

Lecture 9

Neural Networks

**CS328 - Numerical Methods for
Visual Computing and Machine Learning**

Prof. Wenzel Jakob

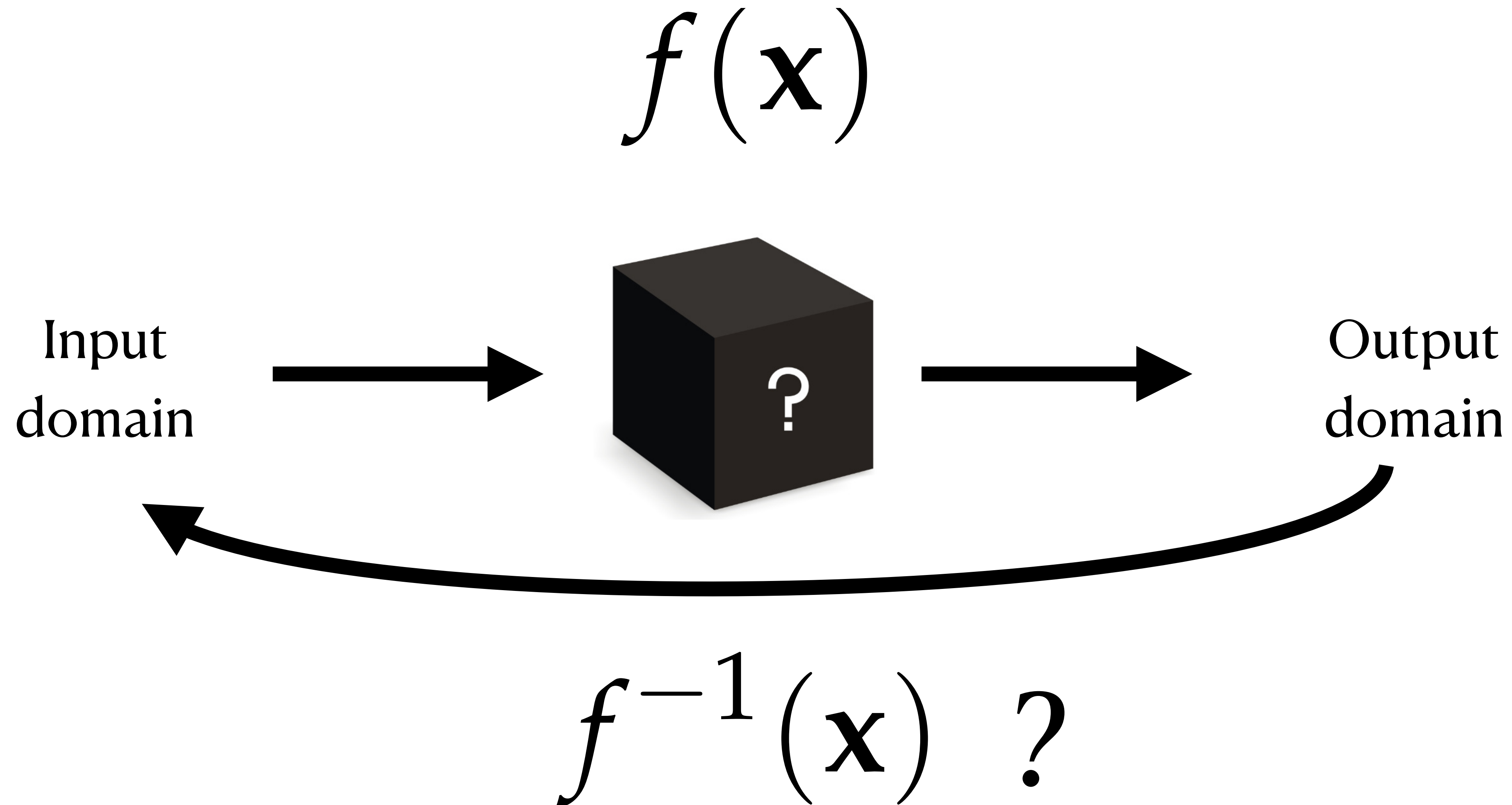
An impossible task

Cannot cover neural networks in just one lecture

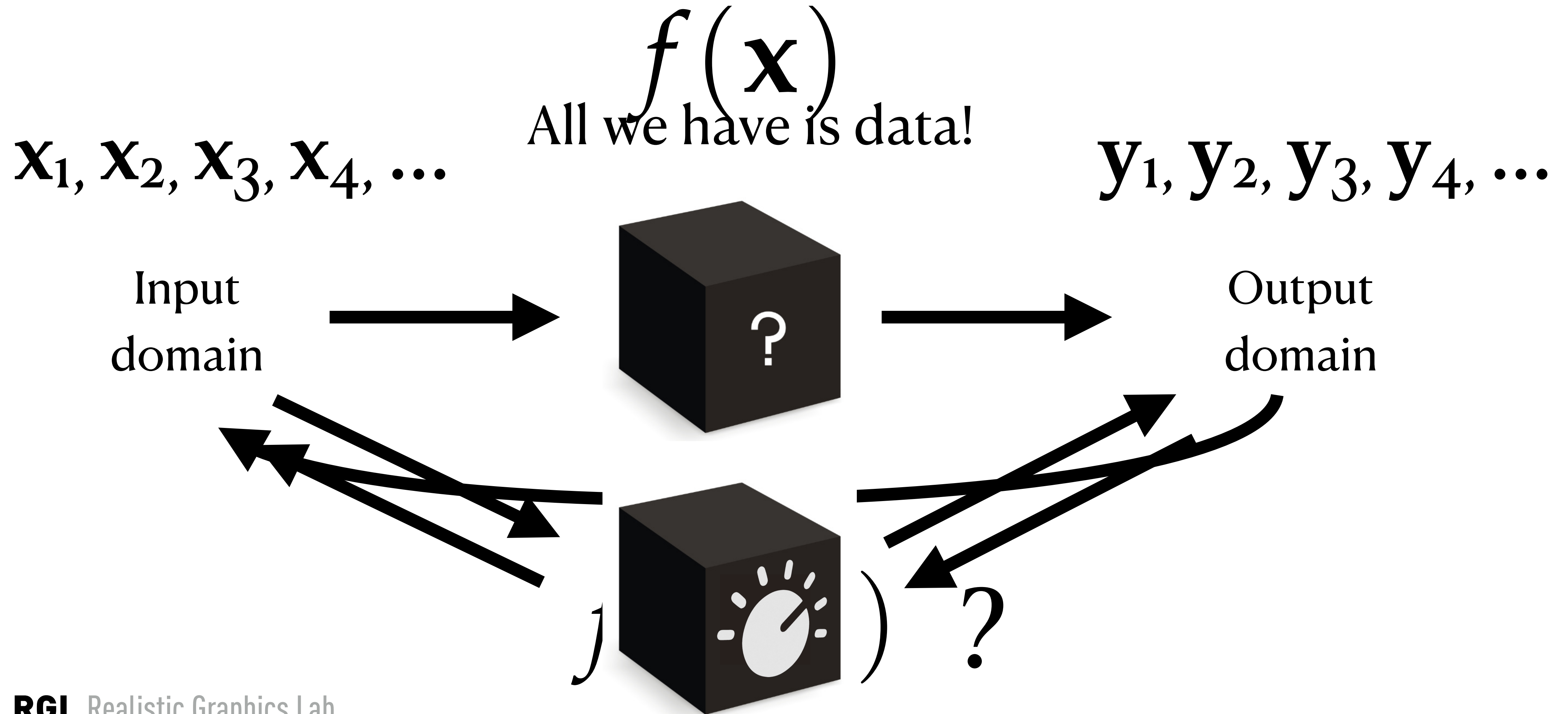
- **Pitch:** What do neural networks enable, and why you should learn about them.
(*Warning:* heavily commercial product demos ahead)
- **Anti-pitch:** Why *specializing completely* on machine learning might *not* be a good career choice.
- **Agenda:**
 - Biological motivations
 - Multilayer perceptrons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

CS-233, CS-433, PHYS-467

Inverse problems



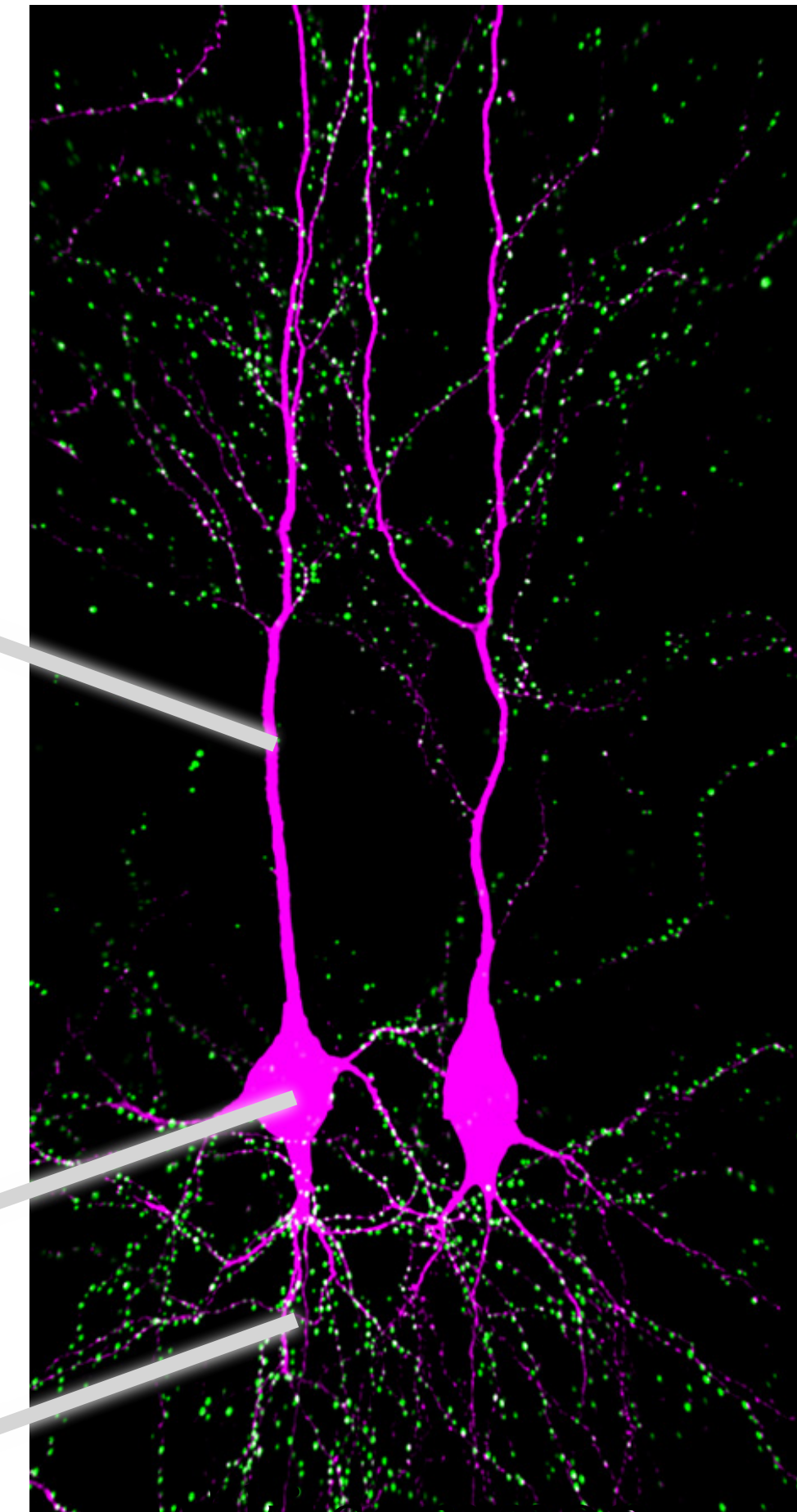
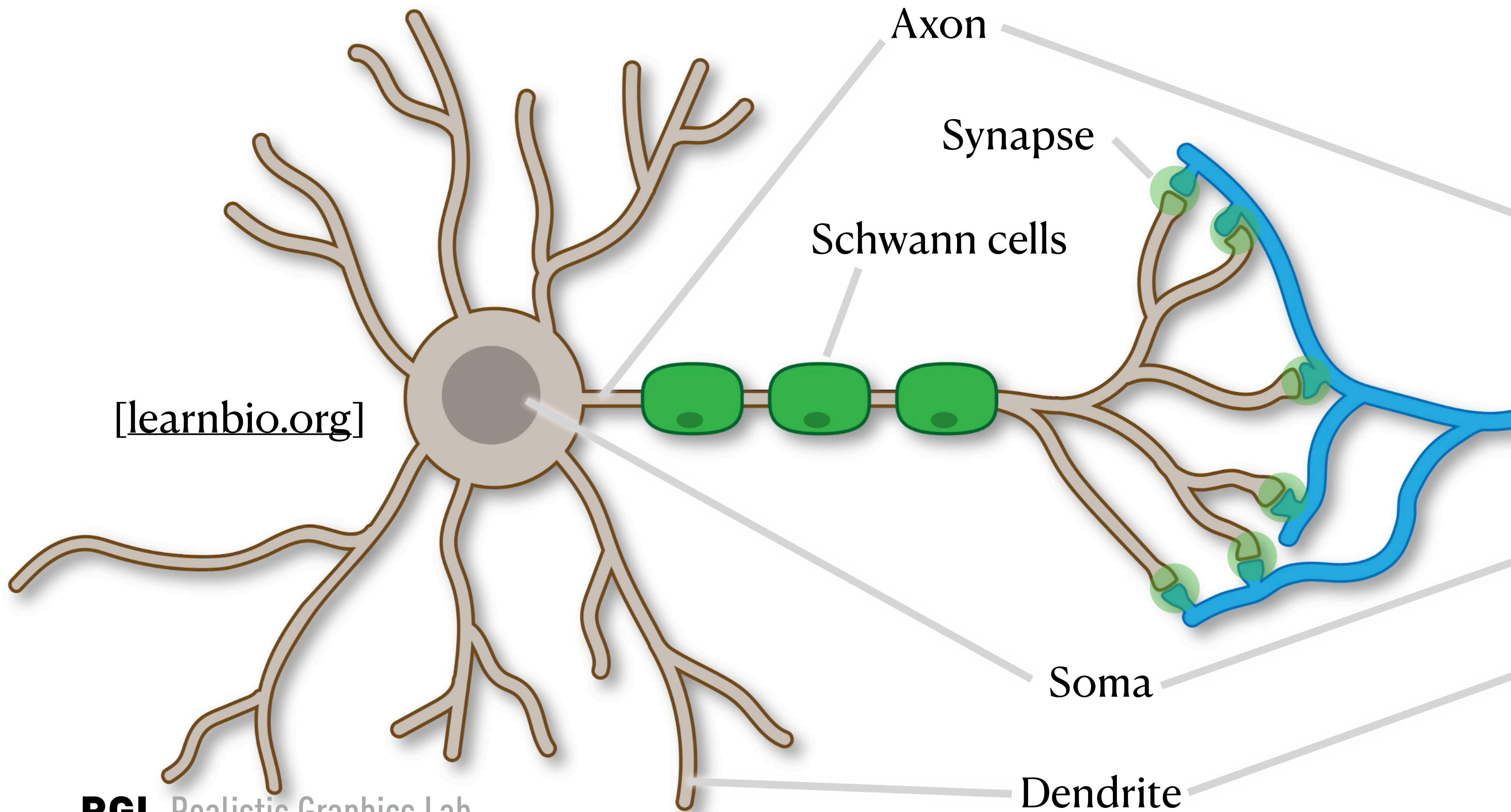
Inverse problems based on data



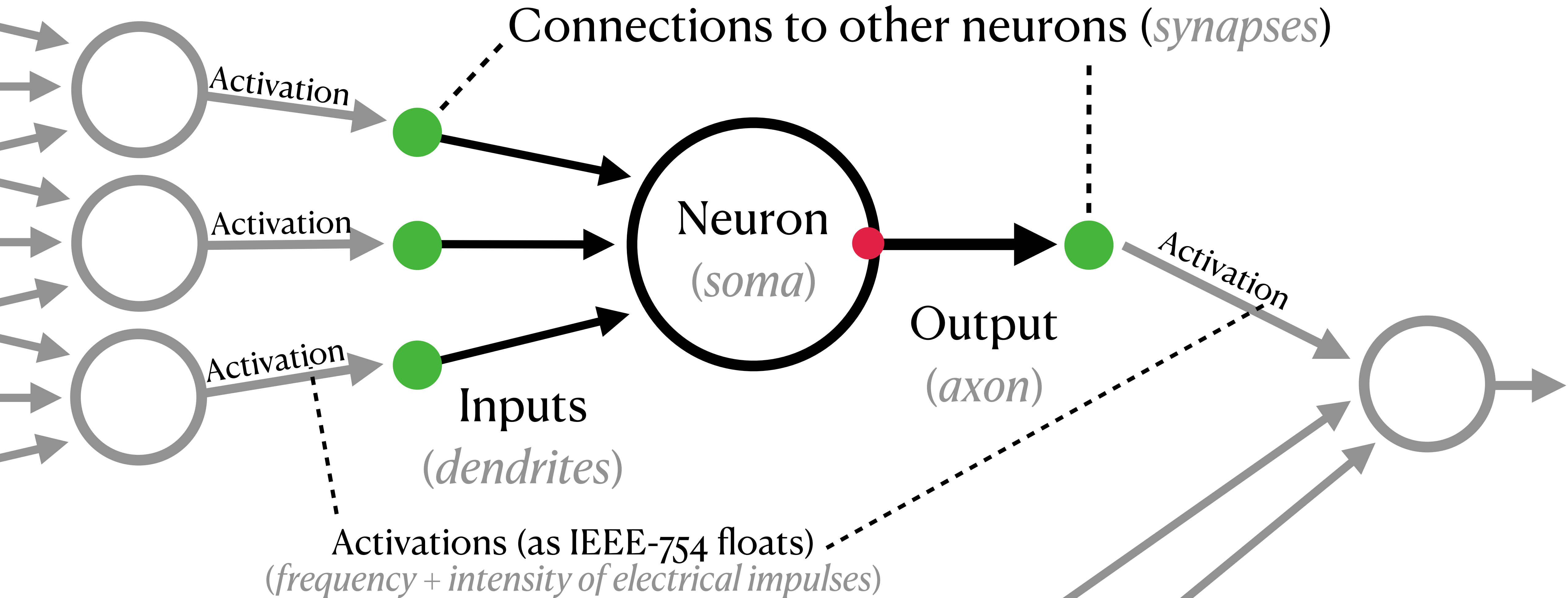
A biological neuron

Very high level view, biological neurons are very complicated.

