INTRODUCTION TO MACHINE LEARNING

SCHOOL OF CULTURE AND SOCIETY
AARHUS UNIVERSITY

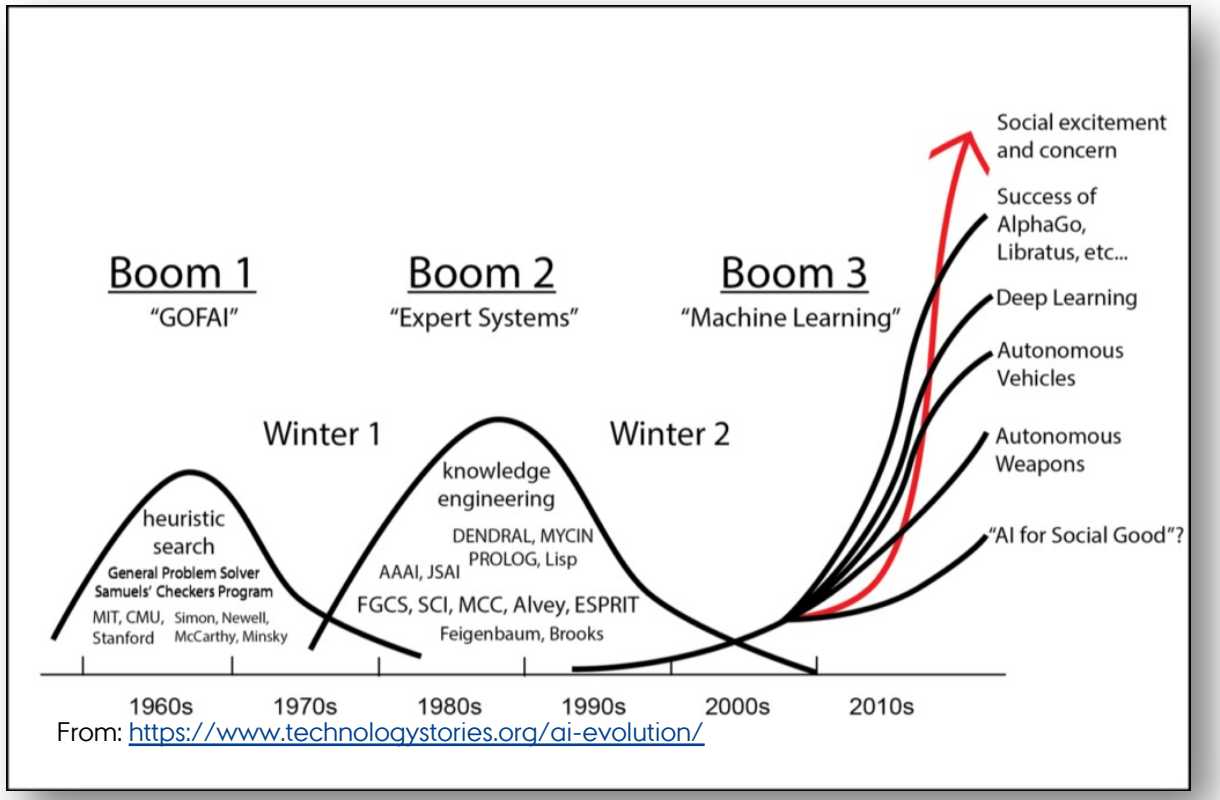
SIMON ENNI
POSTDOC

SCHEDULE

Time	Activity
10:00 – 10:20	Introduction
10:20 – 11:00	**Exercise 1:** Image classification with CNN
11:00-11:15	Break
11:15 – 11:45	**Exercise 1** cont.
11:45 – 12:00	Reflection exercise
12:00 – 12:30	Lunch Break
12:30 – 13:30	**Exercise 2:** Language modeling with ML
13:30 – 14:00	Reflection on exercises, discussion, perspectives

BACK TO THE BASICS

ML - WORKHORSE OF MODERN AI



From: https://www.technologystories.org/ai-evolution/

Machine Learning (ML) is the most recent paradigm within AI – dominated the field since the middle of the 00's

ML: use data rather than code to program computers.

- *Learning* is defined as improving measured performance with respect to a concrete task using experience (data)
- This requires...
  1. a way to measure performance,
  2. a source of data,
  3. a well-defined task, and
  4. (typically) a lot of computation

CLASSICAL ALGORITHMICS

Problem is *clearly and formally specified*:
Input and expected output.
Limited options and generally no ambiguity

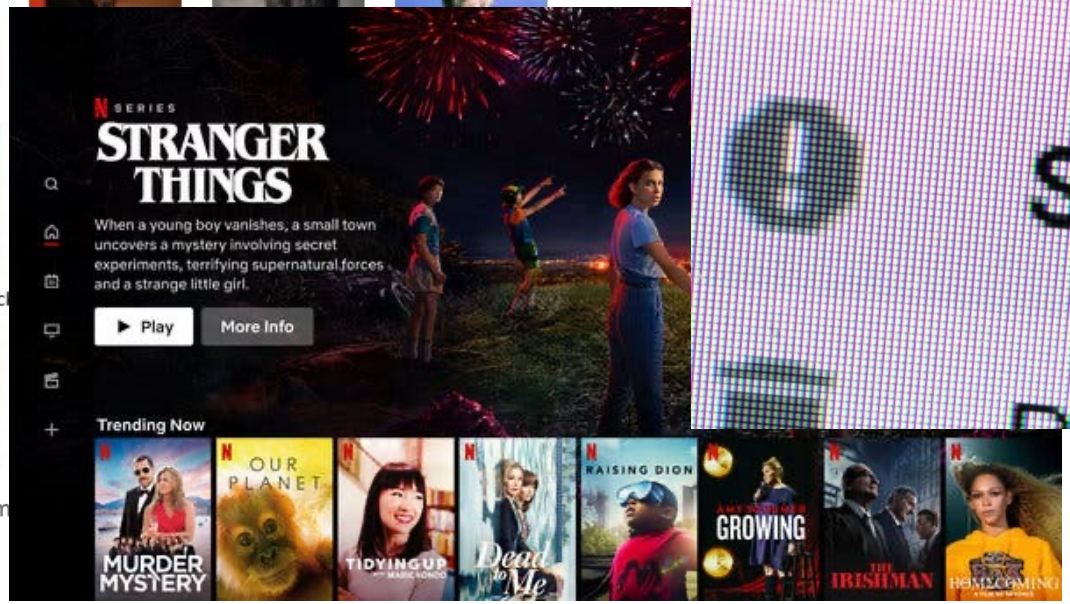
Programmer models the problem as an algorithmic problem and implements an algorithm that *solves* it

