Lecture 9

Neural Networks

CS328 - Numerical Methods for Visual Computing and Machine Learning

Prof. Wenzel Jakob

EPFL RGL Realistic Graphics Lab

An impossible task

Cannot cover neural networks in just on lecture

An impossible task

Cannot cover neural networks in just on lecture

- **Pitch**: What do neural networks enable, and why you should learn about them.
  (*Warning*: heavily commercial product demos ahead)

An impossible task

Cannot cover neural networks in just on lecture

- **Pitch**: What do neural networks enable, and why you should learn about them. (*Warning*: heavily commercial product demos ahead)
- **Anti-pitch**: Why s*pecializing completely* on machine learning might *not* be a good career choice.

An impossible task

Cannot cover neural networks in just on lecture

- **Pitch**: What do neural networks enable, and why you should learn about them. (*Warning*: heavily commercial product demos ahead)
- **Anti-pitch**: Why s*pecializing completely* on machine learning might *not* be a good career choice.
- **Agenda**:
  - Biological motivations
  - Multilayer perceptrons
  - Activation & loss functions

An impossible task

Cannot cover neural networks in just on lecture

- **Pitch**: What do neural networks enable, and why you should learn about them. (*Warning*: heavily commercial product demos ahead)
- **Anti-pitch**: Why s*pecializing completely* on machine learning might *not* be a good career choice.
- **Agenda**:
  - Biological motivations
  - Multilayer perceptrons
  - Activation & loss functions
- **Preview**:
  - Deep & convolutional networks, GANs, diffusion models, transformers

An impossible task

Cannot cover neural networks in just on lecture

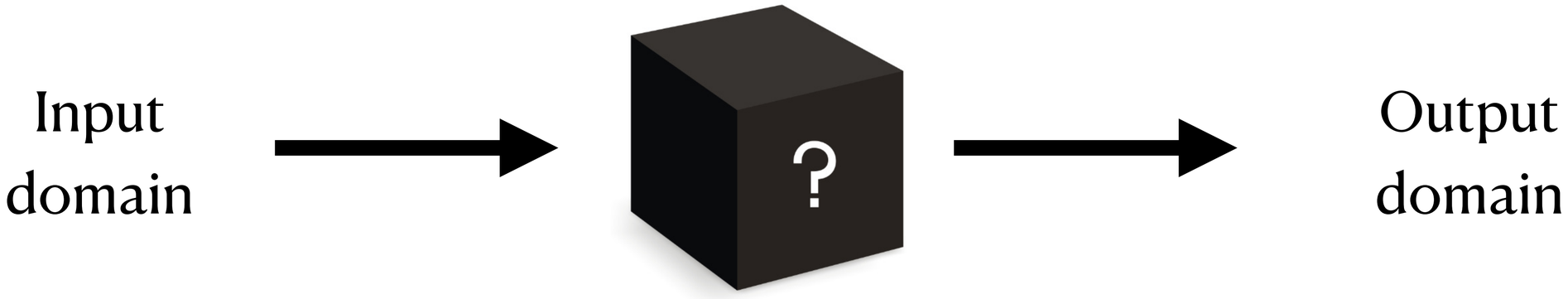
- **Pitch**: What do neural networks enable, and why you should learn about them. (*Warning*: heavily commercial product demos ahead)
- **Anti-pitch**: Why s*pecializing completely* on machine learning might *not* be a good career choice.
- **Agenda**:
  - Biological motivations
  - Multilayer perceptrons
  - Activation & loss functions

CS-233, CS-433, PHYS-467

- **Preview**:
  - Deep & convolutional networks, GANs, diffusion models, transformers

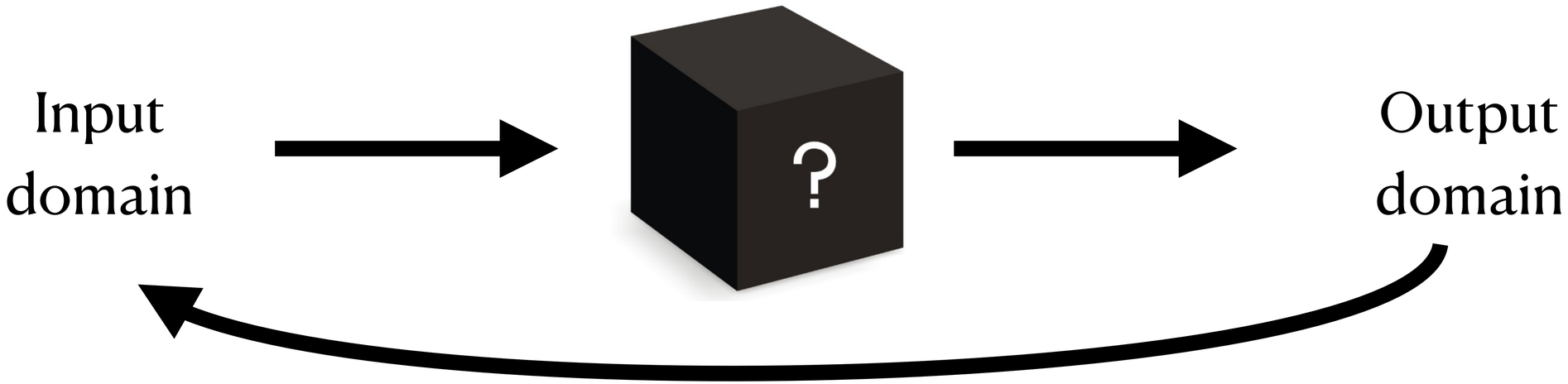
Inverse problems

$$f(\mathbf{x})$$



Inverse problems

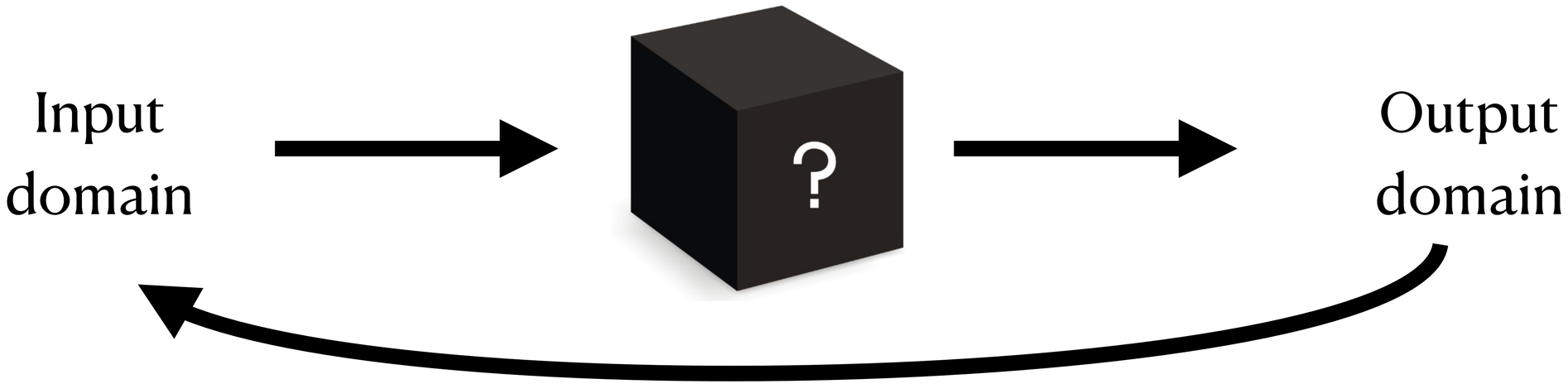
$f(\mathbf{x})$



$f^{-1}(\mathbf{x})$?

Inverse problems based on data

$$f(\mathbf{x})$$



$$f^{-1}(\mathbf{x}) \quad ?$$

Inverse problems based on data

$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \ldots$

All we have is data!

$\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4, \ldots$

Input domain

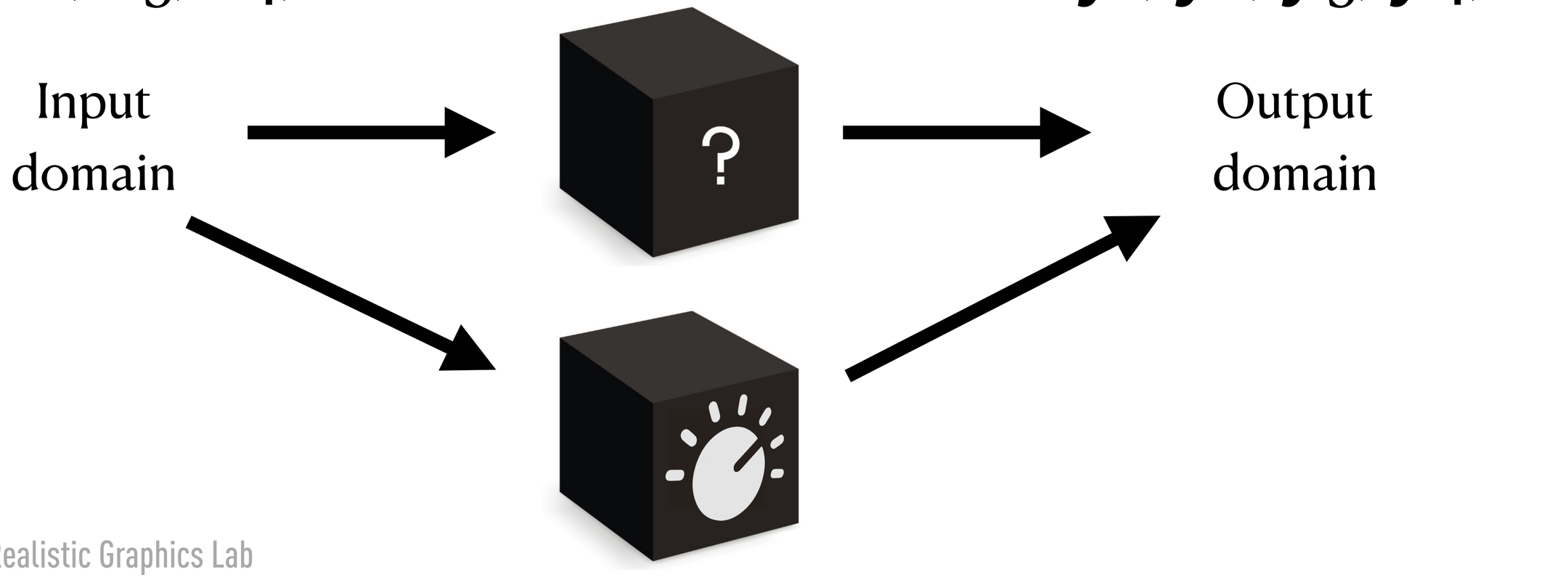
Output domain

Inverse problems based on data

$x_1, x_2, x_3, x_4, \ldots$

All we have is data!

$y_1, y_2, y_3, y_4, \ldots$

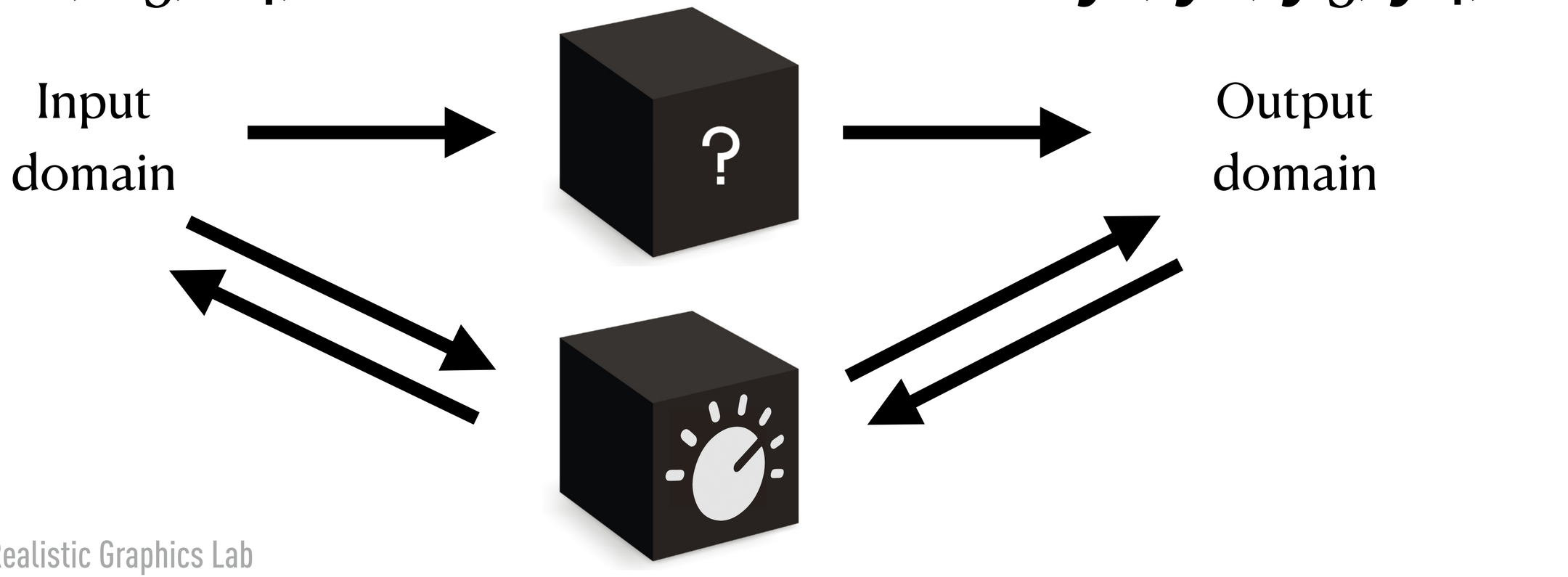


Inverse problems based on data

$x_1, x_2, x_3, x_4, \ldots$

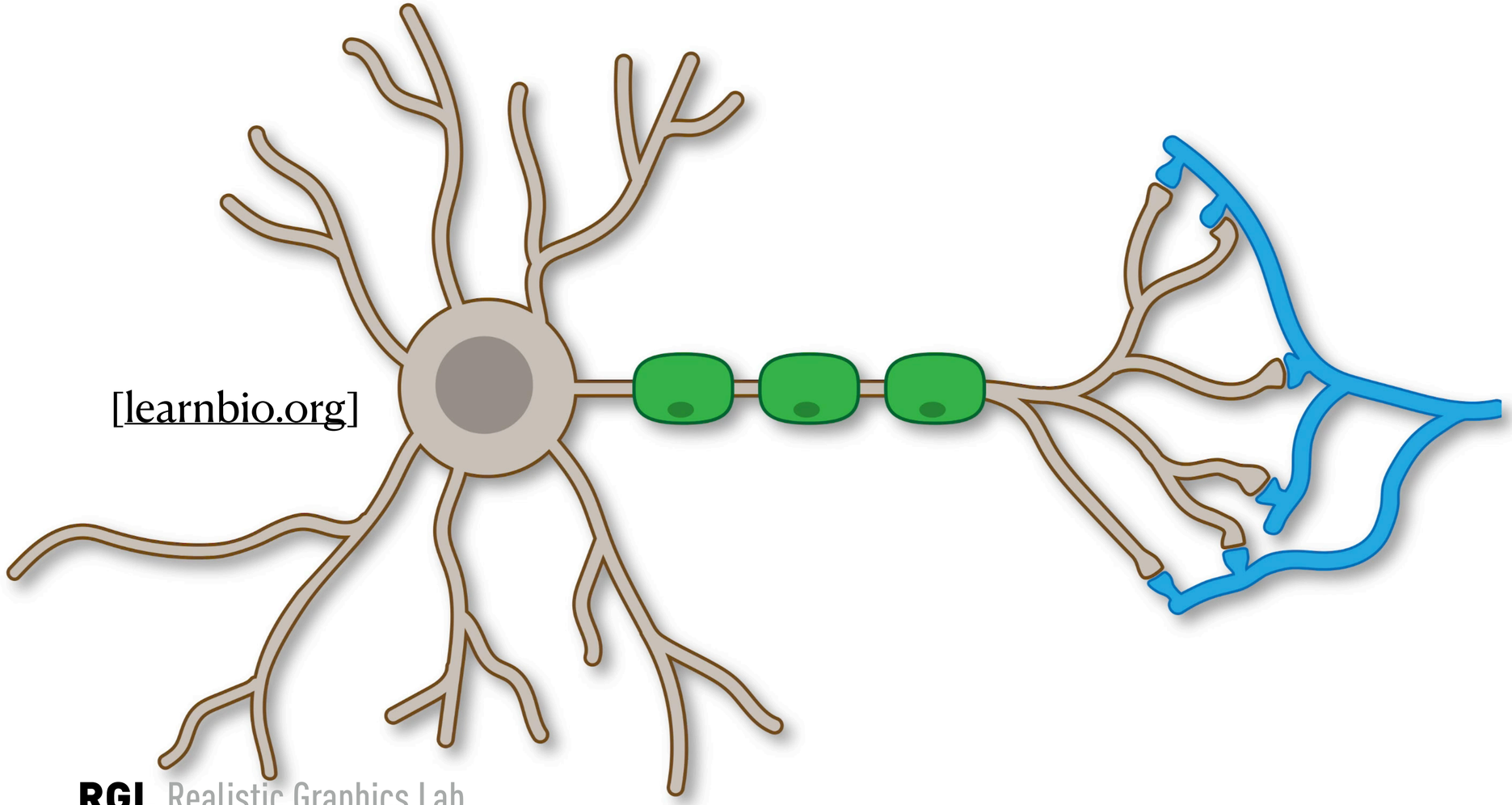
All we have is data!

$y_1, y_2, y_3, y_4, \ldots$

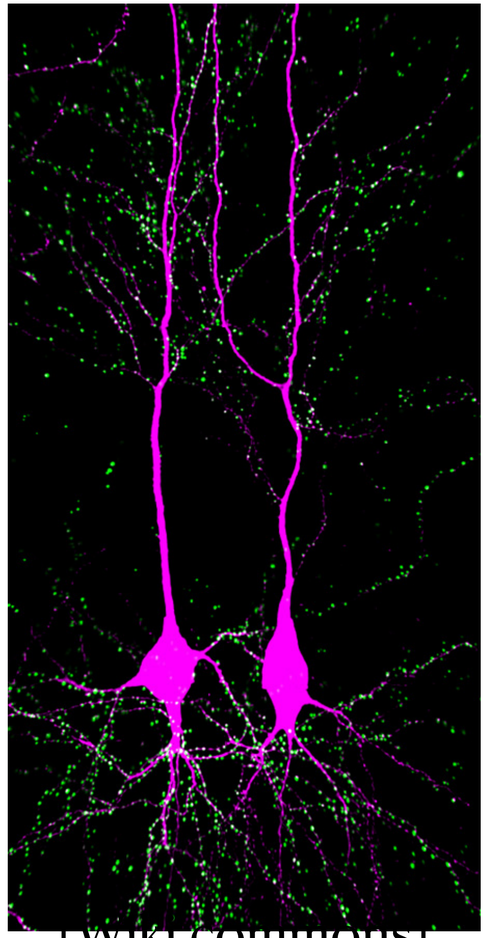


A biological neuron

Very high level view, biological neurons are *very* complicated.



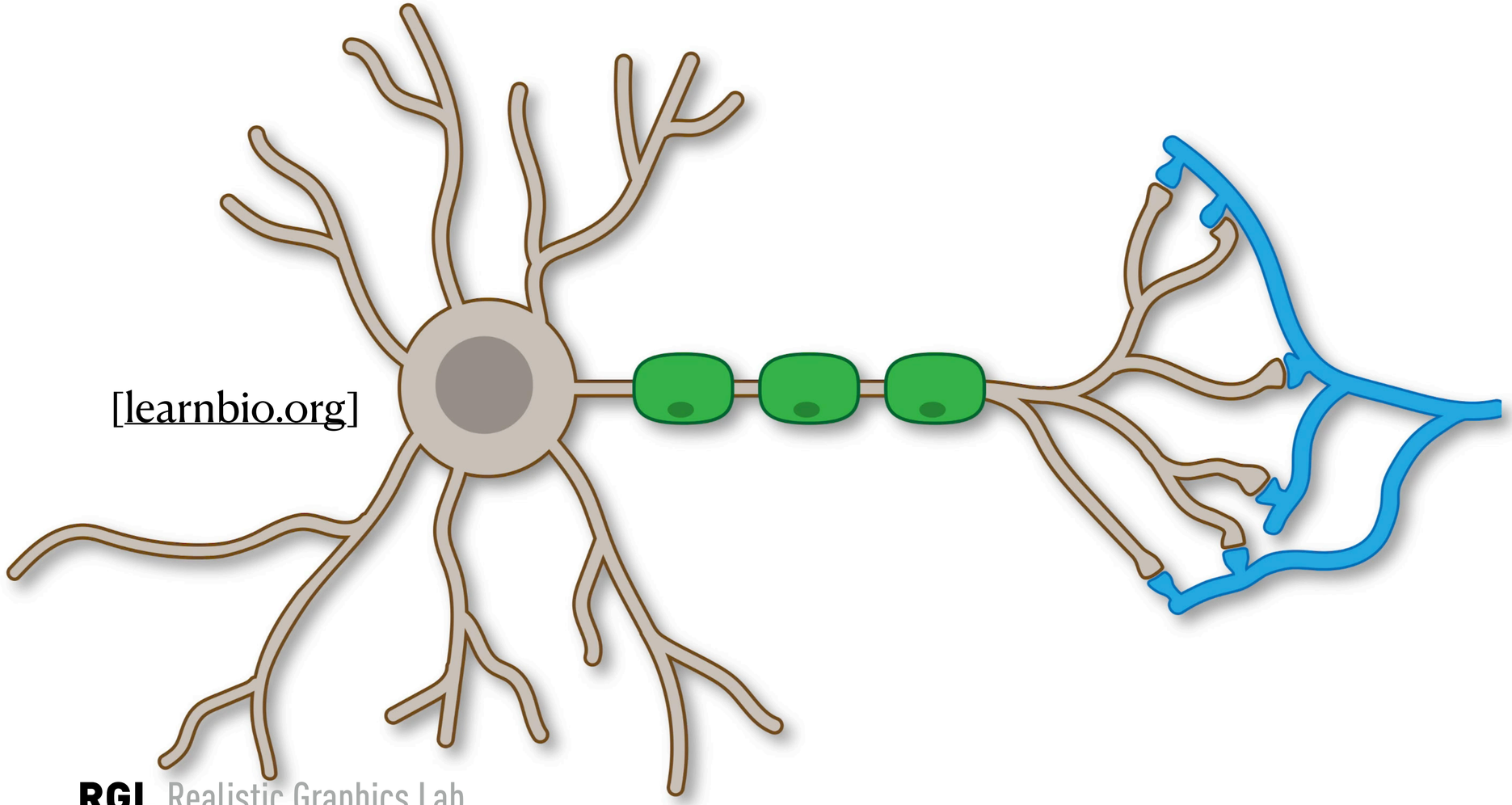
[learnbio.org]



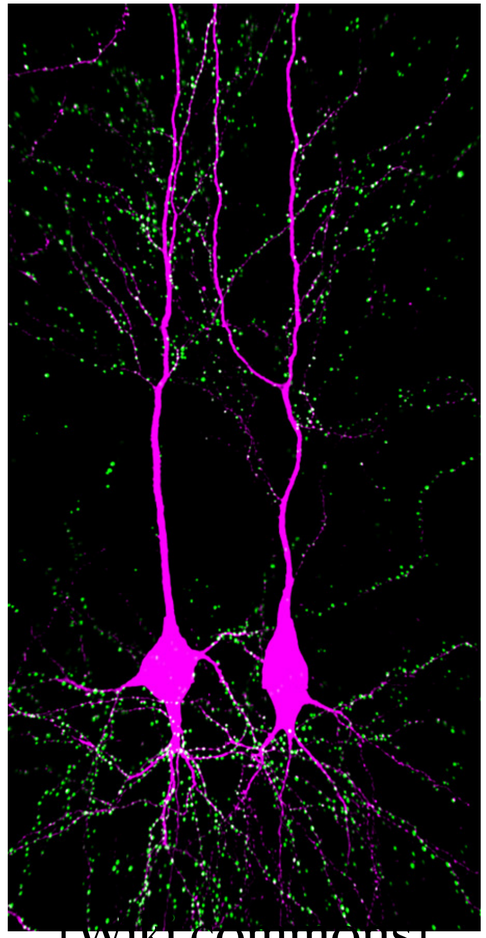
[Wiki commons]

A biological neuron

Very high level view, biological neurons are *very* complicated.



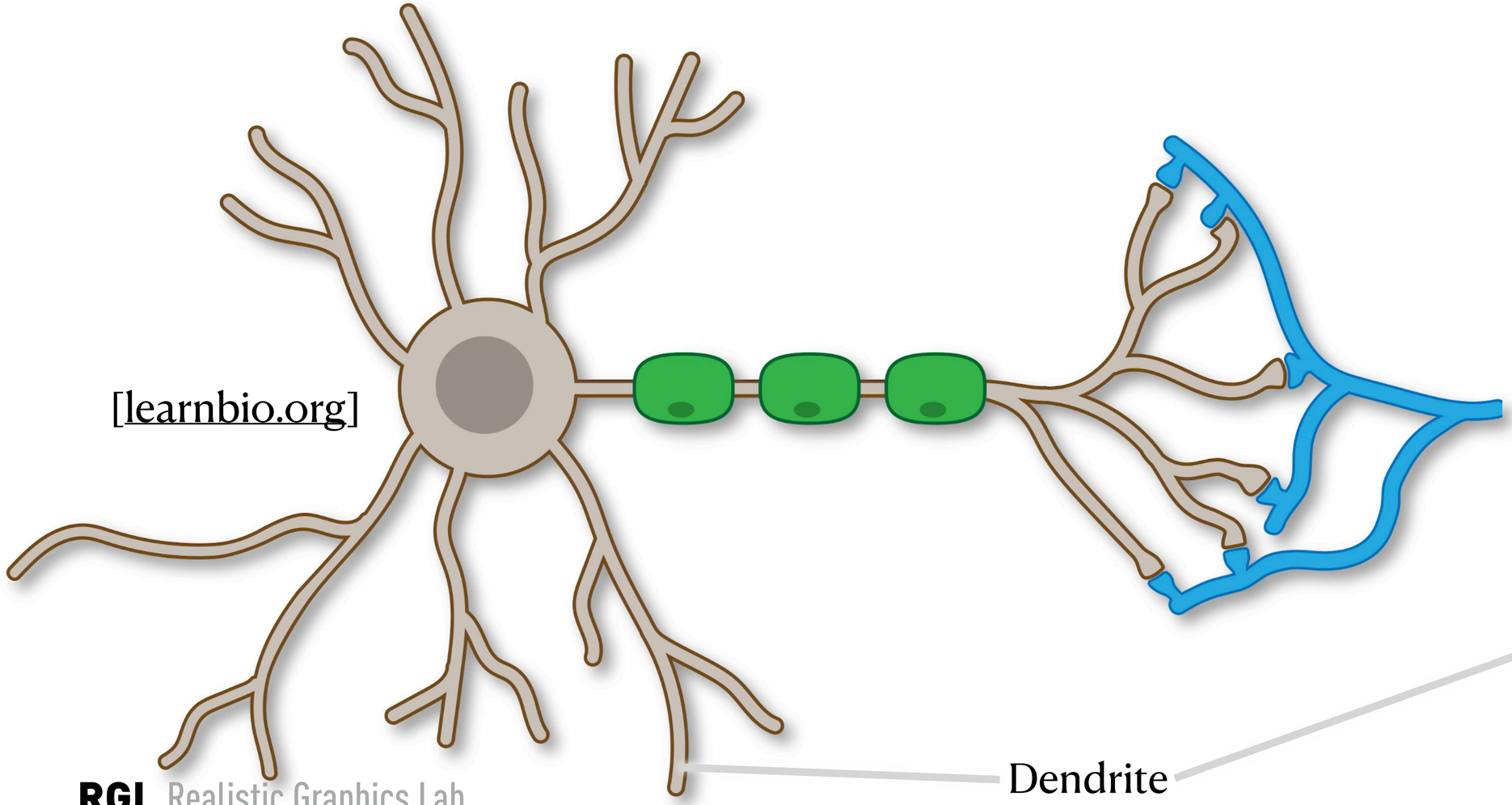
[learnbio.org]



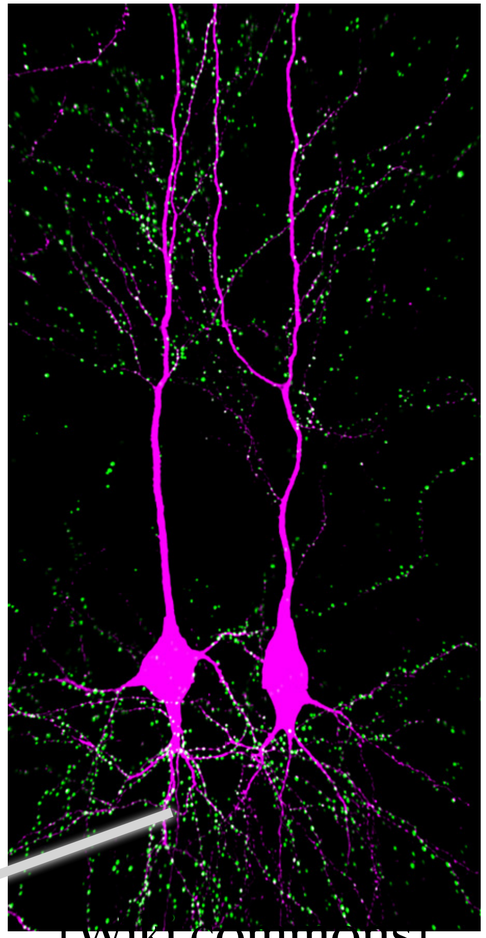
[Wiki commons]

A biological neuron

Very high level view, biological neurons are *very* complicated.



[learnbio.org]

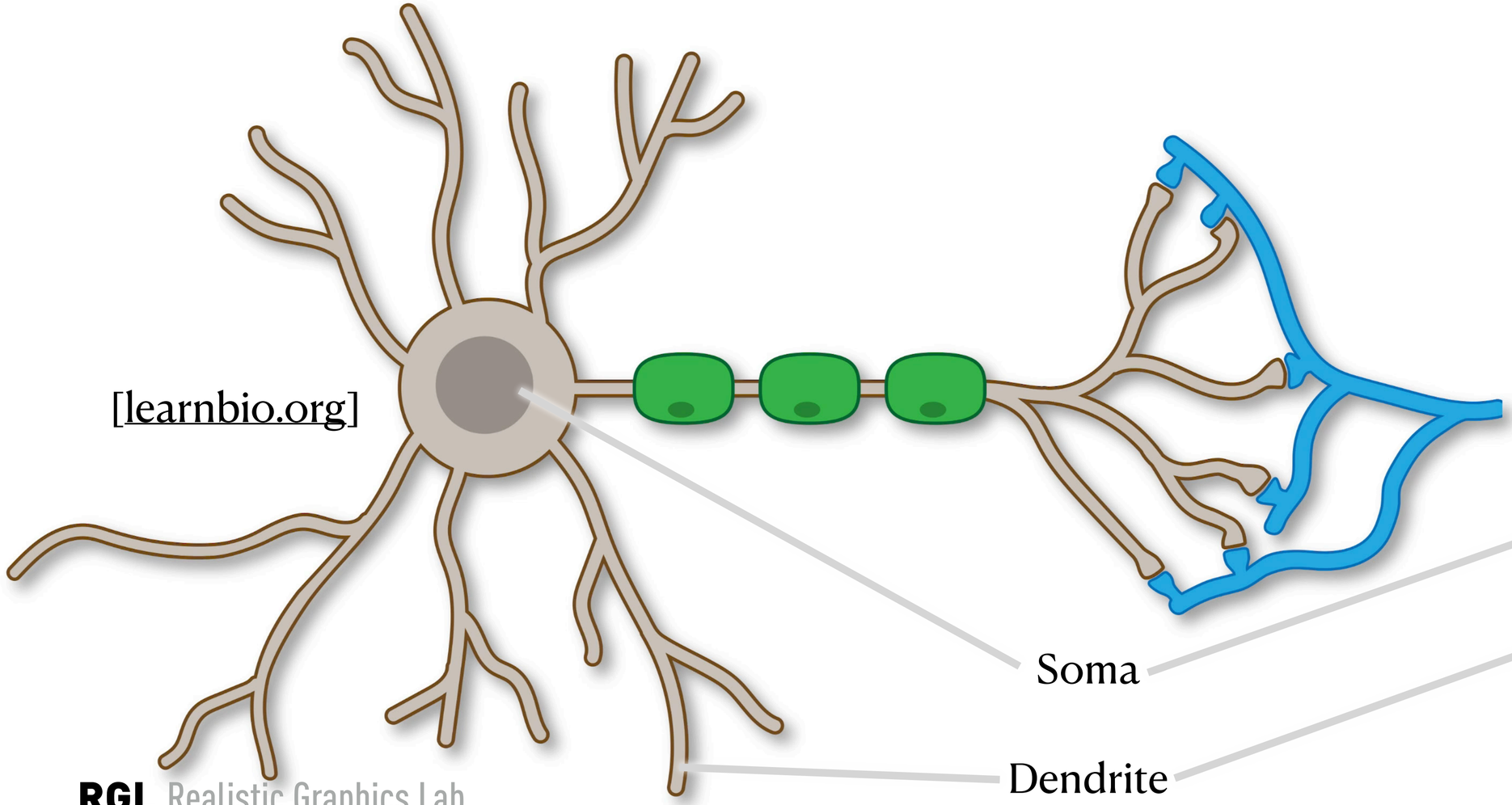


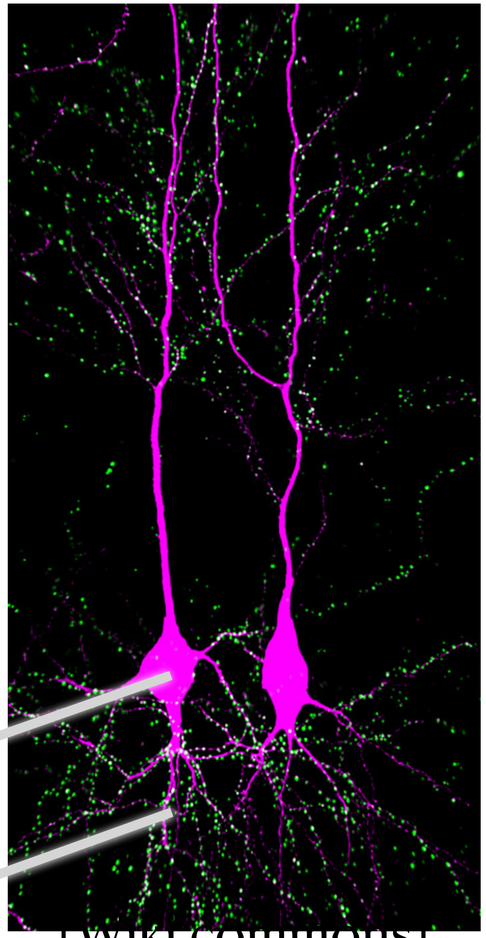
[Wiki commons]

Dendrite

A biological neuron

Very high level view, biological neurons are *very* complicated.