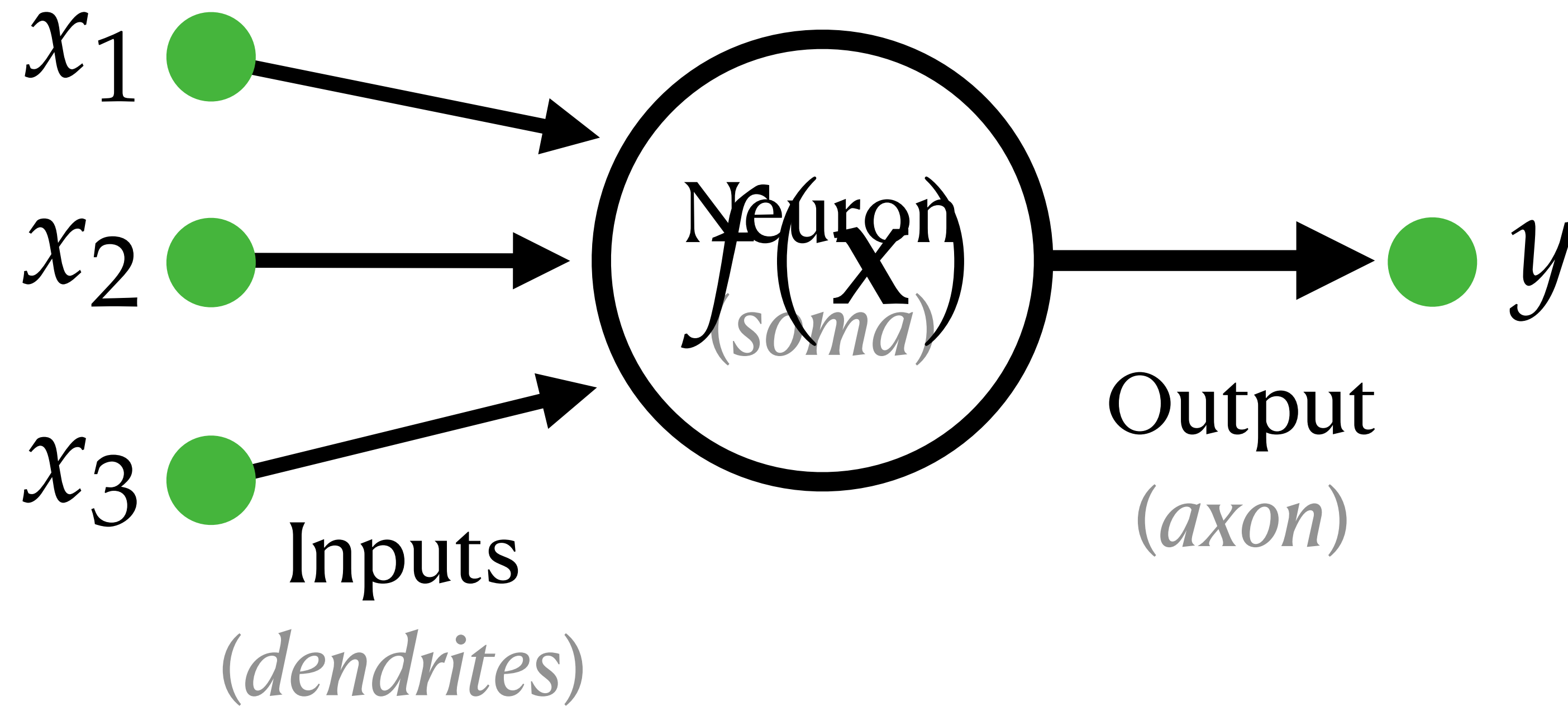
In some parts of the brain, neurons have a **tree-like topology**: many inputs, 1 output



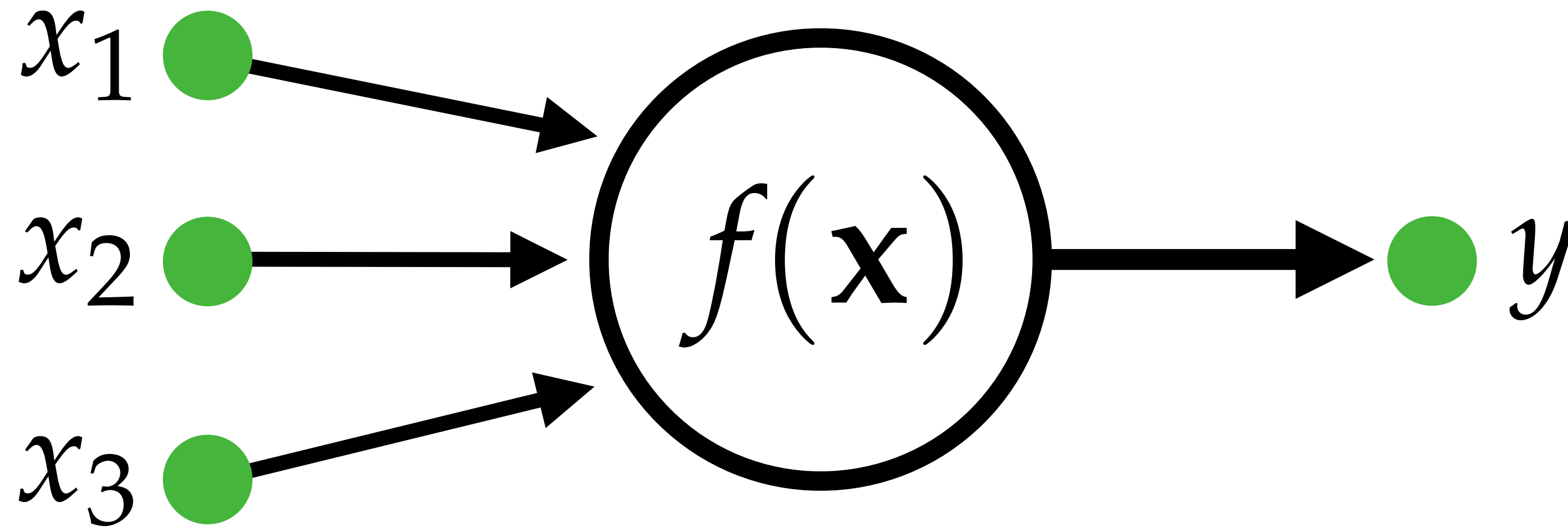
An artificial neuron



An artificial neuron



An artificial neuron — linear case

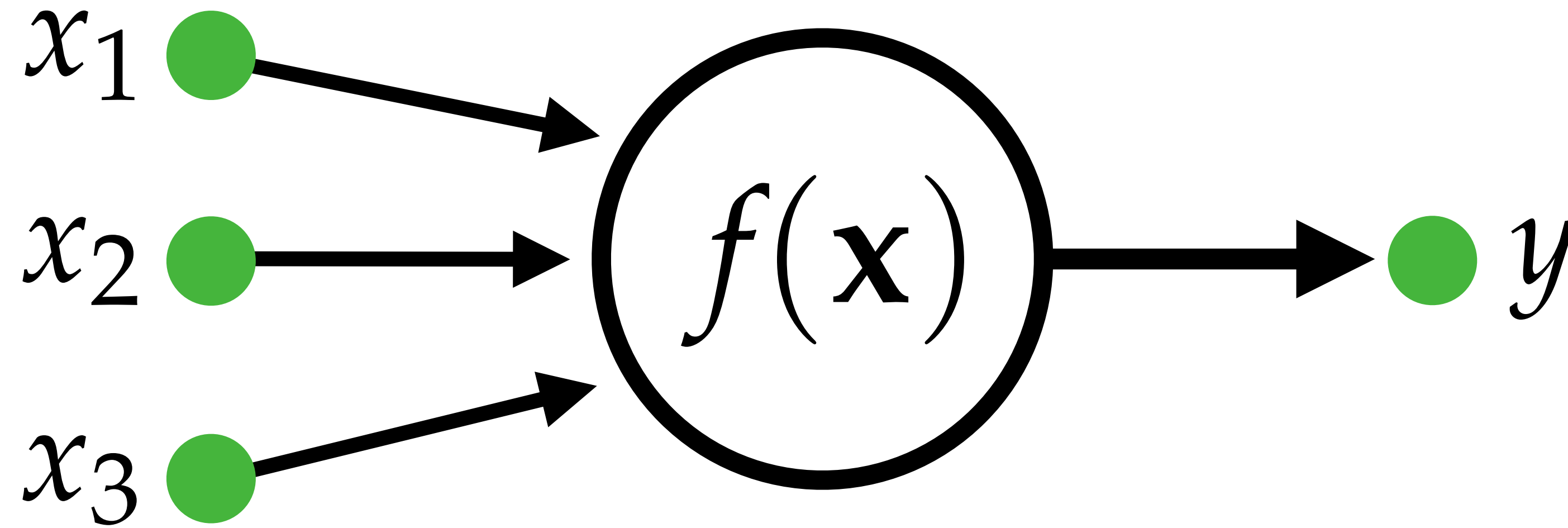


$$f(\mathbf{x}) = w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + b = \mathbf{w} \cdot \mathbf{x} + b$$

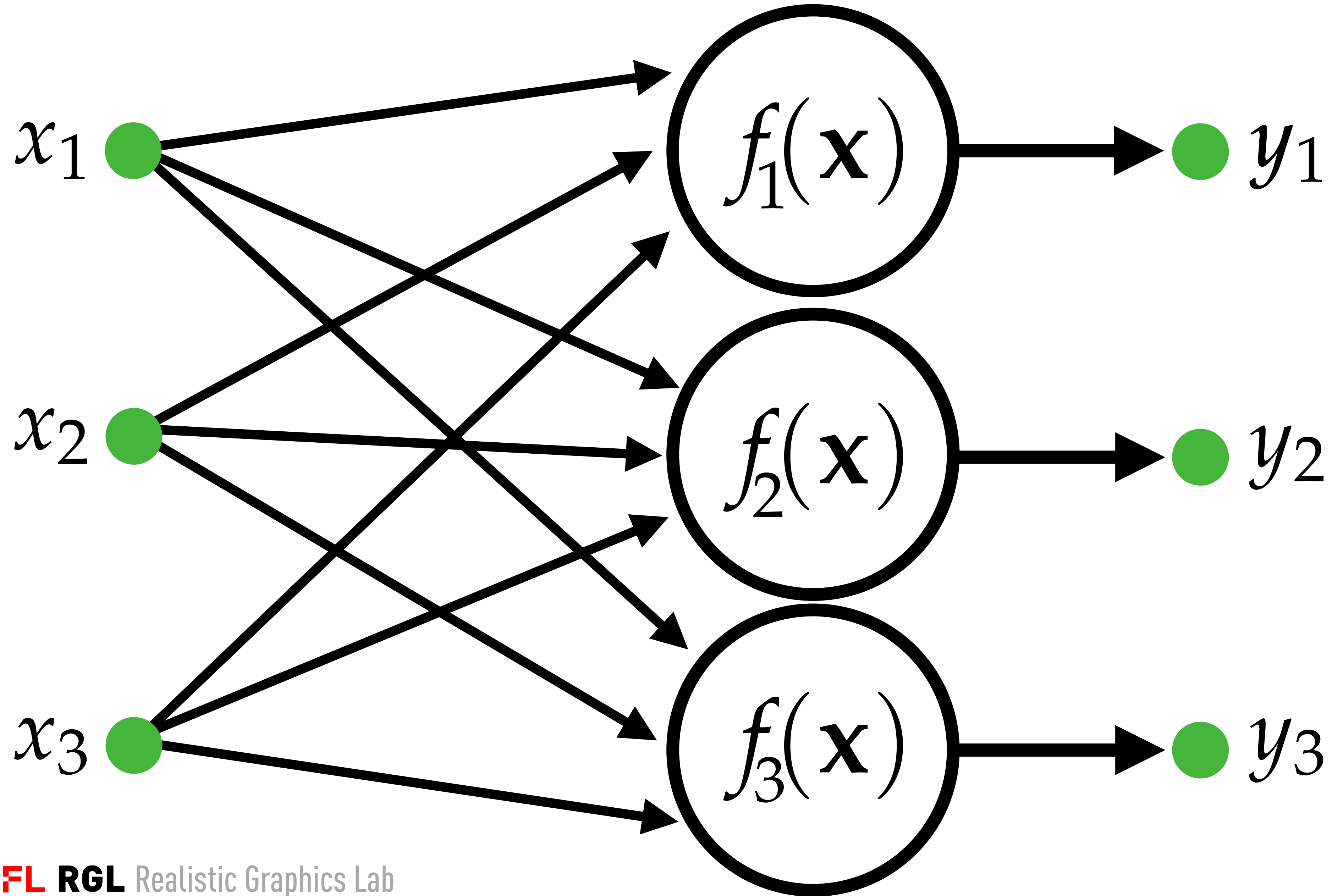
The equation shows the linear function $f(\mathbf{x})$ as a sum of weighted inputs and a bias. The weights w_1, w_2, w_3 and the bias b are highlighted in light blue circles. The vector \mathbf{w} is labeled "Weights" and the bias b is labeled "Bias".

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

An artificial neuron — linear case



An artificial neuron — linear case

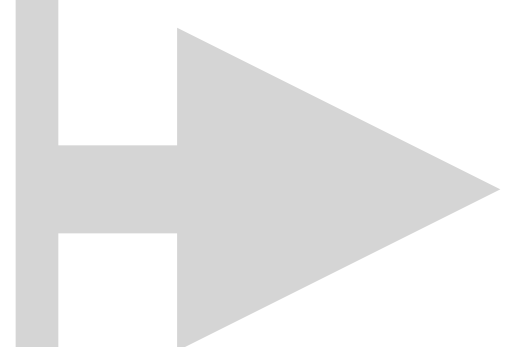


An artificial neuron — linear case

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

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

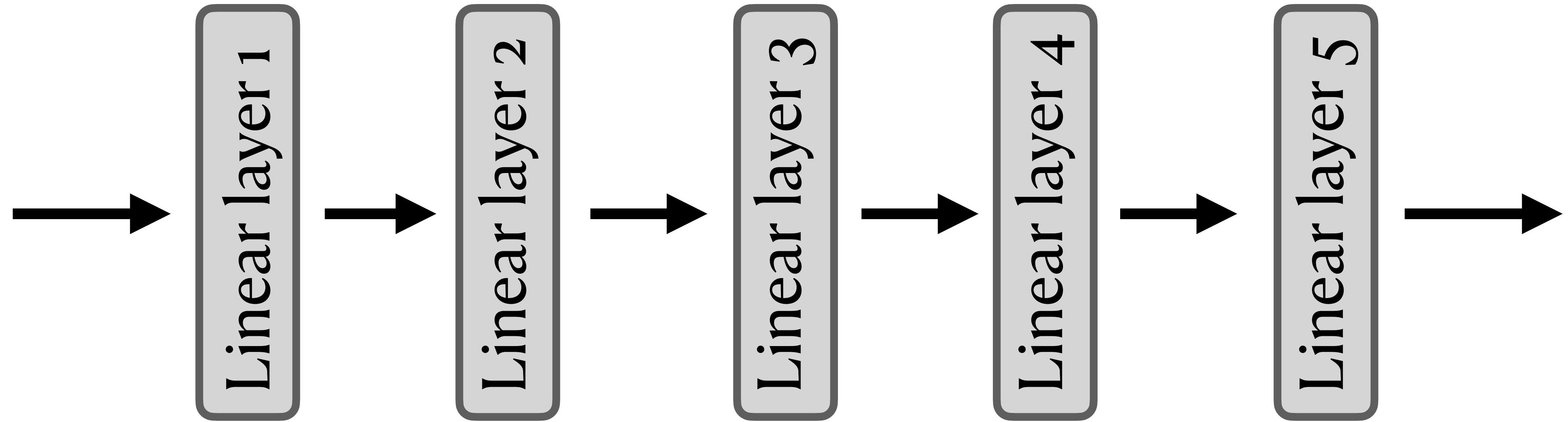
$$f_3(\mathbf{x}) = \mathbf{w}_3 \cdot \mathbf{x} + b_3$$



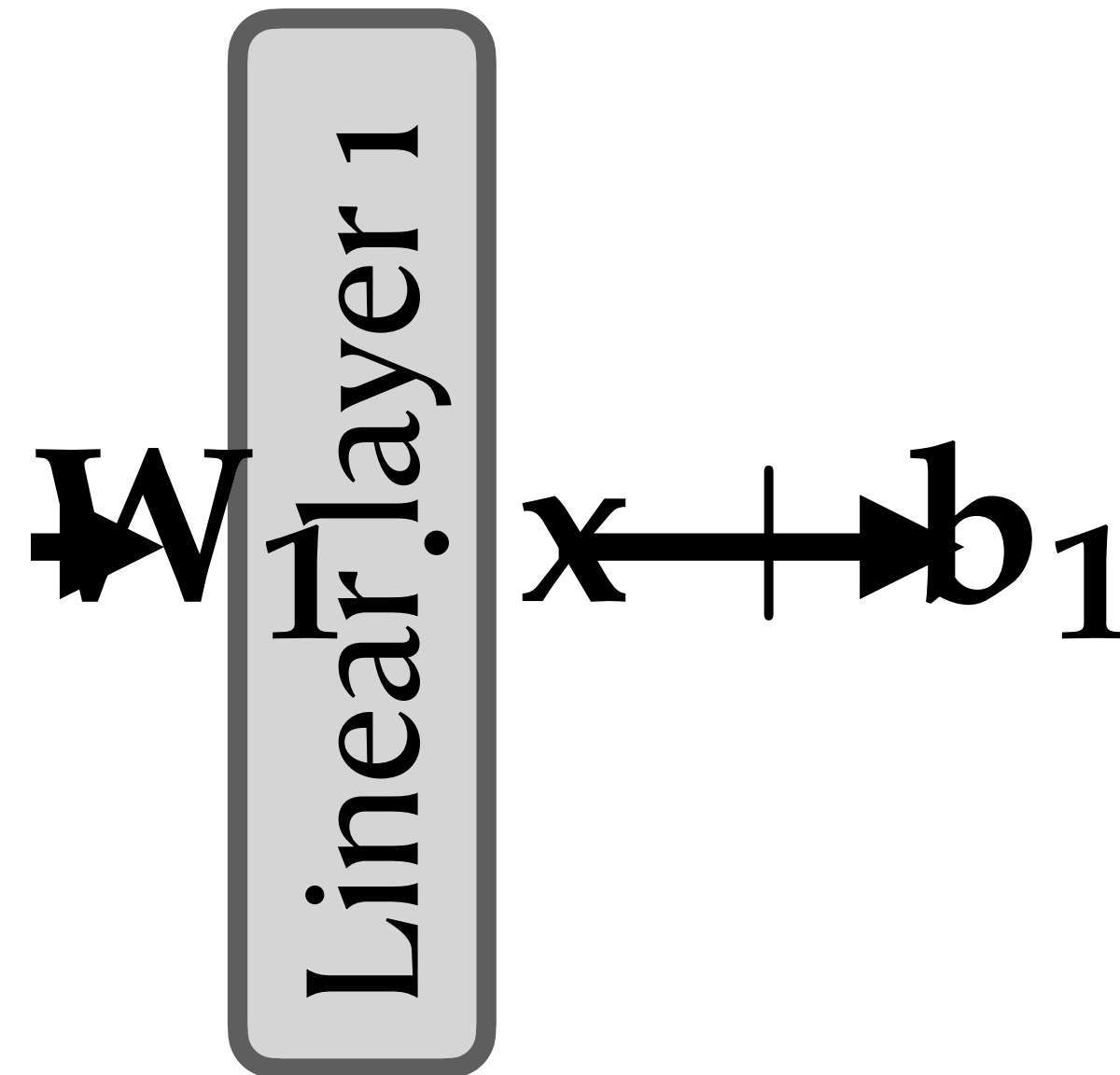
$$\mathbf{f}(\mathbf{x}) = \mathbf{W} \cdot \mathbf{x} + \mathbf{b}$$

"Weight matrix" "Biases"
| |

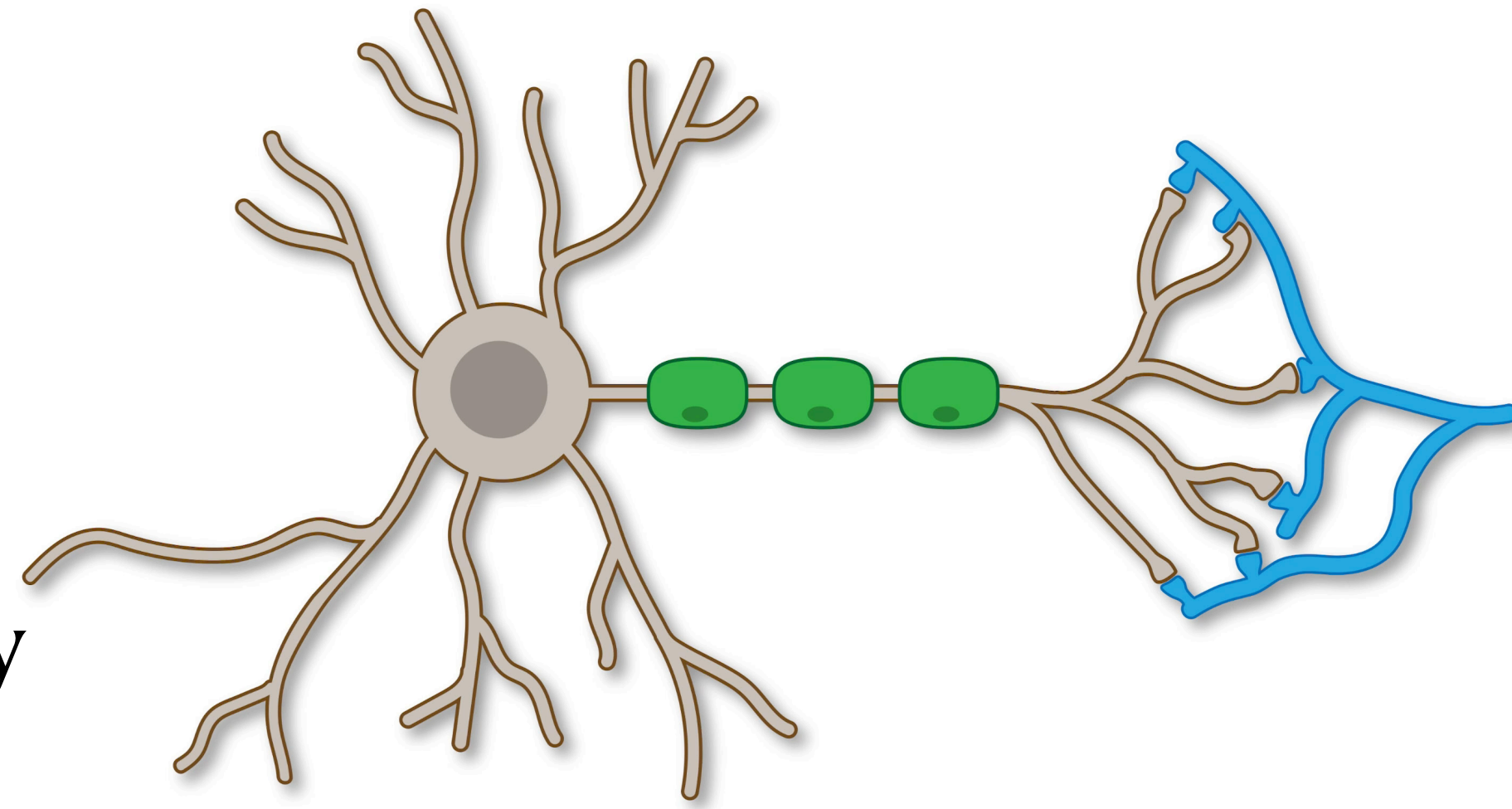
The linear/affine case isn't interesting!



