

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

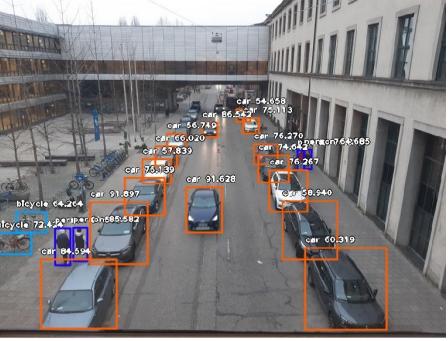
Function approximation

Function approximation

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

what's in the image

an image

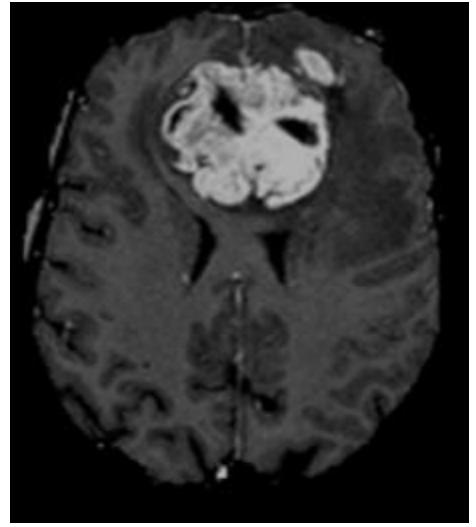
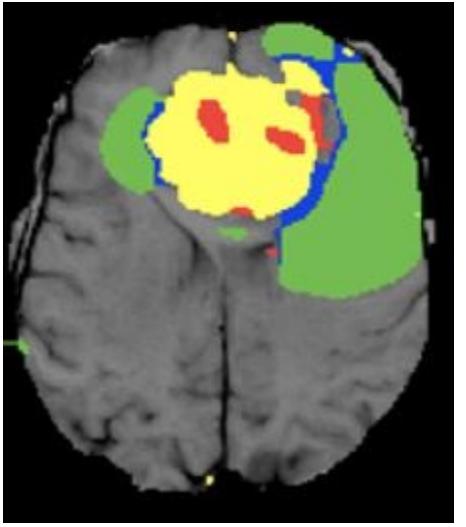


Function approximation

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

what's in the image

an image



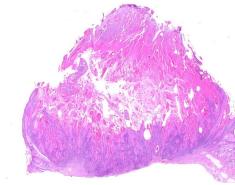
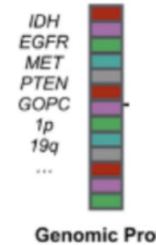
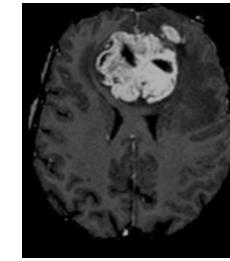
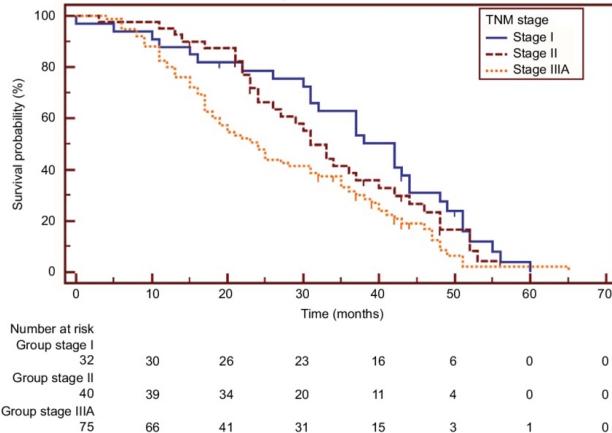
Function approximation

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

what's in the image

an image

The equation $y \approx f(x; \theta)$ represents function approximation. An arrow points from the text "what's in the image" to the input variable x in the equation. Another arrow points from the text "an image" to the output variable y .



| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|-----|------------|---------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1. | 11/10/2020 | Healthy | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2. | 11/10/2020 | Healthy | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3. | 11/10/2020 | Healthy | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4. | 11/10/2020 | Healthy | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5. | 11/10/2020 | Healthy | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6. | 11/10/2020 | Healthy | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7. | 11/10/2020 | Healthy | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8. | 11/10/2020 | Healthy | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9. | 11/10/2020 | Healthy | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10. | 11/10/2020 | Healthy | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11. | 11/10/2020 | Healthy | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12. | 11/10/2020 | Healthy | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13. | 11/10/2020 | Healthy | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14. | 11/10/2020 | Healthy | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15. | 11/10/2020 | Healthy | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16. | 11/10/2020 | Healthy | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17. | 11/10/2020 | Healthy | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18. | 11/10/2020 | Healthy | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 19. | 11/10/2020 | Healthy | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20. | 11/10/2020 | Healthy | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Function approximation

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

outcome (e.g., survival)

heterogeneous information

Training

$$(X, y)$$

Training

$$(X, y)$$

Input data

Labels

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

- ill-defined homogeneous opacity obscuring vessels
- **Silhouette sign:** loss of lung/soft tissue interface
- Air-bronchogram
- Extent to the pleura or fissure, but not crossing it
- No volume loss

Train on features extracted from the data

No opacity
not normal

Round pneumonias are round-ish and while they are well-circumscribed parenchymal opacities, they tend to have irregular margins. They most commonly occur in superior segments of lower lobes and in the majority of cases (98%), they are solitary⁵.

In other words, **representations** or **views** of the data

Air-bronchograms are often present and helpful in clinching the diagnosis. Interestingly, they are only seen in 17% round pneumonia when they occur in adults².

Because the inflammation is often limited to the pulmonary interstitium and the interlobular septa, atypical pneumonia has the radiographic features of patchy reticular or reticulonodular opacities. These opacities are especially seen in the peripheral lung.⁵ Subsegmental and sometimes segmental atelectasis from small airway obstruction may occur. The radiographic features are often more extensive than what is suggested clinically.

Training is a search for representations that make the task easy (or easier) to solve, guided by the labels of the training data set.

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

May show subtle area of lucency superimposed on a region of **consolidation**.

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

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

features of a disease,
process, patient or
system

healthy or not healthy /
outcome

Models and parameters

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


The diagram consists of two labels, "model" and "parameters", positioned at the bottom left and bottom right respectively. Two arrows originate from these labels and point towards the function $f(x; \theta)$ in the center.