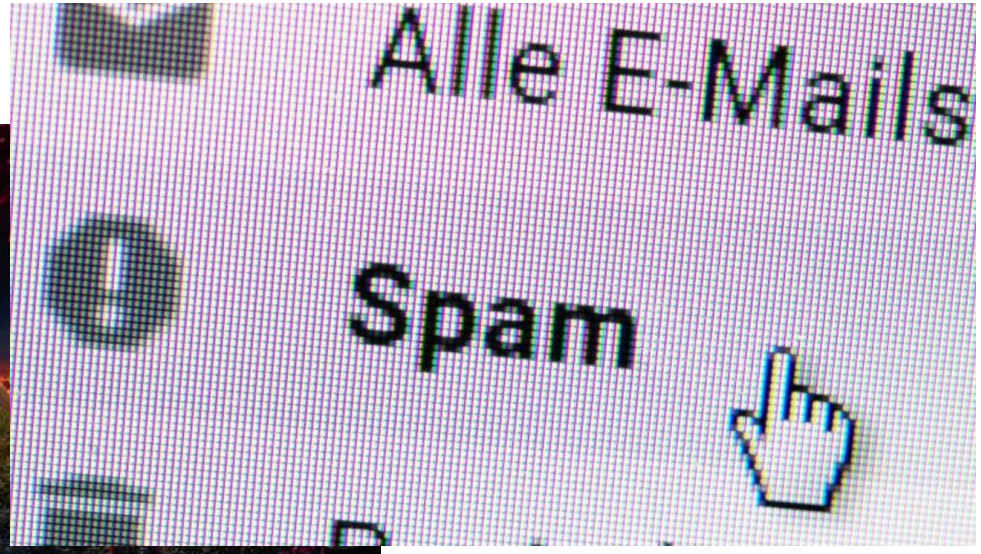
```
if driver has paid:
    open barrier
else:
    stay closed
if barrier is open AND car has past:
    close barrier
else:
    wait
```

MACHINE LEARNING

Adresses problems where solutions can't be identified by *formal criteria* but can be *exemplified* reliably.

Preview of images from





2-and-2: Can you imagine a problem we can solve with classical algorithms and one that is better solved by machine learning?

WHAT IS MACHINE LEARNING?

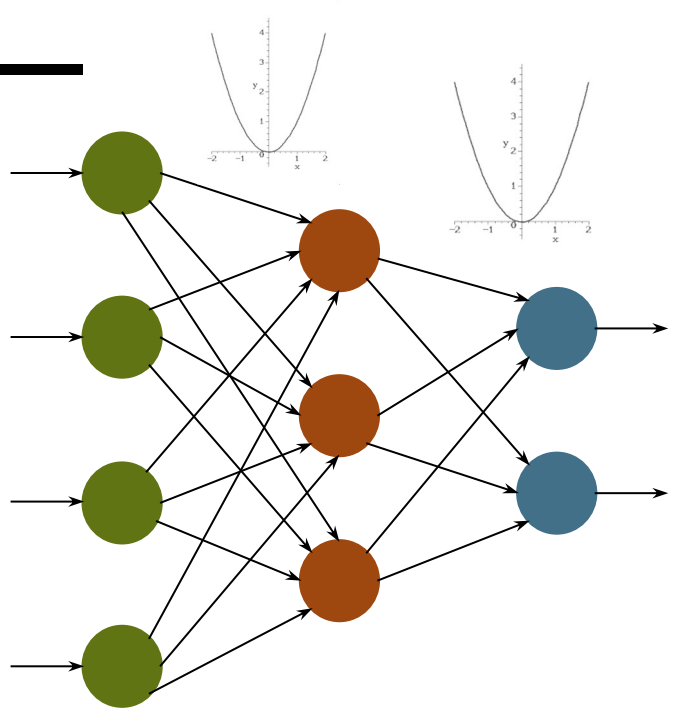
Definition: *A computer program is said to* ***learn*** *from experience* E *with respect to some class of tasks* T *and performance measure* P*, if its performance at tasks in* T*, as measured by* P*, improves with experience* E*.*

Tom M. Mitchell (1997)

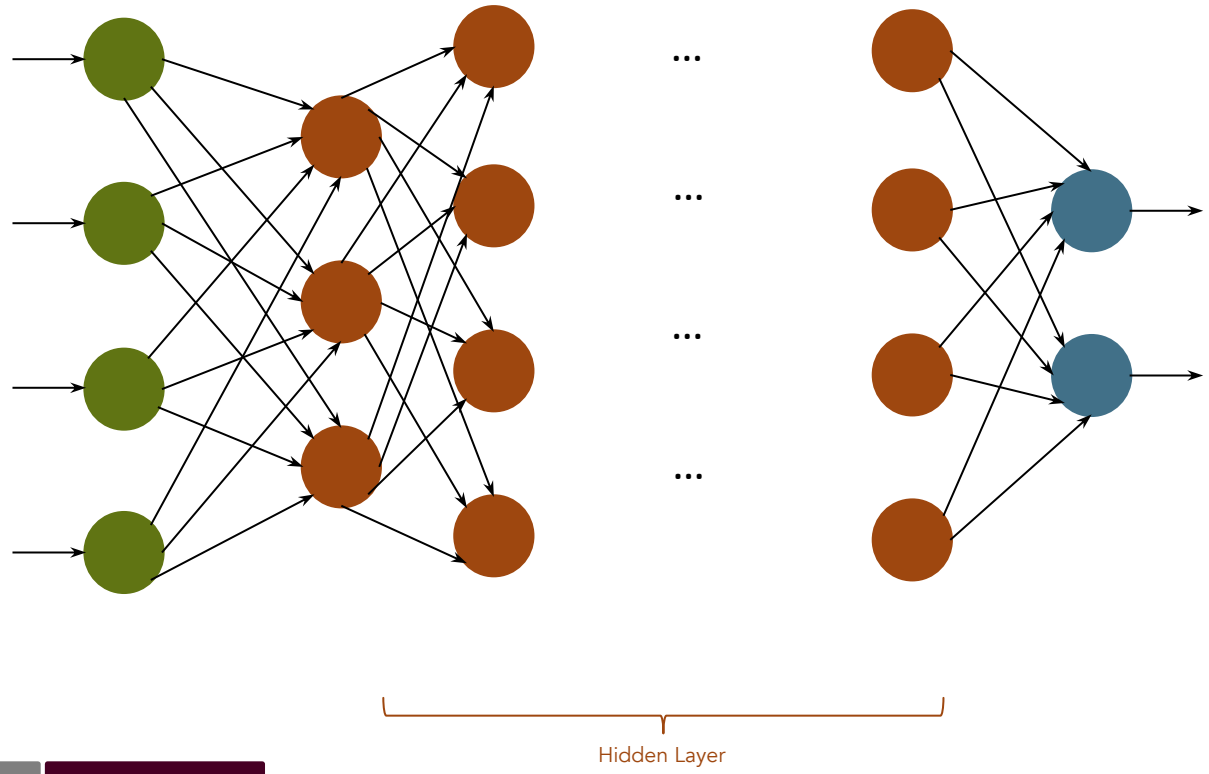
Translation: Machine Learning creates programs, that *measurably improve* at a *concrete task* given *increased experience*.