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

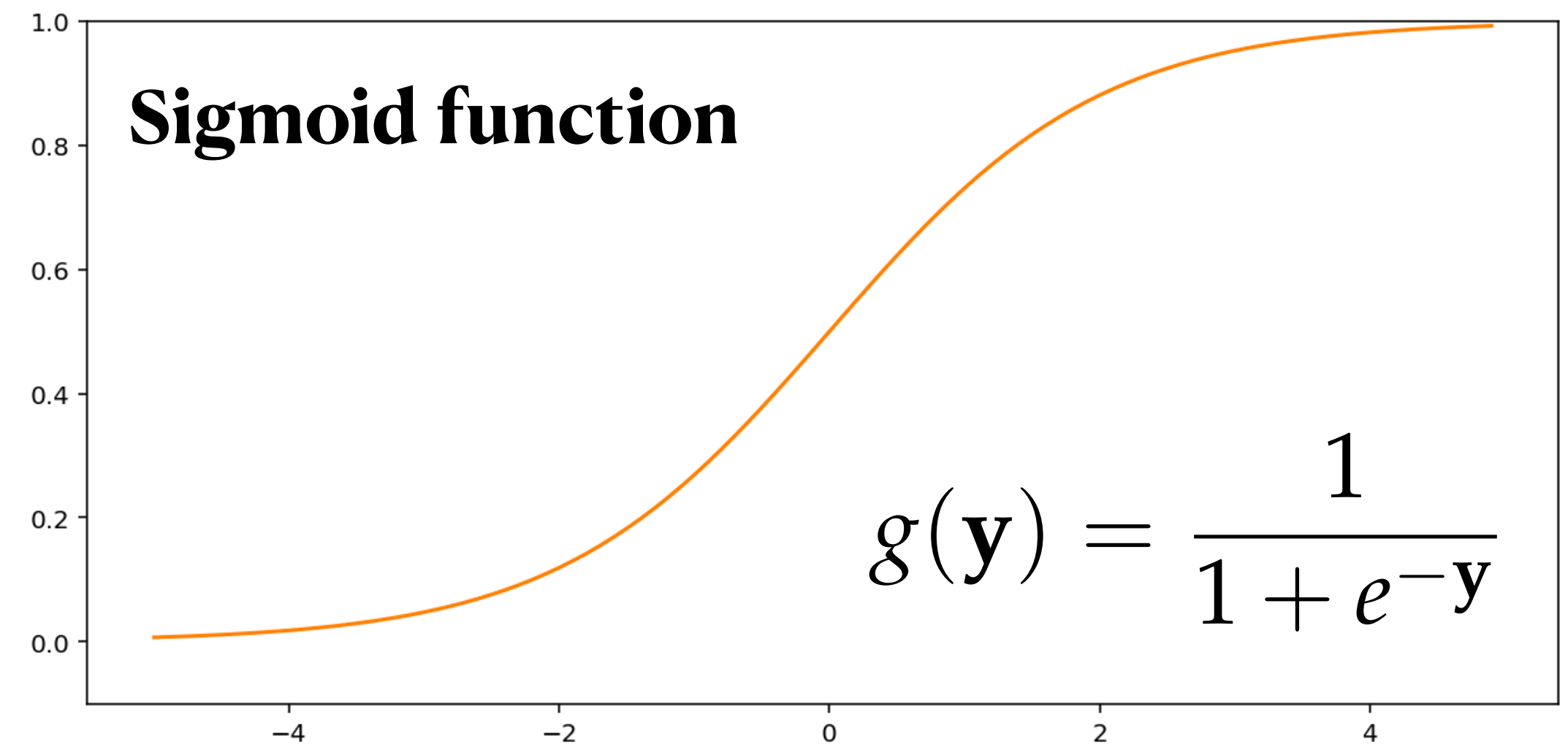
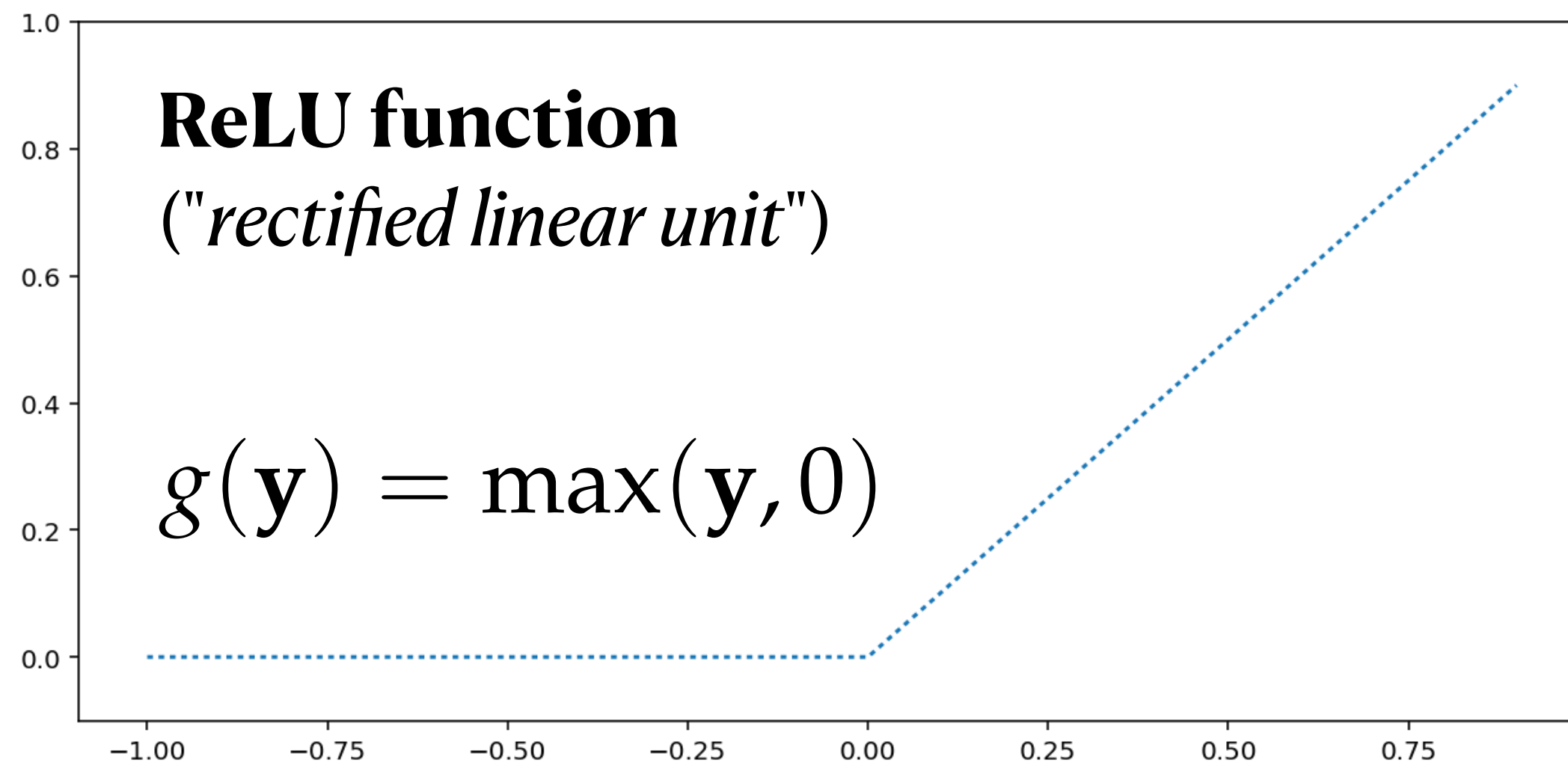


Activation functions



- 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

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

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

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?

Labeled input/output data

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

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

$\text{NN}(\mathbf{x}_1) \approx \mathbf{y}_1?$

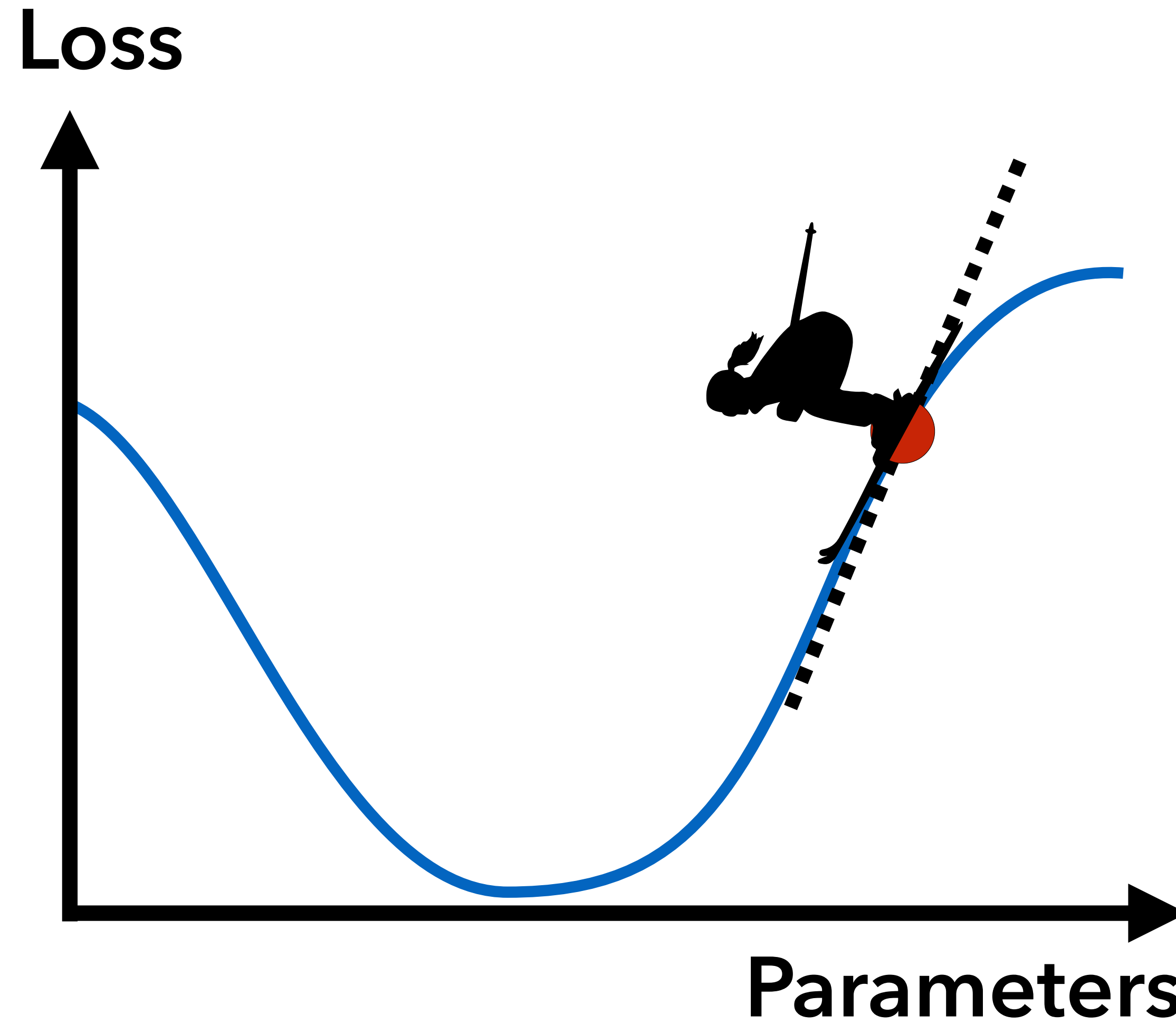
These are unlikely to agree at first.

Quantify discrepancy using a **loss function**:

Example: L_2 loss

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

Gradient descent



Gradient descent

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

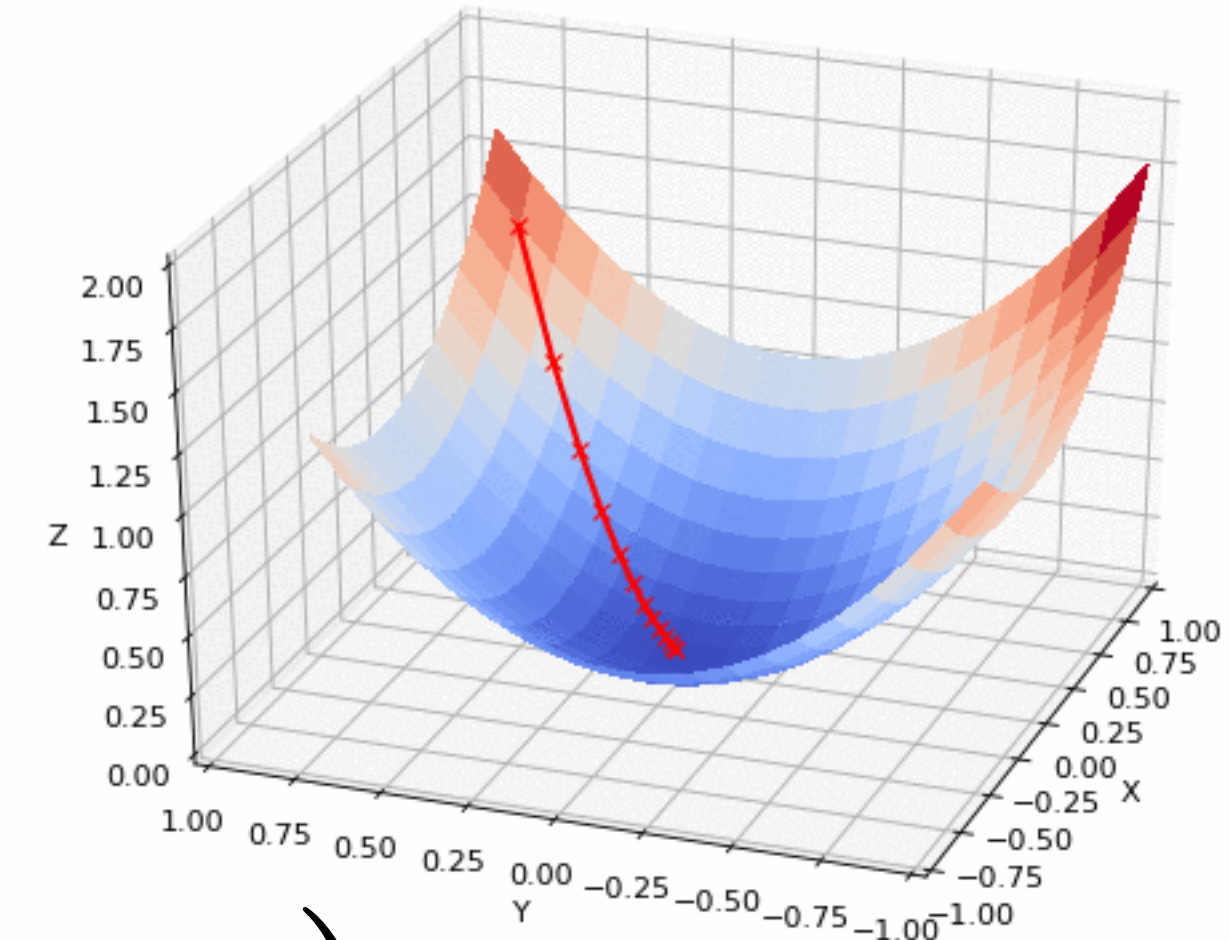
- Every iteration takes a step in the negative gradient direction

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).

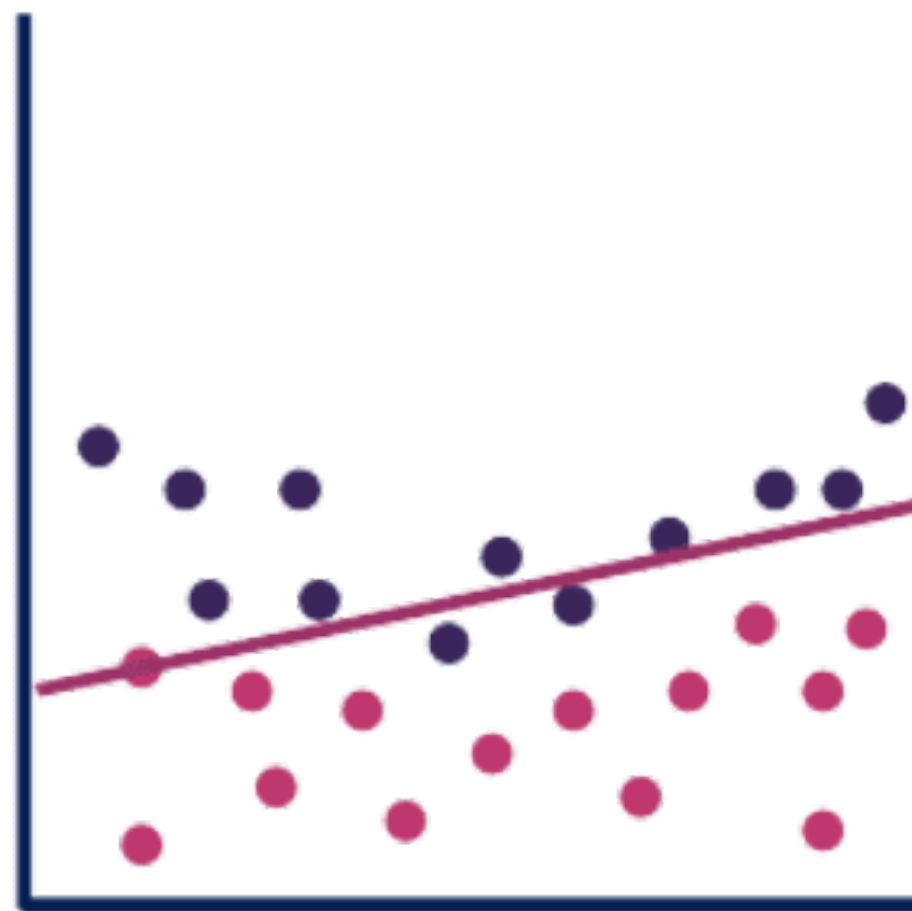
High dimensional spaces

- Derivatives 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 breakthroughs in ML in the last years. You *cannot* train a neural network without gradients.



