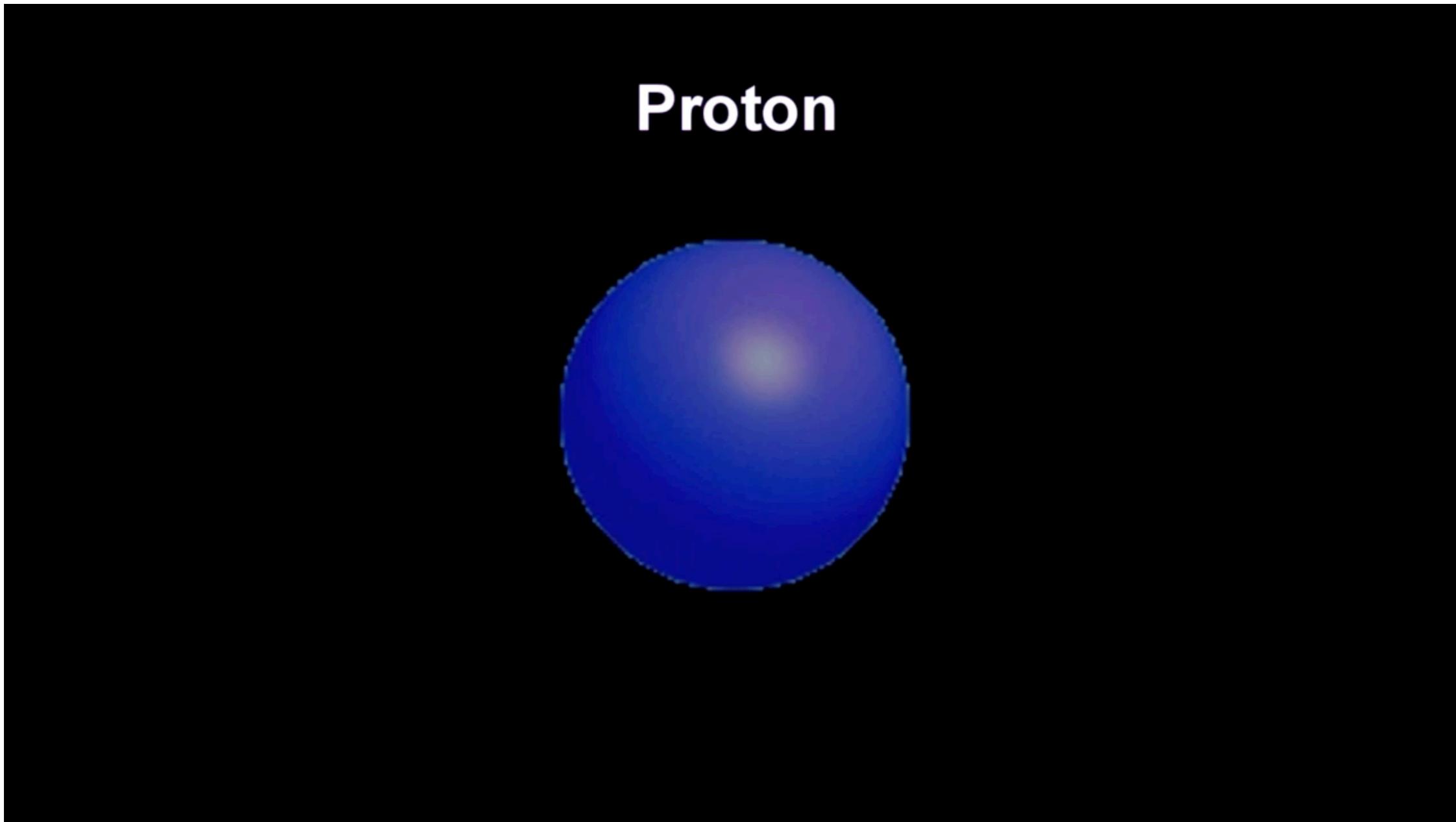


## THEY MAKE UP EVERYTHING

**protons** compose > 99% of all visible mass in Universe, yet there remain **fundamental open questions** about them!

# Why proton structure?

**proton: QCD bound state of quarks and gluons**

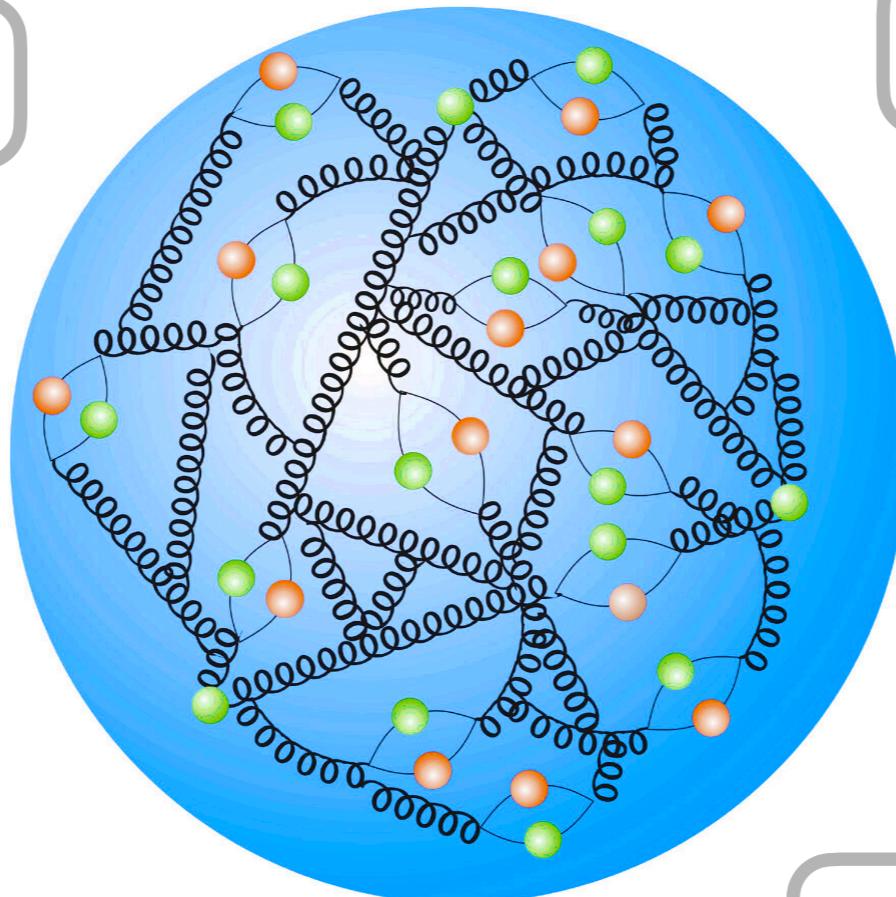


Quantum Mechanics: what is a proton? **It depends on how you look at it!** (e.g. energy/resolution scales)

# Why proton structure?

**proton: QCD bound state of quarks and gluons**

***Origin of mass?***



***Origin of spin?***

***Gluon-dominated matter?***

***Heavy quark content?***

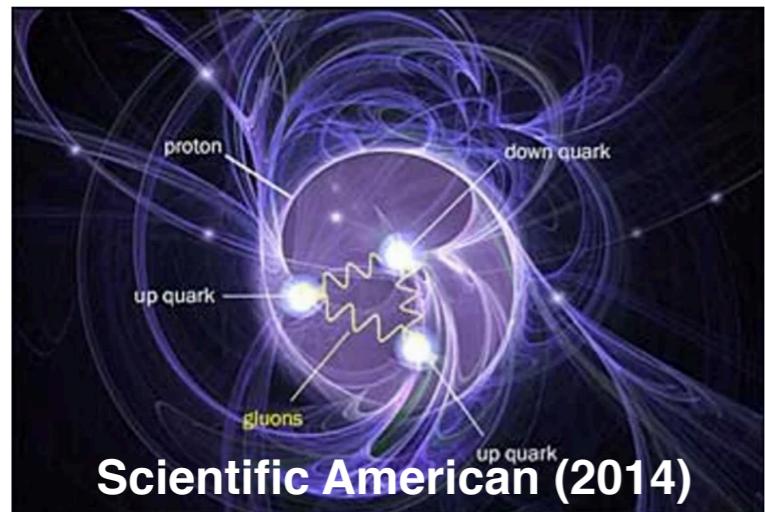
***3D imaging?***

***Nuclear modifications?***

# The proton in the spotlight

THE SCIENCES

## Proton Spin Mystery Gains a New Clue



Scientific American (2014)

### *Non-zero gluon polarisation*

The proton keeps surprising us as an endless source of **fundamental discoveries**

NEWS PARTICLE PHYSICS

## The inside of a proton endures more pressure than anything else we've seen

For the first time, scientists used experimental data to estimate the pressure inside a proton  
BY EMILY CONOVER 1:18PM, MAY 16, 2018



### *Nucleon pressure*

QUANTUM PHYSICS

## Decades-Long Quest Reveals Details of the Proton's Inner Antimatter

27

Twenty years ago, physicists set out to investigate a mysterious asymmetry in the proton's interior. Their results, published today, show how antimatter helps stabilize every atom's core.