Mitchell, T. M. (1997). Machine learning. *Burr Ridge, IL: McGraw Hill*, *45*(37), 870-877.

NEURALE NETWORKS

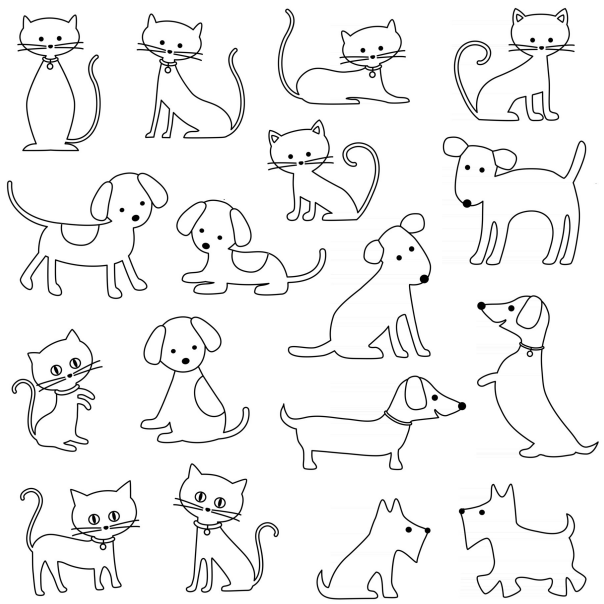


Deep Neural Network



Case: Dog or cat?

Collected data



Features:

- Nose length
- Ear angle
- Tail length
- Whiskers
- Body length
- Leg length
- Coat type
- Colour

Target: Dog or cat

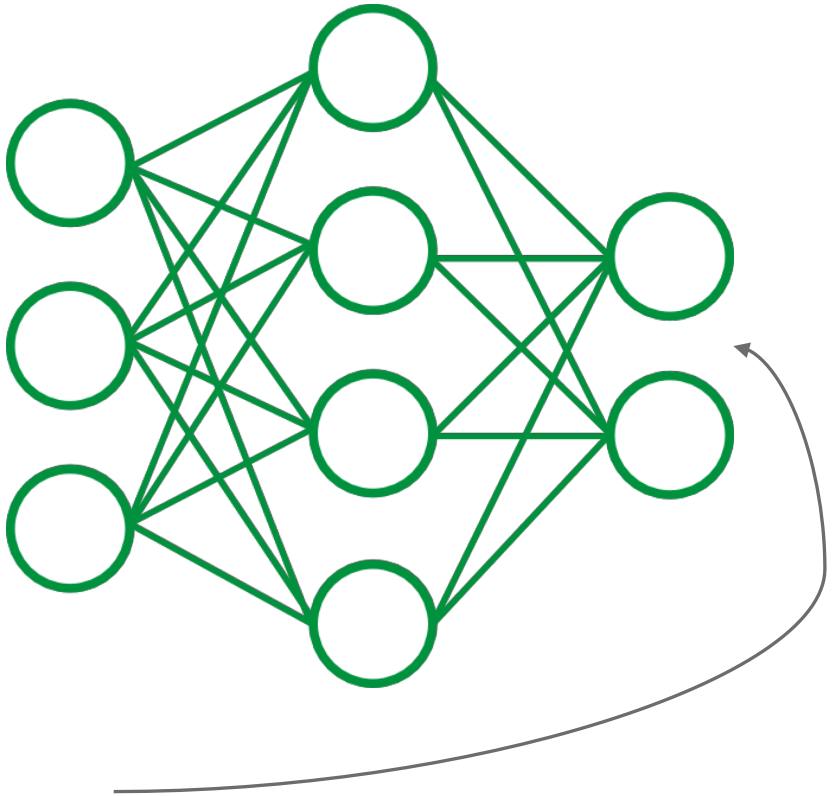
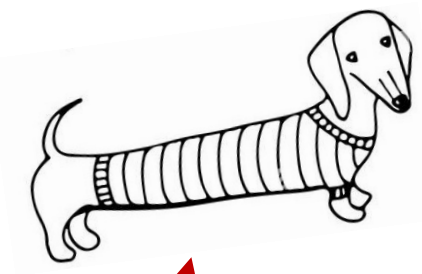


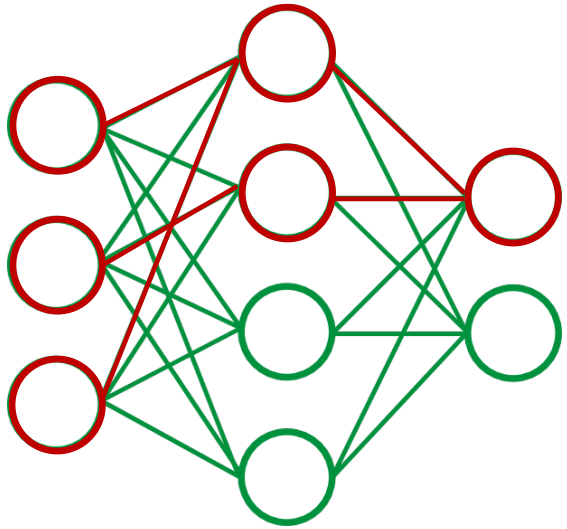
image: https://www.vecteezy.com/vector-art/2512004-cats-and-dogs-black-outline-digital-stamps

Case: Dog or Cat



Features:

- Nose length: 15 cm
- Ear angle: -90°
- Tail length: 8 cm
- Whiskers : no
- Body length: 45 cm
- Leg length: 9 cm
- Coat type: smooth
- Colour: light brown



Prediction:
This is a dog!

New data instance that our model have ***not*** seen before!

Image: https://www.alamy.com/set-of-dog-dachshund-illustrations-image158965096.html