***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**

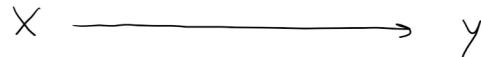
02

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

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

Theory versus practice

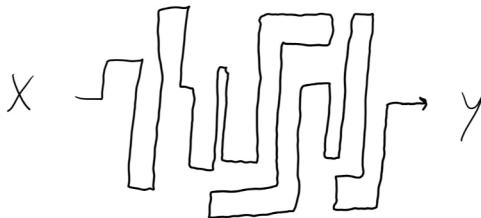
Theory



In theory:

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

Practice



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**

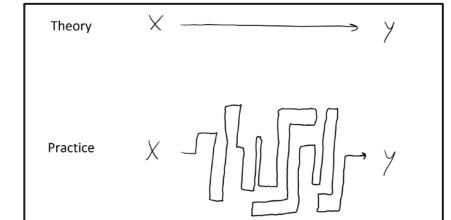
02

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

03

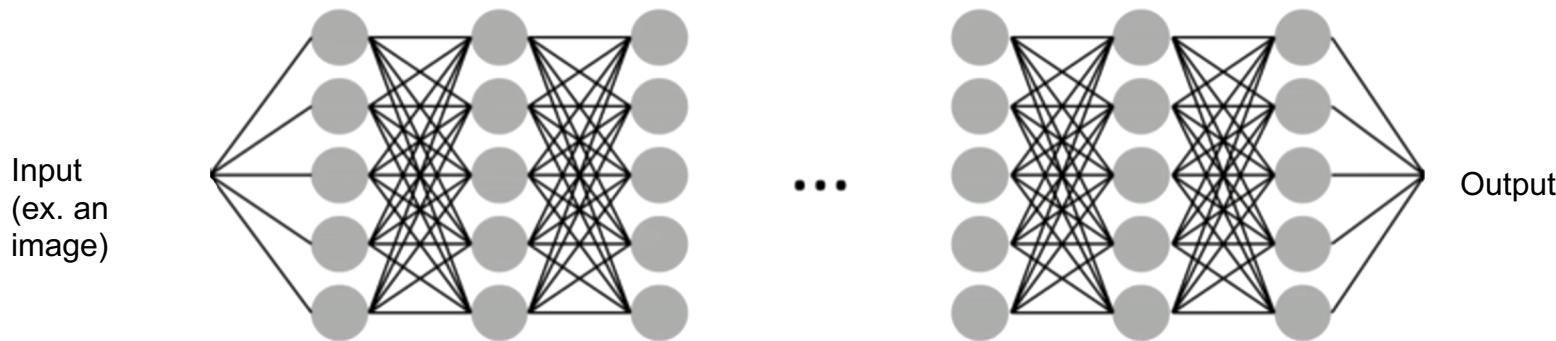
In practice: some models are better suited to some tasks than others.

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



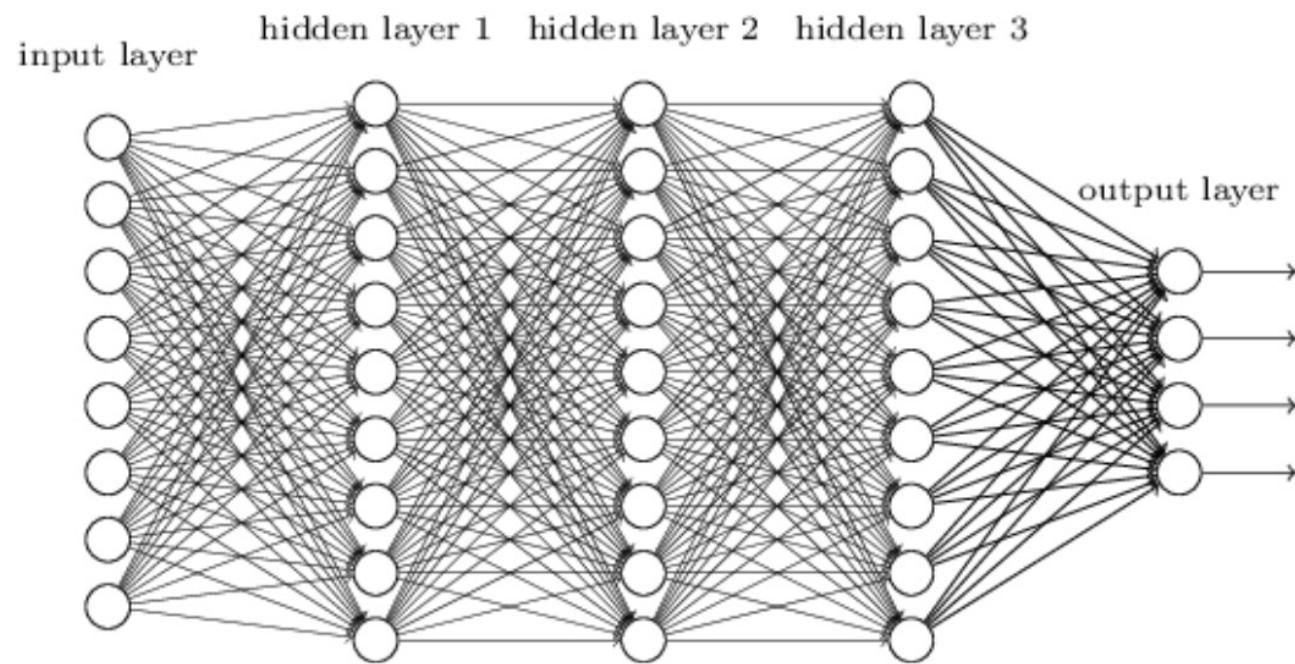
Deep learning

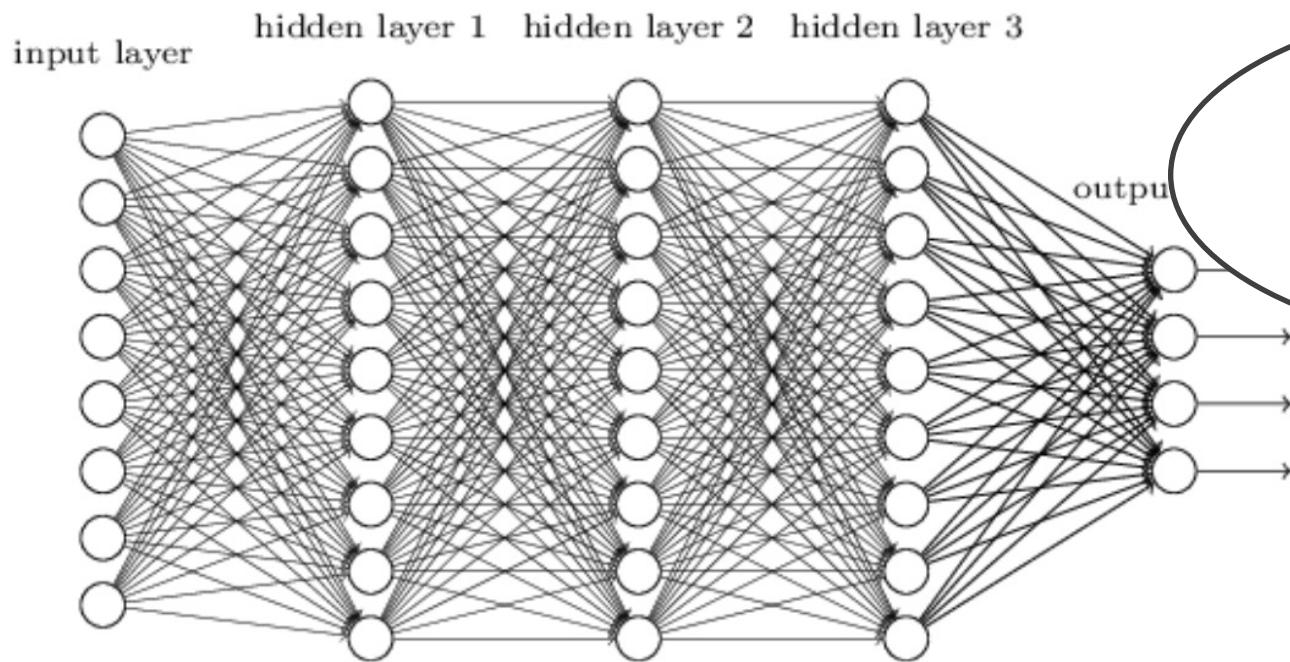
$$y \approx f(x; \theta)$$



Artificial neural networks

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





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:

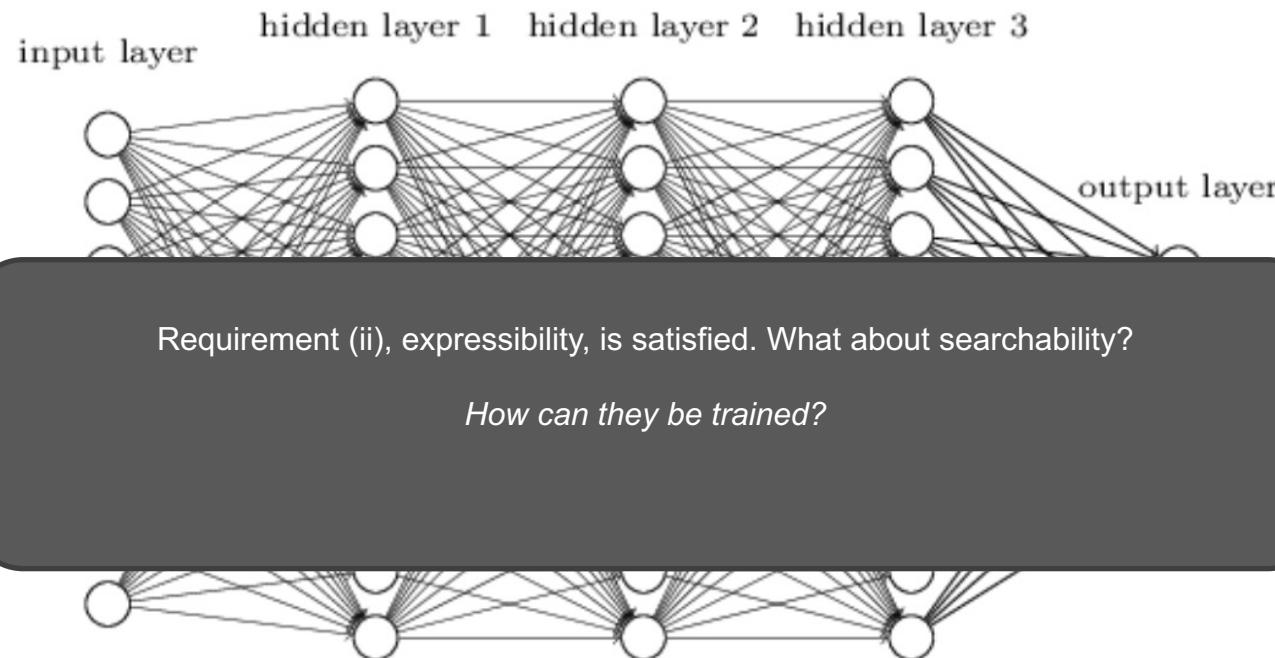
$$y = \sigma_{k+1}(W_{k+1}^T \sigma_k(W_k^T \left(\dots \left(\sigma_1(W_1^T x) \right) \dots \right)) = f(x; \theta)$$

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.



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

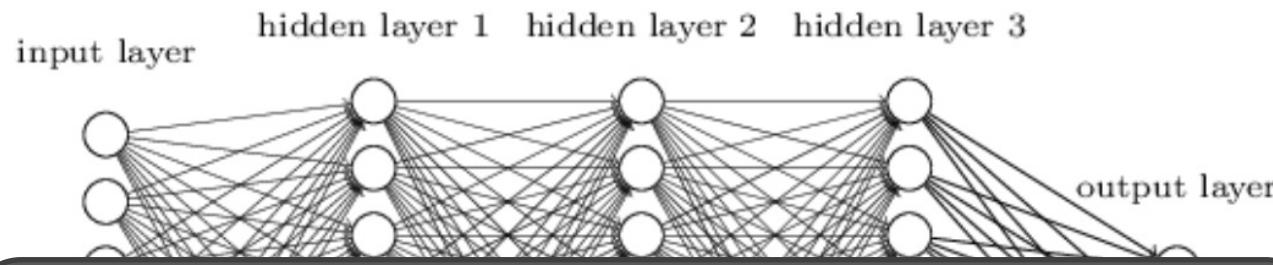
$$y = \sigma_{k+1}(W_{k+1}^T \sigma_k(W_k^T \left(\dots \left(\sigma_1(W_1^T x) \right) \dots \right)) = f(x; \theta)$$

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 \mathcal{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:

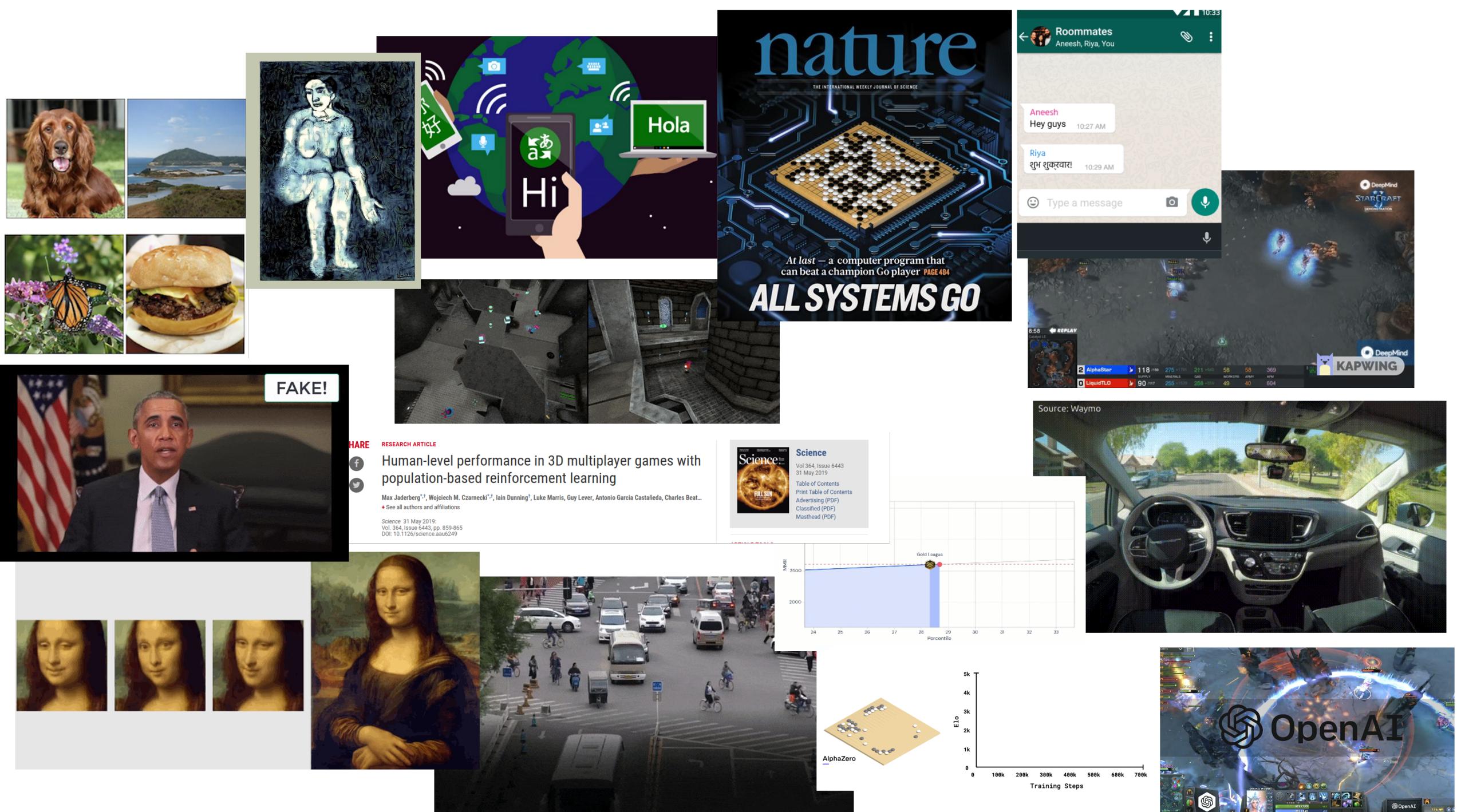
$$y = \sigma_{k+1}(W_{k+1}^T \sigma_k(W_k^T \left(\dots \left(\sigma_1(W_1^T x) \right) \dots \right)) = f(x; \theta)$$

