Deep Learning

Outline

- Introduction
- Artificial neural networks
- Backpropagation
- Convolutional neural networks (CNN)
- Recurrent neural networks (RNN)
- Autoencoders (AE)
- Generative adversarial networks (GAN)
- Deep Q-network

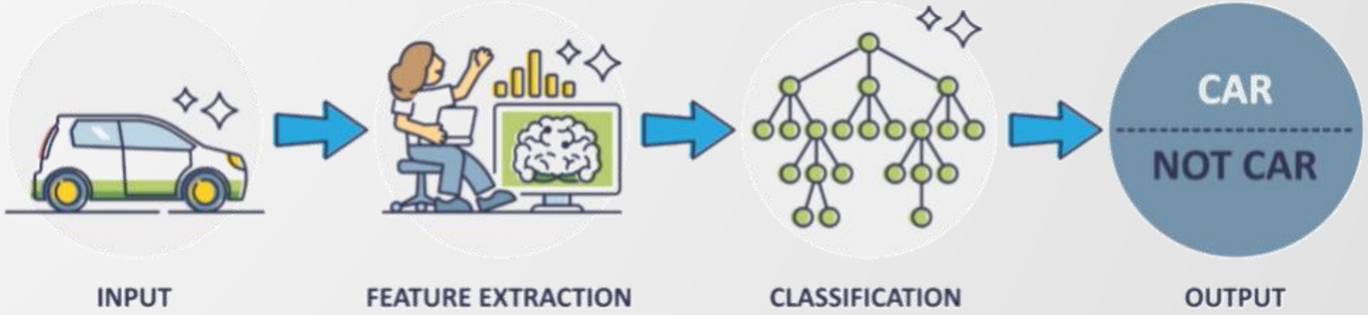
Introduction

> “*Deep Learning (DL) is a ML technique that constructs artificial neural networks to mimic the structure and function of the human brain.*”

From DeepAI

Deep learning

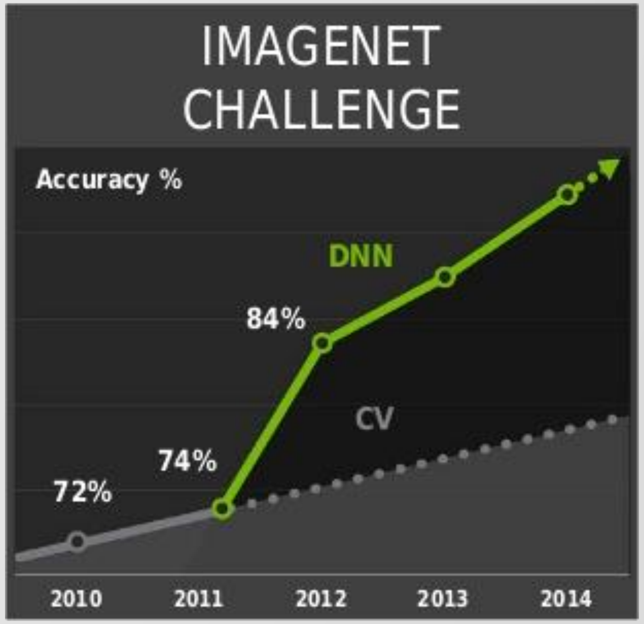
- In traditional ML techniques, raw data are analyzed by a domain expert to identify robust features to reduce the complexity and make patterns more visible to learning algorithms.

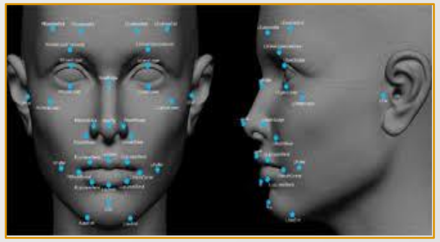


- DL uses a large number of hidden layers to extract features from raw data and transform them into different levels of abstraction (representations).

