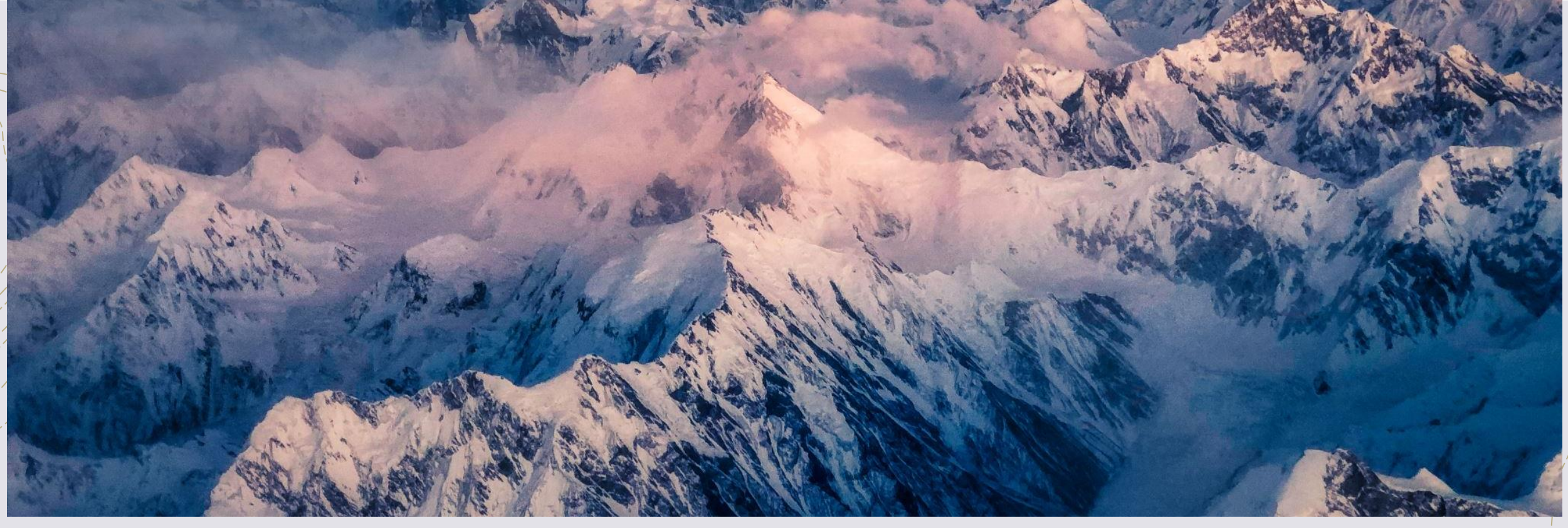


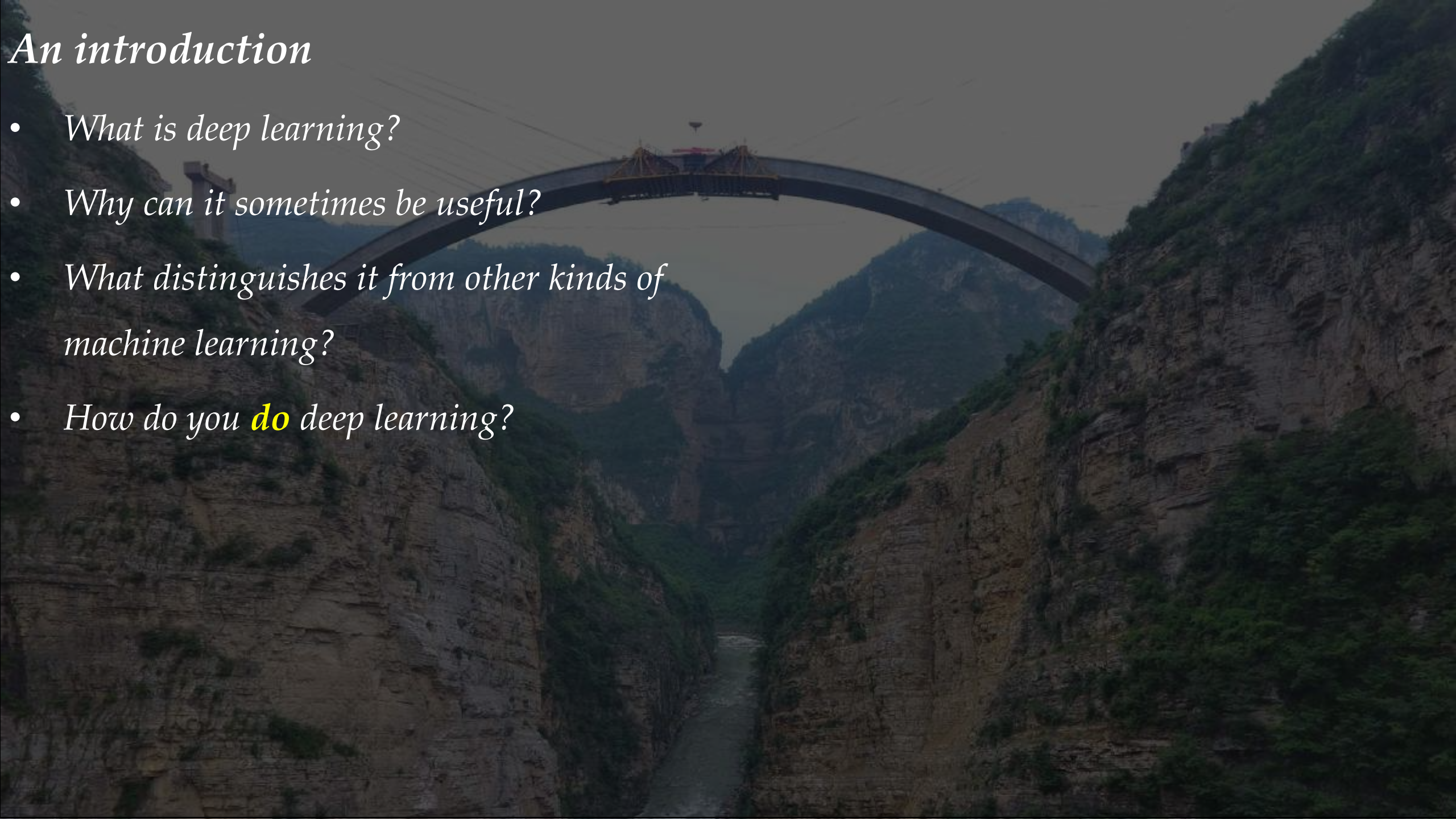
Deep learning: an introduction

ELMED219-2022

Thursday January 13, 2022

Western Norway University of Applied Sciences

Alexander S. Lundervold, lundervold.net



An introduction

- *What is deep learning?*
- *Why can it sometimes be useful?*
- *What distinguishes it from other kinds of machine learning?*
- *How do you **do** deep learning?*

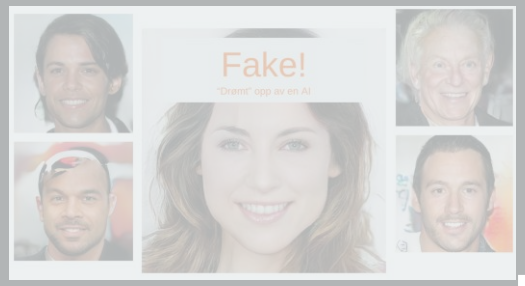
An introduction

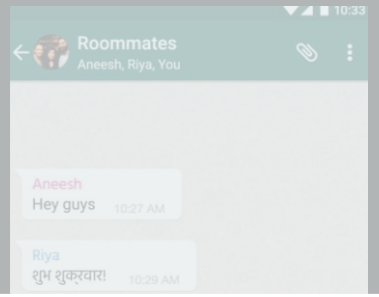
- *What is deep learning?*
- *Why can it sometimes be useful?*
- *What distinguishes it from other kinds of machine learning?*
- *How do you **do** deep learning?*

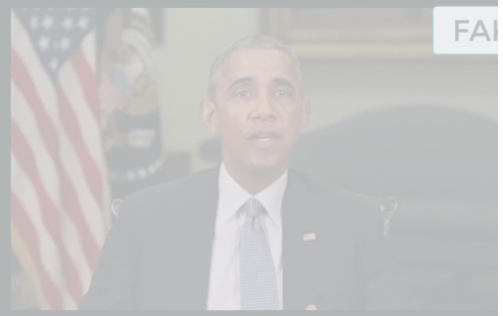
OBS

This is not at all a comprehensive introduction to deep learning, but a quick tour based on some specific points of view on the field.

Many great introductory resources exist! I will point to some at the end.



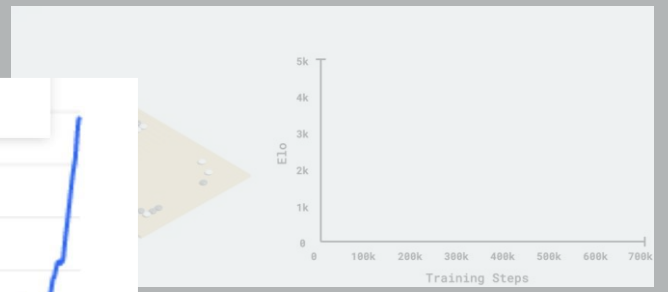


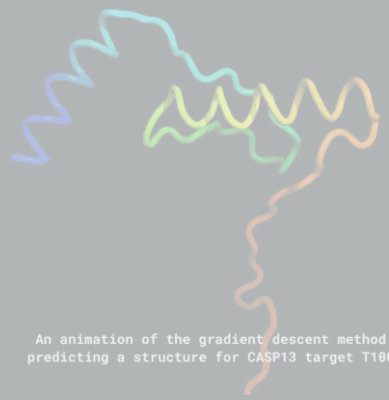


- Papers
- Researchers
- Applications
- Funding
- Investments
- Startups
- Users
- Public attention
- Ethical challenges

2010

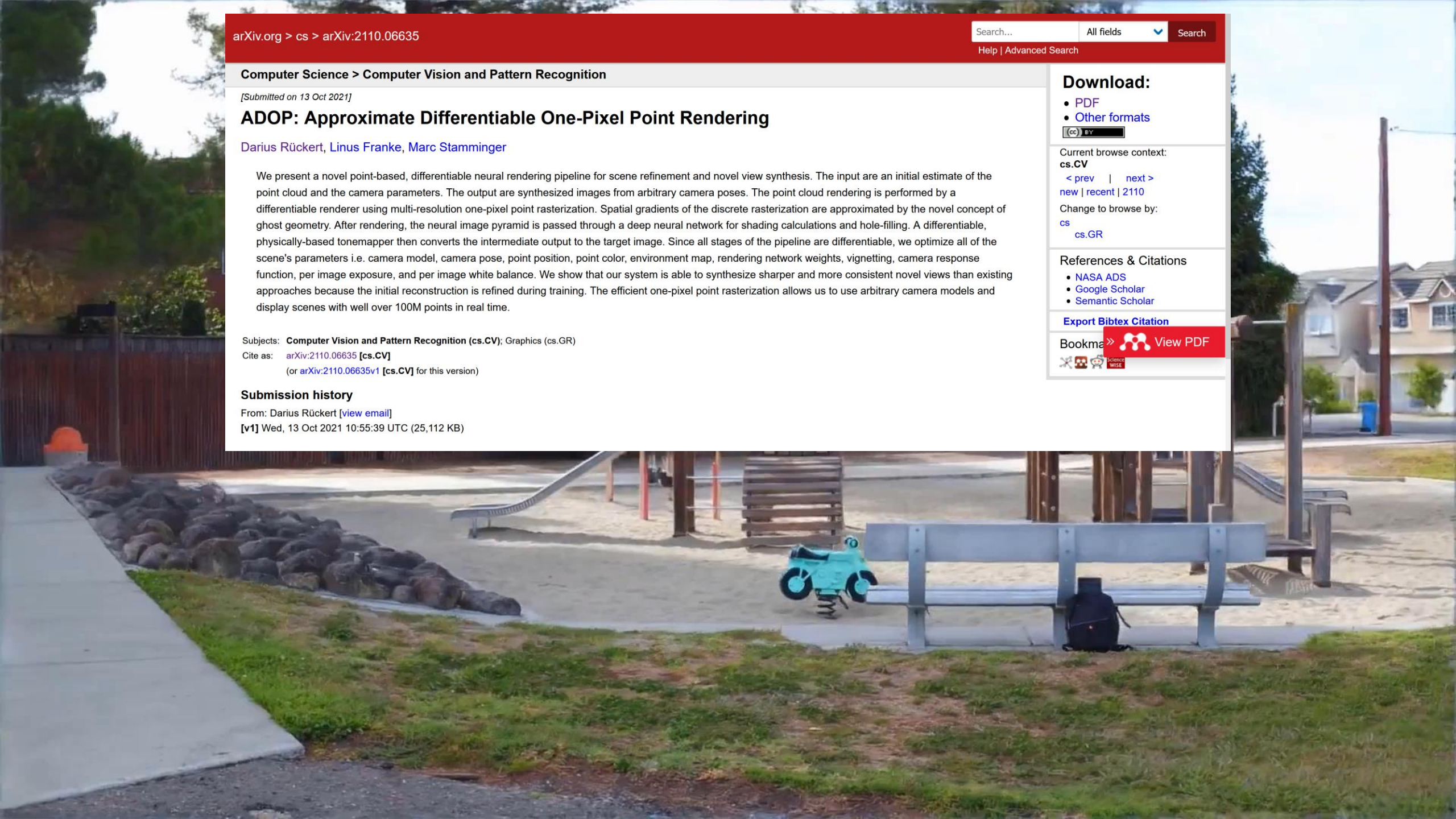
2020





An animation of the gradient descent method predicting a structure for CASP13 target T1008

NETFLIX



Computer Science > Computer Vision and Pattern Recognition

[Submitted on 13 Oct 2021]

ADOP: Approximate Differentiable One-Pixel Point Rendering

Darius Rückert, Linus Franke, Marc Stamminger

We present a novel point-based, differentiable neural rendering pipeline for scene refinement and novel view synthesis. The input are an initial estimate of the point cloud and the camera parameters. The output are synthesized images from arbitrary camera poses. The point cloud rendering is performed by a differentiable renderer using multi-resolution one-pixel point rasterization. Spatial gradients of the discrete rasterization are approximated by the novel concept of ghost geometry. After rendering, the neural image pyramid is passed through a deep neural network for shading calculations and hole-filling. A differentiable, physically-based tonemapper then converts the intermediate output to the target image. Since all stages of the pipeline are differentiable, we optimize all of the scene's parameters i.e. camera model, camera pose, point position, point color, environment map, rendering network weights, vignetting, camera response function, per image exposure, and per image white balance. We show that our system is able to synthesize sharper and more consistent novel views than existing approaches because the initial reconstruction is refined during training. The efficient one-pixel point rasterization allows us to use arbitrary camera models and display scenes with well over 100M points in real time.