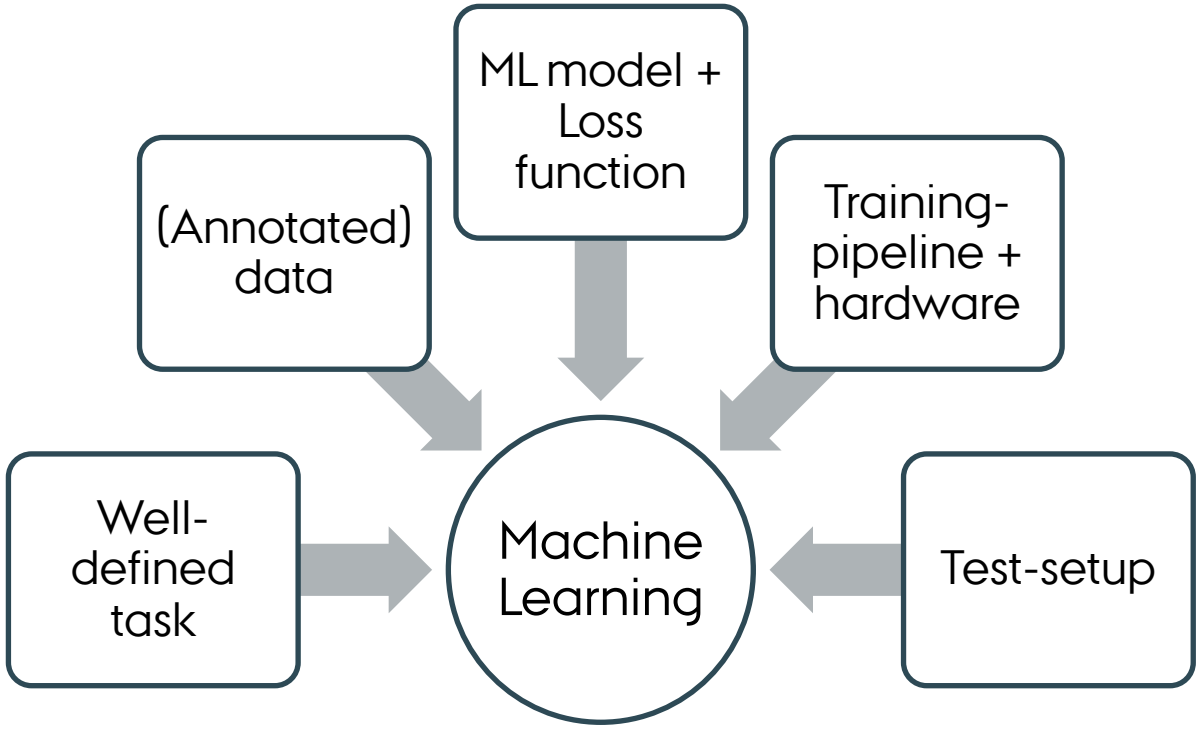
HOW ML WORKS

ML AS A MODELING SCIENCE

A *model* is trained to *minimize its errors* for a *concrete task* using *inductive biases* and a *given dataset*

ML typically works within other subfields based on the type of "experience"/data that is used for learning:

- Data types:
  - Text → Natural Language Processing,
  - Images → Computer Vision
  - Actions and consequences → Reinforcement learning



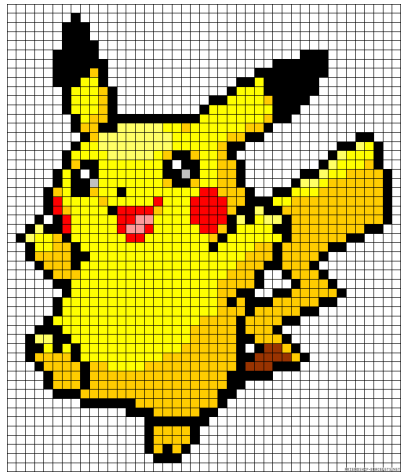
IMAGES AS DATA

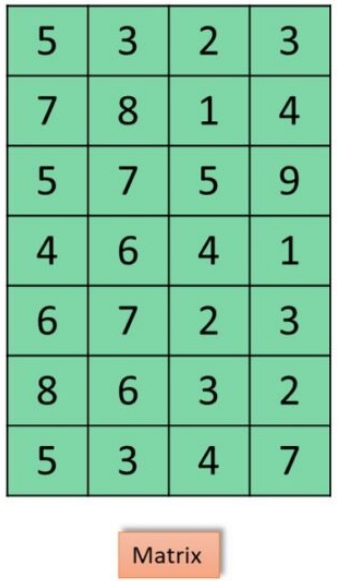
To optimize performance (i.e. ”learn”) the input and output must be *digital*.

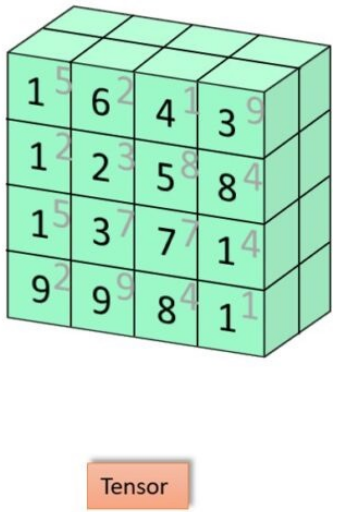
A picture can be represented digitally in a number of ways (that are all just different strings of bits in the end).

- Grid of pixels
- Vector graphics (hierarchy of shapes)
- Matrices and tensors (math)

In ML we often use tensors and matrices for their mathematical properties and speed



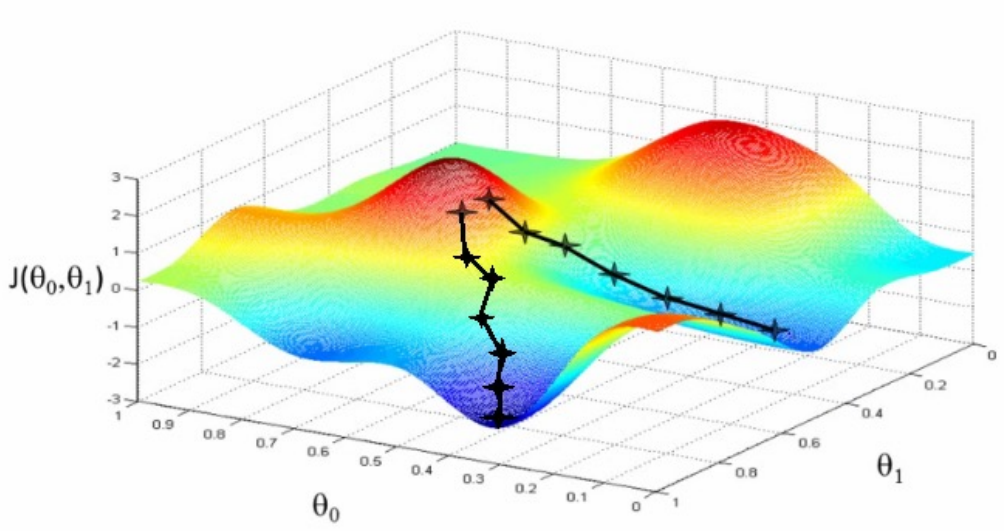




GRADIENT-BASED OPTIMIZATION

Almost all modern ML uses a variant of *gradient-based optimization*.

- (Stochastic) gradient descent exploits that many *loss functions* with ML models can be *differentiated* (i.e. get the slope) with respect to model parameters.
  → This gives us a *direction* in which to change model parameters to reduce loss
- No guarantee of optimality, but correctly calibrating step-size and other hyperparameters leads to a *very efficient* way of getting a *good enough* solution.