Deep learning (2)

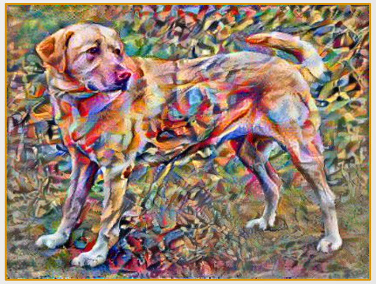
- Since 2012, DL techniques have overcome traditional ML techniques in many application areas:
  - Object detection and localization (e.g., Yolo)
  - Face Recognition, Pedestrian Detection, Traffic Sign Detection
  - Autonomous Car (e.g., PilotNet) and Drones (e.g., TrailNet)
  - Speech Recognition, Language Translation
  - Natural Language Processing
  - Recommendation systems
  - Arts (e.g., Deep Dream, Style Transfer)

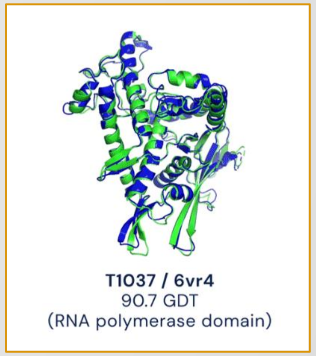


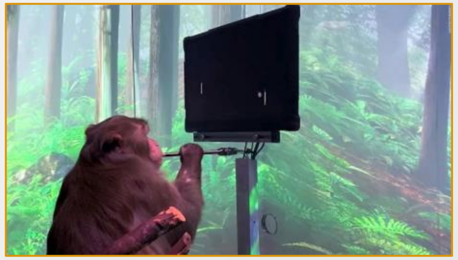


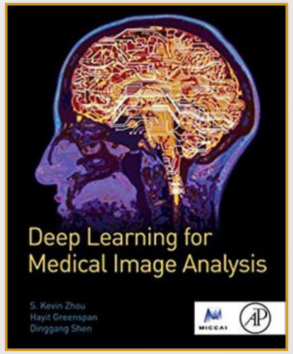
Deep learning (3)

- Image Generation (Stable Diffusion)
- Medical Image analysis (e.g., CheXnet)
- Protein folding (Alpha fold)
- Brain implants (e.g., Neuralink)







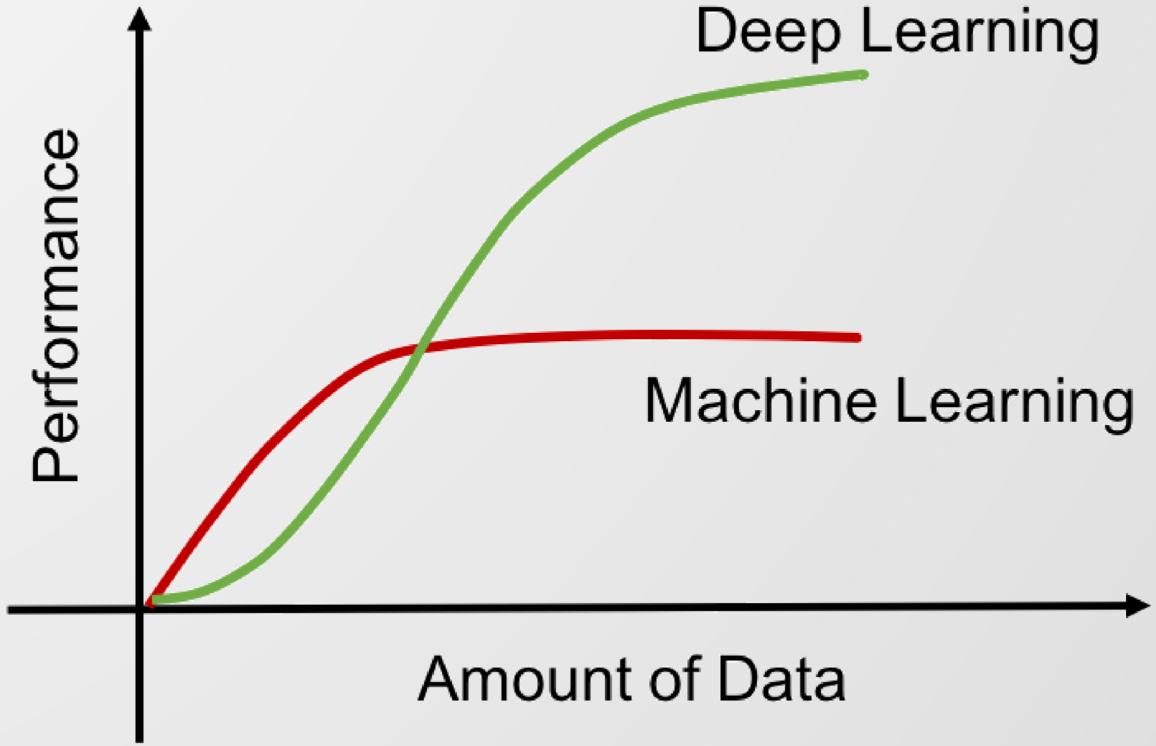


What is changed? Why now?

