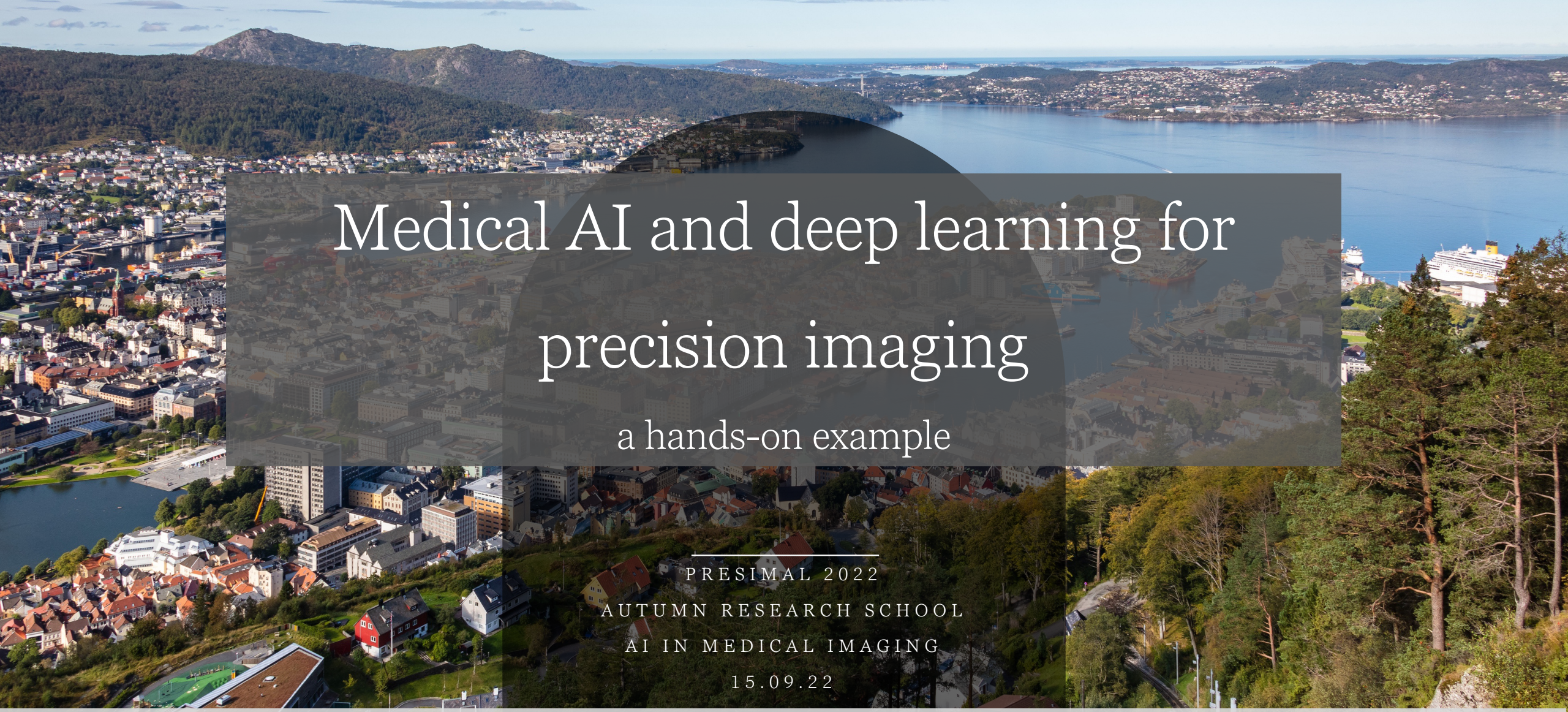


Medical AI and deep learning for precision imaging

a hands-on example

PRESIMAL 2022

AUTUMN RESEARCH SCHOOL

AI IN MEDICAL IMAGING

15.09.22

Western Norway University of Applied Sciences

MMIV

Haukeland University Hospital

Alexander Selvikvåg Lundervold

allu@hvl.no; lundervold.net

Arvid Lundervold

arvid.Lundervold@uib.no

“myScript.sh”

Hauke Bartsch

PhD. Ass.prof at University of Bergen. Senior Researcher at HUS.

Search or jump to... / Pull requests Issues Marketplace Explore

MMIV-ML / presimal2022 Public

Edit Pins | Unwatch 2 | Fork 1 | Star 3

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

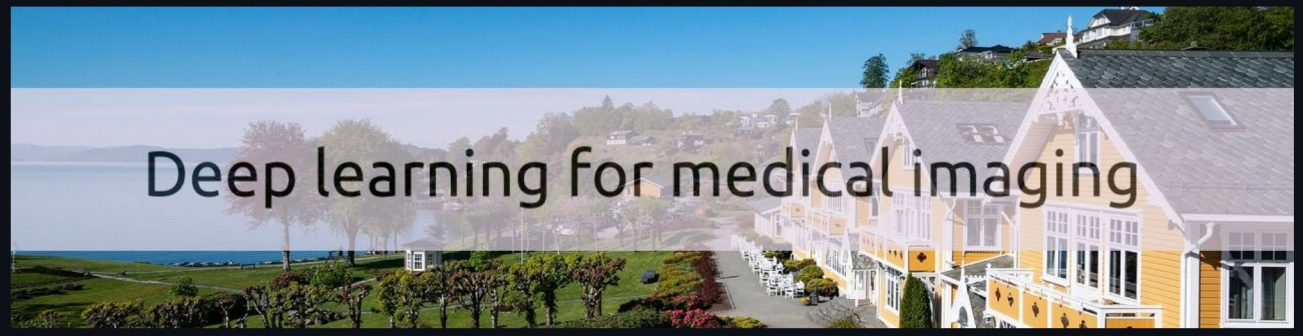
master | 2 branches | 0 tags | Go to file | Add file | Code

alu042 upd 13c7b9d 5 hours ago 21 commits

assets	upd	5 hours ago
notebooks	upd	5 hours ago
.gitignore	ignores	2 days ago
LICENSE	ignore & license added	10 days ago
README.md	Update README.md	23 hours ago
environment.yml	conda env	7 days ago

README.md

PRESIMAL 2022 workshop



About

Material from a workshop in the PRESIMAL Autumn Research School 2022

Readme
GPL-3.0 license
3 stars
2 watching
1 fork

Releases

No releases published
Create a new release

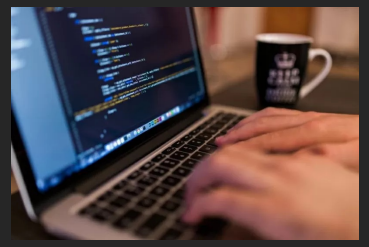
Packages

No packages published
Publish your first package

Languages

Jupyter Notebook 100.0%

Plan



Hands-on

What *is* deep learning?

How do you *do* deep learning?

