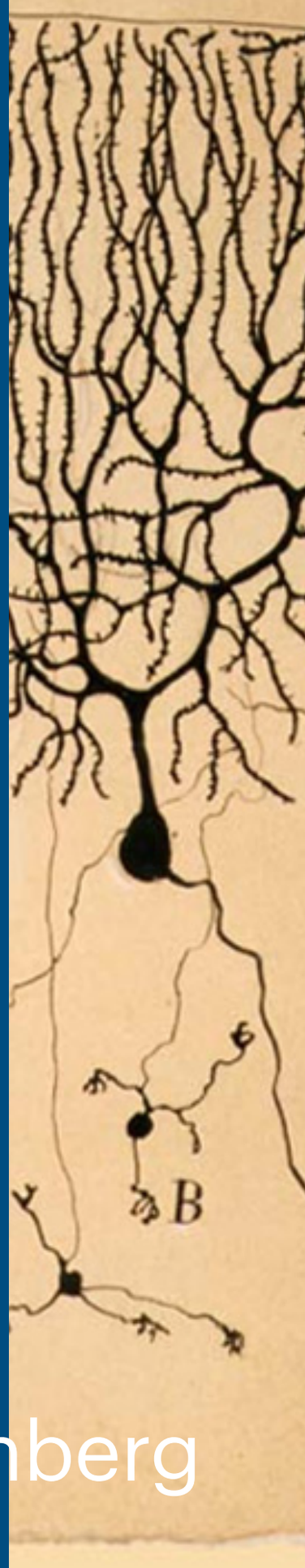
Machine Learning in Three Easy Lessons

...with examples related to physics

Florian Marquardt

Max Planck Institute for the Science of Light
Friedrich-Alexander-Universität Erlangen-Nürnberg

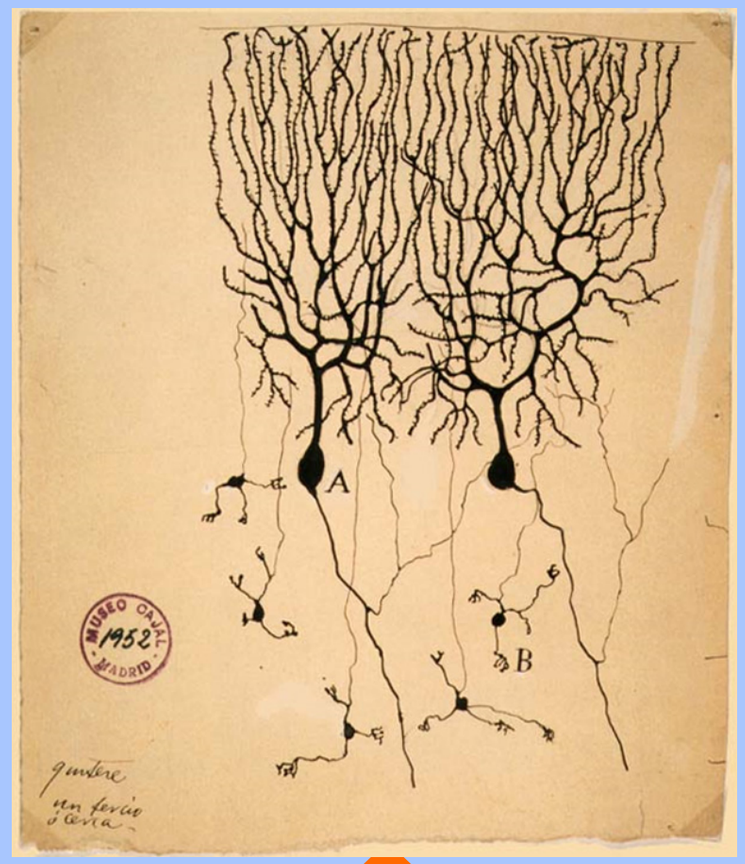


(Pictures: Wikimedia Commons)

(Picture: Wikimedia Commons/Unipro)

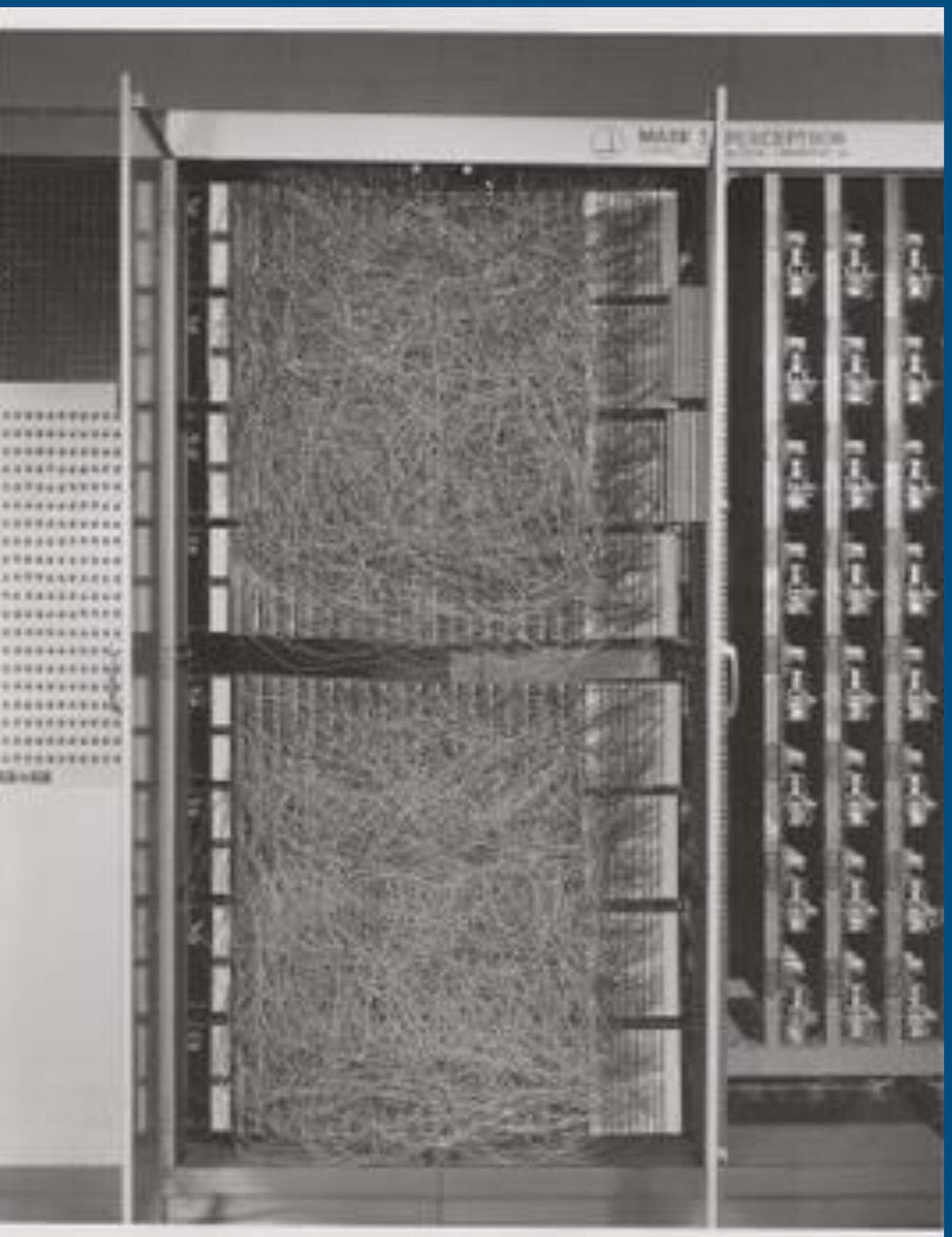
Neural Networks

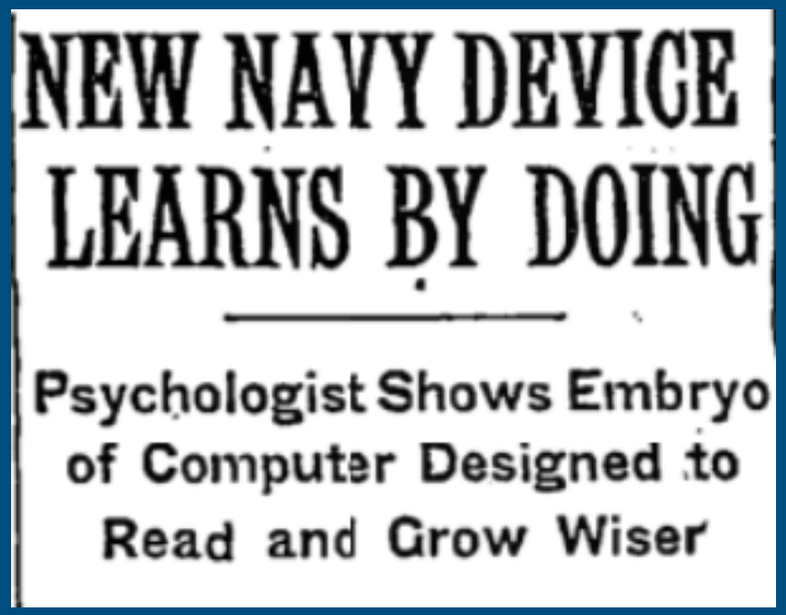
“light bulb”



(Pictures: Wikimedia Commons)

The Perceptron
Rosenblatt 1958





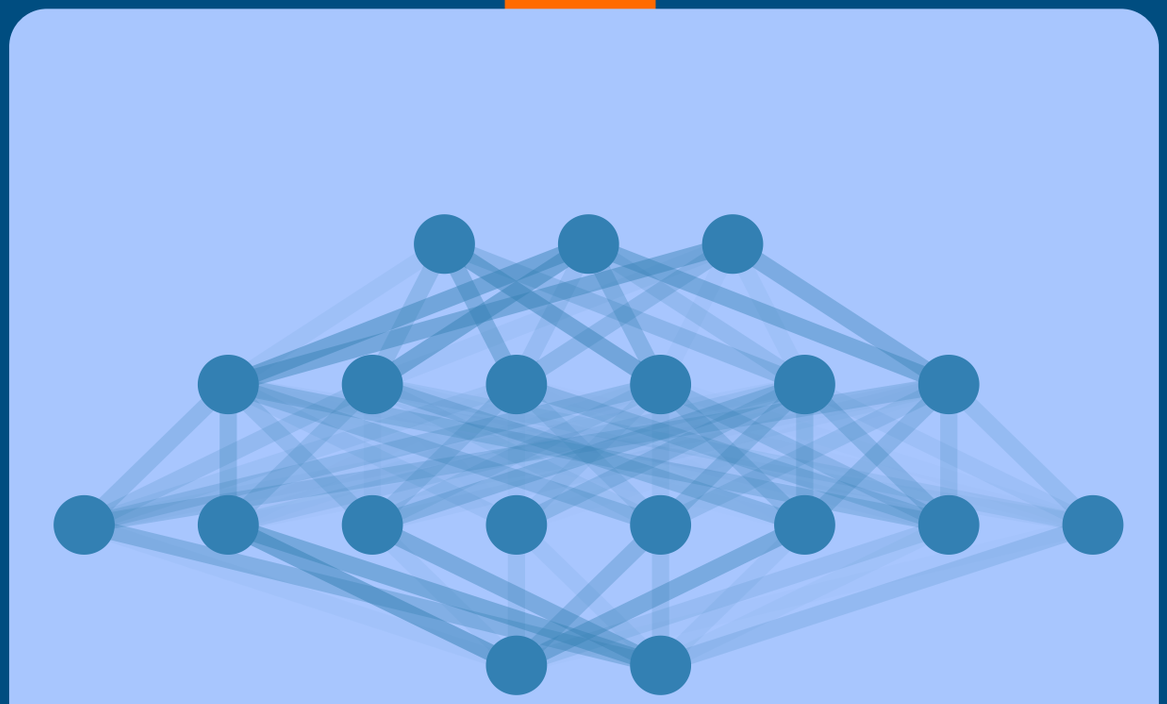
NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

"Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted."
(New York Times 1958)

(Pictures: Wikimedia Commons)

“light bulb”



Artificial Neural Network

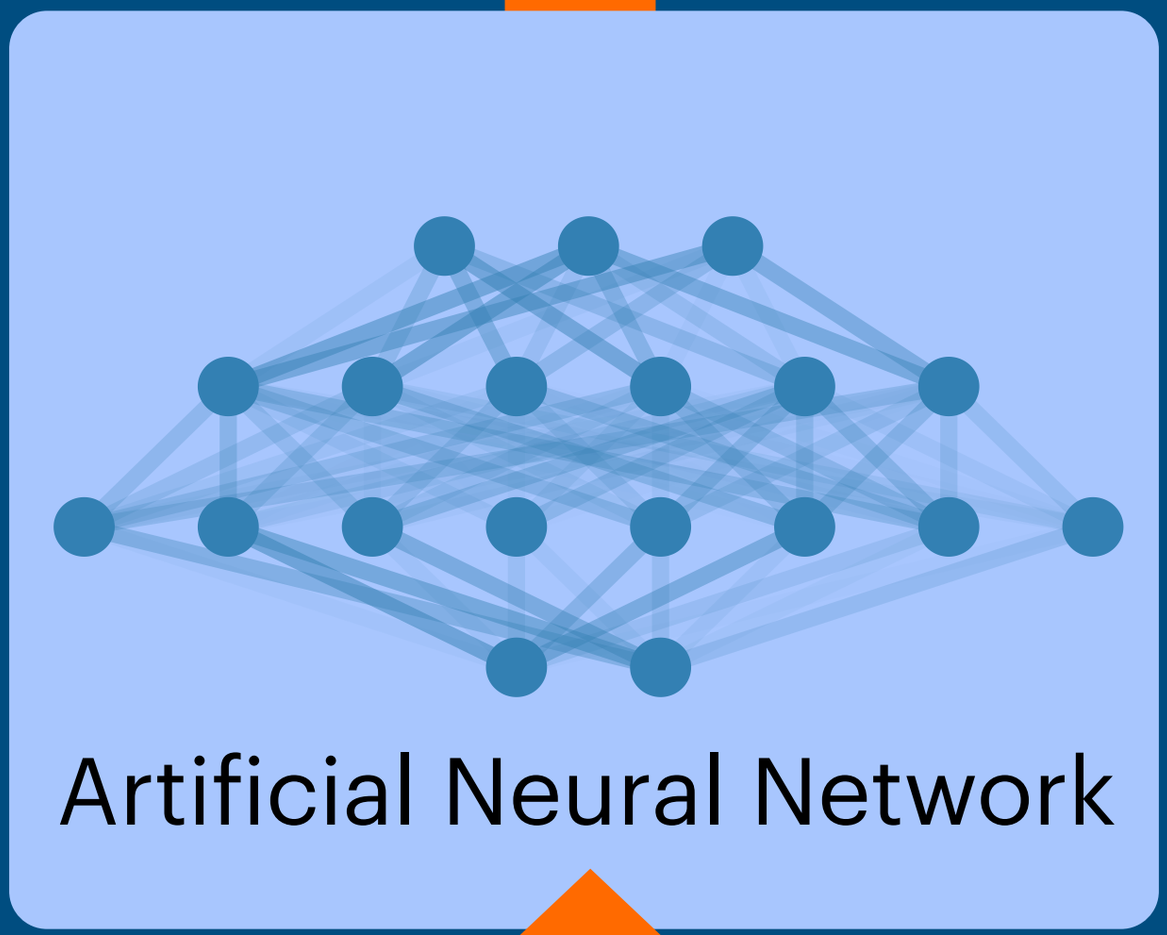
(Pictures: Wikimedia Commons)

2012: A deep neural network beats all other approaches in image recognition

Pictures: image-net.org

...by now: ChatGPT, DALL-E, AlphaFold, ...

"physical parameters: damping=0.5, nonlinearity=0.1,..."



experimental measurements

These lectures: Machine Learning in Three Easy Lessons

close to physics

with hands-on tutorials (no installation required)
prior knowledge: python and numpy

My goal

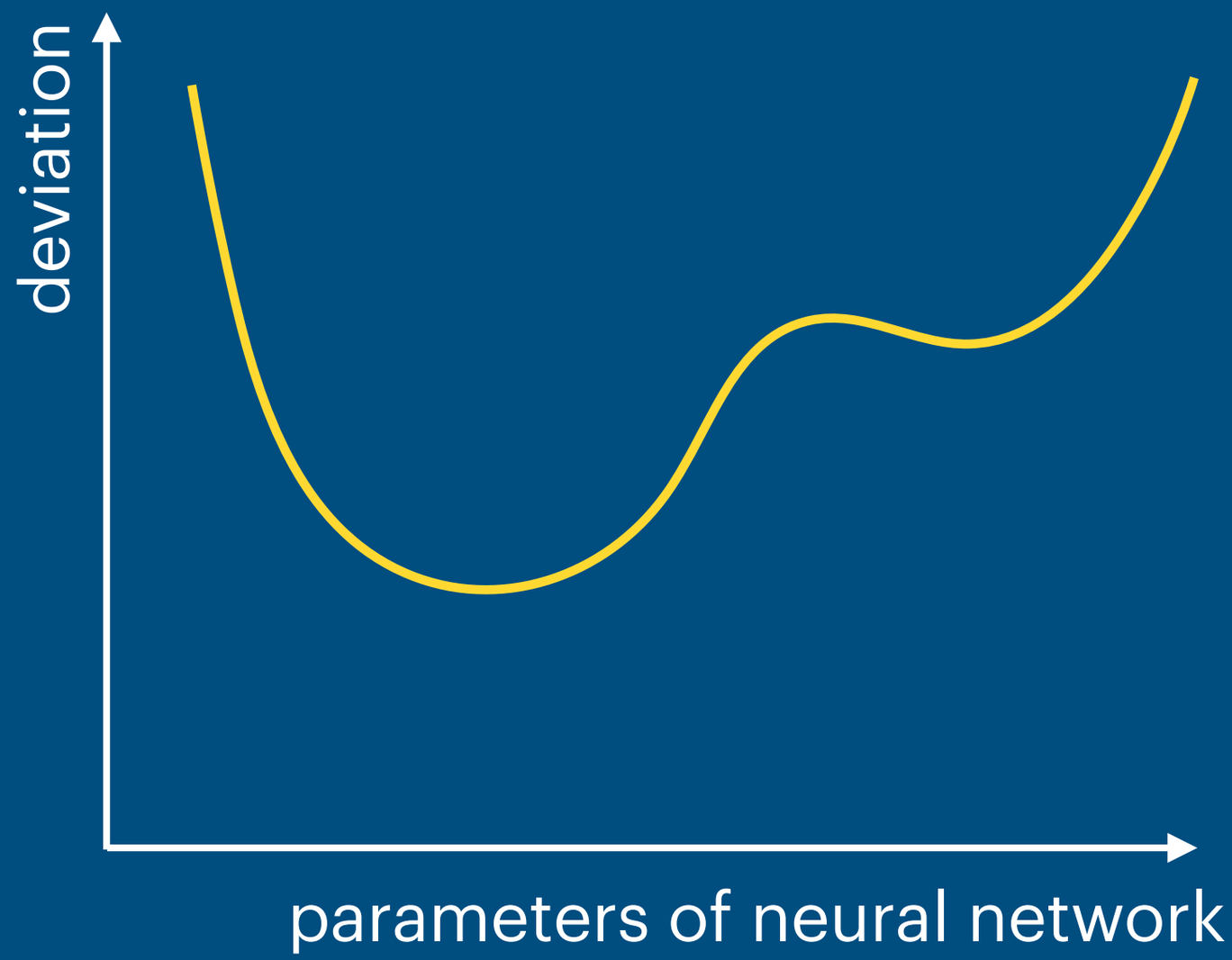
Everyone here will be able to apply neural networks to some research problem of their own (for example by starting from the code provided)

1

Optimization

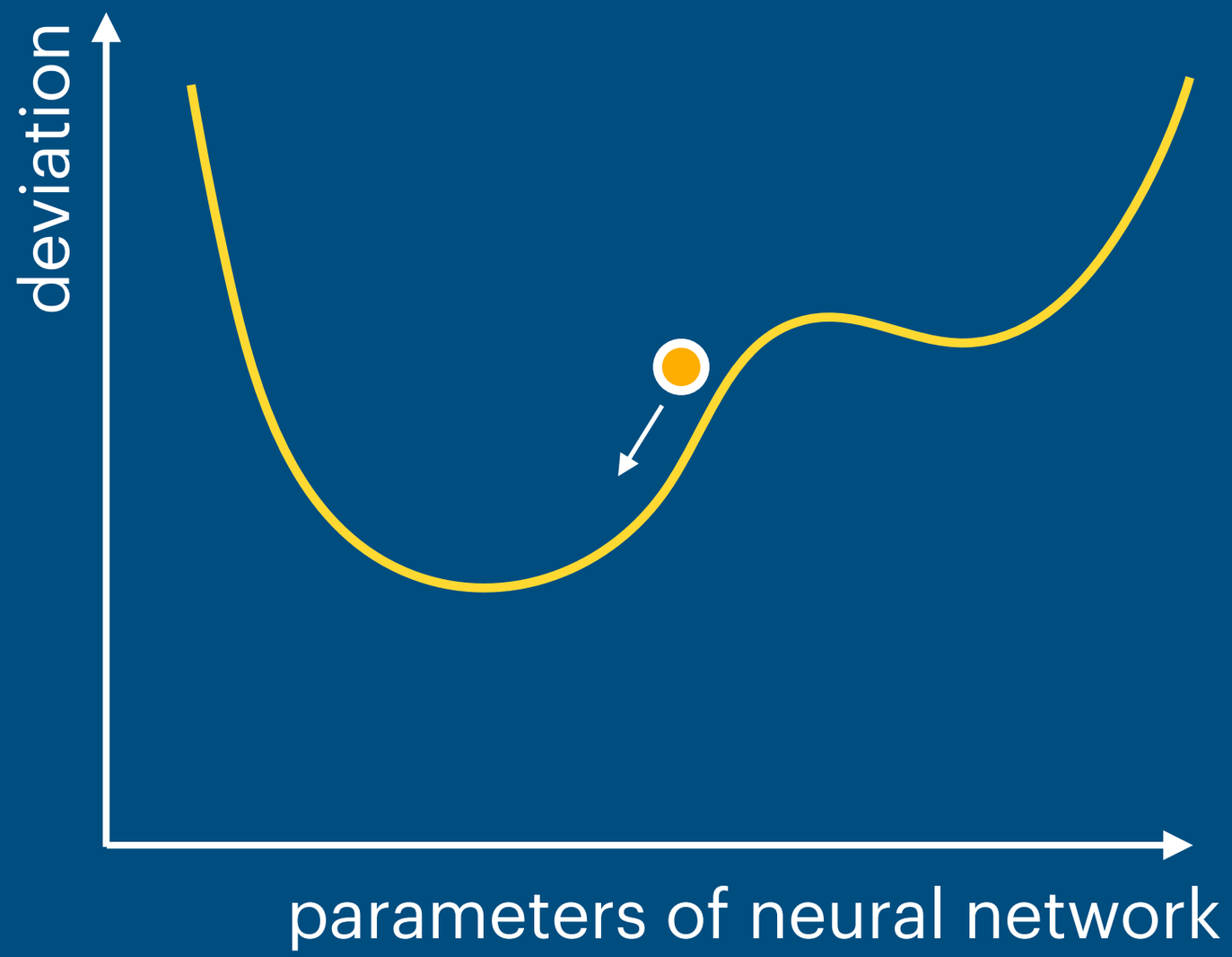
Machine Learning = Optimization

Minimize the average deviation between the desired output of a neural network and its actual output !