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 \mathcal{F} is complex transformation broken down into a composition of simpler ones, parametrized by the layer weights.



**Skin cancer can't hide from deep-learning diagnostics**May 21, 2019 | [Dave Pearson](#) | [Diagnostics](#)**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](#)**Deep learning predicts OCT measures of diabetic macular thickening**By [Steve Lenier](#)

May 21, 2019

Novel Molecules Designed by Artificial Intelligence May Accelerate Drug Discovery**TOPICS:** Artificial Intelligence Biotechnology InSilico Medicine[News Home](#) All News
[HOME > NEWS > ALL NEWS > AI CRACKS THE CODE OF PROTEIN COMPLEXES—PROVIDING A ROAD MAP FOR](#)
AI cracks the code of protein complexes—providing a road map for
software maps thousands of the partnered proteins

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

Results

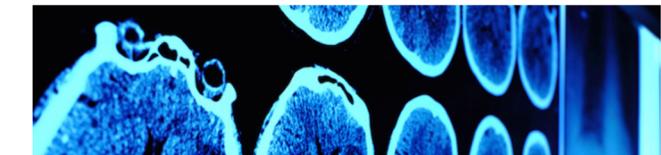
tein complex prediction with AlphaFold-Multimer

chard Evans, Michael O'Neill, Alexander Pritzel, Natasha Antropova, Andrew Green, Augustin Žídek, Russ Bates, Sam Blackwell, Jason Yim, Olaf Röbke-Bodensteiner, Michal Zeliński, Alex Bridgland, Anna Potapenko, Andrew Jethryn Tunyasuvunakool, Rishabh Jain, Ellen Clancy, Pushmeet Kohli, Johnnemis Hassabis
<https://doi.org/10.1101/2021.10.04.463034>

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

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.

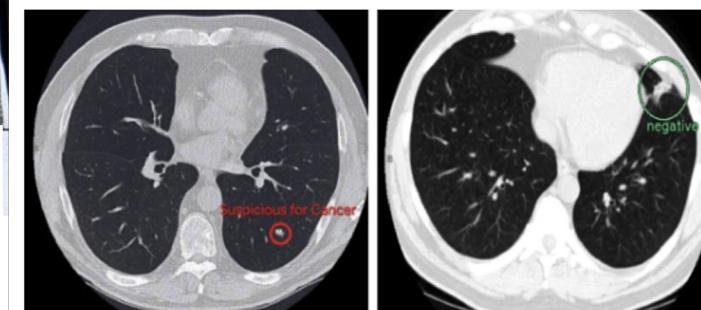
"Interpreting X-ray images to diagnose pathologies like



AI

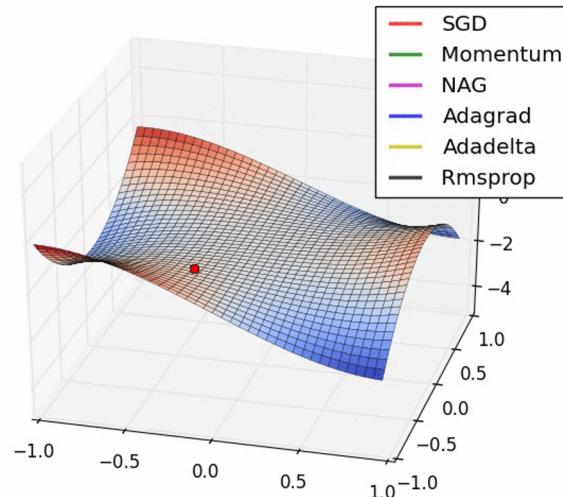
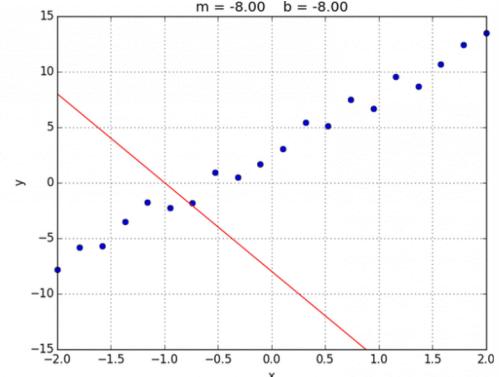
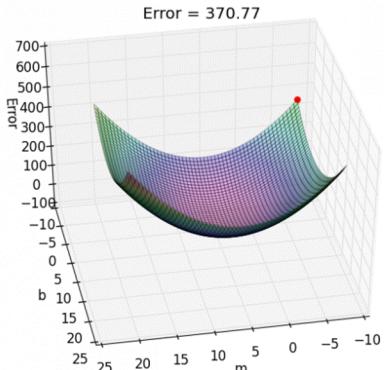
Google's lung cancer detection AI outperforms 6 human radiologists

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

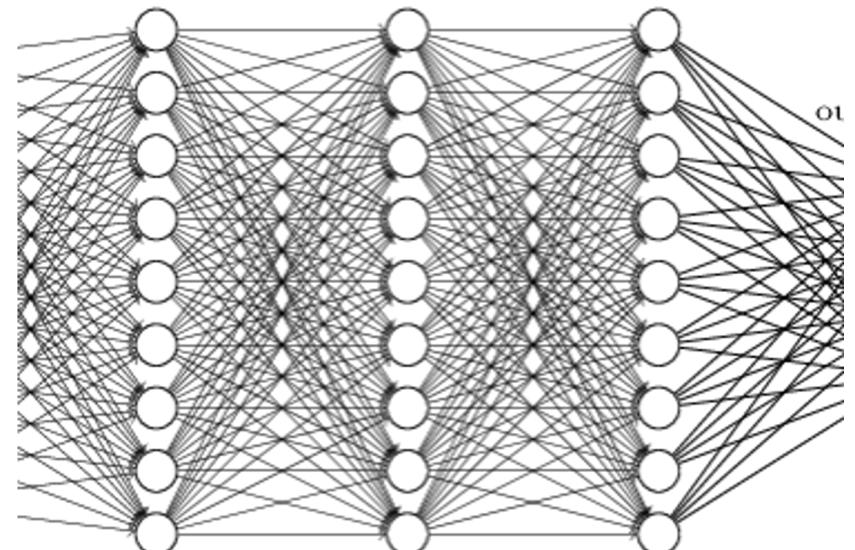


Three basic ingredients:

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*.



hidden layer 1 hidden layer 2 hidden layer 3



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
$$\text{parameter} = \text{parameter} - \text{learning_rate} * \text{gradient}$$

6. Go back to step 2 and repeat the process

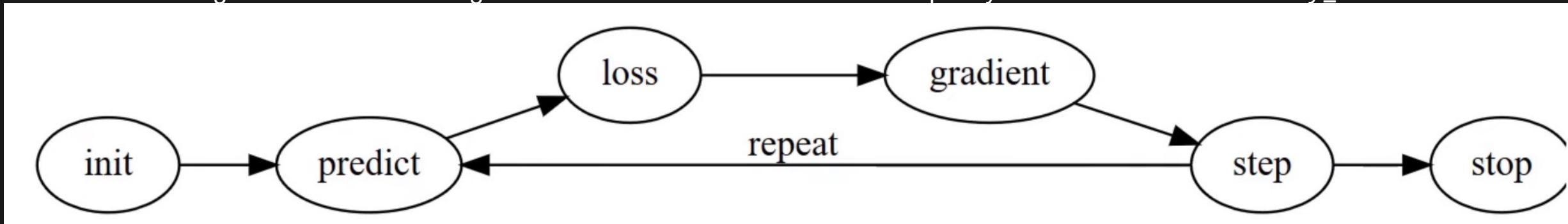
7. Do this over and over for several epochs (one epoch is one pass through all the training data), until you meet some chosen stopping criteria

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
$$\text{parameter} = \text{parameter} - \text{learning_rate} * \text{gradient}$$

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
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
$$\text{parameter} = \text{parameter} - \text{learning_rate} * \text{gradient}$$
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

What is gradient descent?

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

1. Initialize

2. Collect

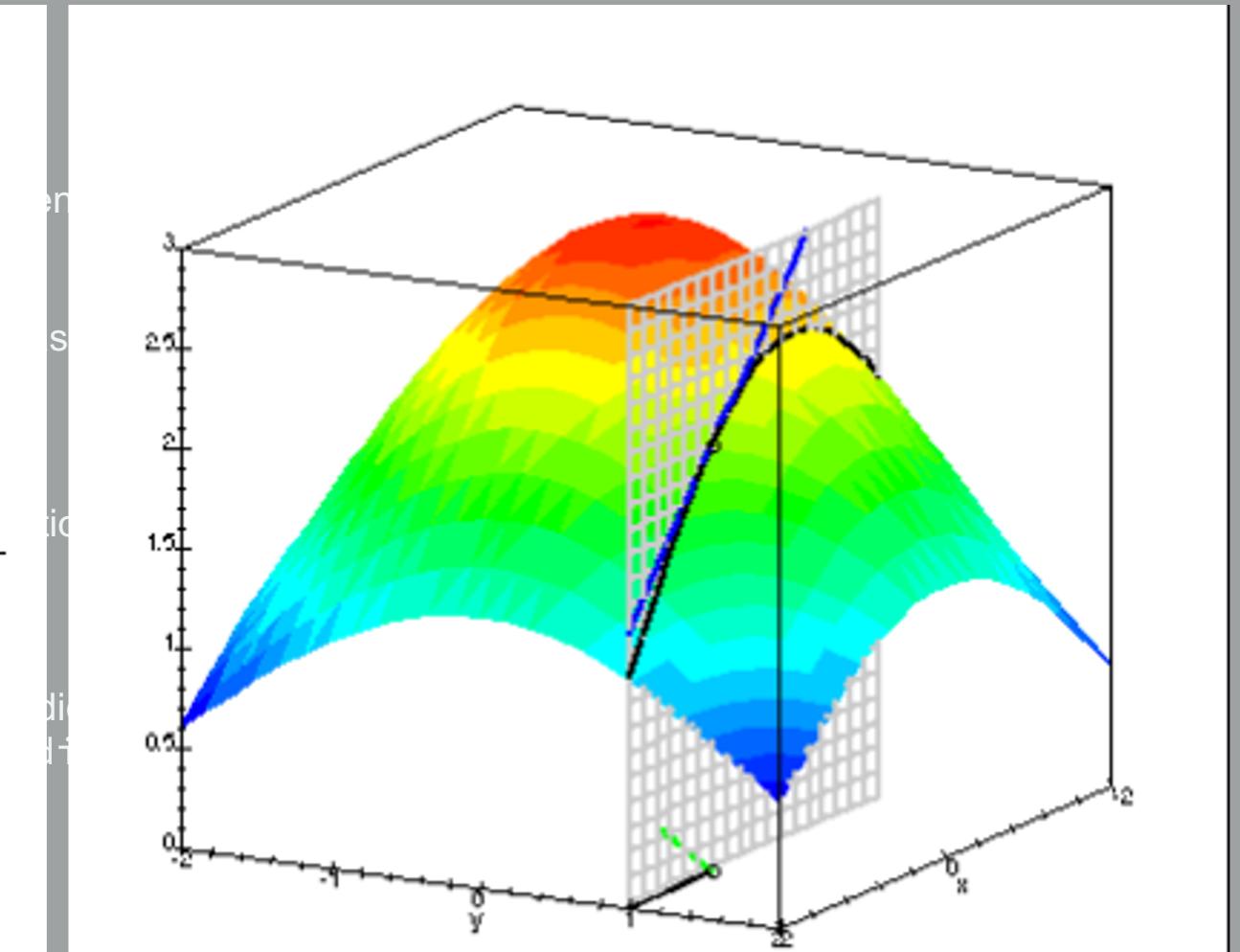
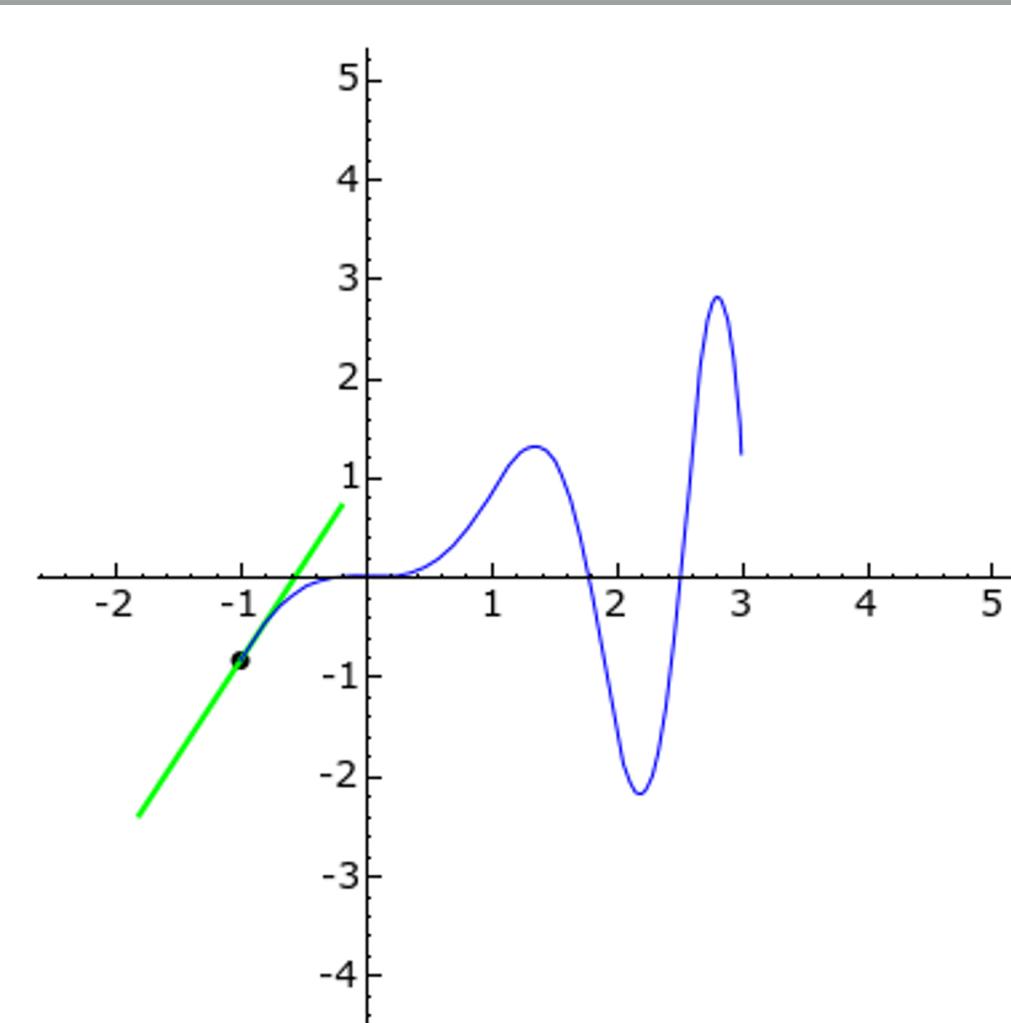
3. Calculate
predictions