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

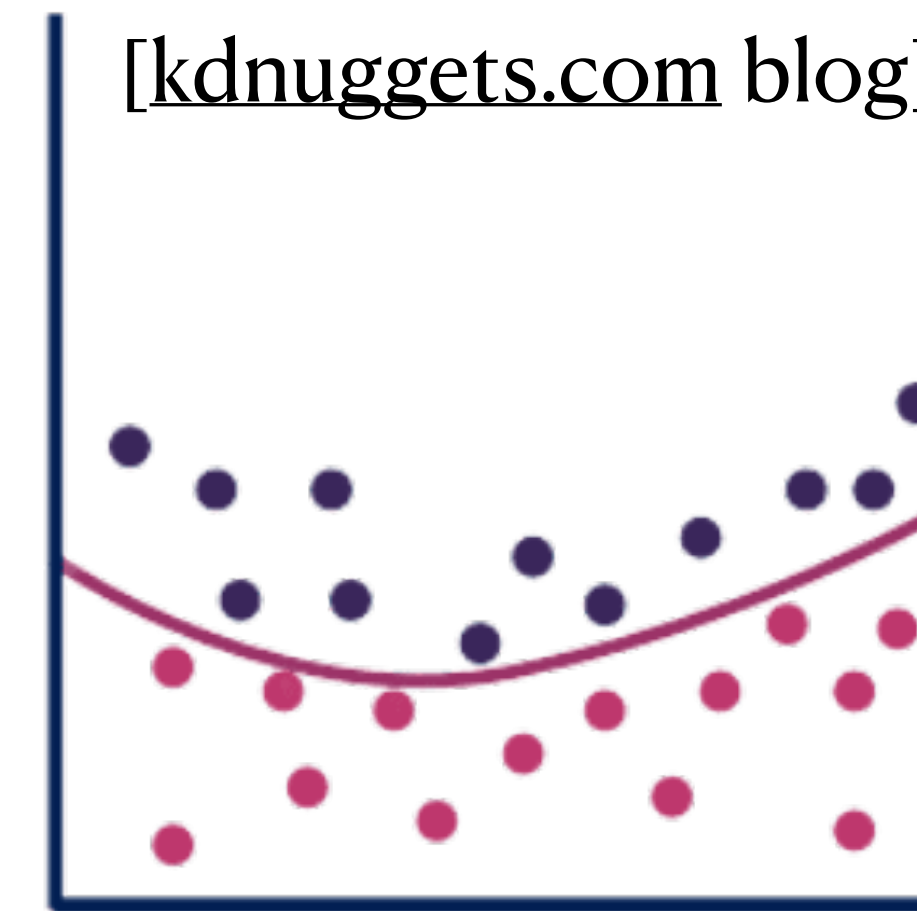
Underfitting vs overfitting



Underfitting



Overfitting

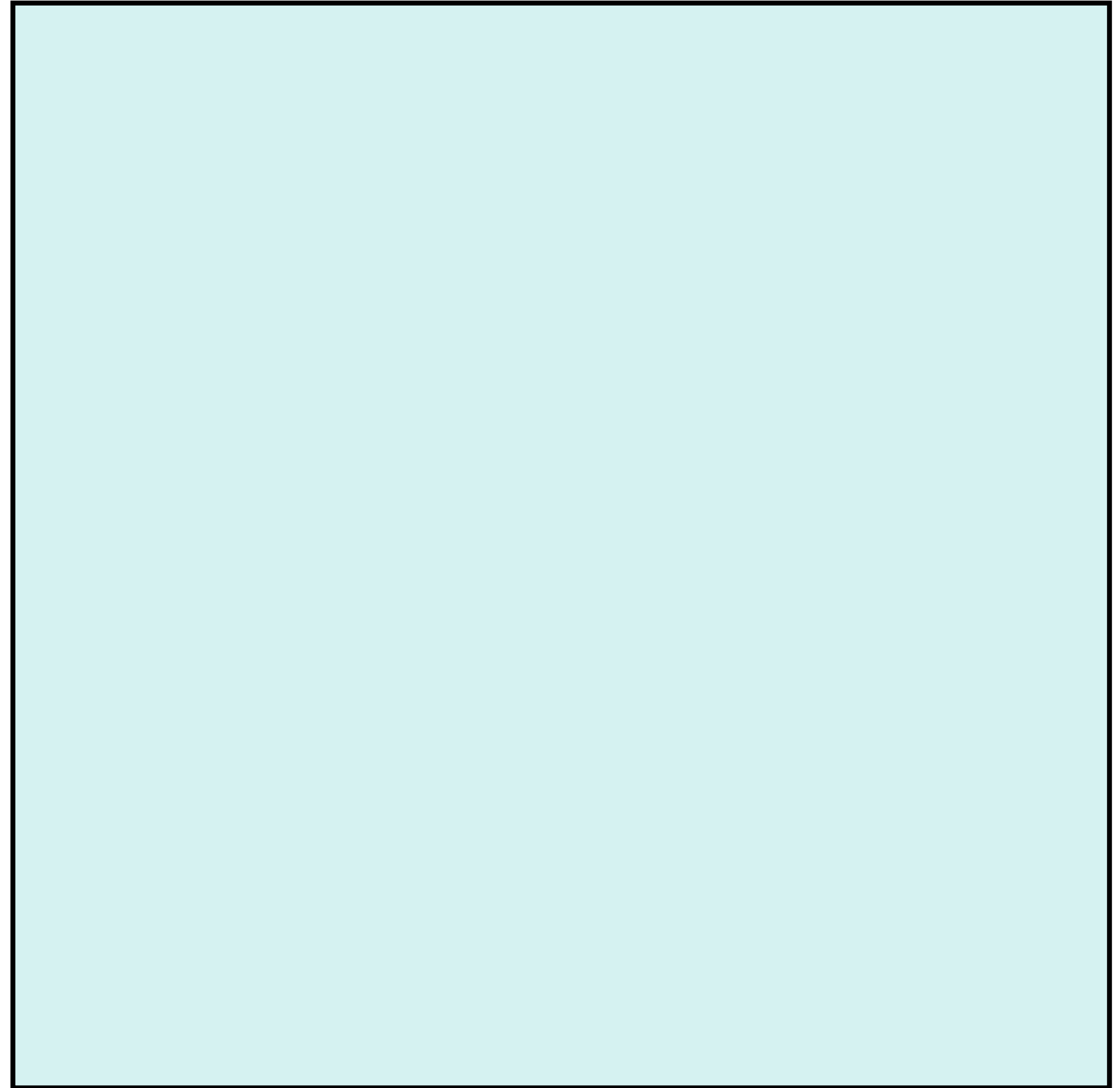
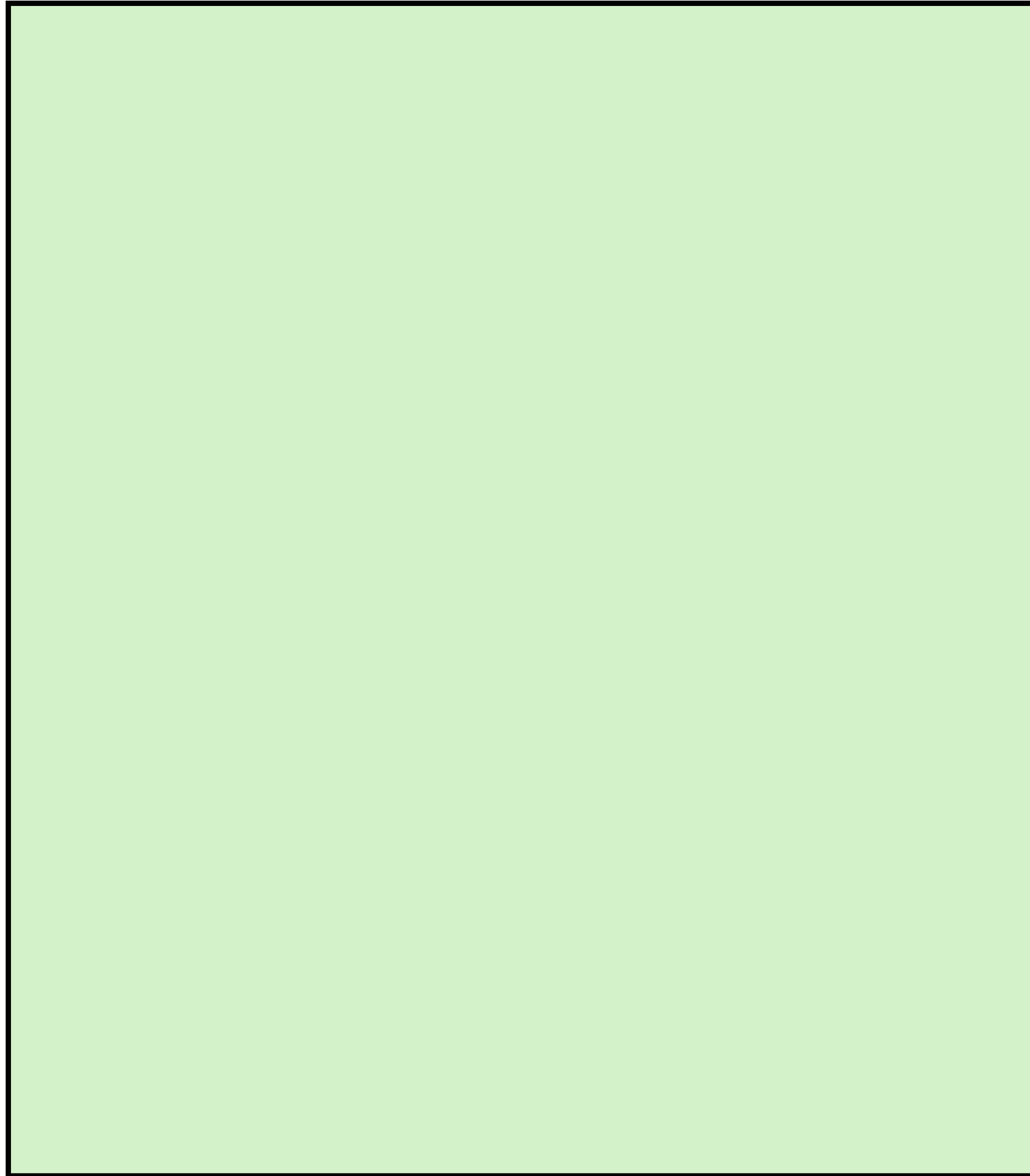


Balanced

[kdnuggets.com blog]

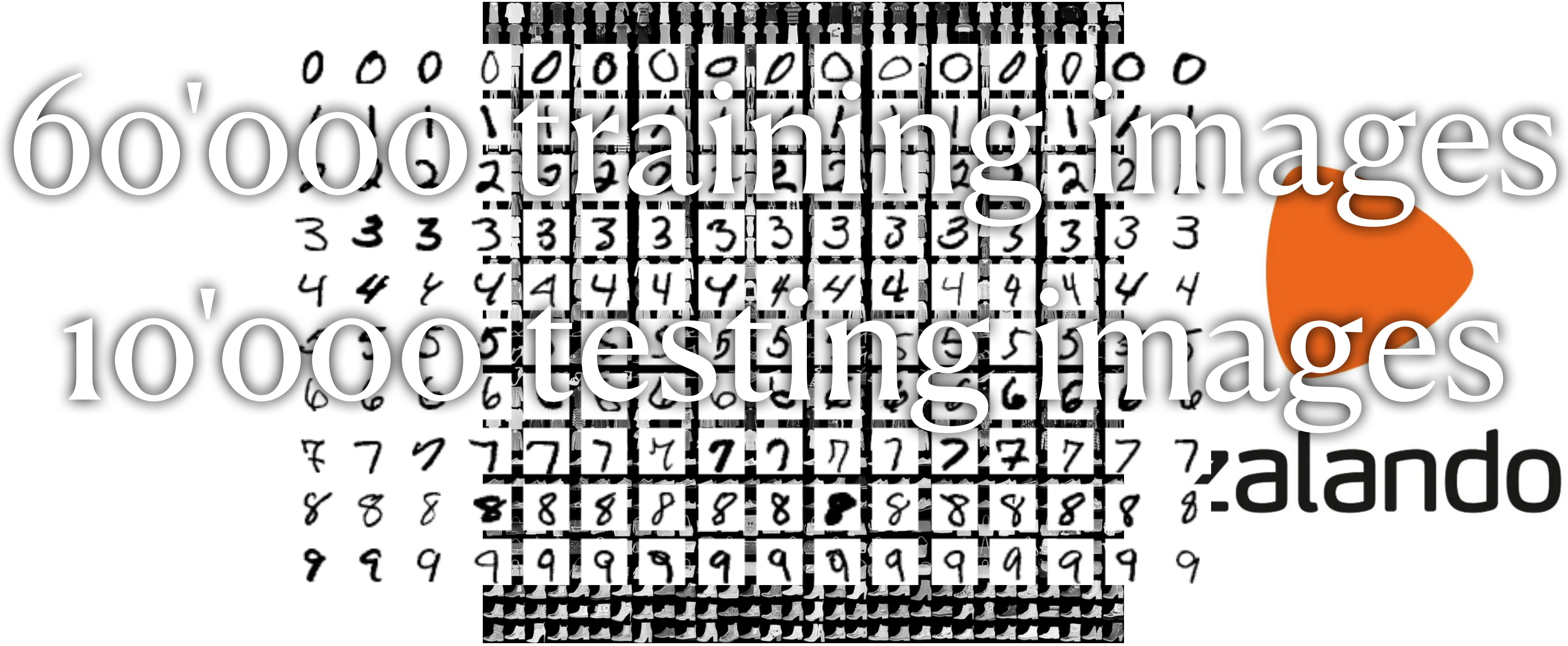
Neural nets often have enough coefficients that they can in principle memorize the entire dataset.

Dangers of training with data



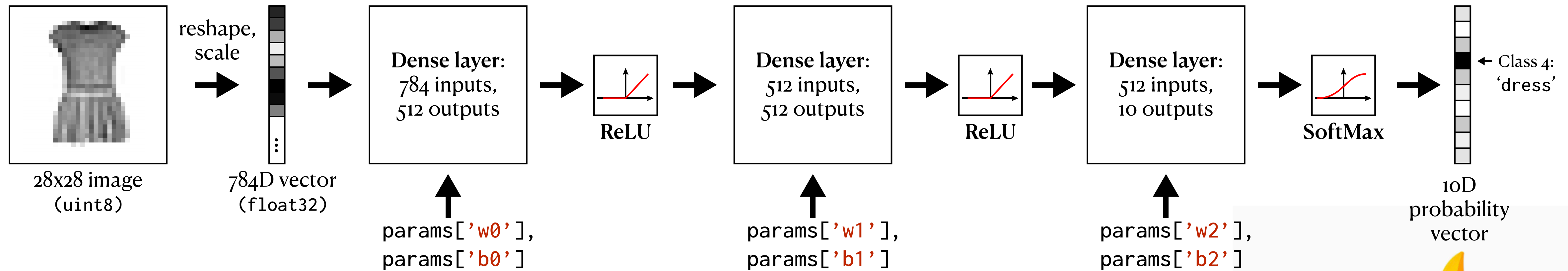
A benchmark problem

"MNIST" and "Fashion-MNIST"



zalando

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

Epoch: 000,000 | Learning rate: 0.03 | Activation: Tanh | Regularization: None | Regularization rate: 0 | Problem type: Classification

DATA: Which dataset do you want to use? (Icons for various datasets)

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

REGENERATE

FEATURES: Which properties do you want to feed in?

- X_1
- X_2
- X_1^2
- X_2^2
- X_1X_2
- $\sin(X_1)$
- $\sin(X_2)$

2 HIDDEN LAYERS

- 4 neurons
- 2 neurons

OUTPUT: Test loss 0.667, Training loss 0.663

Colors shows data, neuron and weight values.

Show test data Discretize output

The outputs are mixed with varying **weights**, shown by the thickness of the lines.

This is the output from one **neuron**. Hover to see it larger.

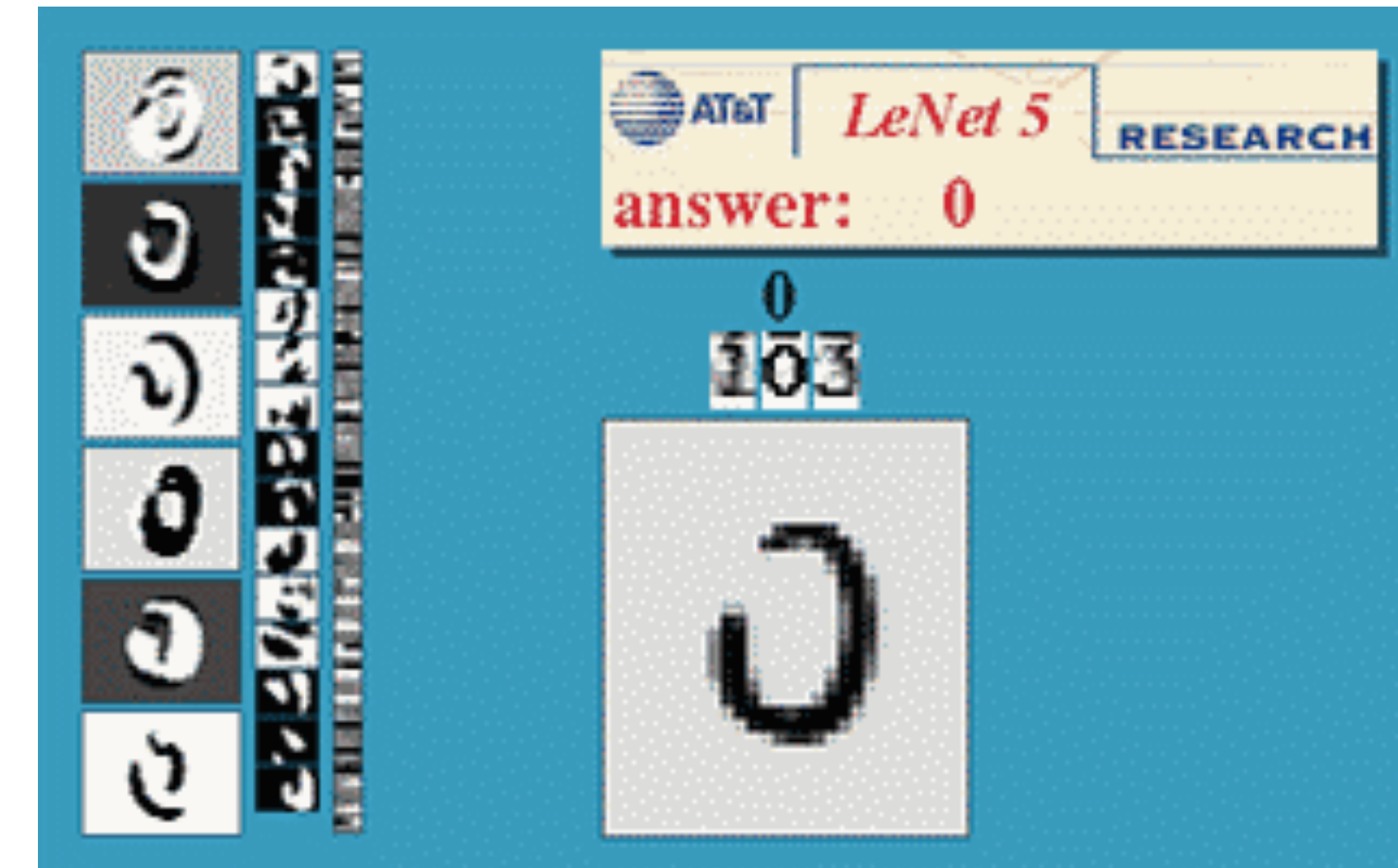
History of neural nets

“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"

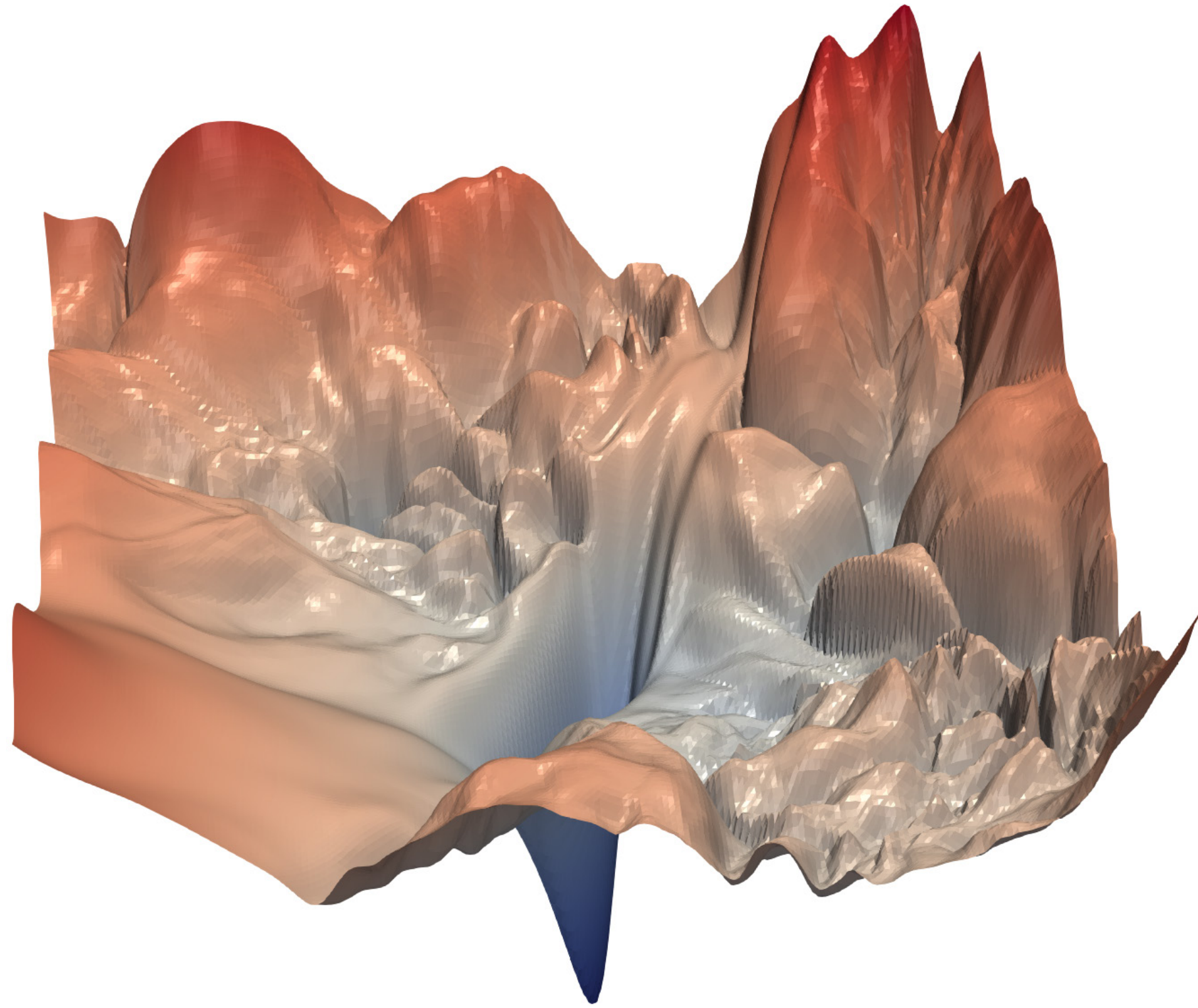
- What we **know**.
 - How all the coefficients were obtained (it is a deterministic calculation after all).
 - Which neural network components work *well*, and which of them work *well together*.
 - When one ingredient works better than another, we can sometimes find a nice mathematical intuition motivating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

