HVL-MMIV-DLN-AI, 2022

Module 1: Deep learning

Tuesday 03.05.22

PART 1: THE BUILDING BLOCKS OF ARTIFICIAL NEURAL NETWORKS

Alexander S. Lundervold

allu@hvl.no; lundervold.net

Objectives

An understanding of neural networks (and PyTorch)

Digging into the building blocks of neural networks: tensors and tensor operations.

Also: deep learning from a geometric point of view.

Deep learning

searching for good hierarchical geometric representations

01

02

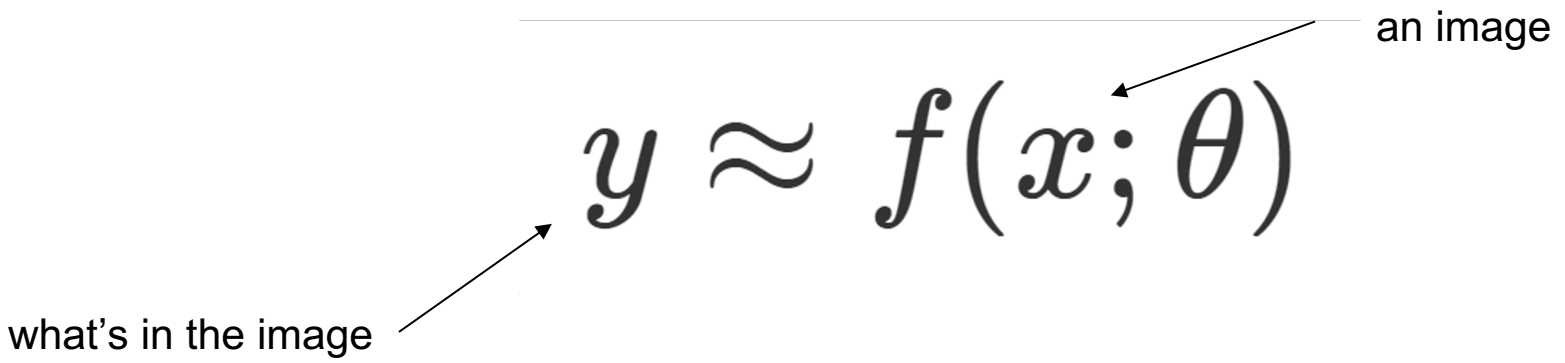
03

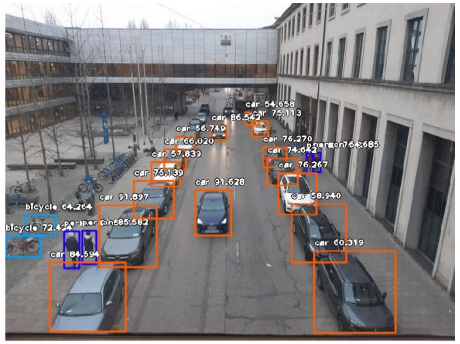
04

05

Function approximation

Function approximation



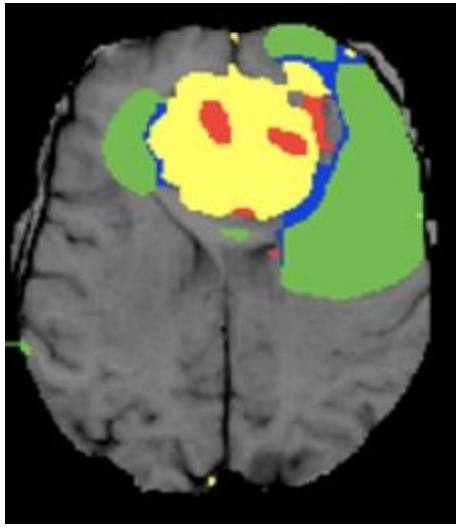


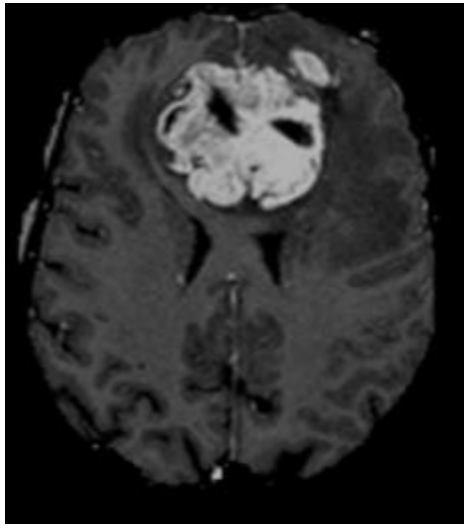
Function approximation

an image

$$y \approx f(x;\theta)$$

what's in the image



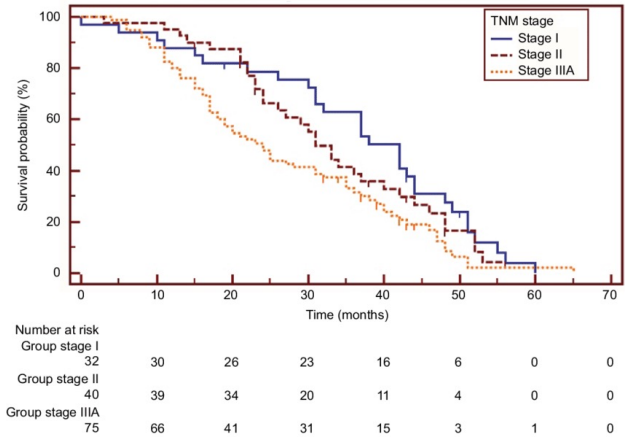


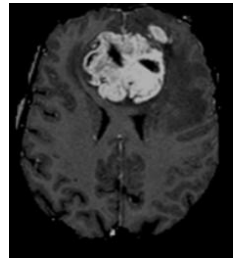
Function approximation

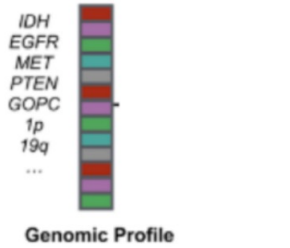
an image

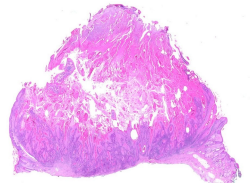
$$y \approx f(x;\theta)$$

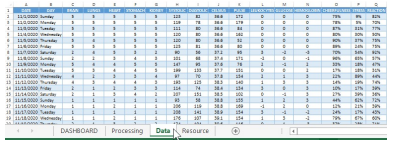
what's in the image











Function approximation

heterogeneous information

$$y \approx f(x;\theta)$$

outcome (e.g., survival)

Training

$$(X, y)$$

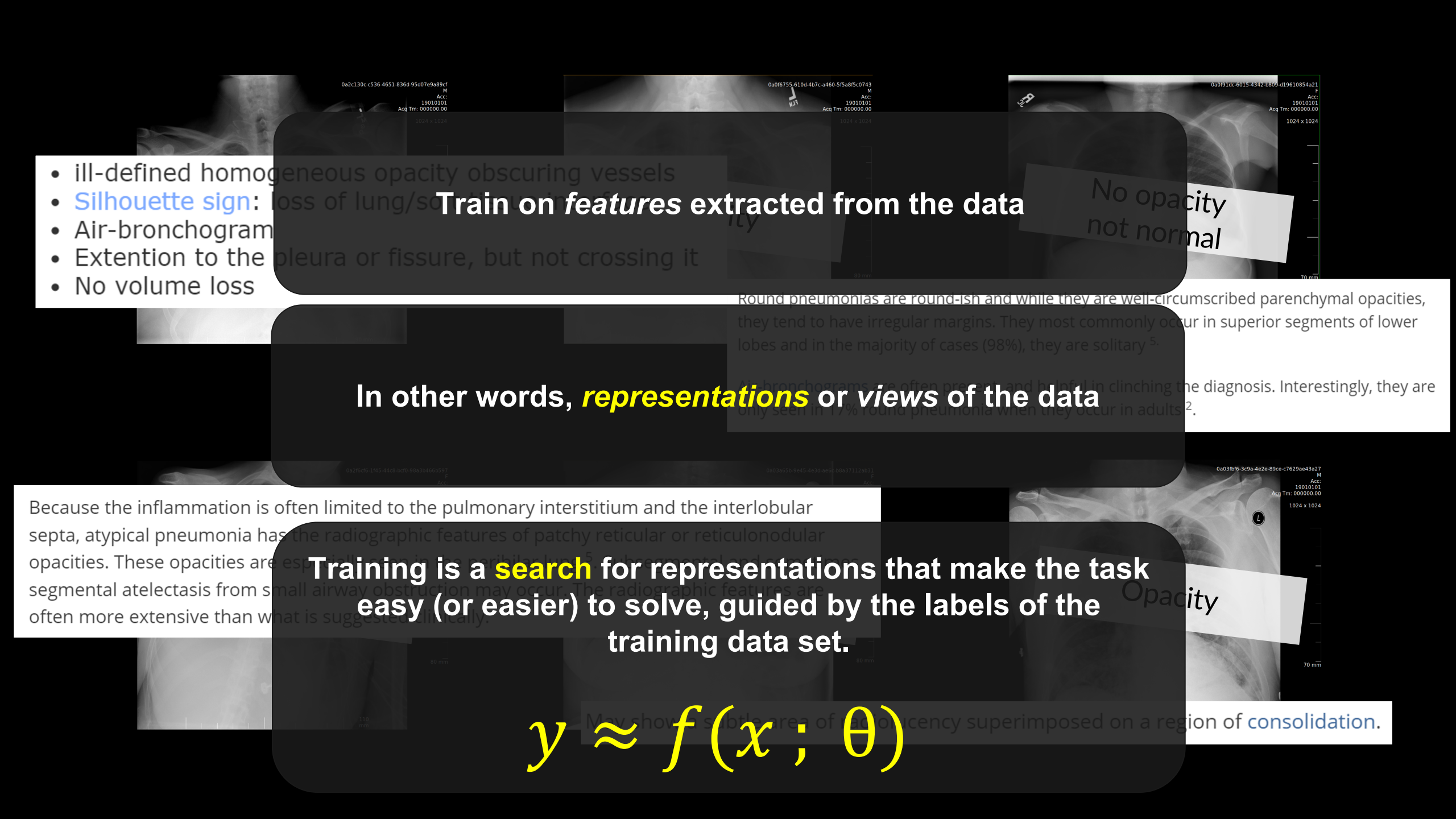
Training

$$(X, y)$$

Input data

Labels

*This is the setup in what's called **supervised learning***


Train on *features* extracted from the data
In other words, *representations* or *views* of the data
Training is a search for representations that make the task easy (or easier) to solve, guided by the labels of the training data set.
$y \approx f(x\,;\,\theta)$