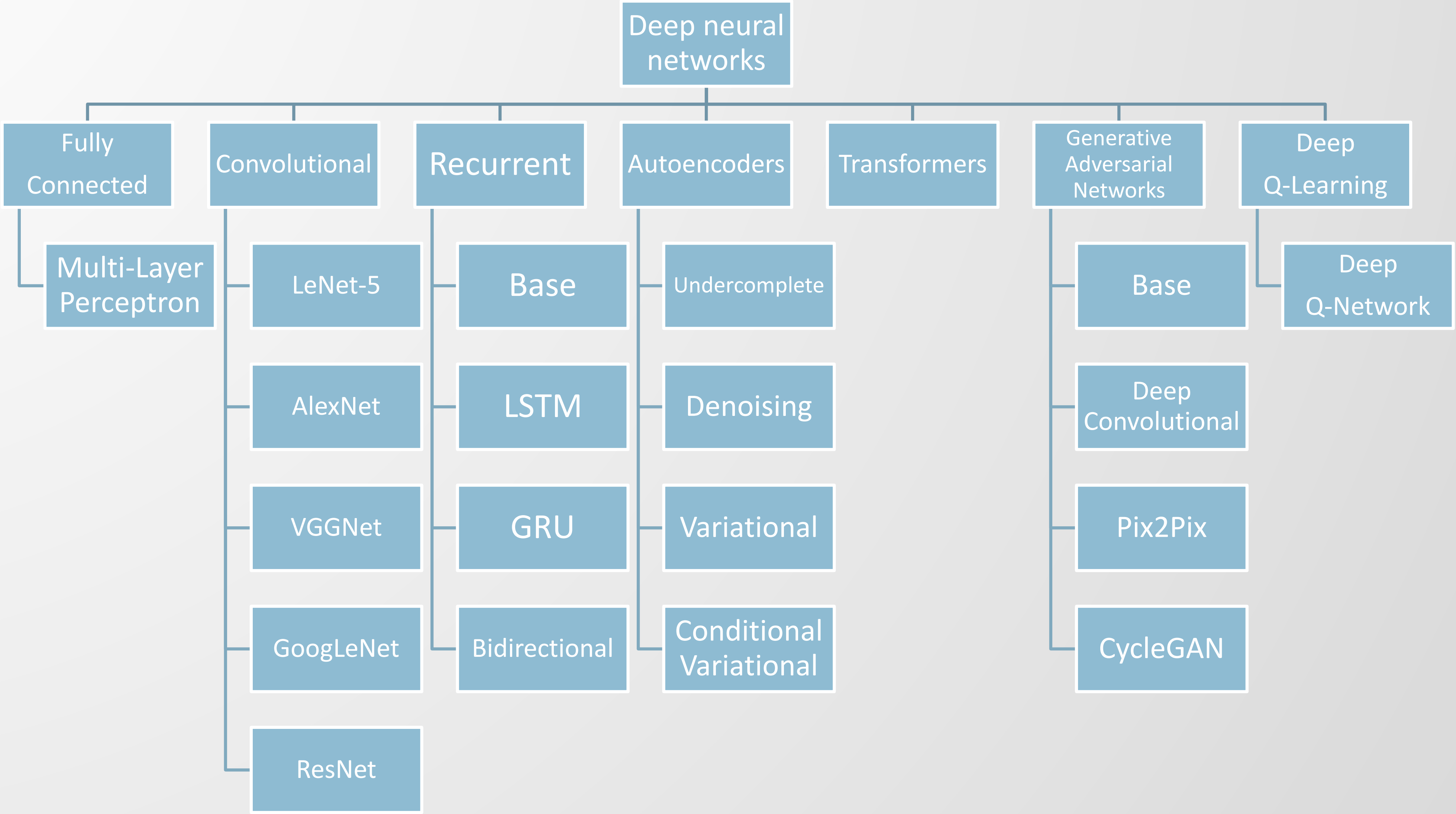
- Many of the core concepts of DL were well known from the end of the last century.
- Why did not DL approaches replace traditional ML techniques for more than ten years?
- What happened that changed things?
- Though there are many factors, the two most crucial components appear to be:
    - appearance of large, high-quality labeled datasets;
    - massively parallel computing with GPUs.