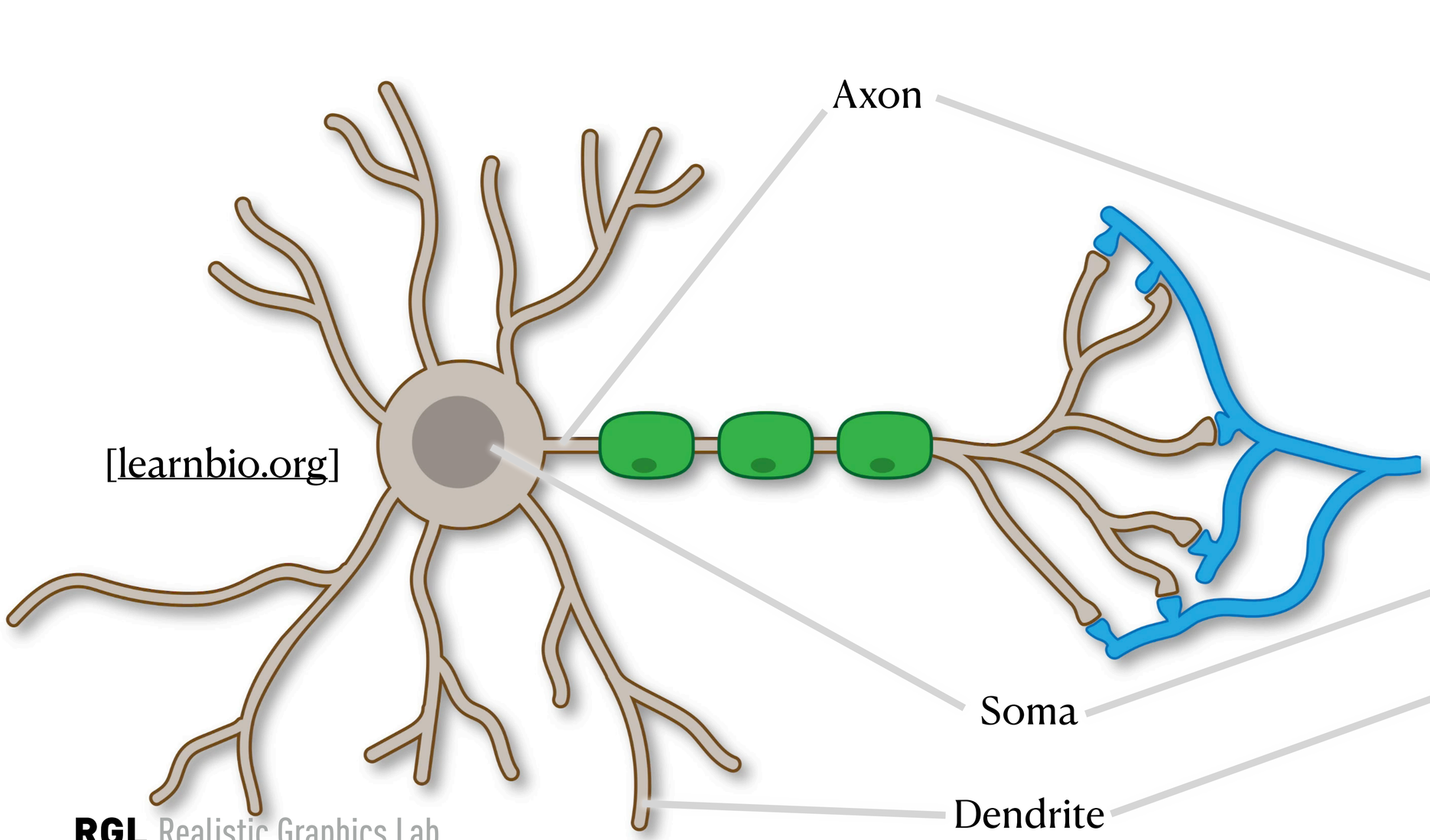


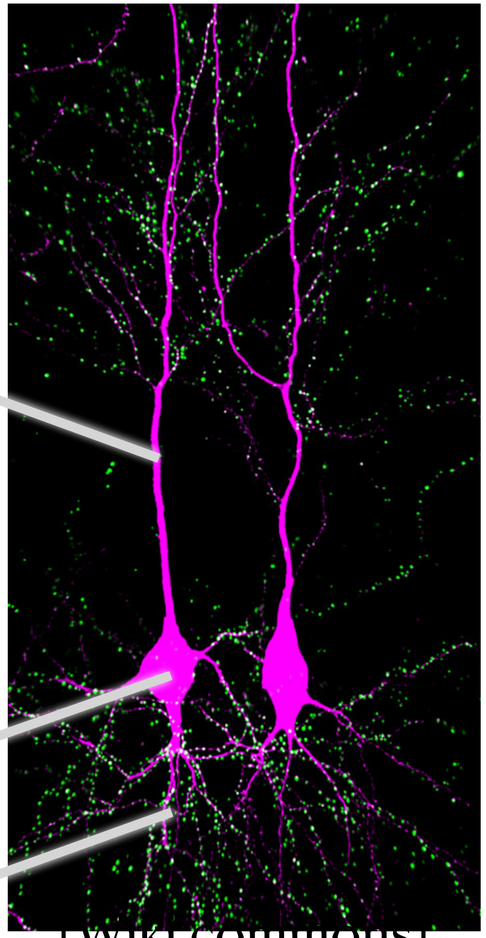


[Wiki commons]

A biological neuron

Very high level view, biological neurons are *very* complicated.

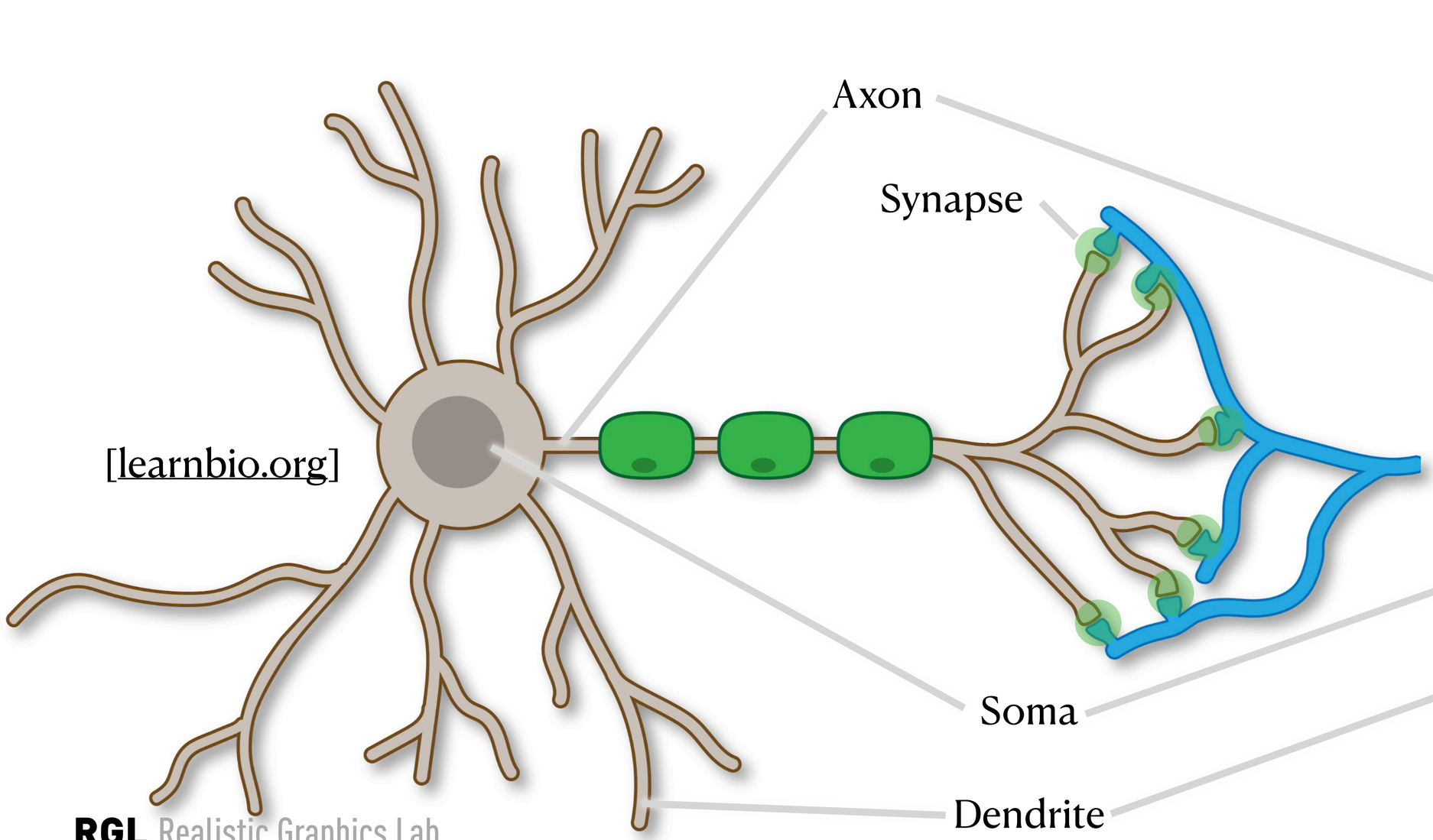


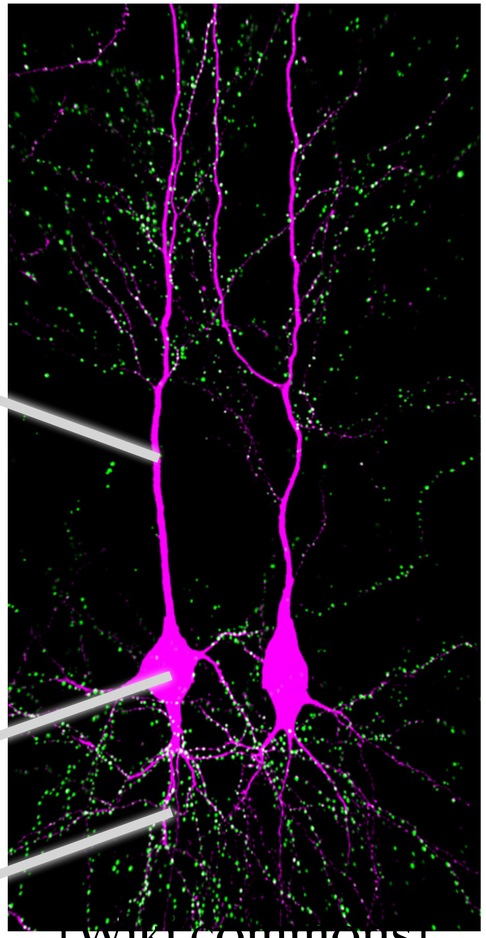


[Wiki commons]

A biological neuron

Very high level view, biological neurons are *very* complicated.



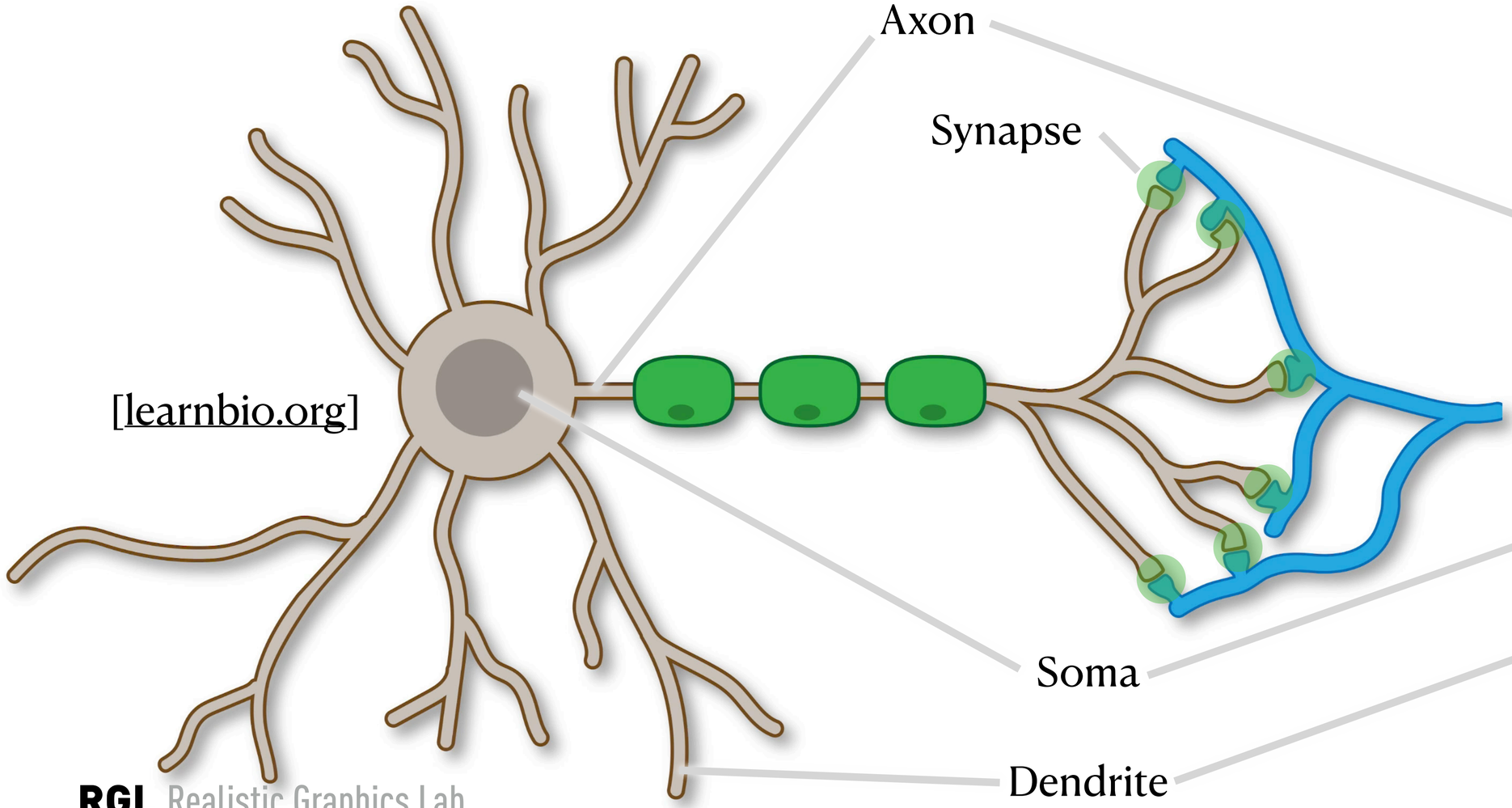


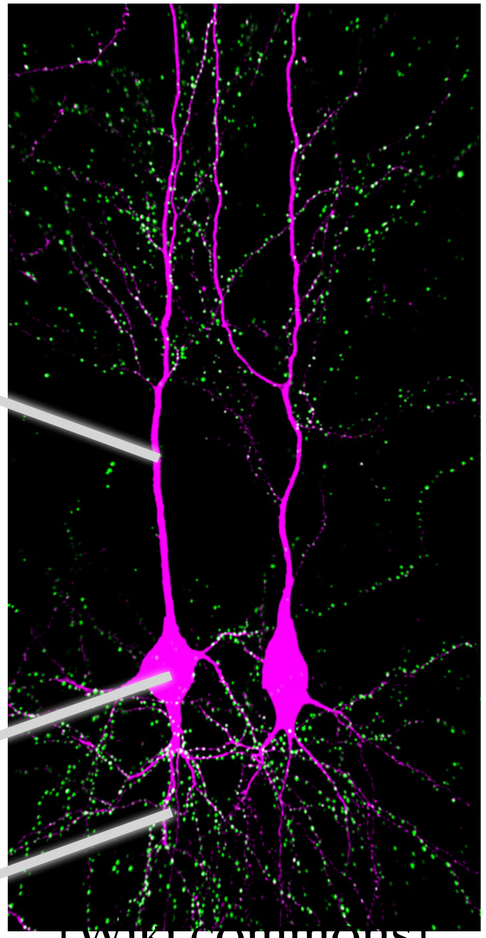
[Wiki commons]

A biological neuron

Very high level view, biological neurons are *very* complicated.

In some parts of the brain, neurons have a **tree-like topology**: many inputs, 1 output



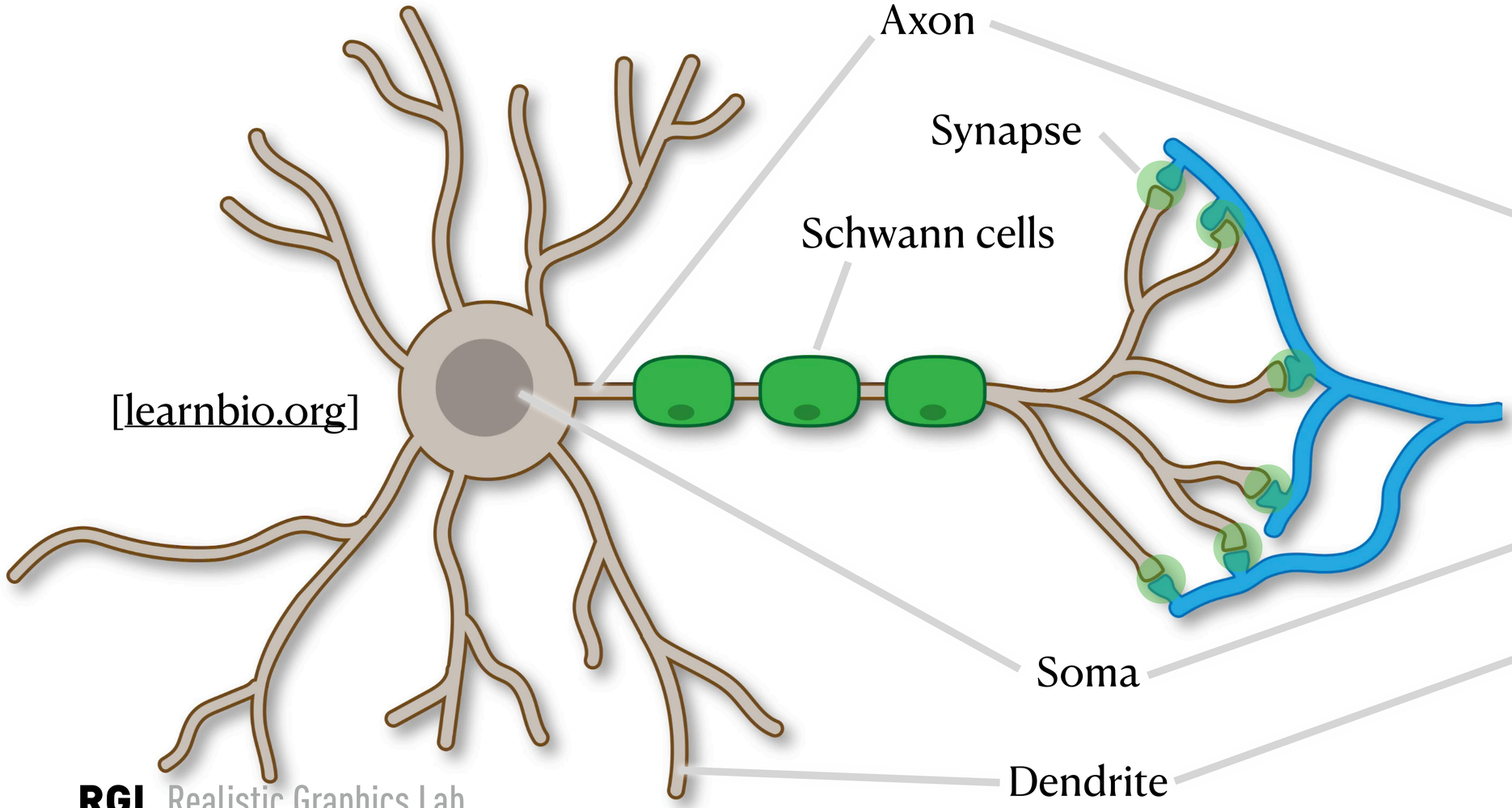


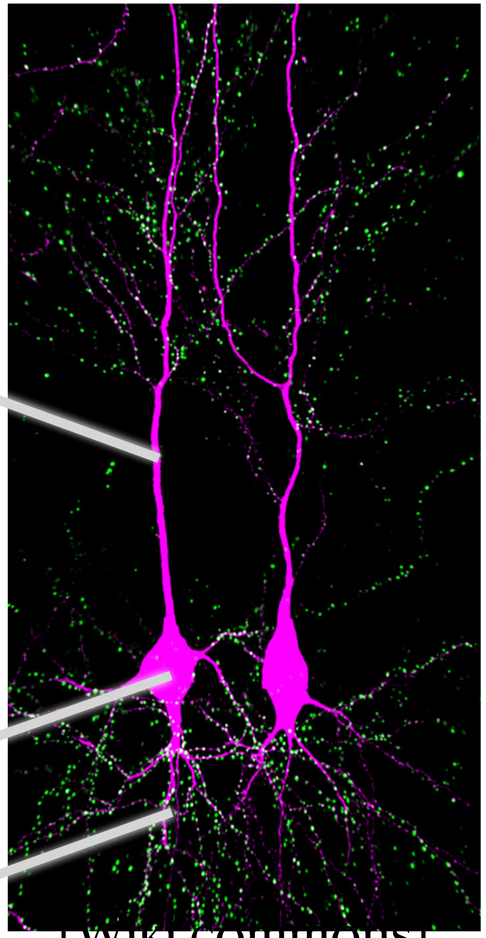
[Wiki commons]

A biological neuron

Very high level view, biological neurons are *very* complicated.

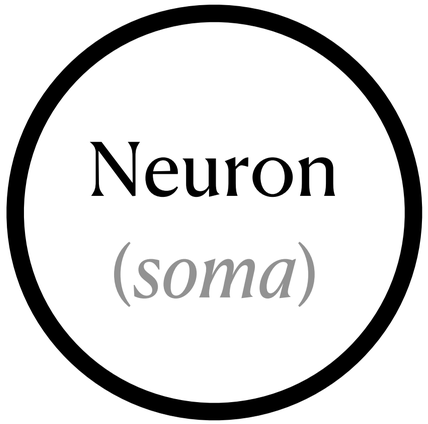
In some parts of the brain, neurons have a **tree-like topology**: many inputs, 1 output



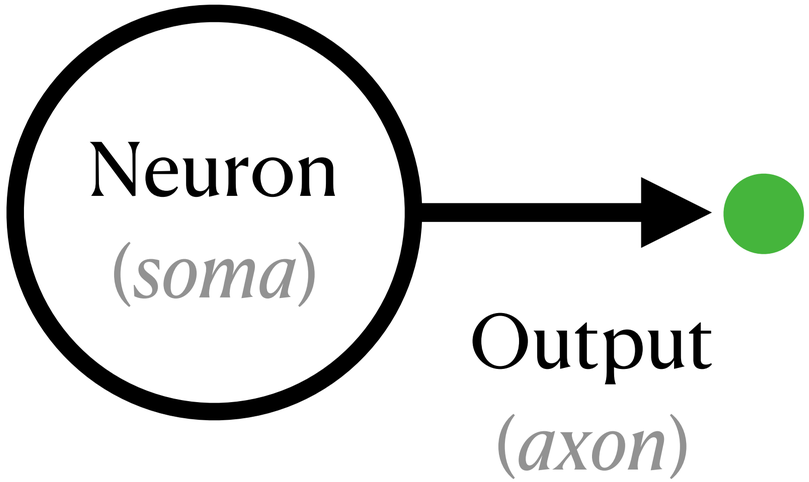


[Wiki commons]