Some perspectives, opportunities and challenges

Deep learning

searching for good hierarchical geometric representations

Deep learning

***searching** for good hierarchical geometric representations*

Deep learning

searching for good hierarchical geometric representations

Deep learning

searching for good hierarchical geometric representations

Deep learning

searching for good hierarchical geometric representations

01

02

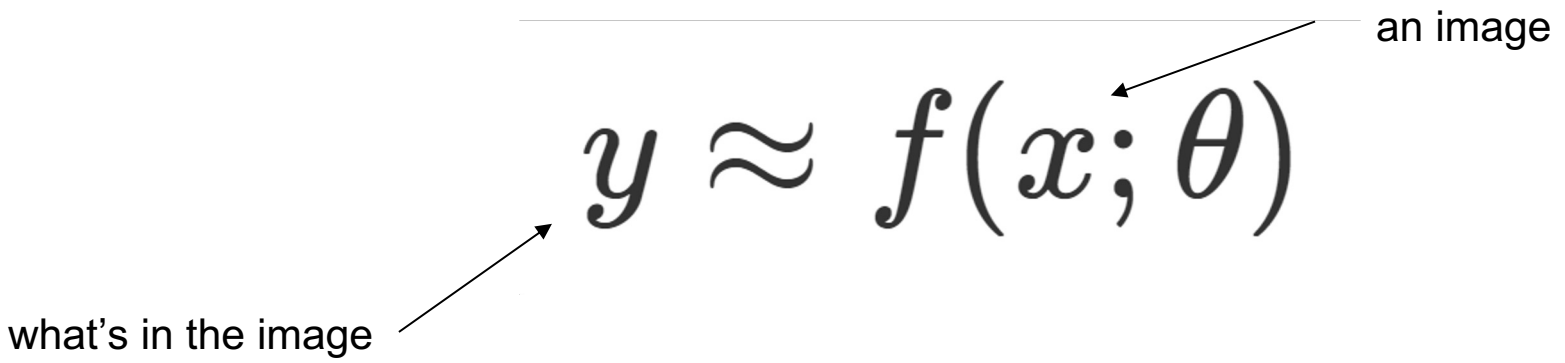
03

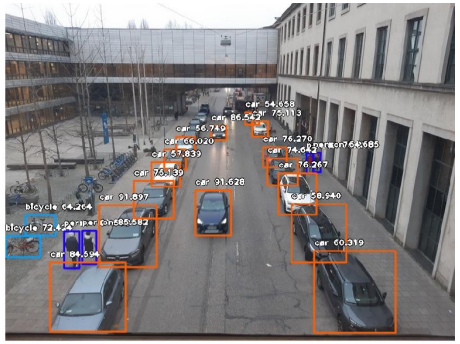
04

05

Function approximation

Function approximation





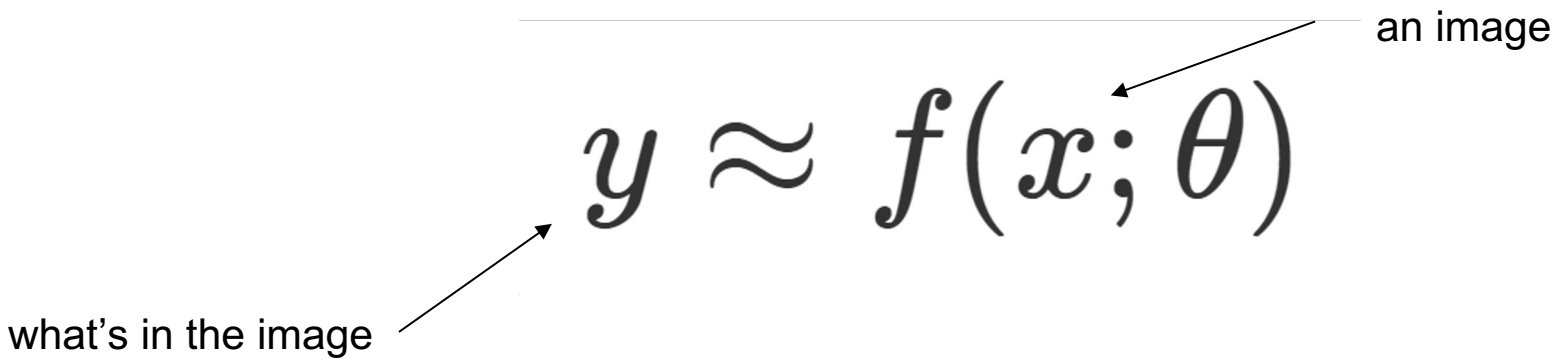
Function approximation

an image

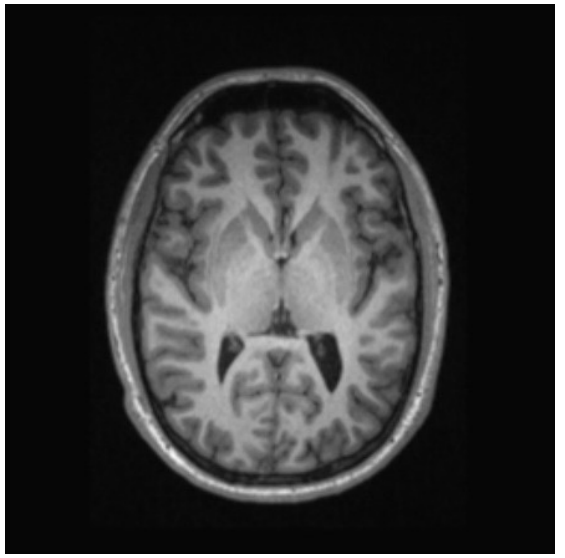
$$y \approx f(x;\theta)$$

what's in the image

Function approximation



AGE

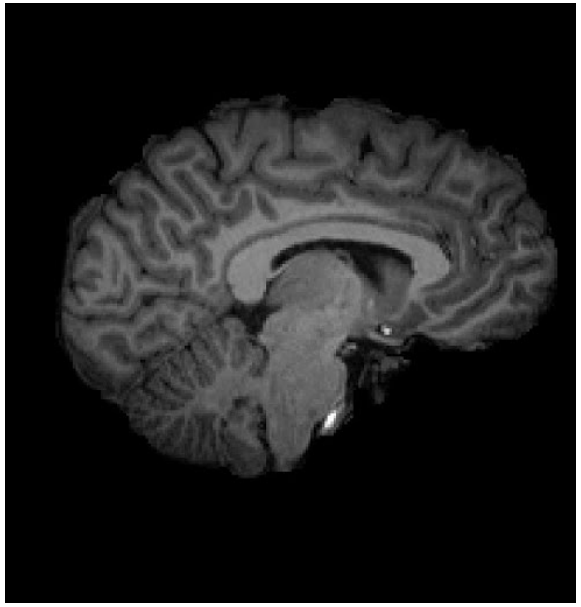


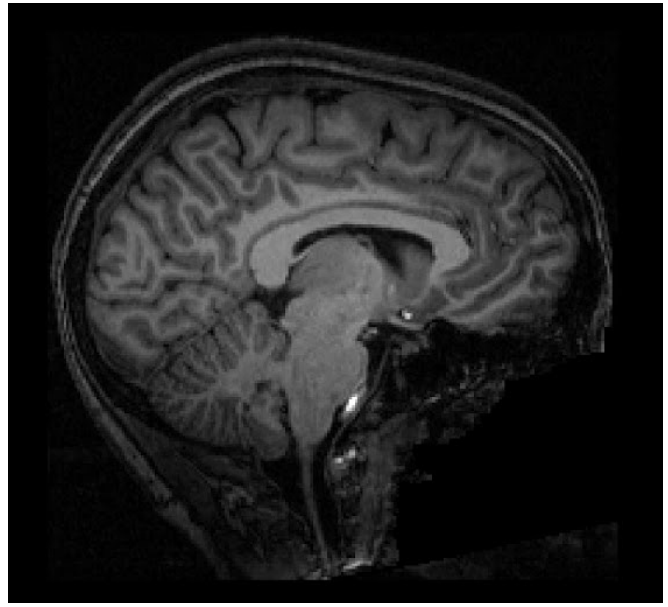
Function approximation

an MRI image

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

”Brain age”



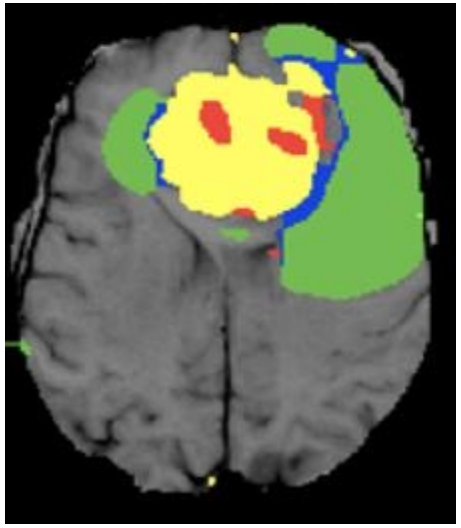


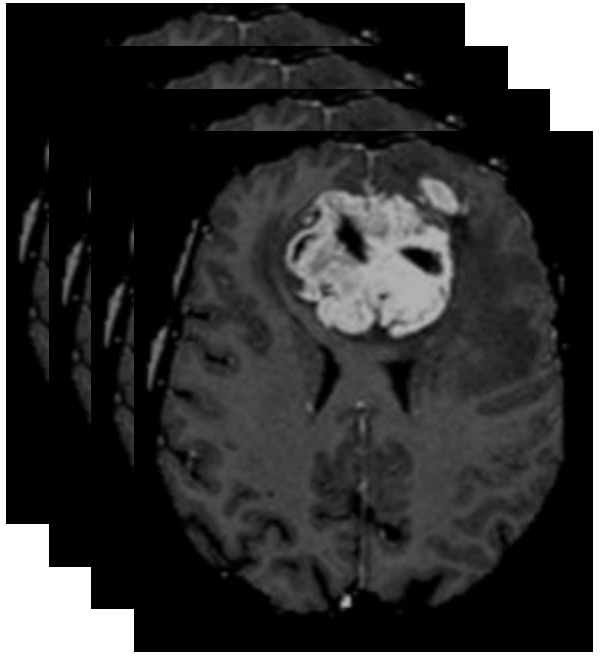
Function approximation

a brain MRI

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

skull-stripped MRI



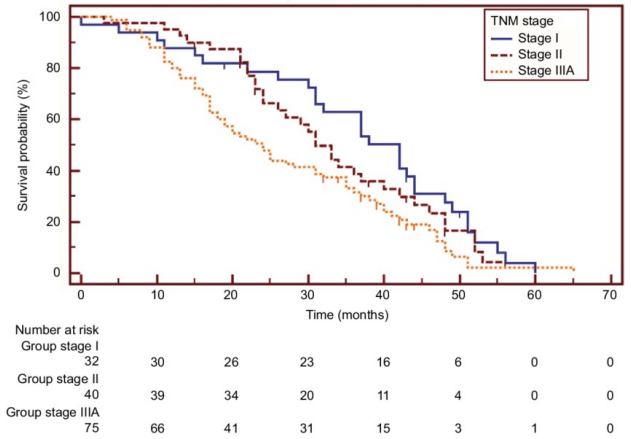


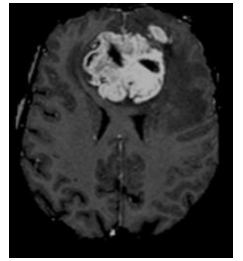
Function approximation

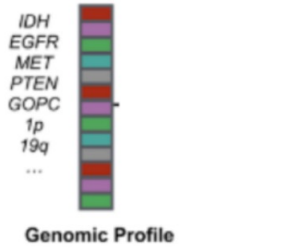
multimodal MRI images

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

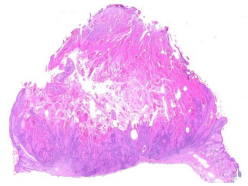
Segmentation of region-of-interests (e.g., tumors)