### *Antimatter asymmetry @ SeaQuest*



Quanta Magazine (2021)

After 40 years of studying the strong nuclear force, a revelation

### *BFKL dynamics*

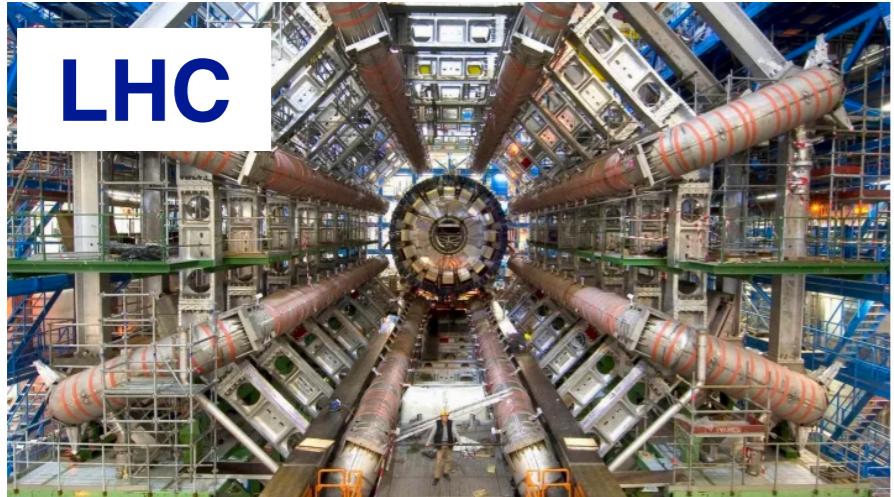
This was the year that analysis of data finally backed up a prediction, made in the mid 1970s, of a surprising emergent behaviour in the strong nuclear force



The Guardian (2017)

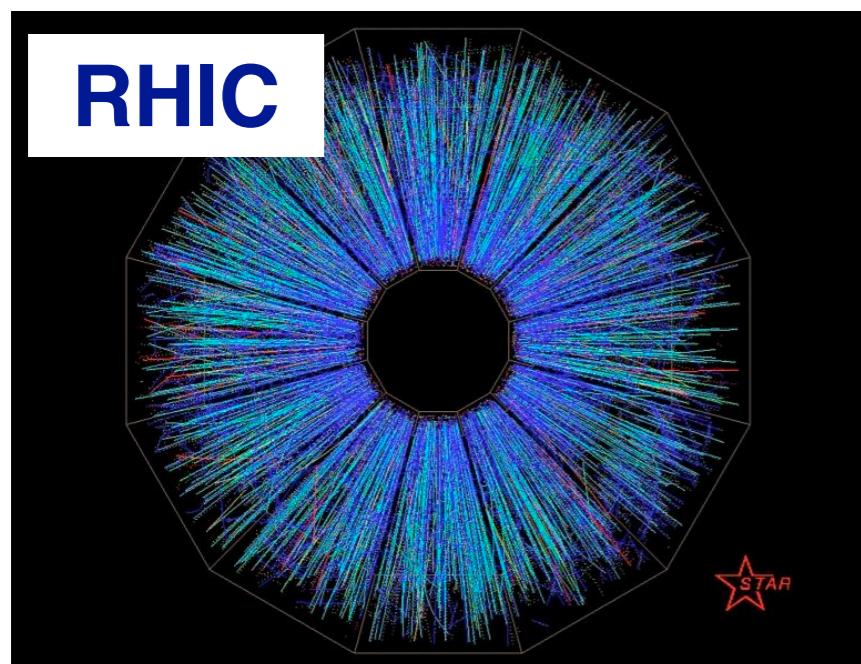
# What about proton structure?

knowledge of quark and gluon substructure of protons also essential for:



New elementary particles  
beyond the Standard Model?

Origins and properties of  
cosmic neutrinos?