CHOOSING INDUCTIVE BIAS

Inductive bias:

"*any basis for choosing one generalization over another, other than strict consistency with the observed training instances"* (Mitchell 1980)

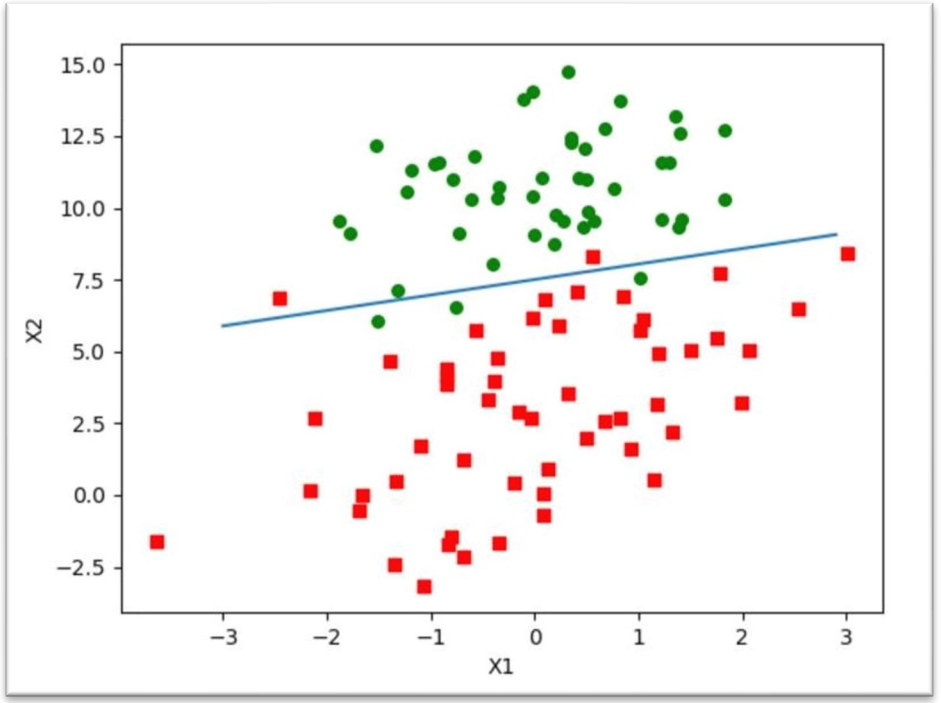
Inductive bias in ML: model type, architecture, loss function, constraints, regularization, etc.

Bias is *necessary* for generalization (Mitchell 1980) and *no bias works every time* (Wolpert 1996).

→ Need to tailor bias to context based on domain knowledge.

Examples:

- Logistic regression biased towards (restricted to) linear models.
- Occams razor biased towards *simple* models.



Mitchell, T. M. (1980). *The need for biases in learning generalizations* (pp. 184-191). New Jersey: Department of Computer Science, Laboratory for Computer Science Research, Rutgers Univ..

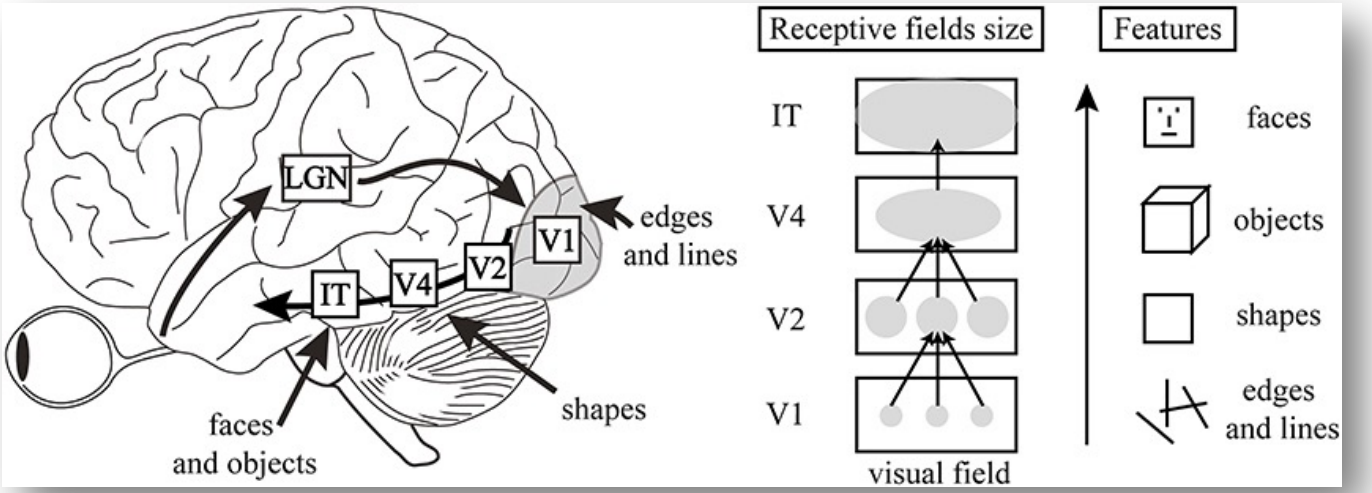
Wolpert, D. H. (1996). The lack of a priori distinctions between learning algorithms. *Neural computation*, *8*(7), 1341-1390.