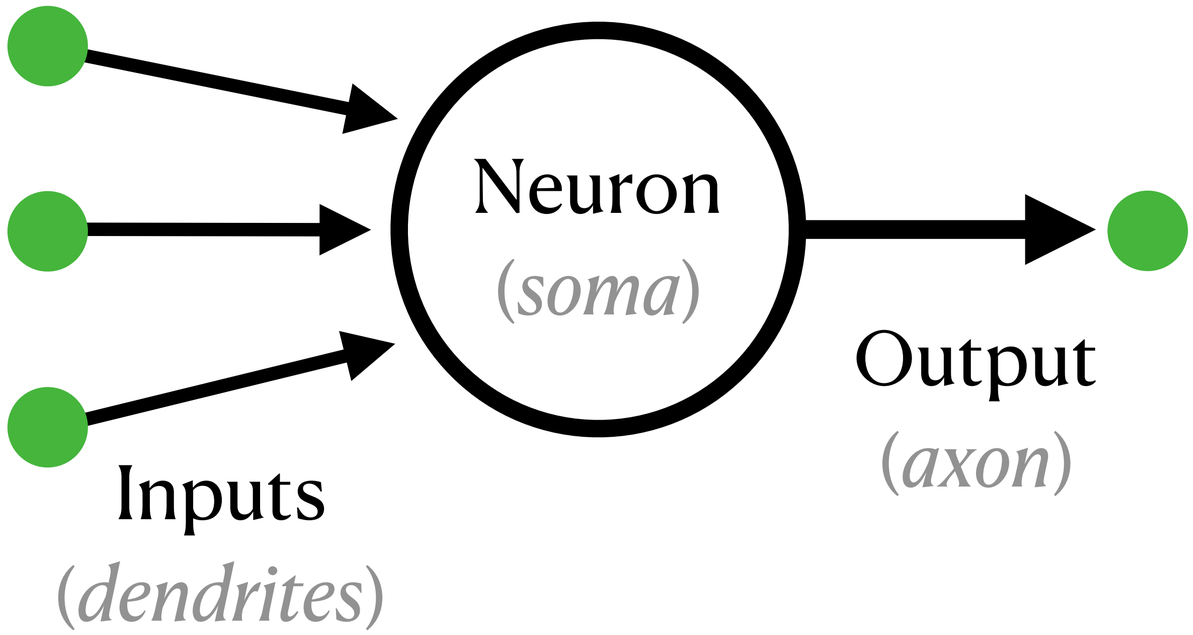
An artificial neuron



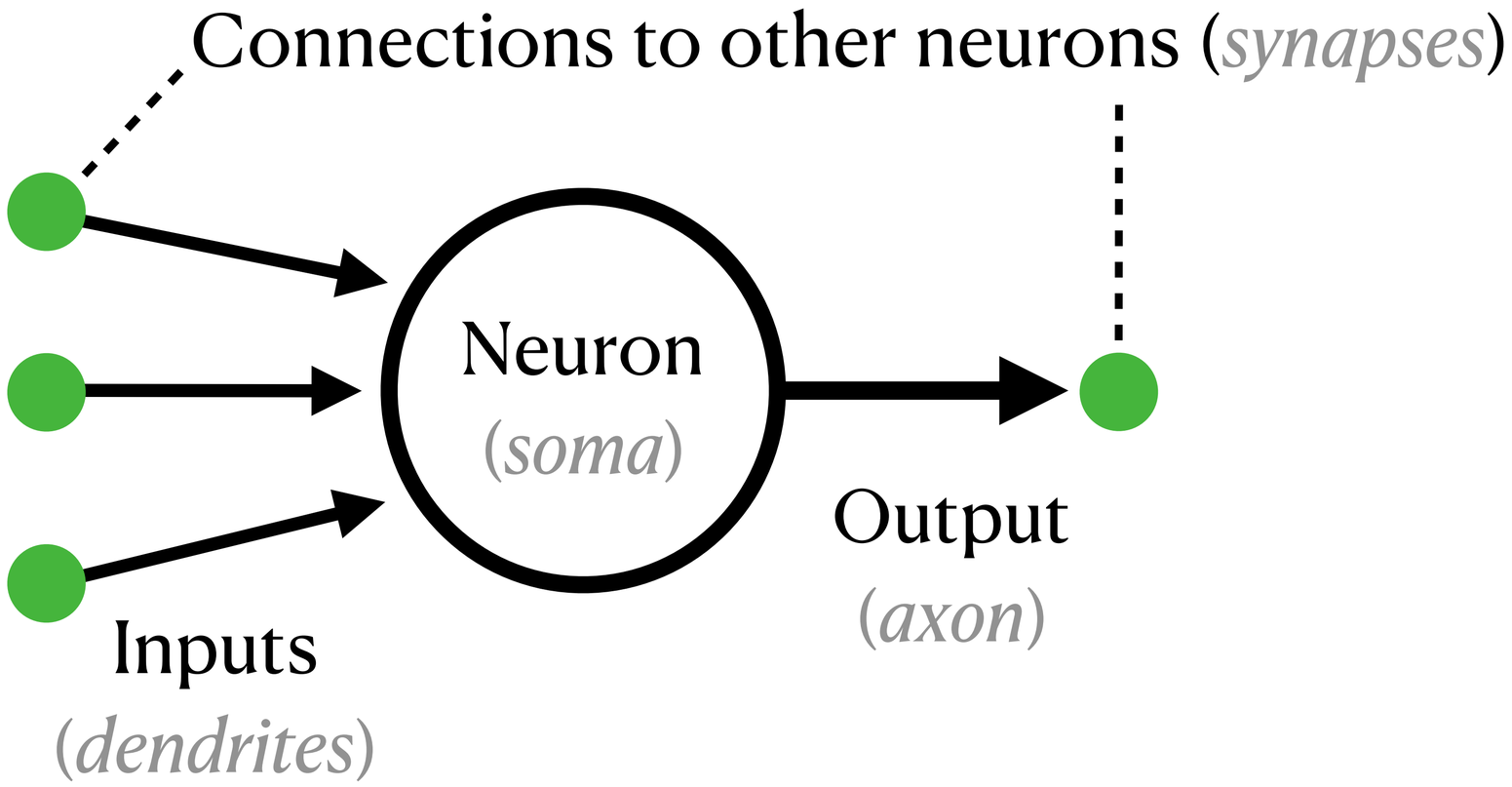
An artificial neuron



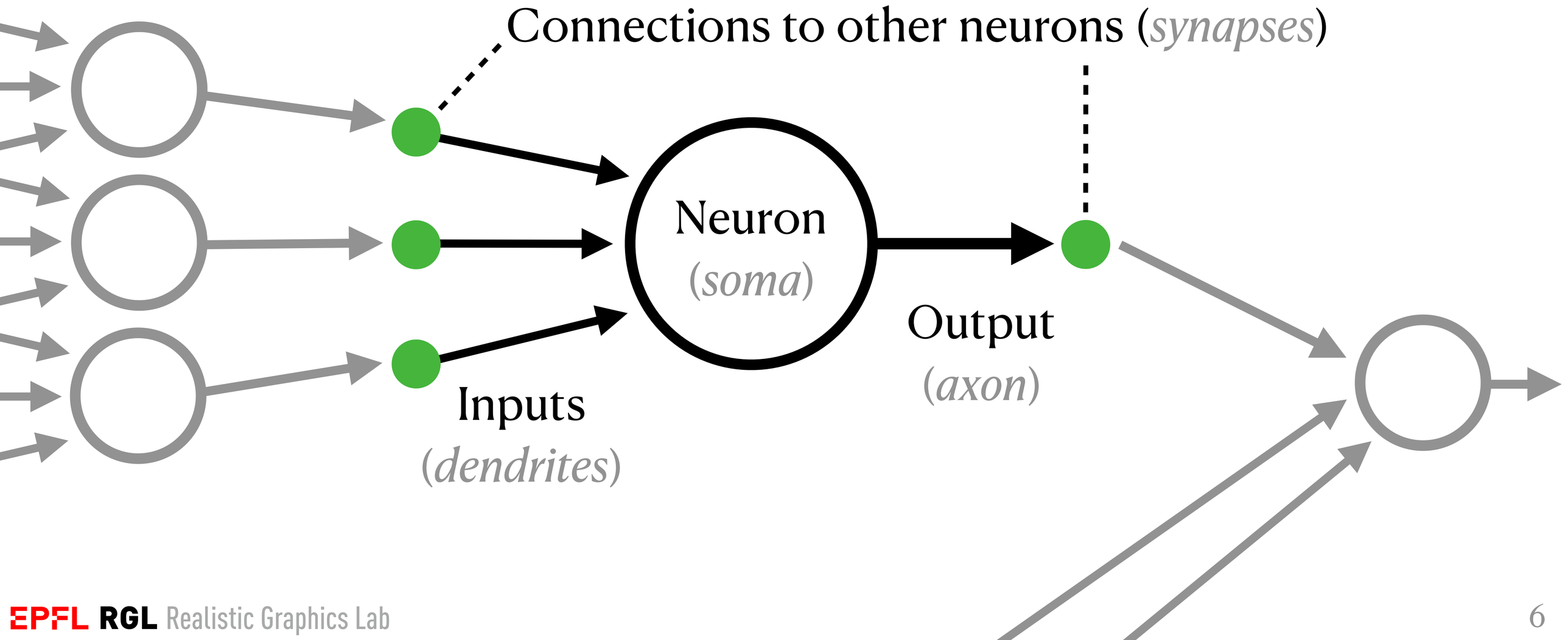
An artificial neuron



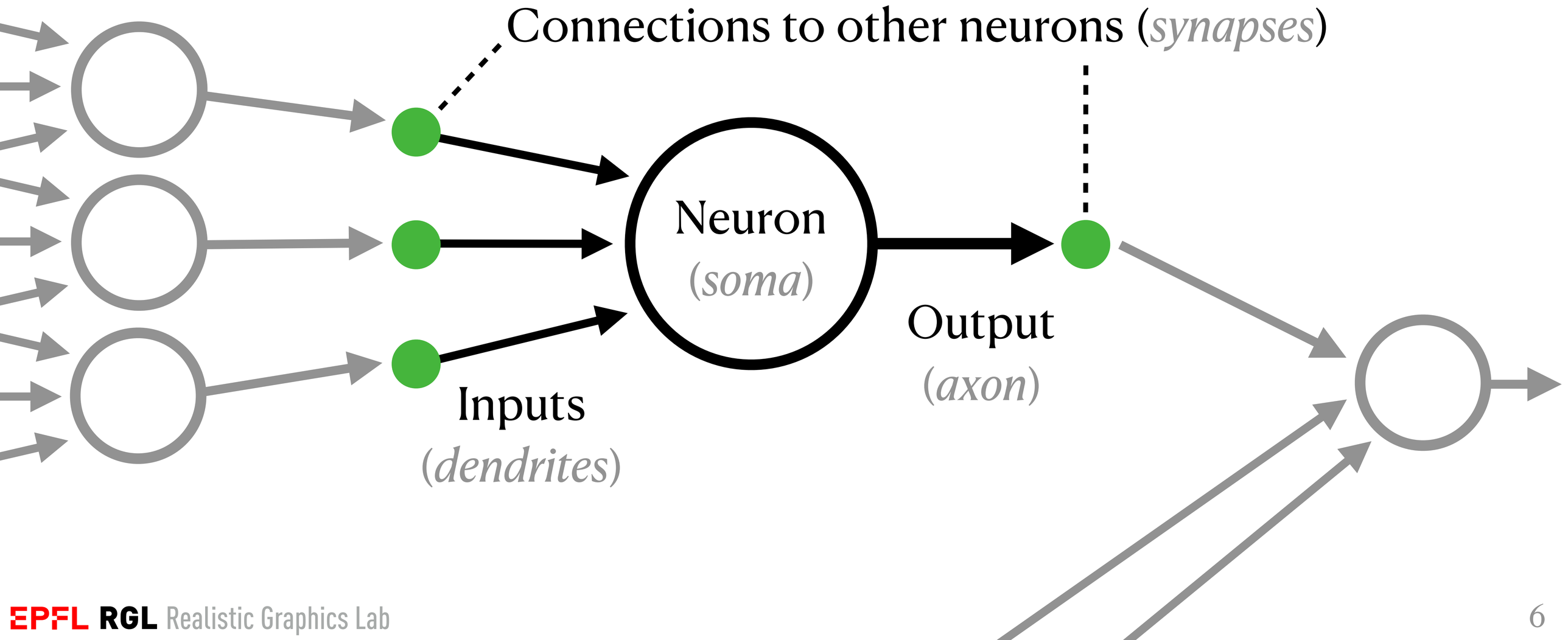
An artificial neuron



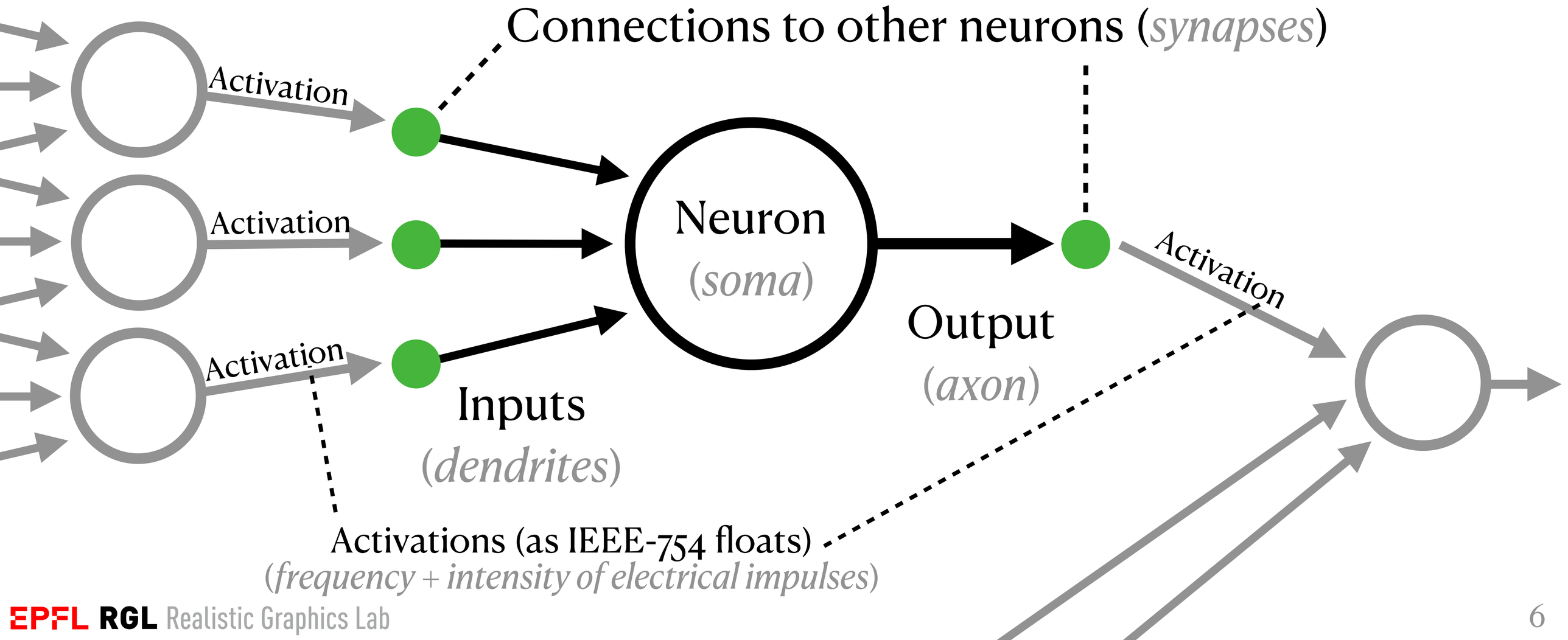
An artificial neuron



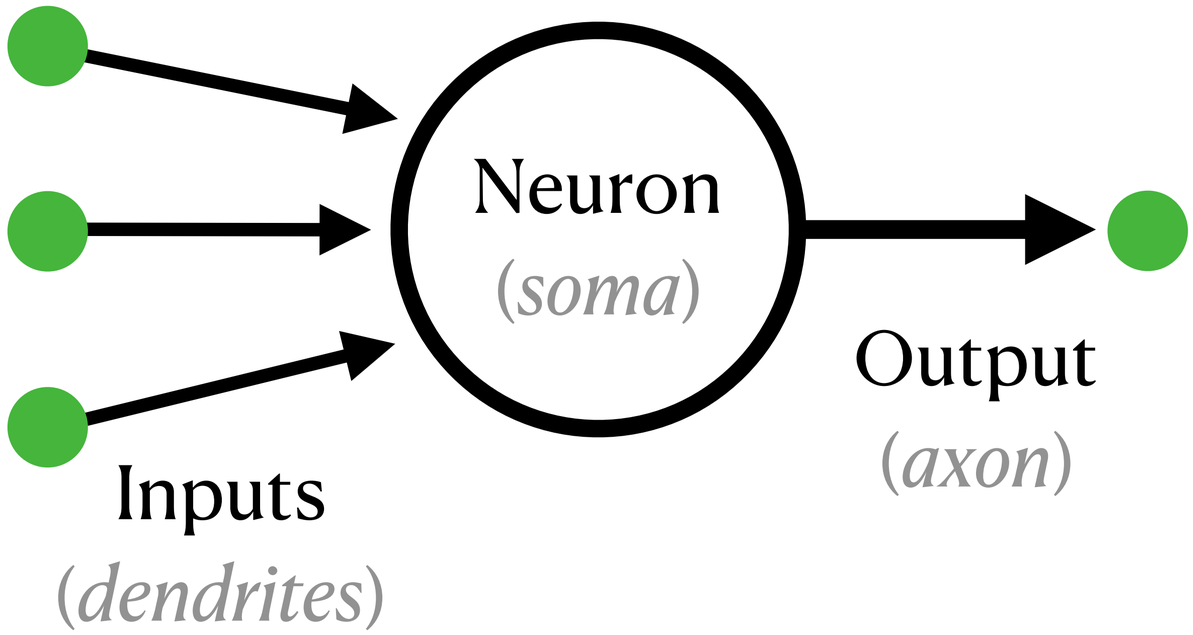
An artificial neuron



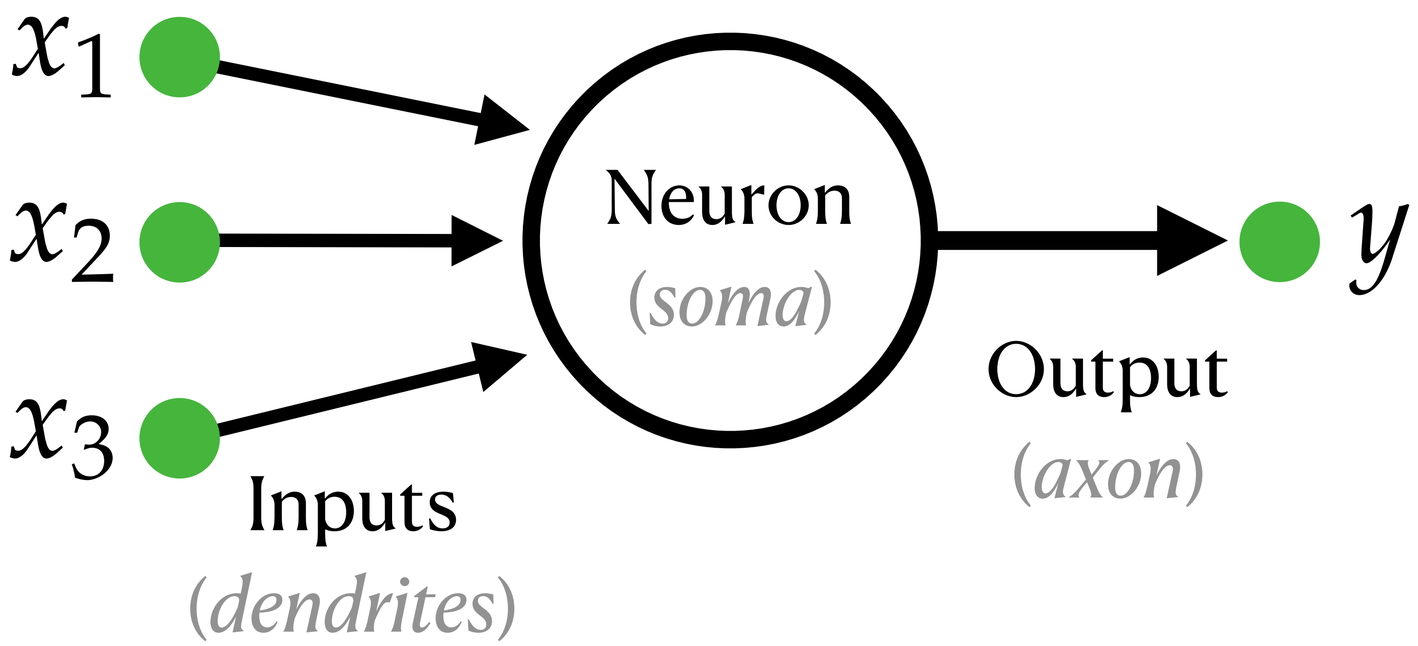
An artificial neuron



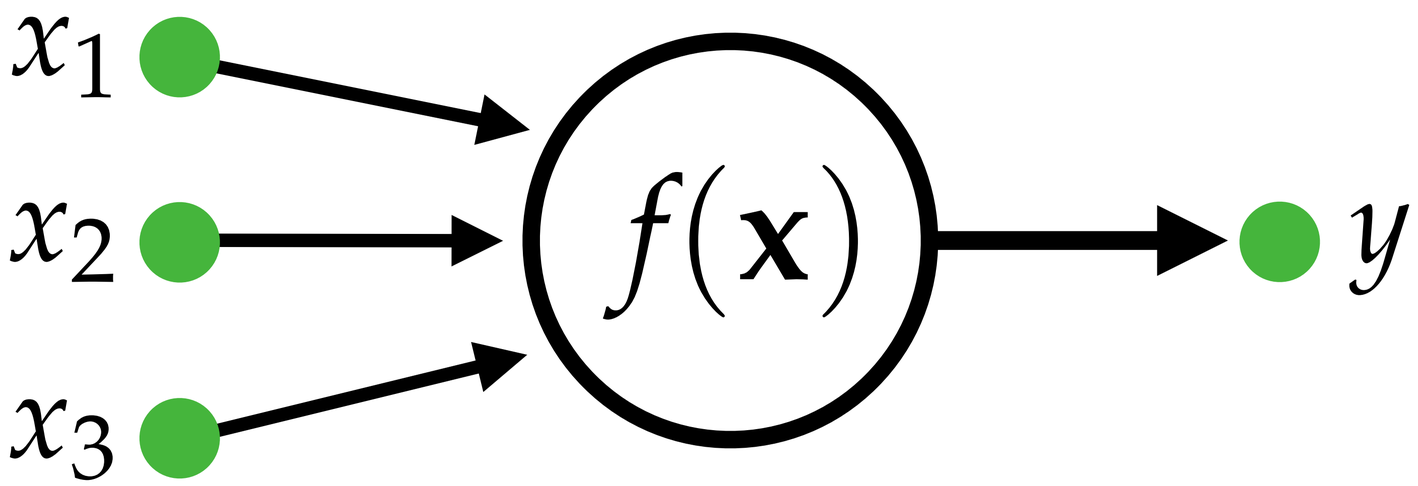
An artificial neuron



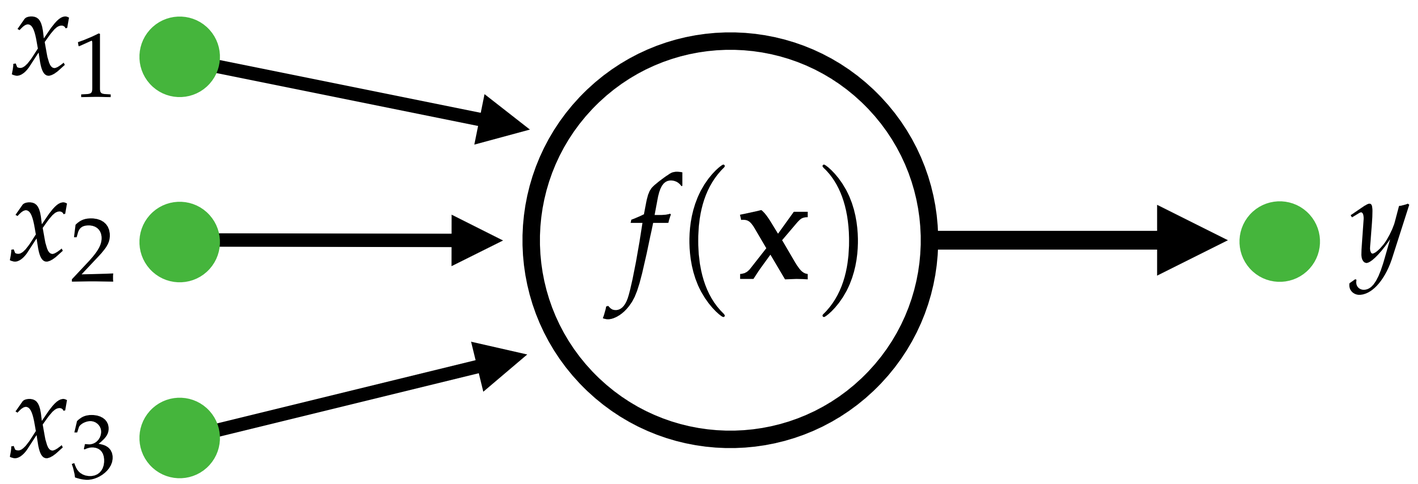
An artificial neuron



An artificial neuron

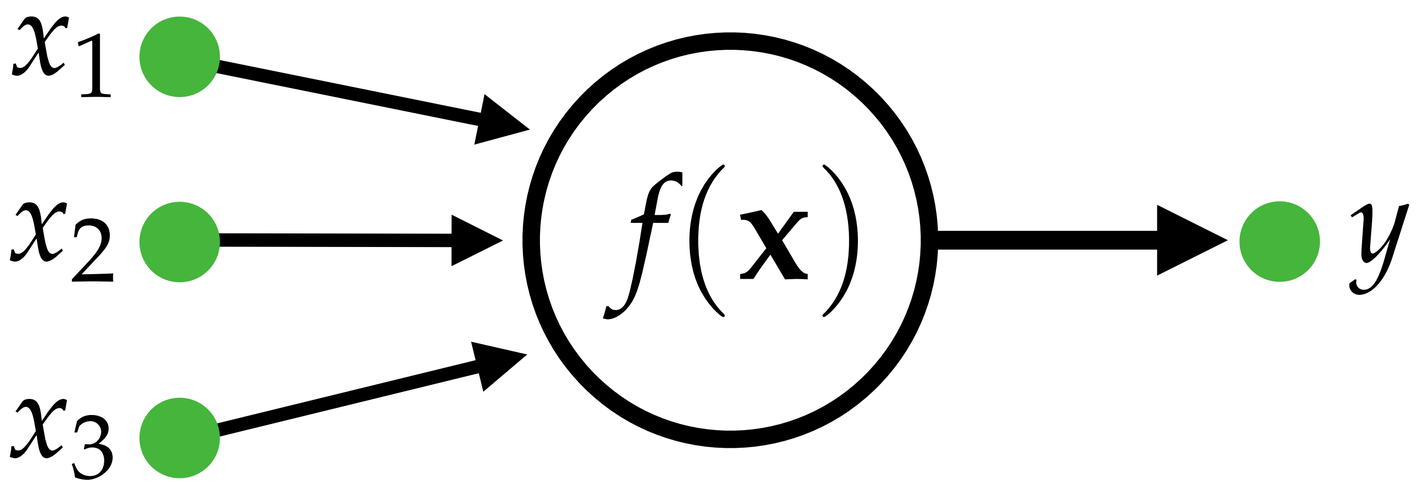


An artificial neuron — linear case



$$f(\mathbf{x}) = w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + b$$

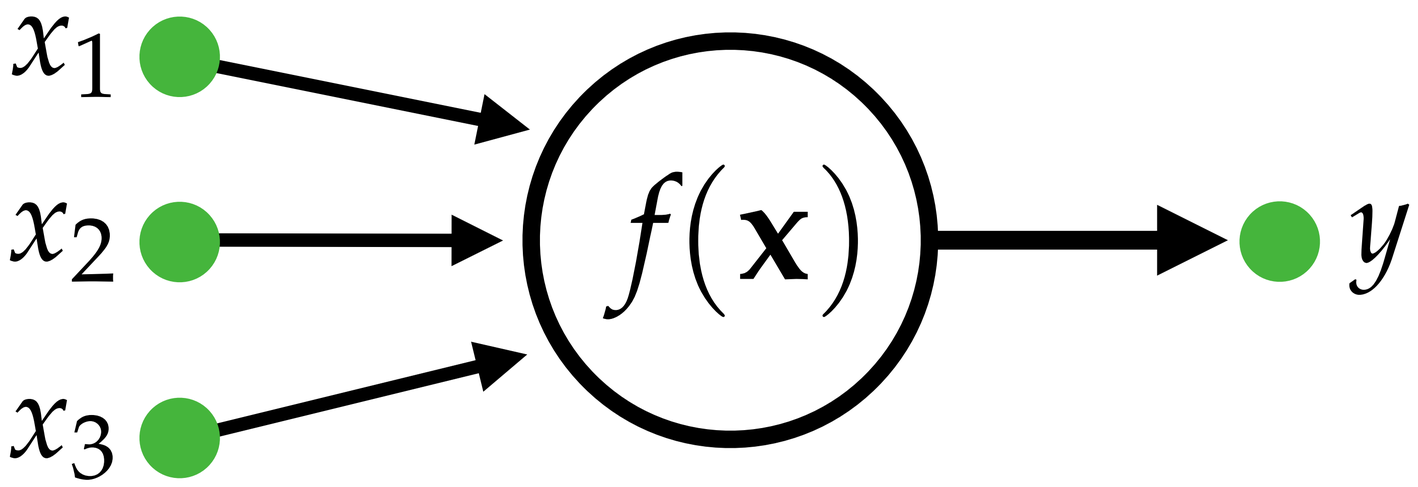
An artificial neuron — linear case



$$f(\mathbf{x}) = w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + b$$

Unknown parameters. We must *optimize* ("train") the network to find good values for these.

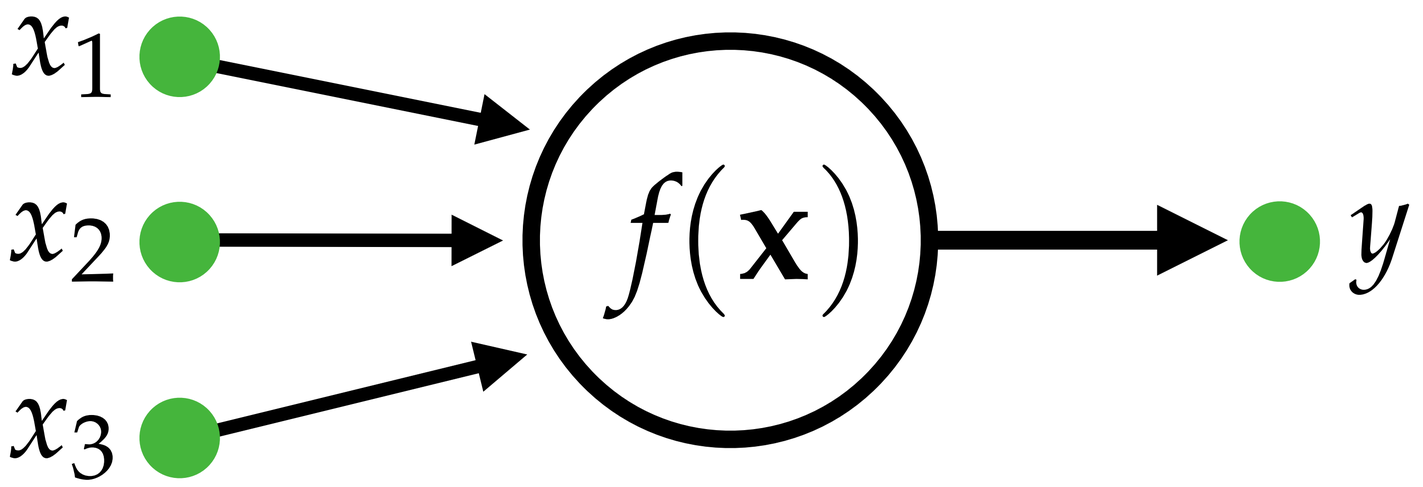
An artificial neuron — linear case



$$f(\mathbf{x}) = w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + b = \mathbf{w} \cdot \mathbf{x} + b$$

Unknown parameters. We must *optimize* ("train") the network to find good values for these.

An artificial neuron — linear case

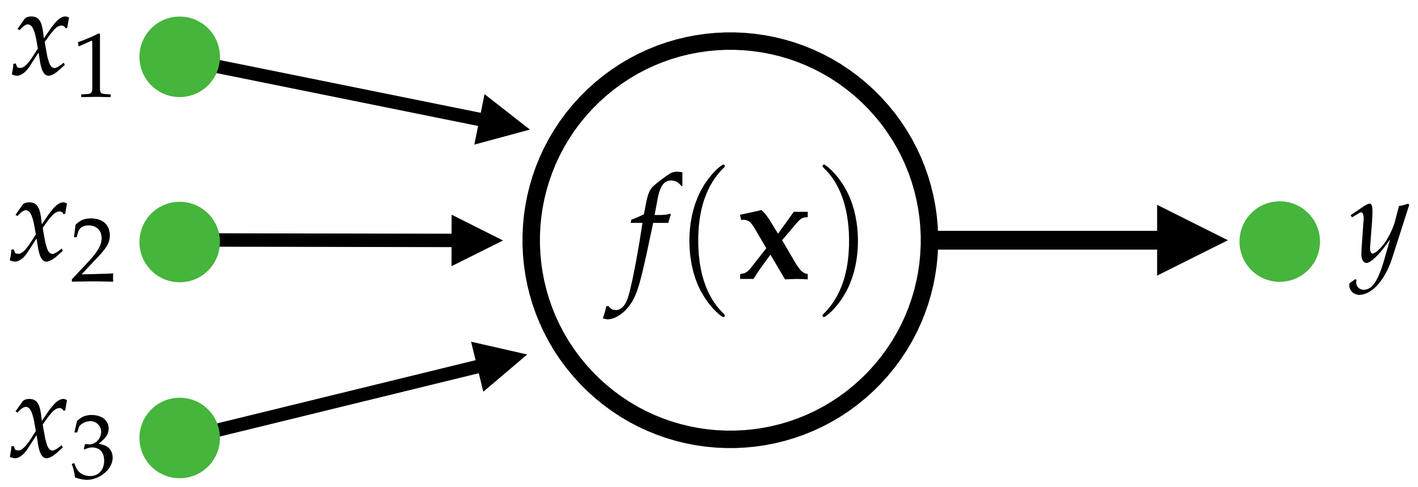


"*Weights*" "*Bias*"

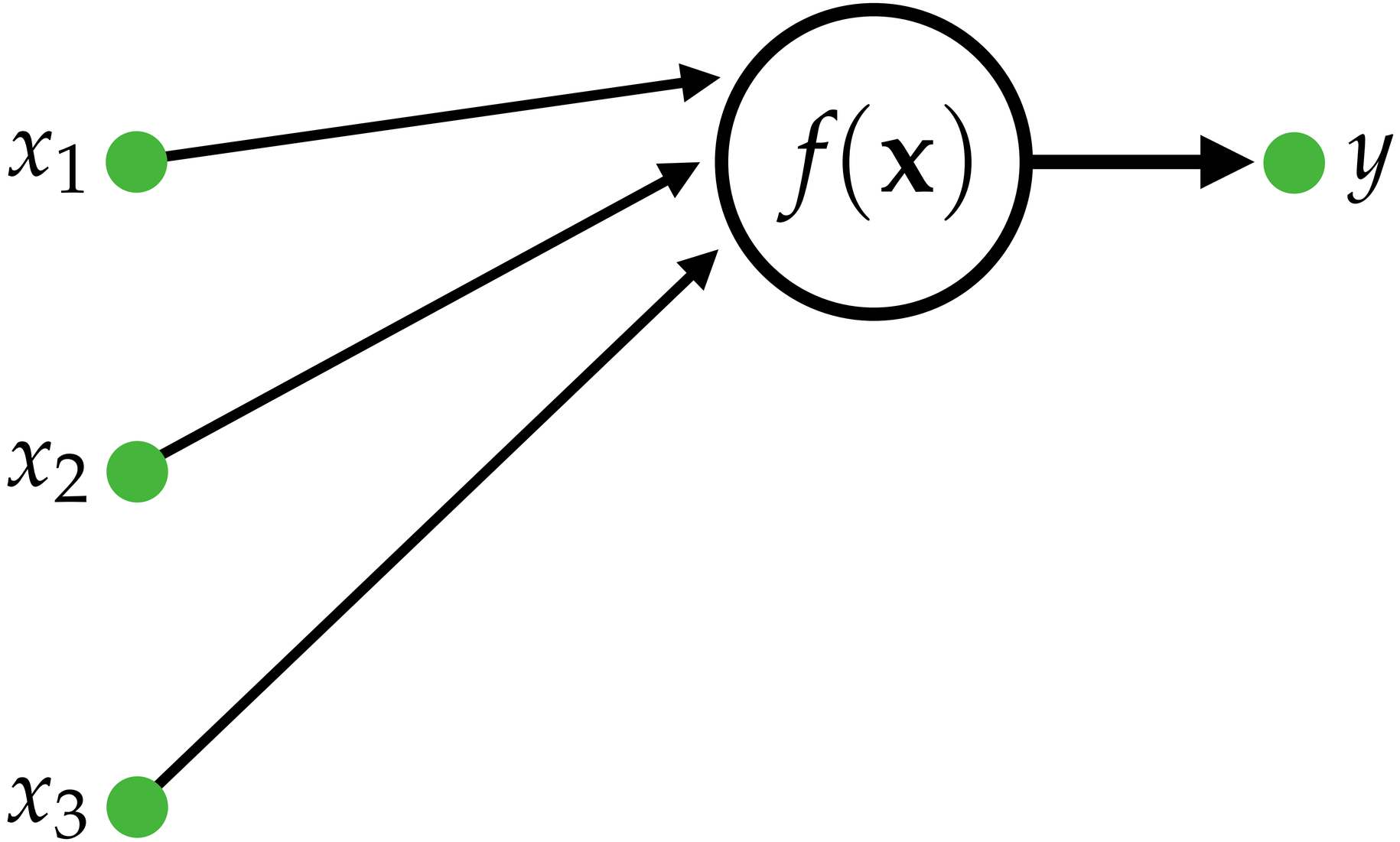
$$f(\mathbf{x}) = w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + b = \mathbf{w} \cdot \mathbf{x} + b$$

Unknown parameters. We must *optimize* ("train") the network to find good values for these.

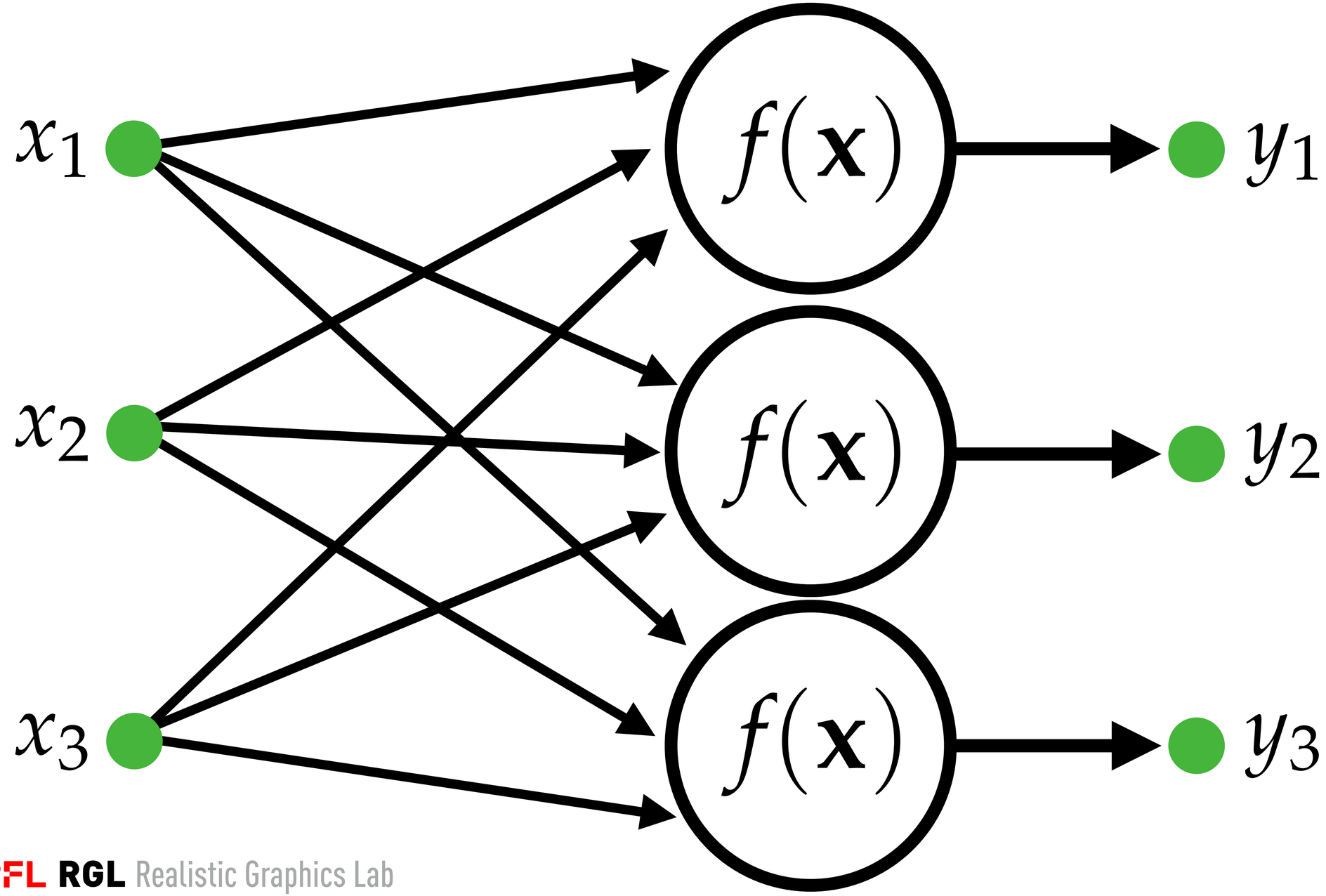
An artificial neuron — linear case



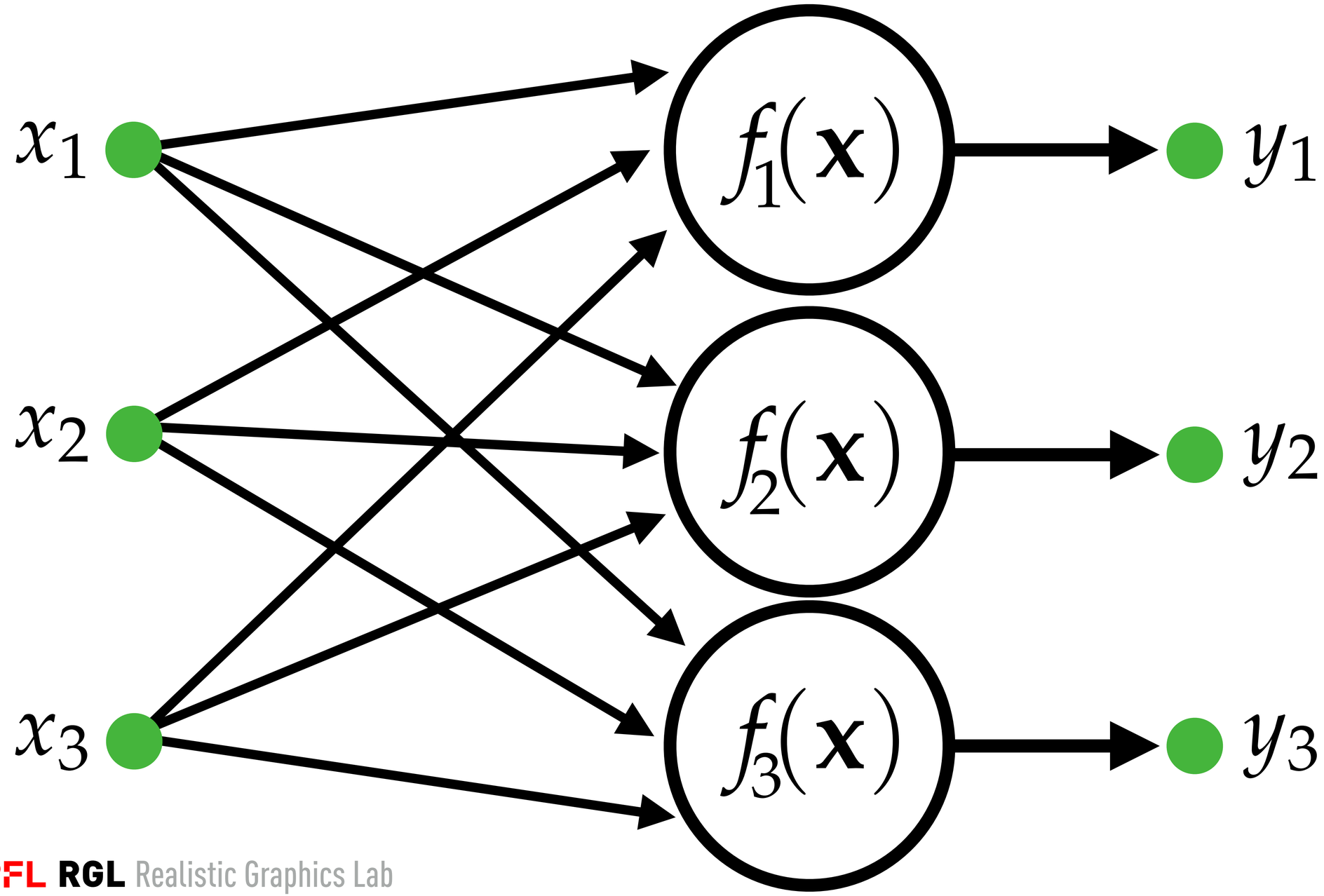
An artificial neuron — linear case



An artificial neuron — linear case



An artificial neuron — linear case



An artificial neuron — linear case

$f_1(\mathbf{x})$

$f_2(\mathbf{x})$

$f_3(\mathbf{x})$

An artificial neuron — linear case

$$f_1(\mathbf{x}) = \mathbf{w}_1 \cdot \mathbf{x} + b_1$$

$$f_2(\mathbf{x}) = \mathbf{w}_2 \cdot \mathbf{x} + b_2$$

$$f_3(\mathbf{x}) = \mathbf{w}_3 \cdot \mathbf{x} + b_3$$

An artificial neuron — linear case

$$f_1(\mathbf{x}) = \mathbf{w}_1 \cdot \mathbf{x} + b_1$$

$$f_2(\mathbf{x}) = \mathbf{w}_2 \cdot \mathbf{x} + b_2$$

$$f_3(\mathbf{x}) = \mathbf{w}_3 \cdot \mathbf{x} + b_3$$

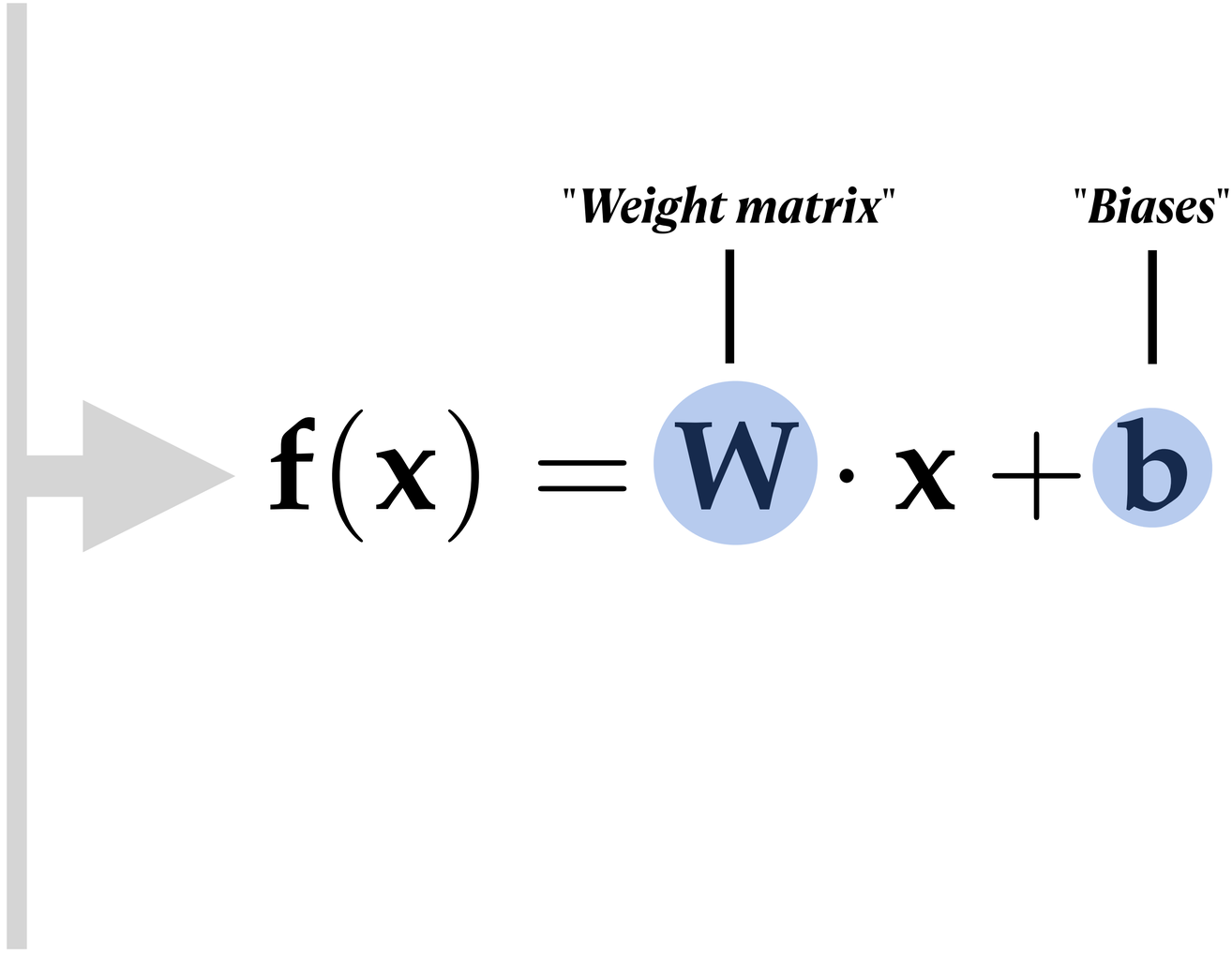
$$\mathbf{f}(\mathbf{x}) = \mathbf{W} \cdot \mathbf{x} + \mathbf{b}$$