Subjects: **Computer Vision and Pattern Recognition (cs.CV)**; Graphics (cs.GR)
Cite as: arXiv:2110.06635 **[cs.CV]**
(or arXiv:2110.06635v1 **[cs.CV]** for this version)

Submission history

From: Darius Rückert [view email]
[v1] Wed, 13 Oct 2021 10:55:39 UTC (25,112 KB)

Deep learning

searching for good hierarchical representations

01

02

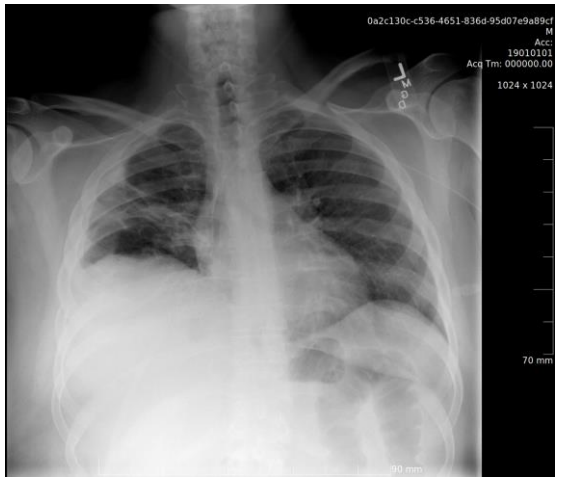
03

04

05

Function approximation

Opacity



Function approximation

features of a customer and an image a product

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

what is in the image

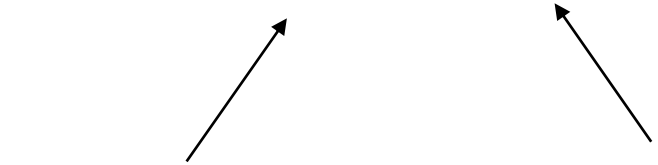
is it a cat

Training

$$(X, y)$$

Training

$$(X, y)$$



Input data

Labels

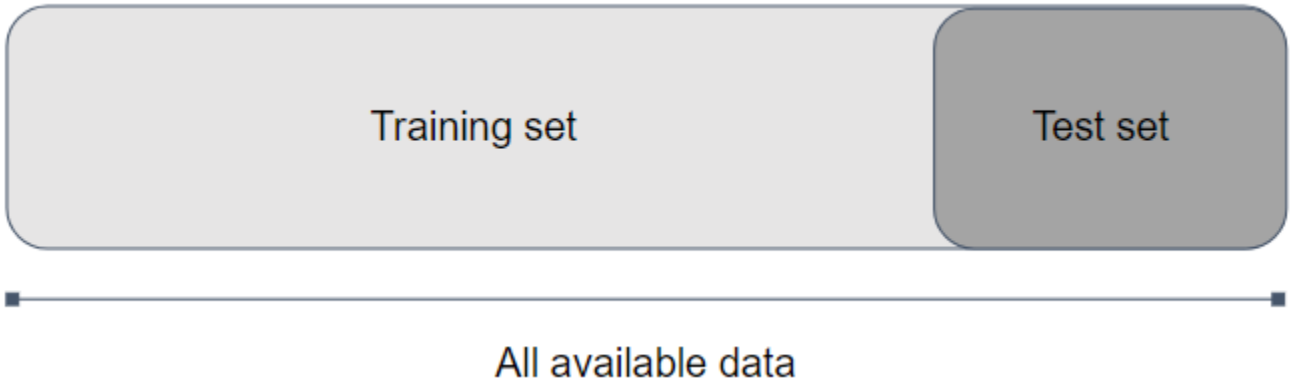
*This is the setup in what's called **supervised learning***

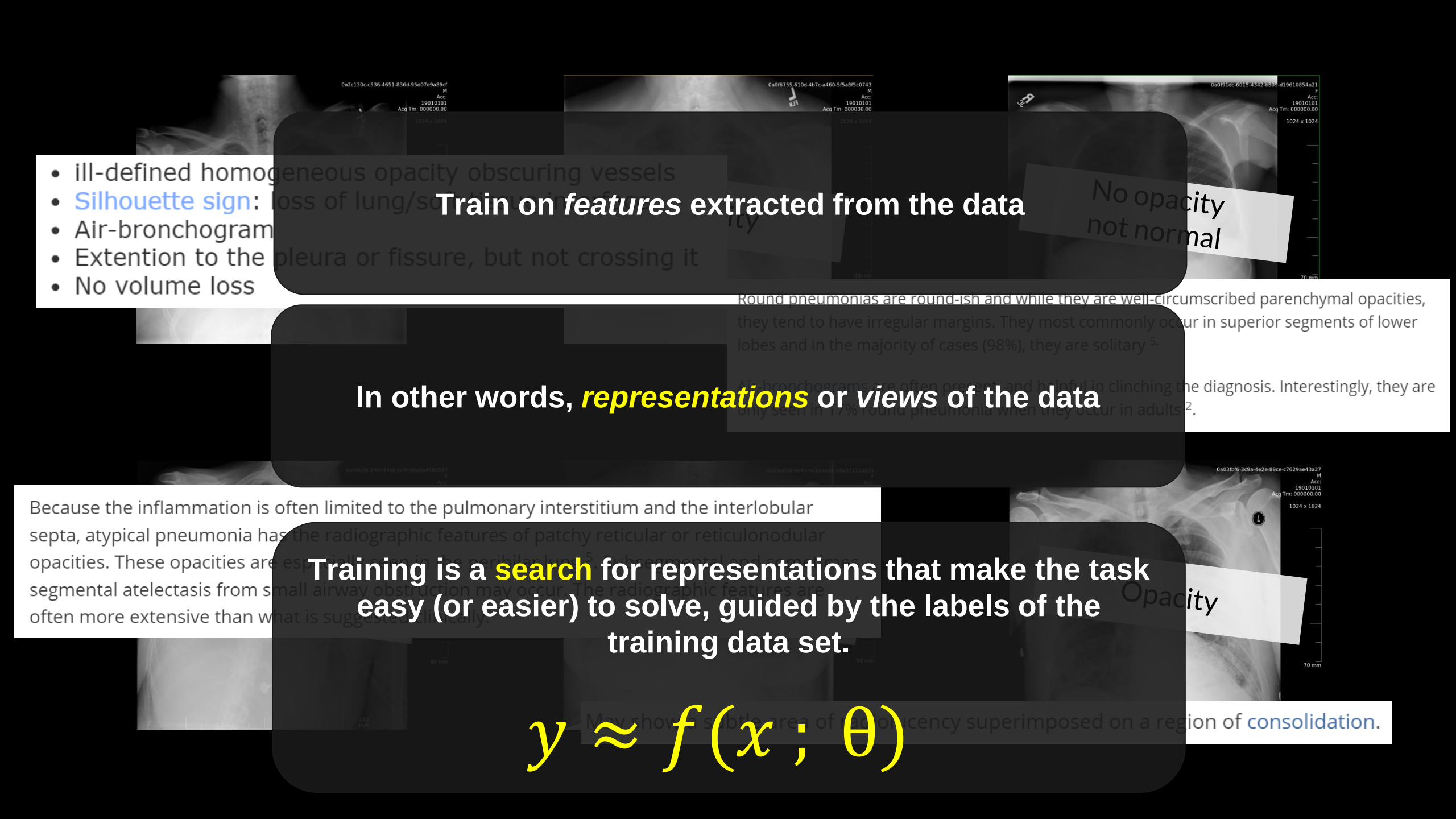
Note: good performance on the training data (X, y) is not what we're after. Any model of sufficiently high *capacity* can be made to *fit* a any training data set. I.e., produce correct outputs y for all the training instances in X.

What we care about is the performance on *new unseen* data X.

The **generalization** performance of the model.

To *simulate* the situation of having new data, one typically sets aside part of one's data set for estimating generalization performance.




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 user and a product

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

buy or not buy / leave or not leave (churn)

Models and parameters

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

model **parameters**

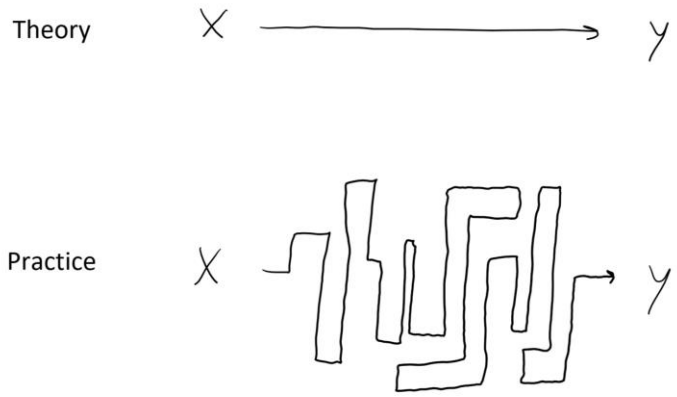
***Training* is a search for useful representations through a space defined by a family $\mathcal{F}$ of parametrized models**

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



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

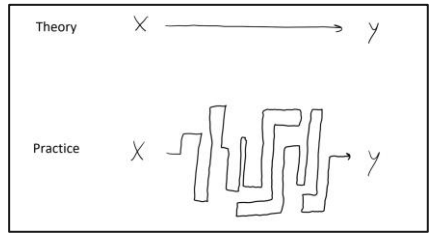
The family $\mathcal{F}$ of models considered, i.e., the *hypothesis space*, should be (i) efficiently searchable, (ii) leading to expressive models that can (ii) generalize beyond the training data distribution (i.e., work well on new data)

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

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

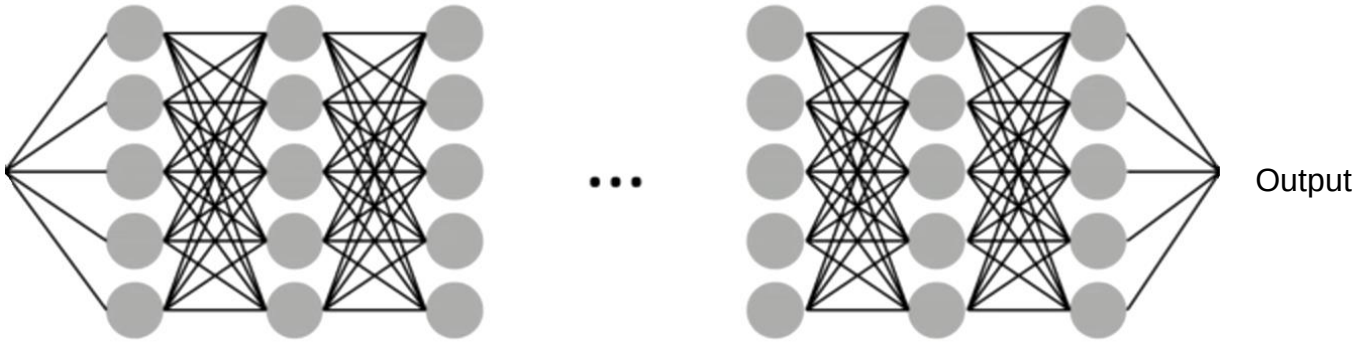
03 In practice: some models are better suited to some tasks than others.