01 Machine learning: **function approximation** based on **training**

features of a disease, process, patient or system

$$y \approx f(x\,;\,\theta)$$

healthy or not healthy / outcome

Models and parameters

$$y \approx f(x; \theta)$$

model **parameters**

***Training* is a search for useful representations through a space defined by a family $\mathcal{F}$ of parametrized models**

01 Machine learning: **function approximation** based on **training**

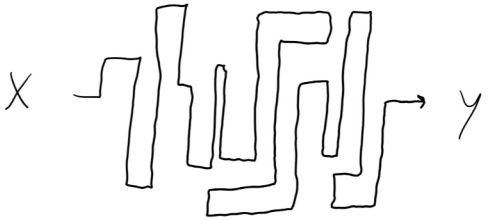
02 We pick the model family F. The parameters Θ are found automatically through training

$$y \approx f(x; \theta)$$

Theory versus practice

Theory

Practice



In theory:

can get arbitrarily good approximations by training simple models
(*universal approximation* and *no free lunch theorems*)

In practice:

which **f** we use and how the training is done makes a huge difference

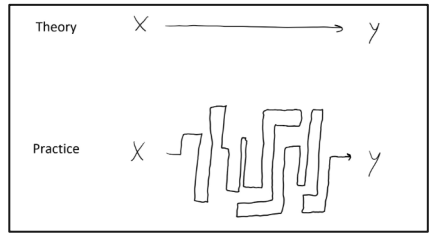
The family $\mathcal{F}$ of models considered, i.e., the *hypothesis space,* should be (i) efficiently searchable, (ii) leading to expressive models that can (ii) generalize beyond the training data distribution (i.e., work well on new data)

01 Machine learning: **function approximation** based on **training**

02 We pick the model family F. The parameters Θ are found automatically through training

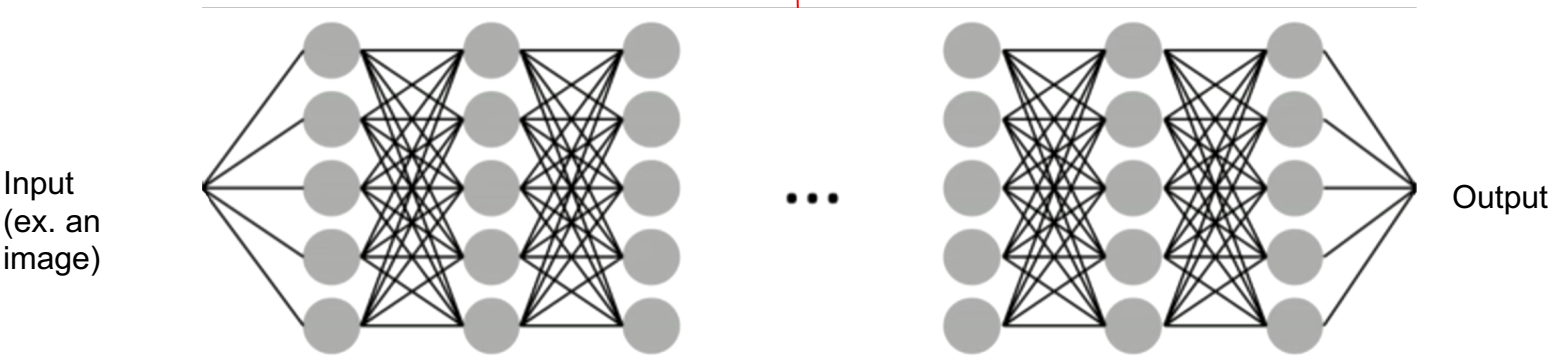
03 In practice: some models are better suited to some tasks than others.

$$y \approx f(x;\theta)$$



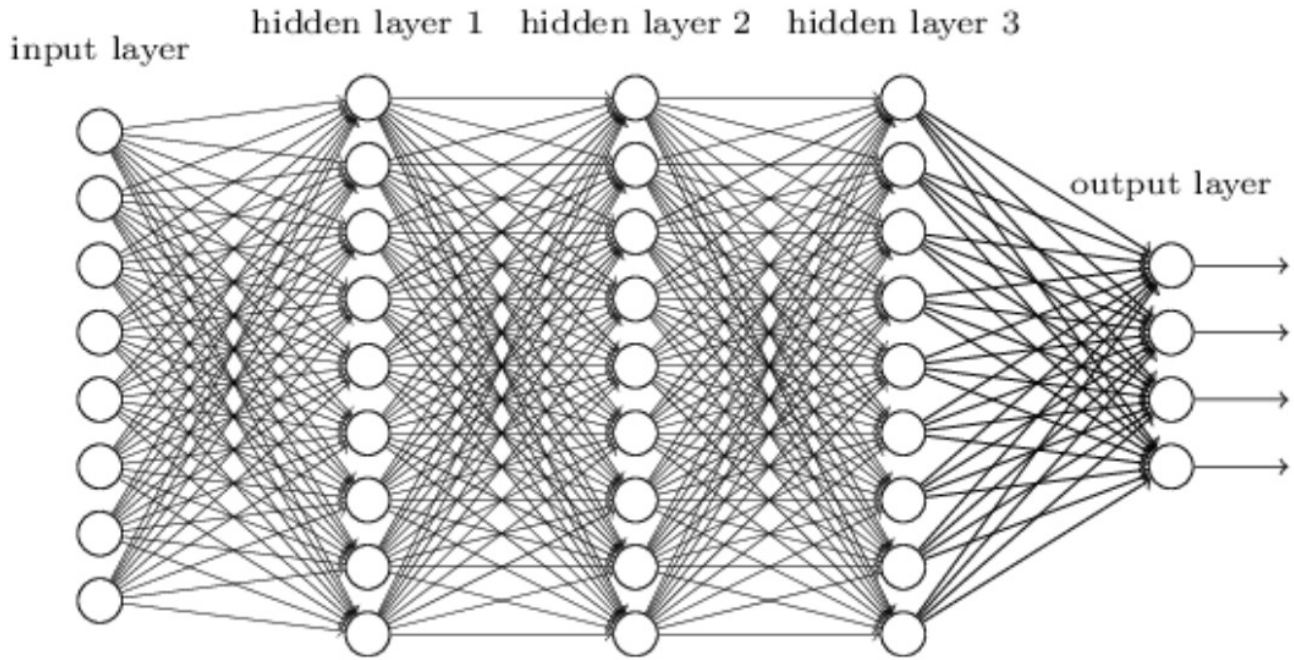
Deep learning

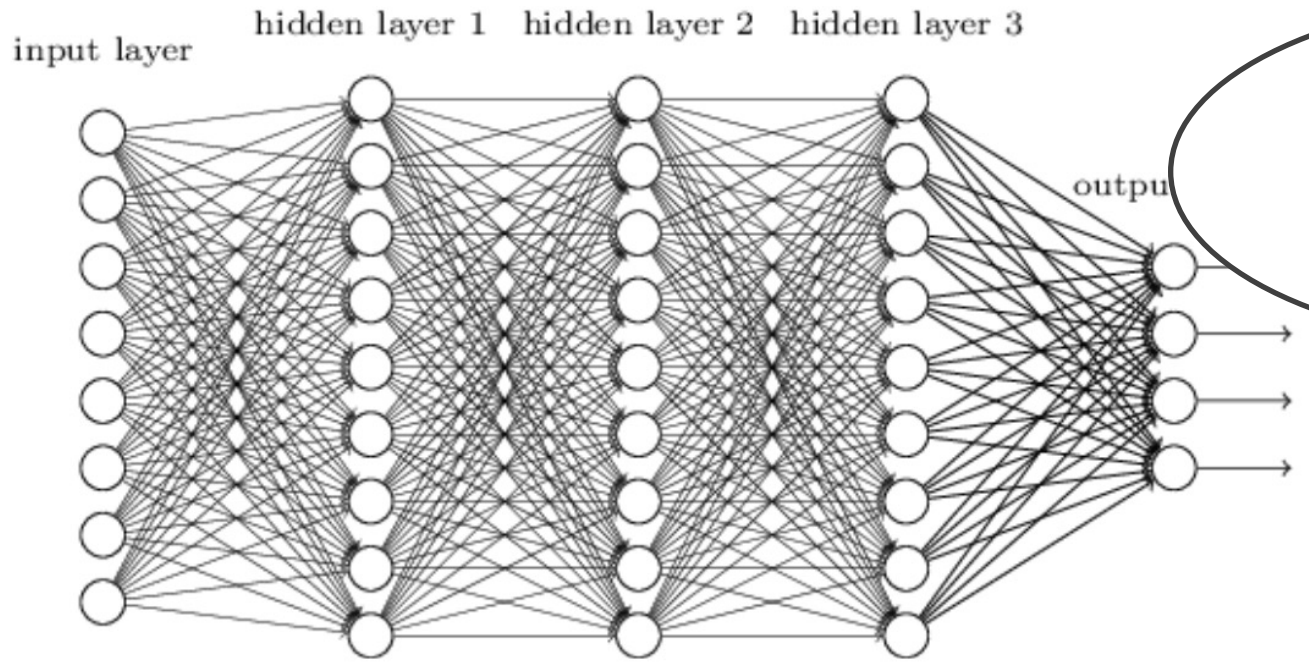
$$y \approx f(x; \theta)$$



Artificial neural networks

Computational graphs of simple units (“neurons”) connected in various specific ways, parametrized by parameters associated to each layer.