An artificial neuron — linear case

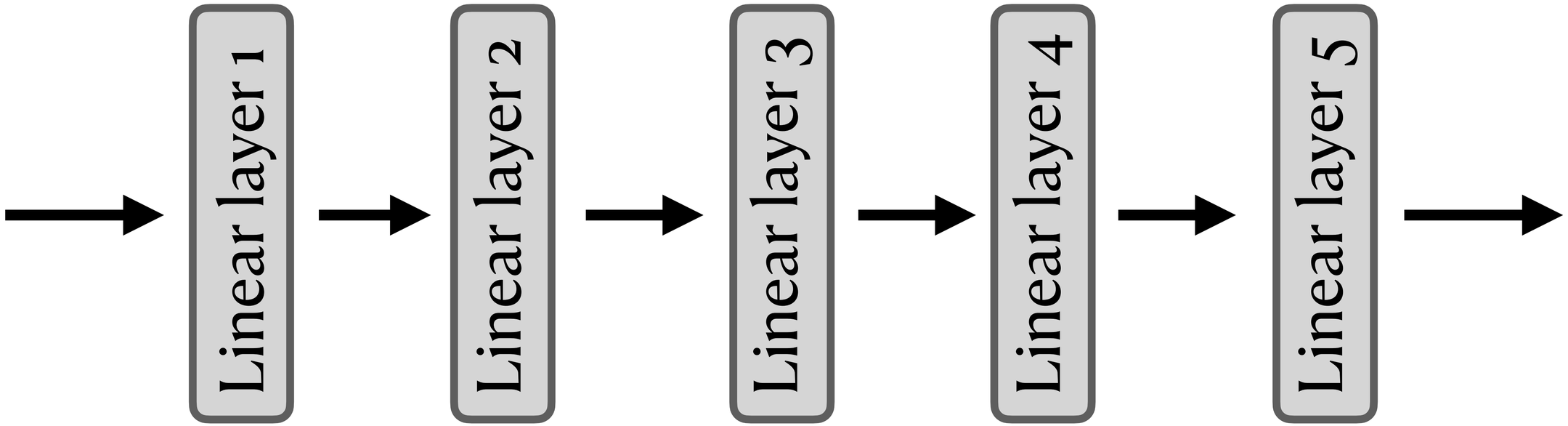
$$f_1(\mathbf{x}) = \mathbf{w}_1 \cdot \mathbf{x} + b_1$$

$$f_2(\mathbf{x}) = \mathbf{w}_2 \cdot \mathbf{x} + b_2$$

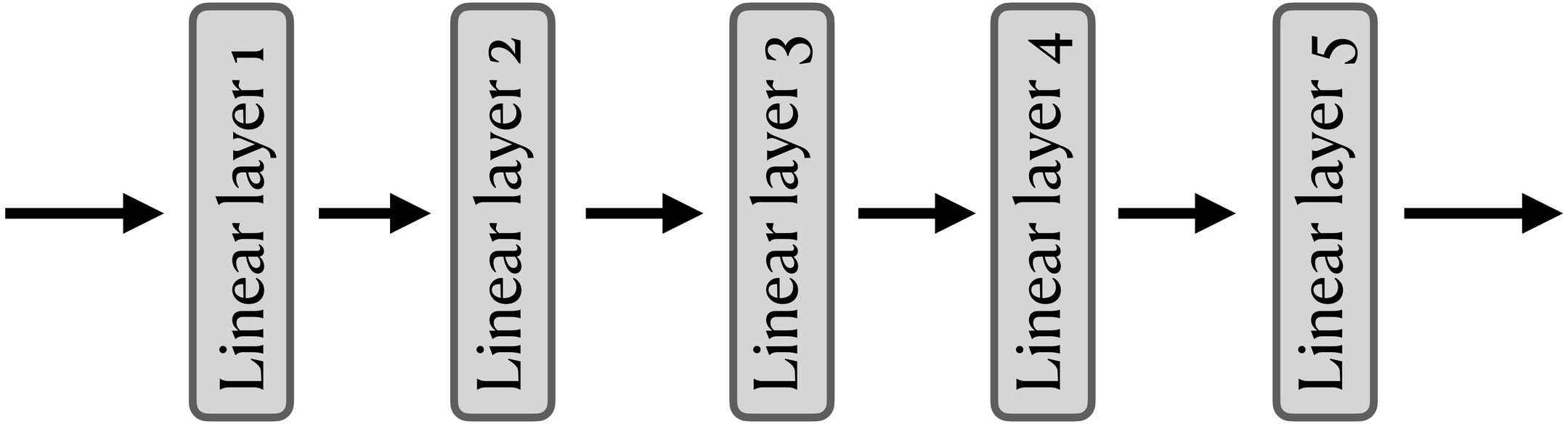
$$f_3(\mathbf{x}) = \mathbf{w}_3 \cdot \mathbf{x} + b_3$$



The linear/affine case isn't interesting!

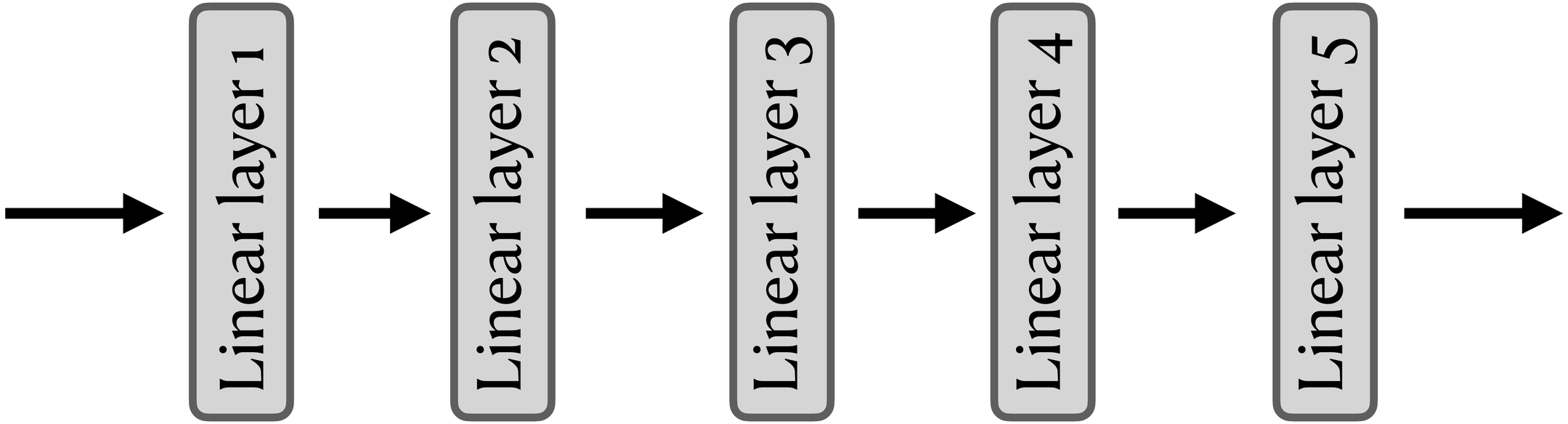


The linear/affine case isn't interesting!



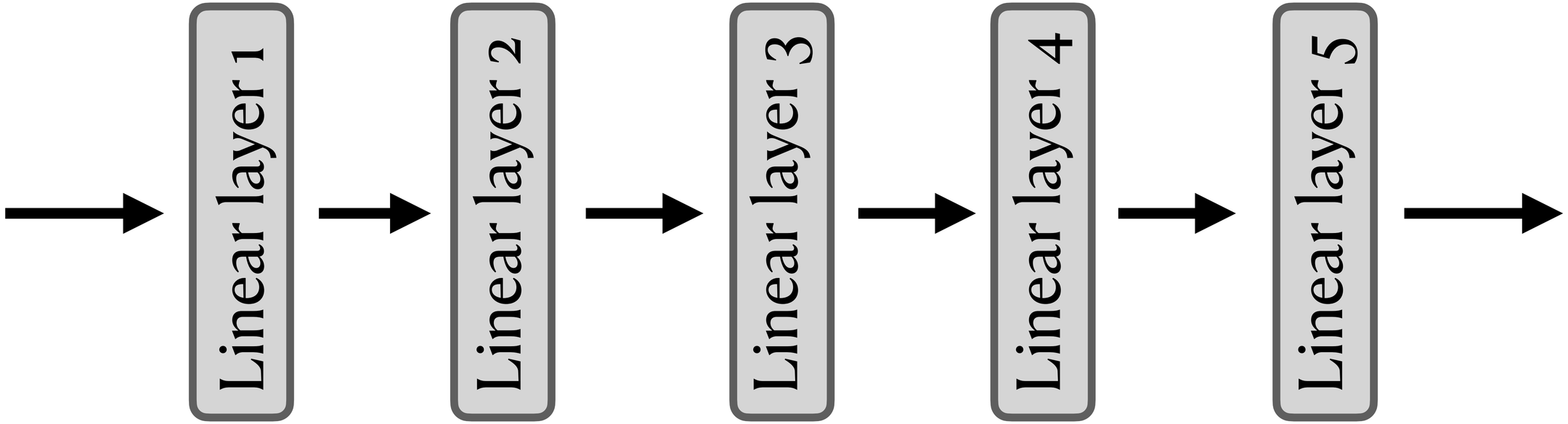
$$\mathbf{f}(\mathbf{x}) = \mathbf{W}_1 \cdot \mathbf{x} + \mathbf{b}_1$$

The linear/affine case isn't interesting!



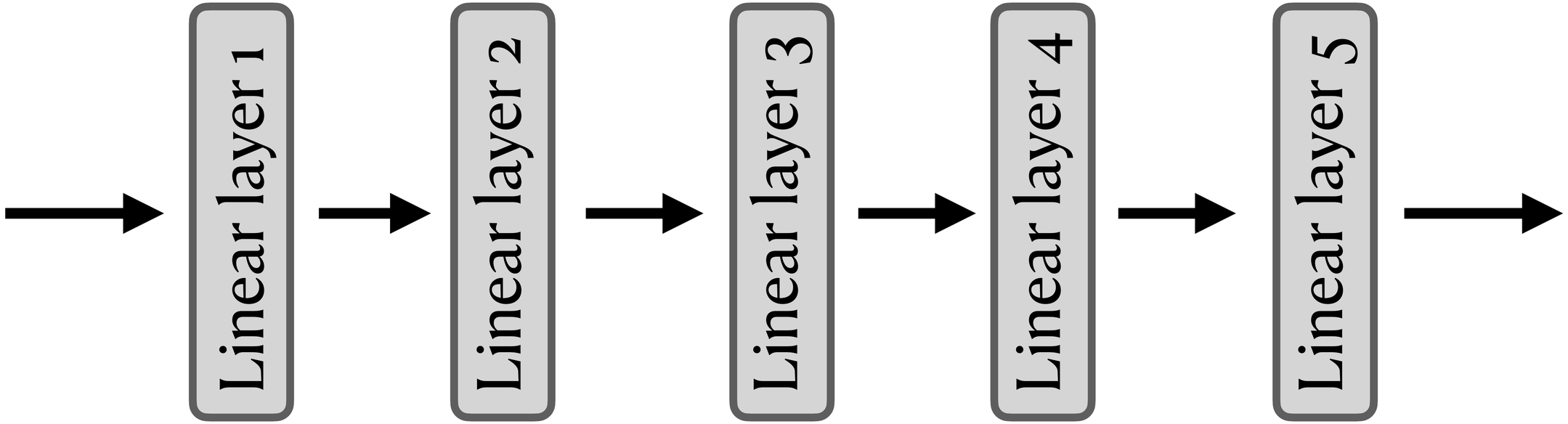
$$\mathbf{f}(\mathbf{x}) = \mathbf{W}_2(\mathbf{W}_1 \cdot \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$$

The linear/affine case isn't interesting!

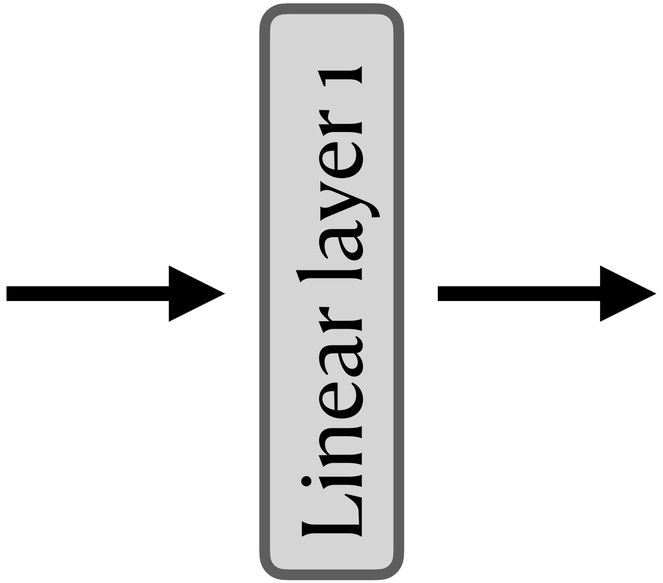


$$\mathbf{f}(\mathbf{x}) = \cdots \mathbf{W}_3(\mathbf{W}_2(\mathbf{W}_1 \cdot \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2) + \mathbf{b}_3 \cdots$$

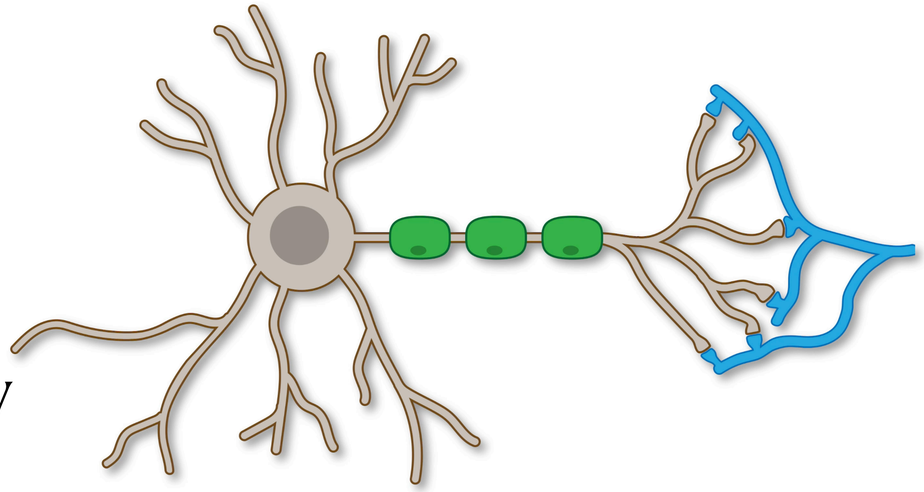
The linear/affine case isn't interesting!



The following is *equivalent*:



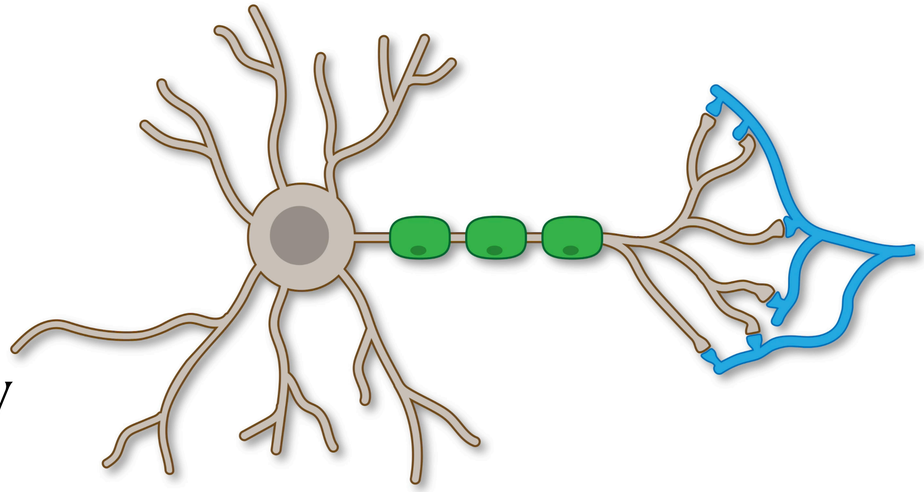
Activation functions



- Not all stimulation causes the neuron to "fire" in a proportional way

- **Activation functions** insert nonlinearities into the network, which are key to their expressiveness.

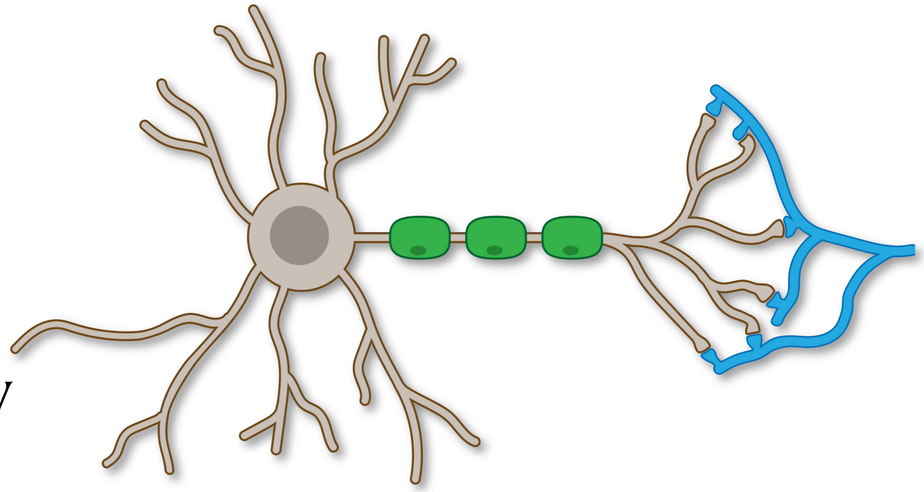
Activation functions



- Not all stimulation causes the neuron to "fire" in a proportional way

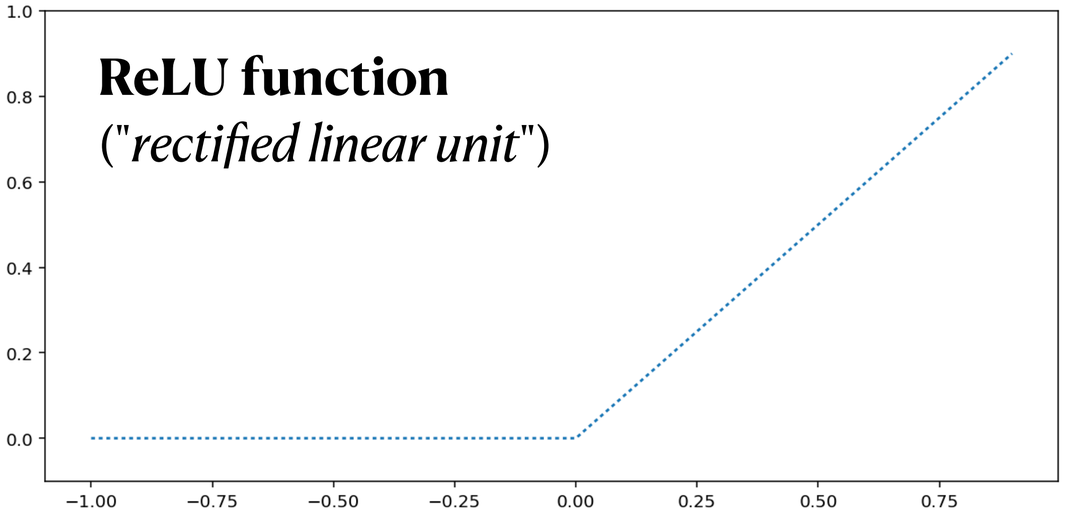
- **Activation functions** insert nonlinearities into the network, which are key to their expressiveness.

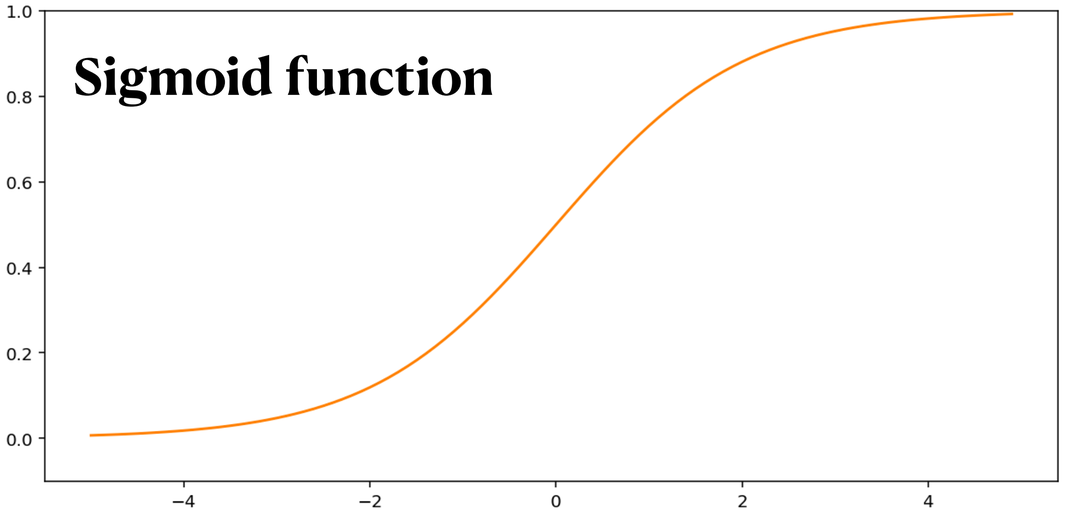
Activation functions



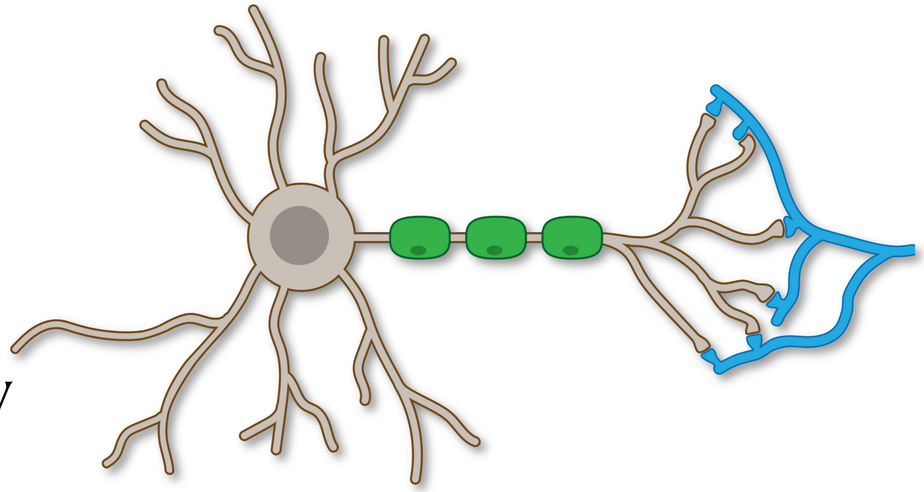
- Not all stimulation causes the neuron to "fire" in a proportional way

- **Activation functions** insert nonlinearities into the network, which are key to their expressiveness.



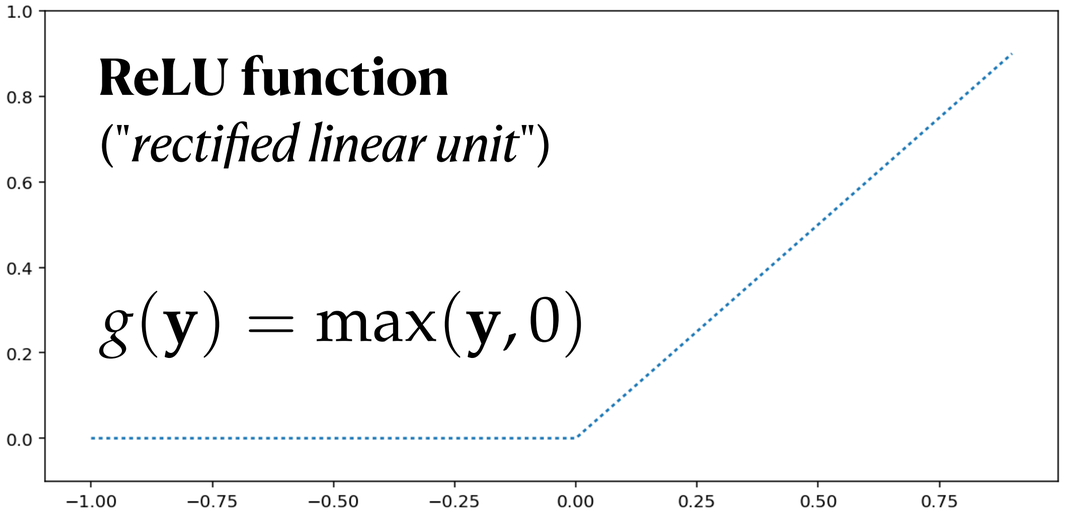


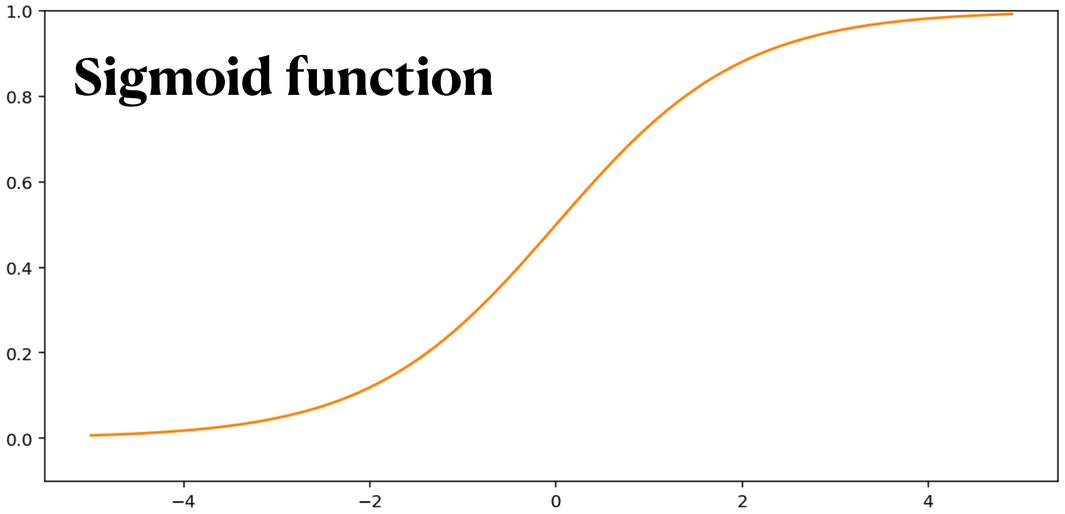
Activation functions



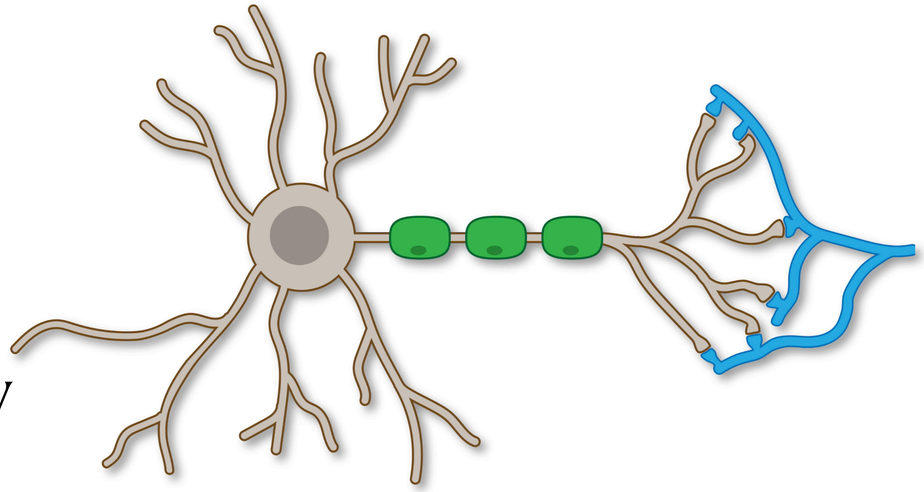
- Not all stimulation causes the neuron to "fire" in a proportional way

- **Activation functions** insert nonlinearities into the network, which are key to their expressiveness.



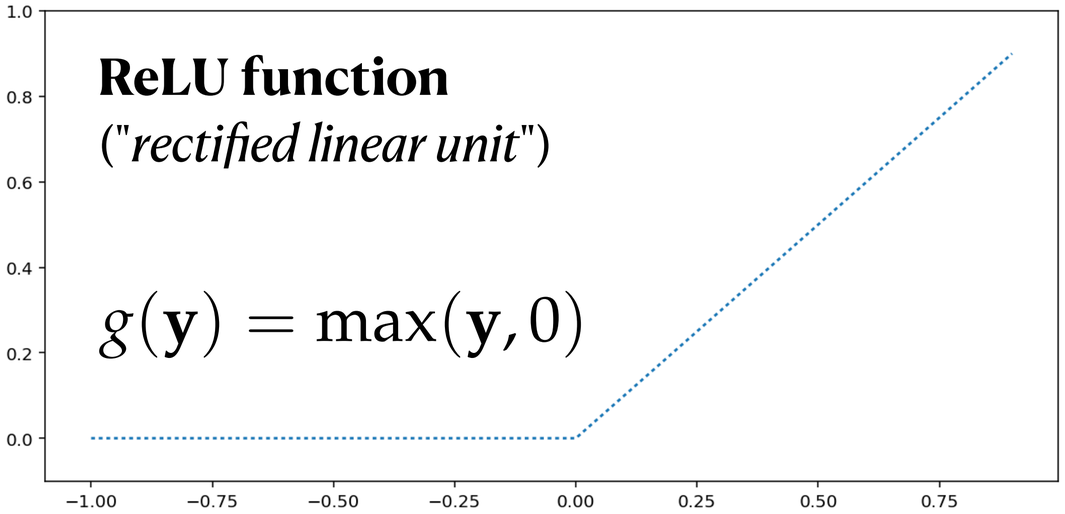


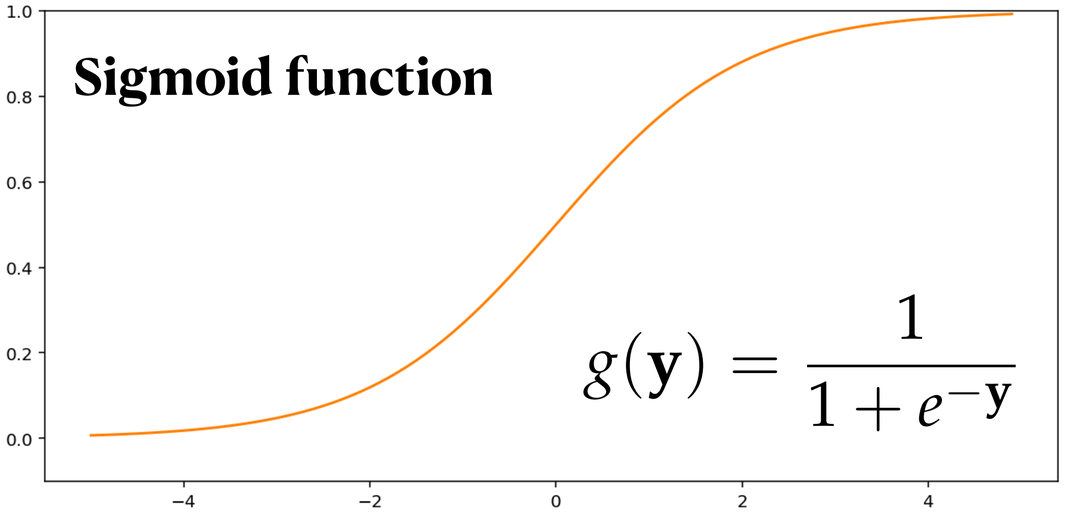
Activation functions



- Not all stimulation causes the neuron to "fire" in a proportional way

- **Activation functions** insert nonlinearities into the network, which are key to their expressiveness.





Dense layers with activation function

$$\mathbf{y} = f(\mathbf{x}) = \mathbf{W} \cdot \mathbf{x} + \mathbf{b}$$

Affine transformation

Dense layers with activation function

$$\mathbf{y} = f(\mathbf{x}) = \mathbf{W} \cdot \mathbf{x} + \mathbf{b}$$ Affine transformation

$$\mathbf{z} = g(\mathbf{y})$$ Nonlinearity

Dense layers with activation function

$$\mathbf{y} = f(\mathbf{x}) = \mathbf{W} \cdot \mathbf{x} + \mathbf{b}$$

Affine transformation

$$\mathbf{z} = g(\mathbf{y})$$

Nonlinearity

1024 neurons, 1024 inputs: **1049600** trainable parameters for just 1 layer 😨

Dense layers with activation function

$$\mathbf{y} = f(\mathbf{x}) = \mathbf{W} \cdot \mathbf{x} + \mathbf{b}$$

Affine transformation

$$\mathbf{z} = g(\mathbf{y})$$

Nonlinearity

1024 neurons, 1024 inputs: **1049600** trainable parameters for just 1 layer 😨

Most common setup: affine layer with ReLU activation function:

$$\mathbf{z} = \max(\mathbf{W} \cdot \mathbf{x} + \mathbf{b}, 0)$$

How do we compute the weights?