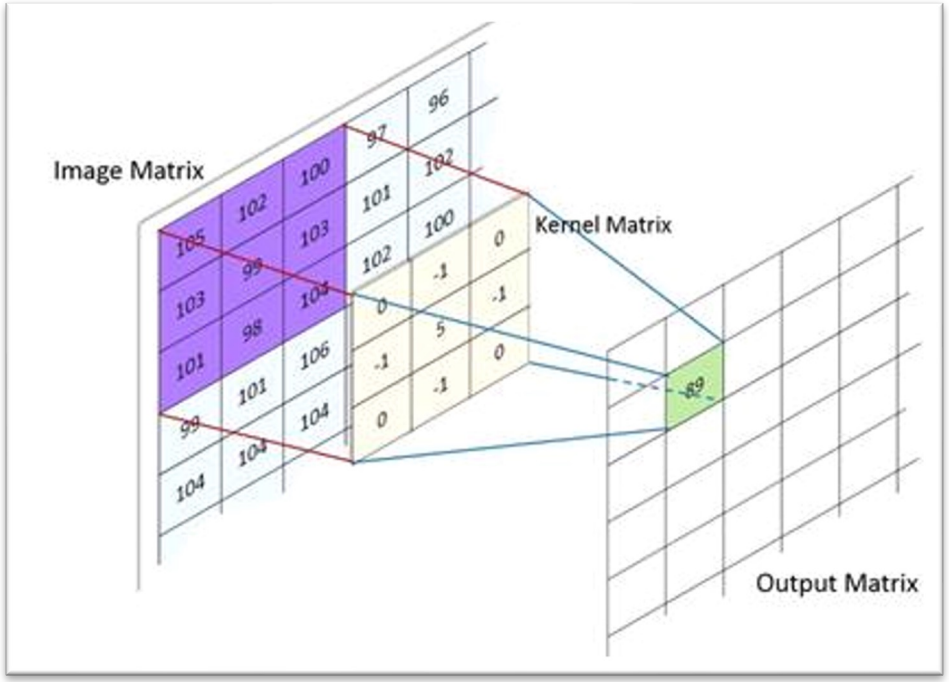
EXAMPLE: CNNS

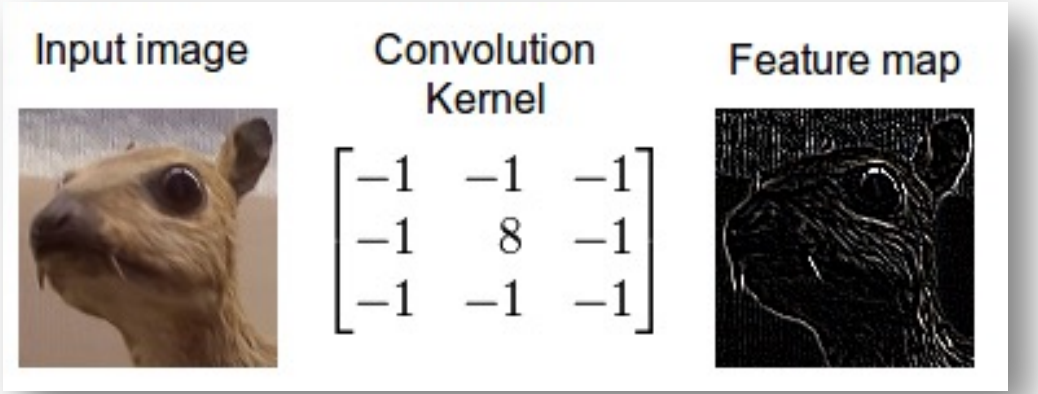
Convolutional Neural Networks are neural networks using special convolutional kernels.

Inspired by model for a monkeys' visual cortices

- Inherentent *translational invariance.*





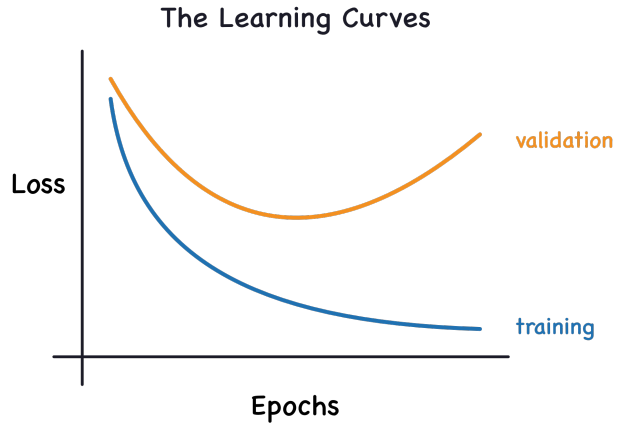


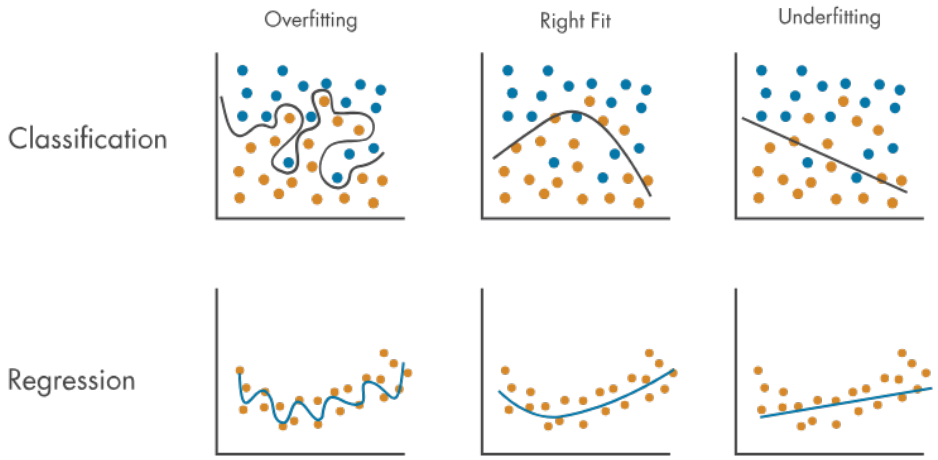
OVERFITTING

Big NNs are very powerful and this means that they easily *overfit* the data – that is, they end up *memorizing* data instead of *learning* the patterns that tie them together.

There are *many* different techniques to avoid overfitting, and we use a few in the exercises.

The most important ways to reduce overfitting is to engineer datasets to be as *exemplary* as possible (i.e. class balance, showcase as much diverse variation as possible, etc.) and choosing the right *inductive bias* when fitting a model.





PYTORCH

PyTorch is one of the most popular Deep Learning frameworks in the world → built to resemble the popular NumPy library.

PyTorch (like all DL frameworks) is focused on doing efficient tensor-operations and a strong automatic differentiation engine.

Examples are taken from the official PyTorch tutorial.

```
shape = (2,3,)
rand_tensor = torch.rand(shape)
ones_tensor = torch.ones(shape)
zeros_tensor = torch.zeros(shape)

print(f"Random Tensor: \n {rand_tensor} \n")
print(f"Ones Tensor: \n {ones_tensor} \n")
print(f"Zeros Tensor: \n {zeros_tensor}")
```

Out:

```
Random Tensor:
 tensor([[0.3904, 0.6009, 0.2566],
        [0.7936, 0.9408, 0.1332]])

Ones Tensor:
 tensor([[1., 1., 1.],
        [1., 1., 1.]])

Zeros Tensor:
 tensor([[0., 0., 0.],
        [0., 0., 0.]])
```

PYTORCH: BASICS

```python
# Get cpu, gpu or mps device for training.
device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)
print(f"Using {device} device")

# Define model
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10)
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = NeuralNetwork().to(device)
print(model)
```

```python
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)
```

```python
def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        # Compute prediction error
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        if batch % 100 == 0:
            loss, current = loss.item(), (batch + 1) * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
```

EXERCISES

WHAT IS A GOOD ML PROBLEM?

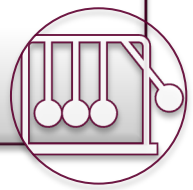
A ”good” ML problem has:

- A stable and predictable relation between input and output data.
- Strong and well-curated data that exemplify the problem with as much breadth as possible.
- Concrete, "objective" and measurable criteria of success for the training algorithm.
- A way to test the model in realistic situations.
- No better alternative solutions.

MLS FUNDAMENTAL LIMITATIONS

- ML only learns associations and has problems learning causal relationships
- Example: Google Flu Trends

Causality



- ML alone cannot develop new concepts and generally has problems following abstract systems
- Example: ChatGPT has problems calculating without a calculator

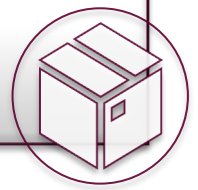
Abstraction

- ML has no built-in values nor any understanding of culture and ethics
- Example: LAPD predictive policing

Judgement

- ML models are black boxes and their outputs are difficult to impossible to meaningfully explain
- Example: Huskies wolves

Transparency



REFELCTION ON ML MODELING

Which choices did we make?