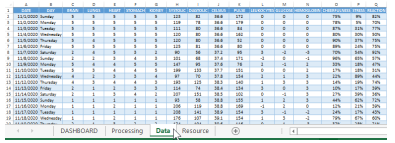






Genomic Profile





Function approximation

heterogeneous information

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

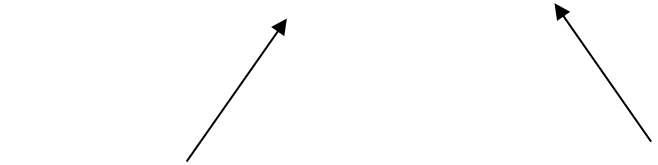
outcome (e.g., survival)

Training

$$(X, y)$$

Training

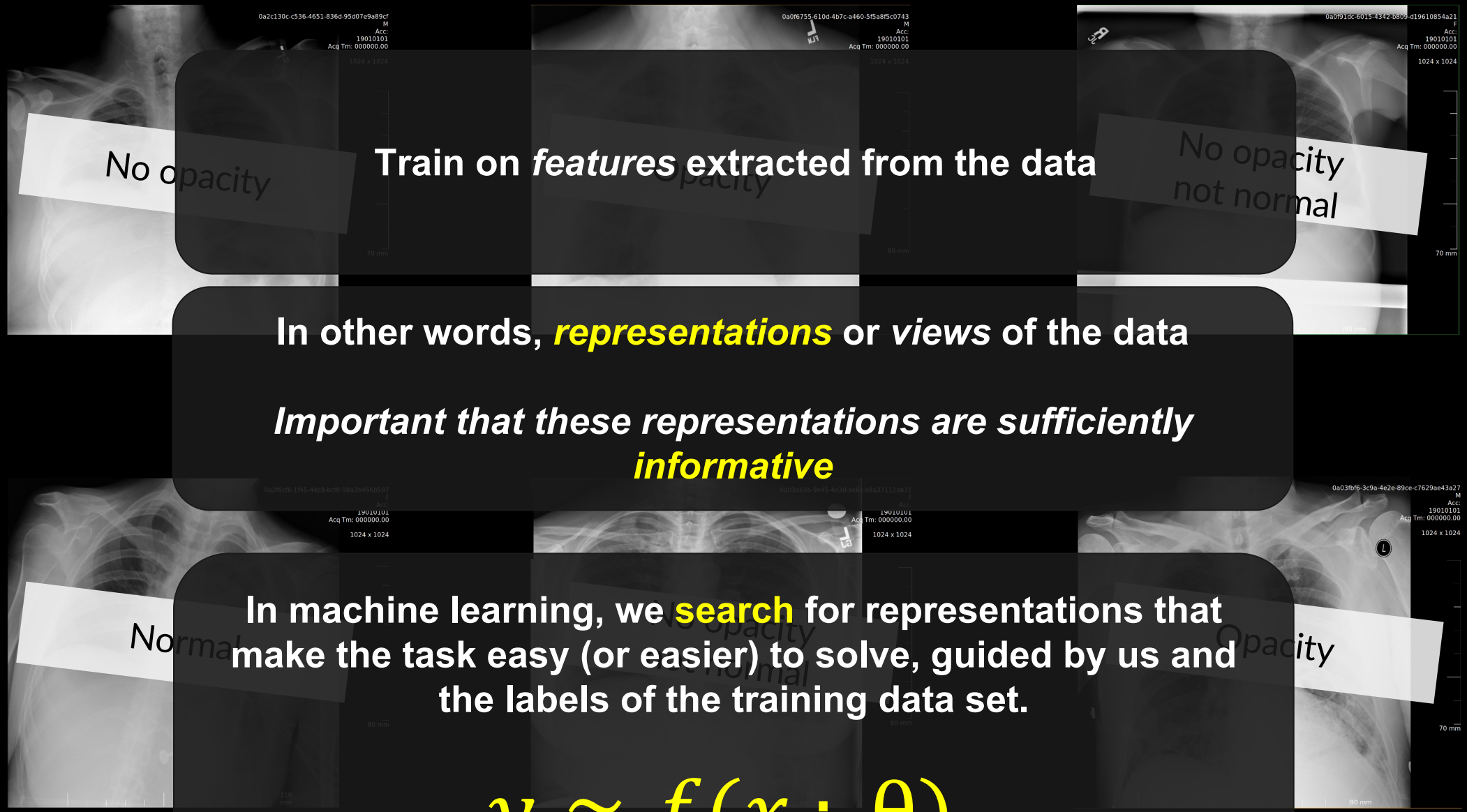
$$(X, y)$$



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

Important that these representations are sufficiently informative

In machine learning, we **search** for representations that make the task easy (or easier) to solve, guided by us and 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**

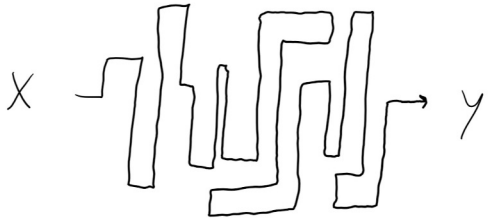
02 We pick the model family F. The parameters Θ are found automatically through training

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

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

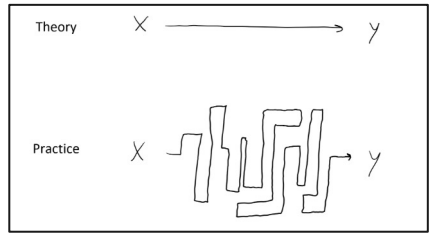
The family $\mathcal{F}$ of models considered, i.e., the *hypothesis space,* should be
(i) efficiently searchable, (ii) leading to expressive models that can
(iii) generalize beyond the training data distribution (i.e., work well on new data)

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

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

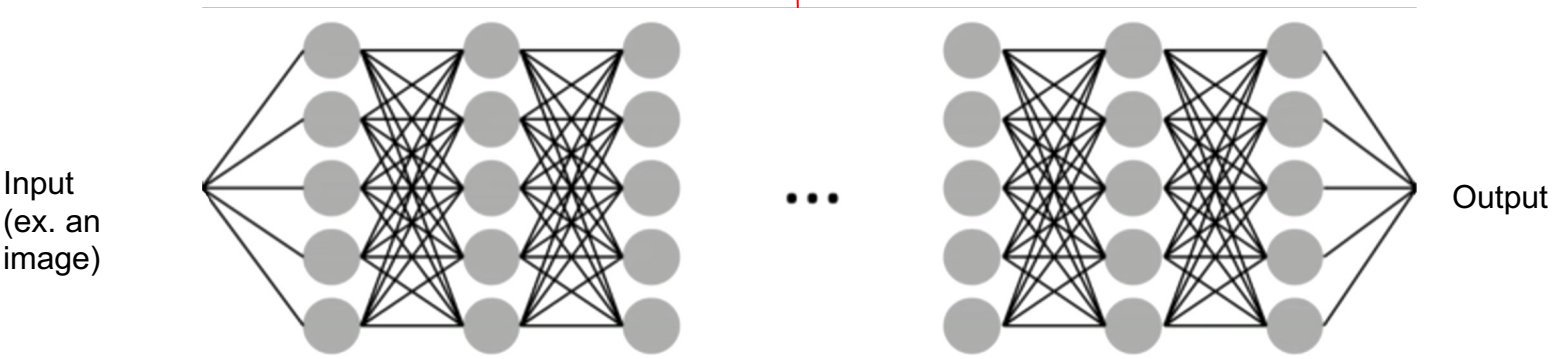
03 In practice: some models are better suited to some tasks than others.