4. Calculate
loss

5. Minimize

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. Init.

2. Col.

3. Cal.
pred.

4. Cal.
loss.

5. Min.

6. Go

7. Do this over and
over until the chosen stopping



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

We'll see this translated into code soon

1. Initialize the parameters

2. Collect a

3. Calculate predictions

4. Calculate loss. In other words, calculate how far off our predictions were from the true values. This would change the loss function.

5. Move each parameter in the direction that would decrease the loss. In other words, move each parameter in the direction that would increase the probability of getting the correct prediction. This would change the learning rate.

6. Go back to step 2 and repeat

7. Do this often enough until the loss function is small enough or until some chosen stopping criteria

The screenshot shows a Jupyter Notebook interface with a dark theme. The title bar reads '1.0-asl-nnets_building_blocks_part1.ipynb'. The main area displays a 'Table of Contents' with the following structure:

- [1 Introduction](#)
 - [1.1 Objectives](#)
 - [1.2 Notebooks](#)
- [2 Setup](#)
- [3 Load data](#)
 - [3.1 Load data into PyTorch](#)
 - [3.1.1 Inspecting the data](#)
 - [3.2 Data loaders](#)
- [4 A simple neural network in PyTorch](#)
 - [4.1 A fully-connected neural net in PyTorch](#)
- [5 Training the network](#)
 - [5.1 1: Initialization](#)
 - [5.2 2: Grab a batch and predict](#)
 - [5.2.1 Feeding the batch to the network](#)
 - [5.3 3: Calculate loss](#)
 - [5.4 4: Calculate gradients](#)
 - [5.5 5: Step / update the weights](#)
 - [5.5.1 Did that help?](#)
 - [5.6 6: Go back to step 2 and repeat](#)
 - [5.7 7: Iterate until stopping criterion is met](#)
 - [5.8 Evaluate the results](#)
- [6 Wrapping up](#)
- [7 The "deep" in deep learning](#)
- [8 Convolutional neural networks](#)
 - [8.1 A CNN in PyTorch](#)



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**

02

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

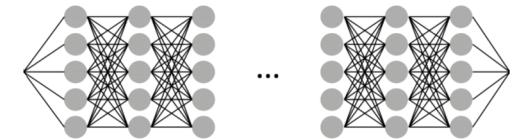
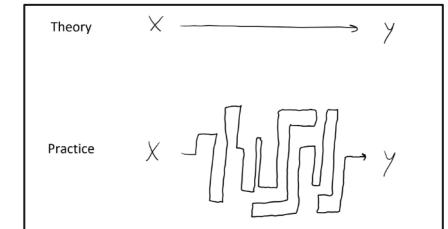
03

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

04

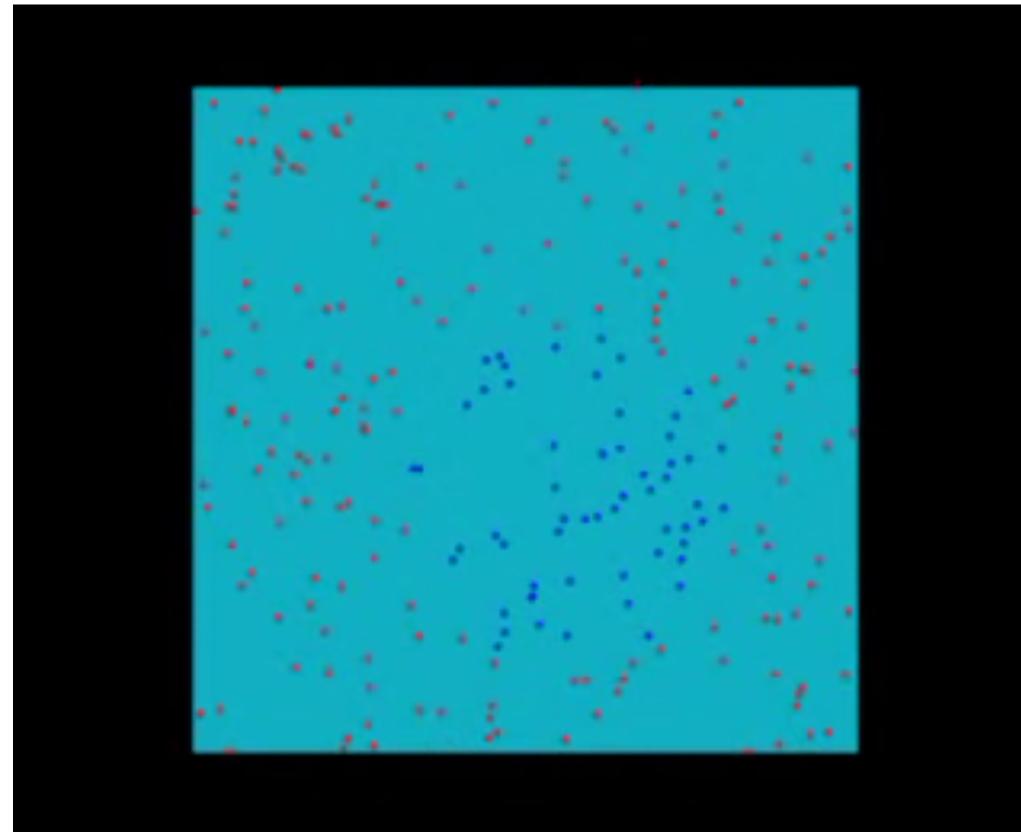
In deep learning: a particular choice of \mathcal{F} . A **composition** of simple functions, trained jointly, typically using GPUs or other accelerators.