Optimization via Gradient Descent

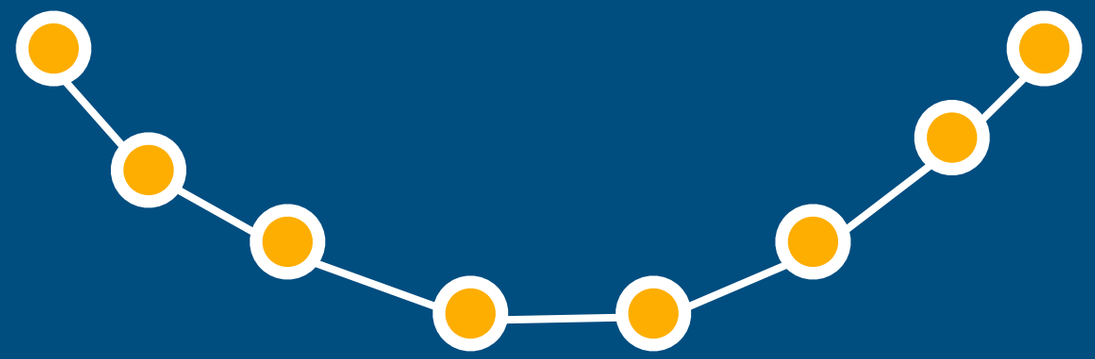
Walk down the hill
(direction of steepest descent = negative gradient)



Physics Example

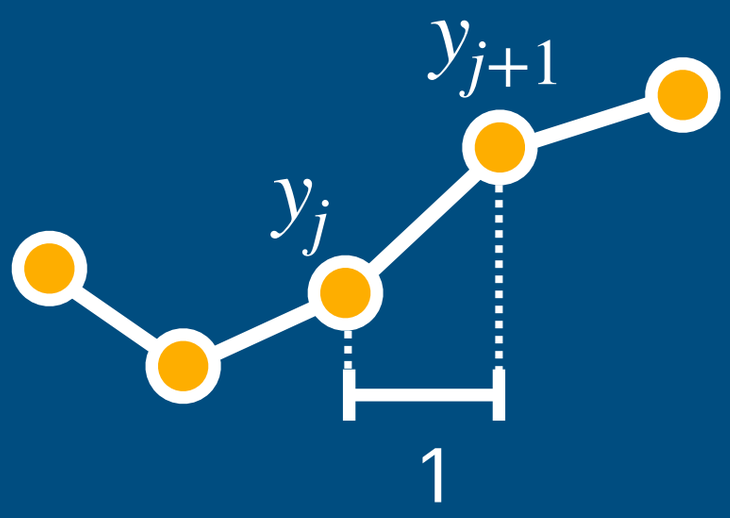
Optimization via Gradient Descent

Physics example: minimize the energy of a mechanical structure to find its equilibrium shape!



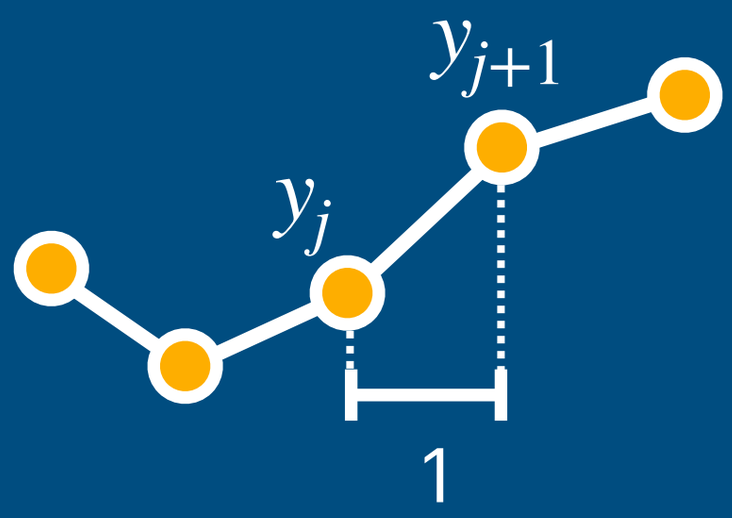
Here: chain of masses, connected via springs, under gravity

A chain of masses with springs



Spring energy: $$\sum_{j=0}^{N-1} \frac{K}{2} \left(\sqrt{(y_{j+1} - y_j)^2 + 1} - l \right)^2$$

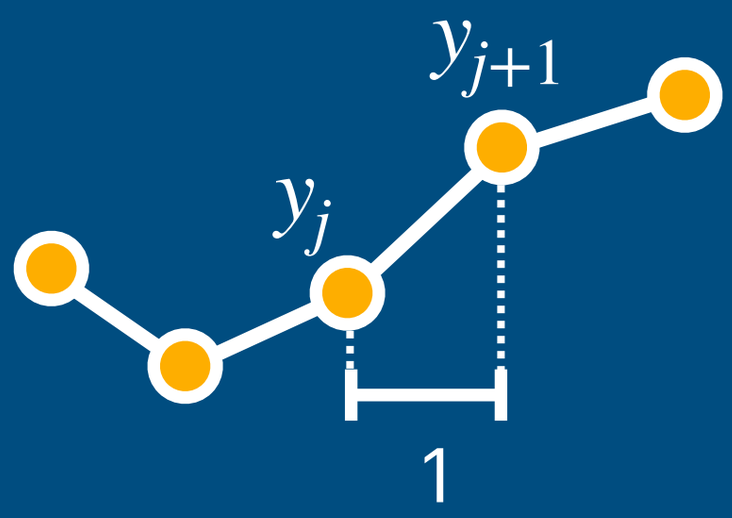
A chain of masses with springs



Spring energy: $$\sum_{j=0}^{N-1} \frac{K}{2} \left(\sqrt{(y_{j+1} - y_j)^2 + 1} - l \right)^2$$

numpy: `(y[1:]-y[:-1])**2`

A chain of masses with springs



Spring energy: $$\sum_{j=0}^{N-1} \frac{K}{2} \left(\sqrt{(y_{j+1} - y_j)^2 + 1} - l \right)^2$$

numpy:

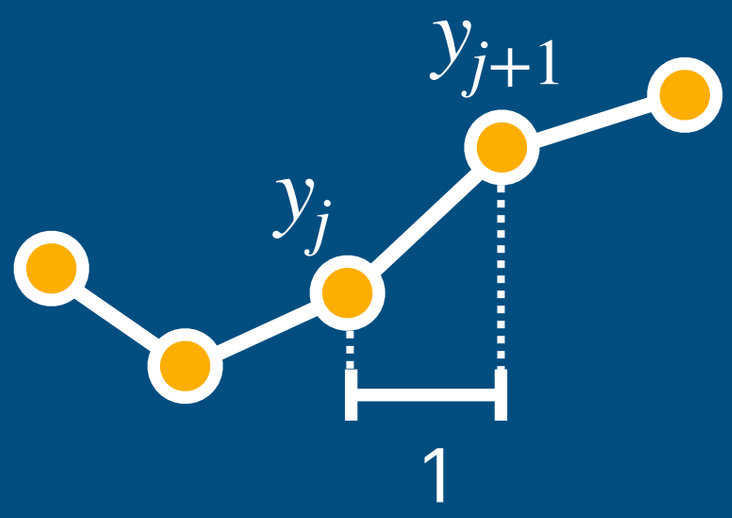
```
(y[1:]-y[:-1])**2
```

```
y = [8,7,9,5]
```

```
y[1:] = [7,9,5]
```

```
y[:-1] = [8,7,9]
```

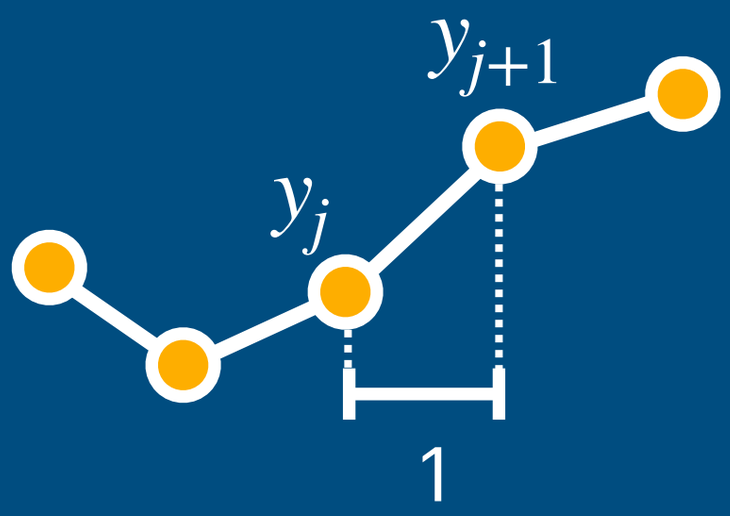
A chain of masses with springs



Spring energy: $$\sum_{j=0}^{N-1} \frac{K}{2} \left(\sqrt{(y_{j+1} - y_j)^2 + 1} - l \right)^2$$

```
np.sum((np.sqrt( (y[1:]-y[:-1])**2 + 1 )-length)**2)
```

A chain of masses with springs




```
def E(y, spring_constant, length):
    … calculate energy …
    return energy
```

…can also add gravity mgy_j

Minimize energy: Gradient descent

$$y_j^{\text{new}} = y_j^{\text{old}} - \eta \frac{\partial E}{\partial y_j}$$

step size
(later: "learning rate")

Minimize energy: Gradient descent

How to get the gradient?

Option 1: Calculate by hand and write function grad_E

Option 2: Use "automatic differentiation" library

Interlude: jax

Automatic Differentiation: **jax**

Take an arbitrary function and produce another function that calculates its gradient.

```
def f( x ):
    return jnp.sum( x**2 )
```

$$f(x) = \sum_j x_j^2$$

```
def grad_f( x ):
    return 2*x
```

$$\frac{\partial f}{\partial x_k} = 2x_k$$

Automatic Differentiation: **jax**

Take an arbitrary function and produce another function that calculates its gradient.

```
def f( x ):
    return jnp.sum( x**2 )
```

$$f(x) = \sum_j x_j^2$$

```
from jax import grad

grad_f = grad( f )
```

...is equivalent to:

```
def grad_f( x ):
    return 2*x
```

$$\frac{\partial f}{\partial x_k} = 2x_k$$

Two steps to use jax

(1) Instead of numpy use jax.numpy for everything:

```
import jax.numpy as jnp

y = jnp.sin( x )
```

Avoid for loops when possible, for efficiency.

Use arrays for everything.

No conditionals: replace

```
if x<0:
    y=-x
else:
    y=x**2
```

by

```
y = (x<0)*(-x) + (x>=0)*(x**2)
```

Two steps to use jax

(2) Functional programming:

All program functions must be like mathematical functions

No side effects:
The function does not change its arguments
and does not change or rely on any global variables etc.
and does not print anything

This makes it much easier to understand the flow of information!

jax is simple but state of the art

jax is simple to use
(numpy + grad + handful of other simple but powerful commands)

jax has very good performance

jax is used in state-of-the-art projects, like DeepMind's AlphaFold

main alternatives: tensorflow, pytorch

Back to physics

Minimize energy: Gradient descent

$$y_j^{\text{new}} = y_j^{\text{old}} - \eta \frac{\partial E}{\partial y_j}$$

step size
(later: "learning rate")

Minimize energy: Gradient descent

Produce new function that calculates the gradient of E:

```
grad_E = grad(E, argnum=0)
```

(with respect to its first argument y, indicated by argnum=0)

Now do gradient descent:

```
y = y - eta * grad_E(y, spring, length, …)
```

Let us try it out!