How do we compute the weights?

Labeled input/output data

$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \ldots$ $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4, \ldots$

How do we compute the weights?

Labeled input/output data

$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \ldots$ $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4, \ldots$

$$\mathrm{NN}(\mathbf{x_1}) \approx \mathbf{y}_1?$$

How do we compute the weights?

Labeled input/output data

$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \ldots$ $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4, \ldots$

$\mathrm{NN}(\mathbf{x_1}) \approx \mathbf{y}_1$?

These are unlikely to agree at first.

How do we compute the weights?

Labeled input/output data

$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \ldots$ $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4, \ldots$

$\mathrm{NN}(\mathbf{x_1}) \approx \mathbf{y}_1$? These are unlikely to agree at first.

Quantify discrepancy using a **loss function**:

How do we compute the weights?

Labeled input/output data

$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \ldots$ $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4, \ldots$

$\mathrm{NN}(\mathbf{x_1}) \approx \mathbf{y}_1$?

These are unlikely to agree at first.

Quantify discrepancy using a **loss function**:

$$L(\mathrm{NN}(\mathbf{x_1}), \mathbf{y}_1)$$

How do we compute the weights?

Labeled input/output data

$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \ldots$ $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4, \ldots$

$\mathrm{NN}(\mathbf{x_1}) \approx \mathbf{y}_1 ?$ These are unlikely to agree at first.

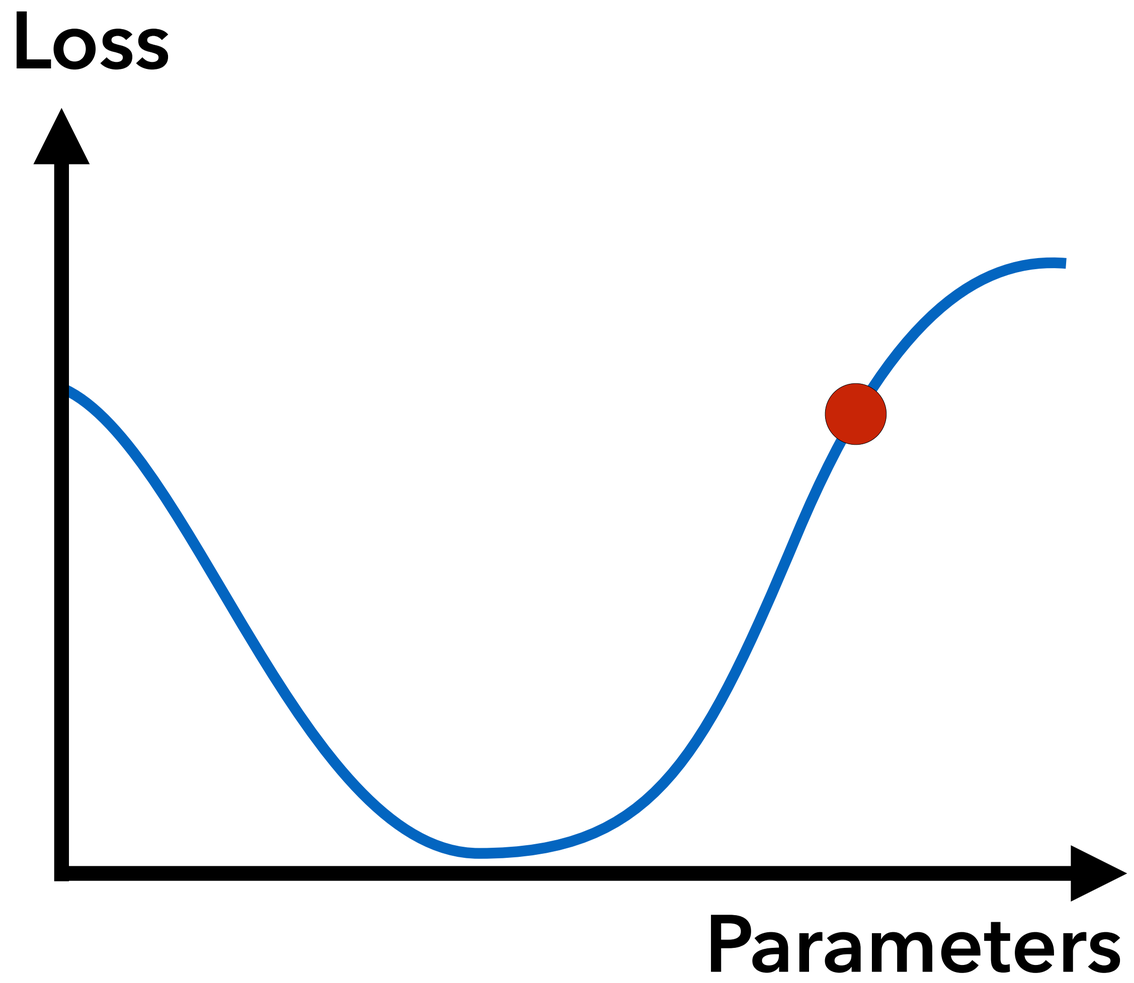
Quantify discrepancy using a **loss function**:

Example: L_2 loss

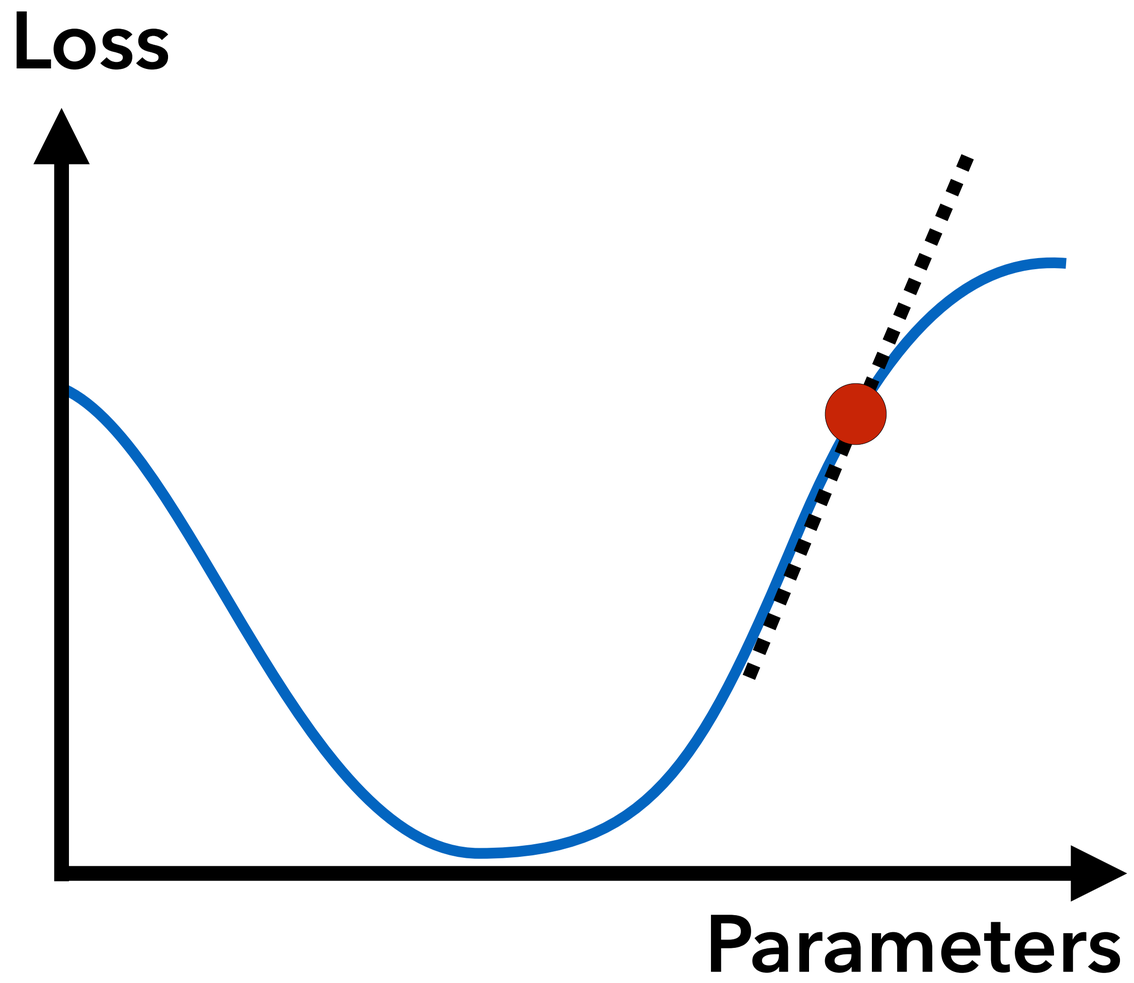
$$L(\mathrm{NN}(\mathbf{x_1}), \mathbf{y}_1) = \|\mathrm{NN}(\mathbf{x_1}) - \mathbf{y}_1\|_2$$

Gradient descent

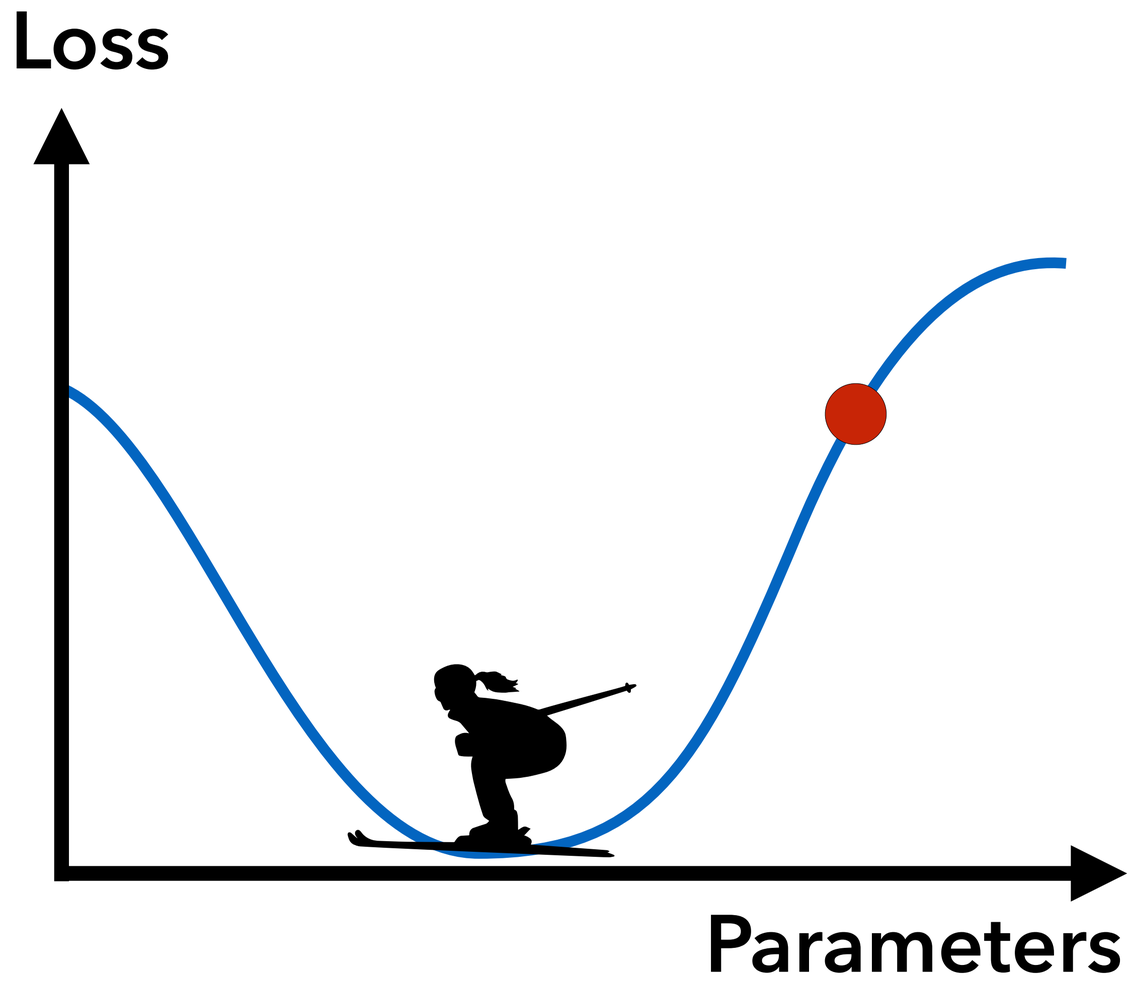
Gradient descent



Gradient descent



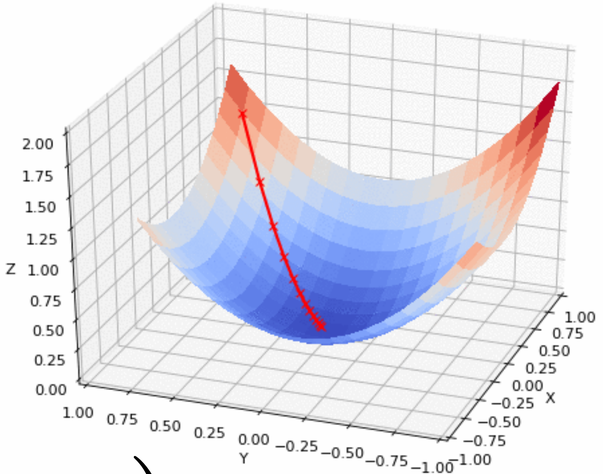
Gradient descent



Gradient descent

(More advanced optimizers often used — we will discuss some next week)

- Every iteration takes a step in the negative gradient direction

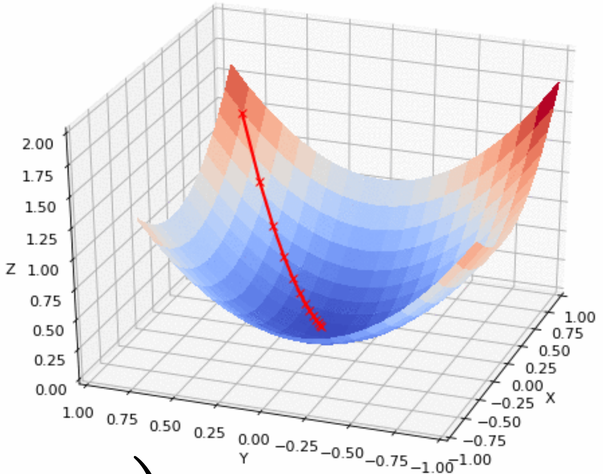


$$\mathbf{p}_{\text{next}} = \mathbf{p} - \eta \frac{\partial}{\partial \mathbf{p}} \sum_{i=1}^{n} L(\text{NN}(\mathbf{p}, \mathbf{x_i}), \mathbf{y_i})$$

Gradient descent

(More advanced optimizers often used — we will discuss some next week)

- Every iteration takes a step in the negative gradient direction



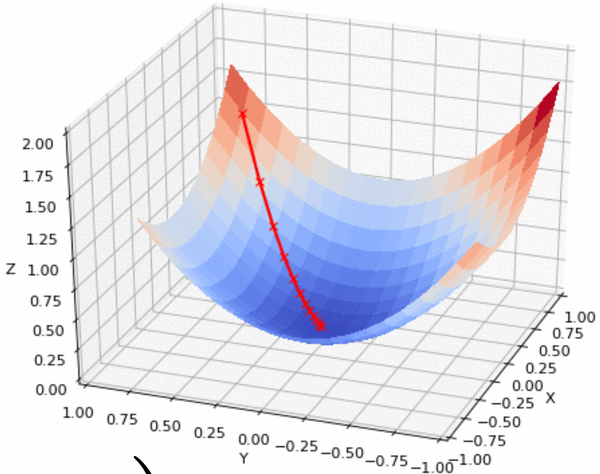
Step size

$$\mathbf{p}_{\text{next}} = \mathbf{p} - \eta \frac{\partial}{\partial \mathbf{p}} \sum_{i=1}^{n} L(\text{NN}(\mathbf{p}, \mathbf{x_i}), \mathbf{y_i})$$

Gradient descent

(More advanced optimizers often used — we will discuss some next week)

- Every iteration takes a step in the negative gradient direction



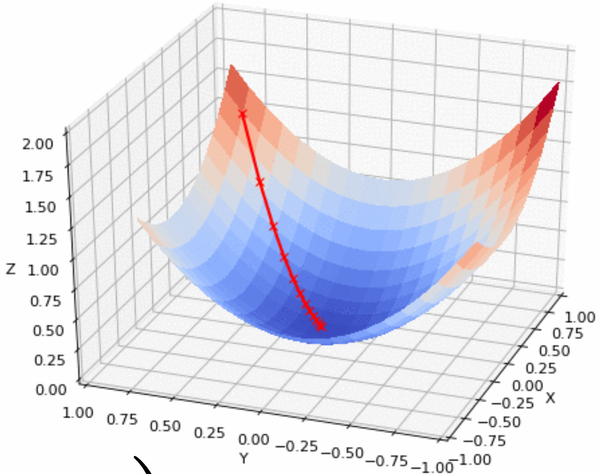
All network parameters Step size

$$\mathbf{p}_{\text{next}} = \mathbf{p} - \eta \frac{\partial}{\partial \mathbf{p}} \sum_{i=1}^{n} L(\text{NN}(\mathbf{p}, \mathbf{x_i}), \mathbf{y_i})$$

Gradient descent

(More advanced optimizers often used — we will discuss some next week)

- Every iteration takes a step in the negative gradient direction



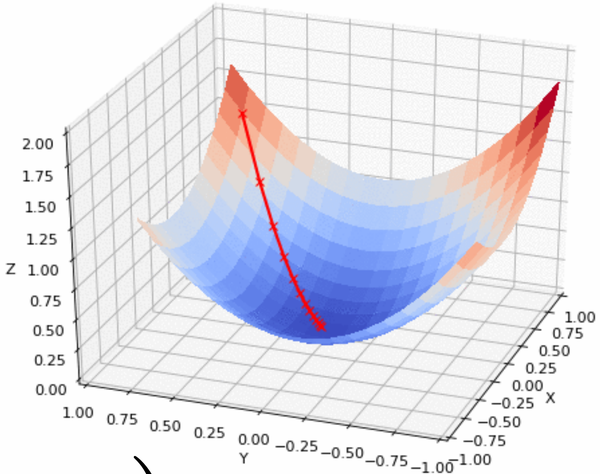
All network parameters — Step size — Easy: it's "just" autodiff (AD)

$$\mathbf{p}_{\text{next}} = \mathbf{p} - \eta \frac{\partial}{\partial \mathbf{p}} \sum_{i=1}^{n} L(\text{NN}(\mathbf{p}, \mathbf{x_i}), \mathbf{y_i})$$

Gradient descent

(More advanced optimizers often used — we will discuss some next week)

- Every iteration takes a step in the negative gradient direction



All network parameters — Step size — Easy: it's "just" autodiff (AD)

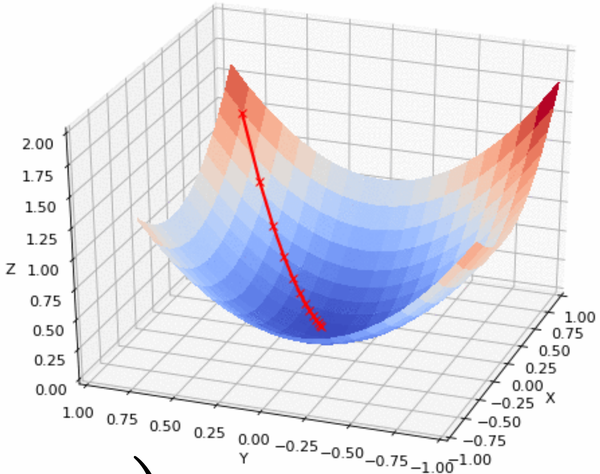
$$\mathbf{p}_{\text{next}} = \mathbf{p} - \eta \frac{\partial}{\partial \mathbf{p}} \sum_{i=1}^{n} L(\text{NN}(\mathbf{p}, \mathbf{x_i}), \mathbf{y_i})$$

- **Downside**: each gradient step is quite expensive since we must visit every data point

Gradient descent

(More advanced optimizers often used — we will discuss some next week)

- Every iteration takes a step in the negative gradient direction



All network parameters — Step size — Easy: it's "just" autodiff (AD)

$$\mathbf{p}_{\text{next}} = \mathbf{p} - \eta \frac{\partial}{\partial \mathbf{p}} \sum_{i=1}^{n} L(\text{NN}(\mathbf{p}, \mathbf{x_i}), \mathbf{y_i})$$

- **Downside**: each gradient step is quite expensive since we must visit every data point
- **Solution**: pick *random* subset (even just 1 works).
  This is called Stochastic Gradient Descent (SGD).

High dimensional spaces

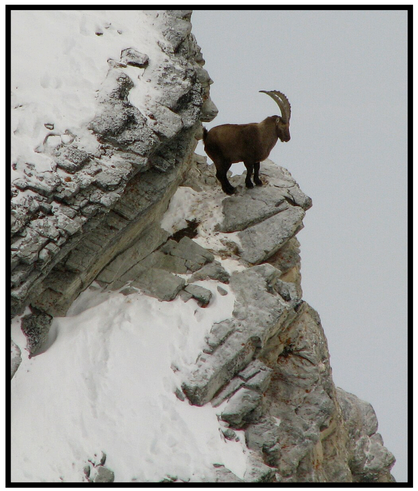
- Derivative are crucial especially in **higher dimensions**.
  - In 1-D, can move in two directions
  - in N-D can move in 2^N "diagonal" directions alone. That's just *too many* to check.
  - The gradient points into the direction of ascent and maps the behavior of the function locally.
  - Foundation of all breakthroughts in ML in the last years. You *cannot* train a neural network without gradients.

MidJourney: A sign post with many different hikes in Switzerland.

Underfitting vs overfitting

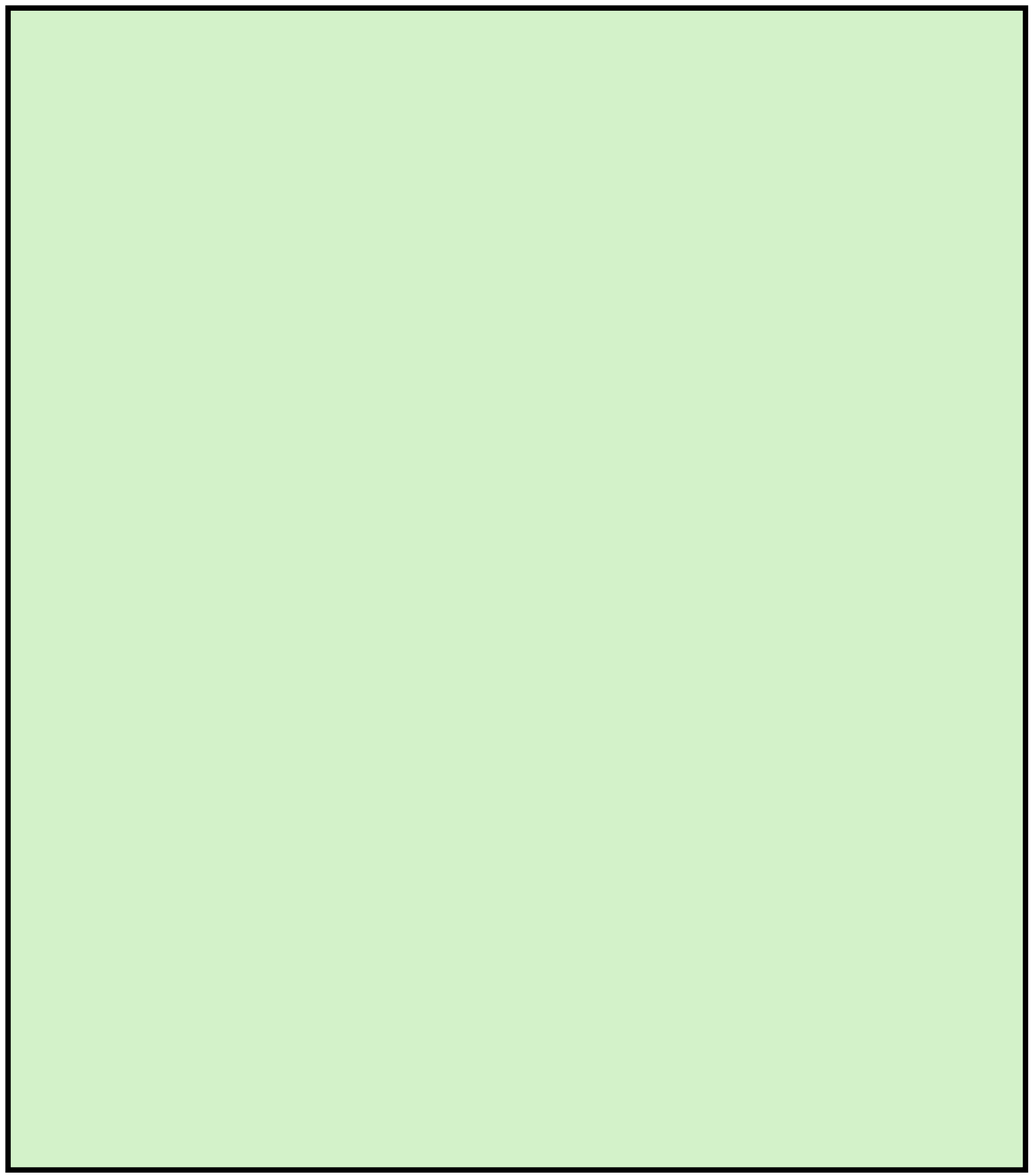
Neural nets often have enough coefficients that they can in principle memorize the entire dataset.

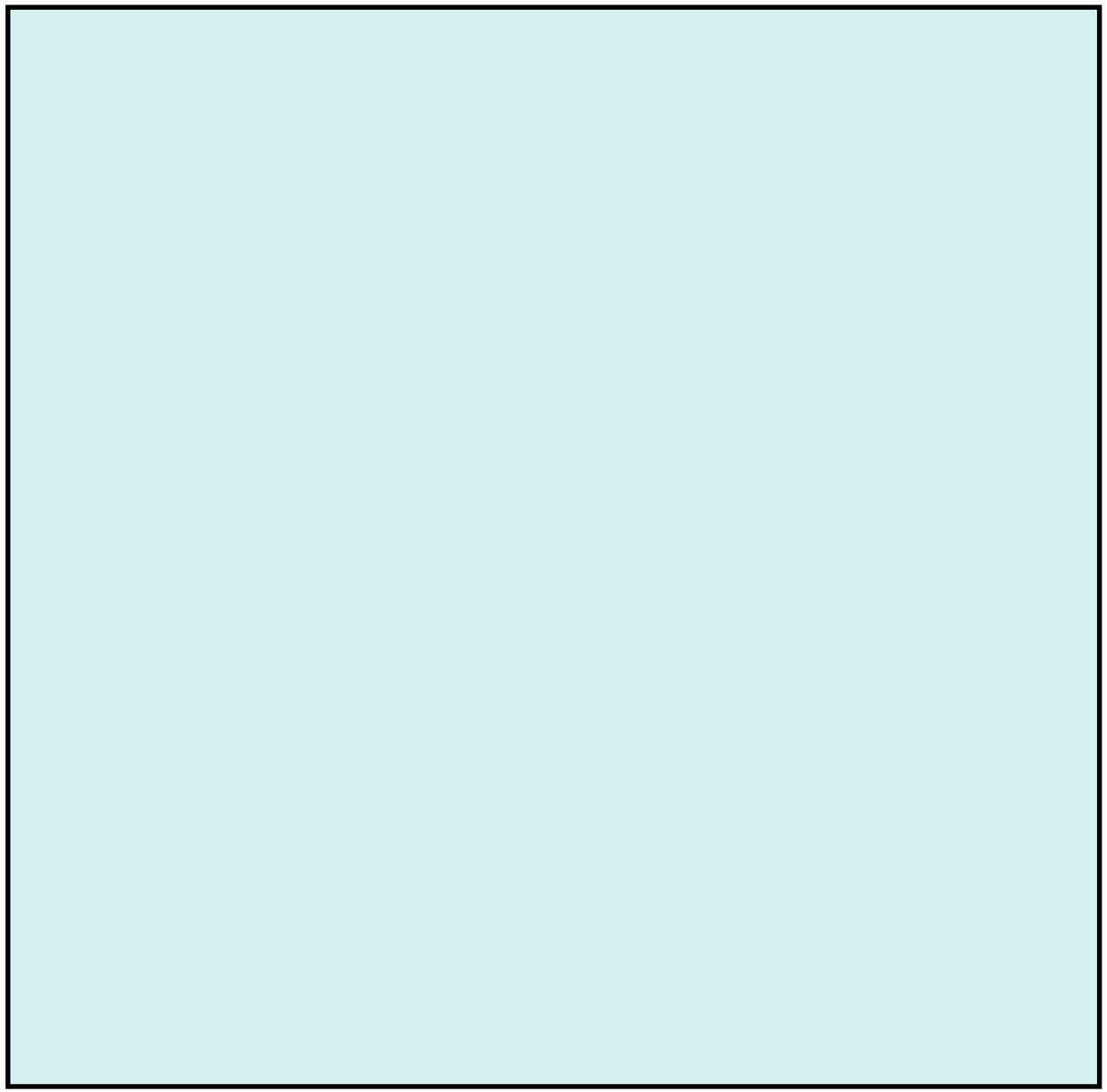
Dangers of training with data



Dangers of training with data

Dangers of training with data



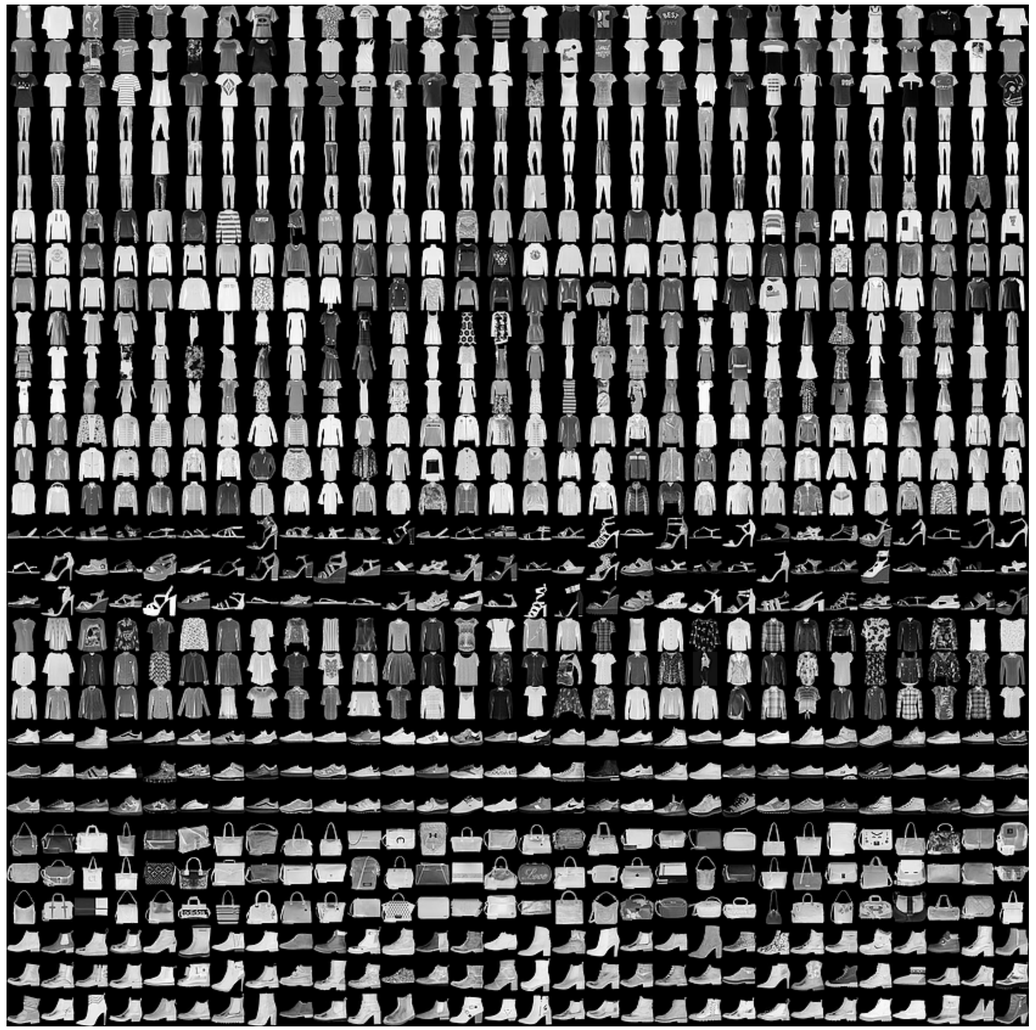


A benchmark problem

"MNIST" and "Fashion-MNIST"

A benchmark problem

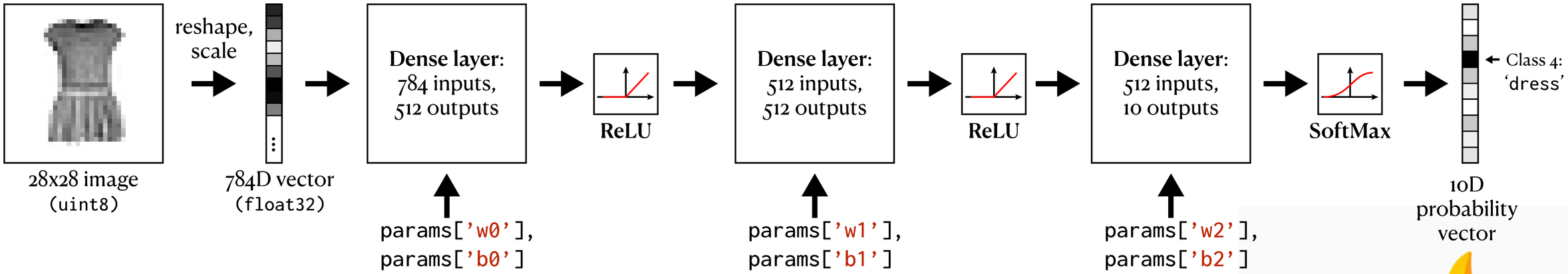
"MNIST" and "Fashion-MNIST"



zalando


60'000 training images
10'000 testing images

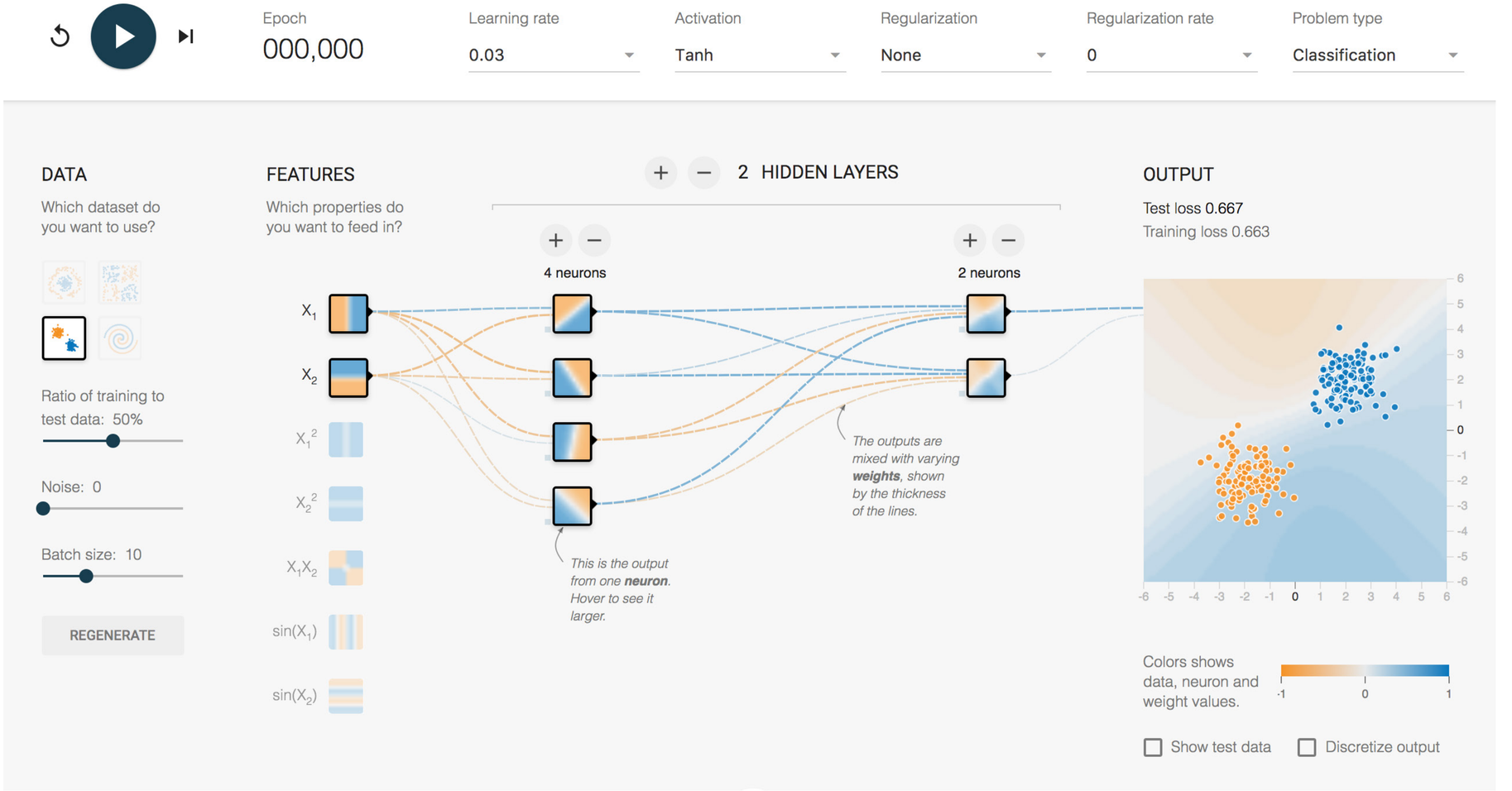
Homework assignment 4: MyTorch



- Forward & reverse-mode AD framework
- A simple neural network with 3 layers (discussed next lecture)
- Reaches ~85% accuracy on Fashion-MNIST
- All built by yourself using only NumPy!