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

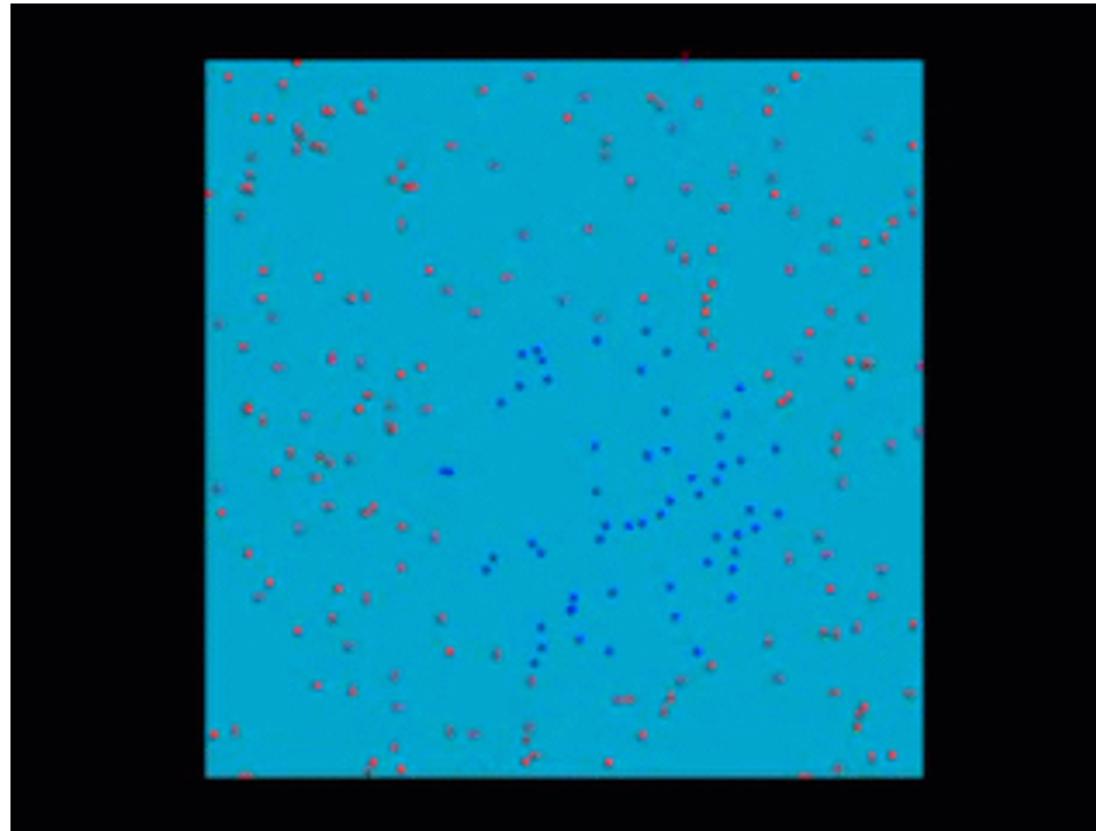


Representations and the power of transformations

Transformations

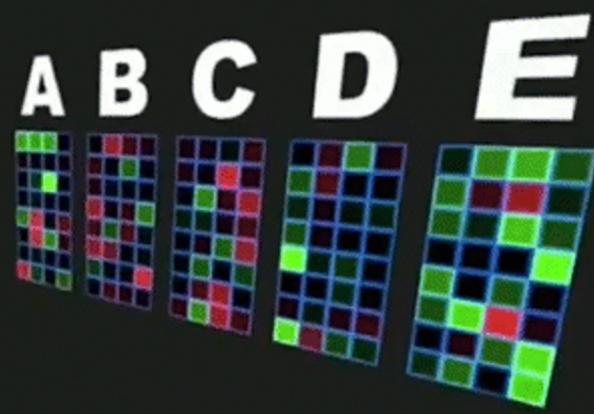


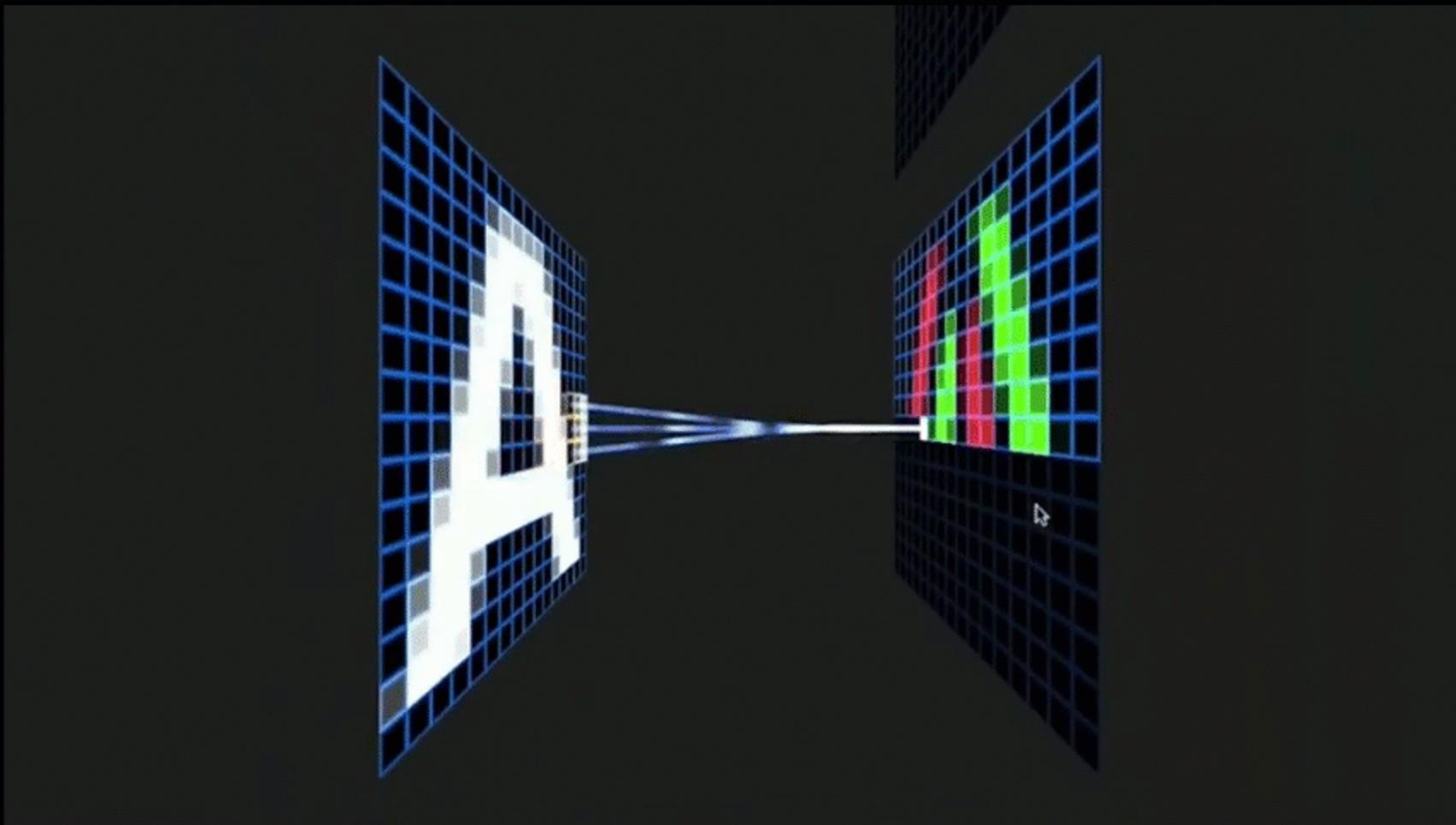
Transformation



What transformations should be applied?

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





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.

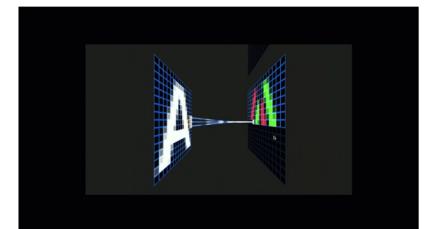
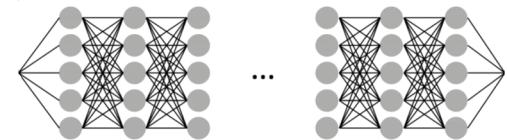
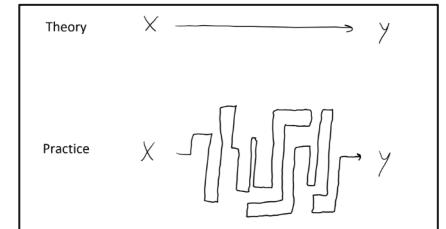
04

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.

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



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

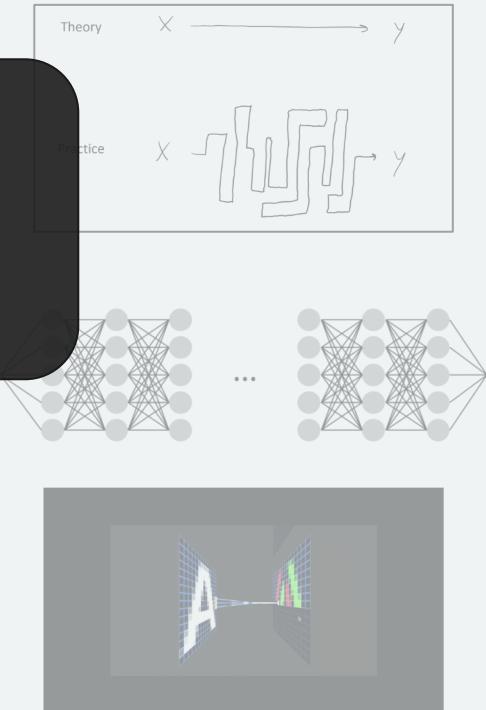
searching for good hierarchical geometric representations

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!

WARNING

WARNING

Not a panacea!

The screenshot shows the DataCamp learning platform interface. On the left, a dark sidebar menu includes 'My Progress', 'My Bookmarks', 'Leaderboard', 'Assignments', 'CATALOG' (with 'Tracks', 'Competitions BETA', and 'Assessments' listed), 'Courses' (selected), 'Practice', 'Projects', and 'Live Events'. The main content area displays an 'INTERACTIVE COURSE' titled 'Supervised Learning with scikit-learn'. It features a green 'Continue Course' button and a white 'Bookmark' button. Below the title, course statistics are shown: 4 hours, 17 Videos, 54 Exercises, 302,704 Participants, and 4,200 XP. A 'Course Description' section explains that machine learning teaches machines and computers to learn from existing data to make predictions on new data, using Python and scikit-learn libraries. To the right, a purple box lists tracks this course is part of: 'Data Scientist with Python', 'Machine Learning Fundamentals with Python', and 'Machine Learning Scientist with Python'.