MidJourney: a hand-written booklet of cooking recipes

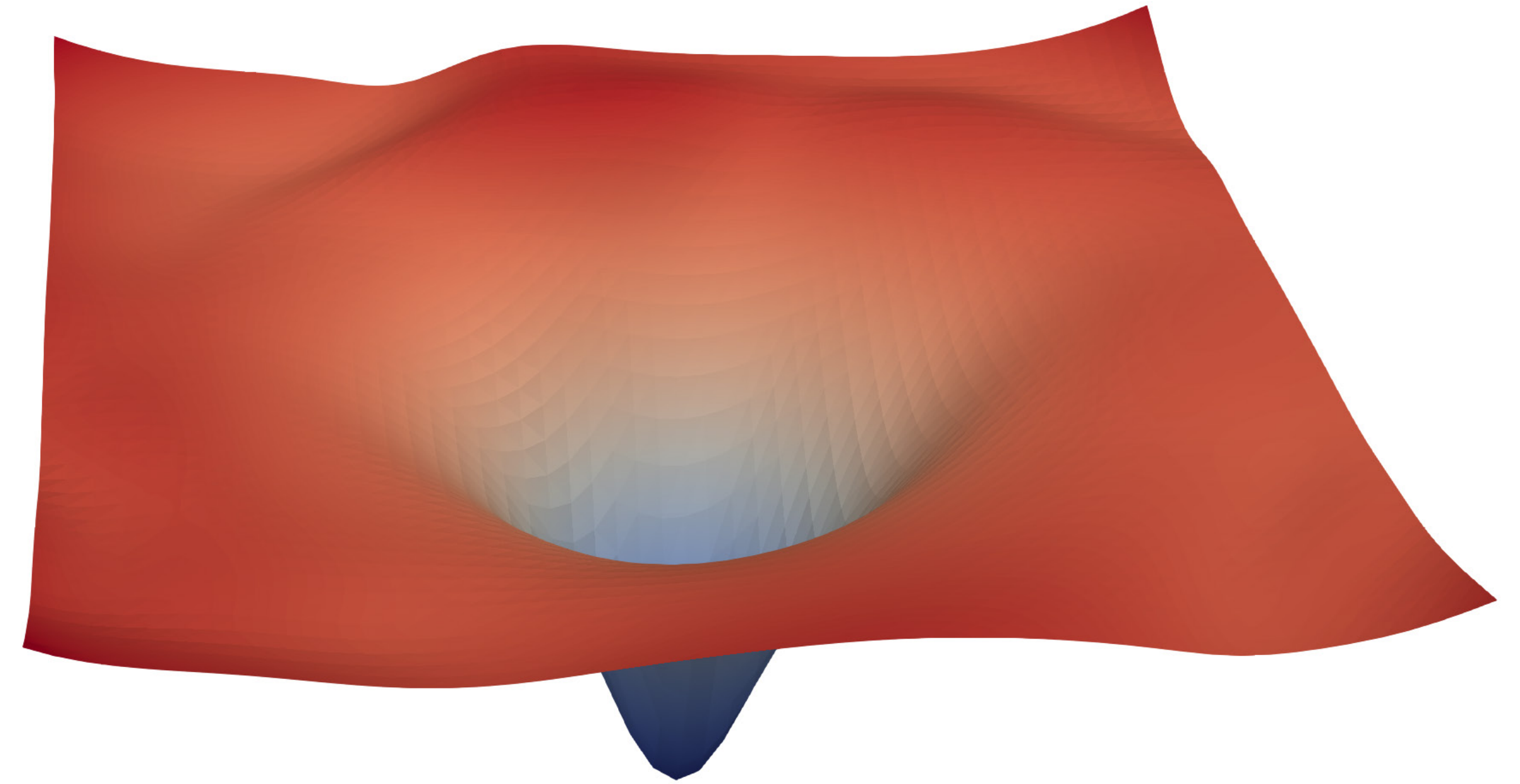


Optimization landscape

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



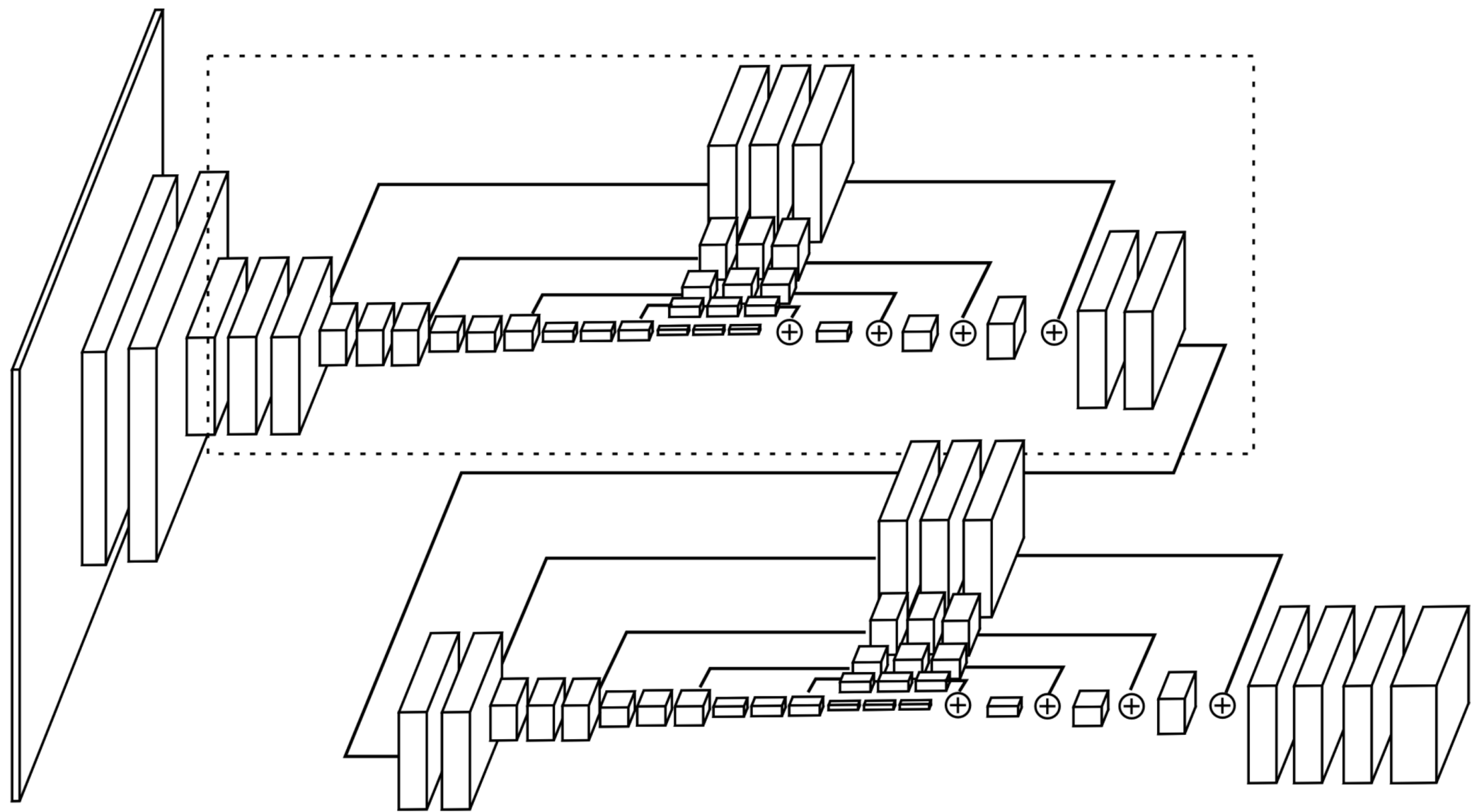
A "bad" network architecture (unstable training)



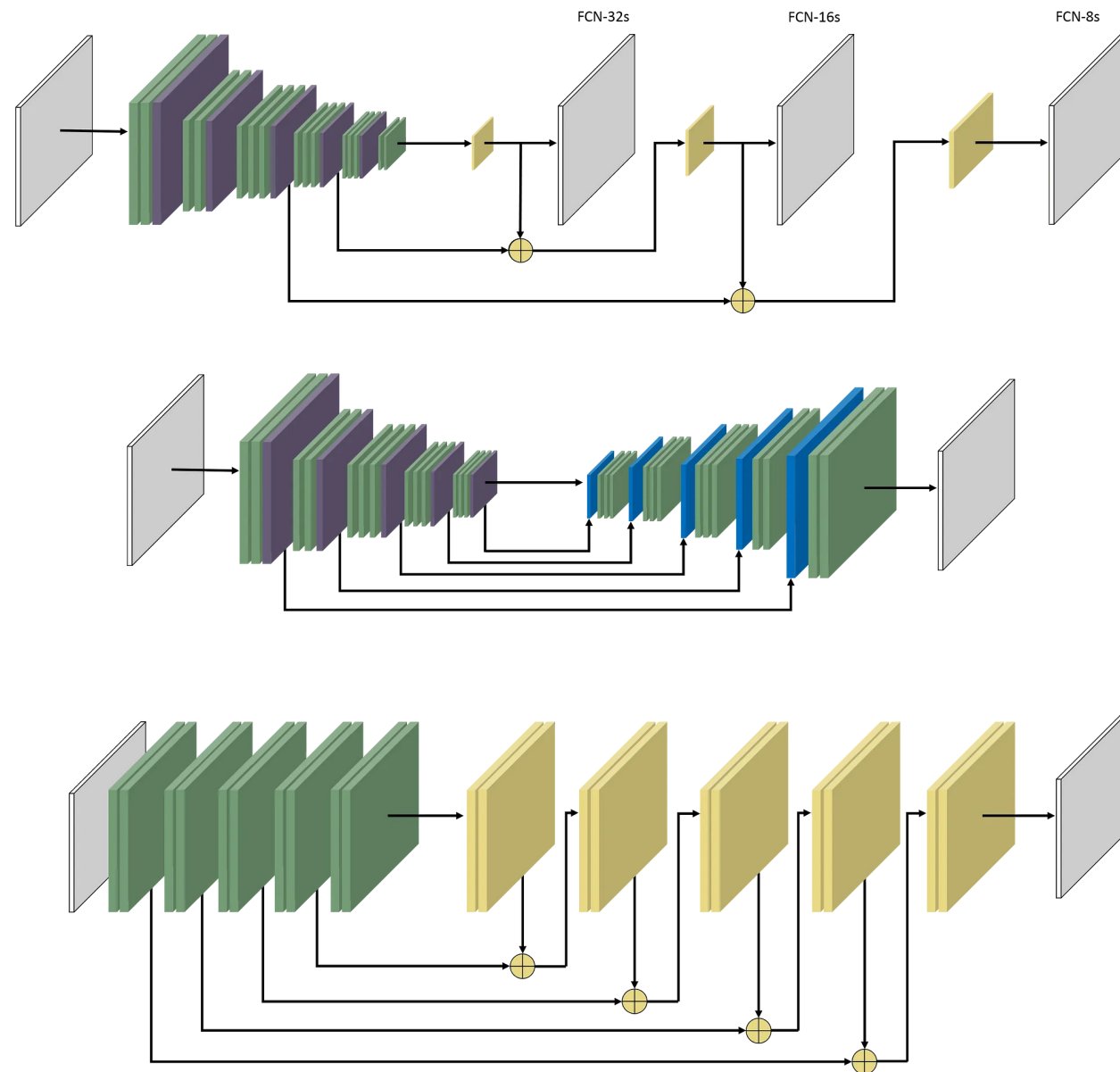
A "good" network architecture (unstable training)

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

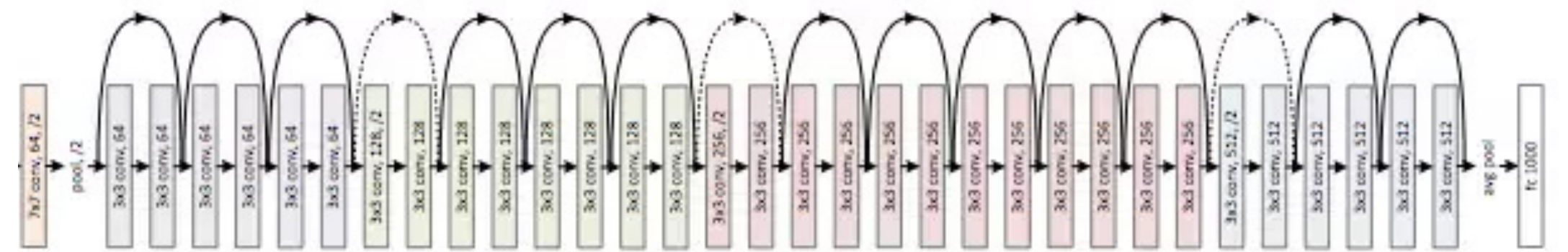
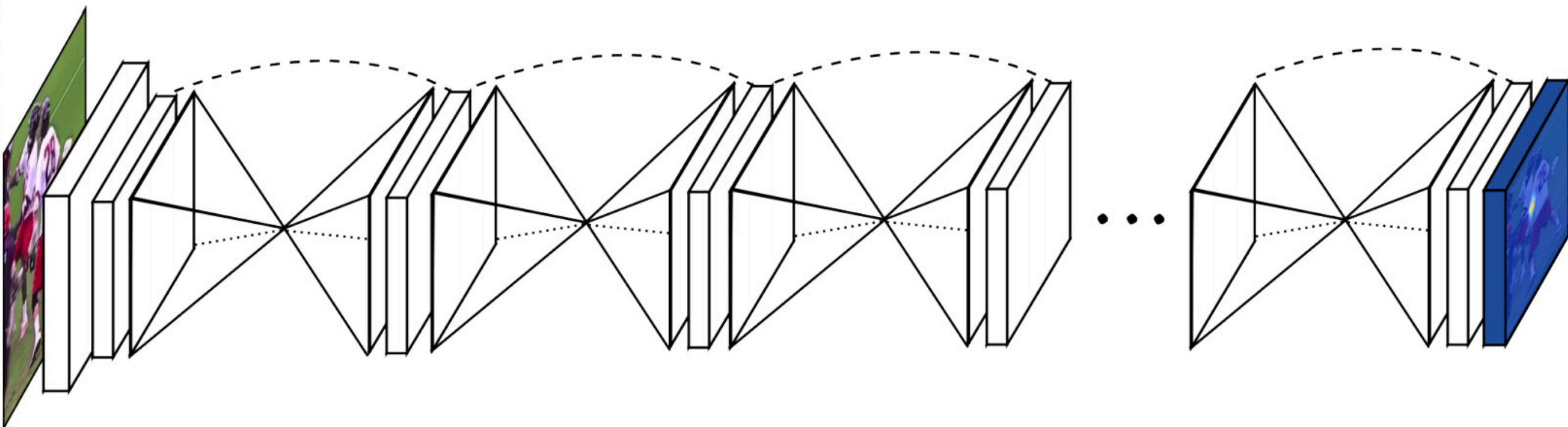
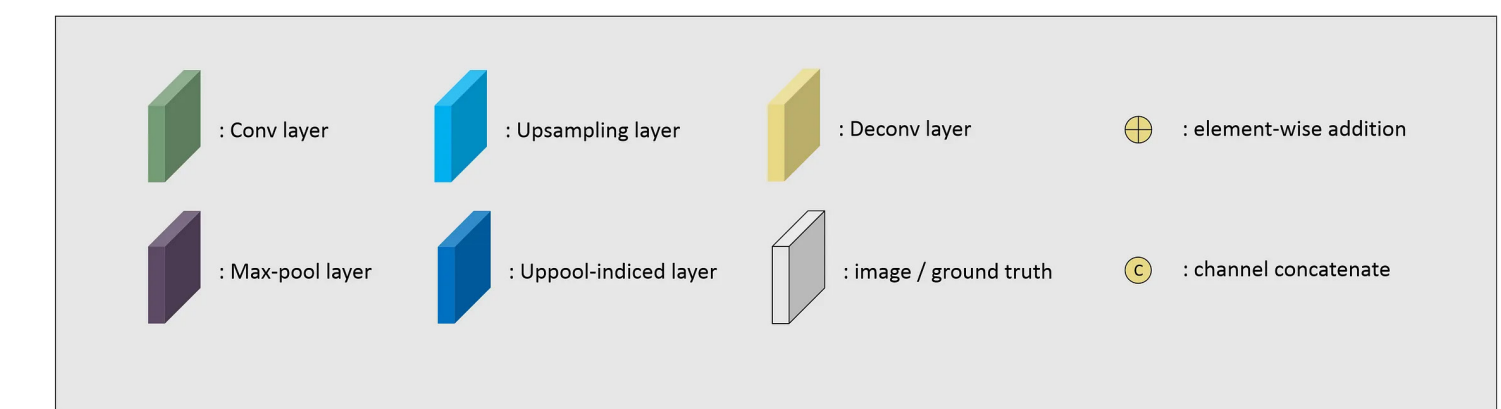
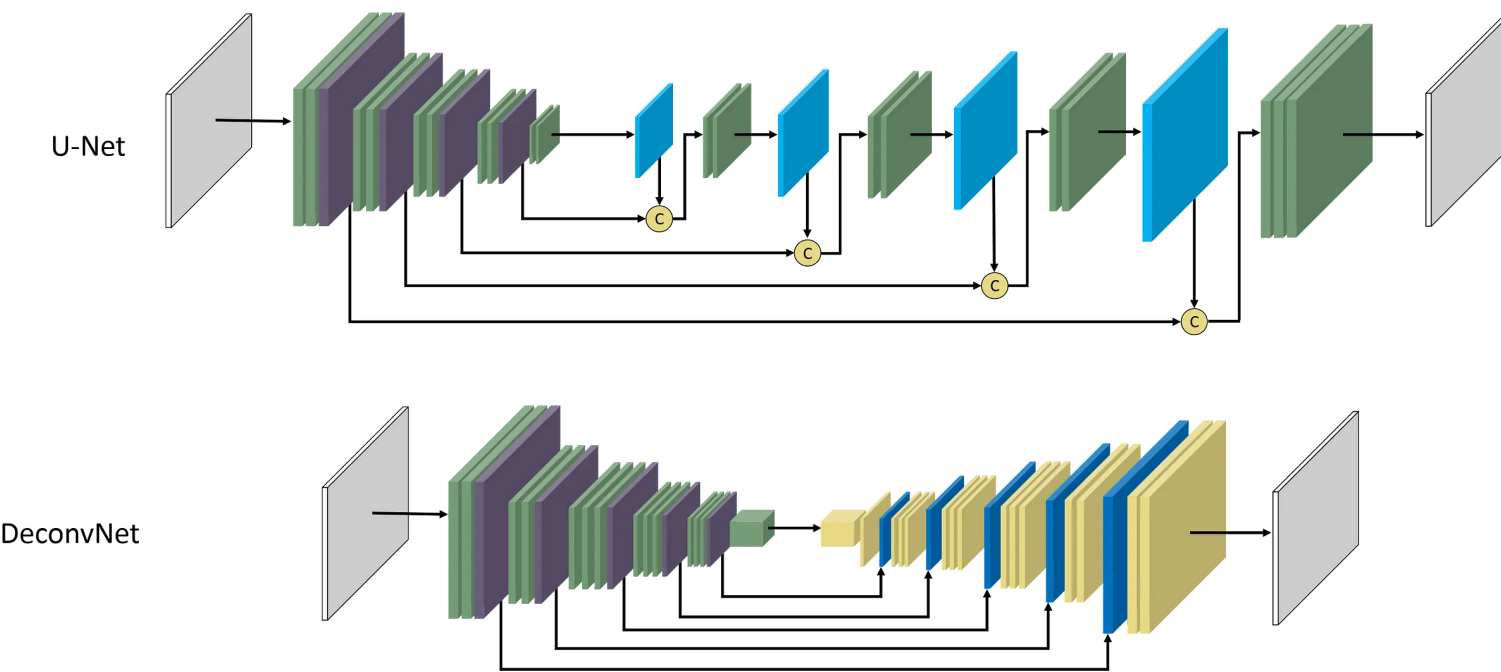
An architectural zoo



[Newell et al .2016]



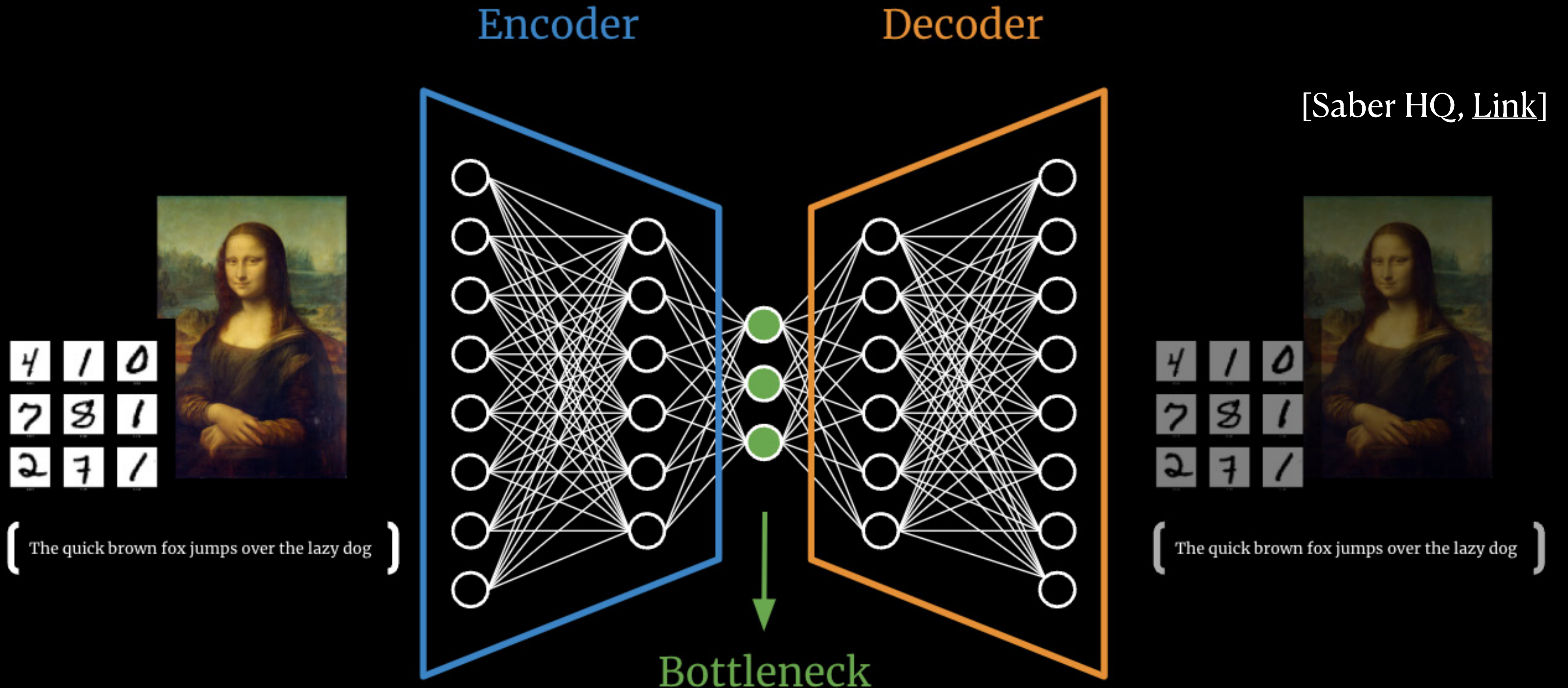
[Sunner Li, Link]



[He et al .2015]

Latent vectors and bottlenecks

Encoder-decoder architectures and related "computational games"



[Saber HQ, Link]