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



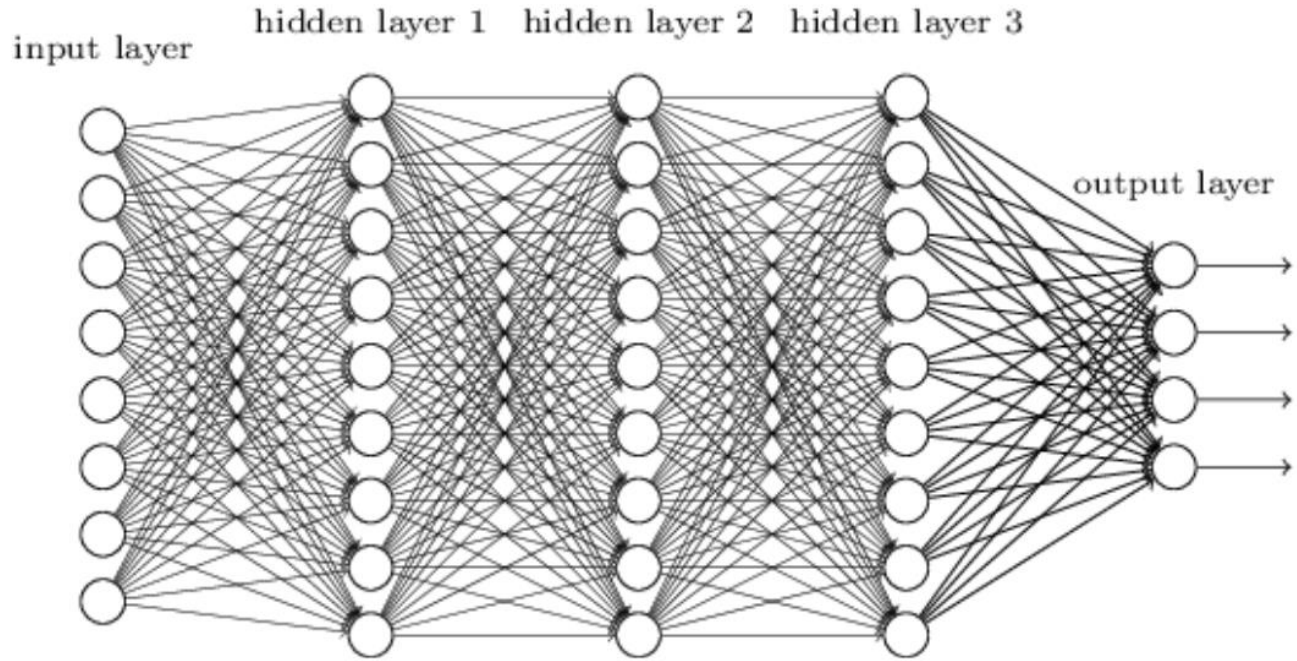
Deep learning

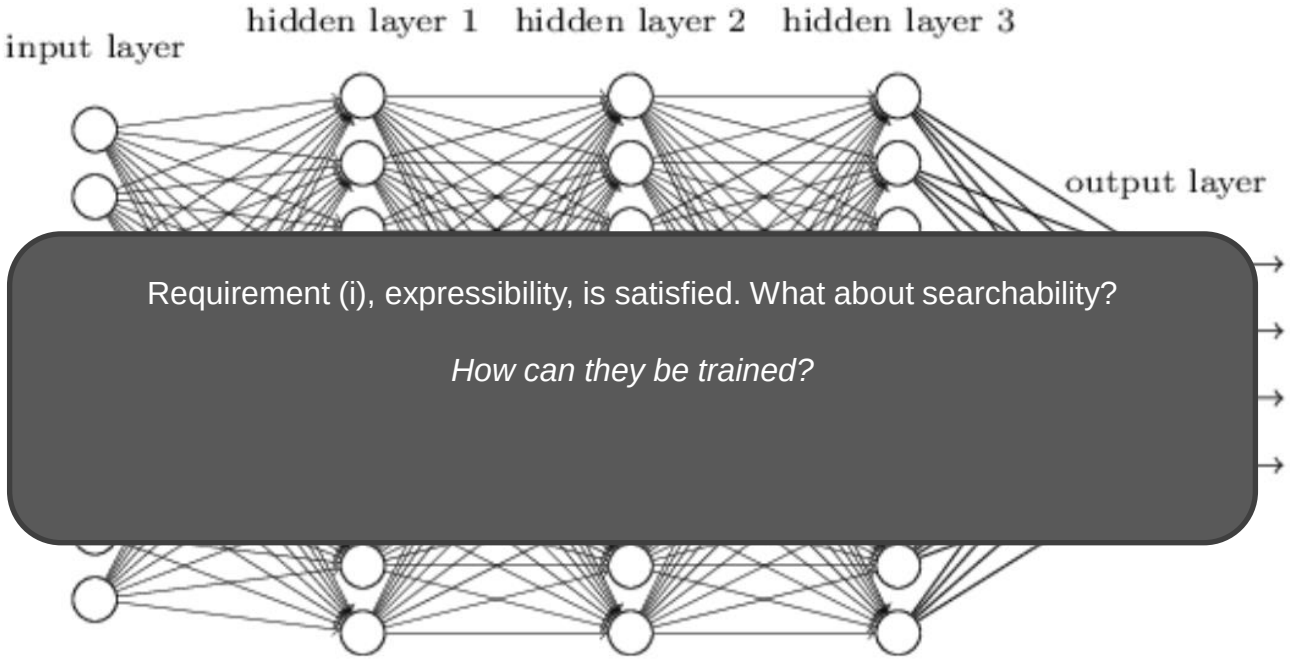
$$y \approx f(x; \theta)$$



Artificial neural networks

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


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



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

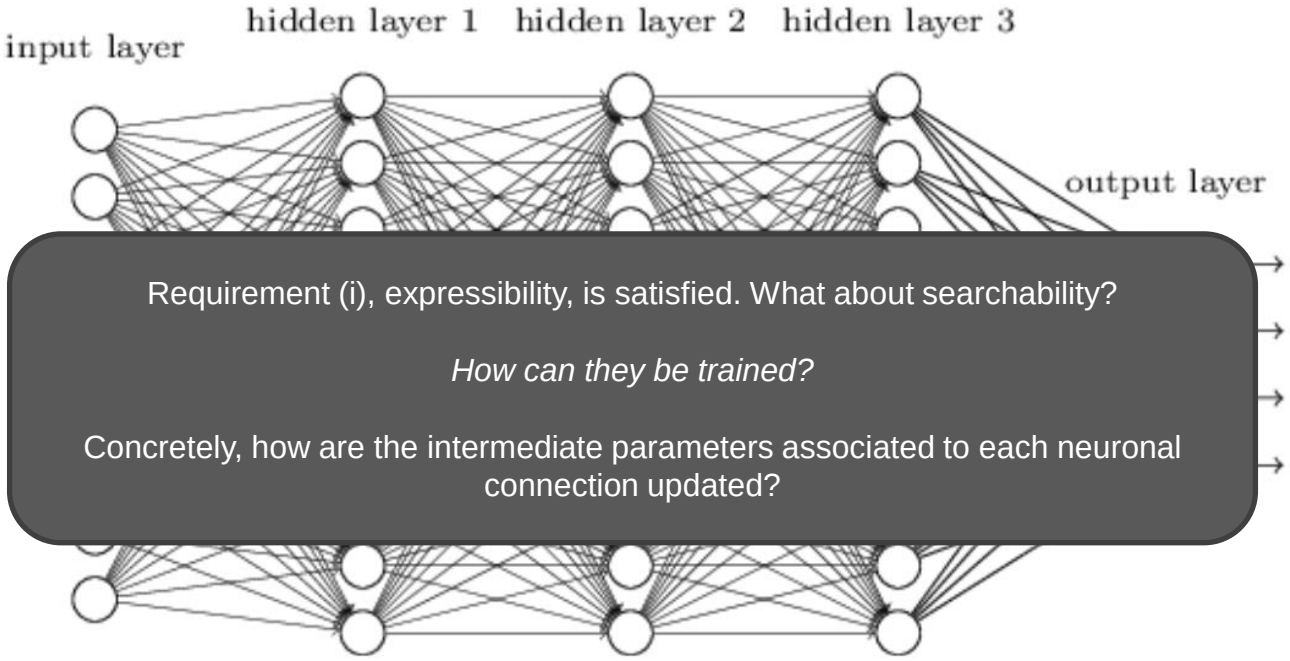
$$y = \sigma_{k+1}(W_{k+1}^T \sigma_k(W_k^T \left(\cdots \left(\sigma_1(W_1^T x) \right) \cdots \right) = f(x; \theta)$$

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 (i), expressibility, is satisfied. What about searchability?

How can they be trained?

Concretely, how are the intermediate 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

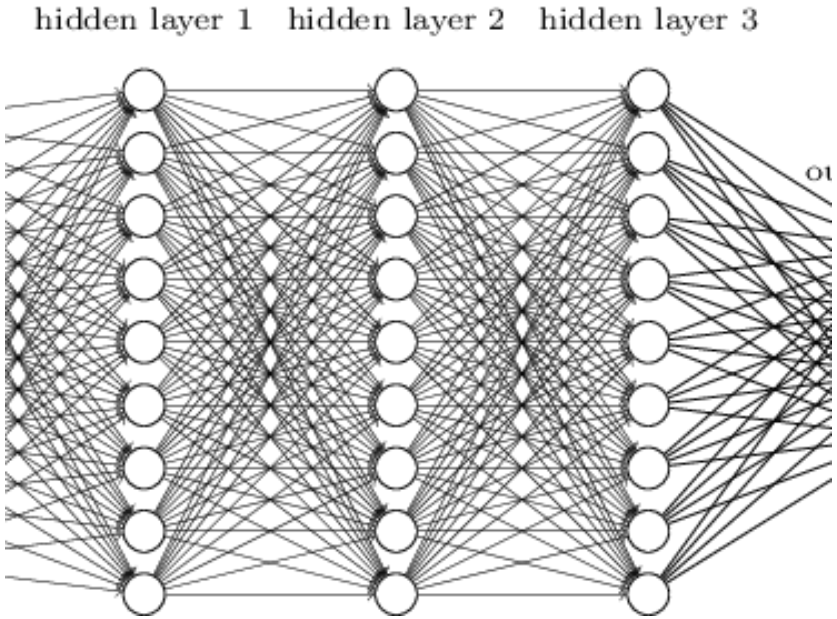
Each family member of F is complex transformation broken down into a composition of simpler ones, parametrized by the layer weights.

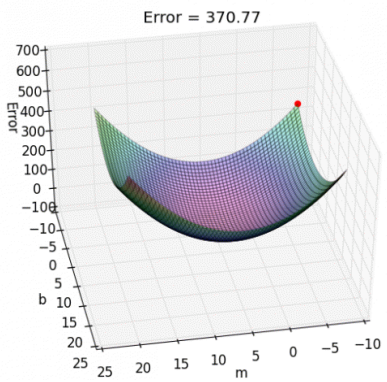
Three basic ingredients:

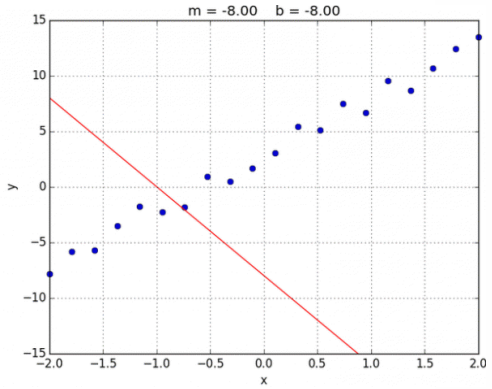
1. A **loss function**: Used to measure the network's performance on the training data. A **feedback signal** for the training process.

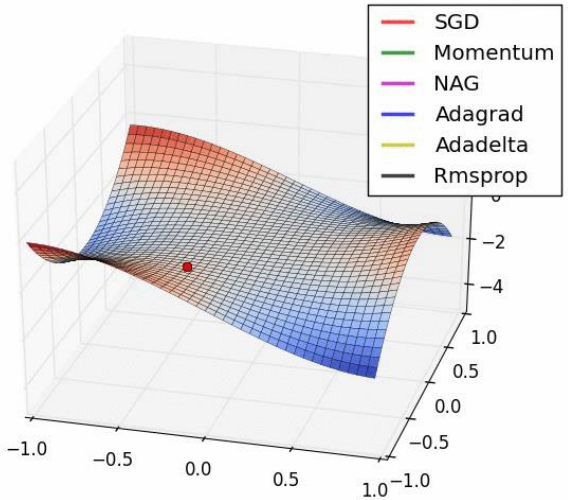
2. An **optimizer**: Used to update the *parameters* of the network to increase performance as measured by the loss function. **Gradient descent** and **backpropagation**.

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









1.Start with a neural network that has trainable parameters (weights)

2.Collect a *batch* of training data from `X` with the corresponding labels from `y`.

3.Put a batch `X` into the network and collect the output predictions `y_pred` at the output layer.

4.Use the loss function to measure the discrepancy between the known labels `y` and the predictions `y_pred` for this batch

5.Update all the weights to reduce the discrepancy for this batch:

1. Calculate each weight's contribution to the loss, in other words, find the gradients of the weights, by using backpropagation

2. Move each weight a little bit in the opposite direction of its gradient (gradient descent)
   ```
   weight = weight - learning_rate * gradient
   ```

6.Do this over and over for several epochs (one epoch is one pass through all the training data)

1.Start with a neural network that has trainable parameters (weights)

2.Collect a *batch* of training data from `X` with the corresponding labels from `y`.

3.Put a batch `X` into the network and collect the output predictions `y_pred` at the output layer.

4.Use the loss function to measure the discrepancy between the known labels `y` and the predictions `y_pred` for this batch

5.Update all the weights to reduce the discrepancy for this batch:

What is gradient descent?