FlorianMarquardt.github.io/MachineLearningThreeEasyLessons

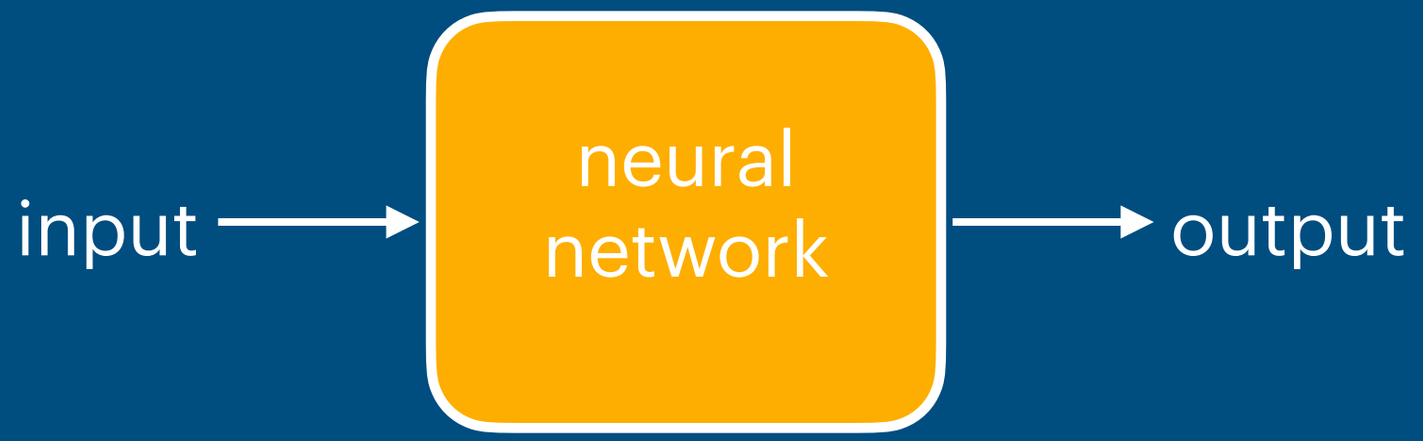
(Lesson 1)

2

Neural Networks

Machine Learning = Optimization

Minimize the average deviation between the desired output of a neural network and its actual output !



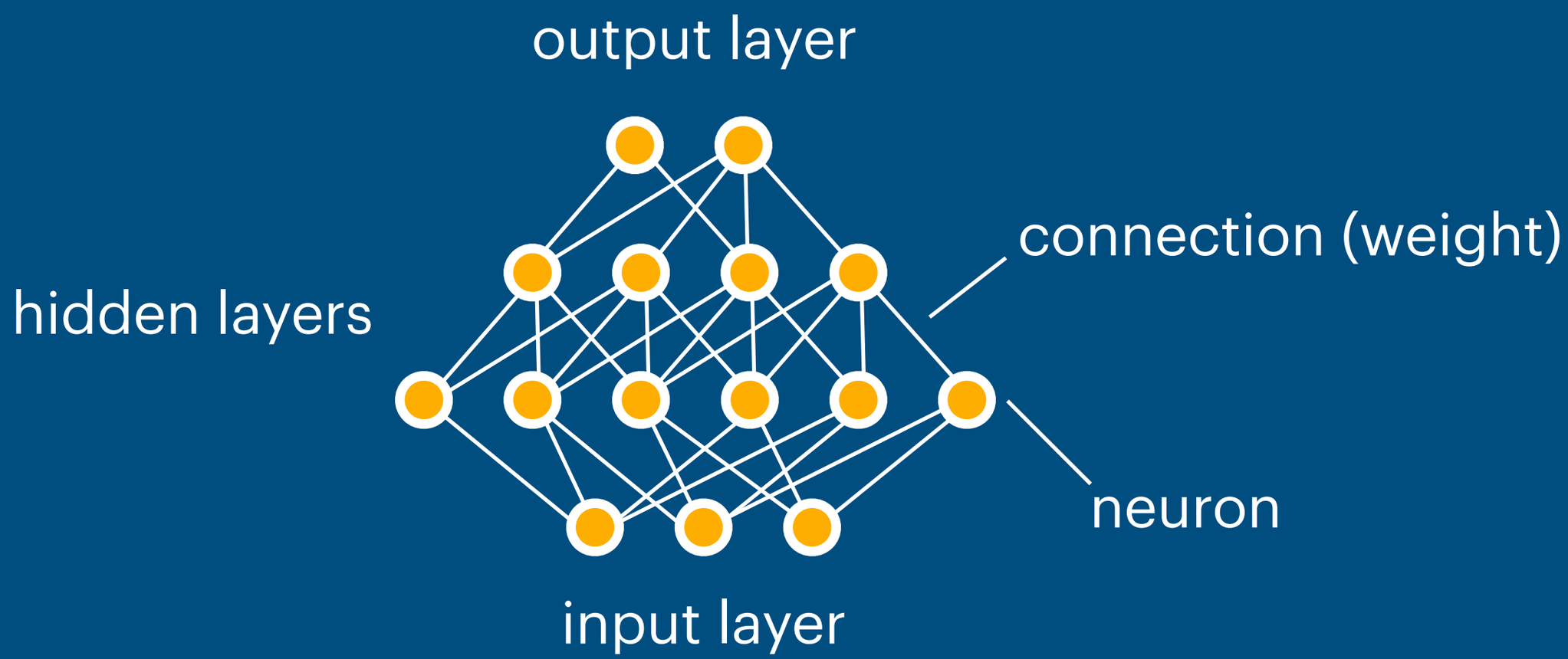
"loss" (or "cost") = average of (target output – actual output)^2

averaged over all possible inputs

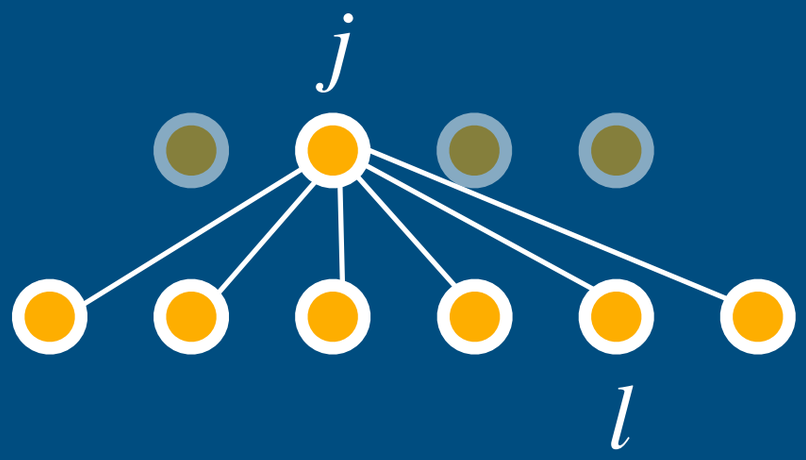
Neural network structure

Artificial neural network = high-dimensional function depending on many parameters

Artificial neural network = cartoon of biological neural network



Neural network evaluation



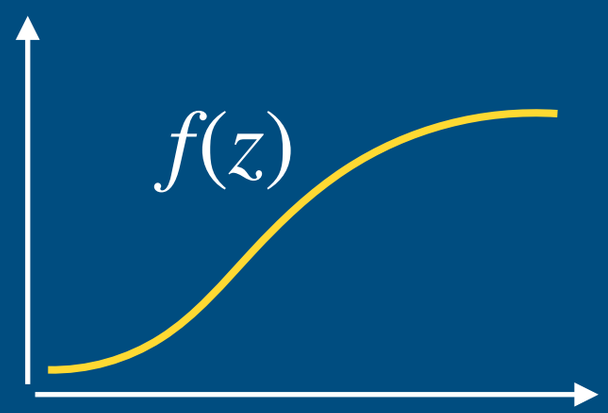
x = vector of neuron activations in lower layer

(1) calculate $z_j = \sum_l w_{jl} x_l + b_j$

weights

biases

nonlinear activation function f:



(2) and then $x_j^{\text{new}} = f(z_j)$

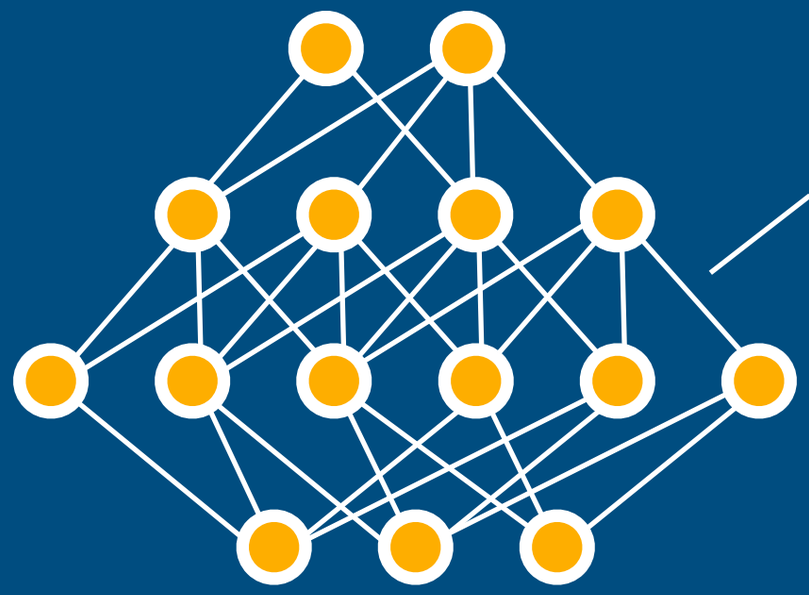
x^{new}= vector of neuron activations in upper layer

Neural network evaluation

layer by layer:

calculate $z_j = \sum_l w_{jl} x_l + b_j$ and then $x_j^{\text{new}} = f(z_j)$

w: weight matrix, b: bias vector (different for each layer)



Neural network evaluation

layer by layer:

calculate $z_j = \sum_l w_{jl} x_l + b_j$ and then $x_j^{\text{new}} = f(z_j)$

w: weight matrix, b: bias vector (different for each layer)

```
x = jnp.matmul( w, x) + b
x = f( x )
```

Neural network evaluation

layer by layer:

calculate $z_j = \sum_l w_{jl} x_l + b_j$ and then $x_j^{\text{new}} = f(z_j)$

w: weight matrix, b: bias vector (different for each layer)

Simplest neural network:

```
def NN( x, weights, biases ):
  for w,b in zip(weights,biases):
     x = jnp.matmul( w, x ) + b
     x = f( x )
  return x
```

weights: list of matrices
biases: list of vectors

Neural network evaluation

Slightly nicer:

```
def NN( x, params ):
   for w,b in zip(params['weights'],params['biases']):
      x = jnp.matmul( w, x ) + b
      x = f( x )
   return x
```

params: a dictionary containing the weights and biases

Loss function

simplest choice: "mean square error"