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



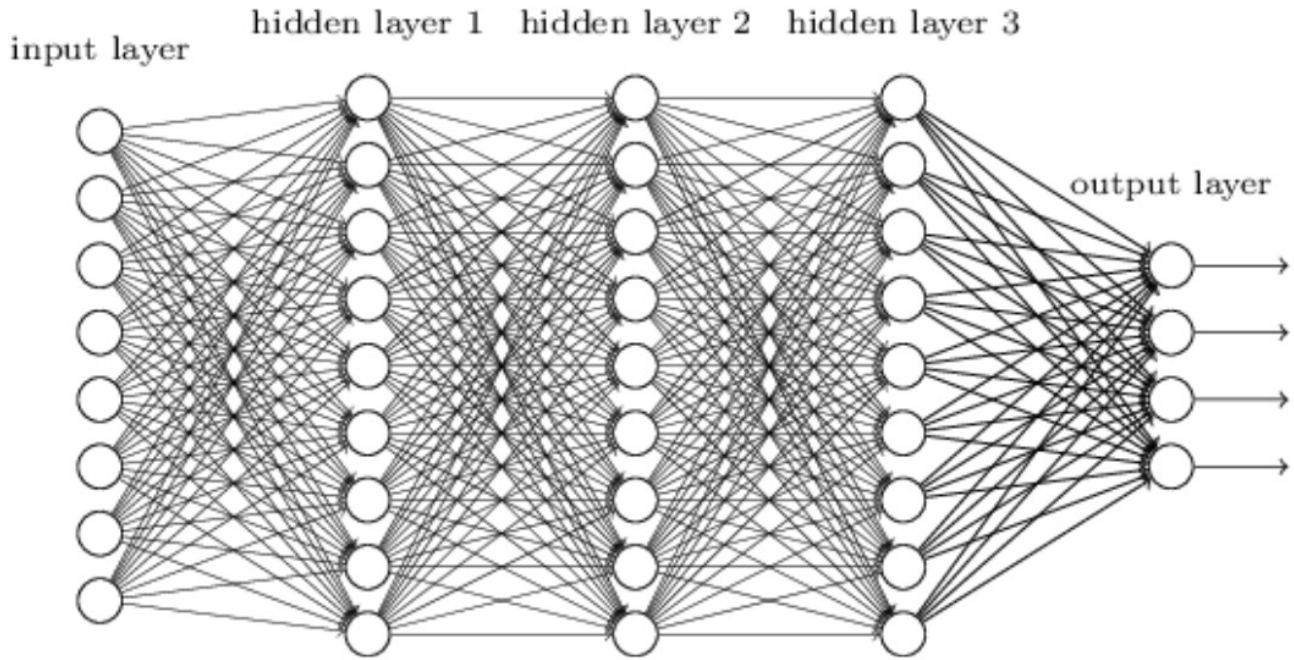
Deep learning

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

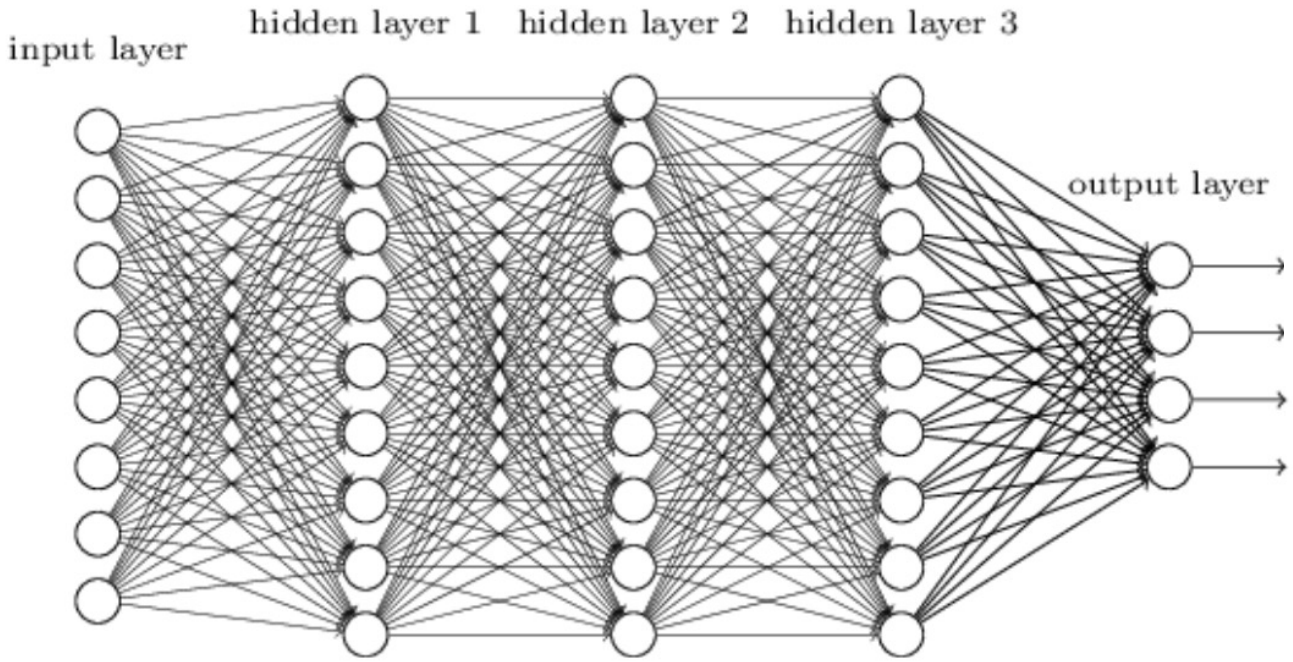


Artificial neural networks

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


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

A geometric analogy:
uncrumpling paper



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

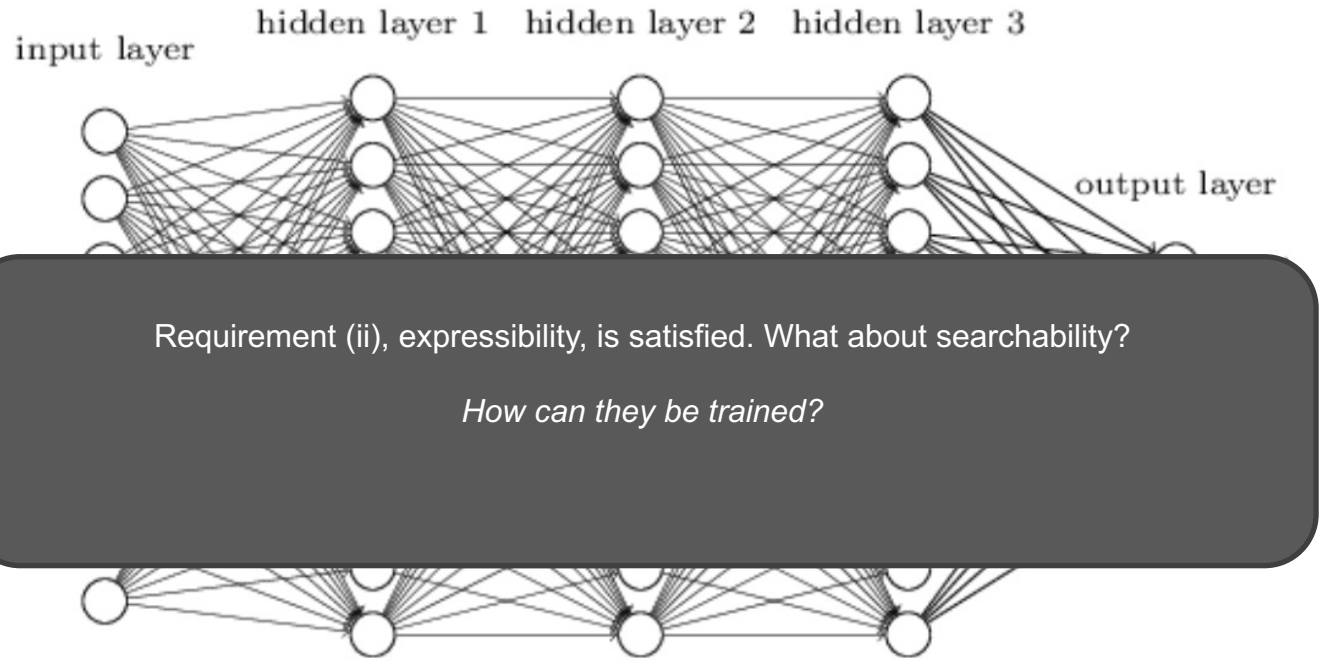
$$y = \sigma_{k+1}(W_{k+1}^T \sigma_k(W_k^T \left(\cdots \left(\sigma_1(W_1^T x) \right) \cdots \right) = f(x; \theta)$$

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:

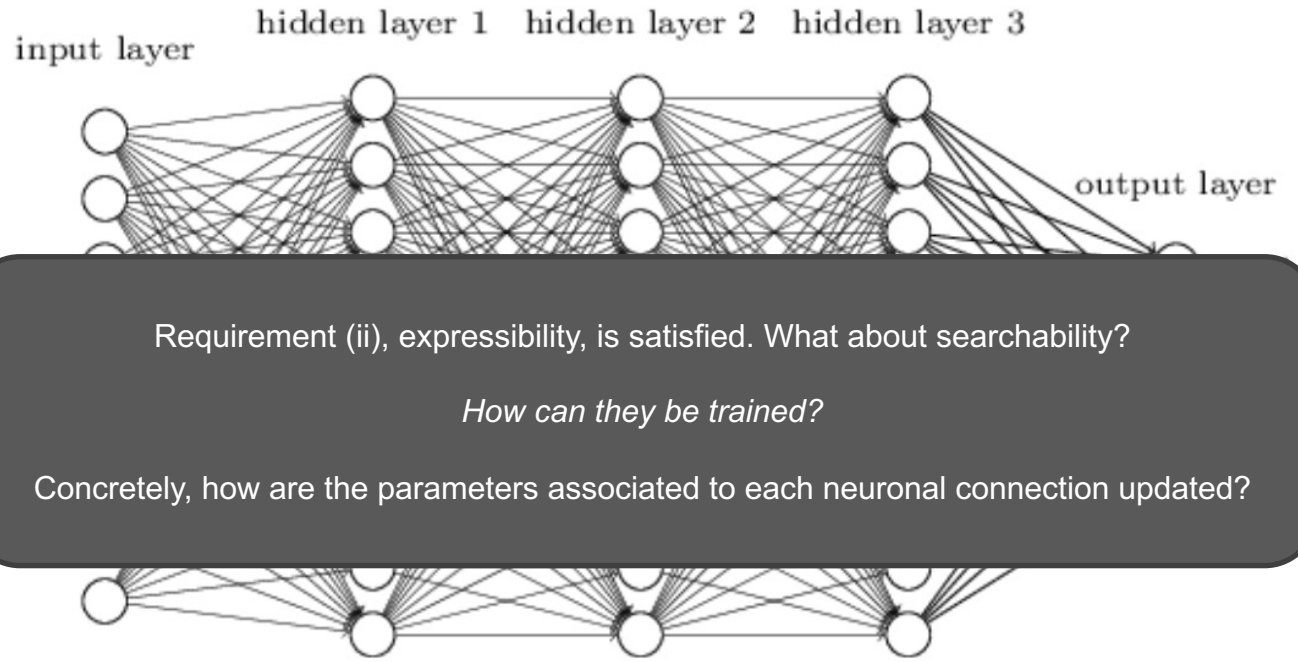
$$y = \sigma_{k+1}(W_{k+1}^T \sigma_k(W_k^T \left(\cdots \left(\sigma_1(W_1^T x) \right) \cdots \right) = f(x; \theta)$$