GPUs play an outsized role in this space

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



```
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




How can I help you today?

Compare business strategies for transitioning from budget to luxury vs. luxury to bu...

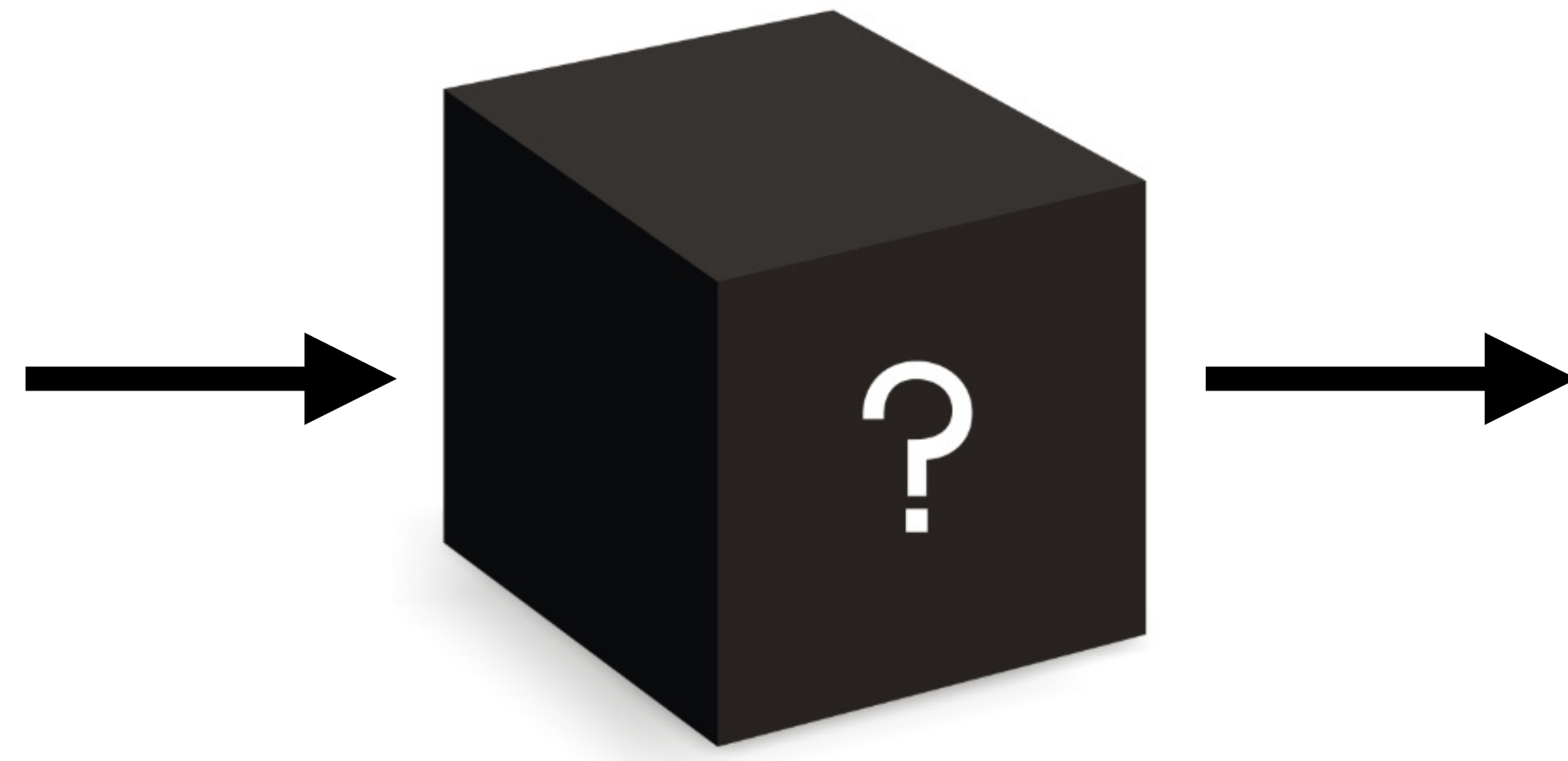
Plan a trip to experience Seoul like a local

Show me a code snippet of a website's sticky header

Help me study vocabulary for a college entrance exam

Message ChatGPT... 

Machine translation



Traditional vs neural approach



[DE]

[Meta M2M
model, Link]

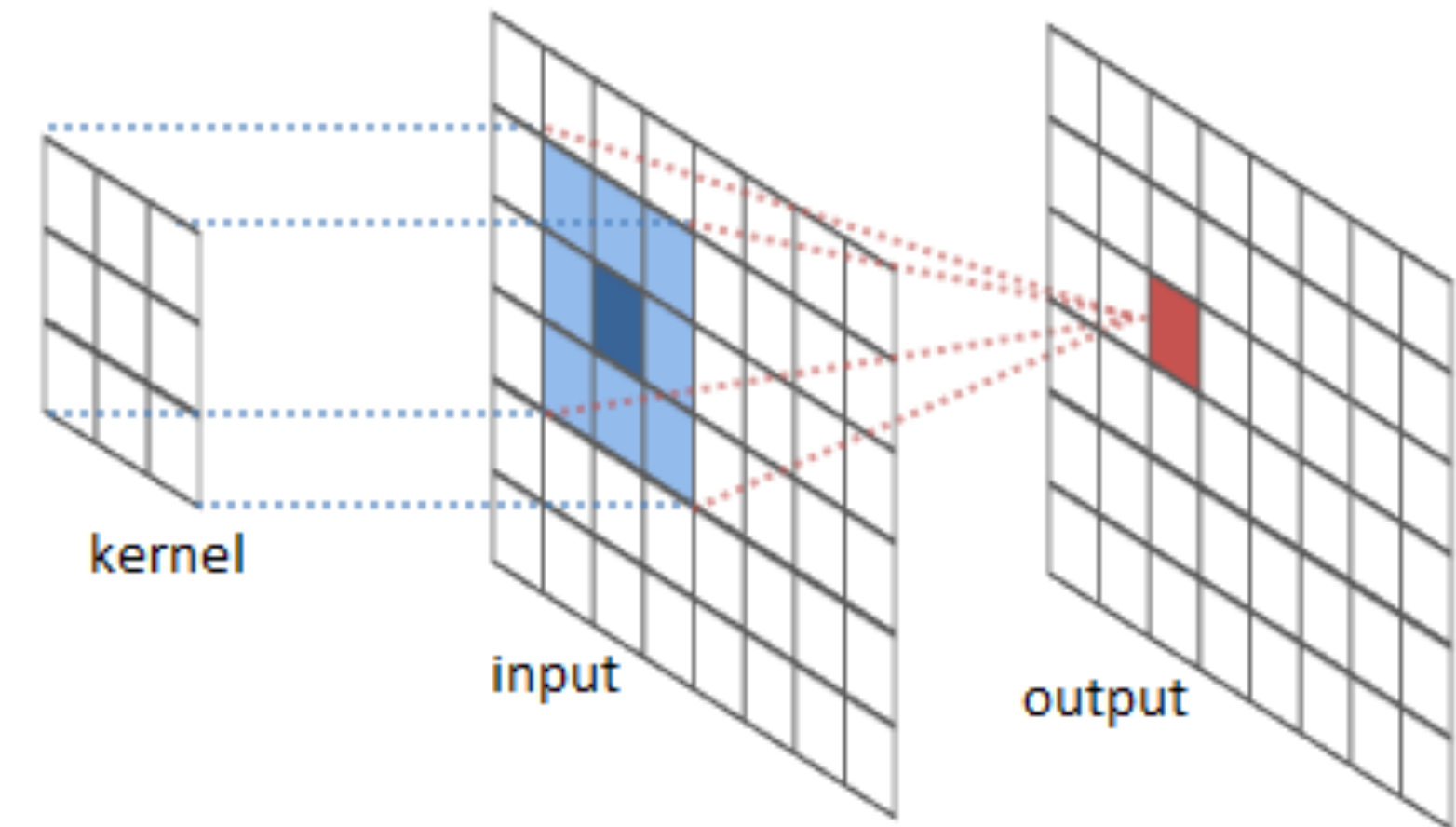
[EN]

Convolutional Neural Networks (CNNs)

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

- Local kernel

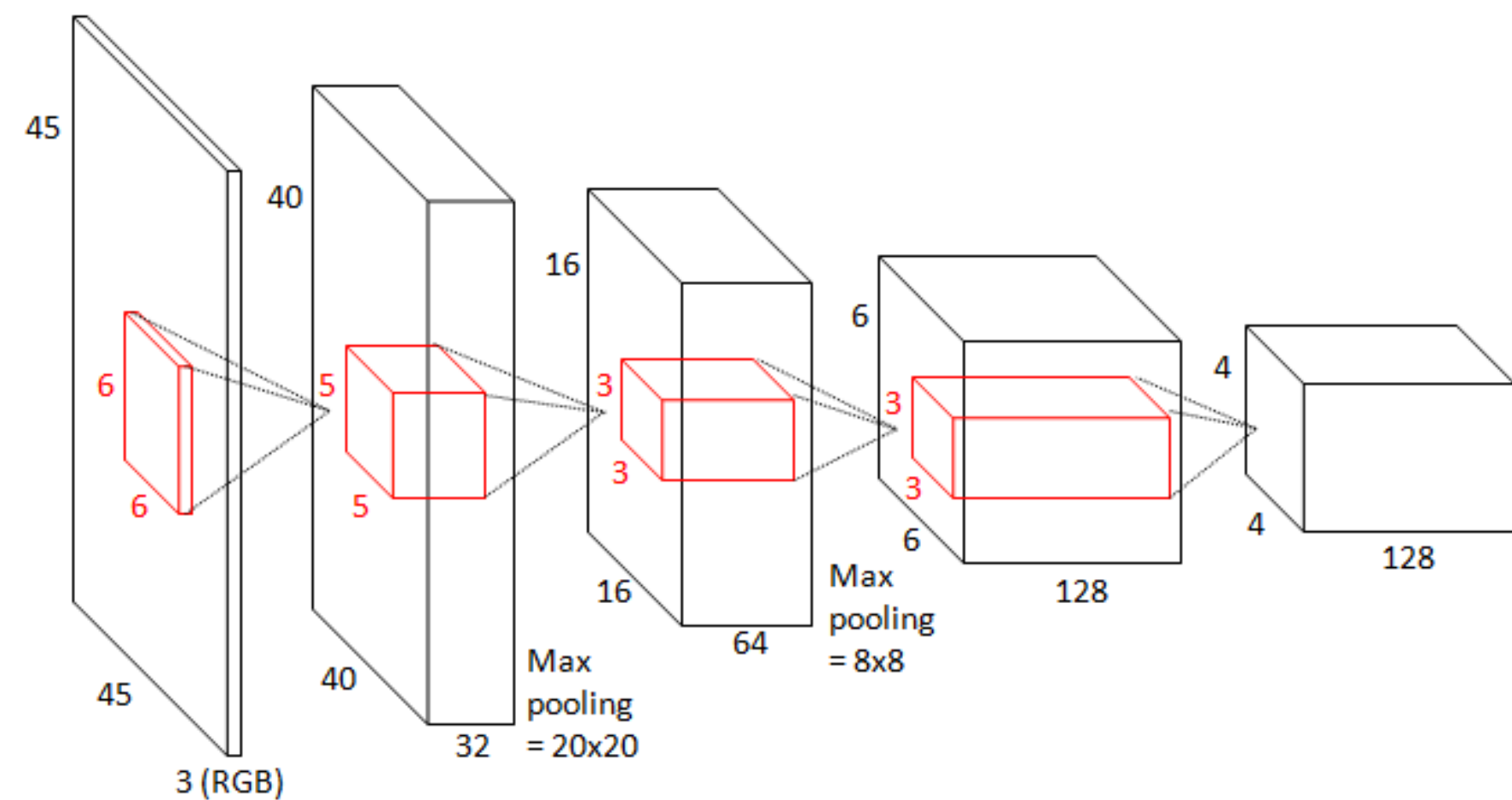
$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



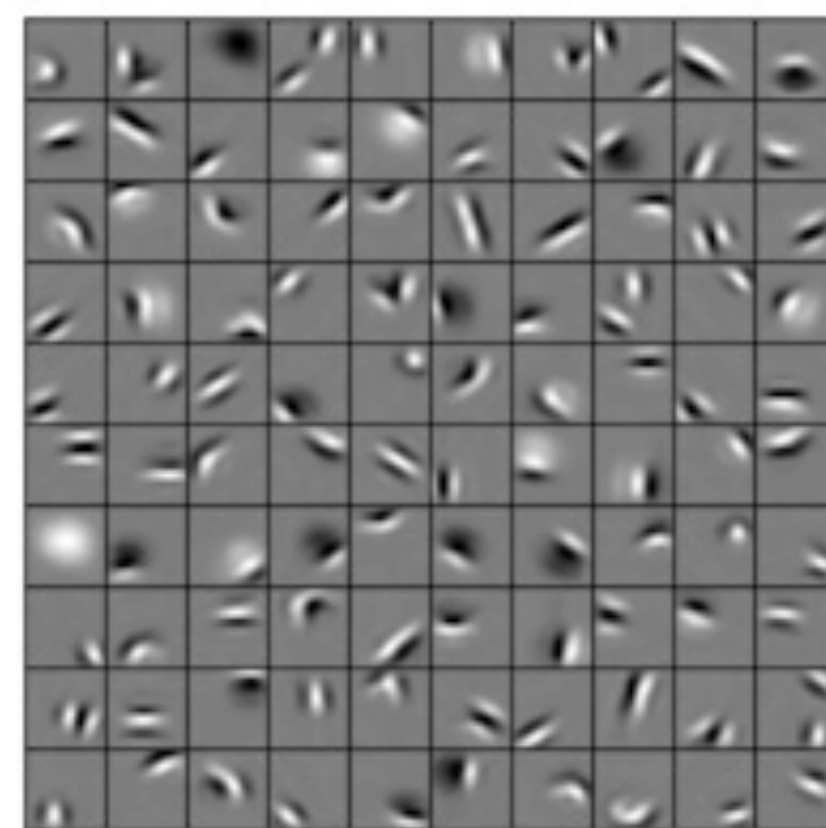
Convolutional Neural Networks (CNNs)

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

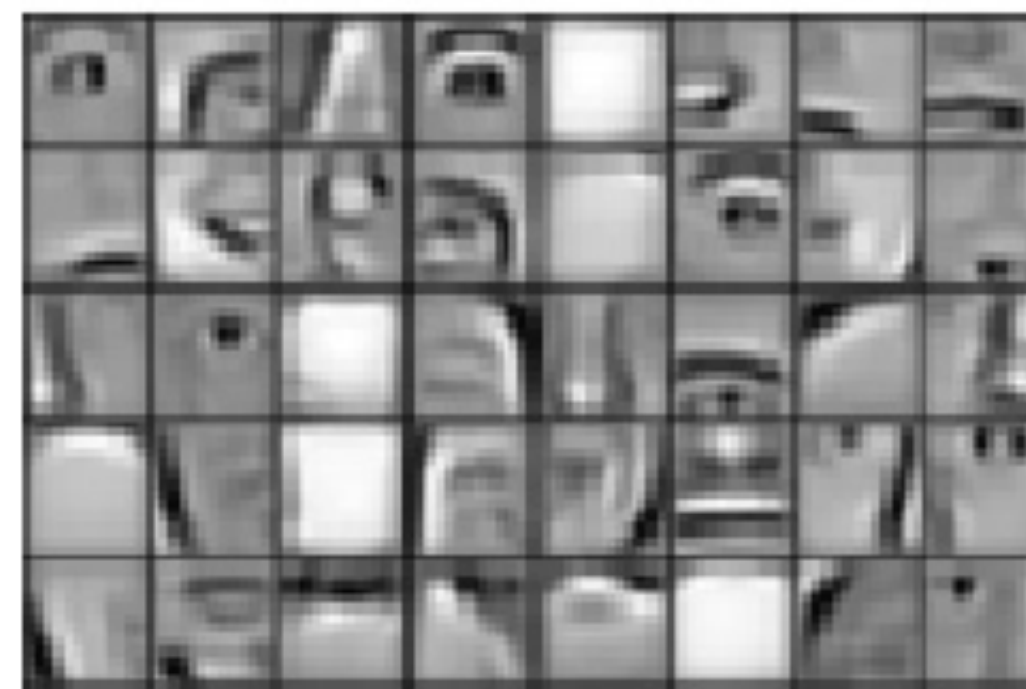
- Weight sharing due to local nature of operations



Low-level



Mid-level

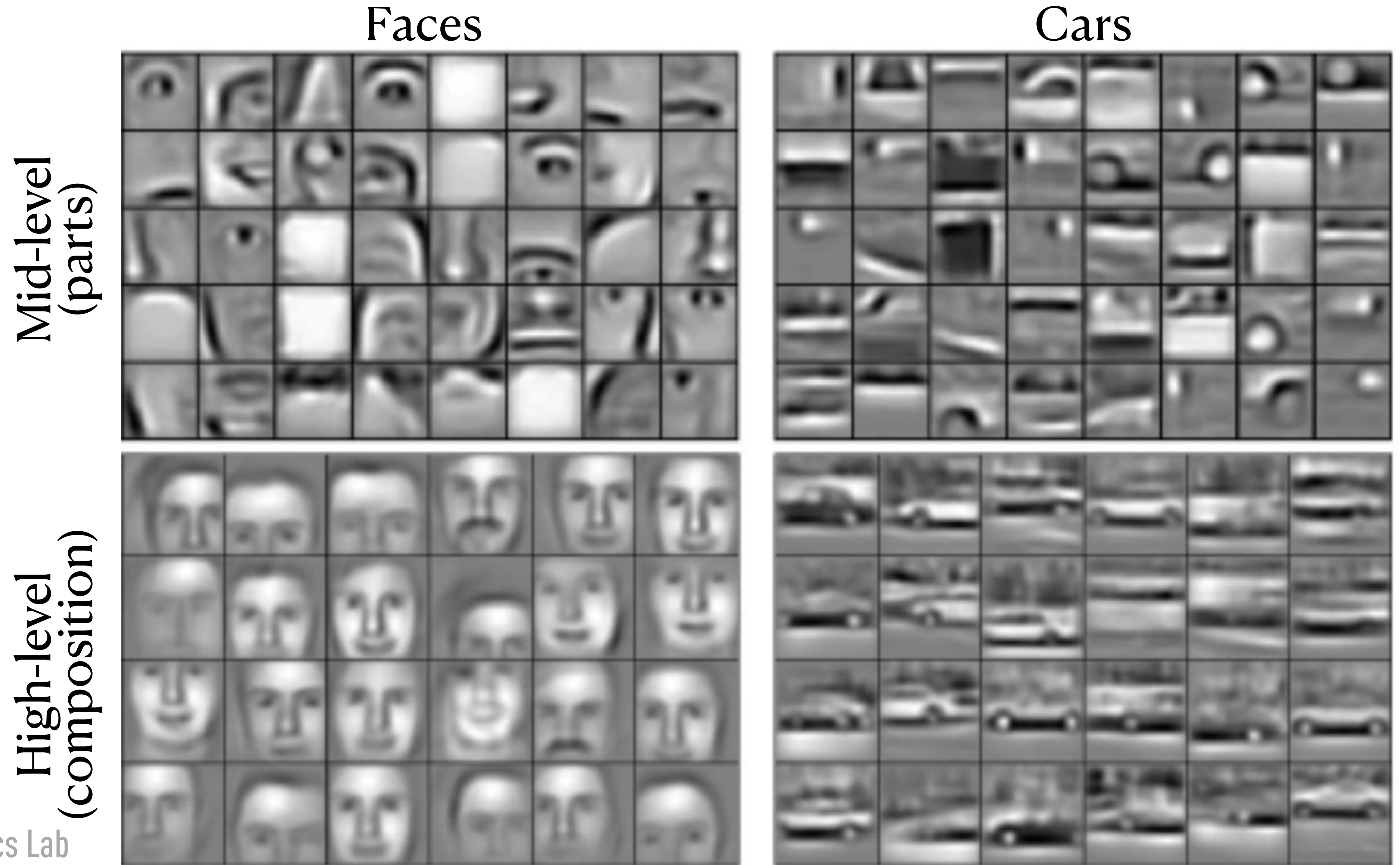


High-level



Convolutional Neural Networks (CNNs)

CNNs influenced by the properties of the training dataset



Neural Networks + Automotive applications

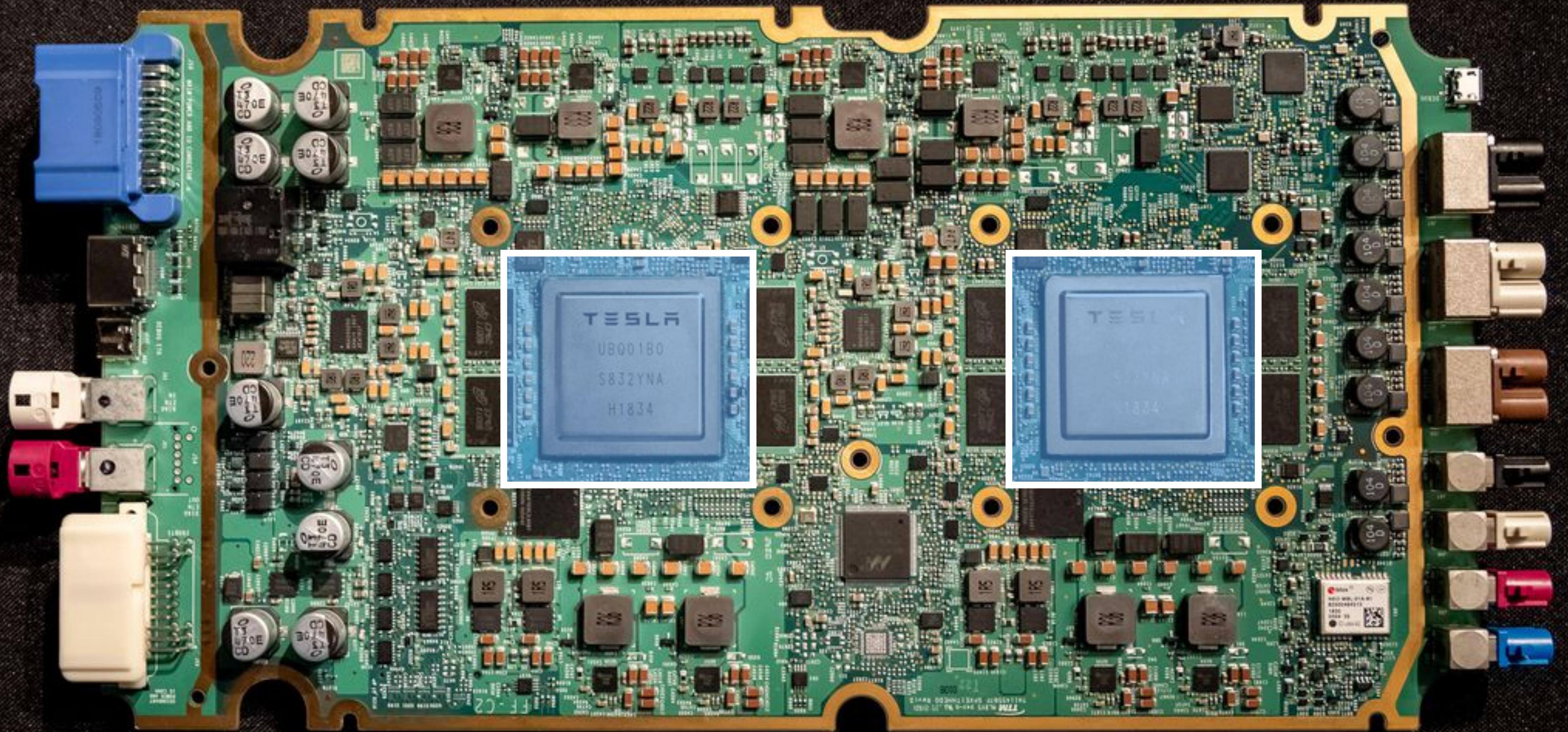