1. Calculate each weight's contribution to the loss, in other words, find the gradients of the weights, by using backpropagation

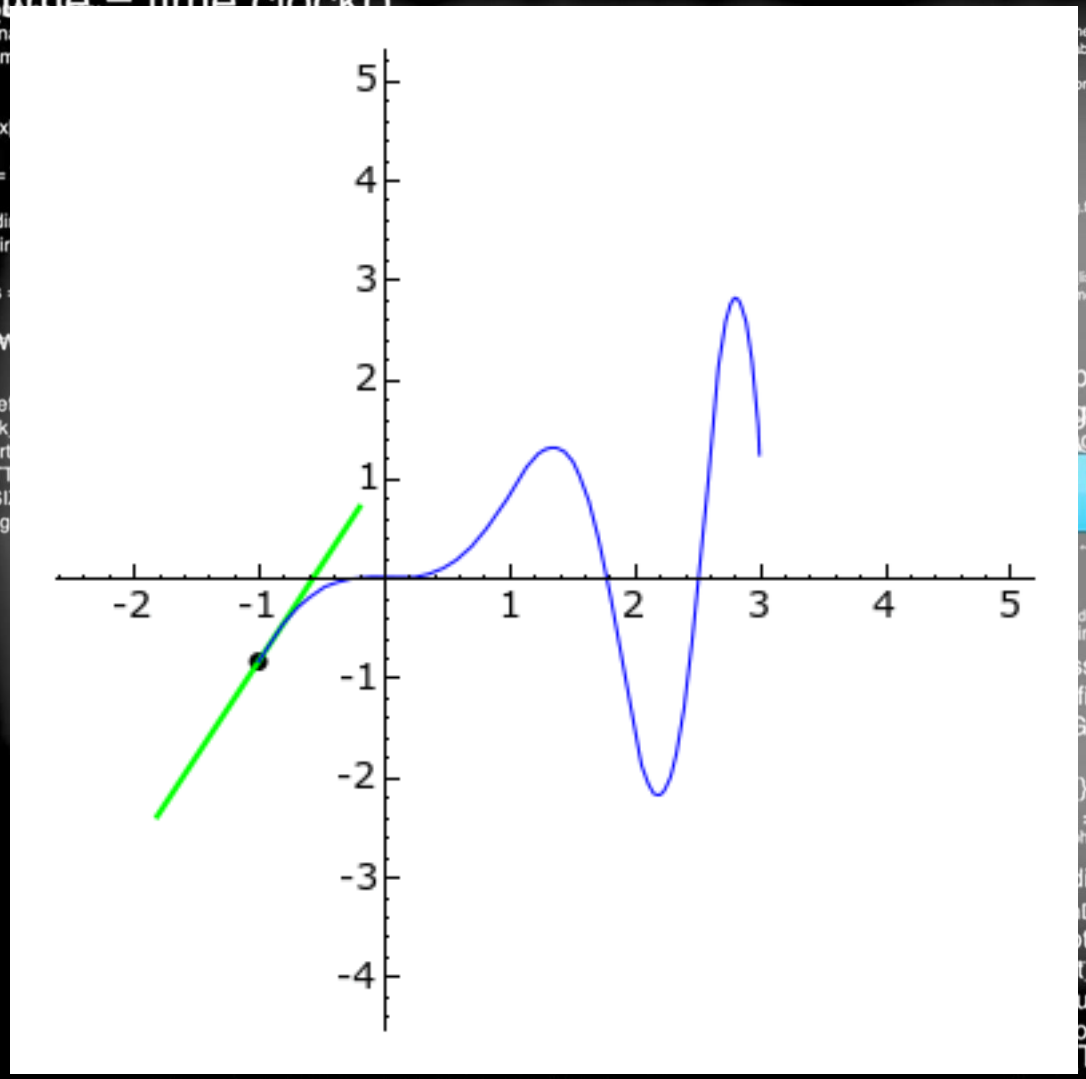
2. Move each weight a little bit in the opposite direction of its gradient (gradient descent)

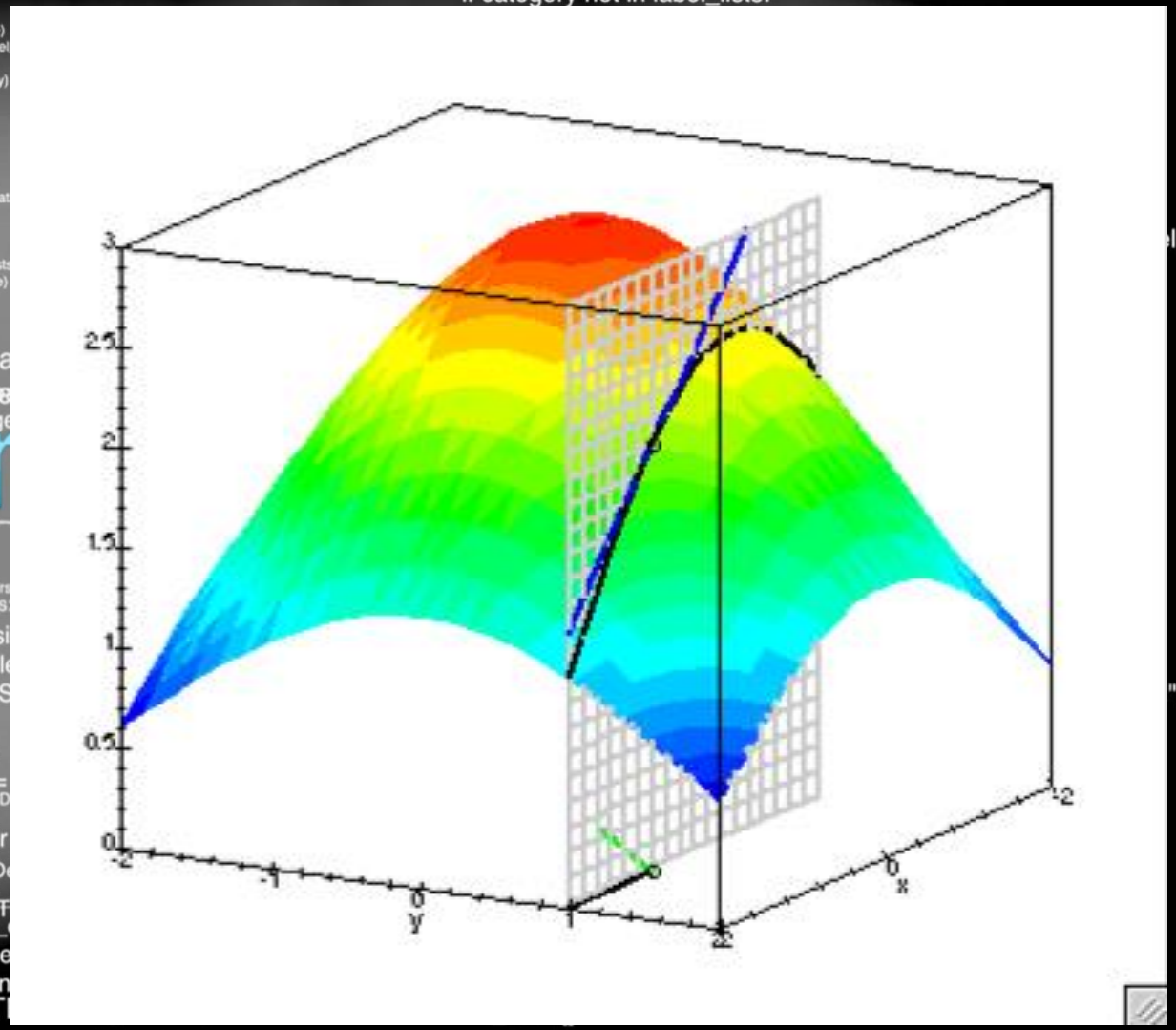
```
weight = weight - learning_rate * gradient
```

6.Do this over and over for several epochs (one epoch is one pass through all the training data)


best_params = None
best_validation_loss = numpy.inf
test_score = 0.
start_time = time.clock()
void QueueRunner::Run(Session* sess, const string& enqueue_op) {
bool decremented = false;
while (!should_stop_) {
std::vector<Tensor> outputs;
auto status = sess->Run({}, {}, {enqueue_op}, &outputs);
if (status.ok()) {
continue;
} else if (queue_closed_exception_types_.count(
static_cast<int>(status.code())) > 0) {
mutex_lock l(mu_);
runs_--;
decremented = true;
should_stop_ = true;
// If all enqueue ops have finished, run the close op.
if (runs_ == 0 && !close_op_name_.empty()) {
std::vector<Tensor> outputs;
auto s = sess->Run({}, {}, {close_op_name_}, &outputs);
if (!s.ok()) {
status_ = status;
ange_input_producer(epoch_size, shuffle=False).dequeue()
data, [0, i * num_steps], [batch_size, num_steps])

```
best_params = None
best_validation_loss = numpy.inf
test_score = 0.
start_time = time.clock()
```






```
RESIZED_INPUT_TENSOR_NAME]))
return sess.graph, bottleneck_tensor, jpeg_data_tensor, resized_input_tenso
best_params = None
best_validation_loss = numpy.inf
test_score = 0.
start_time = time.clock()
```

```
import glob
import hashlib
import os.path
import random
import re
```

```
dir_name = os.path.basen
with gfile r
graph_def
  graph_def.ParseFromStrin
  bottleneck_tensor, jpeg_data_tensor, resized_input_tensor = (
    tf.import_graph_def(g
```

```
.FastGFile(model_filename, 'rb') as f:
= tf.GraphDef()
    g(f.read())
raph_def, name='', return_elem
```


-2
0
x
-2

1.Start with a neural network that has trainal

We'll see some of this translated into code soon

2.Collect a *batch* of training data from `X` with the corresponding labels from `y`.

3.Put a ba

4.Use the tch

5.Update

1. C kpropagation
2. M

6.Do this over and over for several epochs (one epoch is one pass through all the training data)

A.S. Lundervold, Oct. 2021

Introduction

This notebook and the next is meant to give you a taste of neural networks and deep learning. Deep learning is a very large field and the taste will be have to be quite small. The goal is to perhaps satisfy some of your curiosity ("What is deep learning? How do you *do* deep learning") and point you towards ways to learn more about the topic.

In this notebook we'll see how to construct a basic deep neural network in PyTorch, and how to train it to perform a task.

GPUs: *Massively parallel linear algebra super-computers*

Developed for computer graphics

Good at exactly what's needed for neural networks!

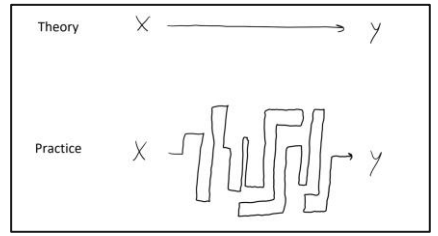
01 Machine learning: **function approximation** based on **training**

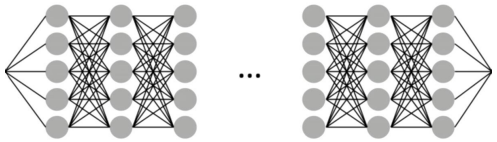
02 We pick the model family $\mathcal{F}$. The parameters Θ are found automatically through training

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.