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

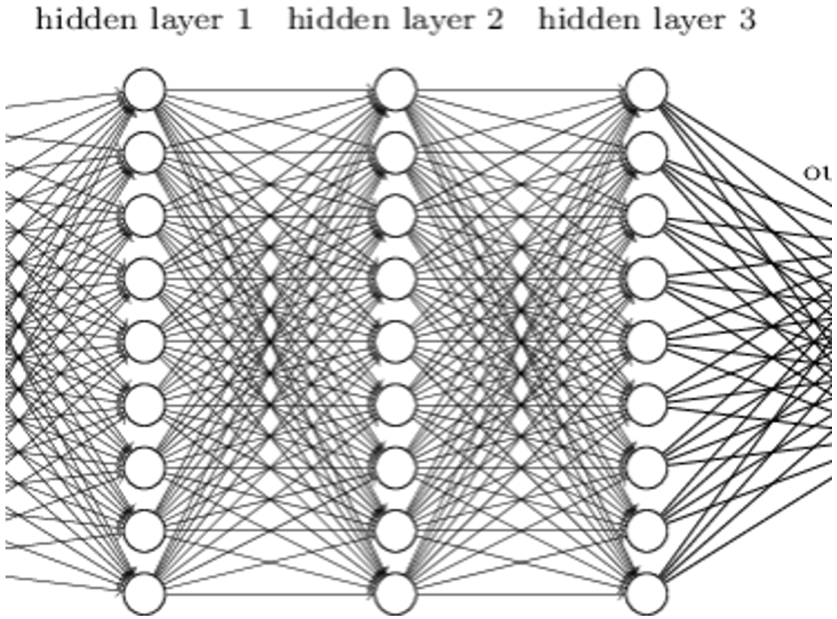
Each family member of F is complex transformation broken down into a composition of simpler ones, parametrized by the layer weights.

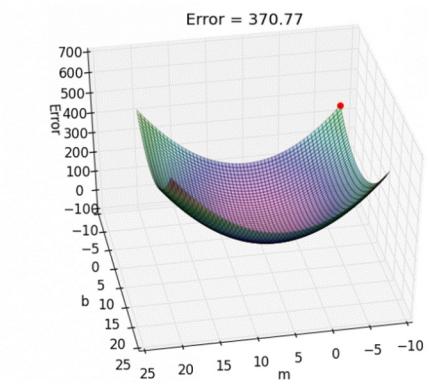
Three basic ingredients:

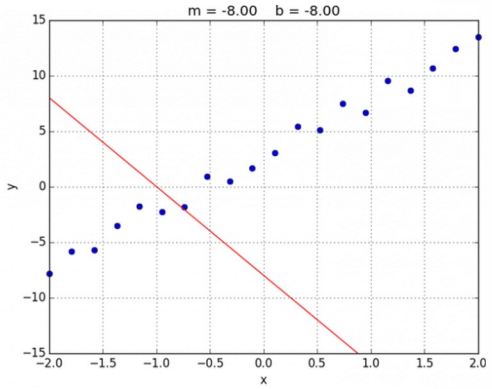
1. A **loss function**: Used to measure the network's performance on the training data. A **feedback signal** for the training process.

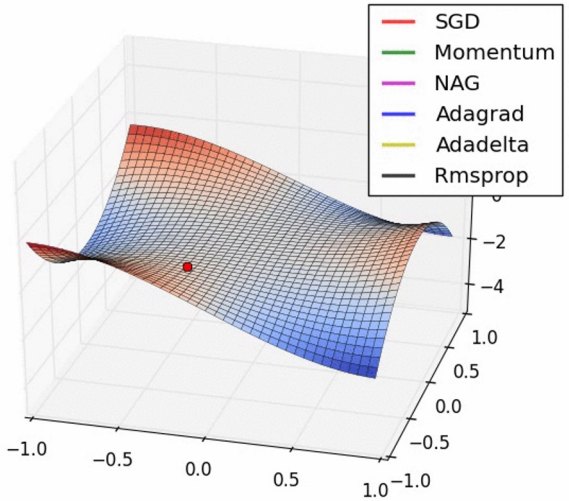
2. An **optimizer**: Used to update the *parameters* of the network to increase performance as measured by the loss function. **Gradient descent** and **backpropagation**.

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

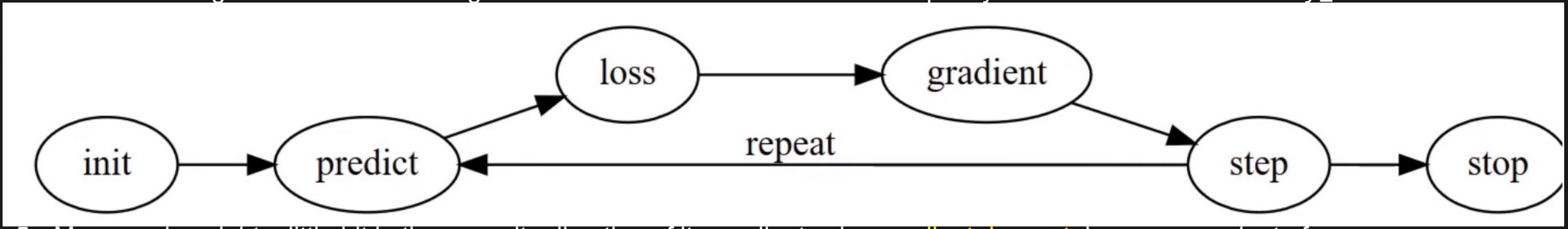
7. Do this over and over for several epochs (one epoch is one pass through all the training data), until you meet some chosen *stopping criteria*