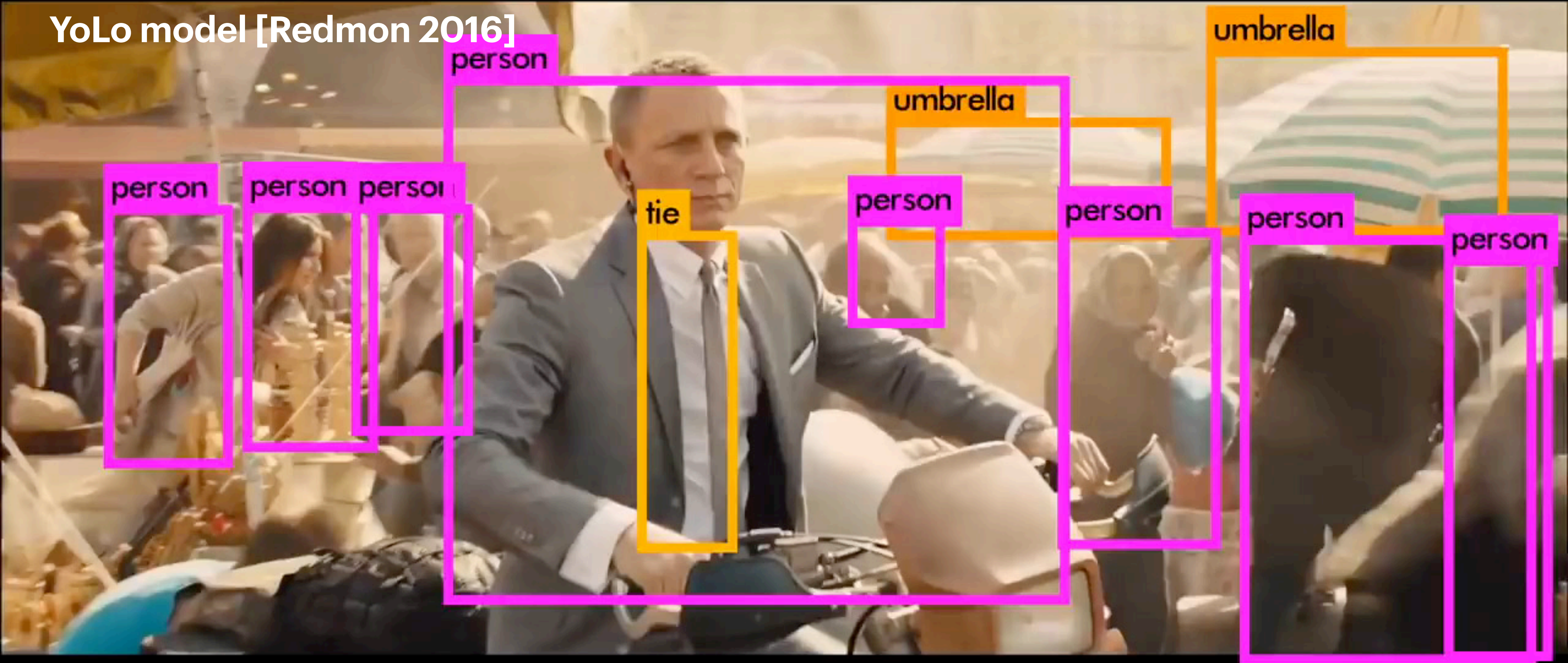


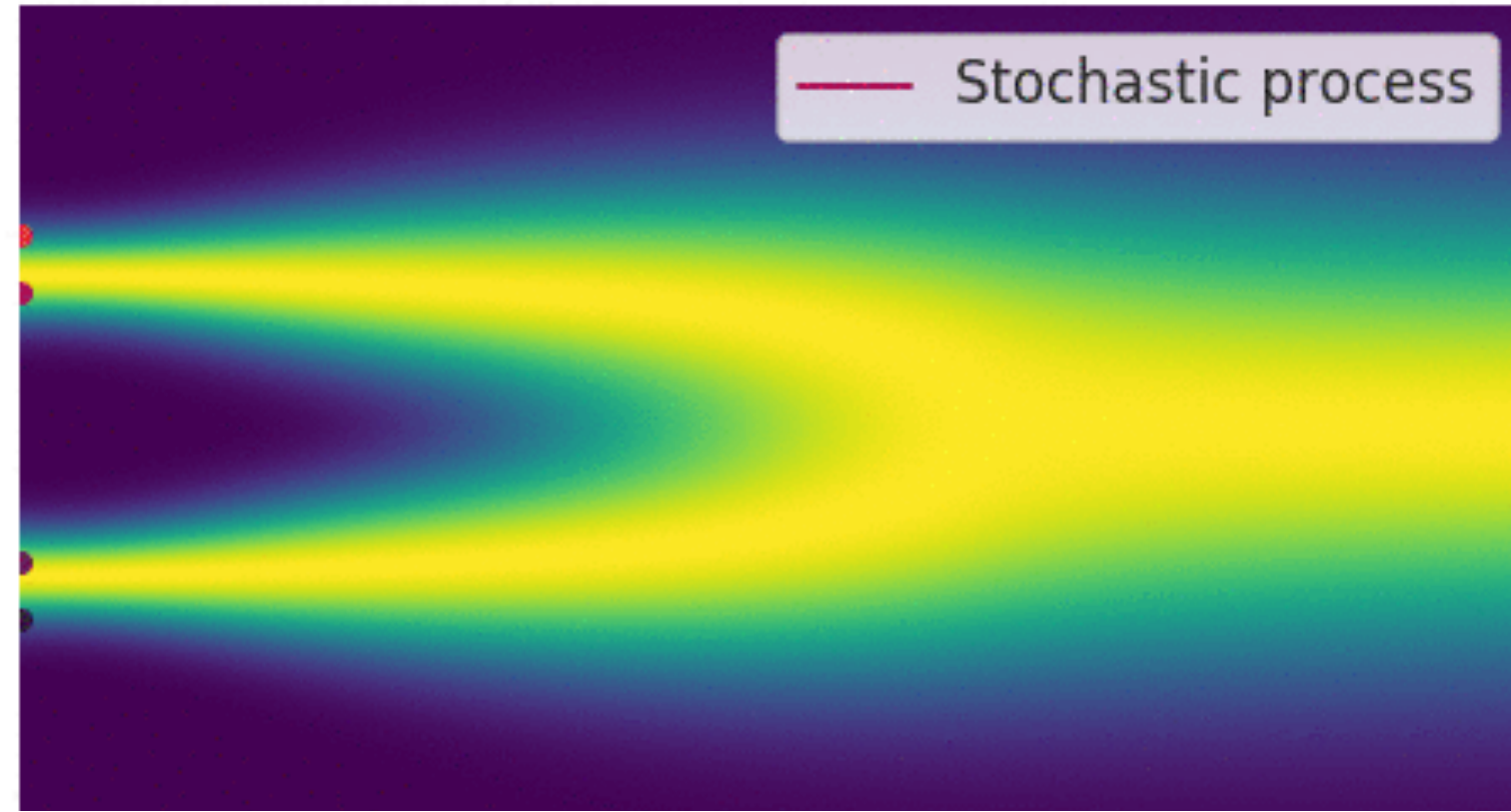
Image and Object classification

YoLo model [Redmon 2016]

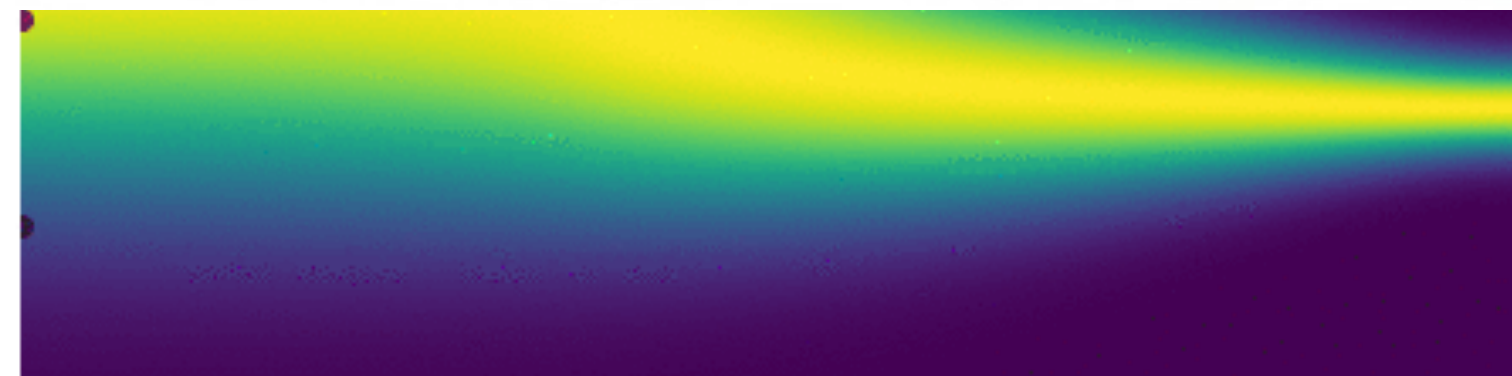


Diffusion models

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



Progressively turning images into noise



Generative image systems: MidJourney

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

1:14 PM a child hypnotized by a perfectly circular spiral on a vintage TV

Job ID: de1133de-2d27-421b-8ed8-f7c9f2f7b8f0

seed 3979930206



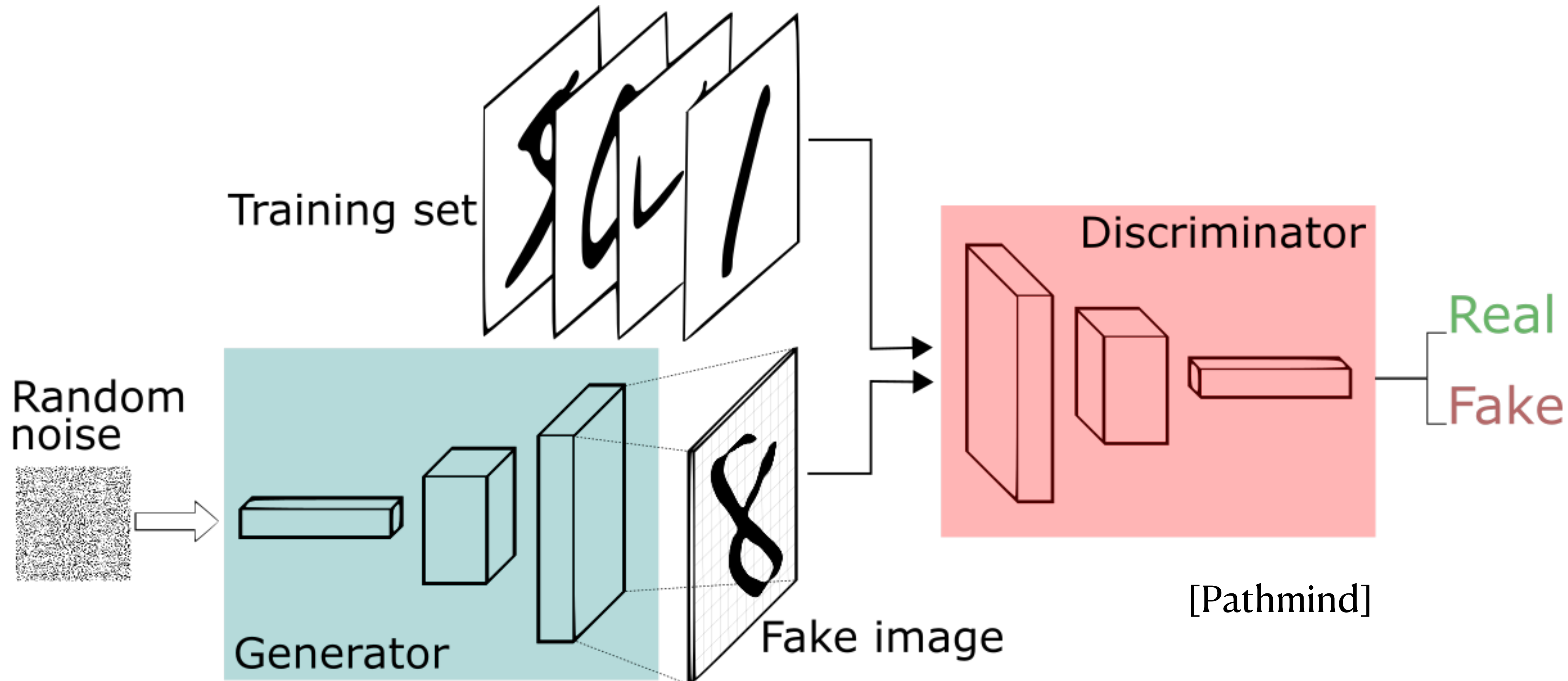
[Jump to message](#)



Message @Midjourney Bot



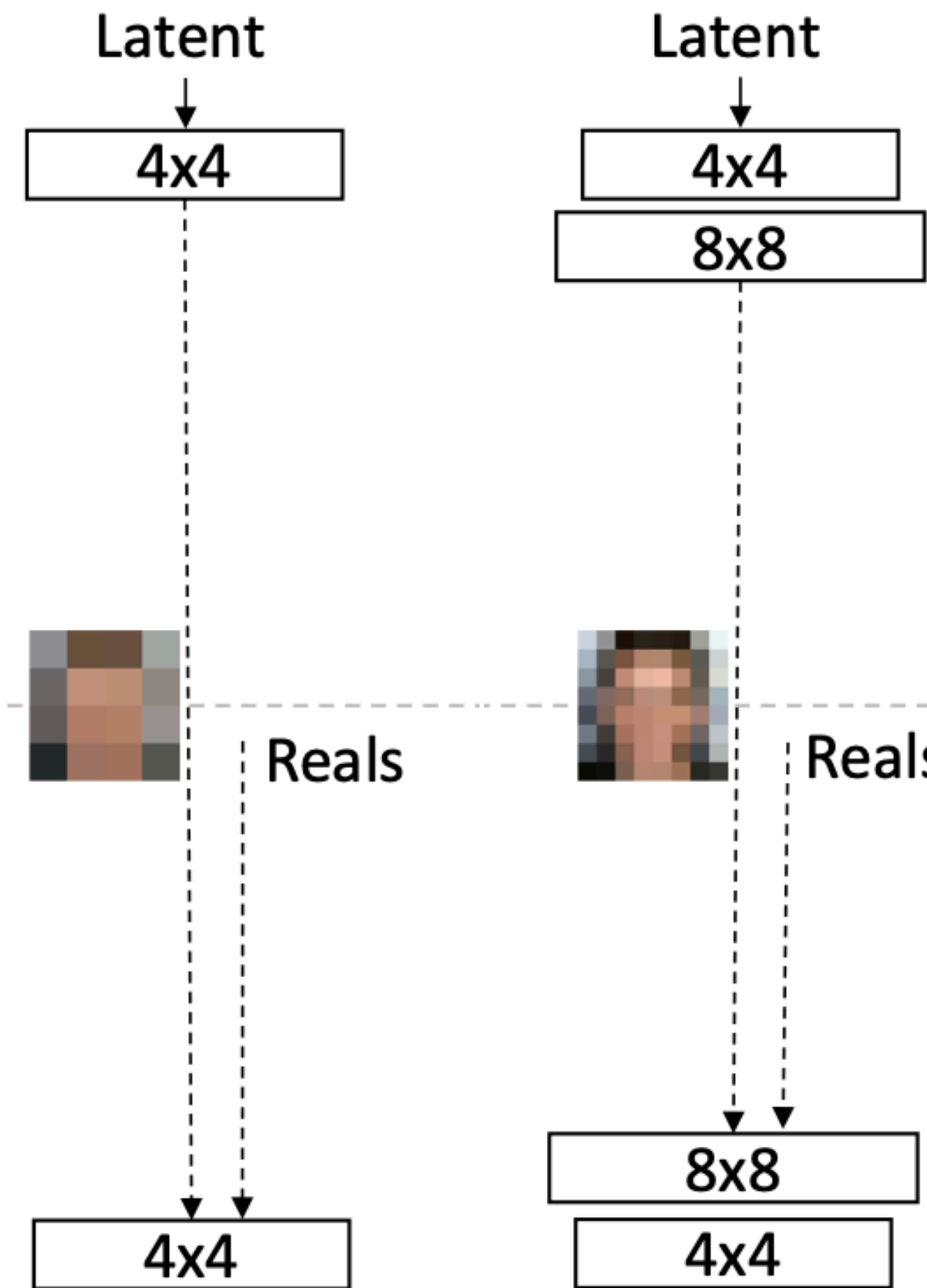
Generative Adversarial Networks



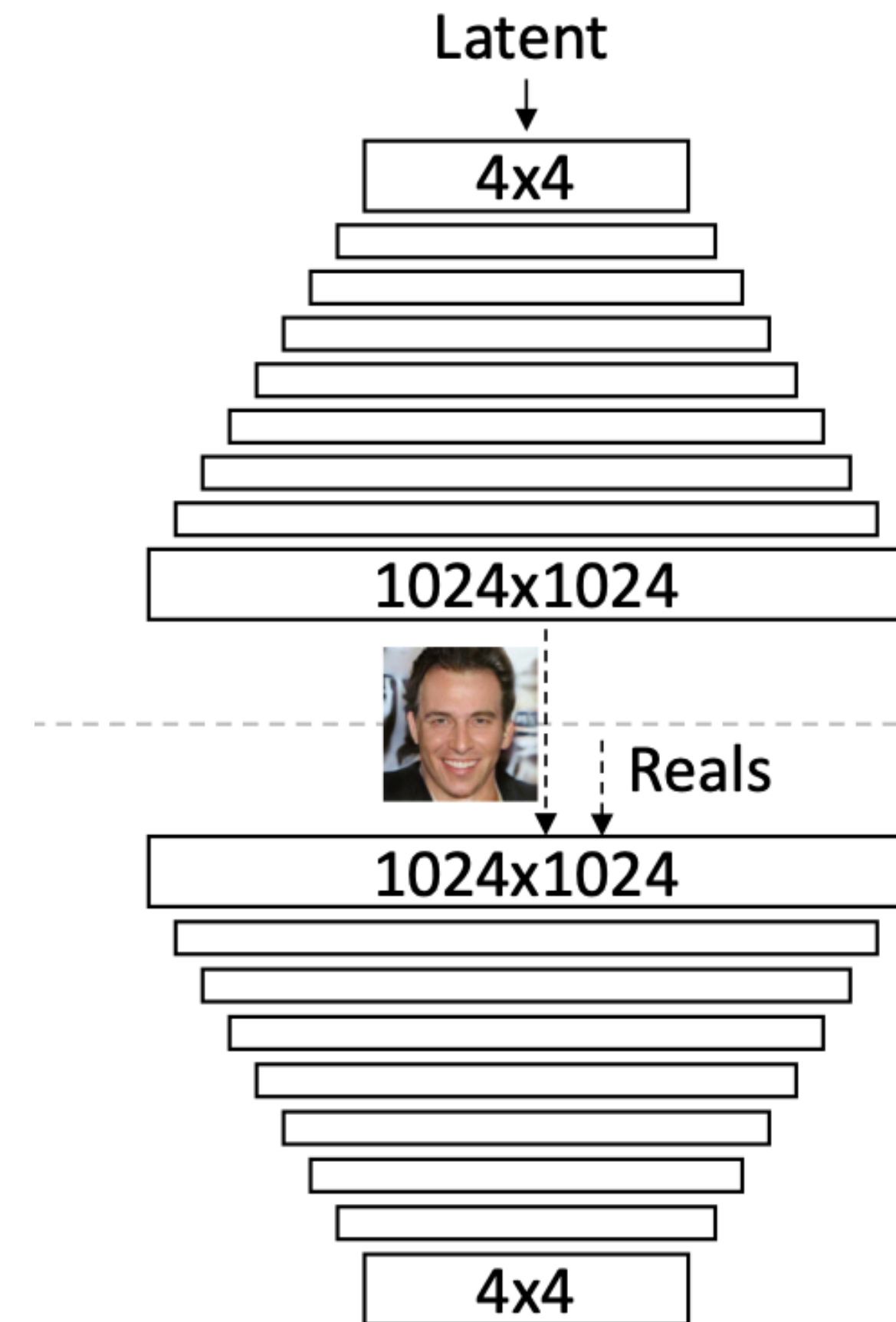
Progressively grown GANs

[Karras et al. ICLR 2018]

Generator



Discriminator



Training progress

Face GANs

Warning:

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

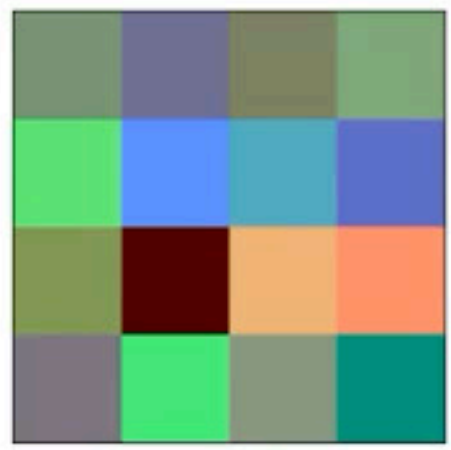
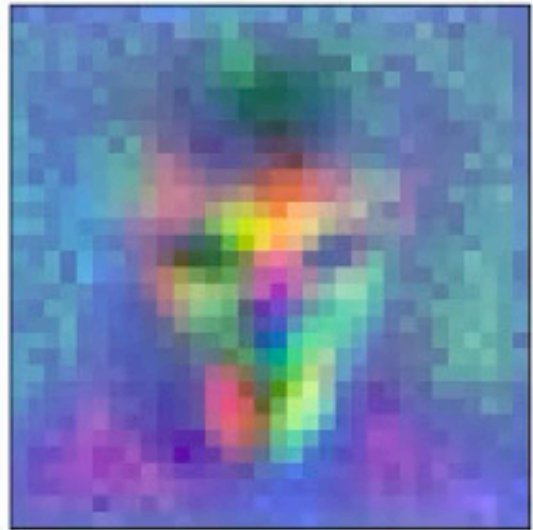

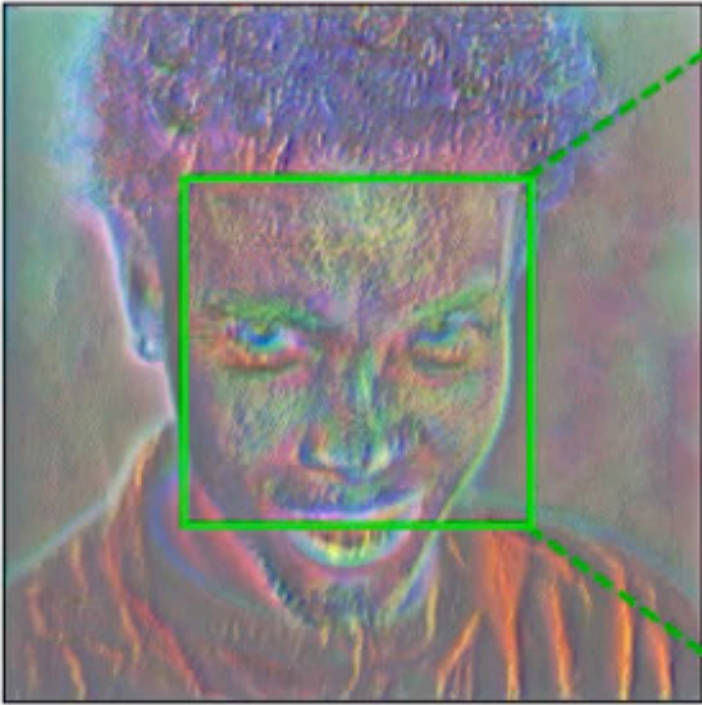



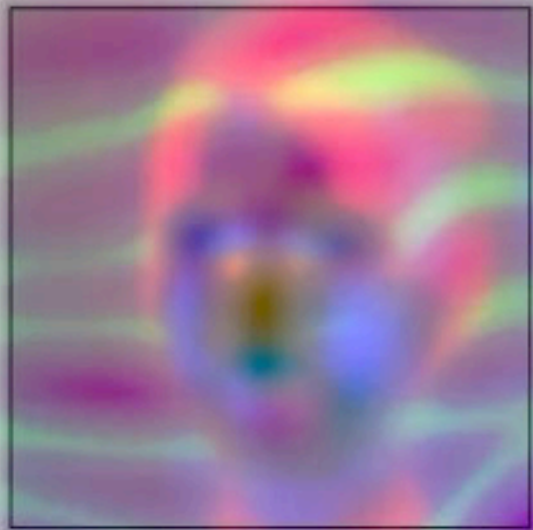
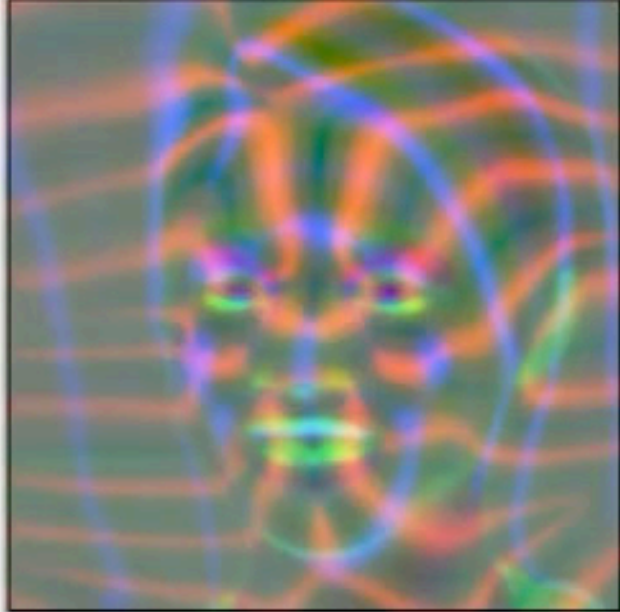
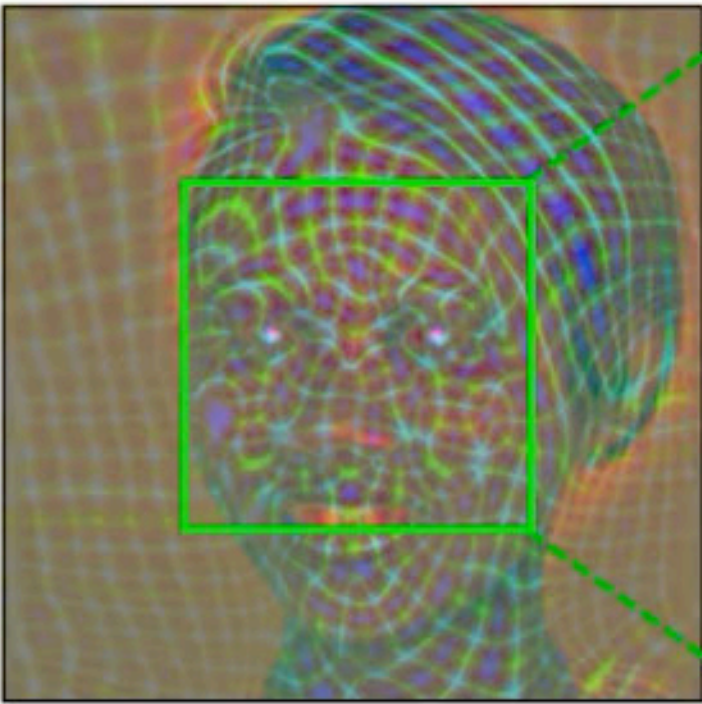
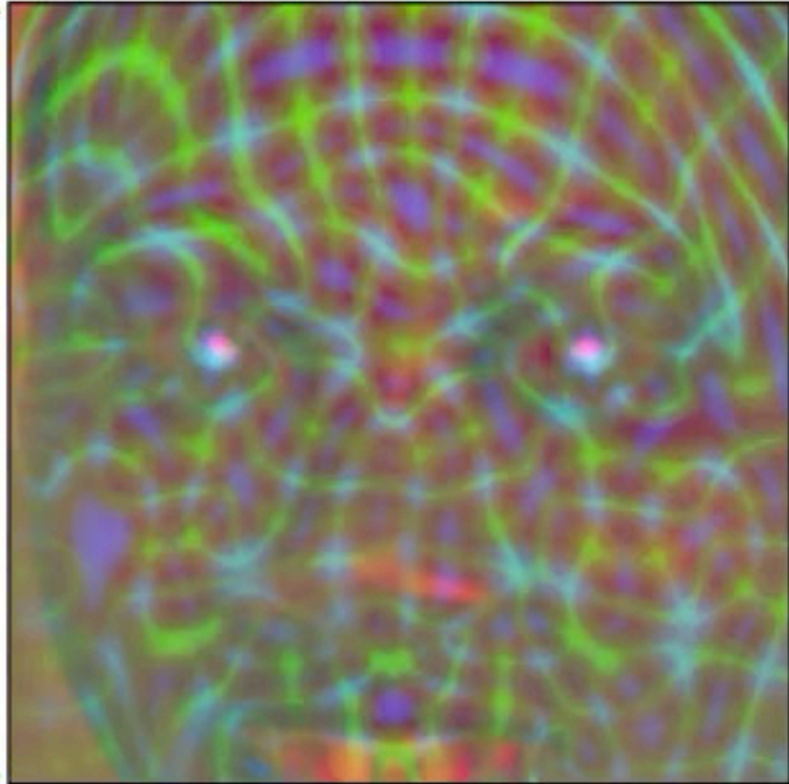

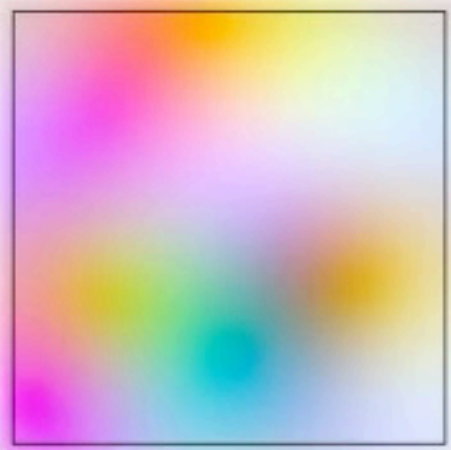
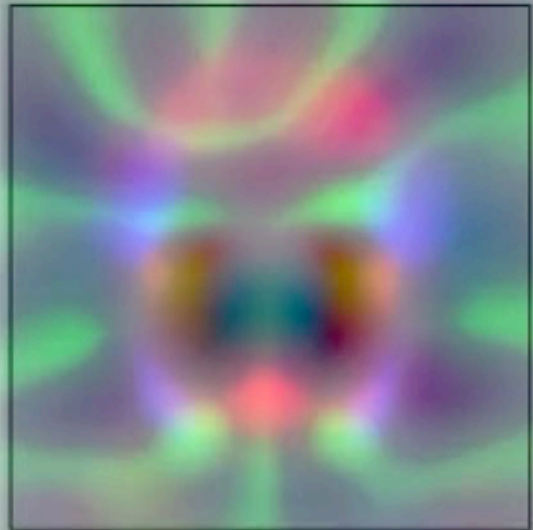
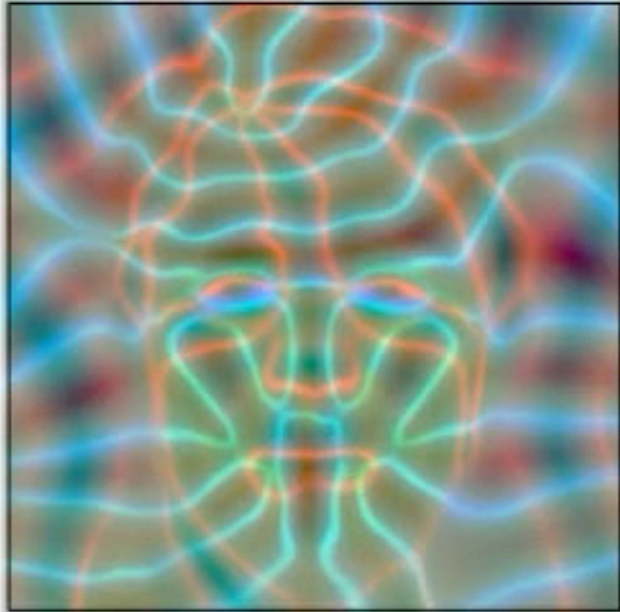