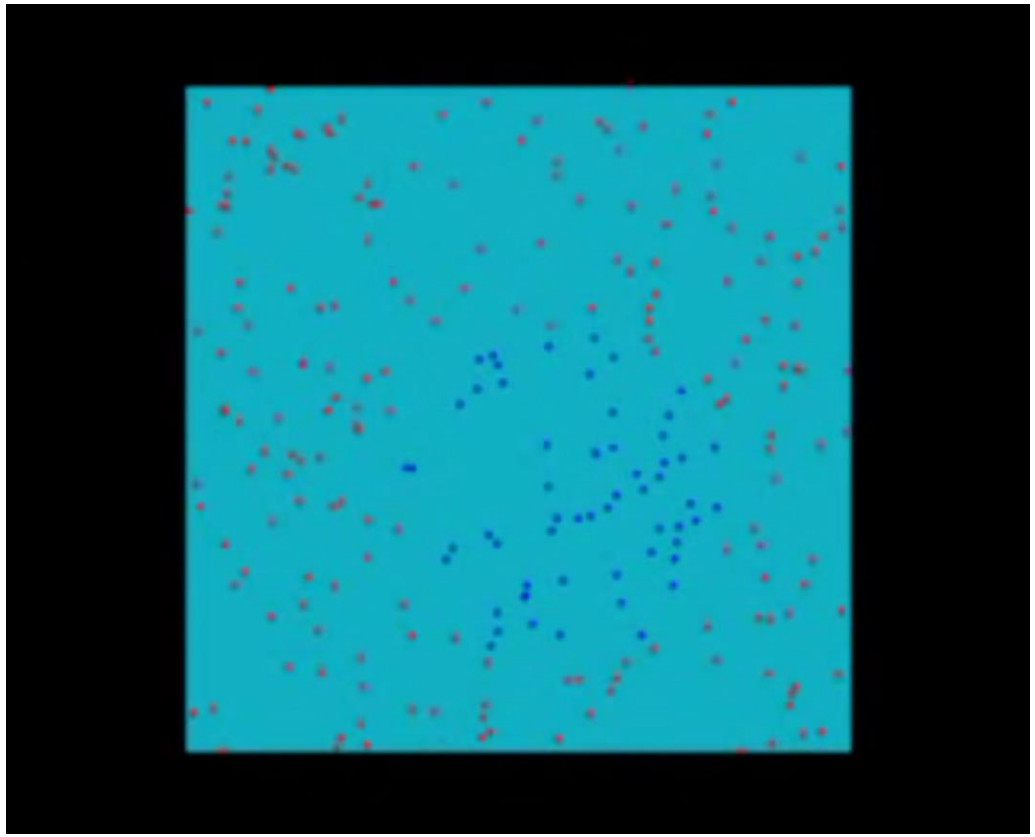
$$y \approx f(x;\theta)$$



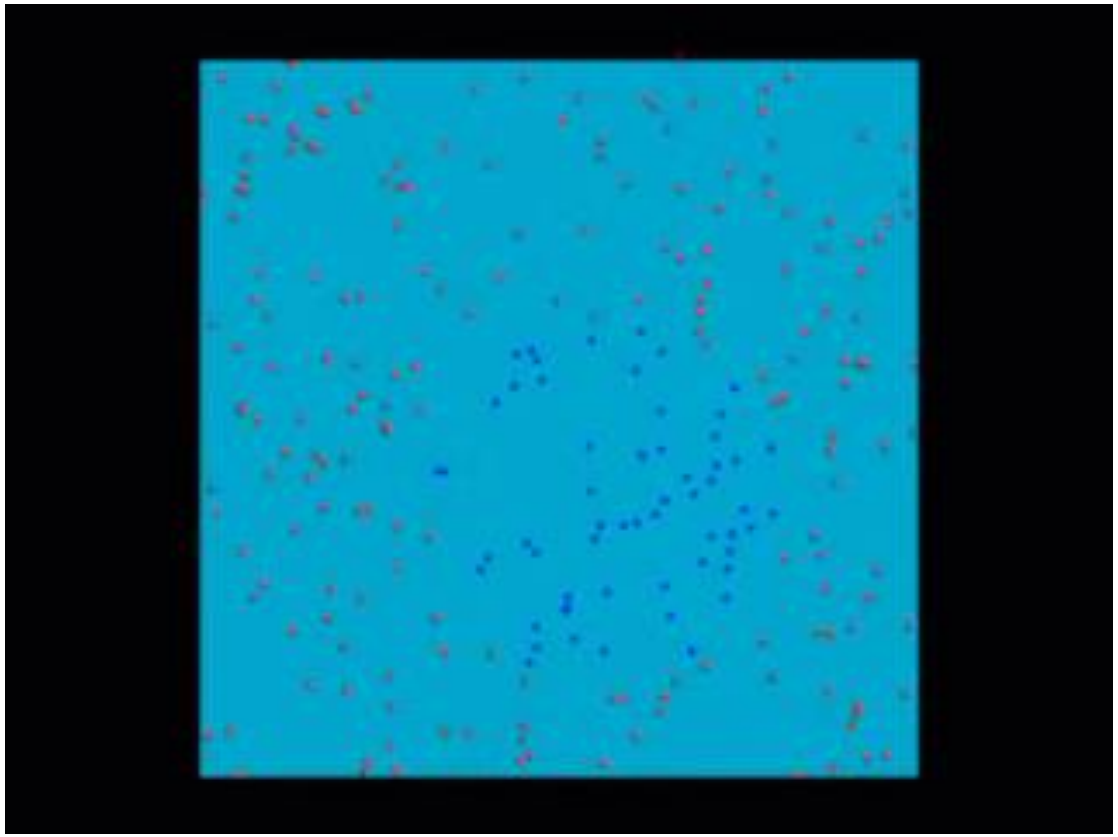


Representations and the power of transformations

Transformations

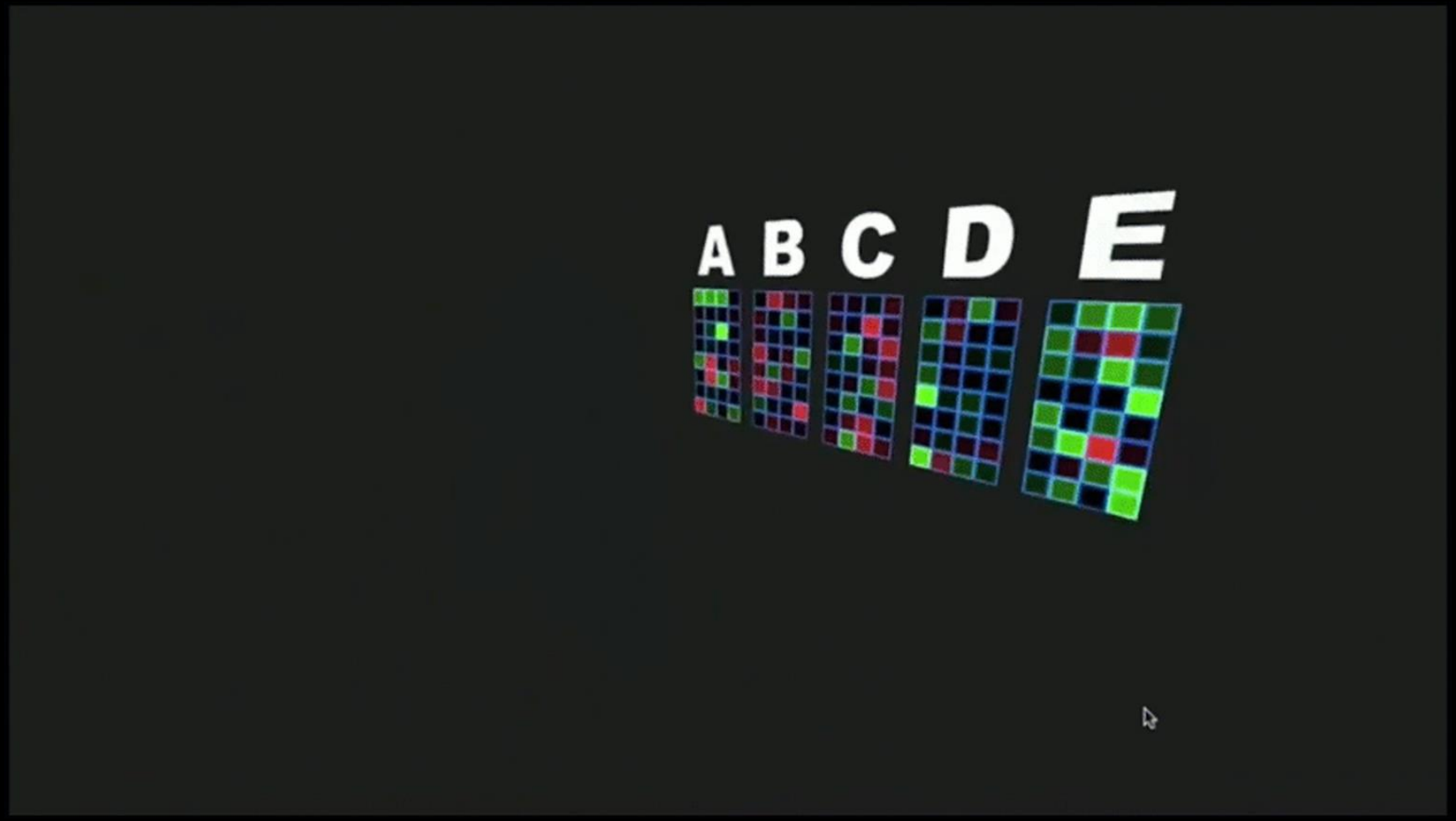


Transformation

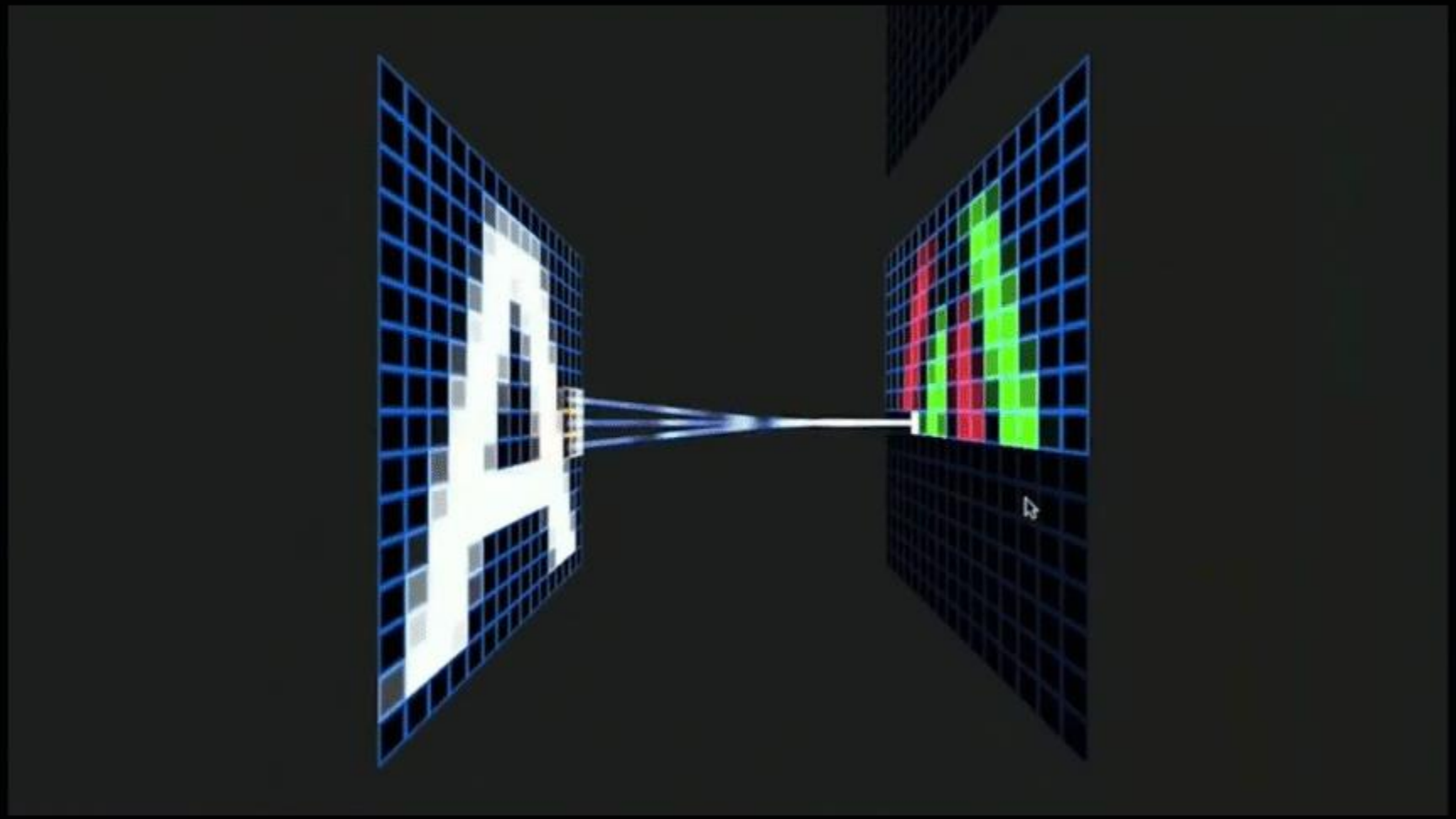


What transformations should be applied?

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



Source: Otavio Good



Source: Otavio Good

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

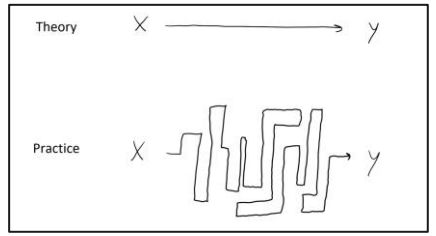
02 We pick the model family $\mathcal{F}$. The parameters Θ are found automatically through training

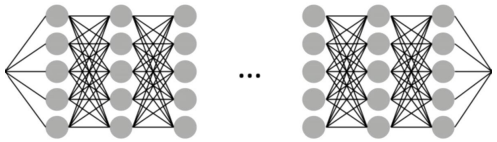
03 In practice: some models are better suited to some tasks than others.

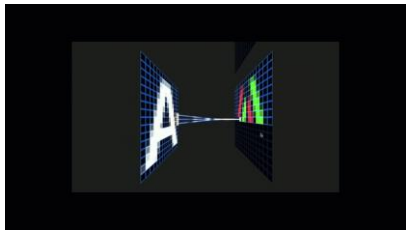
04 In deep learning: a particular choice of $\mathcal{F}$. A **composition** of simple functions, trained jointly, typically using GPUs or other accelerators.

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

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



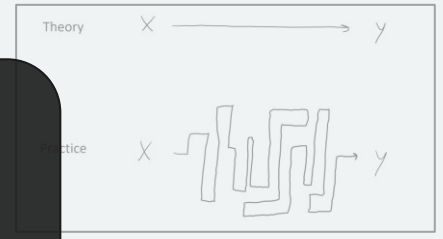




01 Machine learning: **function approximation** based on **training**

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

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



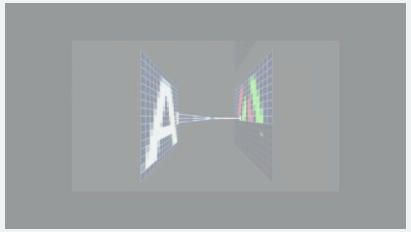
03 In practice: some models are better suited to some tasks than others.