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

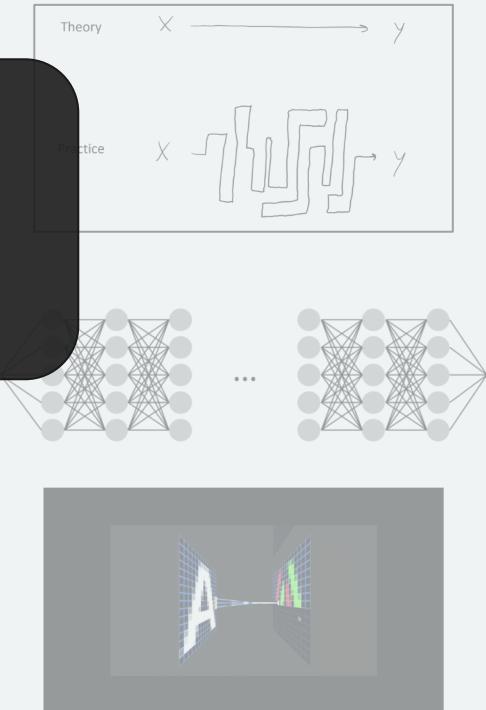
searching for good hierarchical geometric representations

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
$$\text{parameter} = \text{parameter} - \text{learning_rate} * \text{gradient}$$

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)

Let's see this translated into code!

1. Initialize the parameters

2. Collect a

3. Calculate predictions

4. Calculate loss. In other words, calculate how far off our predictions were from the true values. This would change the loss function.

5. Move each parameter in the direction that would decrease the loss. In other words, move each parameter in the direction that would increase the probability of getting the correct prediction. This would change the learning rate.

6. Go back to step 2 and repeat

7. Do this often enough until the loss function is small enough or the chosen stopping criteria is met.

Table of Contents

- [1 Introduction](#)
 - [1.1 Objectives](#)
 - [1.2 Notebooks](#)
- [2 Setup](#)
- [3 Load data](#)
 - [3.1 Load data into PyTorch](#)
 - [3.1.1 Inspecting the data](#)
 - [3.2 Data loaders](#)
- [4 A simple neural network in PyTorch](#)
 - [4.1 A fully-connected neural net in PyTorch](#)
- [5 Training the network](#)
 - [5.1.1 Initialization](#)
 - [5.2.2 Grab a batch and predict](#)
 - [5.2.1 Feeding the batch to the network](#)
 - [5.3.3 Calculate loss](#)
 - [5.4.4 Calculate gradients](#)
 - [5.5.5 Step / update the weights](#)
 - [5.5.1 Did that help?](#)
 - [5.6.6 Go back to step 2 and repeat](#)
 - [5.7.7 Iterate until stopping criterion is met](#)
 - [5.8 Evaluate the results](#)
- [6 Wrapping up](#)
- [7 The "deep" in deep learning](#)
- [8 Convolutional neural networks](#)
 - [8.1 A CNN in PyTorch](#)