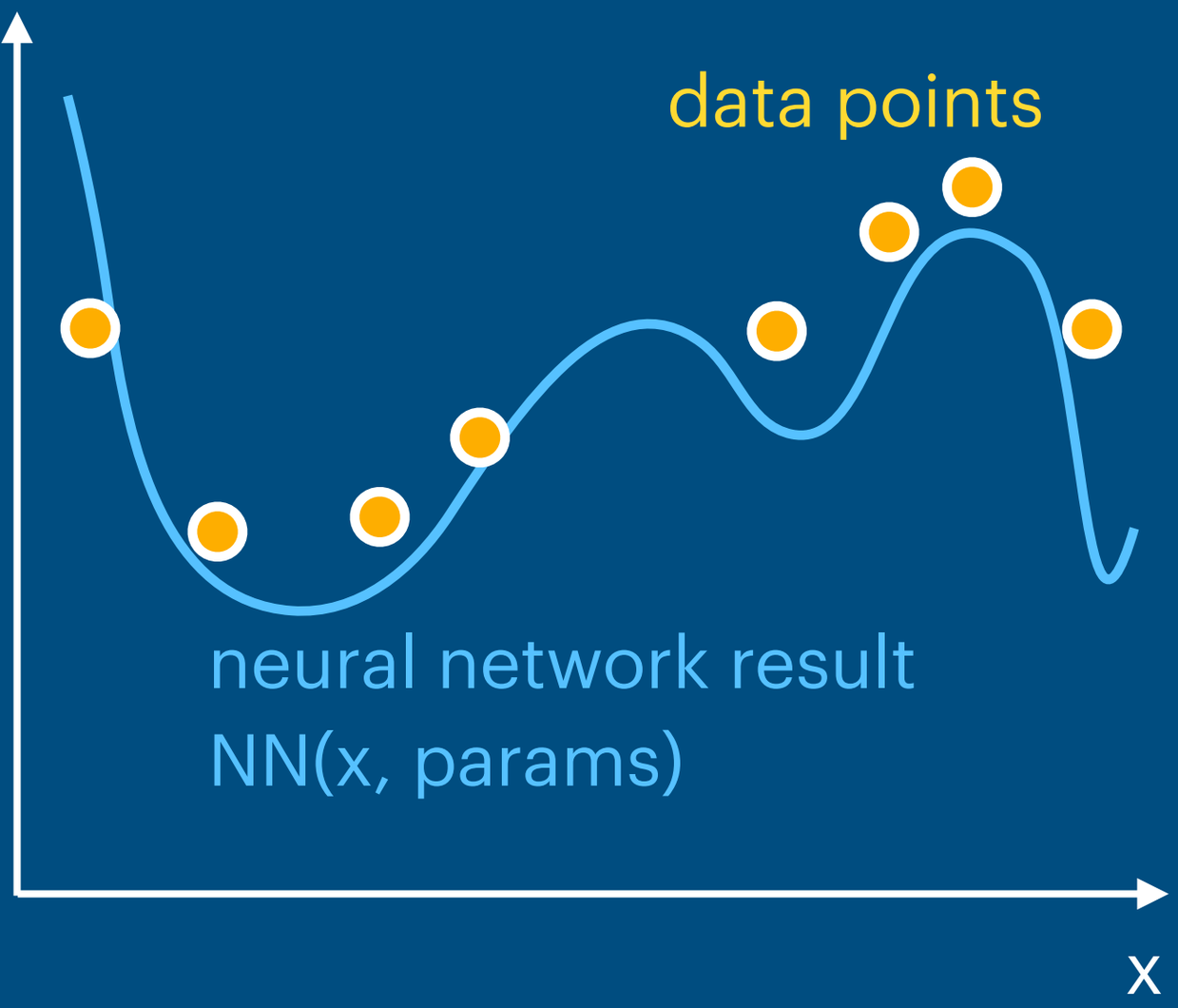
```
def mse_loss( x, y_target, params ):
  return jnp.sum( ( NN( x, params ) - y_target )**2 )
```

Important: this calls the network, and has an explicit dependence on the parameters **`params`**

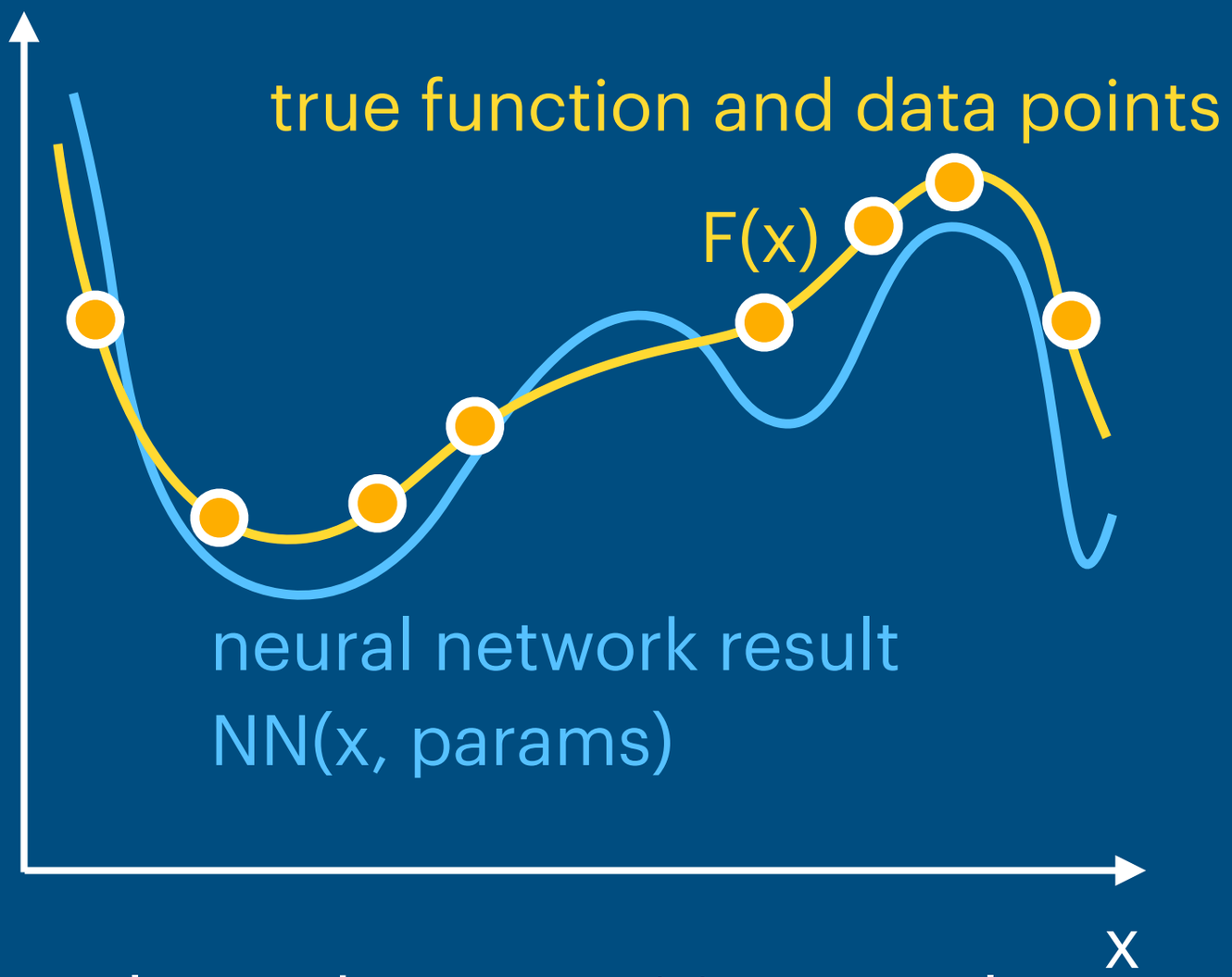
Thus, we can take the gradient with respect to the parameters!

```
mse_loss_grad = grad( mse_loss, argnums=2 )
```

An example: fitting data / a function via a neural network



An example: fitting data / a function via a neural network



In our example: sample random x positions, evaluate true function F(x) to get the data points, and try to minimize mean square error for the network.

Two more things

– jax random numbers (see code)

– batch processing (see code)

processing `x` of shape `[batchsize, input_dim]` in parallel

Let us try it out!

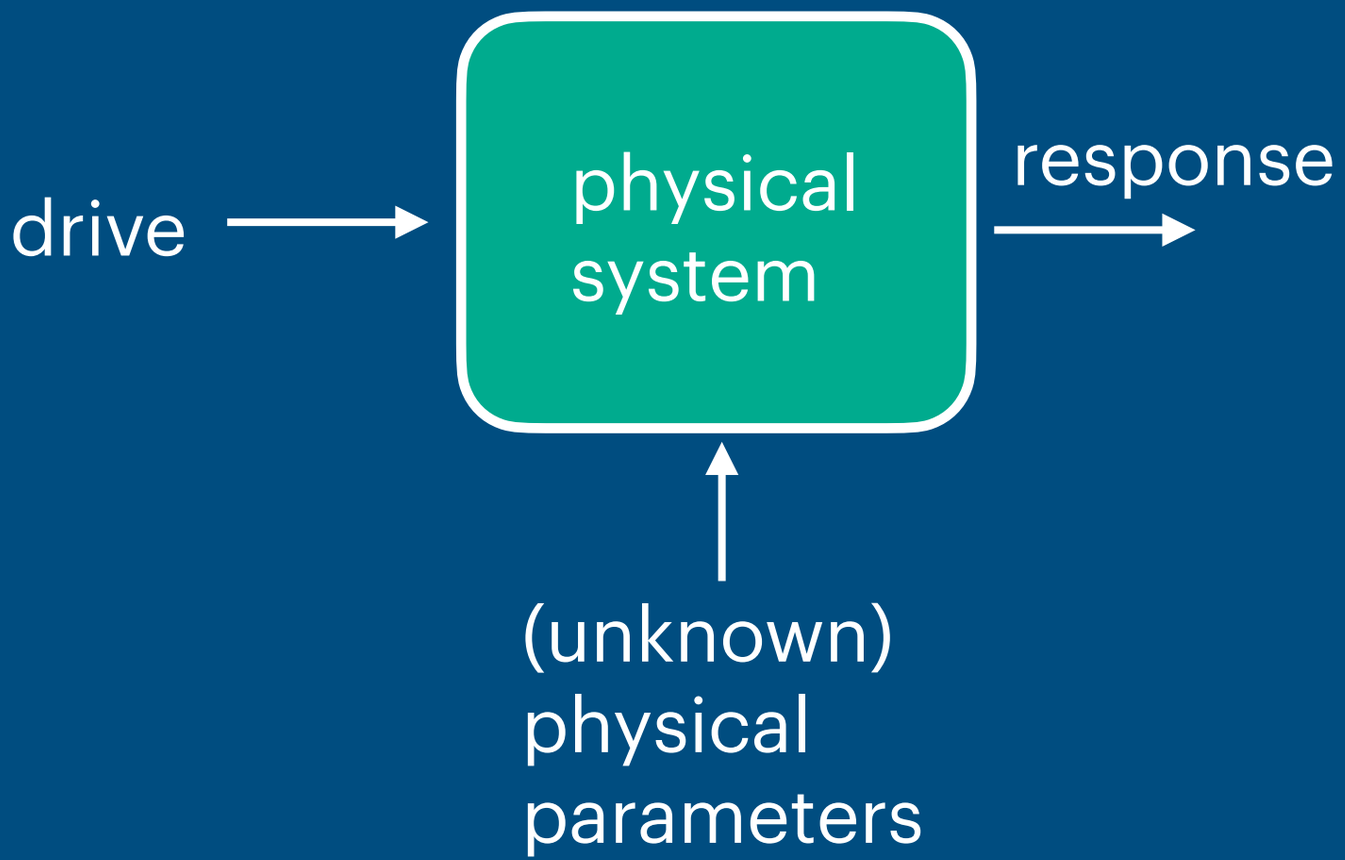
FlorianMarquardt.github.io/MachineLearningThreeEasyLessons

(Lesson 2)