Deep learning

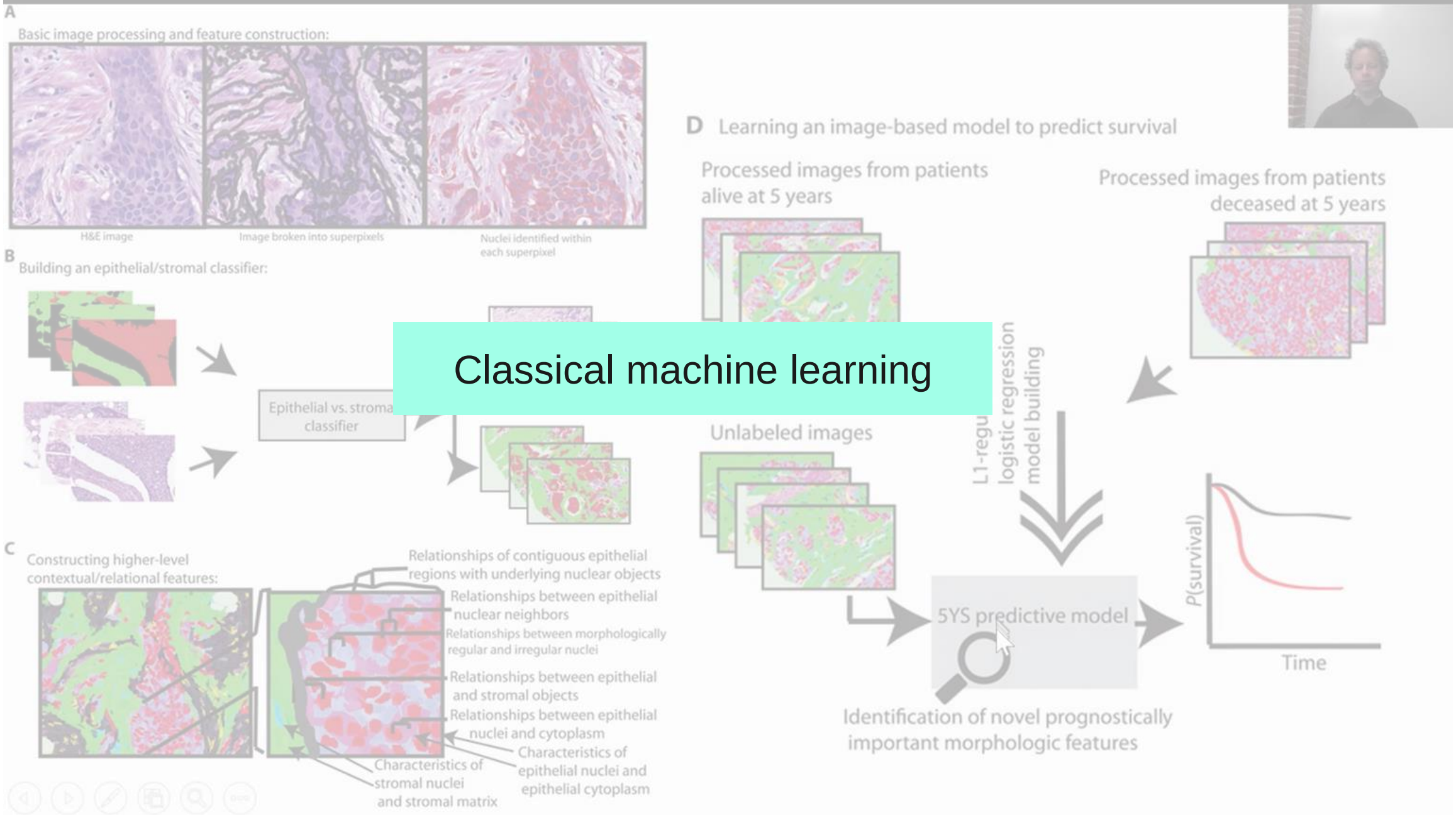
searching for good hierarchical representations

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

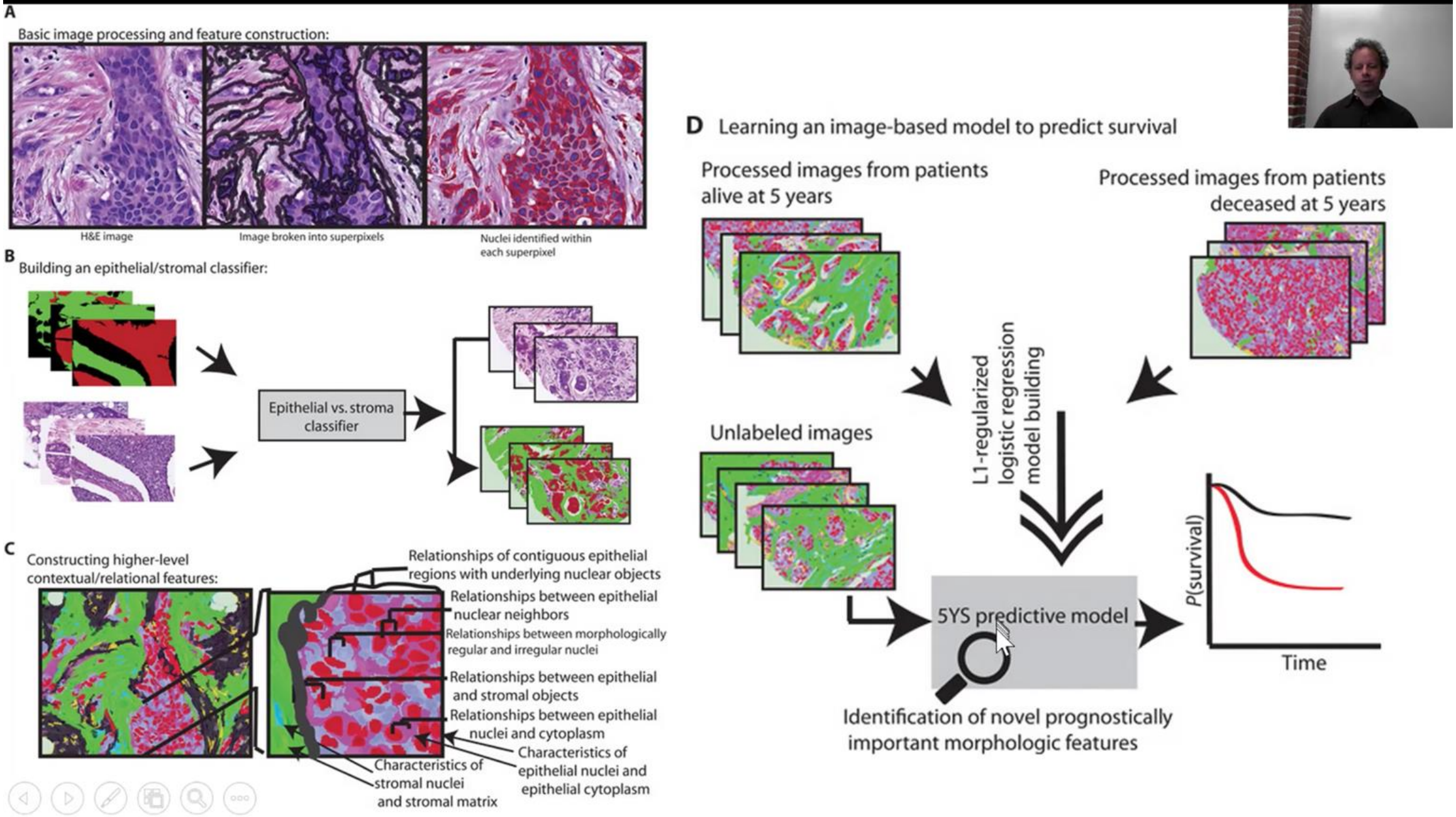
05 In deep learning: finding a **hierarchical set of transformations**, ending in a representation that makes the desired task easy to solve.



Deep learning versus “classical” machine learning



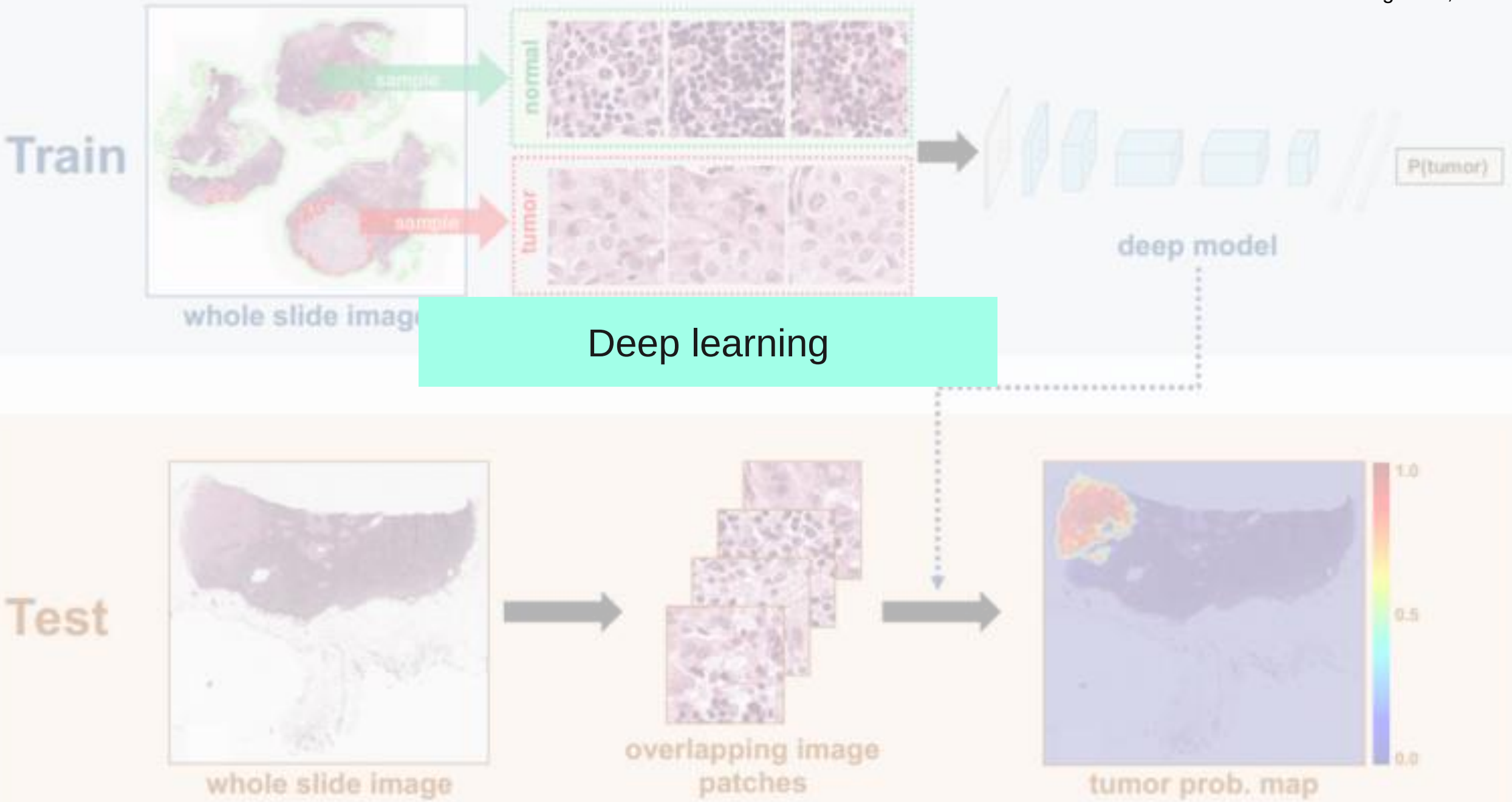
Classical machine learning



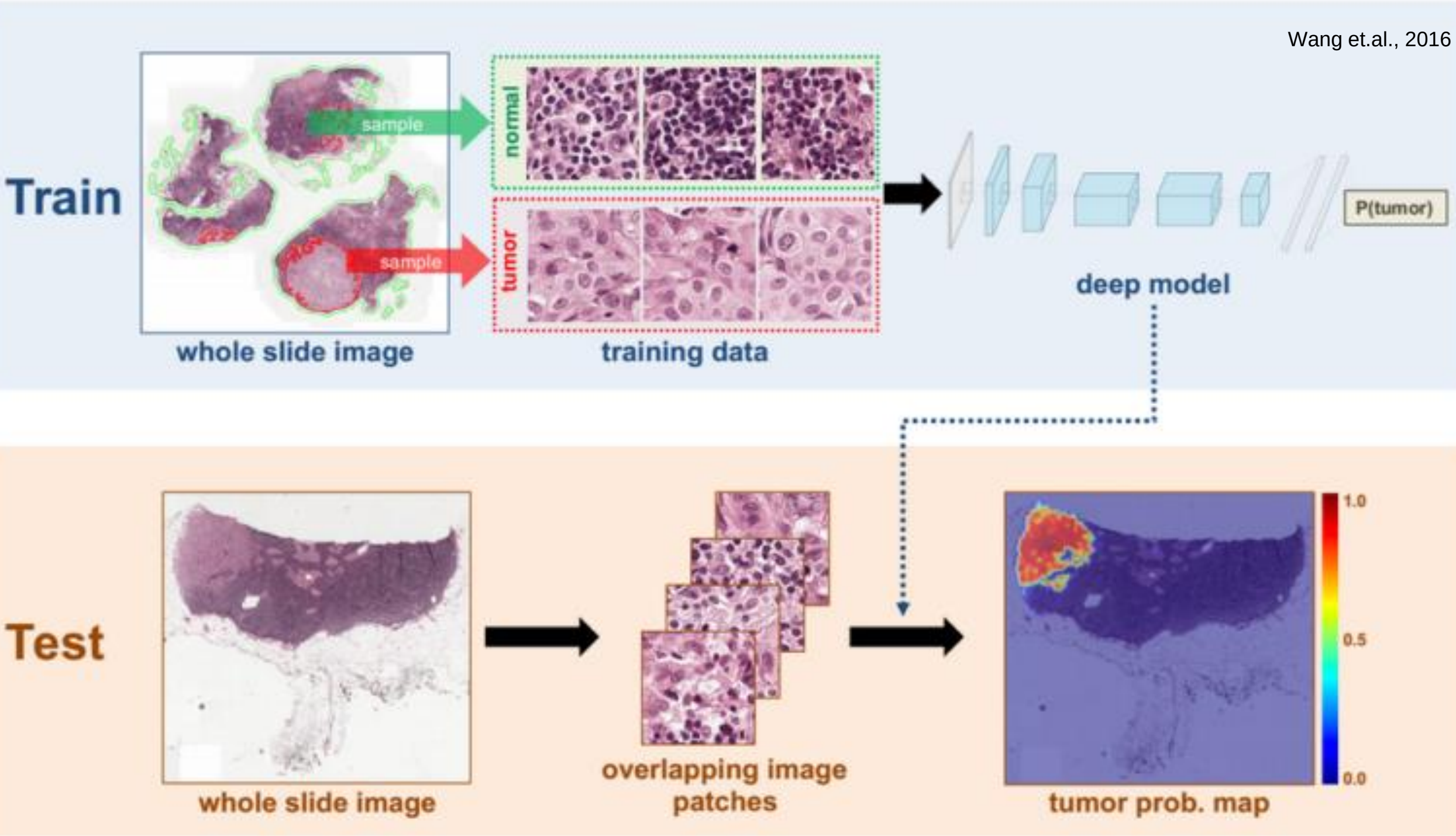
Computational pathologist (C-Path), Stanford University, 2011