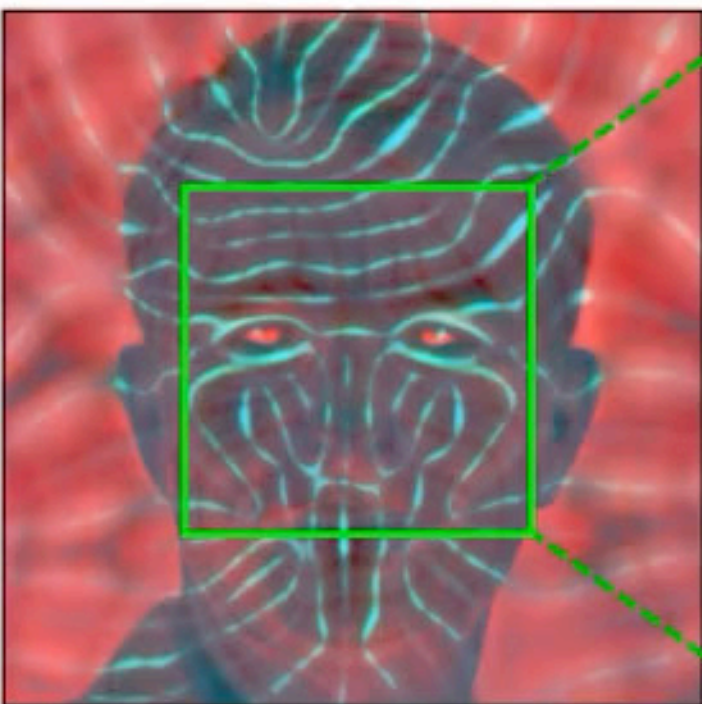


These things exist now, and they are here to stay. Let's see a somewhat disturbing video about a project from **2019**.

In the ML world this is *ancient history*.

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

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

	Input z_0	Internal representations \longrightarrow [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 *latent 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).



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

Showcase 3: Synthesia



Realtime supersampling: NVIDIA DLSS



DLSS Off
Overdrive Mode



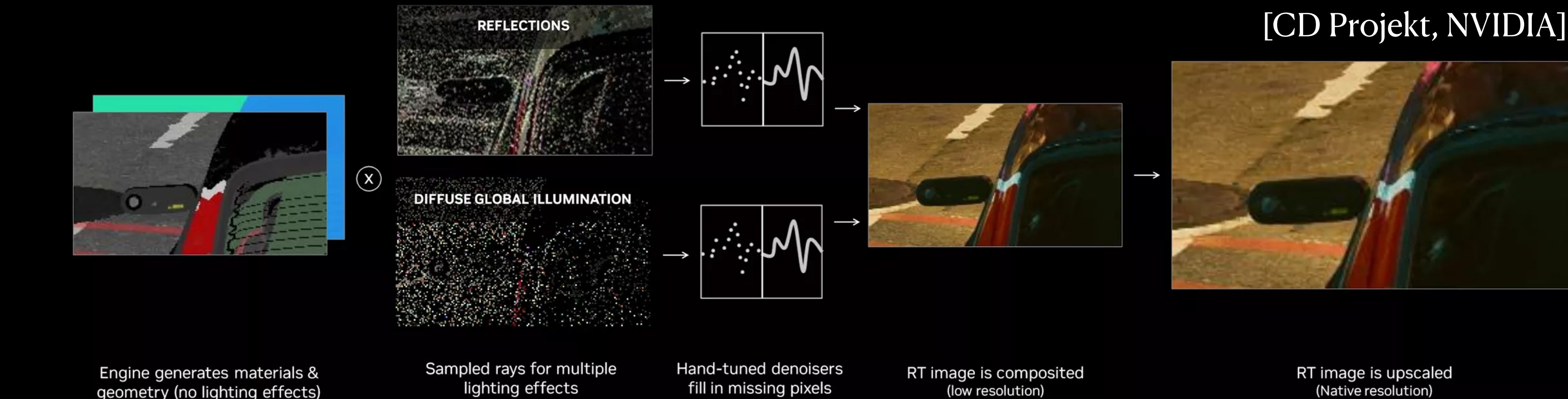
DLSS 3.5
Overdrive Mode With Ray Reconstruction

Denoising & Upscaling for Computer Games



Noisy input frame, only a small part of the pixels were computed.

Additional inputs (these are cheap to compute)



Exponential growth

[Krenn et al. 2023, Nature Machine Intelligence]

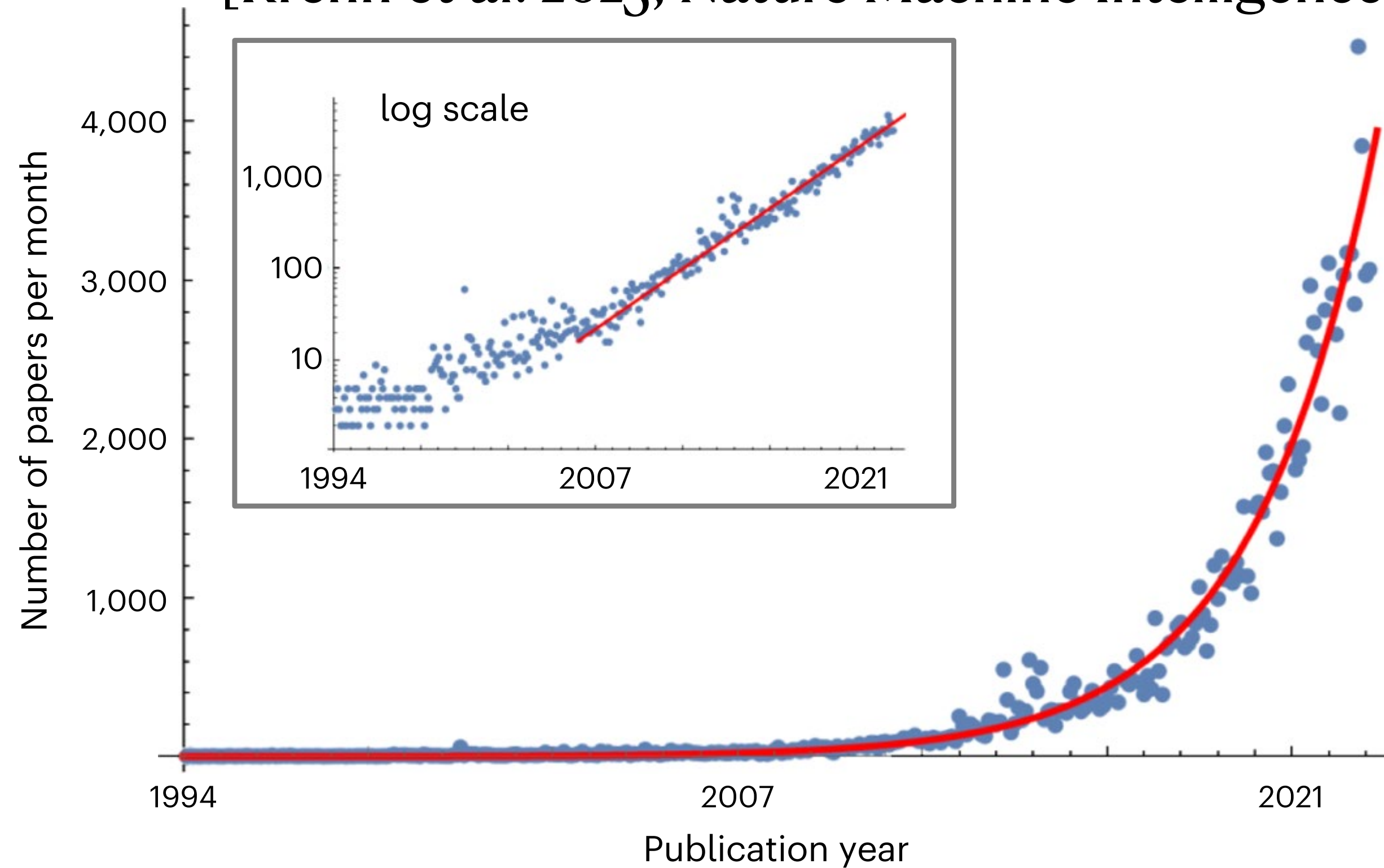


Fig. 1 | The number of papers published per month in the arXiv categories of AI and ML are growing exponentially. The doubling rate of papers per month is roughly 23 months, which might lead to problems for publishing in these fields, at some point. The categories are cs.AI, cs.LG, cs.NE and stat.ML.



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