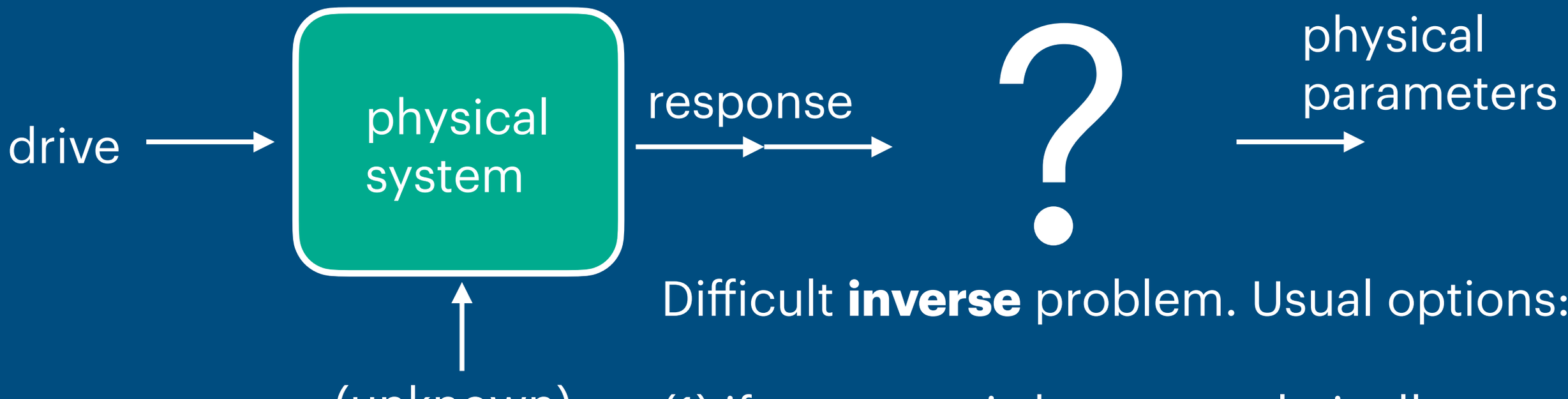
3

Full example: Characterizing a nonlinear oscillator

Physics example: Characterizing a physical system



Physics example: Characterizing a physical system



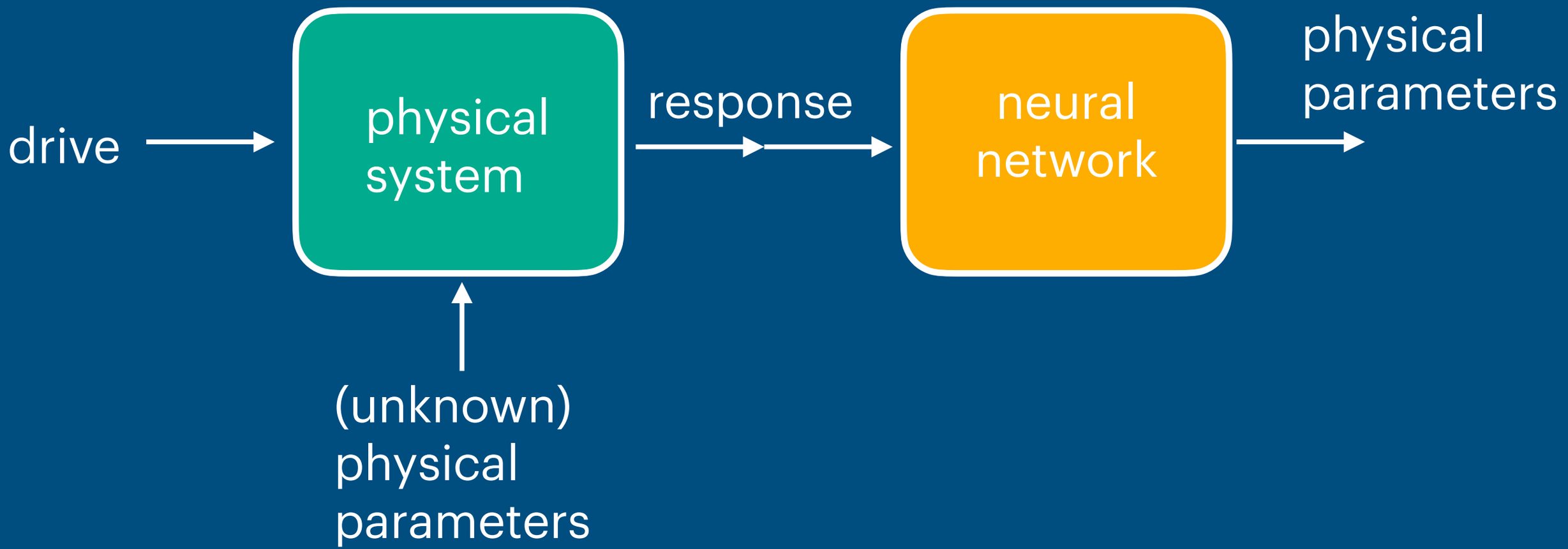
Difficult **inverse** problem. Usual options:

(1) if response is known analytically: nonlinear curve fitting with gradient descent in physical parameters (but: slow, may get stuck)

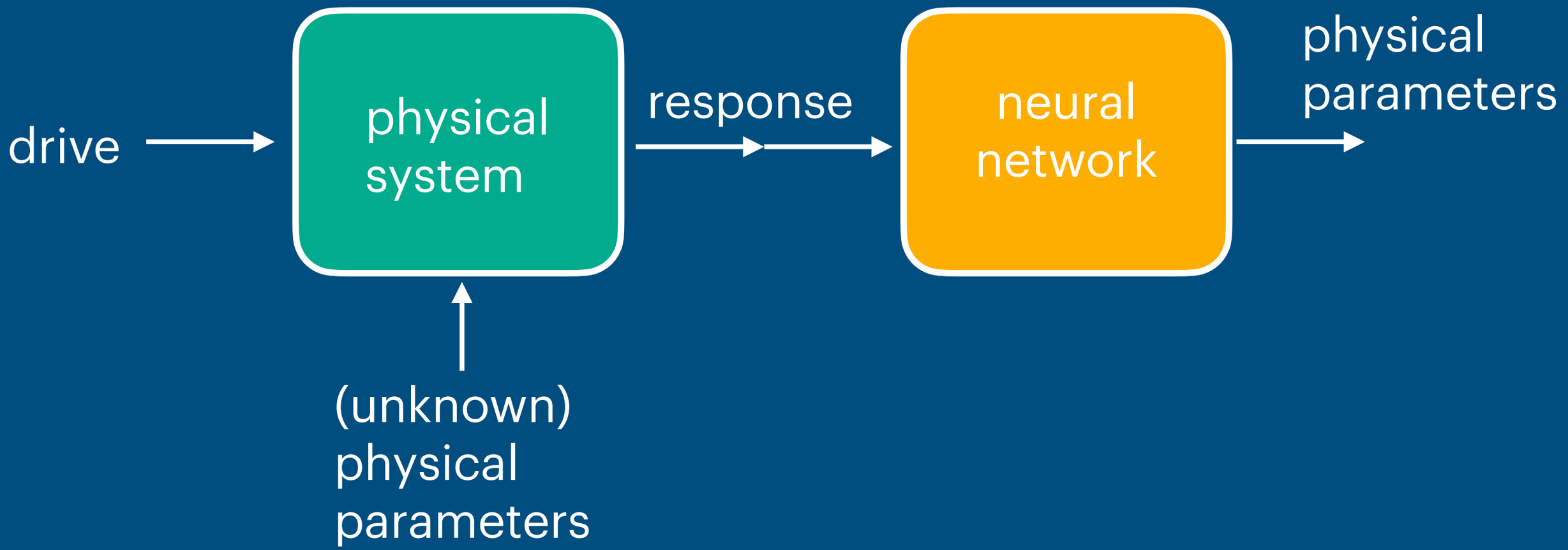
(2) if it is not known analytically, but simulations available: gradient-free optimization of phys. parameters (even slower)

(3) if there are not even simulations: practically impossible

Physics example: Characterizing a physical system

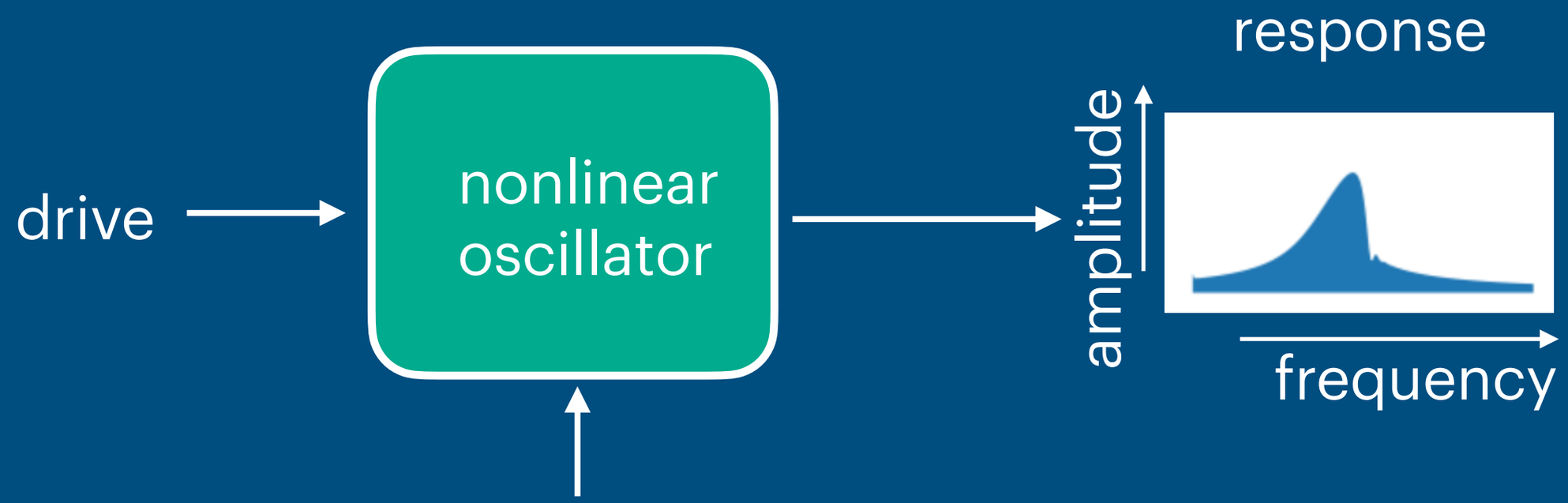


Physics example: Characterizing a physical system



train on many examples where true parameters are known!

Physics example: Characterizing a physical system



Duffing oscillator:

damping

force

$$\dot{z} = -i(\omega_0 - \omega)z - \frac{\gamma}{2}z - i\varepsilon|z|^2 z + if$$