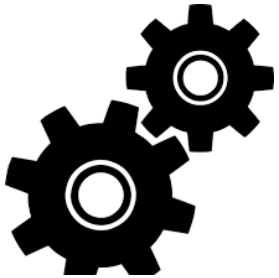
Fra Jeremy Howard

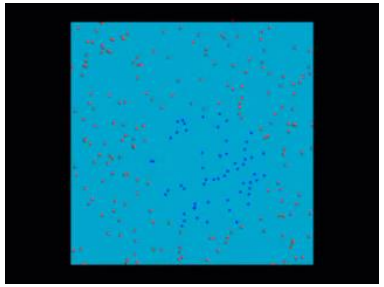
Wang et.al., 2016



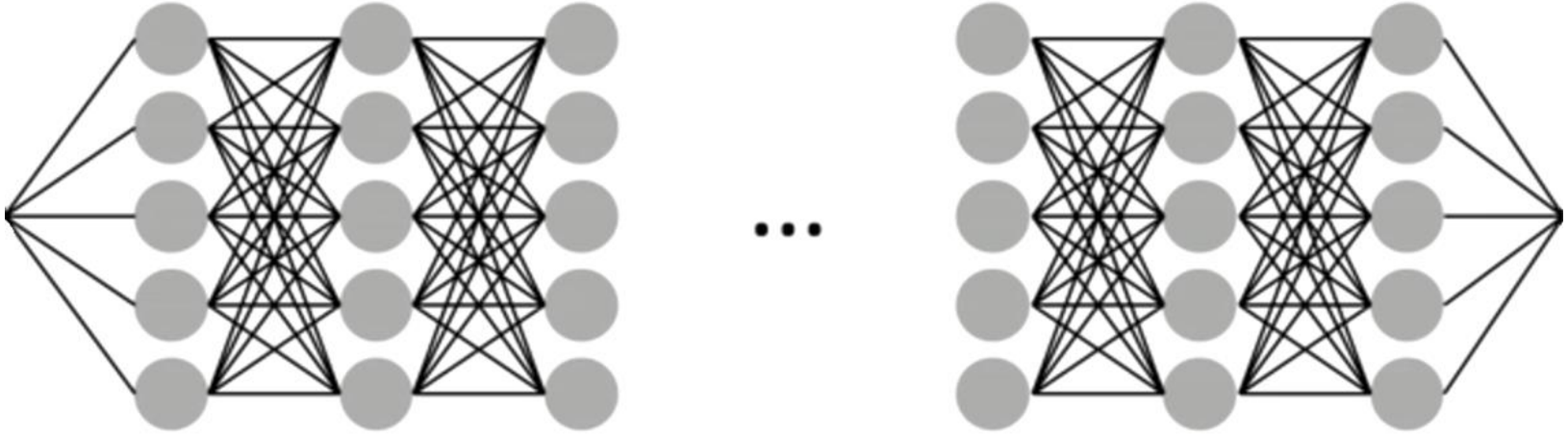
Wang et.al., 2016







Input
(ex image)



Output

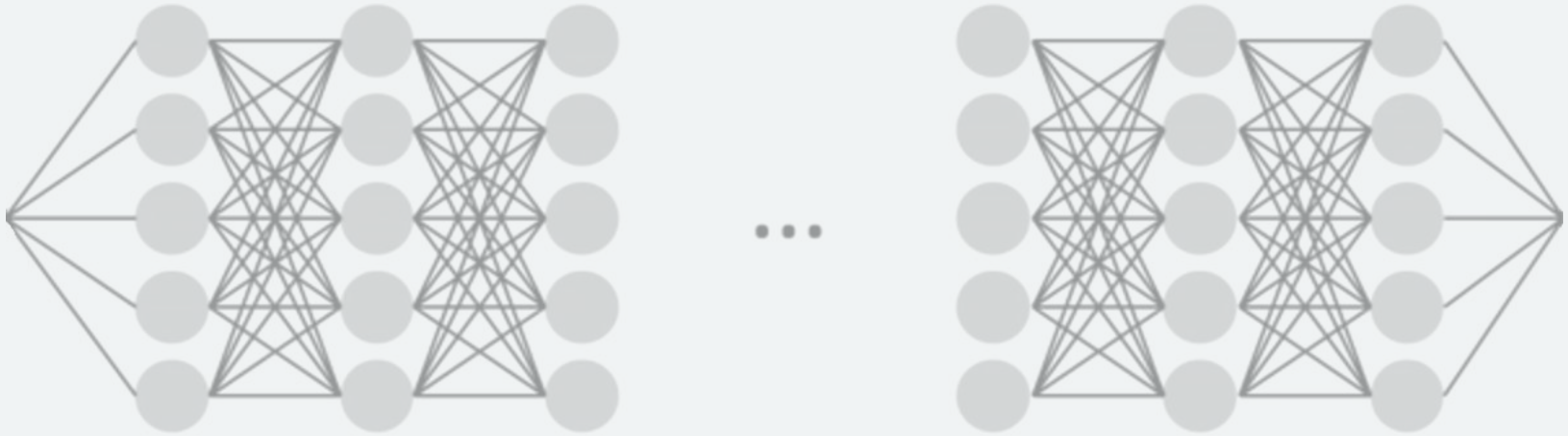
“Classical” machine learning

Deep learning
Artificial neural network

In practice, this is a bit of a lie

(albeit a useful lie, somewhat true)

Input
(ex image)



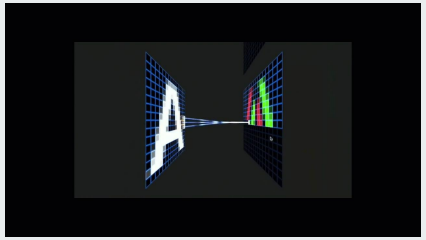
Output

Deep learning

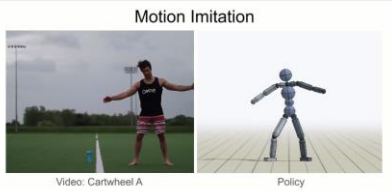
Artificial neural network

As we've seen: not just images; not just simple models

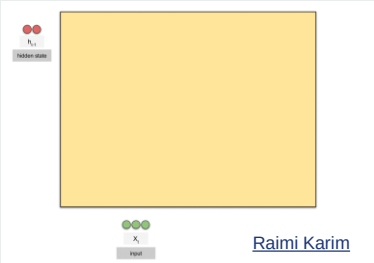
Convolutional neural networks



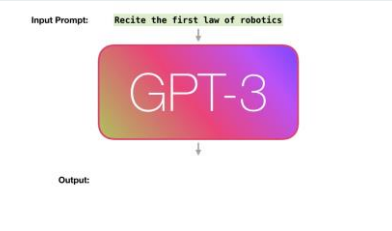
Deep reinforcement learning



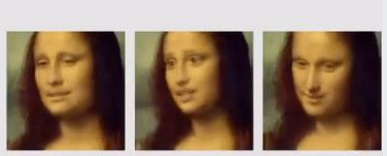
Recurrent neural networks



Transformers

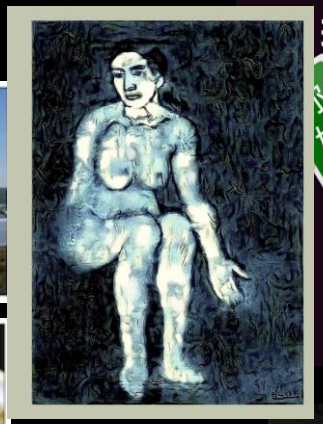


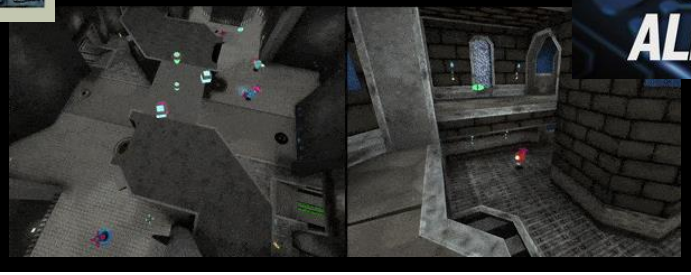
Generative adversarial networks

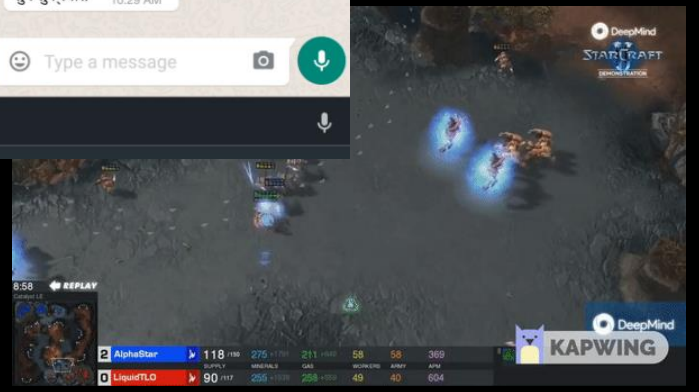


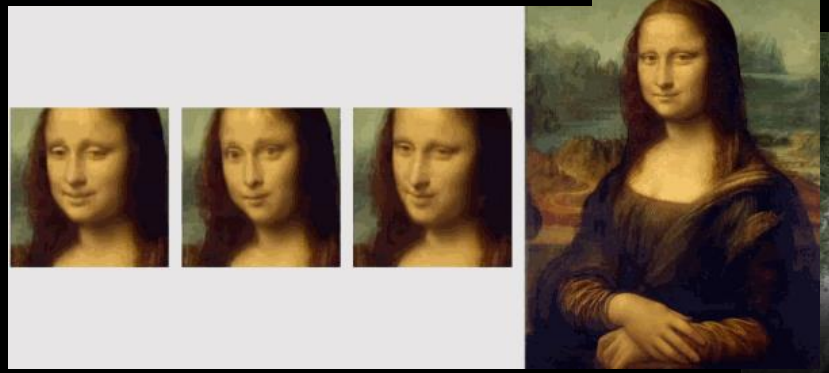
...

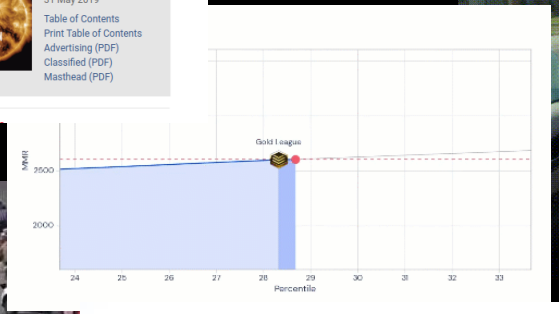
(attention, embeddings, one-shot learning, zero-shot learning, model uncertainty, federated learning, privacy, efficient training, model compression, ….)

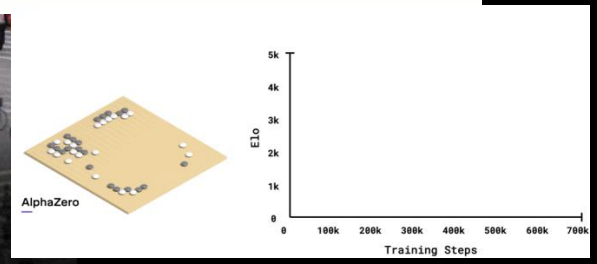












Attributions for illustrations used in the presentation are available upon request

Skin cancer can't hide from deep-learning diagnostics

May 21, 2019 | Dave Pearson | Diagnostics

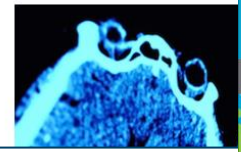
PhoneMD:
Learning to Diagnose Parkinson's Disease from Smartphone Data

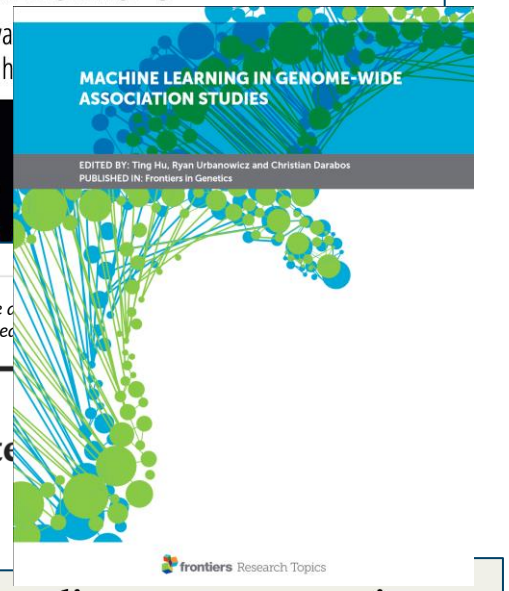
Patrick Schwab

Walter Karlen
Institute of Robotics and Intelligent Systems

Deep Learning Tool Identifies Disease Markers of Alzheimer's

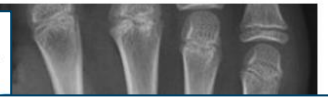
A deep learning tool wa
Alzheimer's disease in h





Can Detect Wrist Fractures

he accuracy of wrist fracture diagnosis.