input layer
hidden layer 1
hidden layer 2
hidden layer 3
output layer



Can be viewed from a geometric point of view.
More about that later

This is a fancy way to draw what's really just a composition of simple functions:

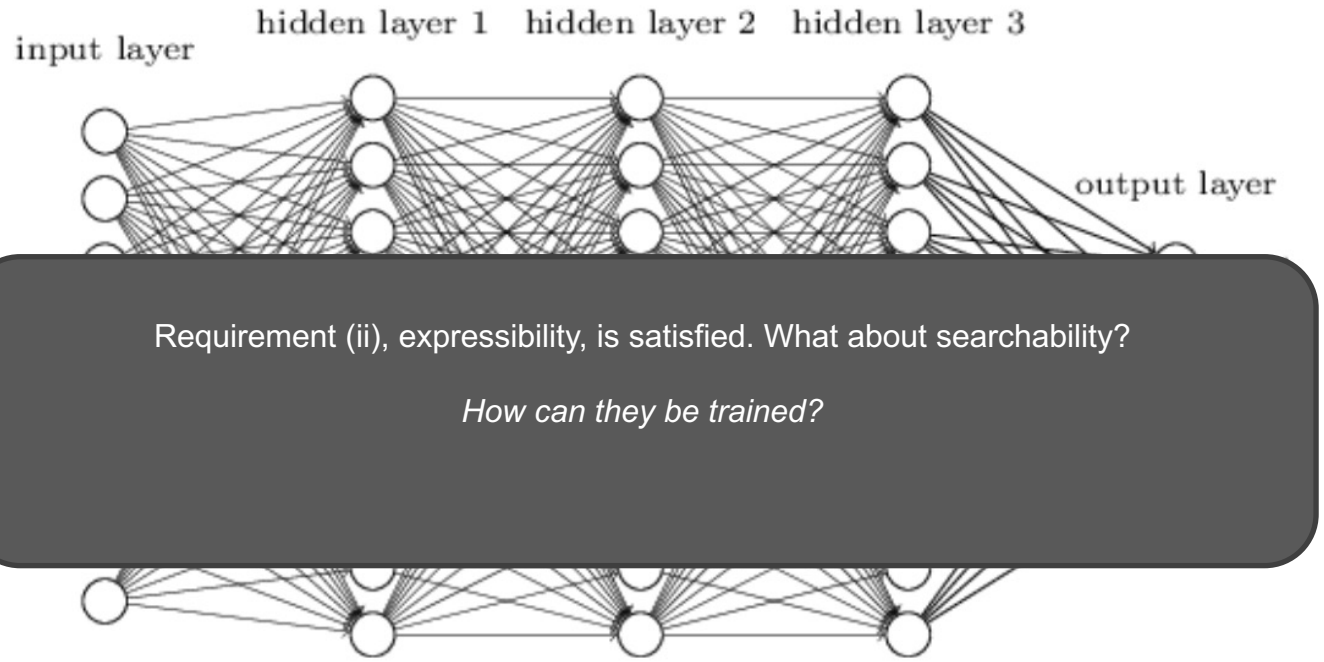
$$y = \sigma_{k+1}(W_{k+1}^T \sigma_k(W_k^T \left(\cdots \left(\sigma_1(W_1^T x) \right) \cdots \right) = f(x; \theta)$$

Input x is sent through several consecutive **layers** (functions). An output is produced.

Think of each layer as a (geometric) **transformation** of its input.

Every layer has a bunch of **parameters**, stored in the matrices W

Each family member of F is complex transformation broken down into a composition of simpler ones, parametrized by the layer weights.



Requirement (ii), expressibility, is satisfied. What about searchability?

How can they be trained?

This is a fancy way to draw what's really just a composition of simple functions:

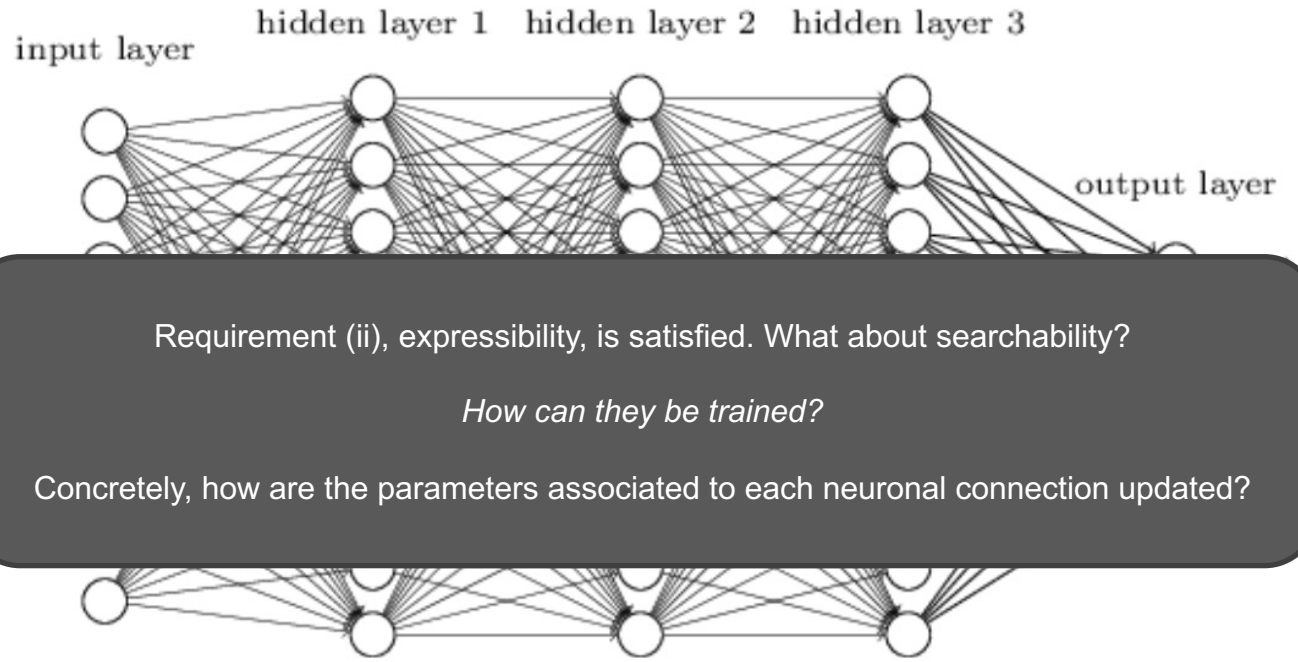
$$y = \sigma_{k+1}(W_{k+1}^T \sigma_k(W_k^T \left(\cdots \left(\sigma_1(W_1^T x) \right) \cdots \right) = f(x; \theta)$$

Input x is sent through several consecutive **layers** (functions). An output is produced.

Think of each layer as a (geometric) **transformation** of its input.

Every layer has a bunch of **parameters**, stored in the matrices W

Each family member of F is complex transformation broken down into a composition of simpler ones, parametrized by the layer weights.



Requirement (ii), expressibility, is satisfied. What about searchability?

How can they be trained?

Concretely, how are the parameters associated to each neuronal connection updated?

This is a fancy way to draw what's really just a composition of simple functions:

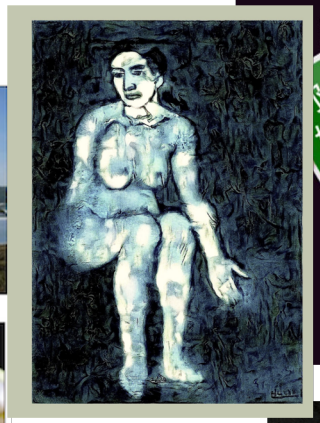
$$y = \sigma_{k+1}(W_{k+1}^T \sigma_k(W_k^T \left(\cdots \left(\sigma_1(W_1^T x) \right) \cdots \right) = f(x; \theta)$$

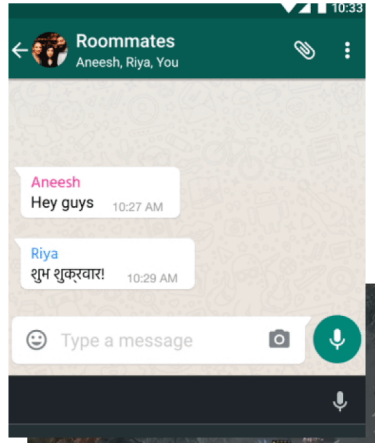
Input x is sent through several consecutive **layers** (functions). An output is produced.

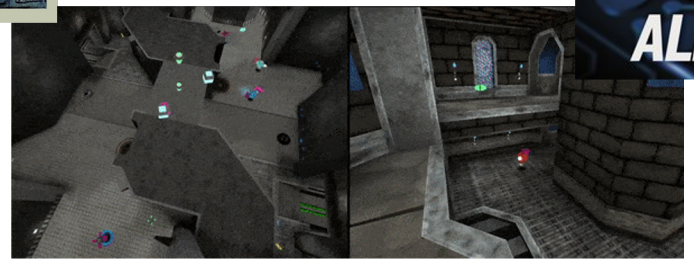
Think of each layer as a (geometric) **transformation** of its input.