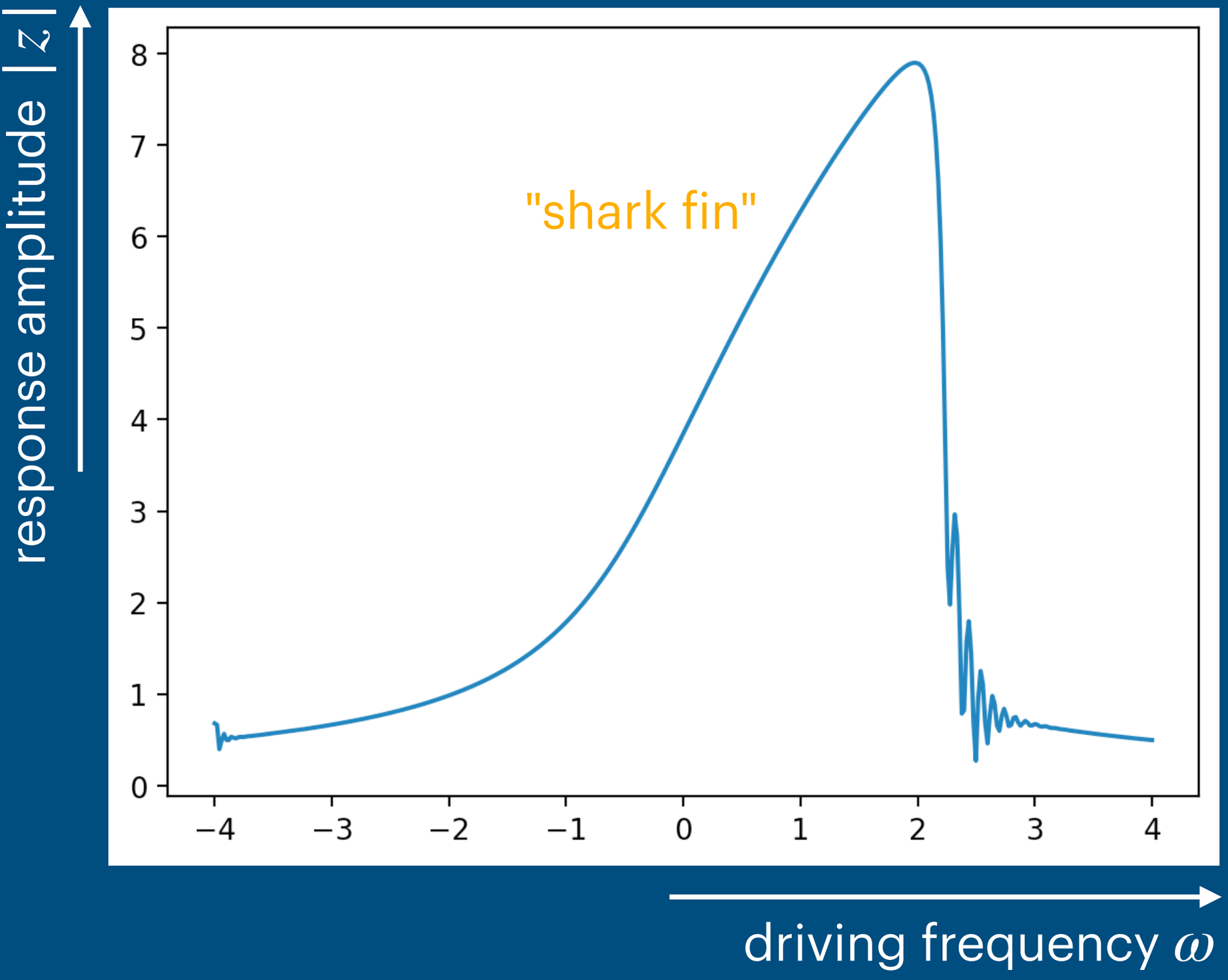
resonance freq.

nonlinearity

complex amplitude $z = q + i\dot{q}$

Physics example: Characterizing a physical system

Response of a Duffing oscillator



here: drive frequency sweep in finite time (leads to wiggles)

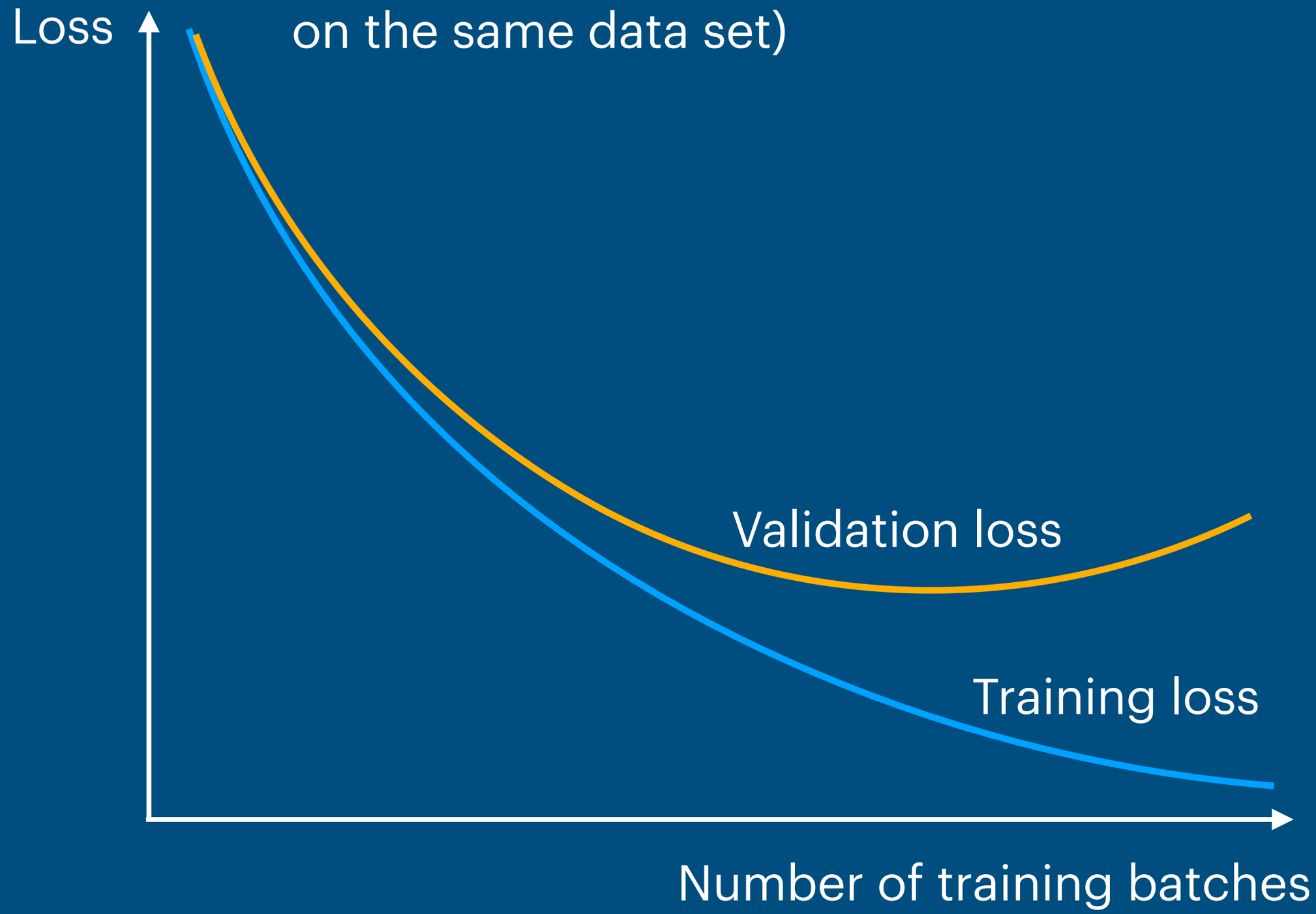
Let us try it out!

FlorianMarquardt.github.io/MachineLearningThreeEasyLessons

(Duffing oscillator)

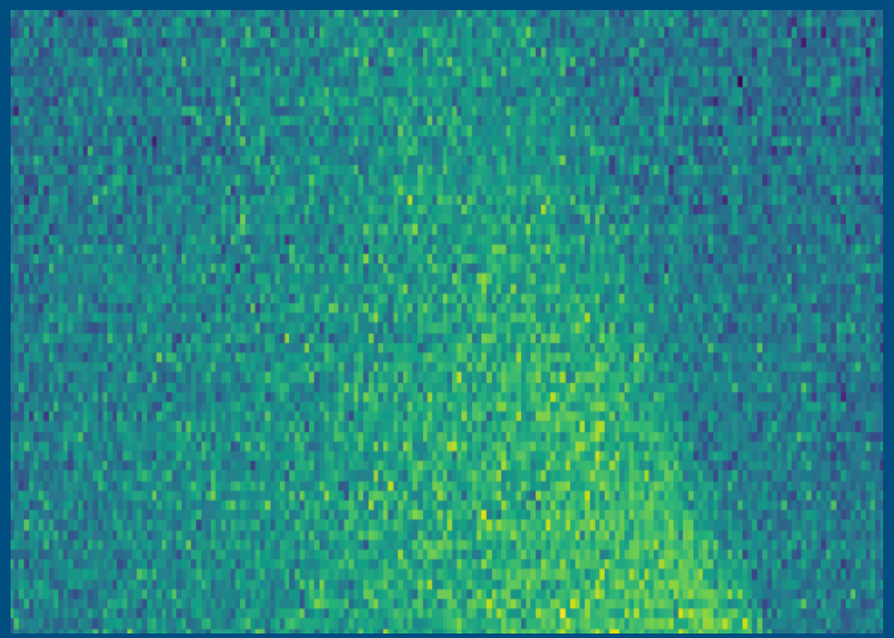
Important Danger: Overfitting

(whenever one repeatedly trains on the same data set)



Images

Dealing with images



neural network

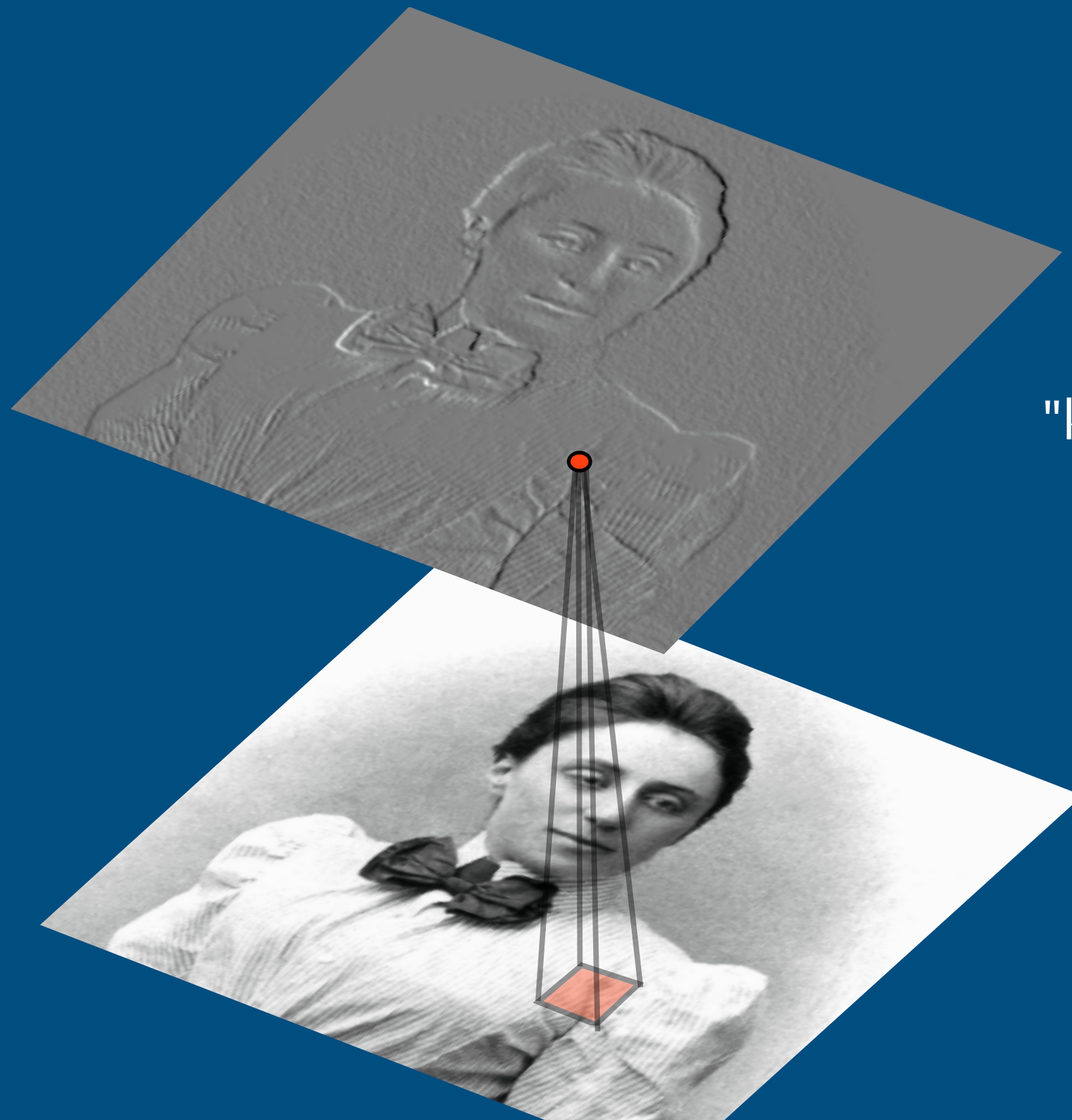
physical parameters:
damping = ...
resonance freq. = ...
nonlinearity = ...