Deep Learning Model Can Predict Br
Five Years in Advance

The system will help develop individual risk management plans.

By Jessica Miley
May 24th, 2019

Deep learning improves detection of polyps during colonoscopy

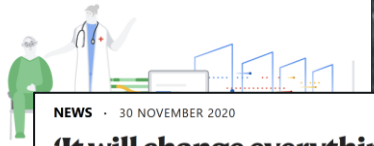
November 07, 2018 | Matt O'C

Article | Open Access | Published: 08 May 2018

Scalable and accurate deep learning with electronic health records

Alvin Rajkomar, Eyal Oren, [...] Jeffrey Dean

npj Digital Medicine 1, Article number: 18 (2018) | C



nature > articles > article

nature

*maintained non-inferior performance
reduced the workload of the second re
by 88%*

Article | Published: 01 January 2020

International evaluation of an AI syste
for breast cancer screening

Scott Mayer McKinney, Marcin Sieniek, [...] Shravya Shetty

Machine learning as a supportive tool to recogni
arrest in emergency calls

Stig Nikolaj Blomberg[a,b,*], Fredrik Folke[a,b,c], Annette Kjær Ersbøll[d], Helle Collatz Christensen[a],

NEWS · 30 NOVEMBER 2020

'It will change everything': DeepMind's AI makes gigantic leap in solving protein structures

Google's deep-learning program for determining the 3D shapes of proteins stands to transform biology, say scientists.

NOVEMBER 15, 2017

Stanford algorithm can diagnose pneumonia better than radiologists

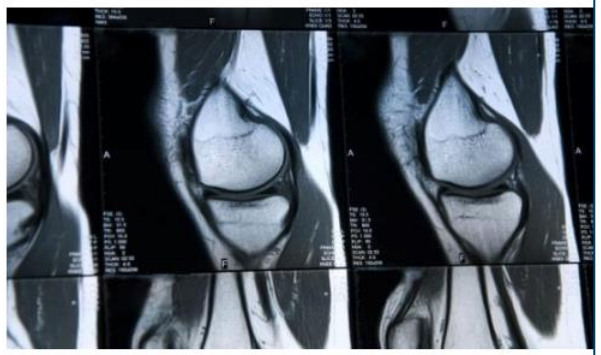
Stanford researchers have developed a deep learning algorithm that evaluates chest X-rays for signs of disease. In just over a month of development, their algorithm outperformed expert radiologists at diagnosing pneumonia.

Deep learning predicts OCT measures of diabetic macular thickening

By Steve Lenier
May 21, 2019

Deep learning detects ACL tears on MRI similar to radi

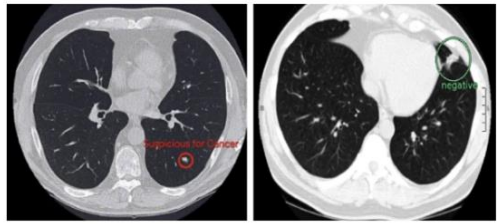
May 17, 2019 | Matt O'Connor | Artificial Intelligence



AI

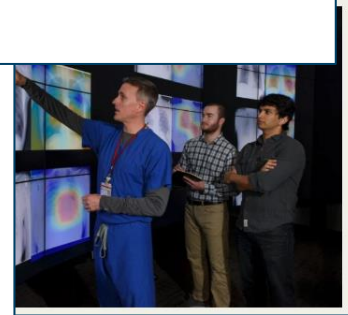
Google's lung cancer detection AI outperforms 6 human radiologists

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

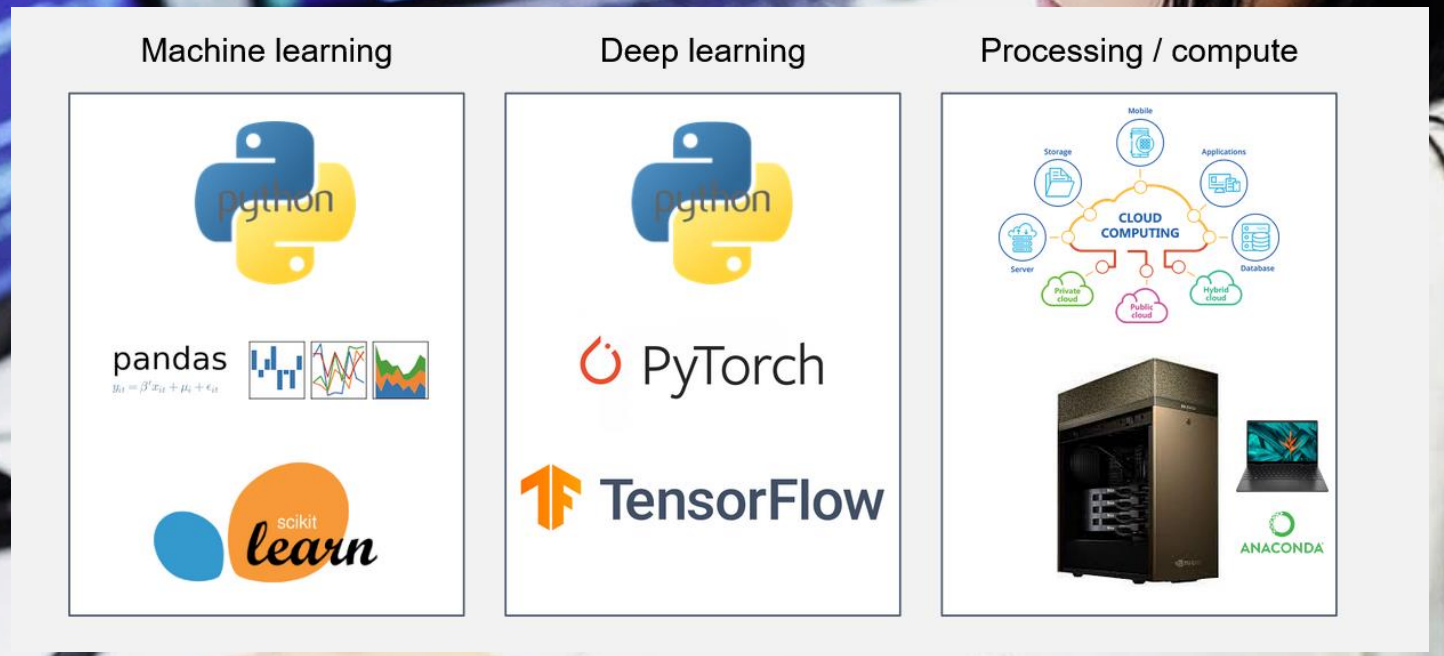


lecules Designed by Artificial Intelligence May Accelerate Drug Discovery

TOPICS: Artificial Intelligence Biotechnology InSilico Medicine



Hands-on: how do you *do* deep learning?

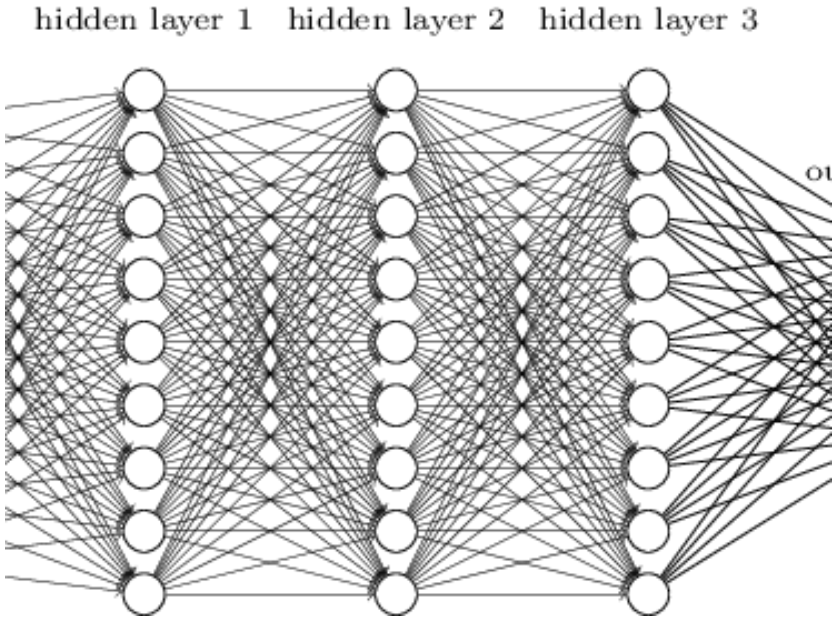


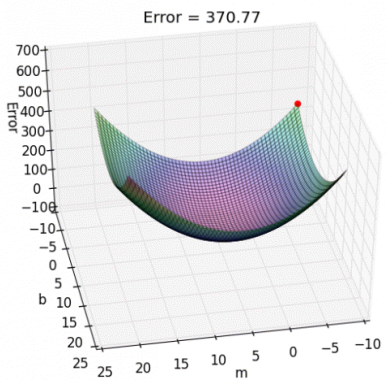
Three basic ingredients:

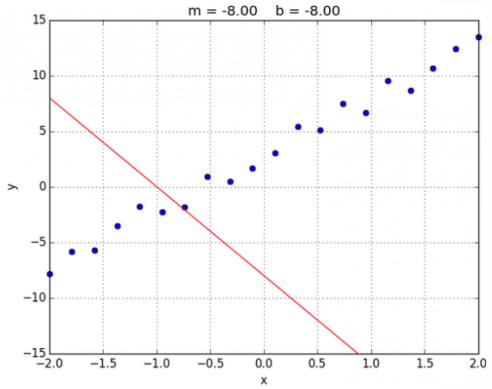
1. A **loss function**: Used to measure the network's performance on the training data. A **feedback signal** for the training process.

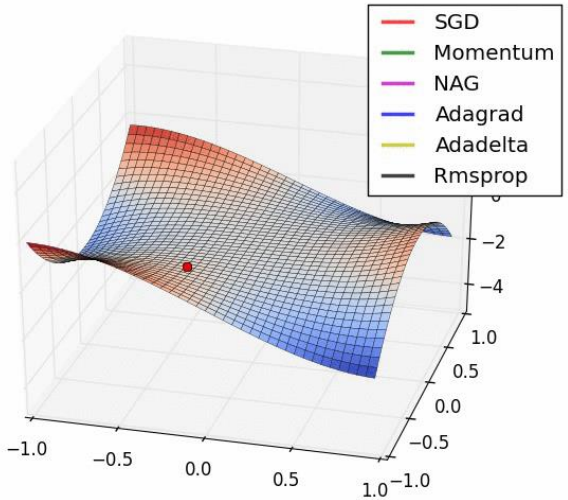
2. An **optimizer**: Used to update the *parameters* of the network to increase performance as measured by the loss function. **Gradient descent** and **backpropagation**.

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







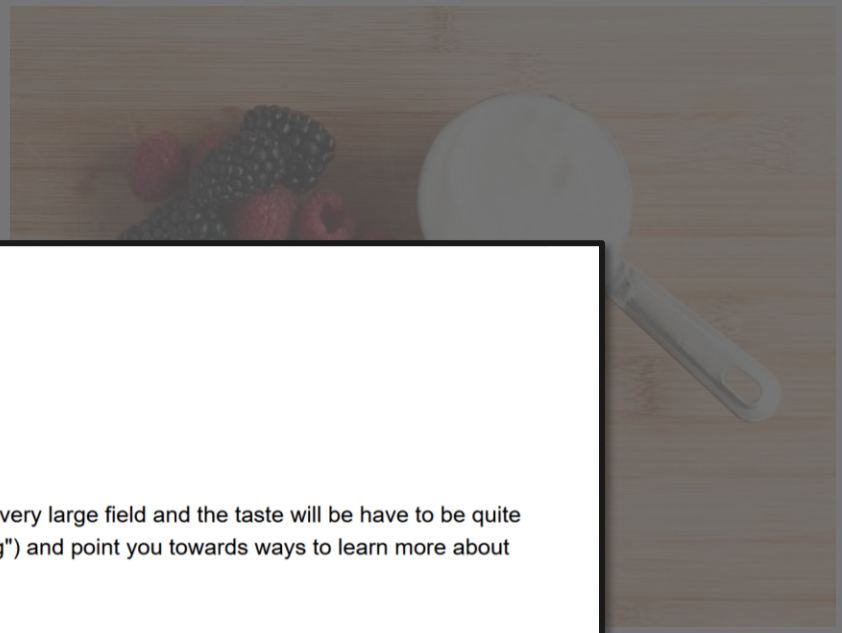


Three basic ingredients:

1. A **loss function**: Used to measure the network's perfor… on the traini… data. A **feedback signal** for the trai…

2. An … to ir… **Gra…**

3. One … exa…



A.S. Lundervold, Oct. 2021

Introduction

This notebook and the next is meant to give you a taste of neural networks and deep learning. Deep learning is a very large field and the taste will be have to be quite small. The goal is to perhaps satisfy some of your curiosity ("What is deep learning? How do you *do* deep learning") and point you towards ways to learn more about the topic.

In this notebook we'll see how to construct a basic deep neural network in PyTorch, and how to train it to perform a task.

PYTORCH