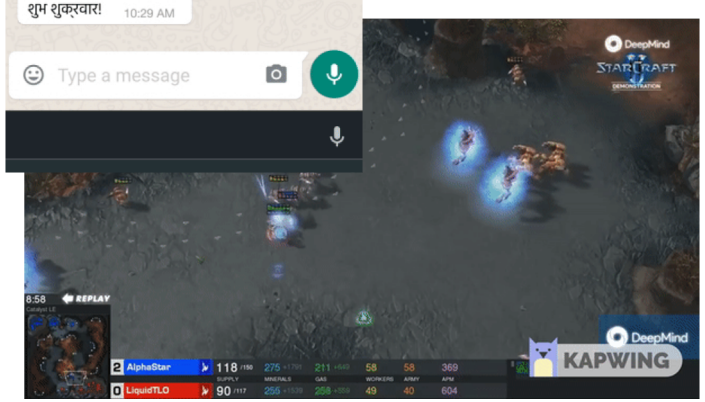
Every layer has a bunch of **parameters**, stored in the matrices W

Each family member of F is complex transformation broken down into a composition of simpler ones, parametrized by the layer weights.









HARE RESEARCH ARTICLE

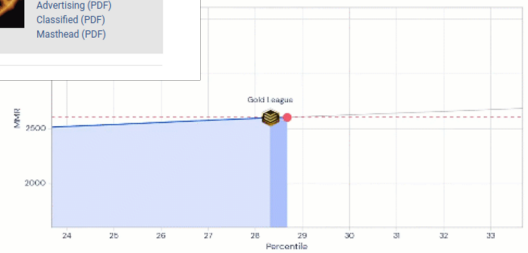
Human-level performance in 3D multiplayer games with population-based reinforcement learning

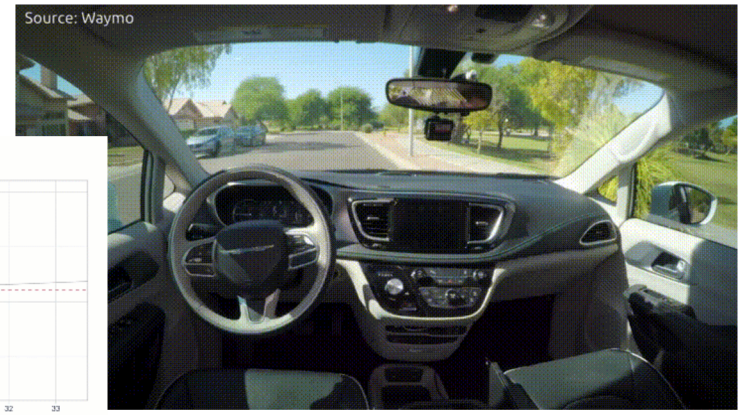
Max Jaderberg*,†, Wojciech M. Czarnecki*,†, Iain Dunning†, Luke Marris, Guy Lever, Antonio Garcia Castañeda, Charles Beat…

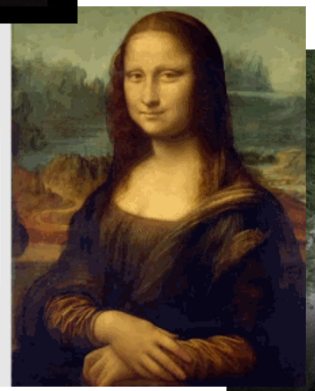
+ See all authors and affiliations

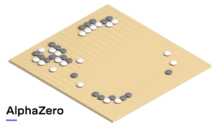
Science 31 May 2019:
Vol. 364, Issue 6443, pp. 859-865
DOI: 10.1126/science.aau6249

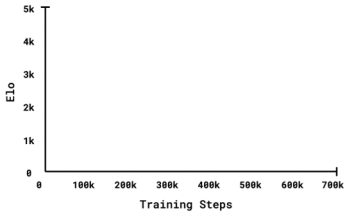
Science
Vol 364, Issue 6443
31 May 2019
Table of Contents
Print Table of Contents
Advertising (PDF)
Classified (PDF)
Masthead (PDF)











Skin cancer can't hide from deep-learning diagnostics

May 21, 2019 | Dave Pearson | Diagnostics

NEWS CAREERS JOURNALS

News Home All News

FDA Approves AI Tool That Can Detect Wrist Fractures

The FDA has approved a new tool that can improve the accuracy of wrist fracture diagnosis.

By Jessica Miley
May 28th, 2018

Deep learning improves detection of polyps during colonoscopy

November 07, 2018 | Matt O'Connor | Artificial Intelligence

Nevit Dilmen/Wikimedia Commons

ME > NEWS > ALL NEWS > AI CRACKS THE CODE OF PROTEIN COMPLEXES—PROVID

WS | BIOLOGY

AI cracks the code of prote
-providing a road map for

oftware maps thousands of the partnered proteins

NOV 2021 · 2:00 PM · BY ROBERT F. SERVICE

Results

tein complex prediction with AlphaFold-Multimer

chard Evans, Michael O'Neill, Alexander Pritzel, Natasha Antropova, Andre
m Green, Augustin Žídek, Russ Bates, Sam Blackwell, Jason Yim, Olaf R
bastian Bodenstein, Michal Zielinski, Alex Bridgland, Anna Potapenko, Andrew
athryn Tunyasuvunakool, Rishub Jain, Ellen Clancy, Pushmeet Kohli, John
emis Hassabis

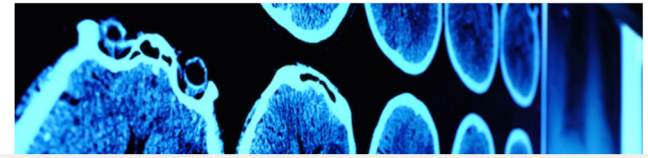
https://doi.org/10.1101/2021.10.04.463034

rticle is a preprint and has not been certified by peer review [what does this mean?].

Science LOG IN

Deep Learning Tool Identifies Disease Markers of Alzheimer's

A deep learning tool was able to rapidly detect markers of Alzheimer's disease in human brain tissue.



NOVEMBER 15, 2017

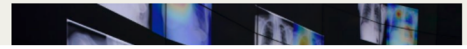
Stanford algorithm can diagnose pneumonia better than radiologists

Stanford researchers have developed a deep learning algorithm that evaluates chest X-rays for signs of disease. In just over a month of development, their algorithm outperformed expert radiologists at diagnosing pneumonia.

BY TAYLOR KUBOTA

Stanford researchers have developed an algorithm that offers diagnoses based off chest X-ray images. It can diagnose up to 14 types of medical conditions and is able to diagnose pneumonia better than expert radiologists working alone. A paper about the algorithm, called CheXNet, was published Nov. 14 on the open-access, scientific preprint website arXiv.

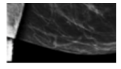
"Interpreting X-ray images to diagnose pathologies like



Deep learning predicts OCT measures of diabetic macular thickening

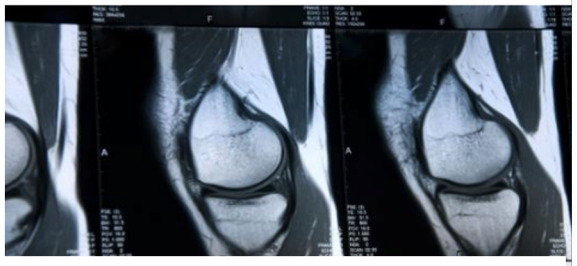
Exudates in the macula showing clinically significant macular edema (CSME) can be deceiving for the diagnosis of abnormal macular thickening. Left: Color fundus photo shows exudates indicating CSME, with associated CST <250 um. Right: CFP shows no CSME, but has associated CST >250 um. (Images courtesy of Jeffrey Wilks, MD, PhD)

By Steve Lenier
May 21, 2019



Deep learning detects ACL tears on MRI sc

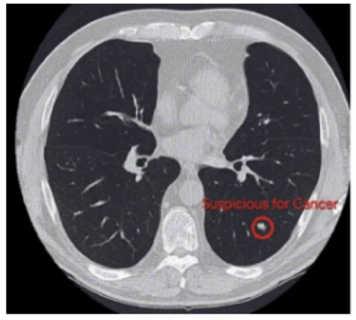
May 17, 2019 | Matt O'Connor | Artificial Intelligence

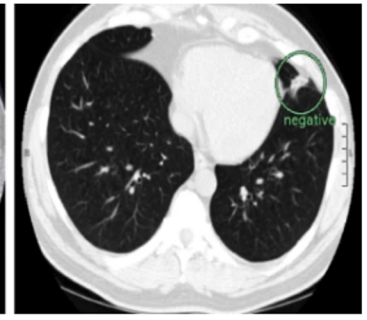


AI

Google's lung cancer detection AI outperforms 6 human radiologists

KHARI JOHNSON @KHARIJOHNSON MAY 20, 2019 8:00 AM





Novel Molecules Designed by Artificial Intelligence May Accelerate Drug Discovery

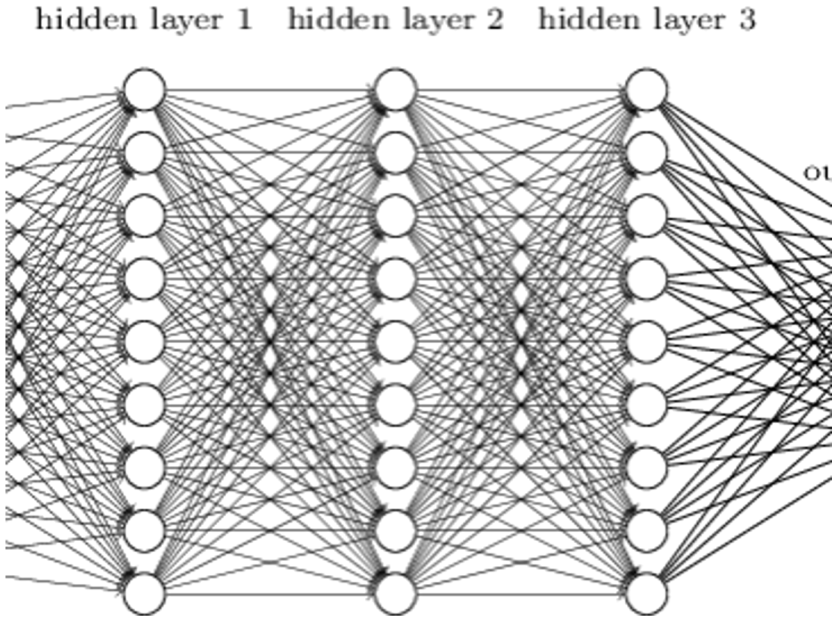
TOPICS: Artificial Intelligence Biotechnology InSilico Medicine