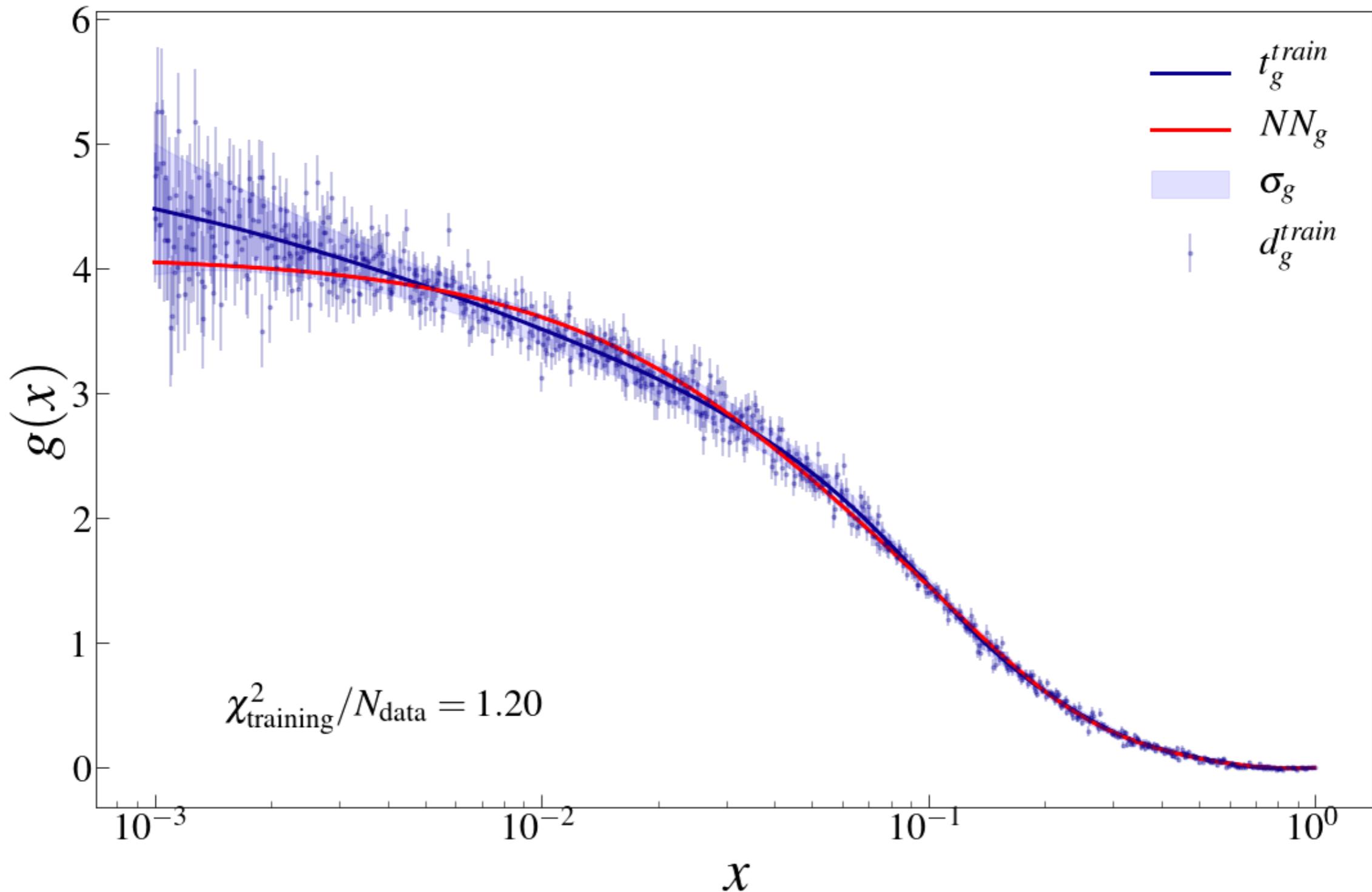
Nature of Quark-Gluon Plasma  
in heavy-ion collisions?

# Tutorial 3: NN training with TensorFlow

- Introduction to **non-linear regression problems** using neural networks.
- You will learn how to **build and train neural networks** with TensorFlow, a powerful machine learning library.
- The goal will be to extract a certain physical property of protons, the distribution of momentum carried by their gluons, **from simulated pseudo-data** based on some known underlying truth
- The neural nets should then aim to **learn this known underlying truth** from the data
- Assess/avoid overfitting** and how to **estimate the model uncertainties**, e.g.  
What happens when different sets of NN parameters give an equally good description of the *experiment* data?