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

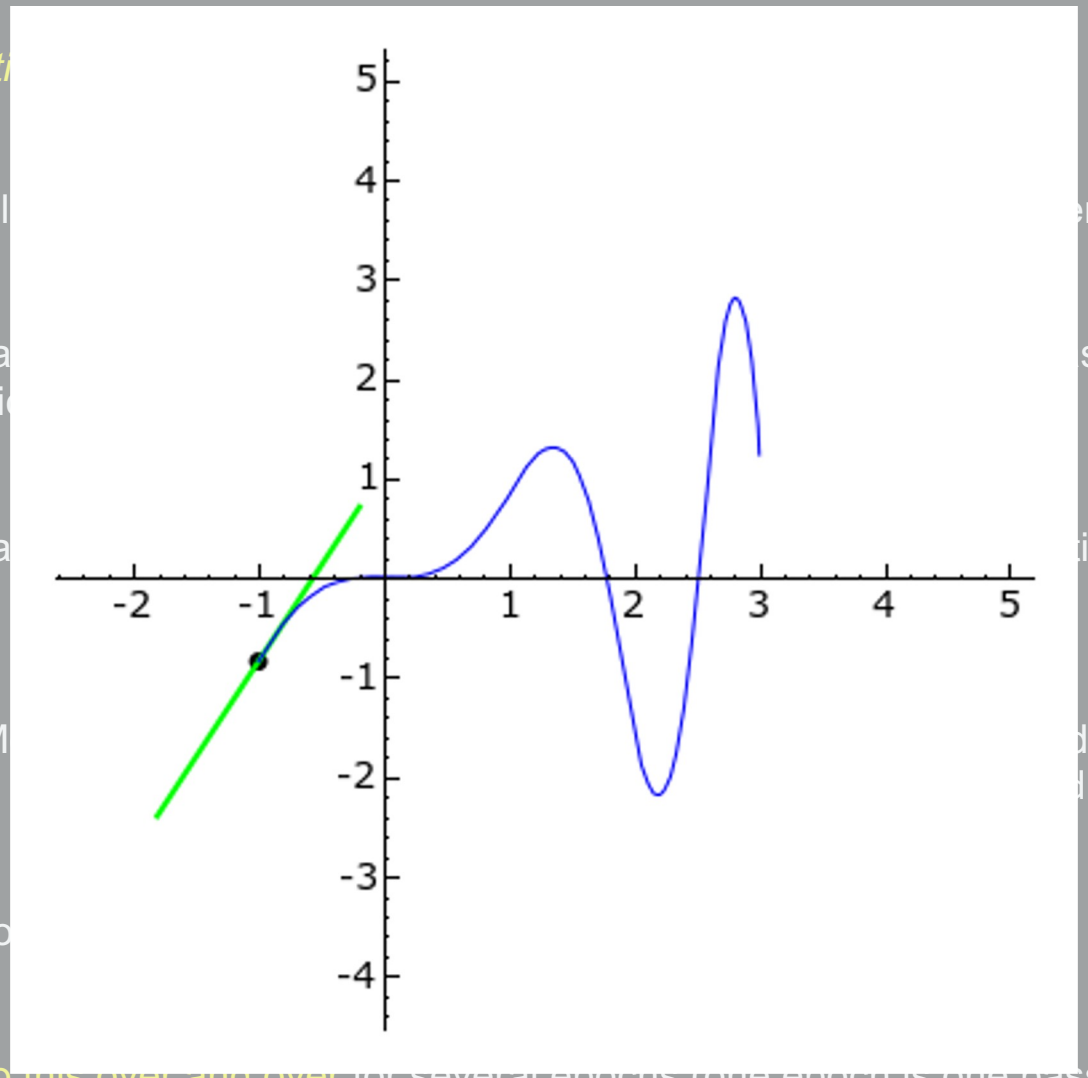
2.Col

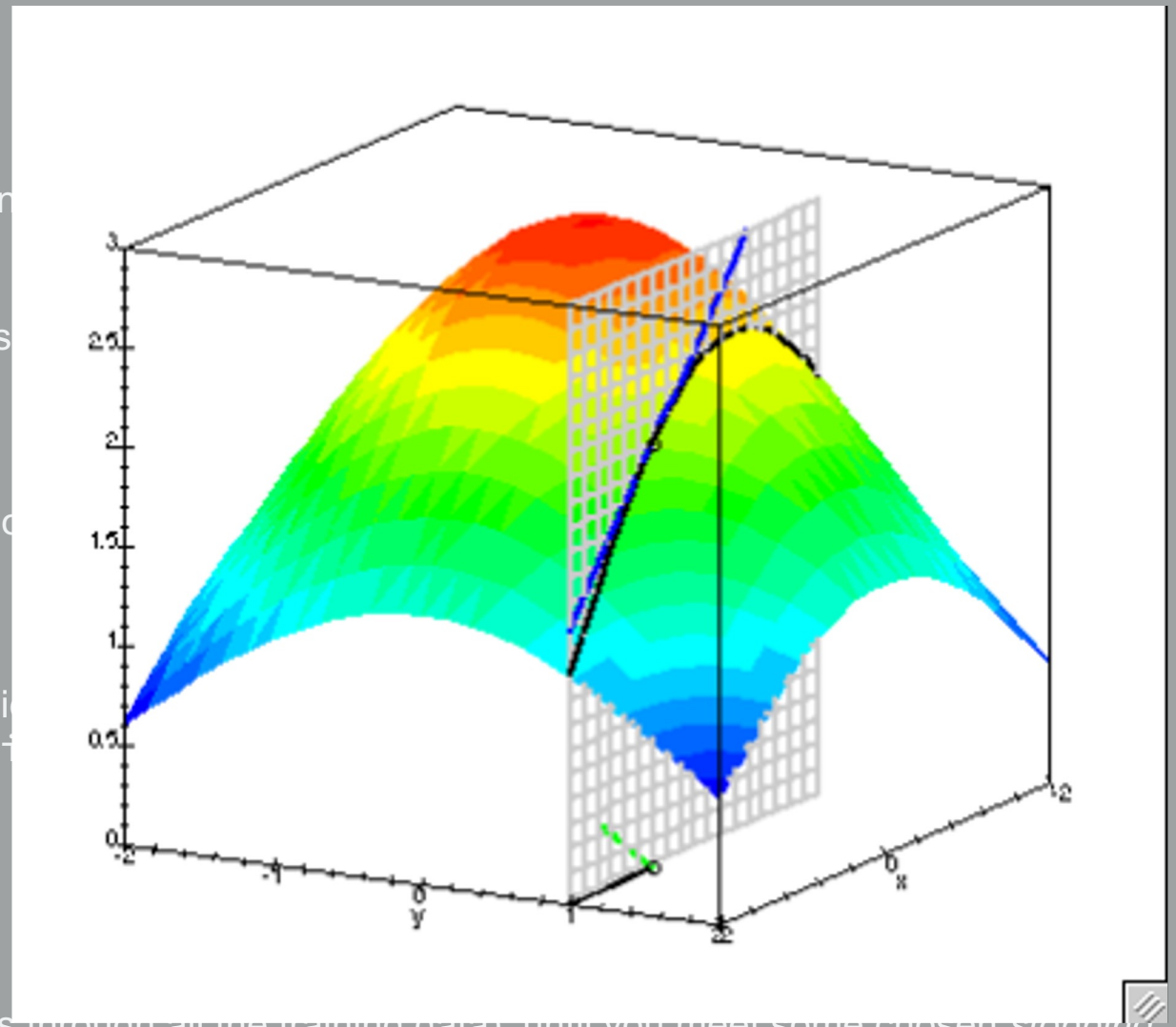
3. Ca
predi

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

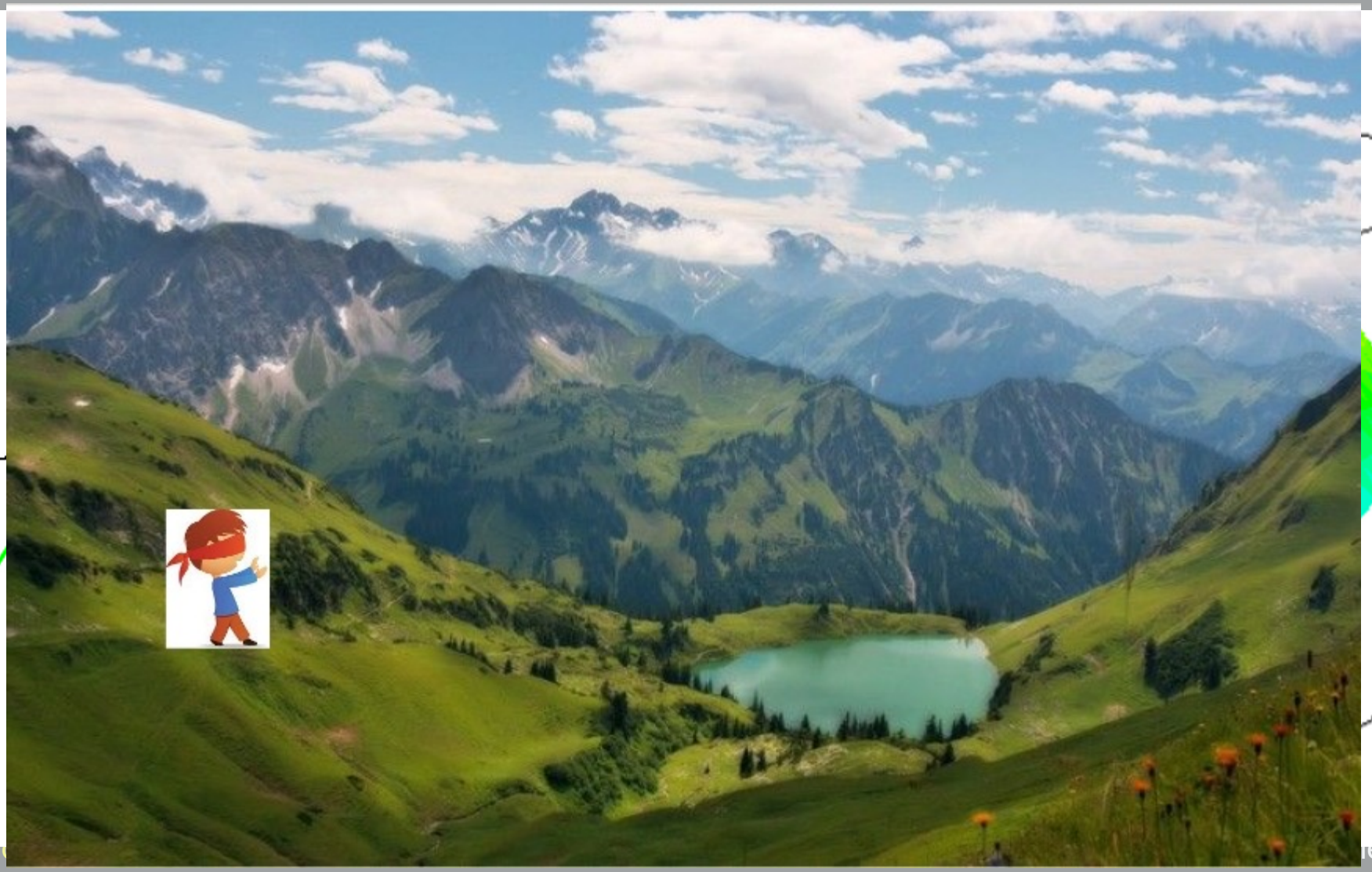
3. Ca
predi

4. Ca
loss.

5. M

6. Go

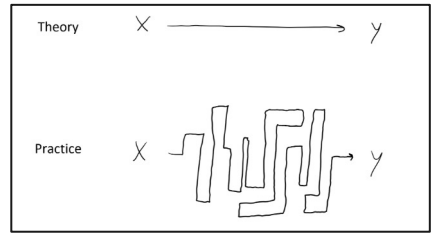
7. Do this over and ... the chosen *stopping criteria*



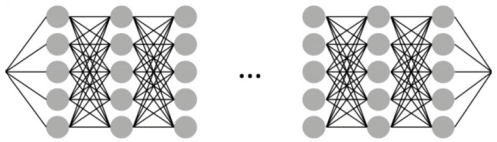
01 Machine learning: **function approximation** based on **training**