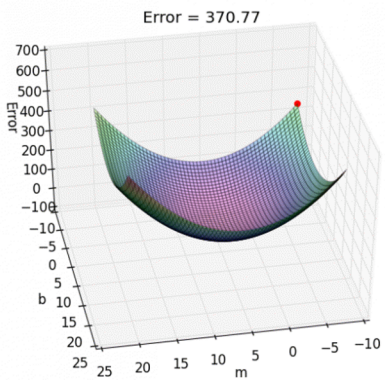
Three basic ingredients:

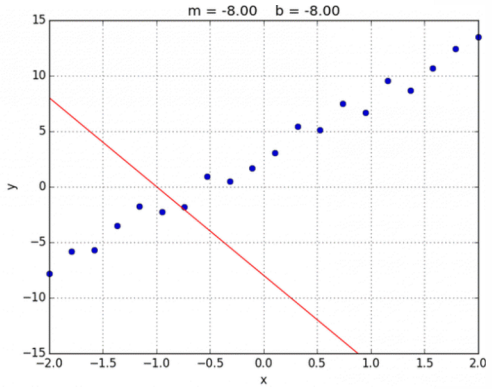
1. A **loss function**: Used to measure the network's performance on the training data. A **feedback signal** for the training process.

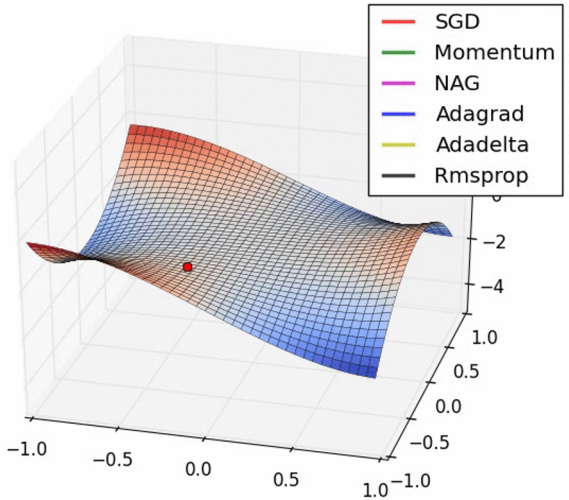
2. An **optimizer**: Used to update the *parameters* of the network to increase performance as measured by the loss function. **Gradient descent** and **backpropagation**.

3. One or more **metrics**: A way to score the model. For example *accuracy* or *mean squared error.*









Define a neural network model with an architecture and a set of trainable parameters (weights and biases)

1. Initialize the parameters

2.Collect a batch of inputs `(X,y)` and use the model with its current parameters to make predictions `y_pred`

3. Calculate how good the model is using the loss function to measure the discrepancy between the known labels `y_true` and the predictions `y_pred` for this batch

4. Calculate the gradients for each parameter using backpropagation. This measures how changing each parameter would change the loss. In other words, each parameter's contribution to the loss.

5. Move each weight a little bit in the opposite direction of its gradient using gradient descent. I.e., some variant of

```
parameter = parameter - learning_rate * gradient
```

6. Go back to step 2 and repeat the process

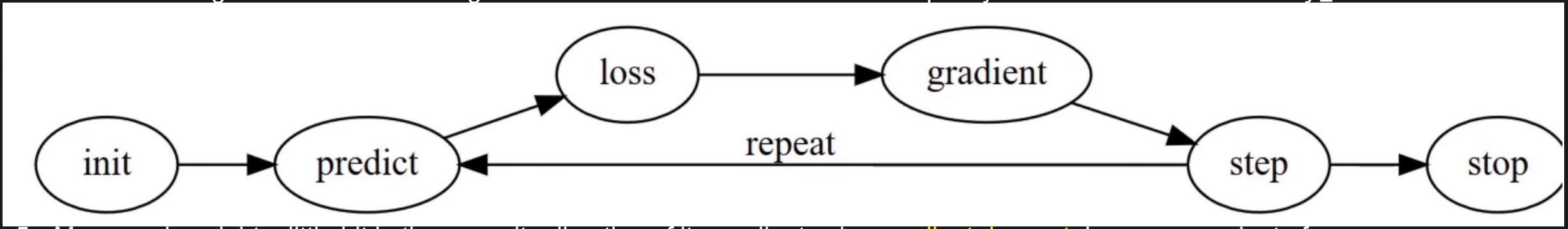
7. Do this over and over for several epochs (one epoch is one pass through all the training data), until you meet some chosen *stopping criteria*

Define a neural network model with an architecture and a set of trainable parameters (weights and biases)

1. Initialize the parameters

2.Collect a batch of inputs `(X,y)` and use the model with its current parameters to make predictions `y_pred`

3. Calculate how good the model is using the loss function to measure the discrepancy between the known labels `y_true` and the



5. Move each weight a little bit in the opposite direction of its gradient using gradient descent. I.e., some variant of
   `parameter = parameter - learning_rate * gradient`

6. Go back to step 2 and repeat the process

7. Do this over and over for several epochs (one epoch is one pass through all the training data), until you meet some chosen *stopping criteria*

Define a neural network model with an architecture and a set of trainable parameters (weights and biases)

1. Initialize the parameters

2.Collect a batch of inputs `(X,y)` and use the model with its current parameters to make predictions `y_pred`

3. Calculate how good the model is using the loss function to measure the discrepancy between the known labels `y_true` and the predictions `y_pred` for this batch

What is gradient descent?

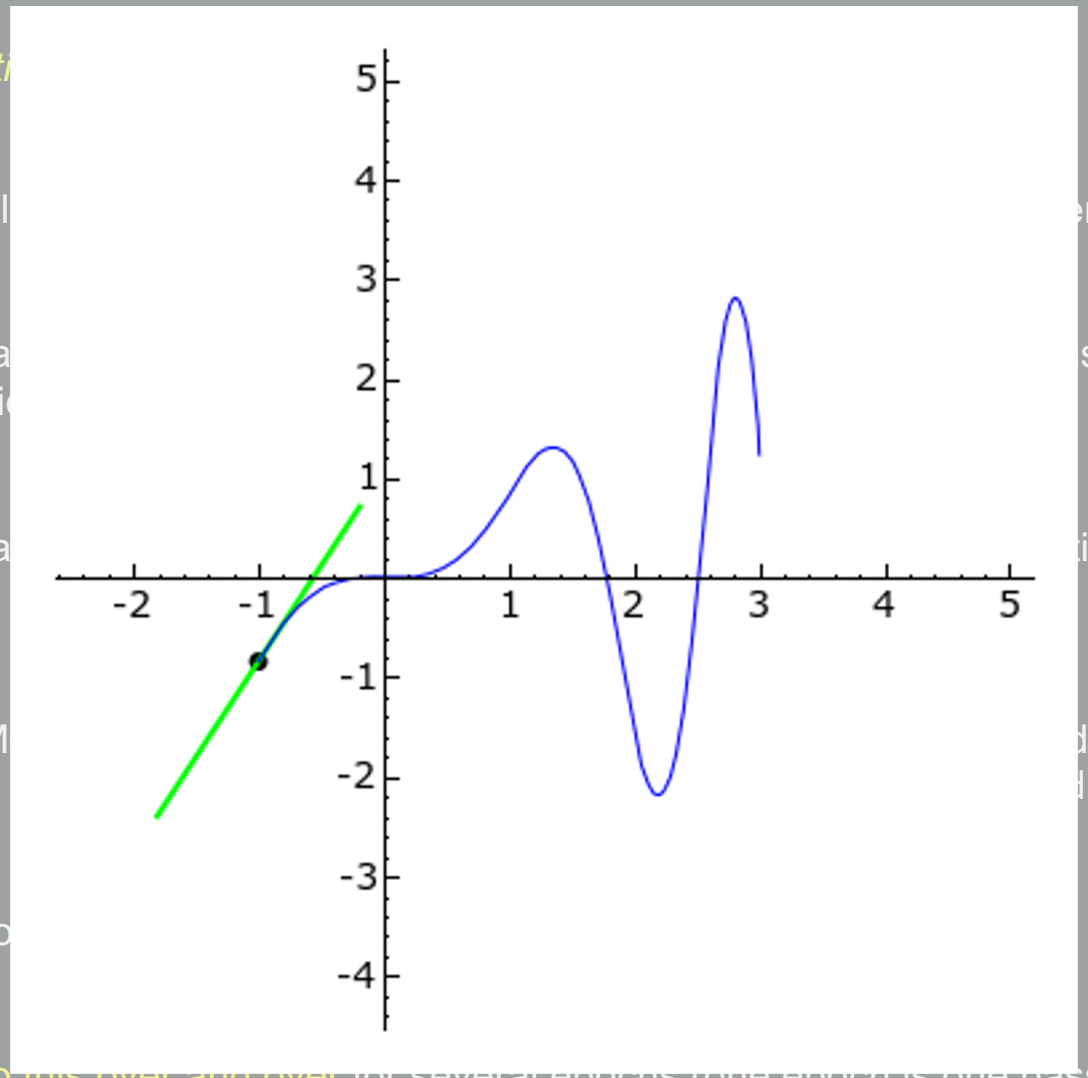
4. Calculate the gradients for each parameter using backpropagation. This measures how changing each parameter would change the loss. In other words, each parameter's contribution to the loss.