- Data source/representation
- Model/inductive bias
- Loss function
- Training method
- Test

Could we have made other choices? Which and how?

TEKST SOM DATA

Tekst → *sekvens* af *symboler*

- Struktureret af *grammatik*
- Men! Irregulært og inkonsekvent
- Natural language: Indeholder fejlstavninger, dårlig grammatik, slang, opfundne ord, etc.

Tekst på computere: Encodings

- ASCII (95 amerikanske tegn + nogle kontroltegn)
  - Fylder kun 7 bit per tegn
- Unicode (seneste version: 149813 tegn fra alle verdens større skriftsystemer + emojis, matematik, etc.)
  - UTF-8 kræver 8 til 32 bits per tegn (sorteret efter almindelighed)

The quick brown fox jumps over the lazy dog.

The quick brown fox jumps over the lazy dog.

The quick brown fox jumps over the lazy dog.

Article Adjective Adjective Noun Verb Preposition Article Adjective Noun

```
 !"#$%&'()*+,-./
0123456789:;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
`abcdefghijklmno
pqrstuvwxyz{|}~
```

HVORDAN ER TEKST REPRÆSENTERET I ML?

Vector encoding: Ord eller tegn?

- Engelsk: 26 tegn (minus emojis osv.), men ~170.000 forskellige ord!
- Men, grammatik er baseret på ord og ikke tegn, og mennesker læser ord og ikke tegn.
- Desuden: Character-level encoding gør sekvenserne meget længere og mindre meningsfulde.

Hvordan encoder man rækkefølge?

- Bag-of-words: smid rækkefølgen væk. Effektivt, men man mister al grammatik.
- One-hot encoding: gem rækkefølgen. Lossless, men meget ineffektivt.
- n-grams: grupper af ord som tokens. Kan få et allerede stort vokabular til at blive

`The brown fox` `the lazy dog`

Bag of words:

The	Brown	Fox	Lazy	Dog
1	1	1	0	0

The	Brown	Fox	Lazy	Dog
1	0	0	0	0

One-hot encoding:

The	Brown	Fox	Lazy	Dog
0	1	0	0	0

The	Brown	Fox	Lazy	Dog
0	0	1	0	0

The brown	Brown fox	The lazy	Lazy dog
1	1	0	0

Bigrams:

The brown	Brown fox	The lazy	Lazy dog
0	0	1	1

OPDELING AF TEKST: TOKENIZATION

For at lave et dataset til ML skal man først finde den rigtige repræsentation af teksten.

ML-modeller er ligeglade med tegn – de skal bare bruge vektorer.

Tokenization: Transformation af tekst til et vokabular af *tokens* (typisk heltal) som har gode egenskaber til det, man skal bruge det til.

- Moderne tokenizers: sub-word tokenization through byte-pair encoding.
- Som med alt andet i moderne ML: effektivitet er nøglen

```
1. aaabdaaabac

2. ZabdZabac
   Z=aa

3. ZYdZYac
   Y=ab Z=aa

4. XdXac
   X=ZY Y=ab Z=aa
```

Byte-pair encoding

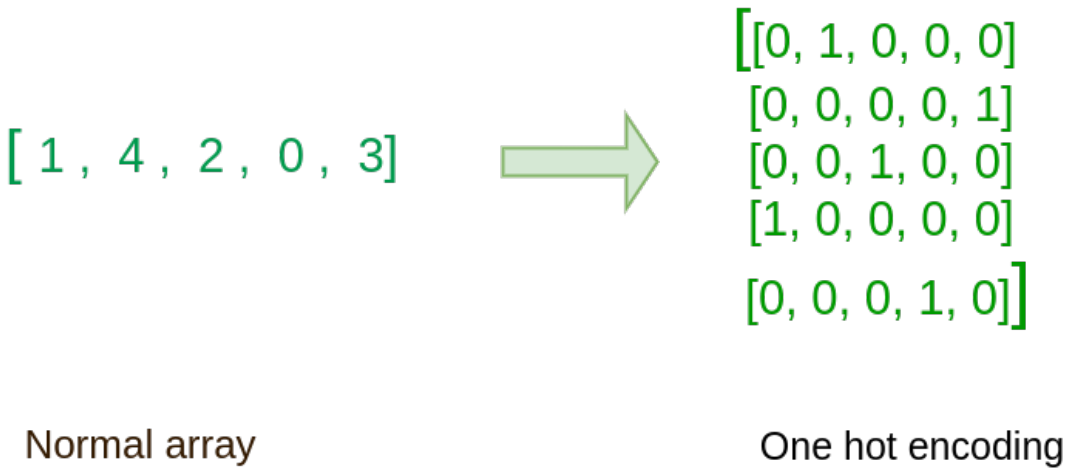
n-grams:	*token*	*izer*	*:*	*texts*	*->*	*series*	*of*	*numerical*	*"*	*t*	*ok*	*ens*	*"*
numbers as "tokens":	30001	7509	25	13399	4613	2168	286	29052	366	83	482	641	1

Eksempler fra Wikipedia

TEKST OG TENSORER

Neural networks forventer data som *tensorer*.

- One-hot vektor og BoW er allerede tensorer, men de har hver deres ulempe.
  - One-hot fylder alt for meget og har en uafhængig basis
  - BoW smider alt sekventiel information og dermed grammatik væk
- BoW er en 1 x *v* tensor (en vektor), hvor *v* er vocabulary size.
- One-hot er en *n* x *v* tensor, hvor n er længden på sekvensen.
- Kan det gøres bedre?