MyTorch

Neural Network Playground

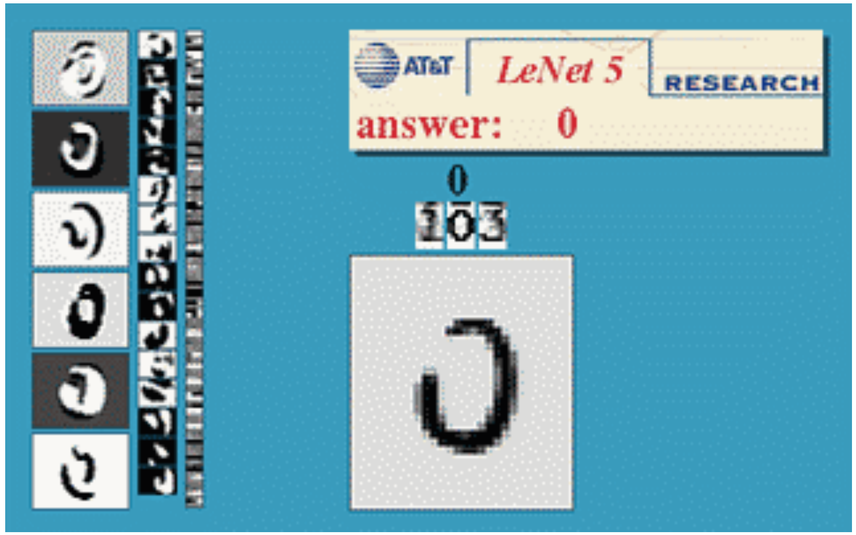


http://playground.tensorflow.org

History of neural nets

“Learning representations by back-propagating errors”

- Reverse-mode AD (in general): Linnainmaa (1970)
- For NN: Werbos (developed 1974, published 1982)
- Popularized: Rumelhart, Hinton, and Williams



Deep neural network

- LeCun, LeNet (1989)

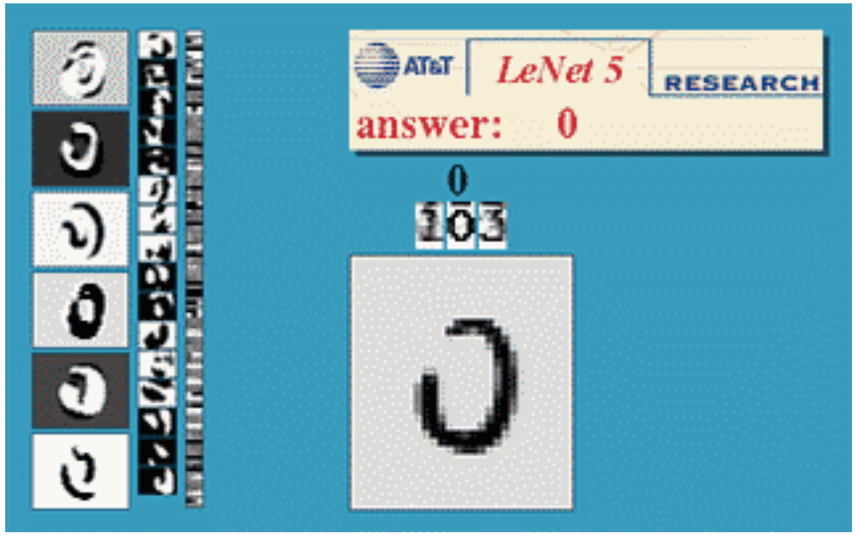
Geoffrey Hinton: neural networks were *unsuccessful* in the 80s/90s because

- **Data**: Labeled datasets were thousands of times too small.
- **Computers**: … were millions of times too slow.
- **Nonlinearity**: poor choice of activation function.
- **Starting point**: weights weren’t initialized in a good way.

History of neural nets

“Learning representations by back-propagating errors”

- Reverse-mode AD (in general): Linnainmaa (1970)
- For NN: Werbos (developed 1974, published 1982)
- Popularized: Rumelhart, Hinton, and Williams



Deep neural network

- LeCun, LeNet (1989)

Geoffrey Hinton: neural networks were *unsuccessful* in the 80s/90s because

- **Data**: Labeled datasets were thousands of times too small.
- **Computers**: … were millions of times too slow.
- **Nonlinearity**: poor choice of activation function.
- **Starting point**: weights weren’t initialized in a good way.

We don't actually know why they "work"

- What we **know**.
  - How all the coefficients were obtained (it is a deterministic calculation after all).
  - Which neural network components work *well*, and which of them work *well together*.
  - When one ingredient works better than another, we can sometimes find a nice mathematical intuition motivating why that is the case. (*the converse isn't always true*)

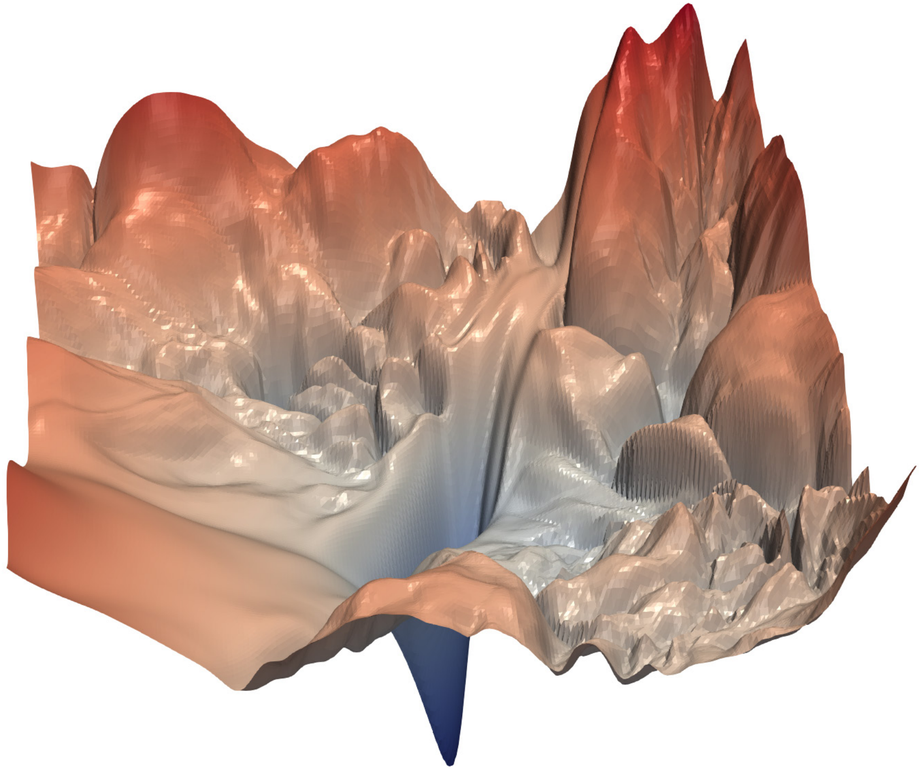
We don't actually know why they "work"

- What we **know**.
  - How all the coefficients were obtained (it is a deterministic calculation after all).
  - Which neural network components work *well*, and which of them work *well together*.
  - When one ingredient works better than another, we can sometimes find a nice mathematical intuition motivating why that is the case. (*the converse isn't always true*)

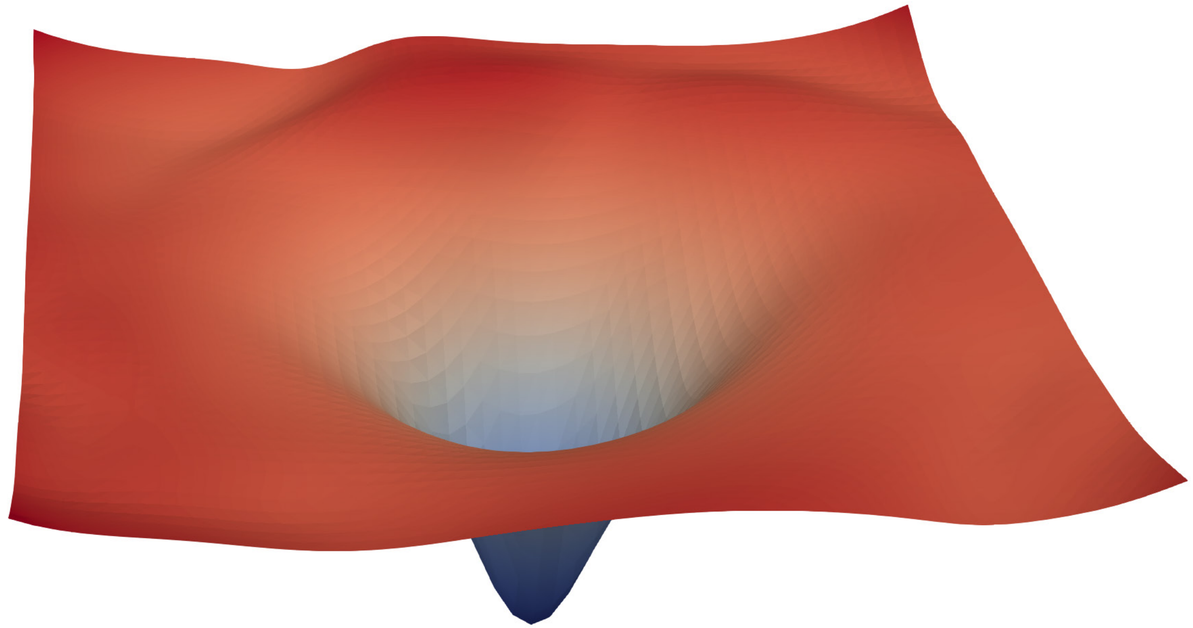
- What we **don't know**
  - What do the coefficients actually *mean?*
  - Will the network generalize to other situations? (must test)
  - **Why does the training process actually work?**

Optimization landscape

Visualization of a random 2D slices of the huge parameter space.



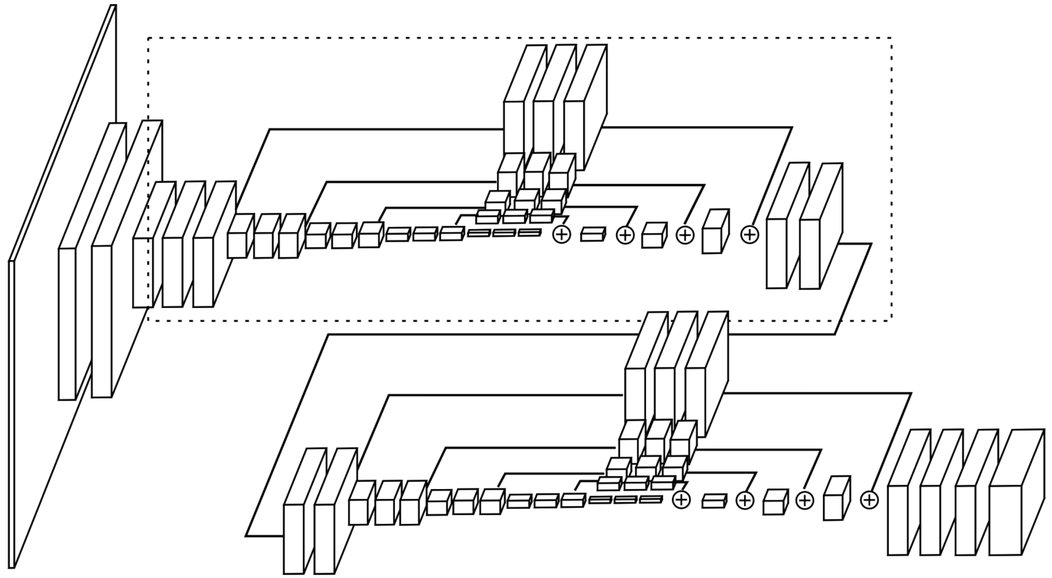
A "bad" network architecture (unstable training)



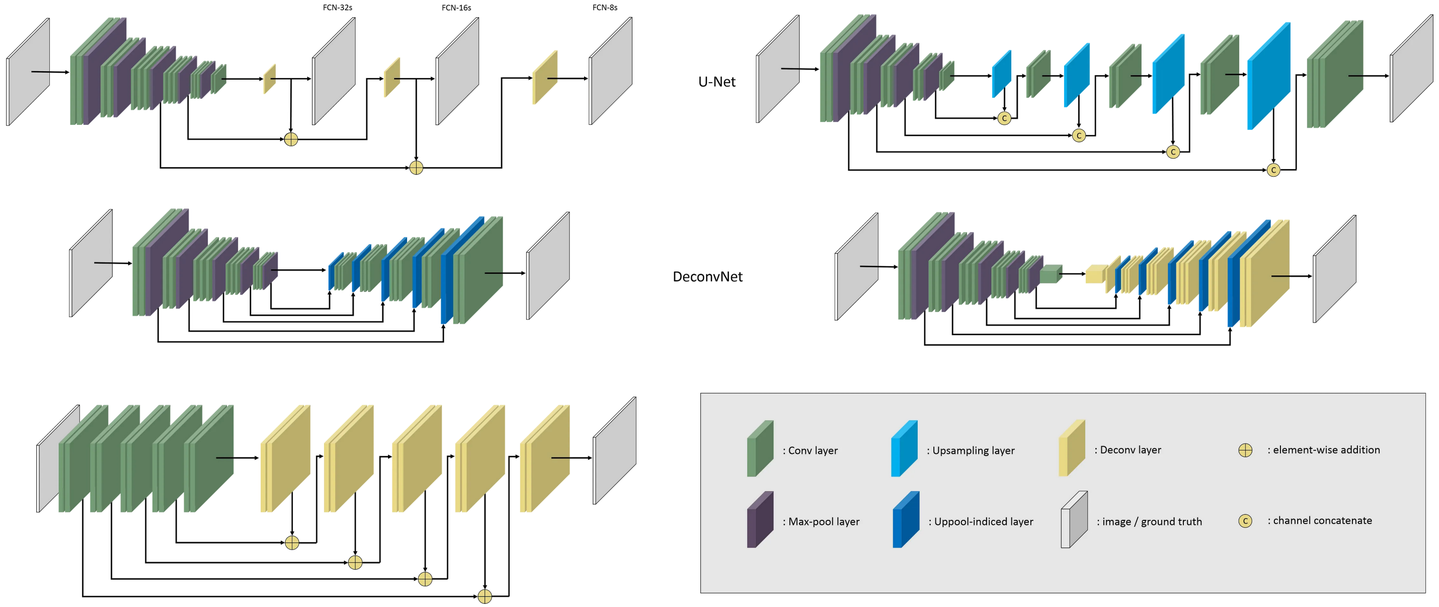
A "good" network architecture (unstable training)

From: Visualizing the Loss Landscape of Neural Nets [Li et al. 2017]

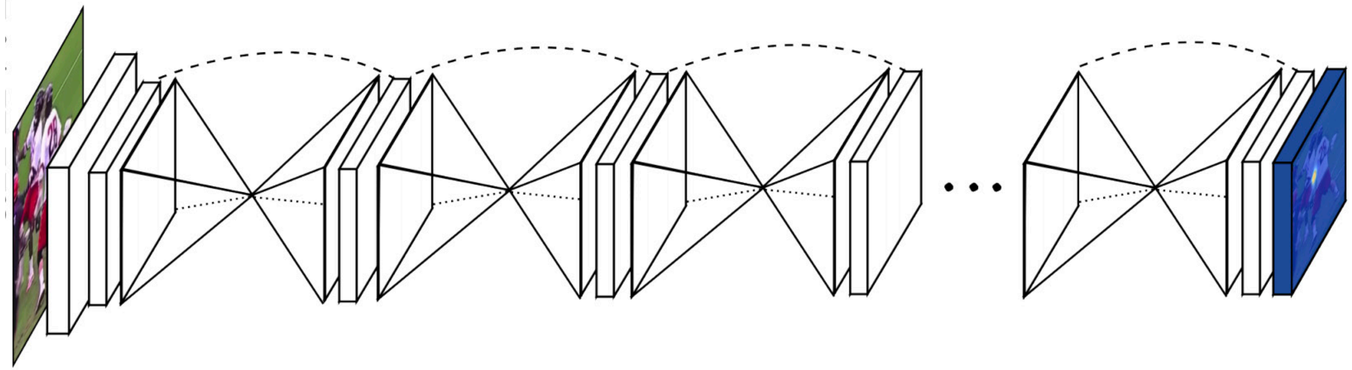
An architectural zoo

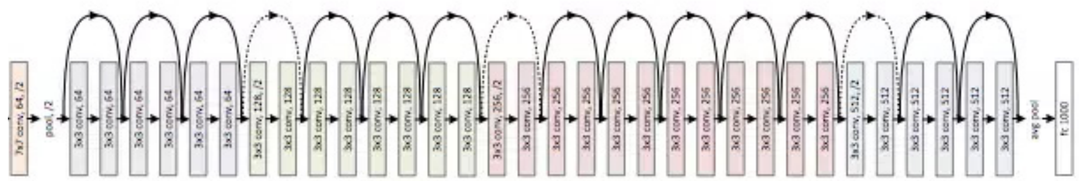


[Newell et al .2016]



[Sunner Li, Link]

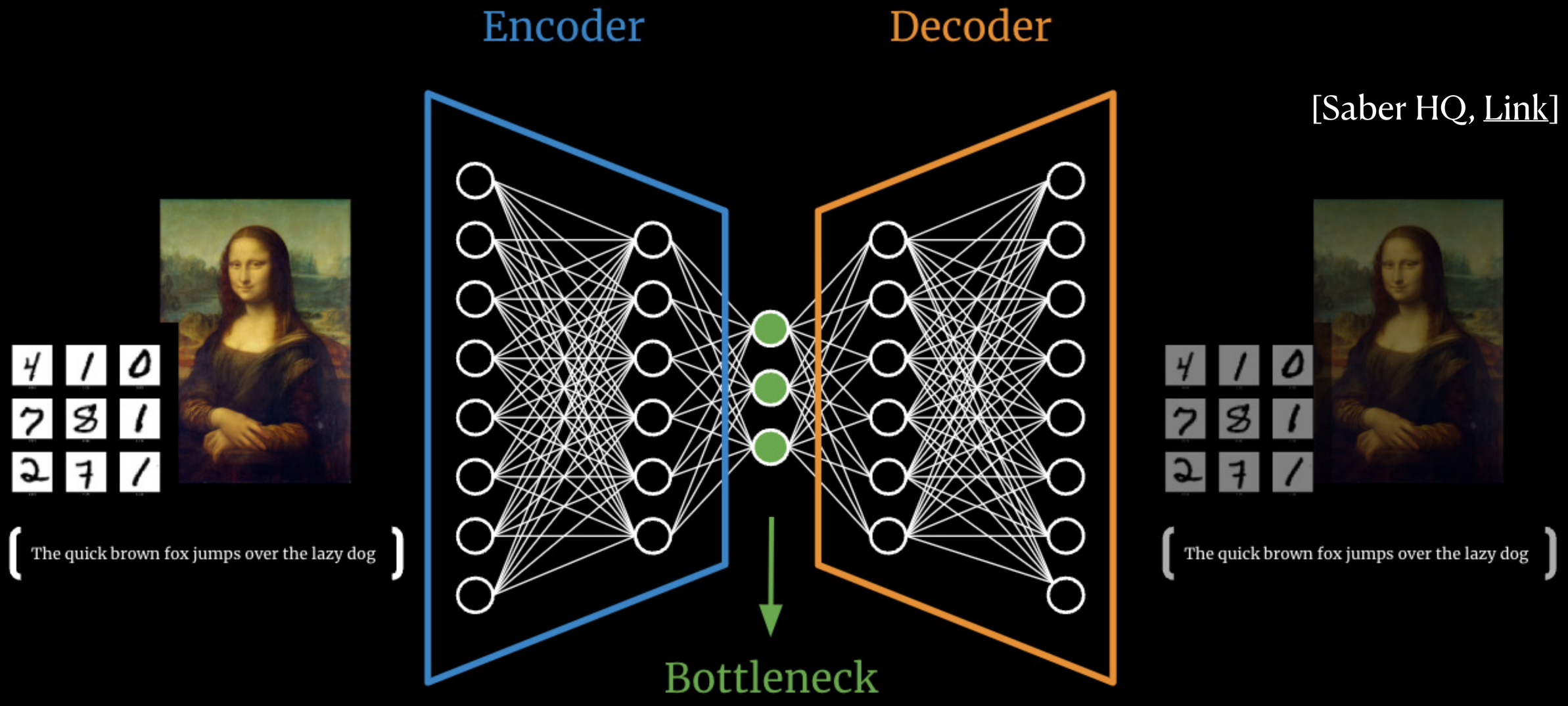




[He et al .2015]

Latent vectors and bottlenecks

Encoder-decoder architectures and related "computational games"



[Saber HQ, Link]

GPUs play an outsize role in this space

- Python code: `C = A @ B`
- PyTorch: directly forwards function call to **cuDNN**

[NVIDIA]

GPUs play an outsize role in this space

- Python code: `C = A @ B`
- PyTorch: directly forwards function call to **cuDNN**

[NVIDIA]

```
I  abbe  cuda/lib64 %  ls -lah /usr/lib/x86_64-linux-gnu/libcudnn*
lrwxrwxrwx 1 root root   27 Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_adv_infer.so.8 -> libcudnn_adv_infer.so.8.9.6
-rw-r--r-- 1 root root 120M Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_adv_infer.so.8.9.6
lrwxrwxrwx 1 root root   27 Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_adv_train.so.8 -> libcudnn_adv_train.so.8.9.6
-rw-r--r-- 1 root root 111M Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_adv_train.so.8.9.6
lrwxrwxrwx 1 root root   27 Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_cnn_infer.so.8 -> libcudnn_cnn_infer.so.8.9.6
-rw-r--r-- 1 root root 545M Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_cnn_infer.so.8.9.6
lrwxrwxrwx 1 root root   27 Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_cnn_train.so.8 -> libcudnn_cnn_train.so.8.9.6
-rw-r--r-- 1 root root 120M Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_cnn_train.so.8.9.6
lrwxrwxrwx 1 root root   27 Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_ops_infer.so.8 -> libcudnn_ops_infer.so.8.9.6
-rw-r--r-- 1 root root  87M Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_ops_infer.so.8.9.6
lrwxrwxrwx 1 root root   27 Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_ops_train.so.8 -> libcudnn_ops_train.so.8.9.6
-rw-r--r-- 1 root root  68M Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_ops_train.so.8.9.6
lrwxrwxrwx 1 root root   17 Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn.so.8 -> libcudnn.so.8.9.6
-rw-r--r-- 1 root root 139K Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn.so.8.9.6
```

GPUs play an outsize role in this space

- Python code: `C = A @ B`
- PyTorch: directly forwards function call to **cuDNN**

[NVIDIA]

```
I  abbe  cuda/lib64 %  ls -lah /usr/lib/x86_64-linux-gnu/libcudnn*
lrwxrwxrwx 1 root root   27 Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_adv_infer.so.8 -> libcudnn_adv_infer.so.8.9.6
-rw-r--r-- 1 root root 120M Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_adv_infer.so.8.9.6
lrwxrwxrwx 1 root root   27 Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_adv_train.so.8 -> libcudnn_adv_train.so.8.9.6
-rw-r--r-- 1 root root 111M Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_adv_train.so.8.9.6
lrwxrwxrwx 1 root root   27 Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_cnn_infer.so.8 -> libcudnn_cnn_infer.so.8.9.6
-rw-r--r-- 1 root root 545M Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_cnn_infer.so.8.9.6
lrwxrwxrwx 1 root root   27 Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_cnn_train.so.8 -> libcudnn_cnn_train.so.8.9.6
-rw-r--r-- 1 root root 120M Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_cnn_train.so.8.9.6
lrwxrwxrwx 1 root root   27 Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_ops_infer.so.8 -> libcudnn_ops_infer.so.8.9.6
-rw-r--r-- 1 root root  87M Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_ops_infer.so.8.9.6
lrwxrwxrwx 1 root root   27 Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_ops_train.so.8 -> libcudnn_ops_train.so.8.9.6
-rw-r--r-- 1 root root  68M Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_ops_train.so.8.9.6
lrwxrwxrwx 1 root root   17 Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn.so.8 -> libcudnn.so.8.9.6
-rw-r--r-- 1 root root 139K Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn.so.8.9.6
```

~1 GiB of shared libraries!

GPUs play an outsize role in this space

- Python code: `C = A @ B`
- PyTorch: directly forwards function call to **cuDNN**

[NVIDIA]

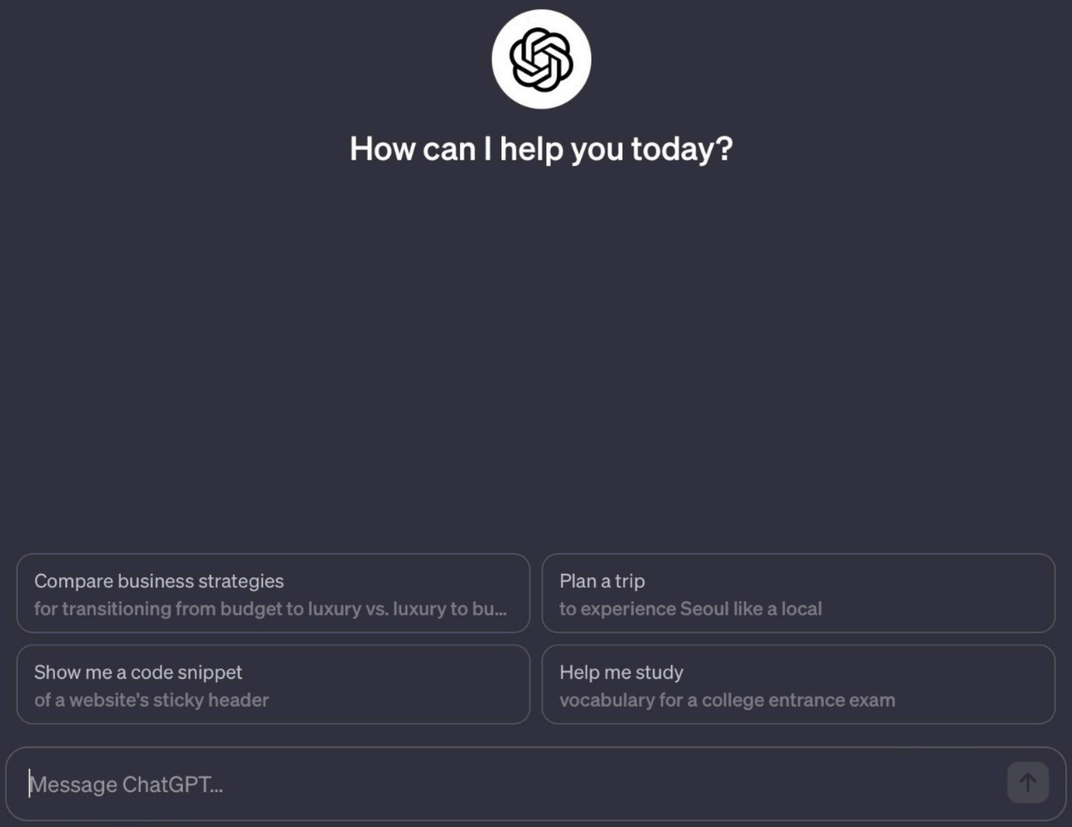
```
I  abbe  cuda/lib64 %  ls -lah /usr/lib/x86_64-linux-gnu/libcudnn*
lrwxrwxrwx 1 root root   27 Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_adv_infer.so.8 -> libcudnn_adv_infer.so.8.9.6
-rw-r--r-- 1 root root 120M Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_adv_infer.so.8.9.6
lrwxrwxrwx 1 root root   27 Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_adv_train.so.8 -> libcudnn_adv_train.so.8.9.6
-rw-r--r-- 1 root root 111M Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_adv_train.so.8.9.6
lrwxrwxrwx 1 root root   27 Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_cnn_infer.so.8 -> libcudnn_cnn_infer.so.8.9.6
-rw-r--r-- 1 root root 545M Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_cnn_infer.so.8.9.6
lrwxrwxrwx 1 root root   27 Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_cnn_train.so.8 -> libcudnn_cnn_train.so.8.9.6
-rw-r--r-- 1 root root 120M Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_cnn_train.so.8.9.6
lrwxrwxrwx 1 root root   27 Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_ops_infer.so.8 -> libcudnn_ops_infer.so.8.9.6
-rw-r--r-- 1 root root  87M Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_ops_infer.so.8.9.6
lrwxrwxrwx 1 root root   27 Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_ops_train.so.8 -> libcudnn_ops_train.so.8.9.6
-rw-r--r-- 1 root root  68M Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn_ops_train.so.8.9.6
lrwxrwxrwx 1 root root   17 Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn.so.8 -> libcudnn.so.8.9.6
-rw-r--r-- 1 root root 139K Oct 31 08:17 /usr/lib/x86_64-linux-gnu/libcudnn.so.8.9.6
```

~1 GiB of shared libraries!

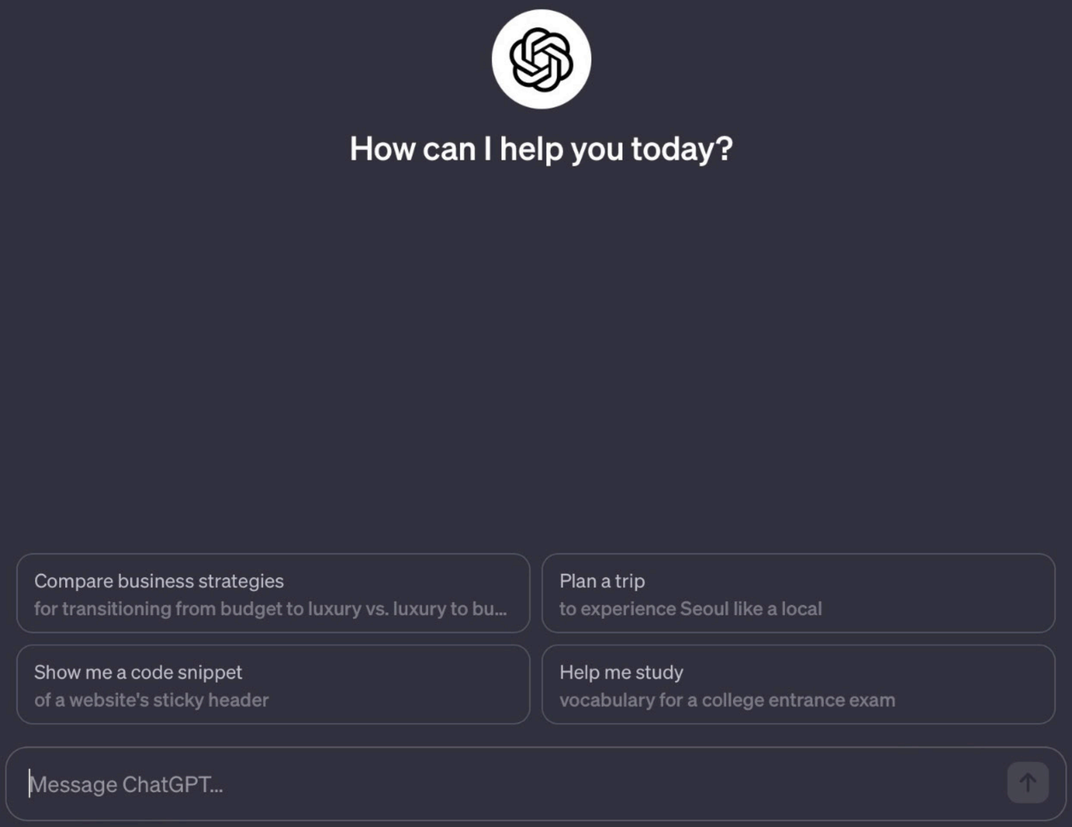
Contains every conceivable kind of matrix multiplication & convolution for every GPU architecture by this vendor.