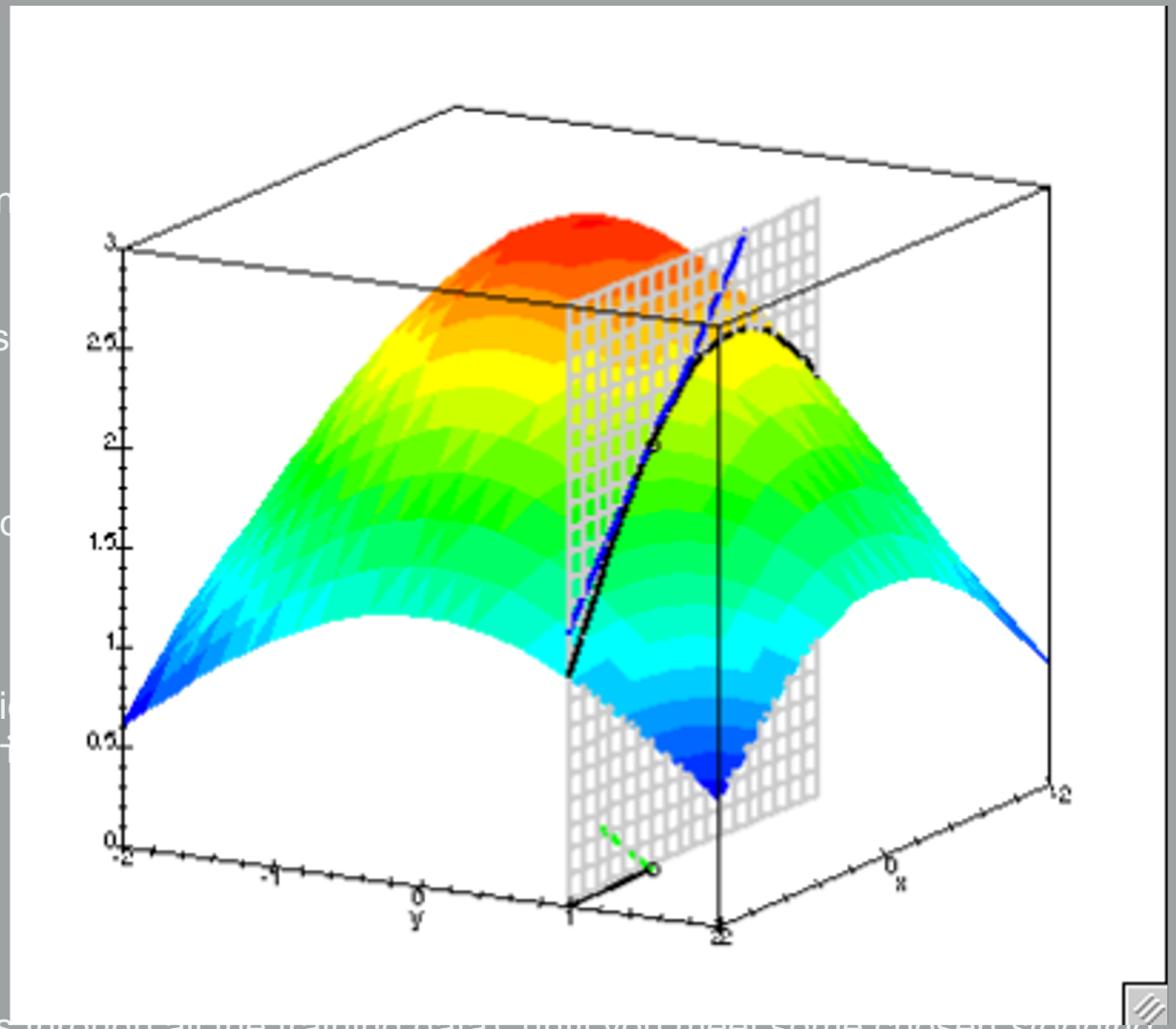
5. Move each weight a little bit in the opposite direction of its gradient using gradient descent. I.e., some variant of
   `parameter = parameter - learning_rate * gradient`

6. Go back to step 2 and repeat the process

7. Do this over and over for several epochs (one epoch is one pass through all the training data), until you meet some chosen *stopping criteria*

Define a neural network model with an architecture and a set of trainable parameters (weights and biases)





1.Initi

2.Col

3. Ca
predi

4. Ca
loss.

5. M

6. Go

7. Do this over and over for several epochs (one epoch is one pass through all the training data), until you meet some chosen *stopping criteria*

Define a neural network model with an architecture and a set of trainable parameters (weights and biases)

1.Initi

2.Col

3. Ca
predi

4. Ca
loss.

5. M

6. Go

7. Do this over and ... the chosen *stopping criteria*

Define a neural network model with an architecture and a set of trainable parameters (weights and biases)

1. Initialize the parameters

2.Collect a

3. Calculate

predictions

4. Calculate

loss. In othe

5. Move ea

par

6. Go back

7. Do this o

criteria

We'll see this translated into code soon

true and the

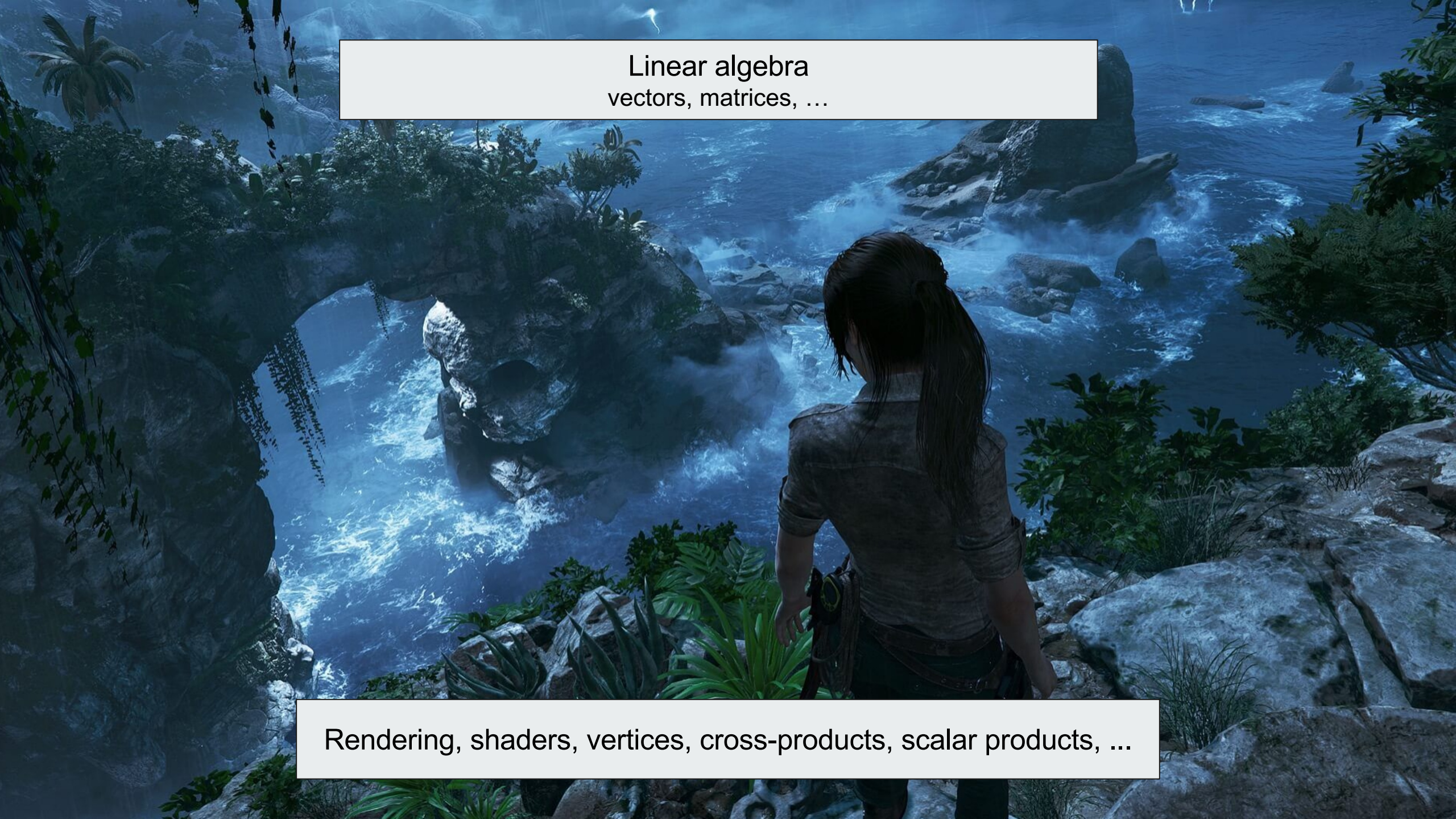
would change the

chosen *stopping*

GPUs: *Massively parallel linear algebra super-computers*

Developed for computer graphics

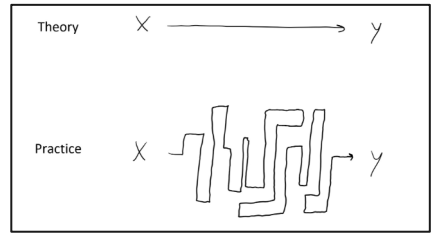
Good at exactly what's needed for neural networks!


Linear algebra
vectors, matrices, …
Rendering, shaders, vertices, cross-products, scalar products, ...