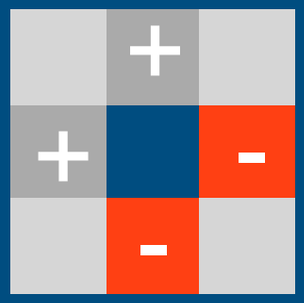
Convolutional Neural Networks for Images

Idea: exploit translational invariance of image features to drastically reduce the number of trainable parameters and speed up training

Convolutional Neural Networks for Images

image filtering: scan some "convolution kernel"/"filter" over source image, multiply pixel values by kernel weights, add together

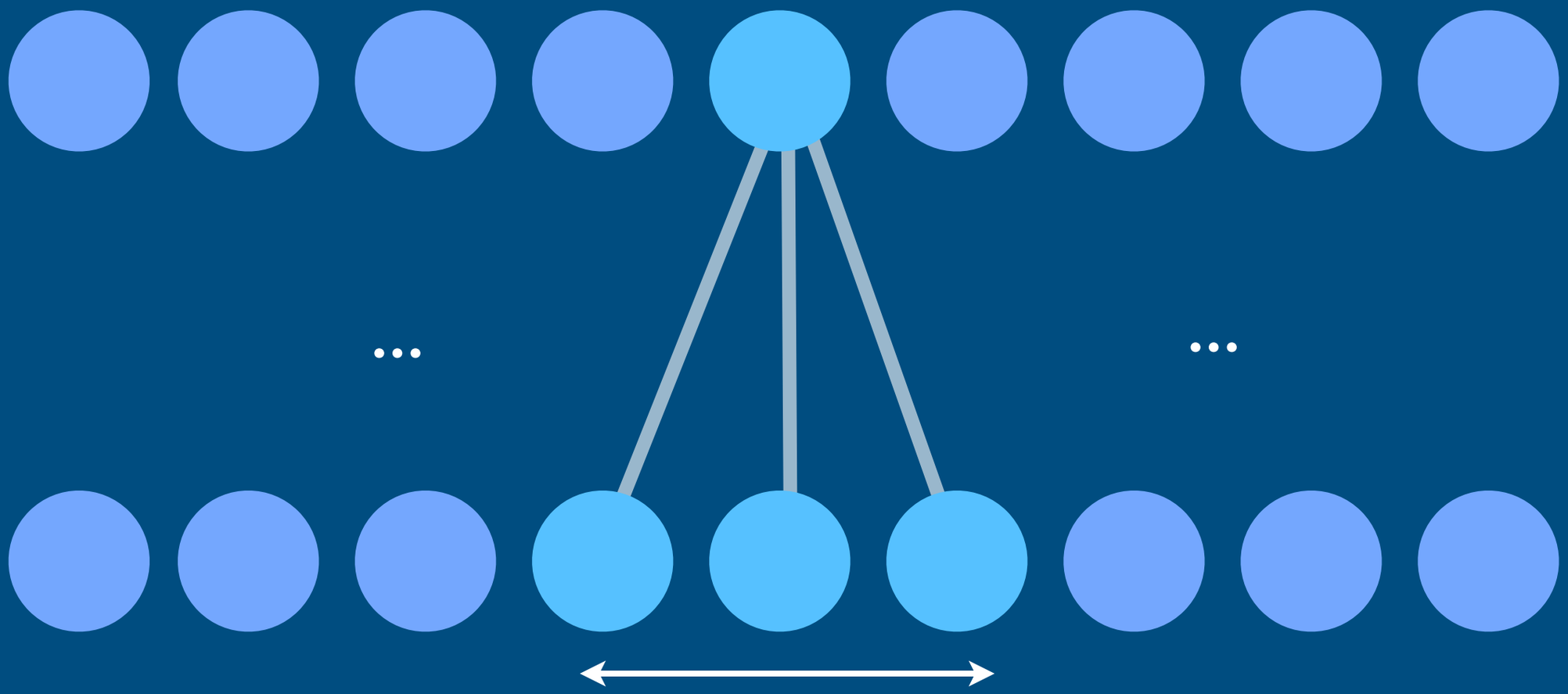




"kernel" (or "filter")

(Picture: Wikipedia)

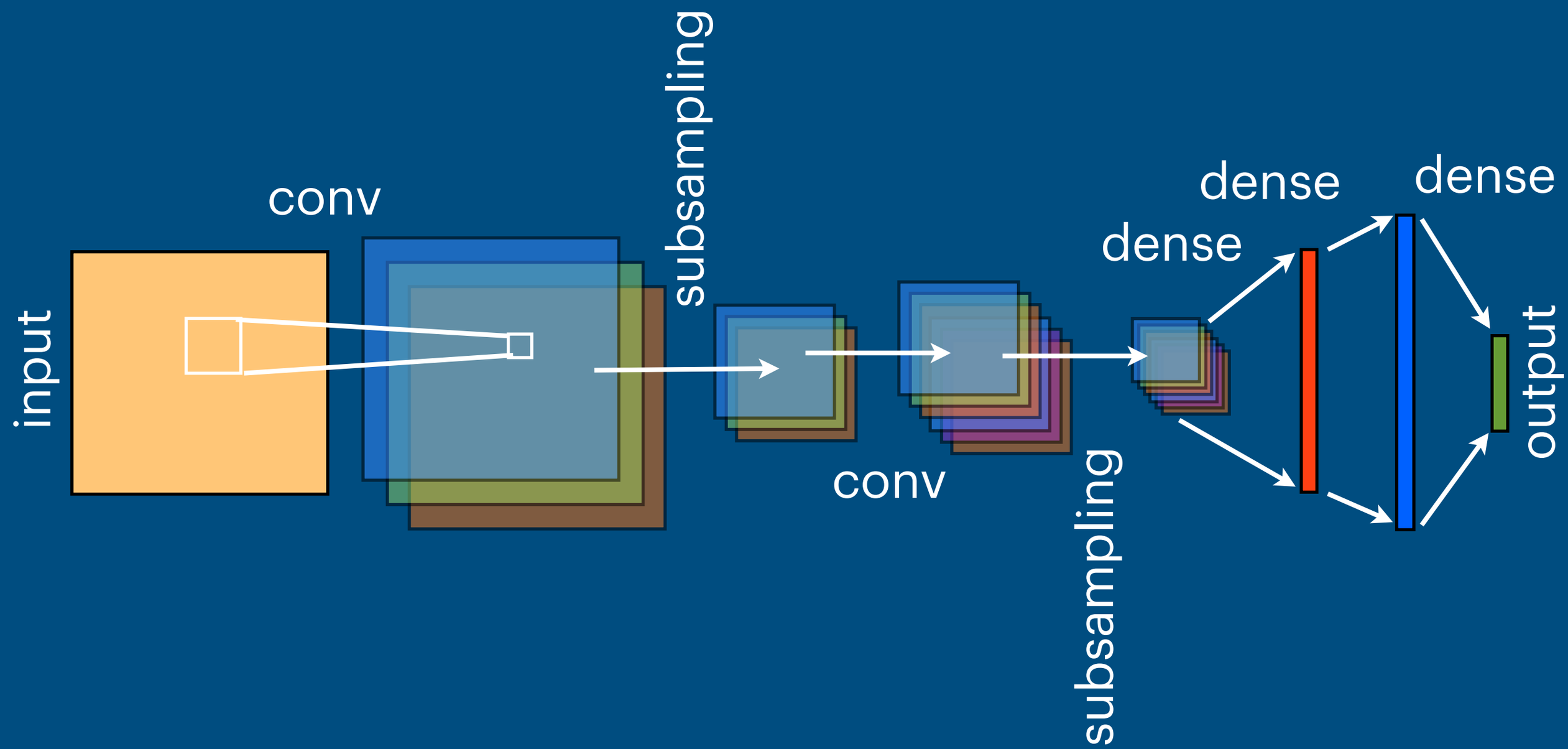
Convolutional Neural Networks for Images



Same weights (=”kernel”=”filter”) used for each neuron in the top layer!

Dramatically reduces number of trainable parameters

A fully developed convolutional neural network



Let us try it out!

FlorianMarquardt.github.io/MachineLearningThreeEasyLessons

(Lesson 3)

Where you can go from here

Directly use what you learned to:

– speed up simulations (map initial conditions to result, sample design to properties, ...)

– interpret noisy measurement traces

– recognize experimental images

– learn resolution enhancement of images

[Technical: jax-based libraries like flax or haiku for easier construction of more advanced neural networks; or similar features in tensorflow/keras and pytorch]

Further concepts (read about them...):

- use "autoencoders" to find compressed representations of unlabeled data

- use "residual networks" and "U-Nets" for advanced image processing

- use "recurrent networks" to analyze time series

- use "graph neural networks" for input like molecular structures and other data of variable size

- use "reinforcement learning" to discover control strategies

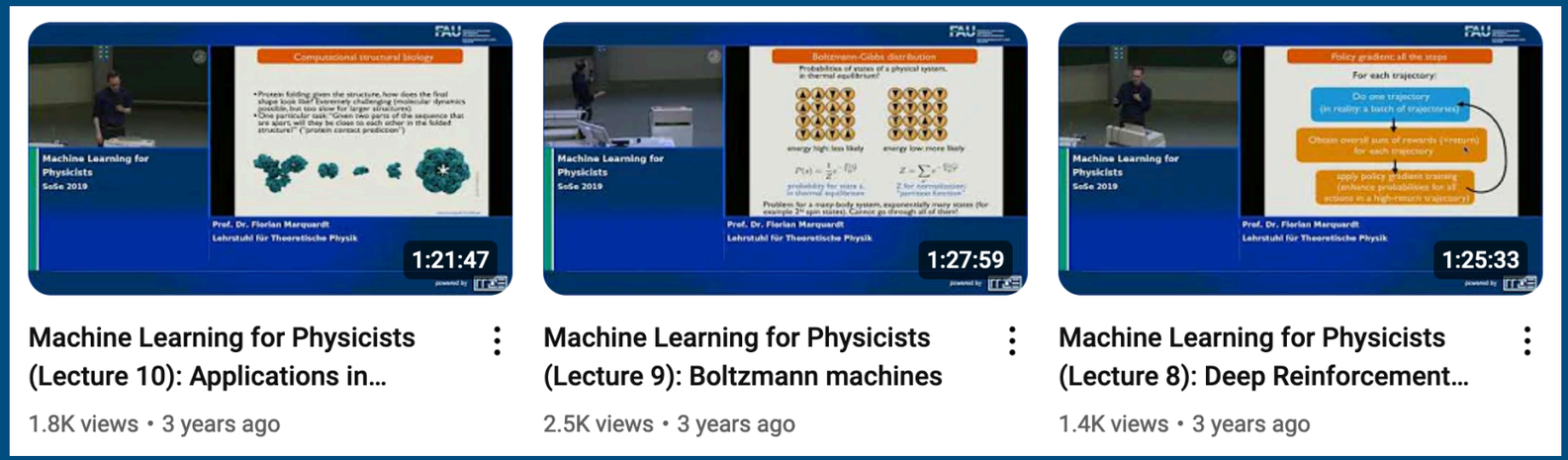
- use "normalizing flows" or "diffusion models" to generate new data similar to existing data

- use "transformers" to analyze and produce sequences with long dependencies

Online lecture series by F.M., designed for physicists

(florian_marquardt_physics YouTube channel)

Machine Learning for Physicists (the basics)



Advanced Machine Learning for Scientific Discovery