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

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



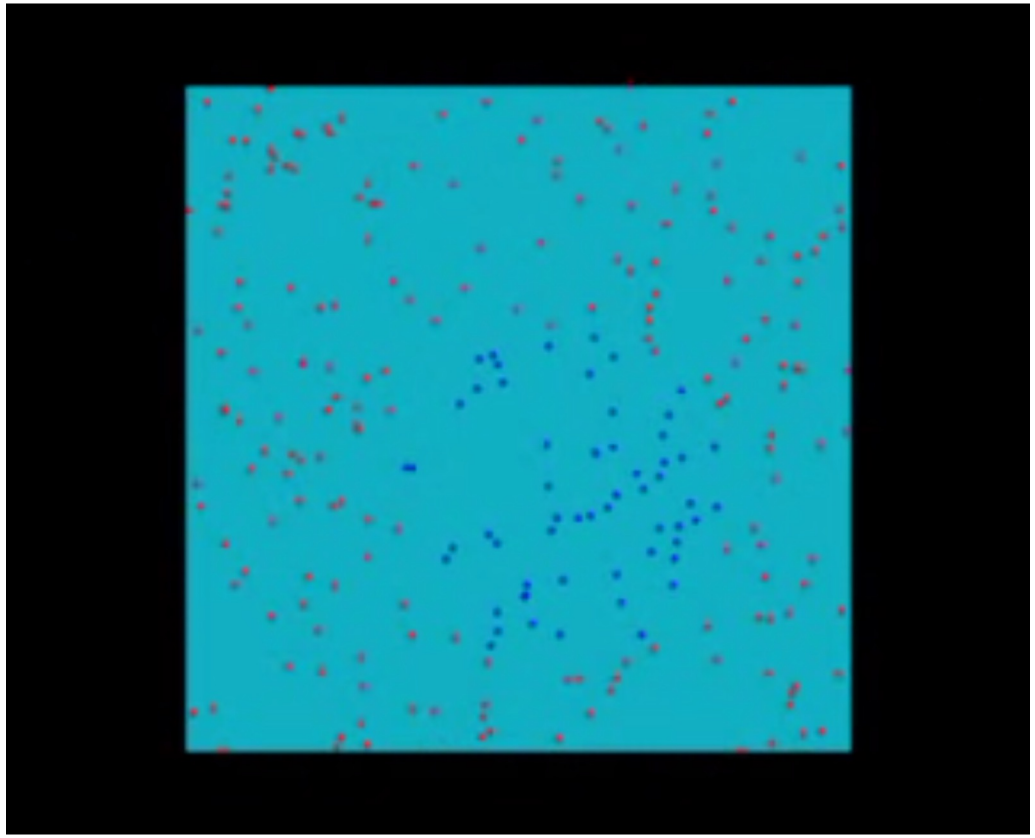
03 In practice: some models are better suited to some tasks than others.



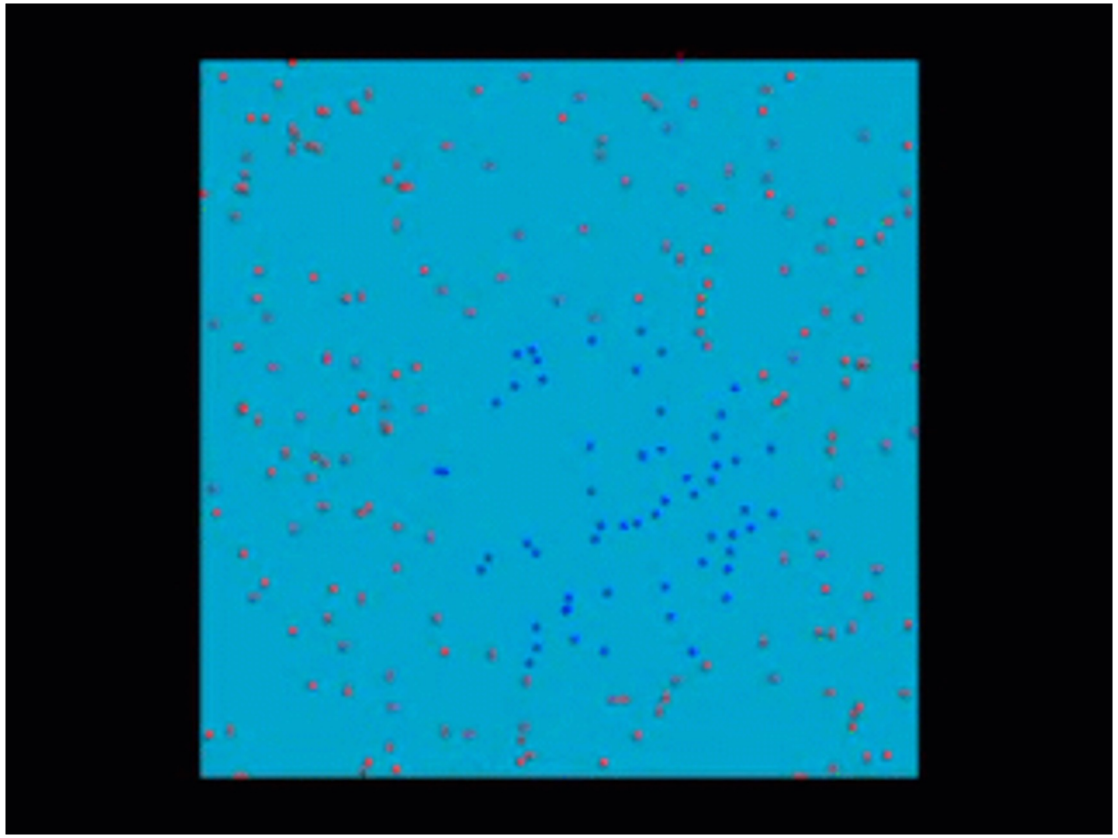
04 In deep learning: a particular choice of f. A **composition** of simple functions, trained jointly, typically using GPUs or other accelerators.