ET MINDRE VEKTORRUM: EMBEDDINGS

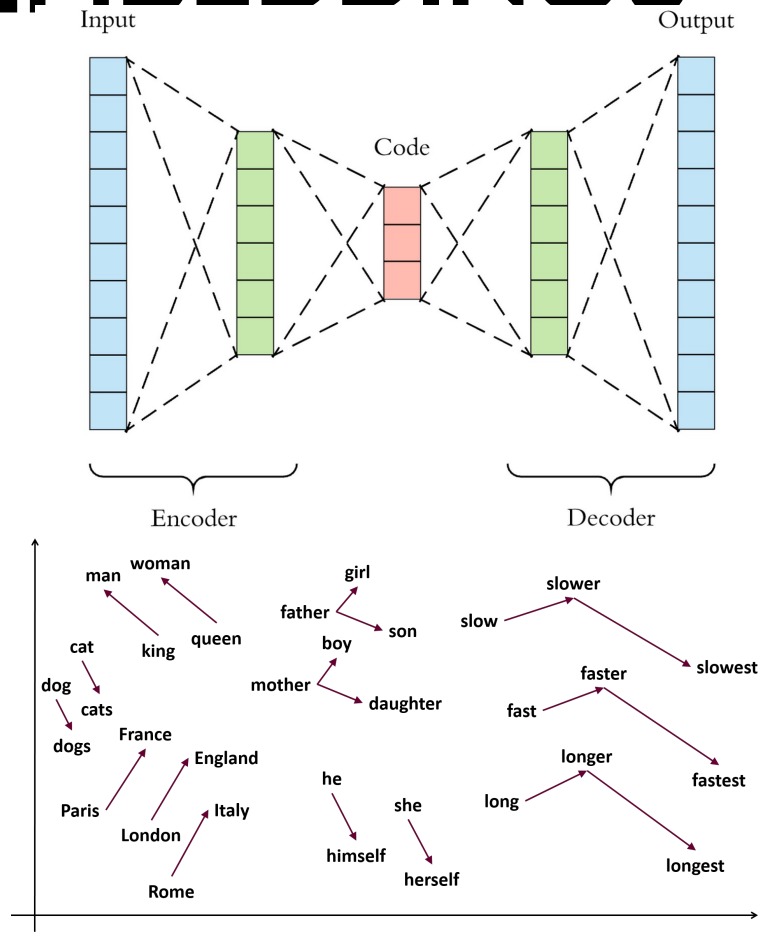
Man kan træne et netværk med en *encoder* (og evt. en *decoder*) arkitektur til at lære gode lav-dimensionelle *embeddings* af høj-dimensionelle vektorer.

Word-embeddings har været populære længe med word2vec, som forsøger at bruge lære en repræsentation af ord ud fra den kontekst, de typisk optræder i (typisk baseret på en slags n-grams eller skip-grams).

Man kan lære embeddings af alle typer data!

Fordele:

- Man får en *kompakt* repræsentation af sin data, hvor vigtige *semantiske forhold* prioriteres i repræsentationen.



DATA AUGMENTATION

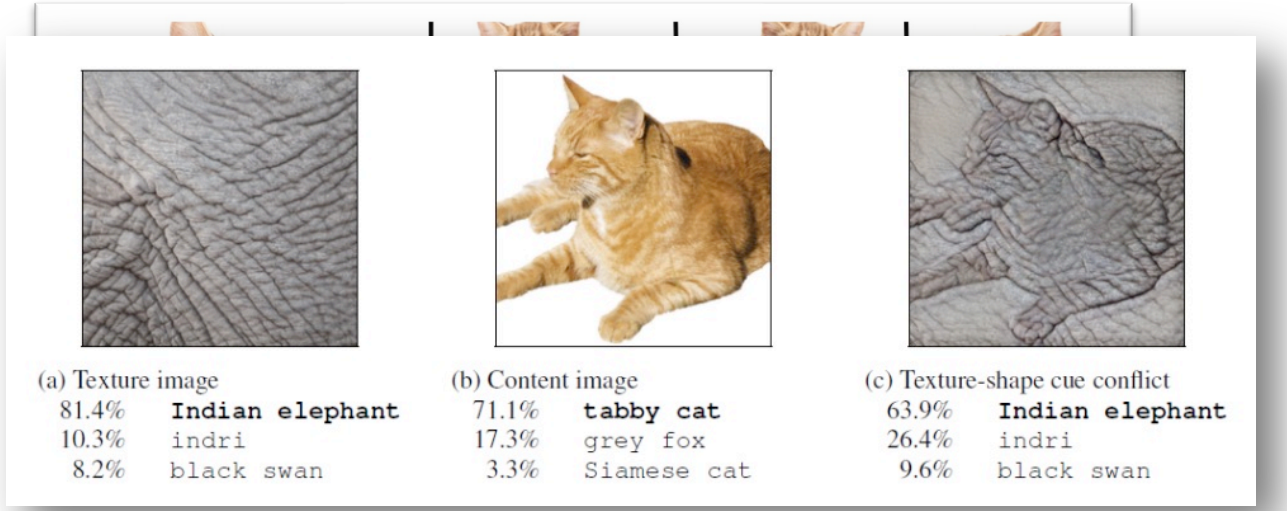
If we have knowledge of input-output relation we might be able to make new training data from existing training data.

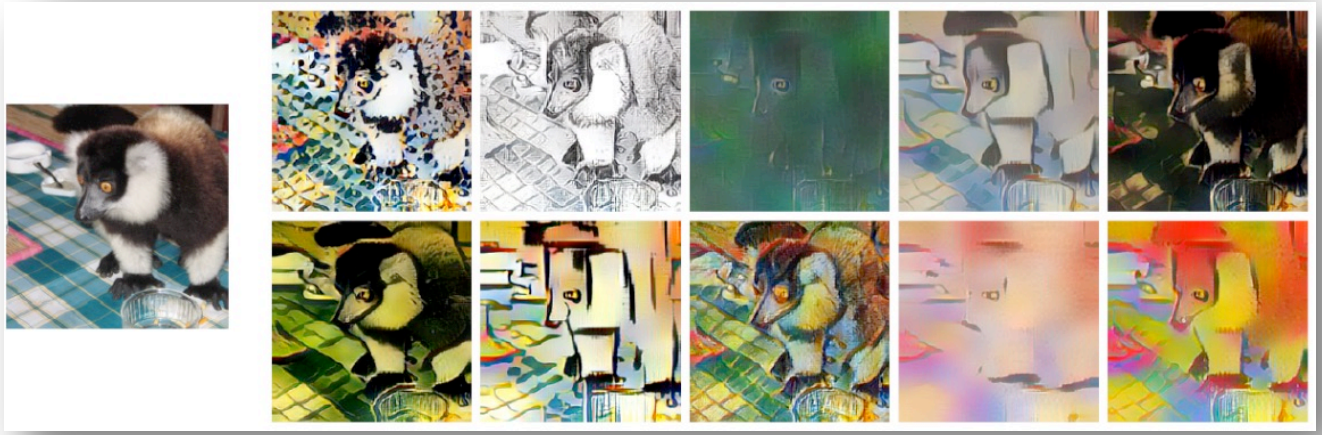
Common sense: rotated pictures of cats are still pictures of cats.

→ We can encode invariance to particular transformations into our data and thereby into our model.

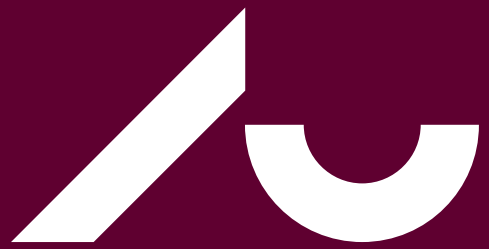
We can also get creative: texture bias and shape bias.

→ Out-of-sample and *out-of-distribution* data.





Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F. A., & Brendel, W. (2018). ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. *arXiv preprint arXiv:1811.12231*.



SCHOOL OF CULTURE AND