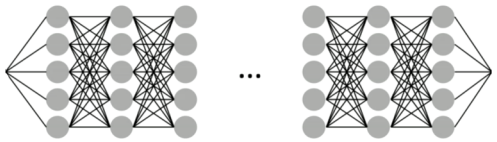
01 Machine learning: **function approximation** based on **training**

$$y \approx f(x;\theta)$$

02 We pick the model family f. The parameters θ are found automatically through training



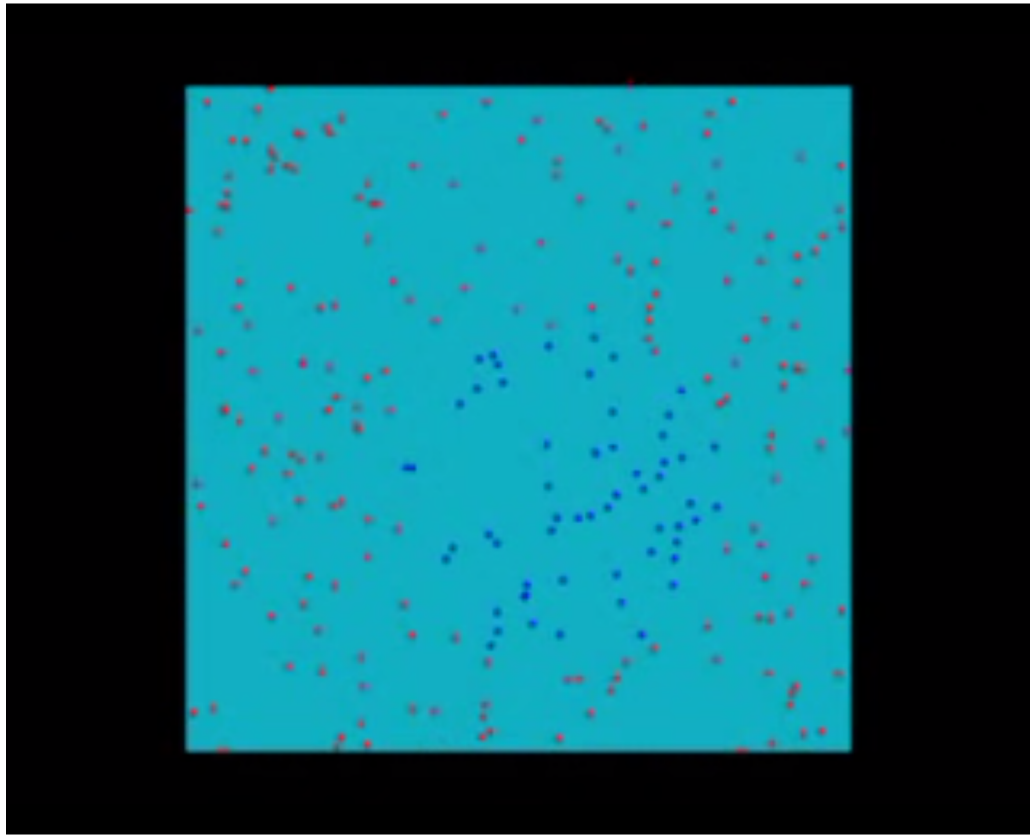
03 In practice: some models are better suited to some tasks than others.



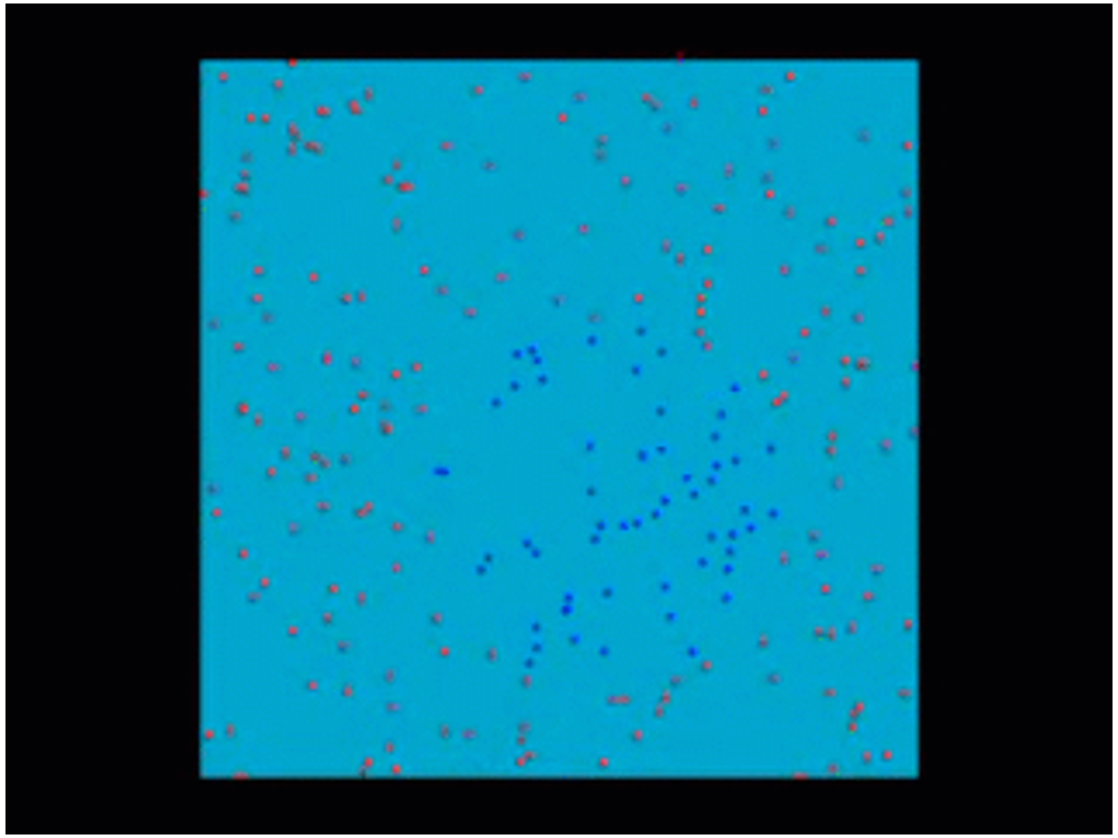
04 In deep learning: a particular choice of f. A **composition** of simple functions, trained jointly, typically using GPUs or other accelerators.