Representations and the power of transformations

Transformations

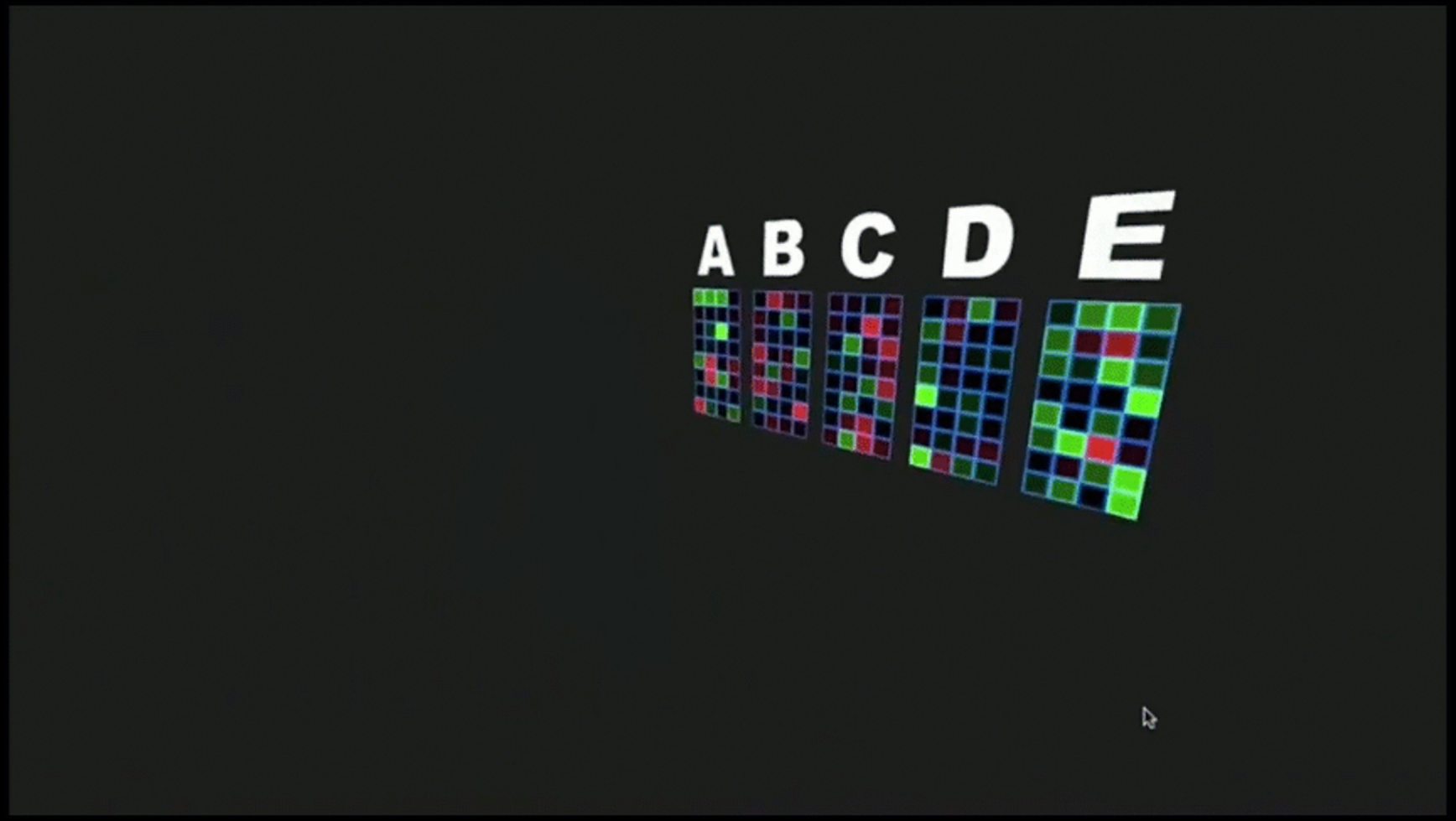


Transformation

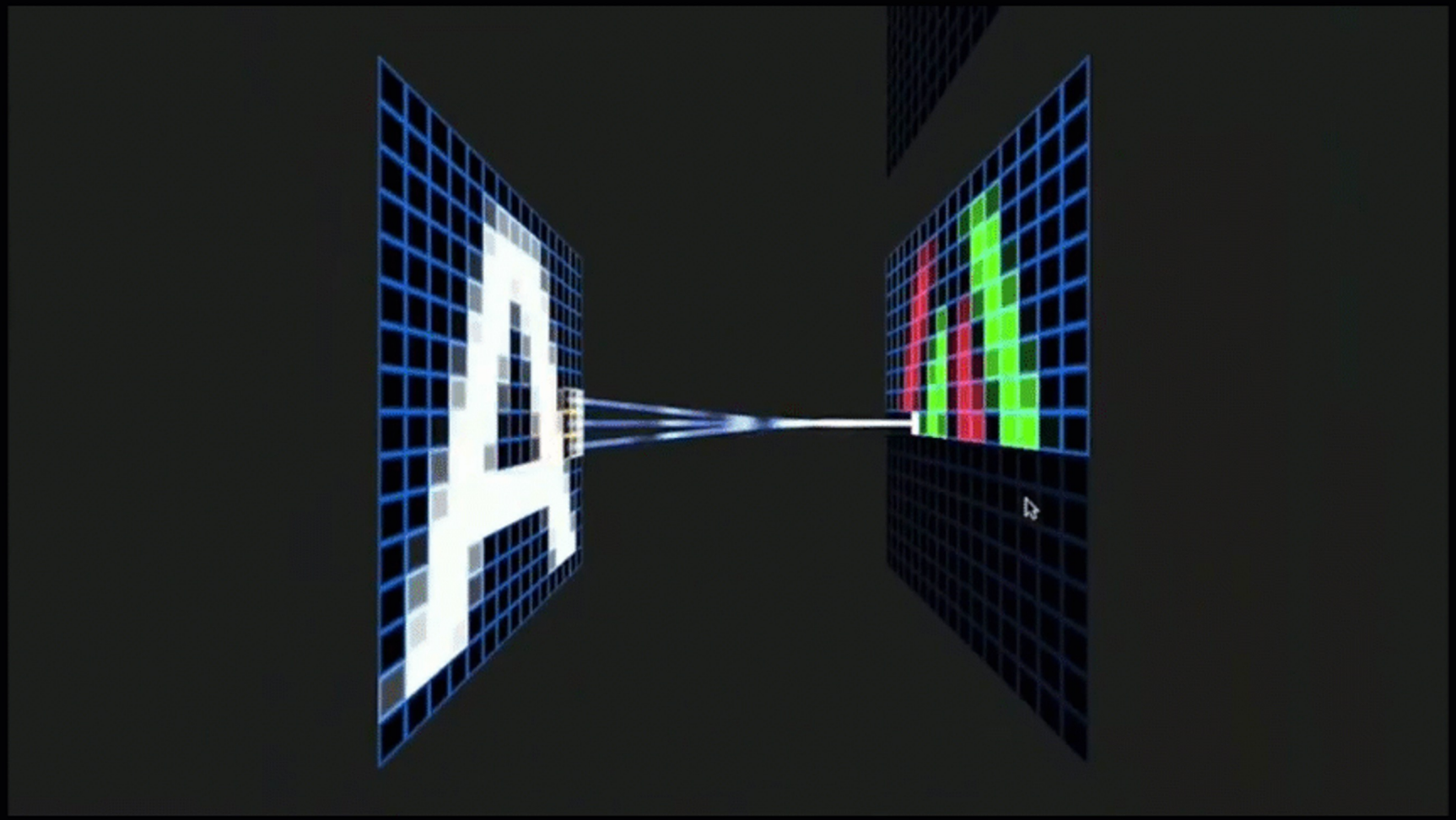


What transformations should be applied?

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



Source: Otavio Good

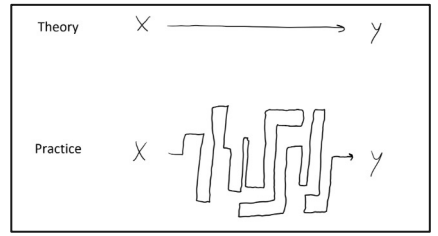


Source: Otavio Good

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

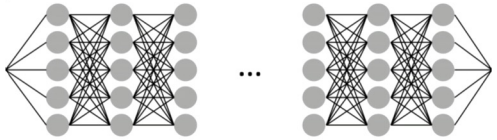
$$y \approx f(x;\theta)$$

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

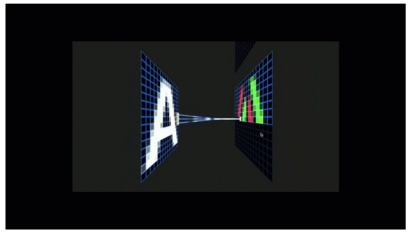


03 In practice: some models are better suited to some tasks than others.

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



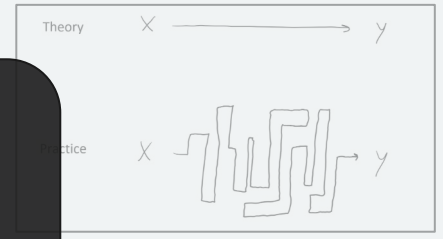
05 In deep learning: finding a **hierarchical set of geometric transformations**, ending in a representation that makes the desired task easy to solve.



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

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

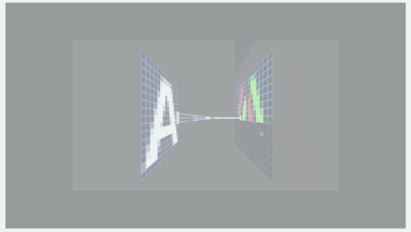
02 We pick the model family $\mathcal{F}$. The parameters Θ are found automatically through training



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

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.



Deep learning

searching for good hierarchical geometric representations

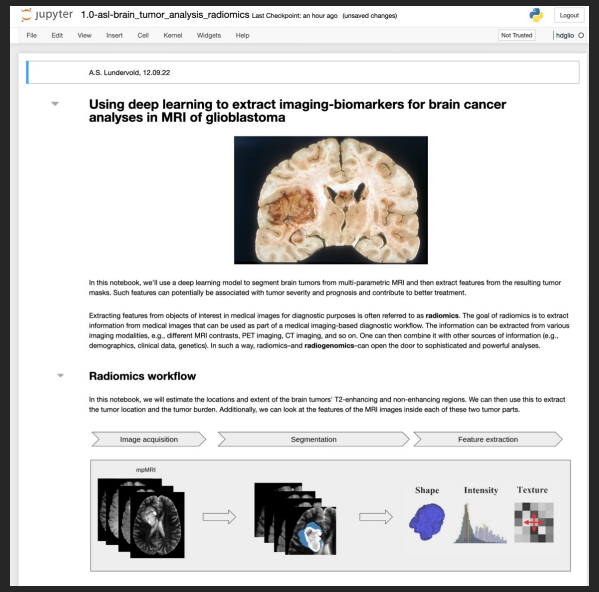
Plan

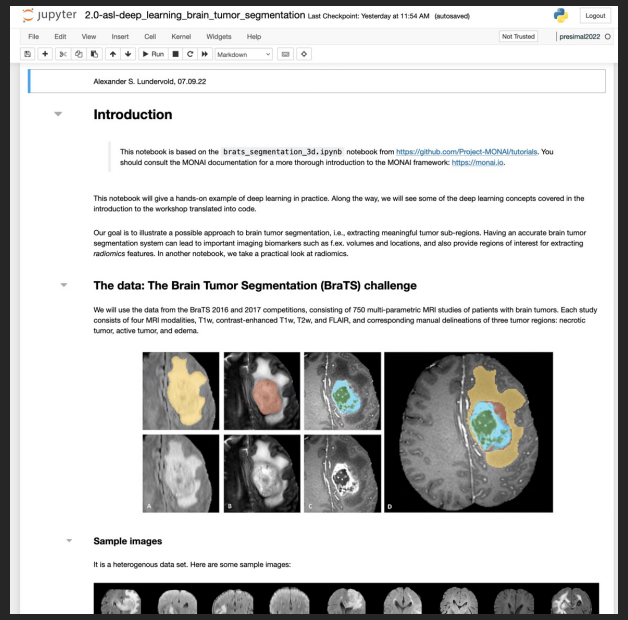
What *is* deep learning?

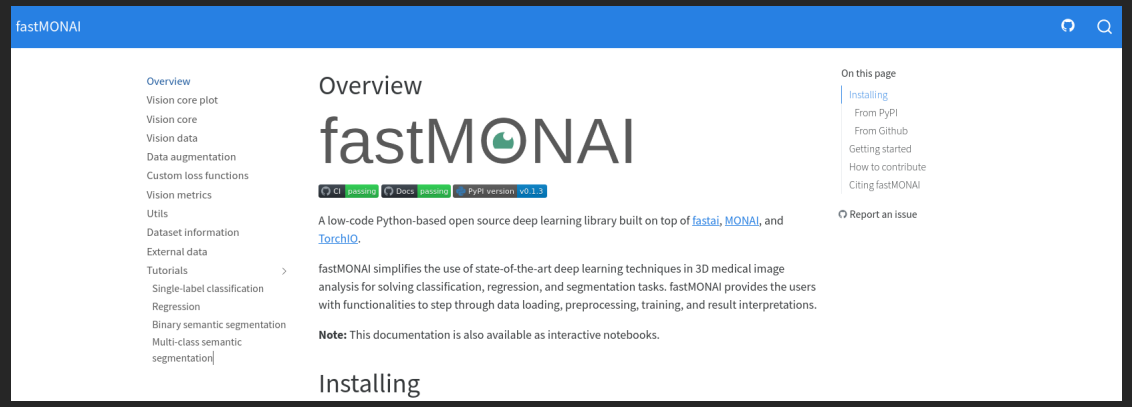
How do you *do* deep learning?

Some perspectives, opportunities and challenges

The workshop notebooks and fastMONAI provide a start. Please get in touch if you want additional pointers!







Western Norway University of Applied Sciences
MMIV
Haukeland University Hospital
Alexander Selvikvåg Lundervold
allu@hvl.no; lundervold.net