Assorted applications of Neural Networks

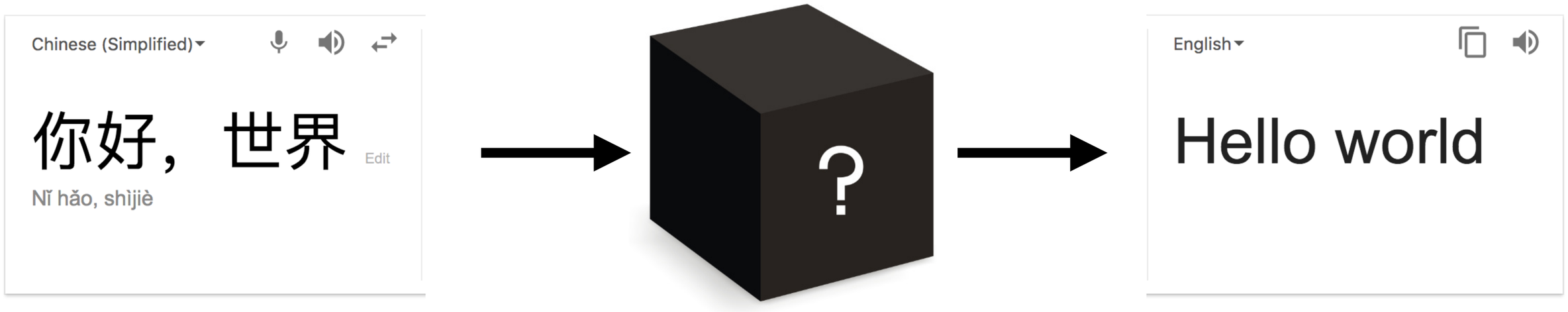
Chat bots: ChatGPT



Chat bots: ChatGPT



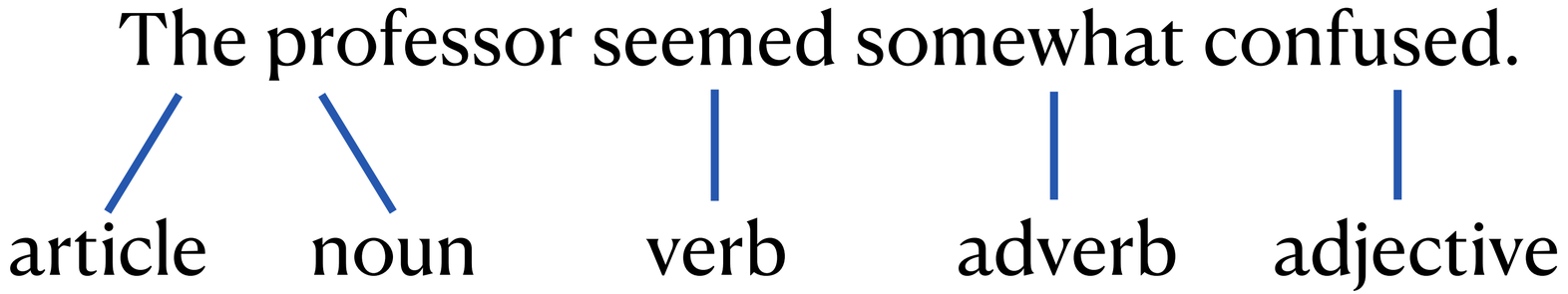
Machine translation



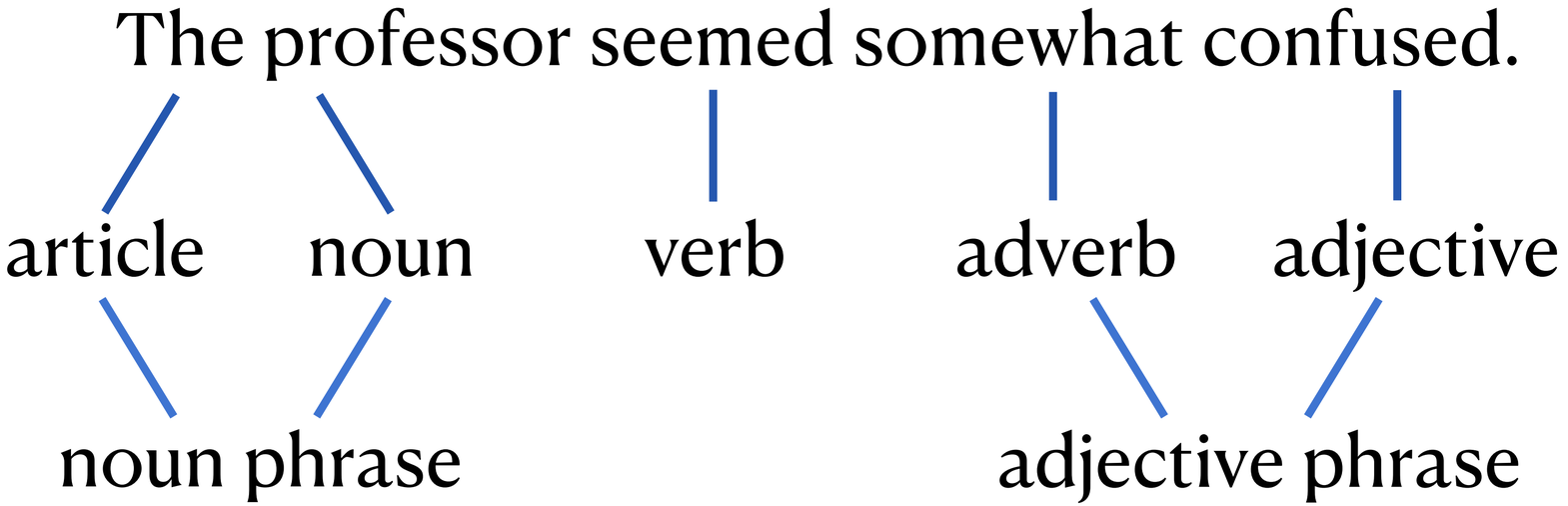
Traditional vs neural approach

The professor seemed somewhat confused.

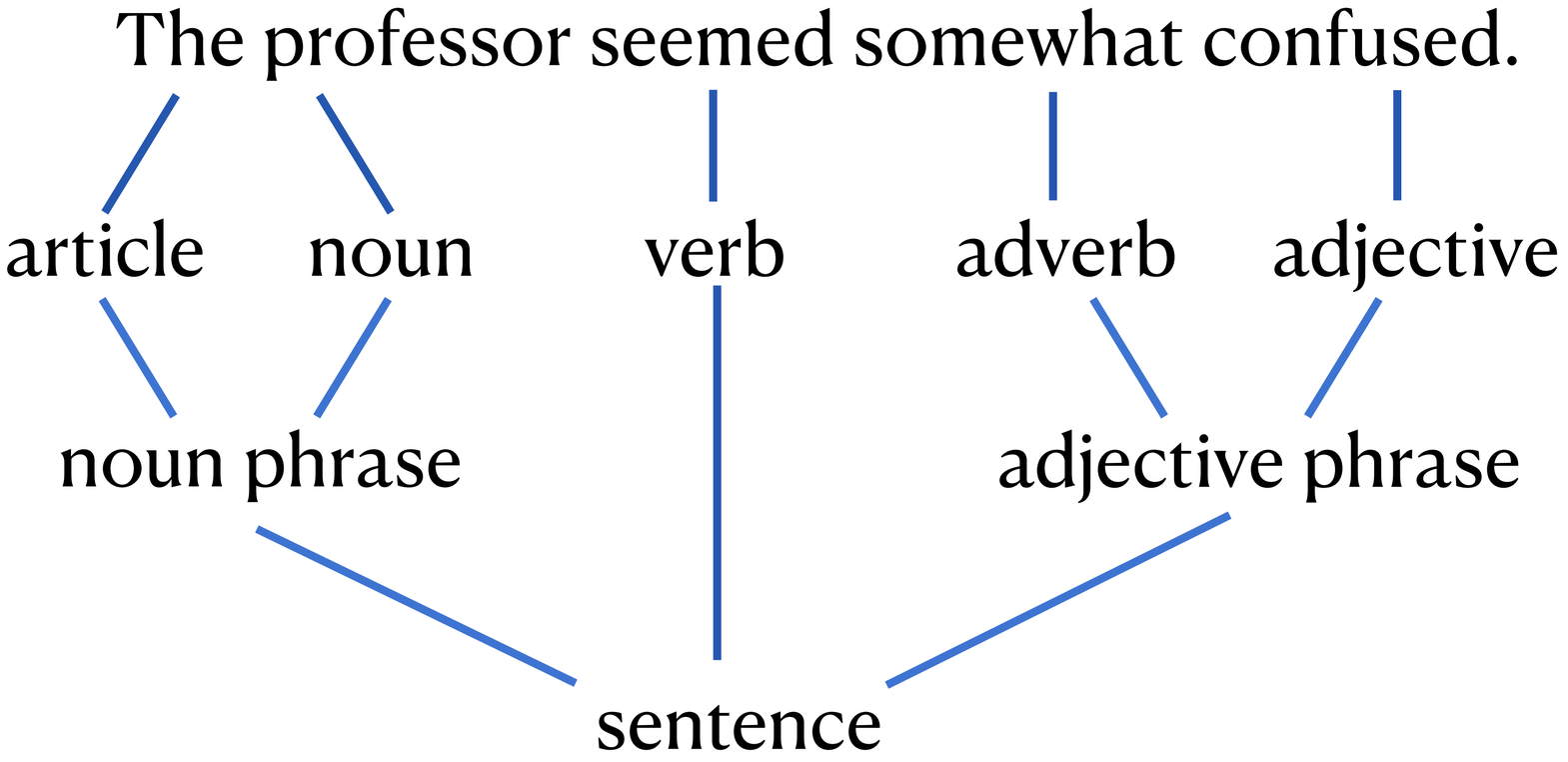
Traditional vs neural approach



Traditional vs neural approach



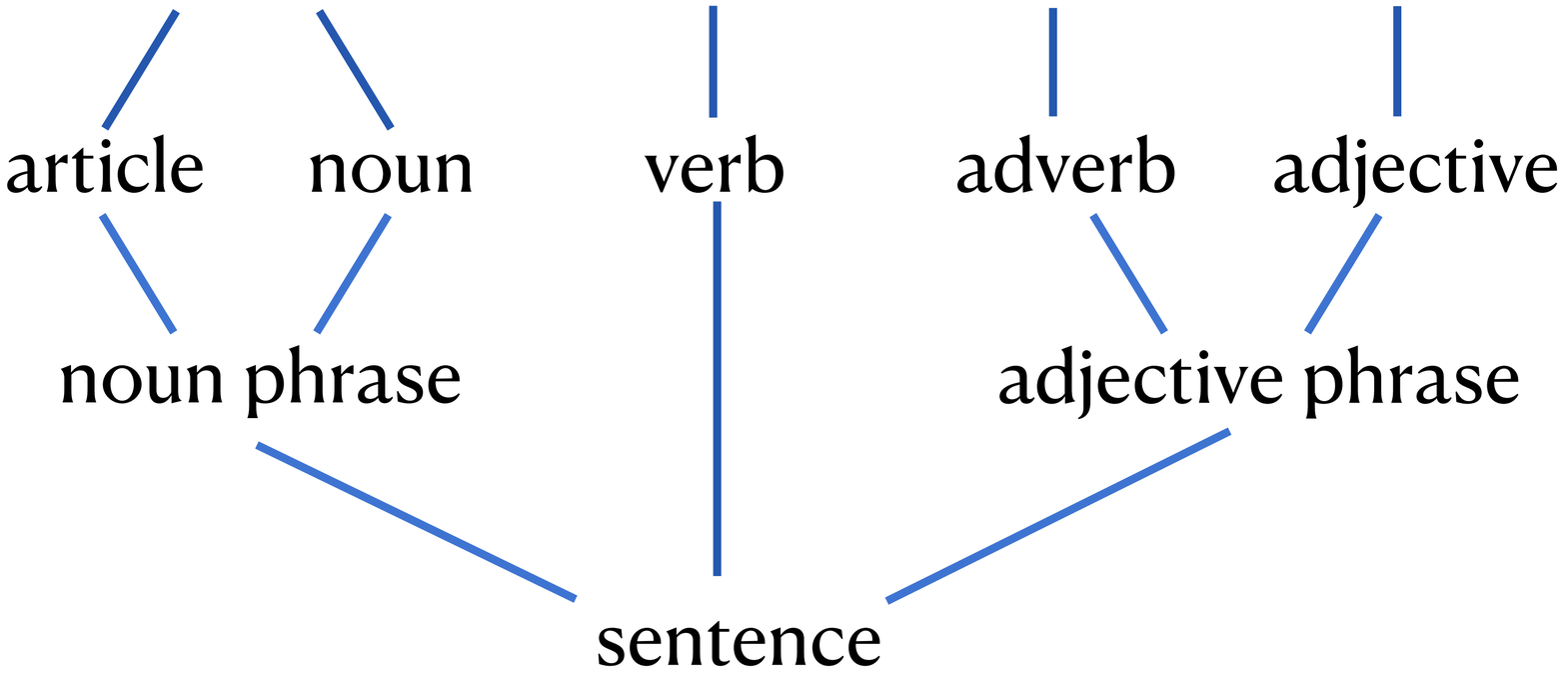
Traditional vs neural approach



Traditional vs neural approach

Der Professor erschien etwas verwirrt. [DE]

The professor seemed somewhat confused. [EN]



Traditional vs neural approach

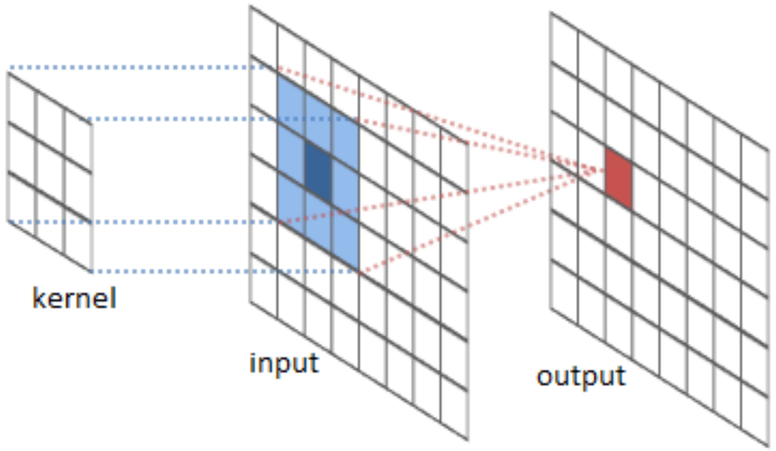
[Meta M2M model, Link]

Convolutional Neural Networks (CNNs)

Filtering of images with local kernels has a long history, for instance for edge detection.

- Local kernel

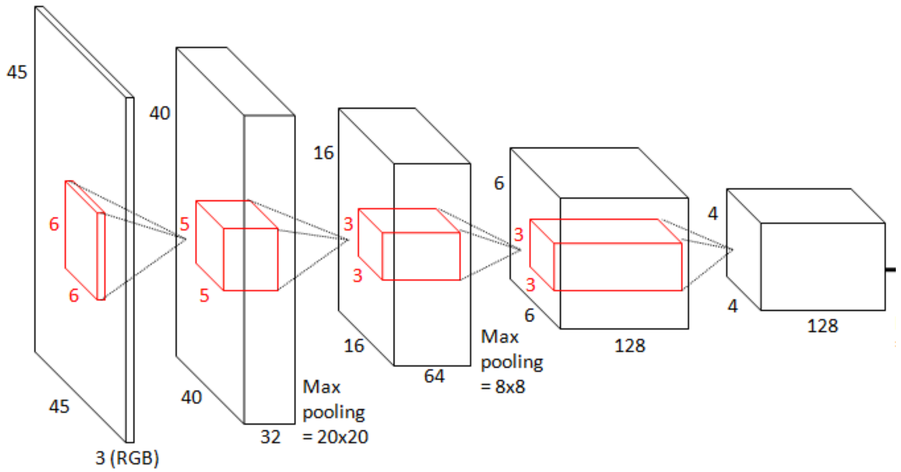
$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



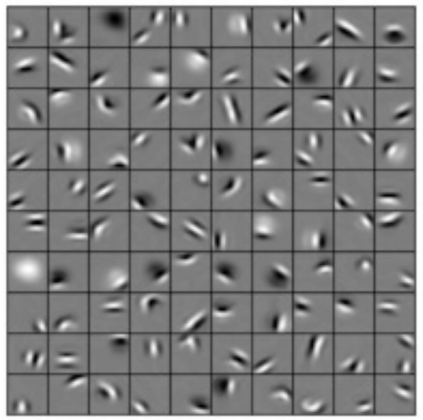
Convolutional Neural Networks (CNNs)

Convolutions with trainable weights — weights *shared* across pixels.

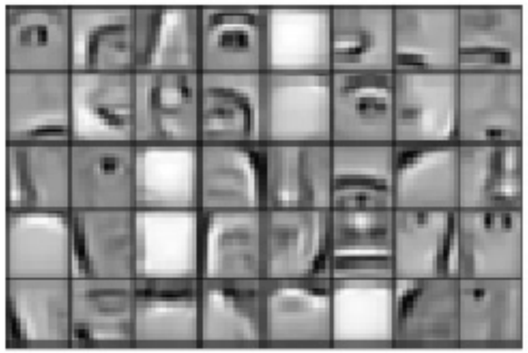
- Weight sharing due to local nature of operations



Low-level



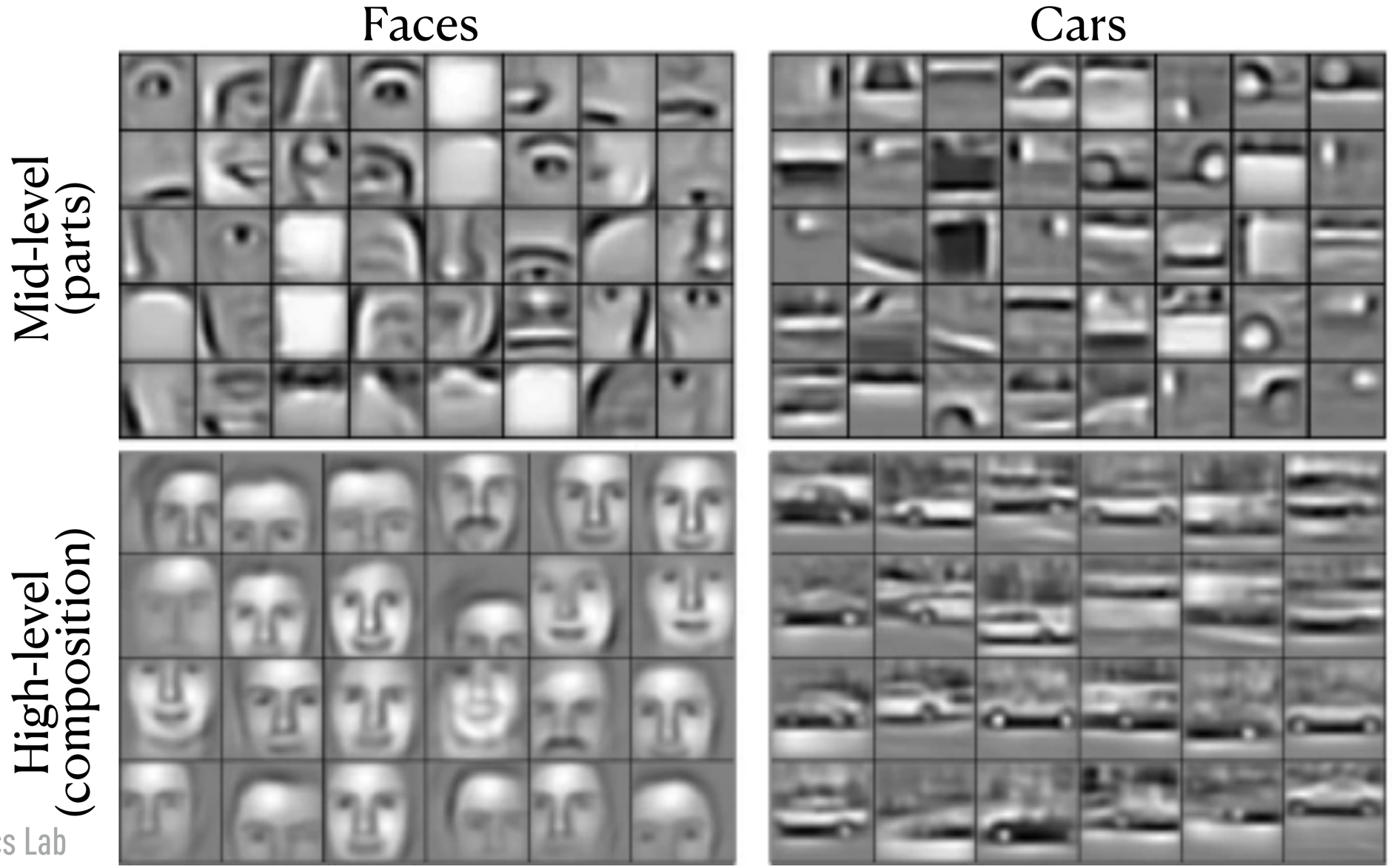
Mid-level



High-level

Convolutional Neural Networks (CNNs)

CNNs influenced by the properties of the training dataset



Neural Networks + Automotive applications

Neural Networks + Automotive applications

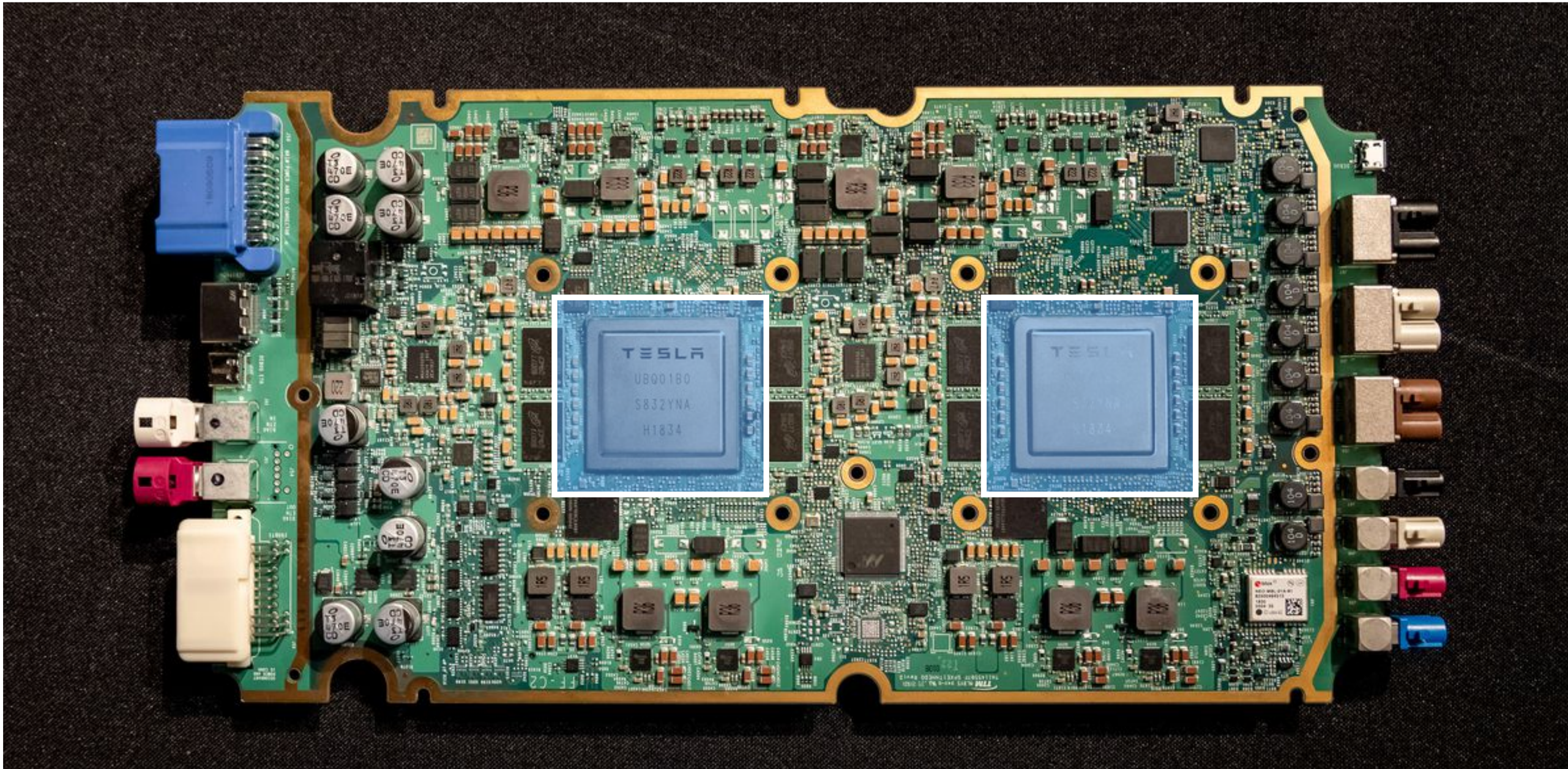


Image and Object classification

YoLo model [Redmon 2016]

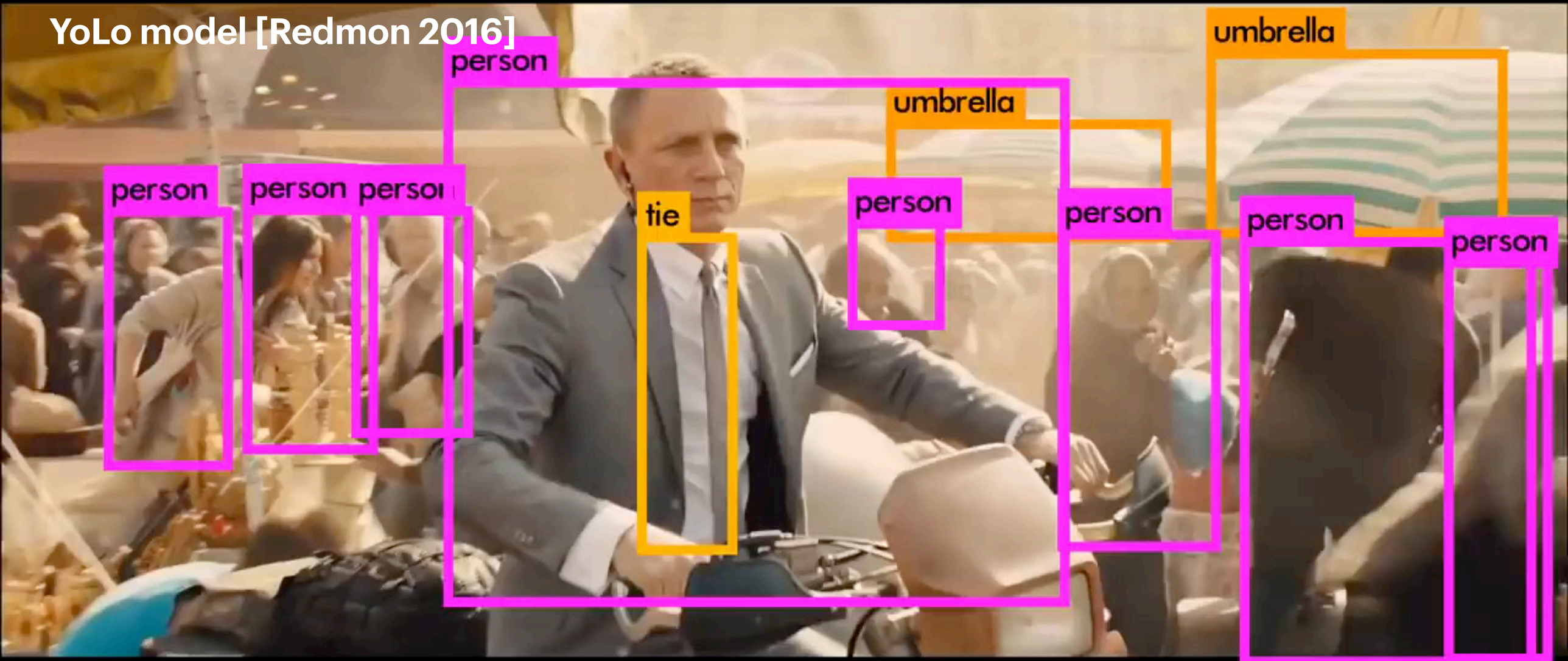
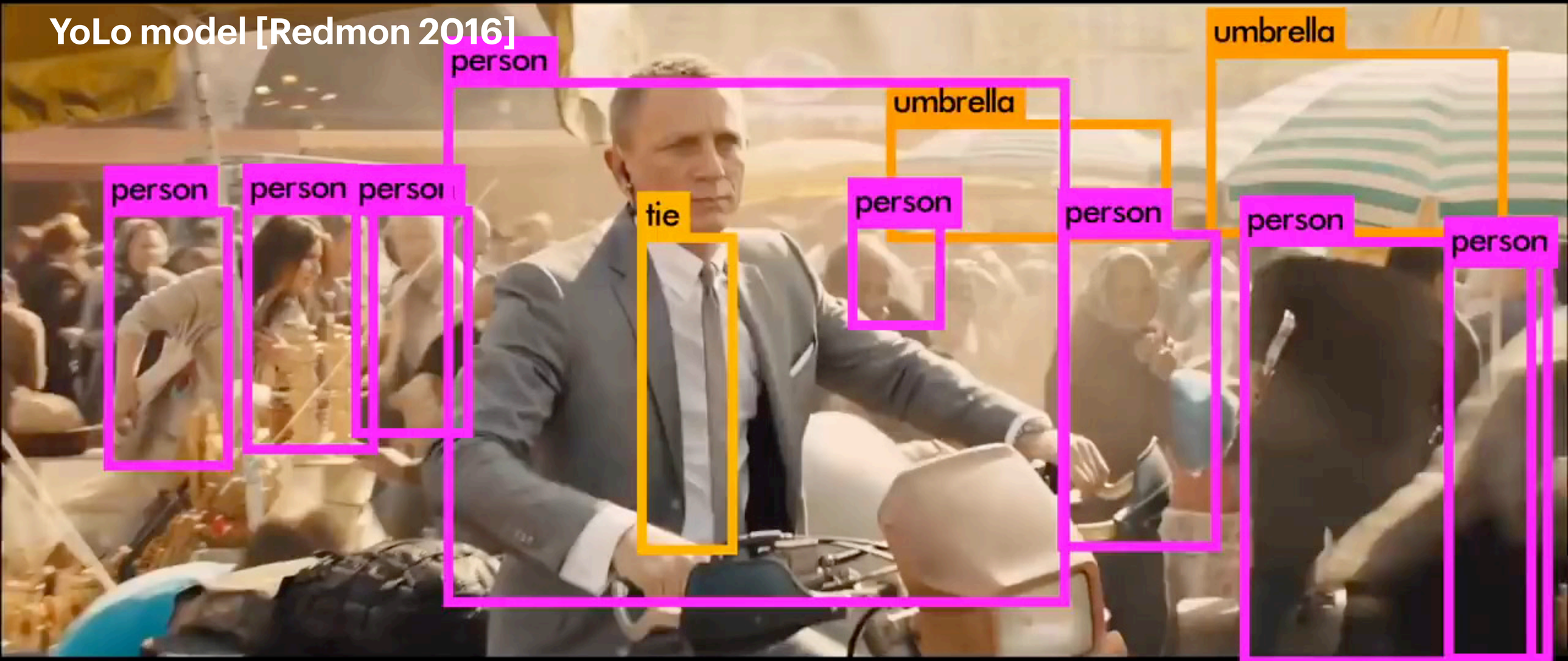


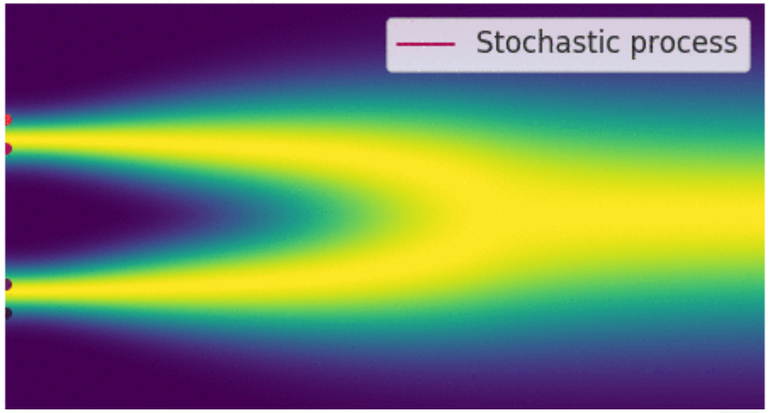
Image and Object classification

YoLo model [Redmon 2016]



Diffusion models

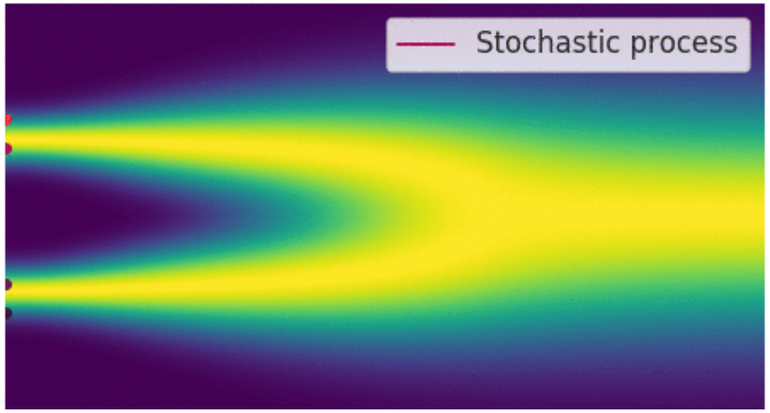
[https://yang-song.github.io/blog/2021/score/]



Progressively turning images into noise

Diffusion models

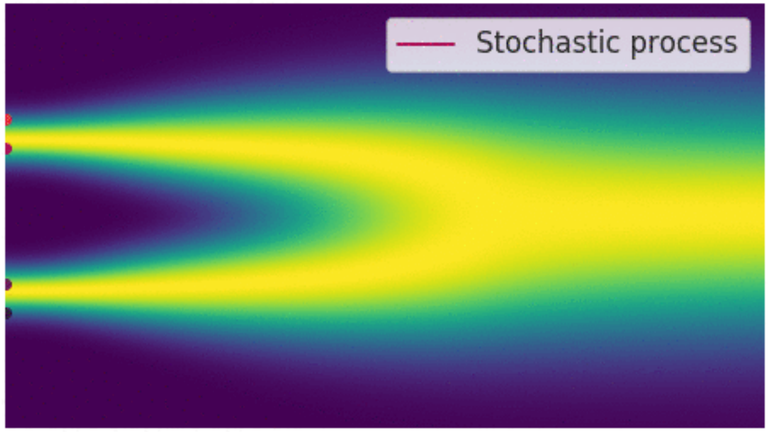
[https://yang-song.github.io/blog/2021/score/]



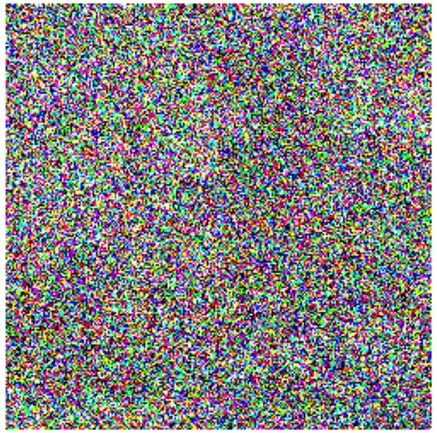
Progressively turning images into noise

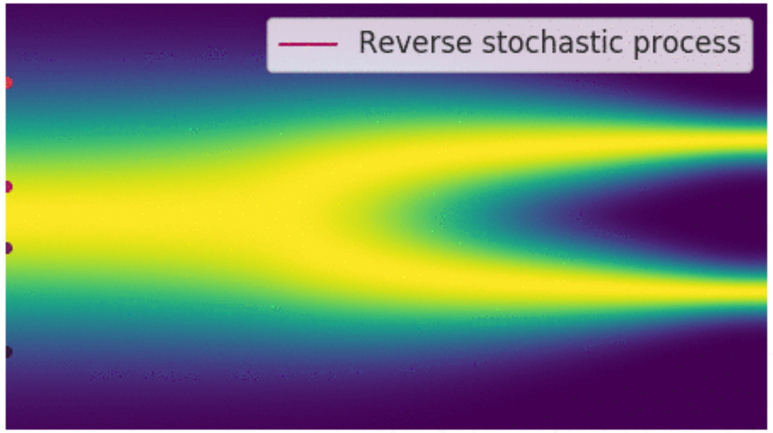
Diffusion models

[https://yang-song.github.io/blog/2021/score/]



Progressively turning images into noise





A learned inverse of the above process.