Representations and the power of transformations

Transformations

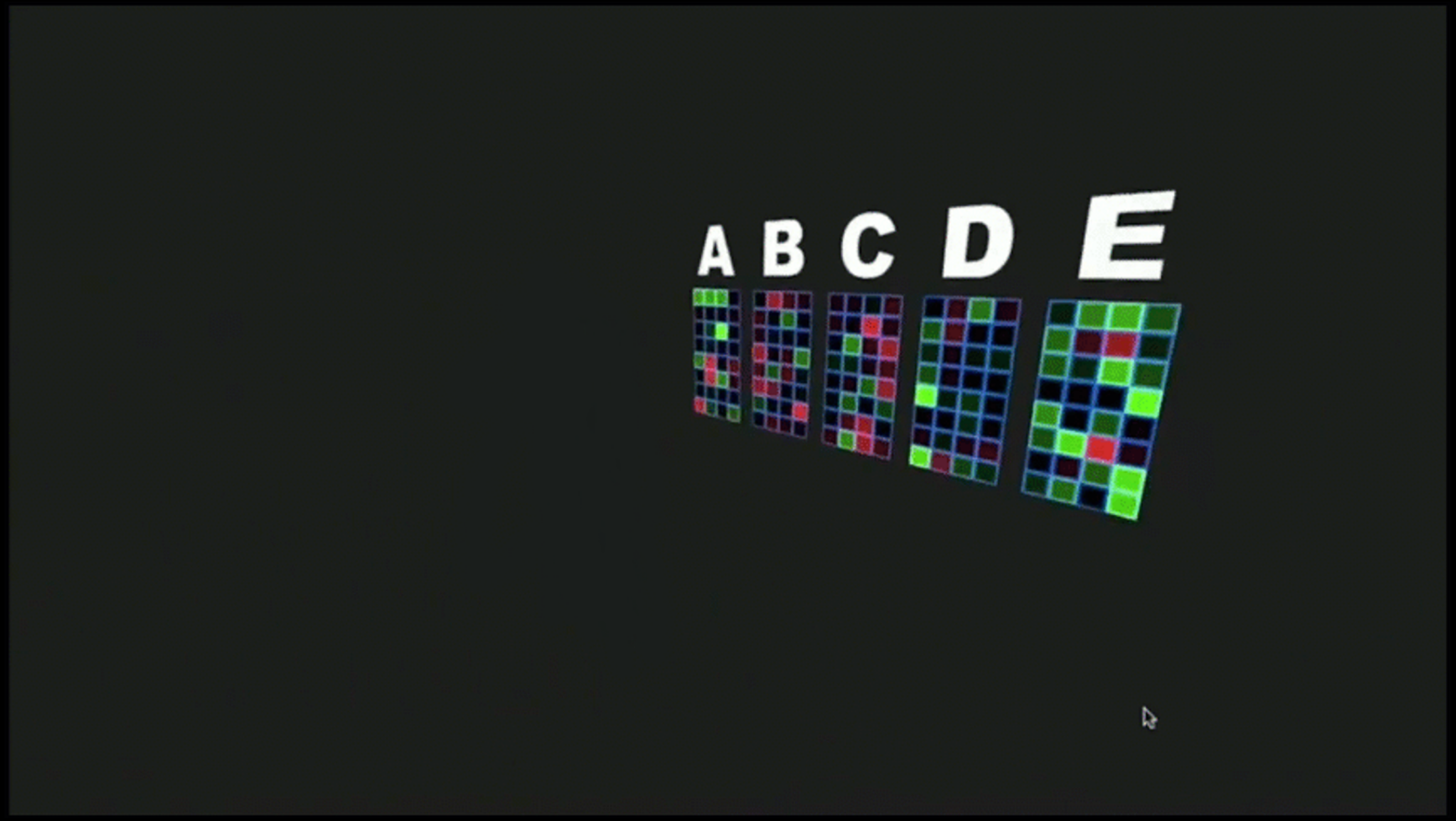


Transformation

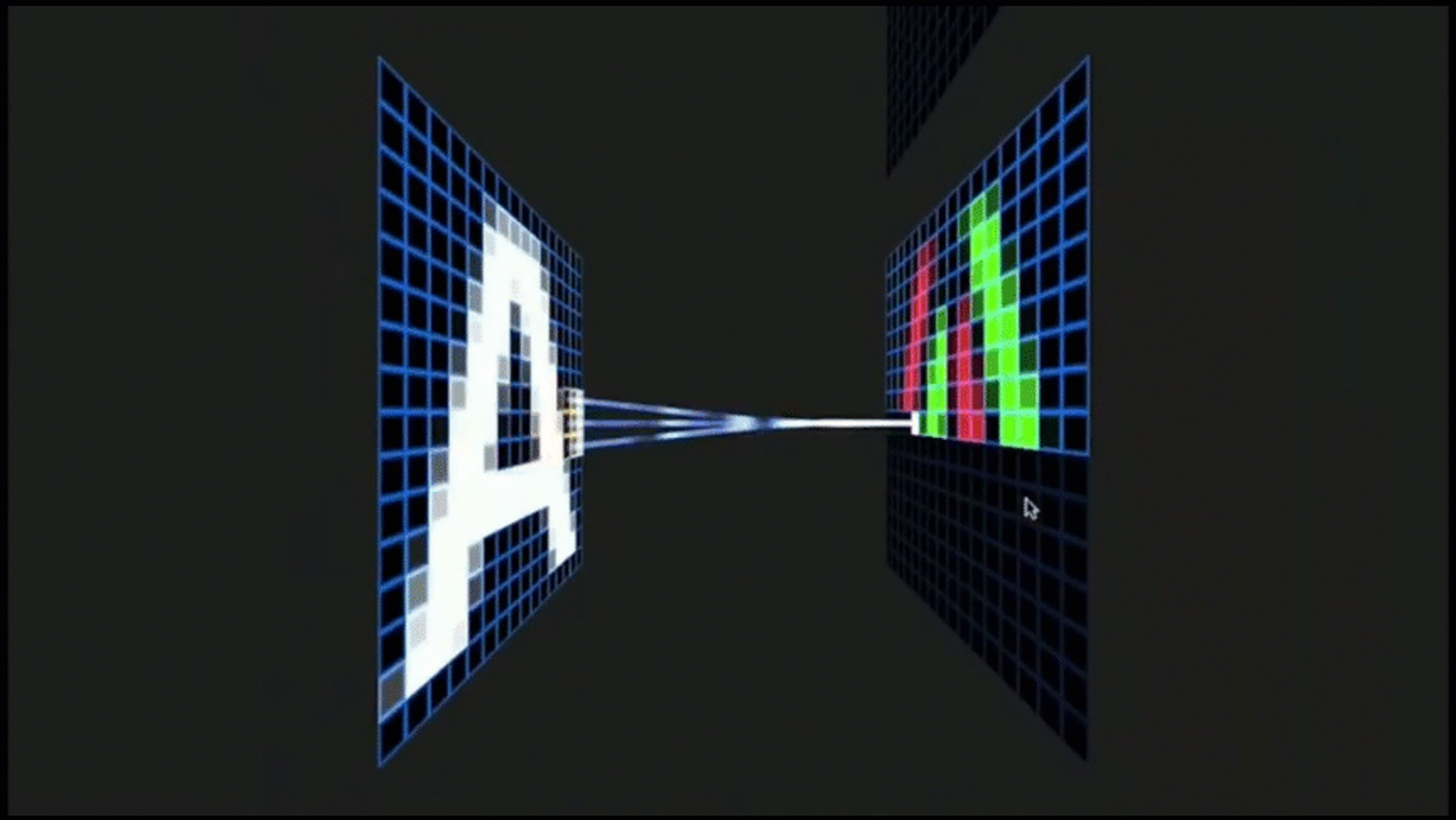


What transformations should be applied?

In deep learning: ***transformations providing useful representations automatically*** **found via training**



Source: Otavio Good

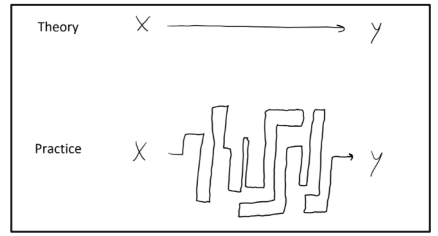


Source: Otavio Good

01 Machine learning: **function approximation** based on **training**

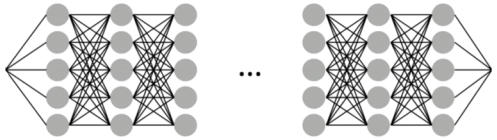
$$y \approx f(x;\theta)$$

02 We pick the model family f. The parameters θ are found automatically through training

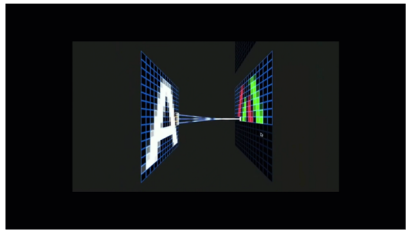


03 In practice: some models are better suited to some tasks than others.

04 In deep learning: a particular choice of f. A **composition** of simple functions, trained jointly, typically using GPUs or other accelerators.



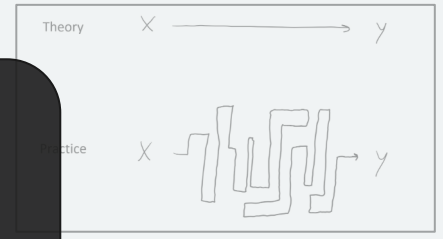
05 In deep learning: finding a **hierarchical set of geometric transformations**, ending in a representation that makes the desired task easy to solve.



01 Machine learning: **function approximation** based on **training**

$$y \approx f(x;\theta)$$

02 We pick the model family $\mathcal{F}$. The parameters Θ are found automatically through training



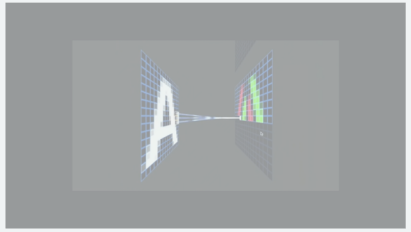
03 In practice: some models are better suited to some tasks than others.

Deep learning

searching for good hierarchical geometric representations

04 In deep learning: a particular choice of $\mathcal{F}$. A **composition** of simple functions, trained jointly, typically using GPUs or other accelerators.

05 In deep learning: finding a **hierarchical set of geometric transformations**, ending in a representation that makes the desired task easy to solve.



Not a panacea!

Not a panacea!

datacamp | Learn | Workspace | Certification | Jobs | Groups | Search Catalog

- My Progress
- My Bookmarks
- Leaderboard
- Assignments

CATALOG

- Tracks
- Competitions BETA
- Assessments
- Courses
- Practice
- Projects
- Live Events

INTERACTIVE COURSE

Supervised Learning with scikit-learn

Continue Course | Bookmark

4 hours | 17 Videos | 54 Exercises | 302,704 Participants | 4,200 XP

Course Description

Machine learning is the field that teaches machines and computers to learn from existing data to make predictions on new data: Will a tumor be benign or malignant? Which of your customers will take their business elsewhere? Is a particular email spam? In this course, you'll learn how to use Python to perform supervised learning, an essential component of machine learning. You'll learn how to build predictive models, tune their parameters, and determine how well they will perform with unseen data—all while using real world datasets. You'll be using scikit-learn, one of the most popular and user-friendly machine learning libraries for Python.

This course is part of these tracks:

Data Scientist with Python

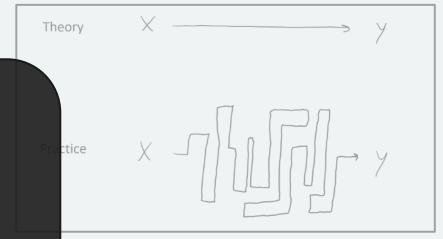
Machine Learning Fundamentals with Python

Machine Learning Scientist with Python

01 Machine learning: **function approximation** based on **training**

$$y \approx f(x;\theta)$$

02 We pick the model family $\mathcal{F}$. The parameters Θ are found automatically through training



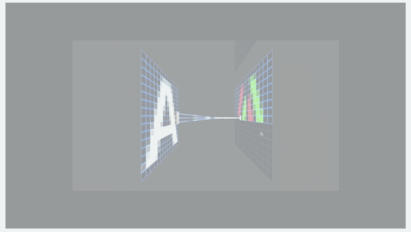
03 In practice: some models are better suited to some tasks than others.

Deep learning

searching for good hierarchical geometric representations

04 In deep learning: a particular choice of $\mathcal{F}$. A **composition** of simple functions, trained jointly, typically using GPUs or other accelerators.

05 In deep learning: finding a **hierarchical set of geometric transformations**, ending in a representation that makes the desired task easy to solve.



Define a neural network model with an architecture and a set of trainable parameters (weights and biases)

1. Initialize the parameters

2.Collect a batch of inputs `(X,y)` and use the model with its current parameters to make predictions `y_pred`

3. Calculate how good the model is using the loss function to measure the discrepancy between the known labels `y_true` and the predictions `y_pred` for this batch

4. Calculate the gradients for each parameter using backpropagation. This measures how changing each parameter would change the loss. In other words, each parameter's contribution to the loss.

5. Move each weight a little bit in the opposite direction of its gradient using gradient descent. I.e., some variant of

```
parameter = parameter - learning_rate * gradient
```

6. Go back to step 2 and repeat the process

7. Do this over and over for several epochs (one epoch is one pass through all the training data), until you meet some chosen *stopping criteria*

Define a neural network model with an architecture and a set of trainable parameters (weights and biases)

1. Initialize the parameters

2.Collect a

3. Calculate

predictions

4. Calculate

loss. In othe

5. Move ea

par

6. Go back

7. Do this o

criteria

Let's see this translated into code!

true and the

would change the

chosen *stopping*