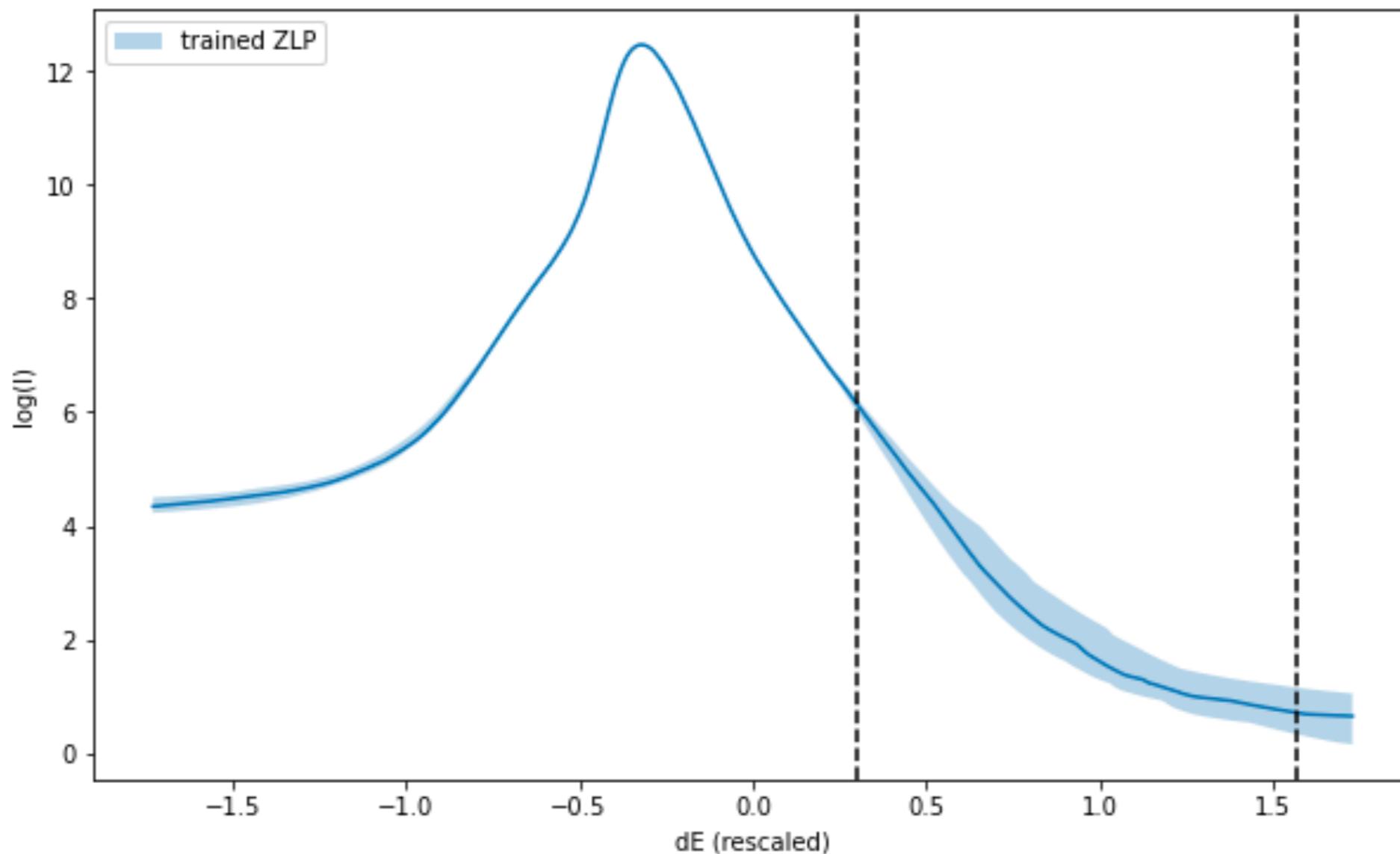
*NB this procedure is known as ``closure test'',  
since by construction the ``truth'' underlying law is  
know (not the case in realistic applications)*

# Tutorial 3: NN training with TensorFlow



*your first ever neural network training!*

# Tutorial 3(extra): NN training with PyTorch



- In the repo you will also find a notebook for NN regression now in the **PyTorch** framework.
- Here the goal is to model the dominant source of background, known as the **Zero-Loss-Peak** which arise in **Electron Energy Loss Spectroscopy** measurements
- If you are particularly brave, try to replace PyTorch by TensorFlow and check results are unchanged!

# Summary and outlook

- 💡 Neural networks are at the core of modern machine learning methods
- 💡 Today we discussed one of the most representative architectures, **deep feed-forward neural networks**
- 💡 Training of models with a very large number of free parameters can be efficiently carried out with **stochastic gradient descent combined with backpropagation**
- 💡 Crucial is to **avoid overfitting**, which will reduce the productivity performance of your NN model
- 💡 We will see many more NN architectures in the rest of this course!