Generative image systems: MidJourney

Other similar products: Stable Diffusion, DALL·E, Adobe Firefly

1:14 PM **a child hypnotized by a pefectly circular spiral on a vintage TV**
Job ID: de1133de-2d27-421b-8ed8-f7c9f2f7b8f0
seed 3979930206

Jump to message

Message @Midjourney Bot

Generative image systems: MidJourney

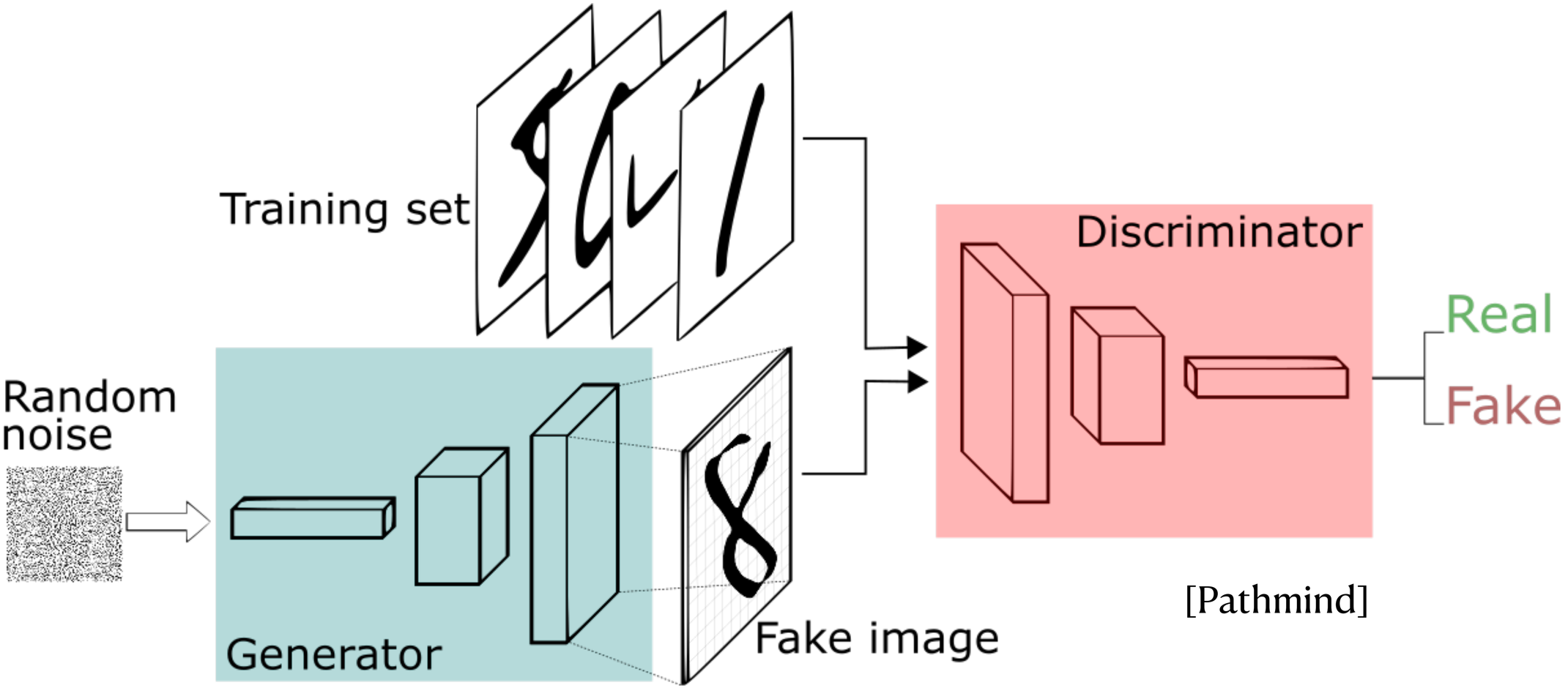
Other similar products: Stable Diffusion, DALL·E, Adobe Firefly

1:14 PM **a child hypnotized by a pefectly circular spiral on a vintage TV**
Job ID: de1133de-2d27-421b-8ed8-f7c9f2f7b8f0
seed 3979930206

Jump to message

Message @Midjourney Bot

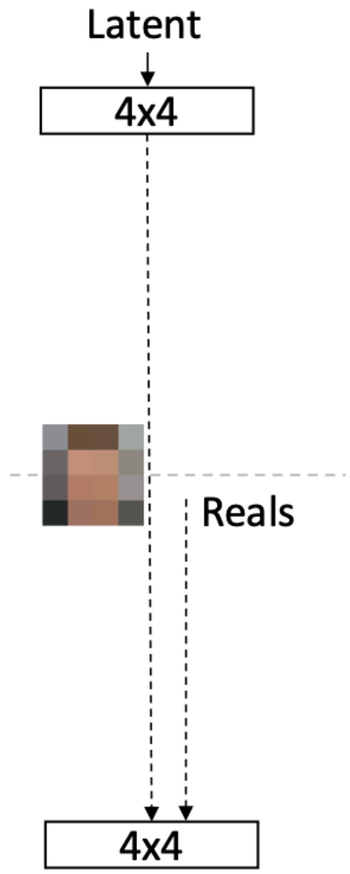
Generative Adversarial Networks



[Pathmind]

Progressively grown GANs

[Karras et al. ICLR 2018]

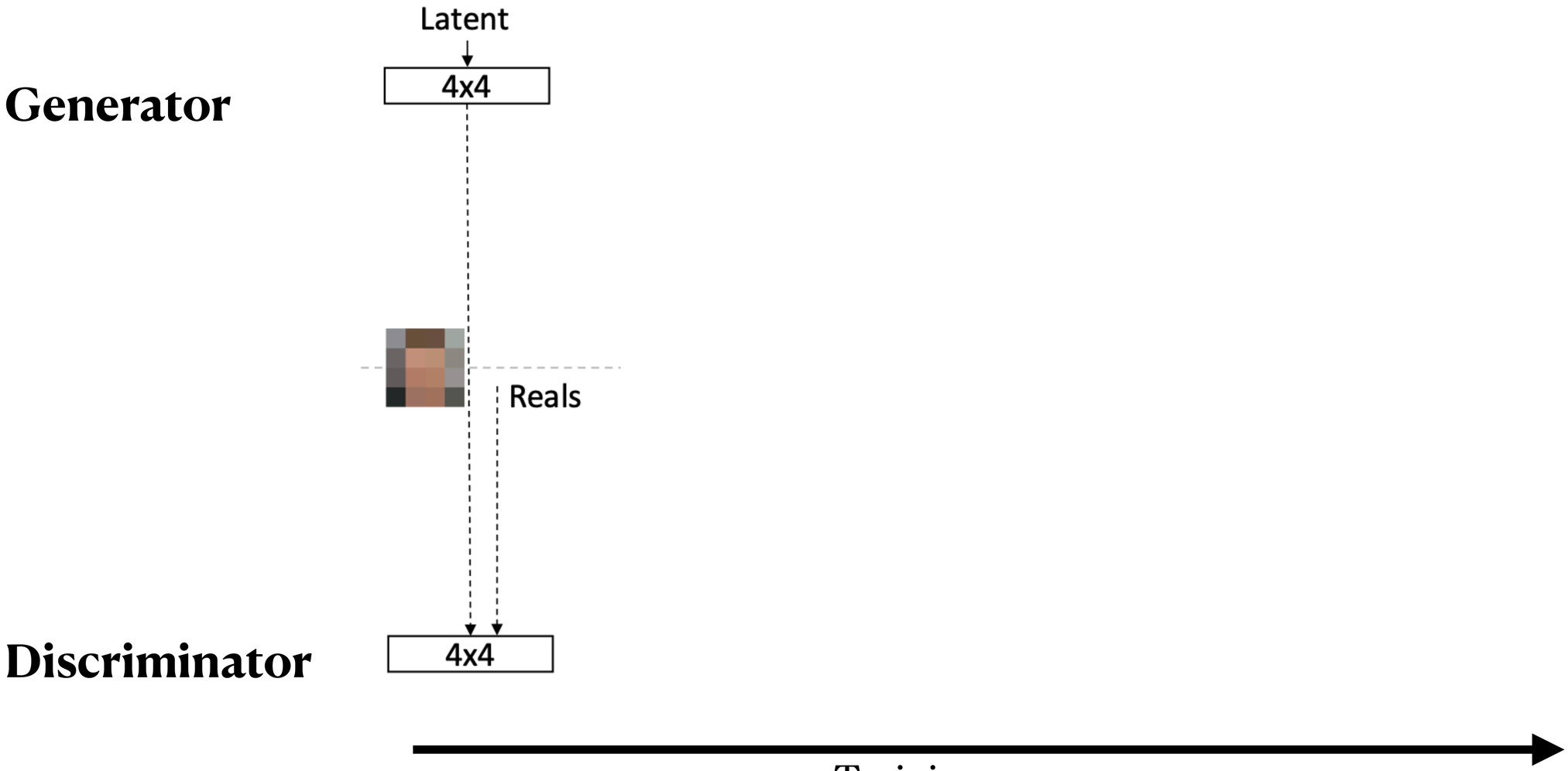


Generator

Discriminator

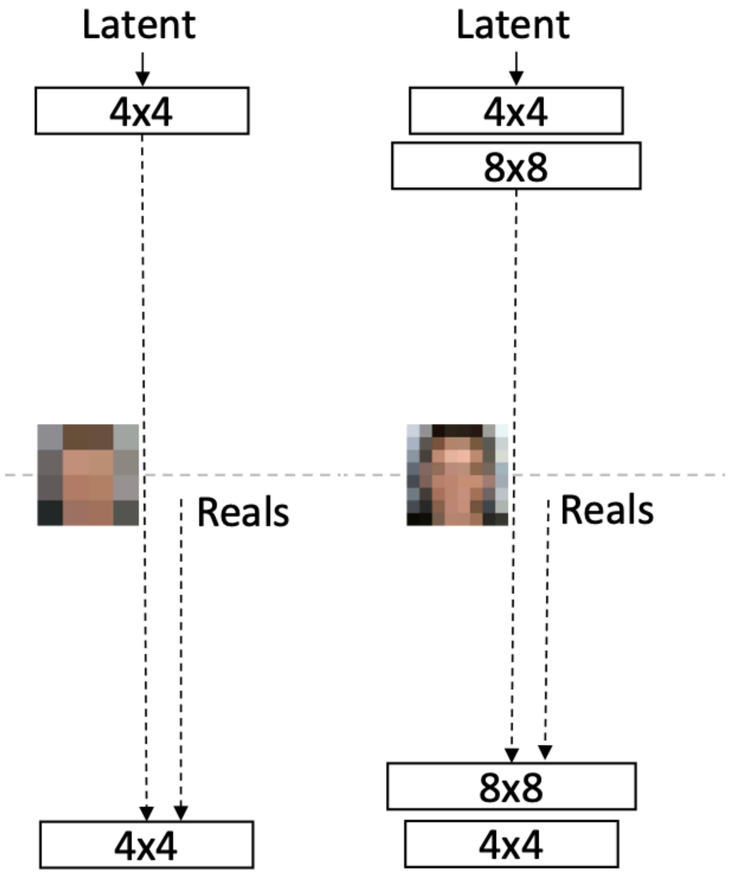
Progressively grown GANs

[Karras et al. ICLR 2018]



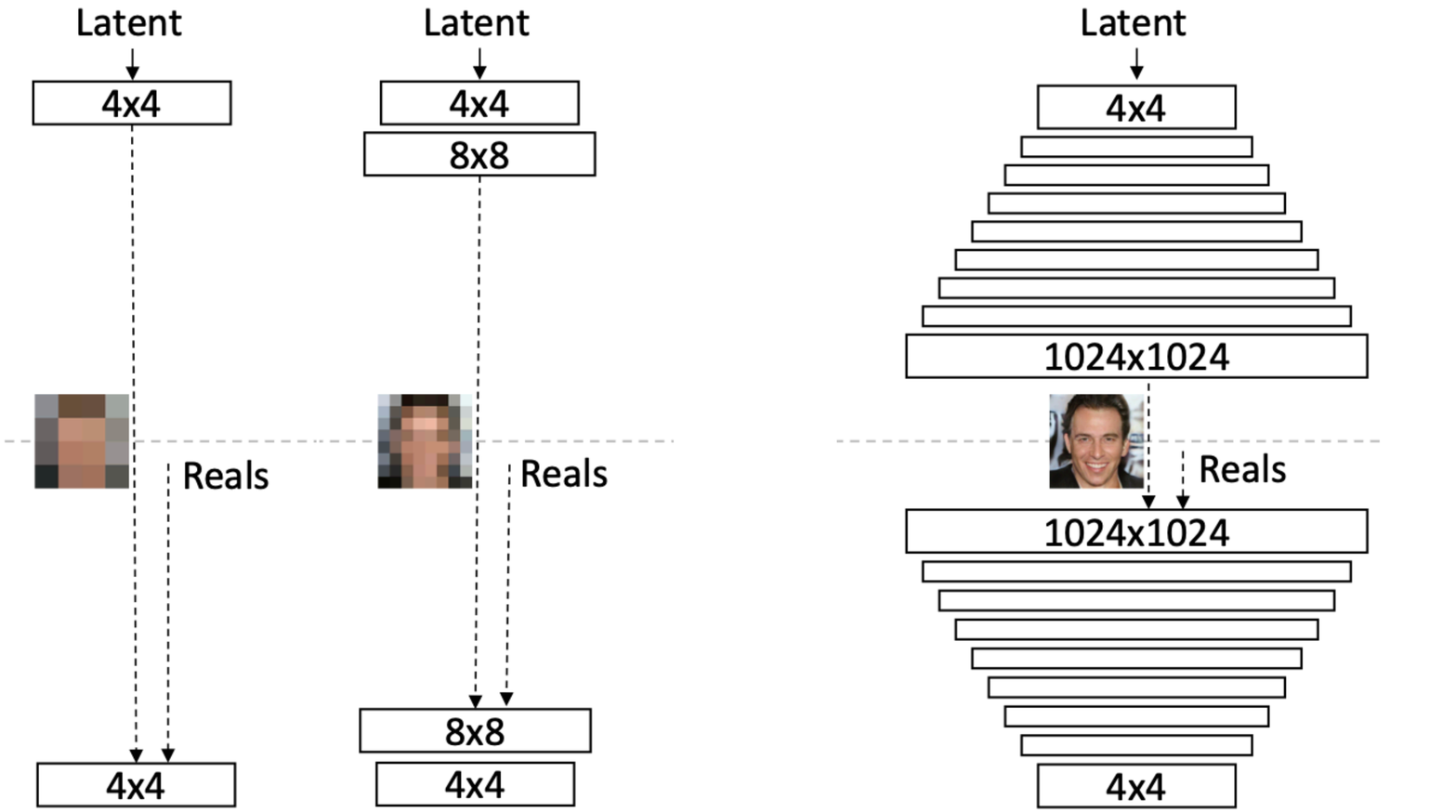
Progressively grown GANs

[Karras et al. ICLR 2018]



Progressively grown GANs

[Karras et al. ICLR 2018]



Face GANs

Warning:
the following sequences blend faces of various genders, ethnicities & ages. This is of course rather objectionable.

These things exist now, and they are here to stay. Let's see a somewhat disturbing video about a project from **2019**.

In the ML world this is *ancient history*.

A Style-Based Generator Architecture for Generative Adversarial Networks [Karras et al. 2019]

Our generator thinks of an image as a collection of “styles”, where each style controls the effects at a particular scale

- Coarse styles → pose, hair, face shape
- Middle styles → facial features, eyes
- Fine styles → color scheme

A Style-Based Generator Architecture for Generative Adversarial Networks [Karras et al. 2019]

Our generator thinks of an image as a collection of “styles”, where each style controls the effects at a particular scale

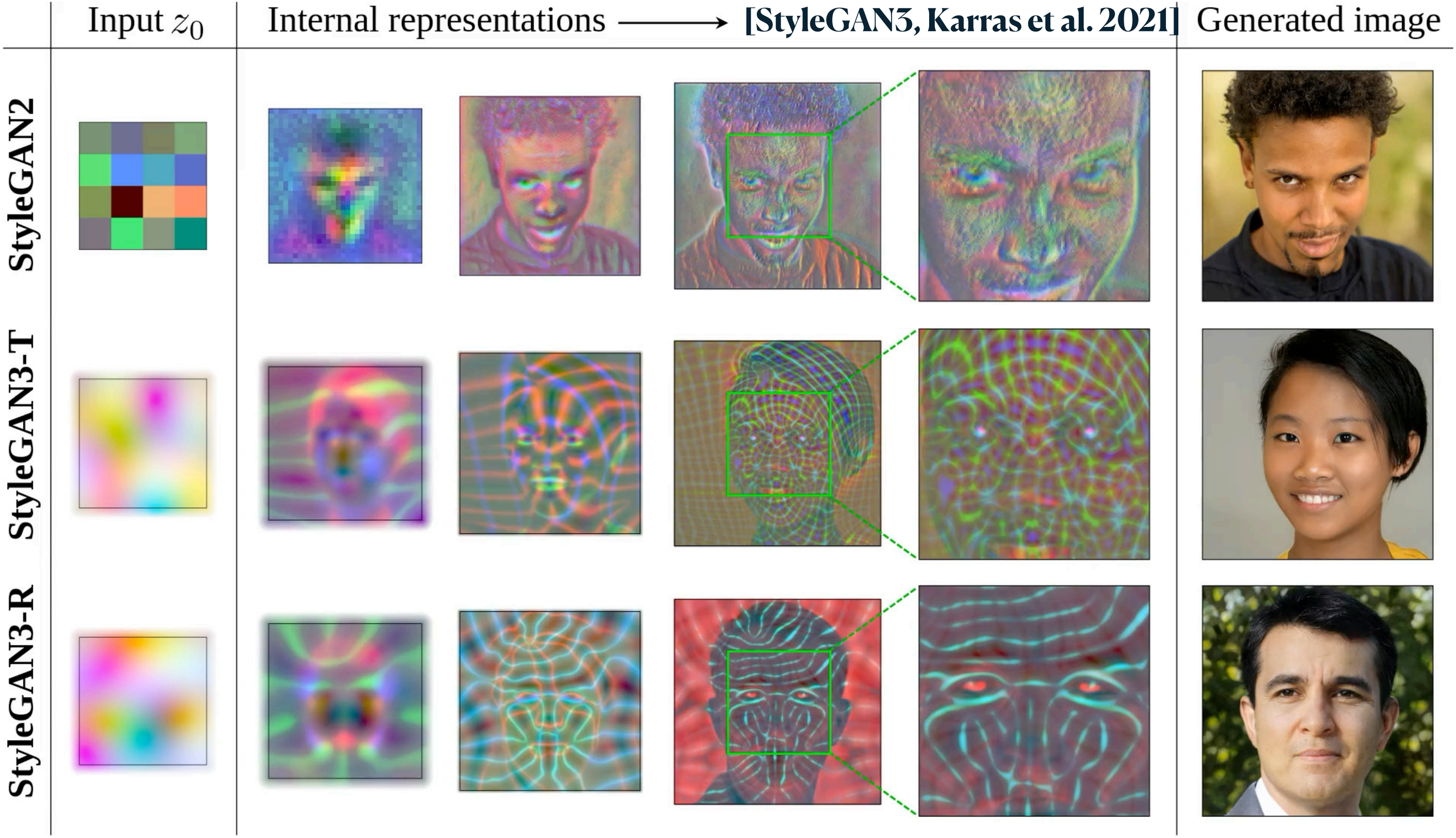
- Coarse styles → pose, hair, face shape
- Middle styles → facial features, eyes
- Fine styles → color scheme

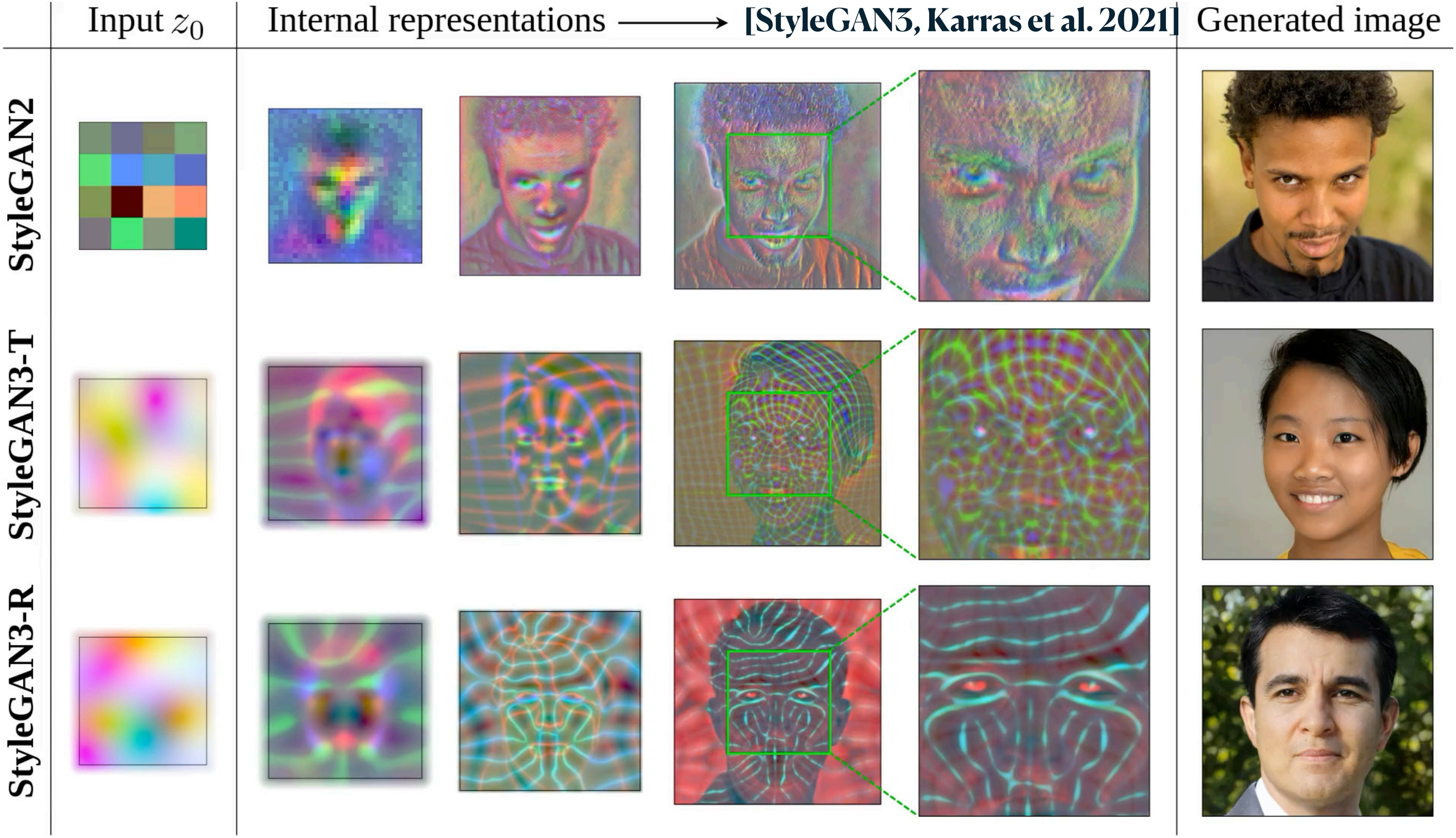
A Style-Based Generator Architecture for Generative Adversarial Networks [Karras et al. 2019]

We can choose the strength at which each style is applied, with respect to an “average face”

A Style-Based Generator Architecture for Generative Adversarial Networks [Karras et al. 2019]

We can choose the strength at which each style is applied, with respect to an “average face”


Input z_0
Internal representations
[StyleGAN3, Karras et al. 2021]
Generated image
StyleGAN2
StyleGAN3-T
StyleGAN3-R


Input z_0
Internal representations
[StyleGAN3, Karras et al. 2021]
Generated image
StyleGAN2
StyleGAN3-T
StyleGAN3-R

GAN inversion

- Given a portrait image of an existing person, run gradient descent to find a l*atent representation* (low-dim. vector)
- .. which is chosen so that re-running the GAN reproduces the original image as closely as possible. This works surprisingly well as you can see.
- Can explore neighborhood in latent space to age the person, change their gender, hairstyle, etc. (Won't show pictures).

Input Inversion

https://yuval-alaluf.github.io/hyperstyle/

Showcase 3: Synthesia

Showcase 3: Synthesia

Showcase 3: Synthesia

Realtime supersampling: NVIDIA DLSS

DLSS Off
Overdrive Mode

DLSS 3.5
Overdrive Mode With Ray Reconstruction

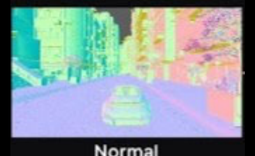
Realtime supersampling: NVIDIA DLSS

DLSS Off
Overdrive Mode

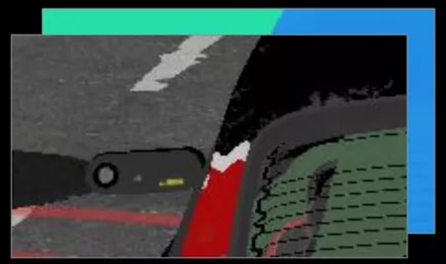
DLSS 3.5
Overdrive Mode With Ray Reconstruction

Denoising & Upscaling for Computer Games

Additional inputs (these are cheap to compute)

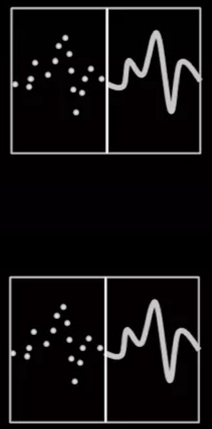


[CD Projekt, NVIDIA]



Engine generates materials & geometry (no lighting effects)

Sampled rays for multiple lighting effects



Hand-tuned denoisers fill in missing pixels

RT image is composited
(low resolution)

RT image is upscaled
(Native resolution)

Exponential growth

[Krenn et al. 2023, Nature Machine Intelligence]

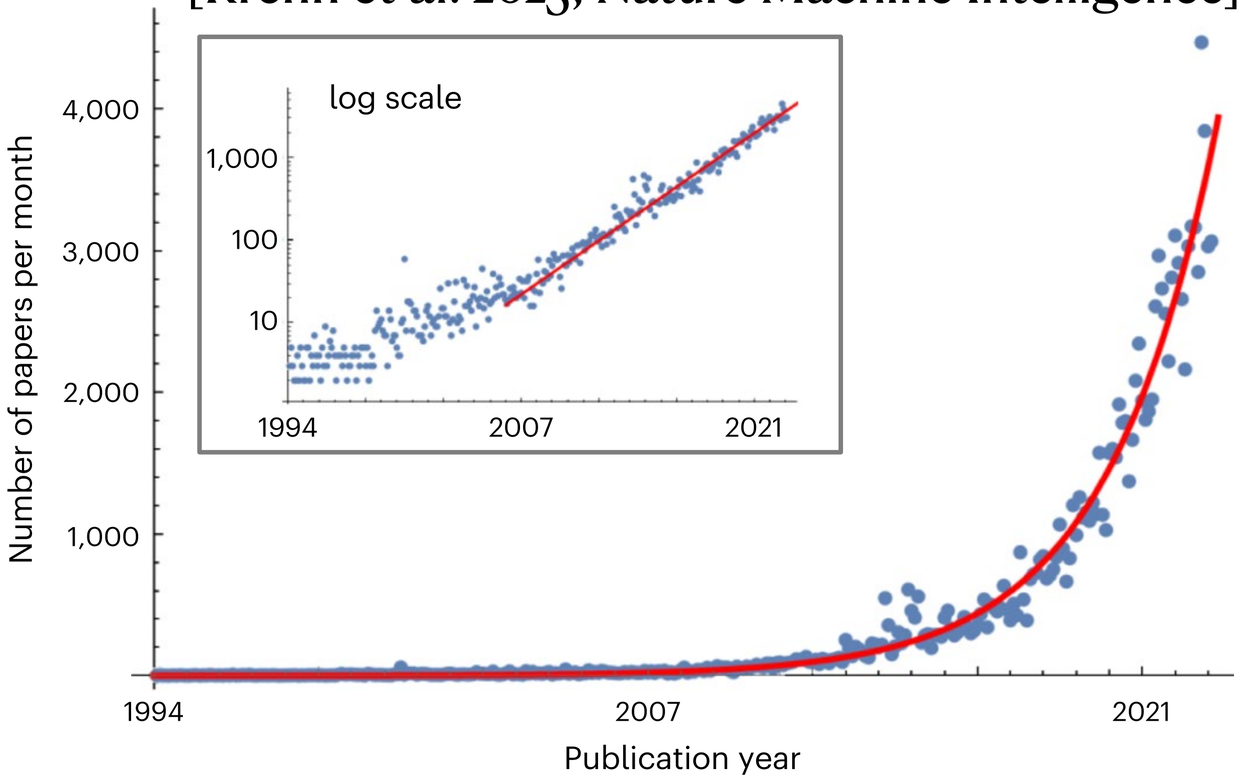


Fig. 1 | The number of papers published per month in the arXiv categories of AI and ML are growing exponentially. The doubling rate of papers per month is roughly 23 months, which might lead to problems for publishing in these fields, at some point. The categories are cs.AI, cs.LG, cs.NE and stat.ML.

MidJourney: *A child hypnotized by a spiral pattern on a vintage TV.*

Exponential growth

[Krenn et al. 2023, Nature Machine Intelligence]

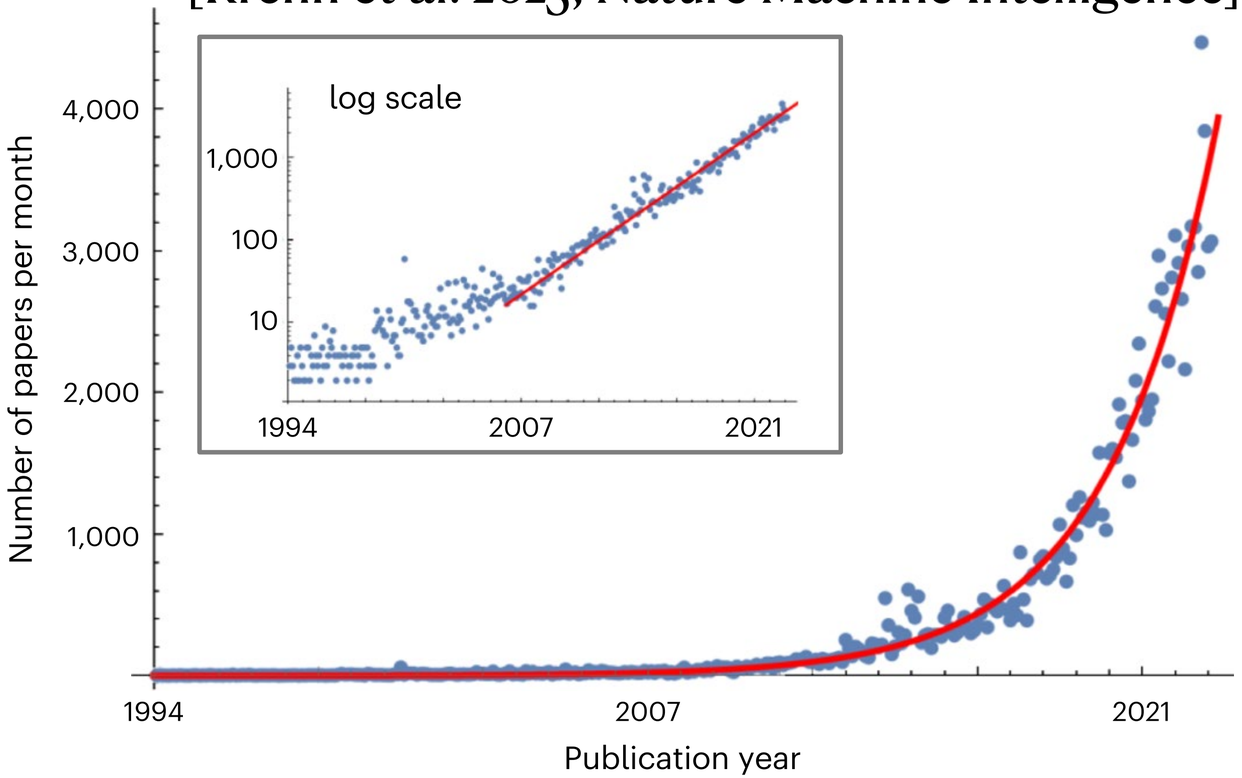


Fig. 1 | The number of papers published per month in the arXiv categories of AI and ML are growing exponentially. The doubling rate of papers per month is roughly 23 months, which might lead to problems for publishing in these fields, at some point. The categories are cs.AI, cs.LG, cs.NE and stat.ML.

MidJourney: *A child hypnotized by a spiral pattern on a vintage TV.*

Exponential growth

[Krenn et al. 2023, Nature Machine Intelligence]

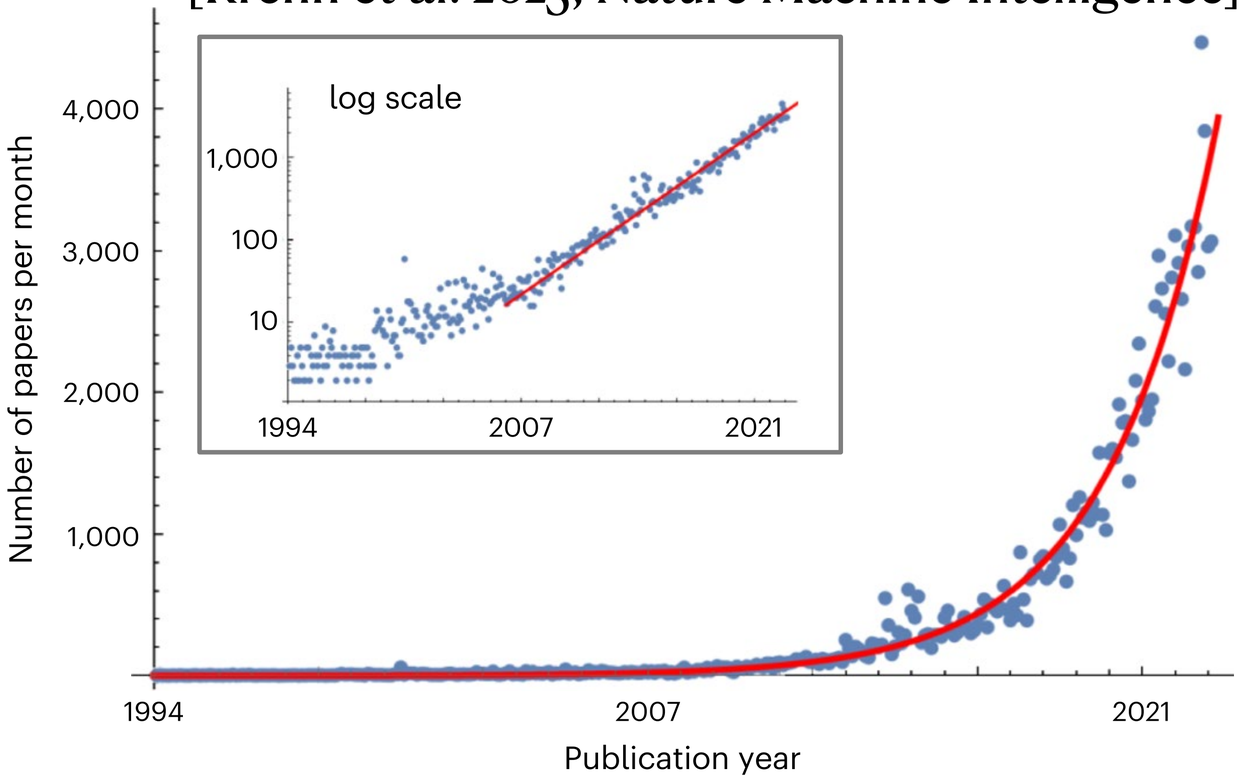


Fig. 1 | The number of papers published per month in the arXiv categories of AI and ML are growing exponentially. The doubling rate of papers per month is roughly 23 months, which might lead to problems for publishing in these fields, at some point. The categories are cs.AI, cs.LG, cs.NE and stat.ML.

MidJourney: *A child hypnotized by a spiral pattern on a vintage TV.*