Why does DL require large amount of data?

- Because DL models contains millions (or even billion) of trainable parameters and they need to see a proportional amount of examples to get good performance.



DL model categories



- Deep neural networks
  - Fully Connected
    - Multi-Layer Perceptron
  - Convolutional
    - LeNet-5
    - AlexNet
    - VGGNet
    - GoogLeNet
    - ResNet
  - Recurrent
    - Base
    - LSTM
    - GRU
    - Bidirectional
  - Autoencoders
    - Undercomplete
    - Denoising
    - Variational
    - Conditional Variational
  - Transformers
  - Generative Adversarial Networks
    - Base
    - Deep Convolutional
    - Pix2Pix
    - CycleGAN
  - Deep Q-Learning
    - Deep Q-Network

DL application areas

Visual recognition

Natural language processing

Virtual assistants

Personalization

Fraud detection

Healthcare

Handwriting generation

Self driving car

Machine translation

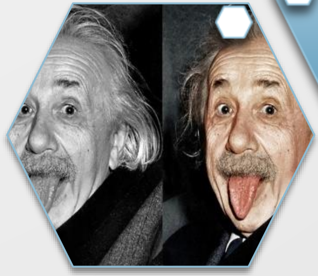
DL application areas (2)

Image captioning

Image – language translation

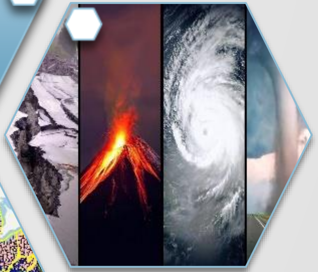
Pixel restoration

Image colorization



Demographic predictions

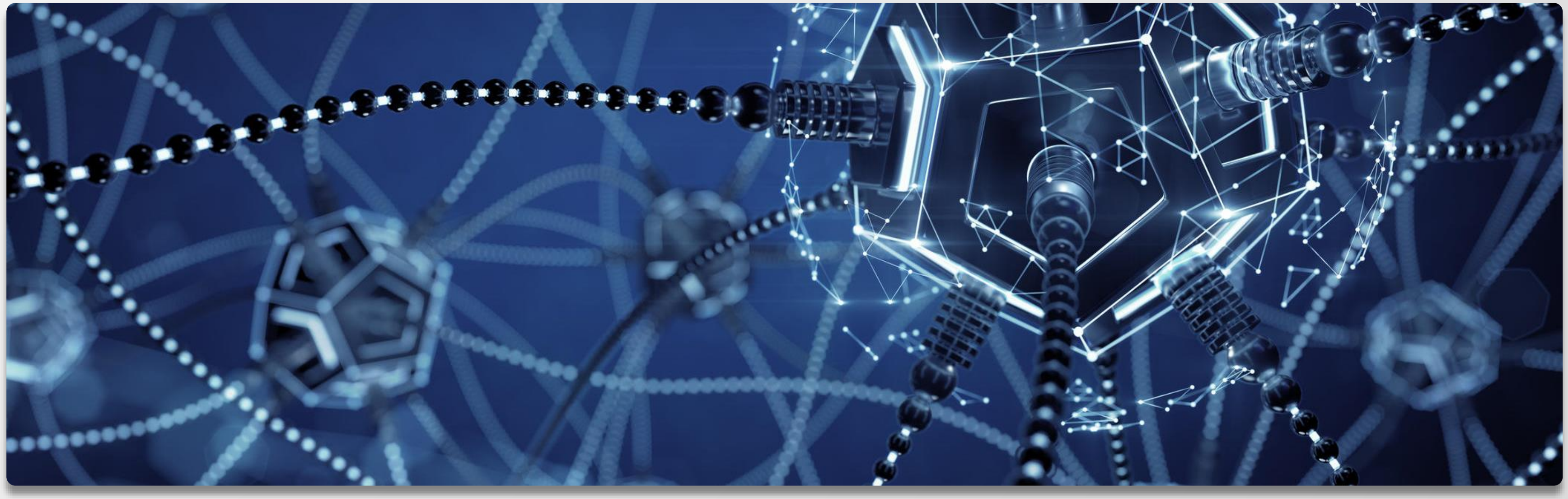
Predicting natural disasters



News analysis

Game playing

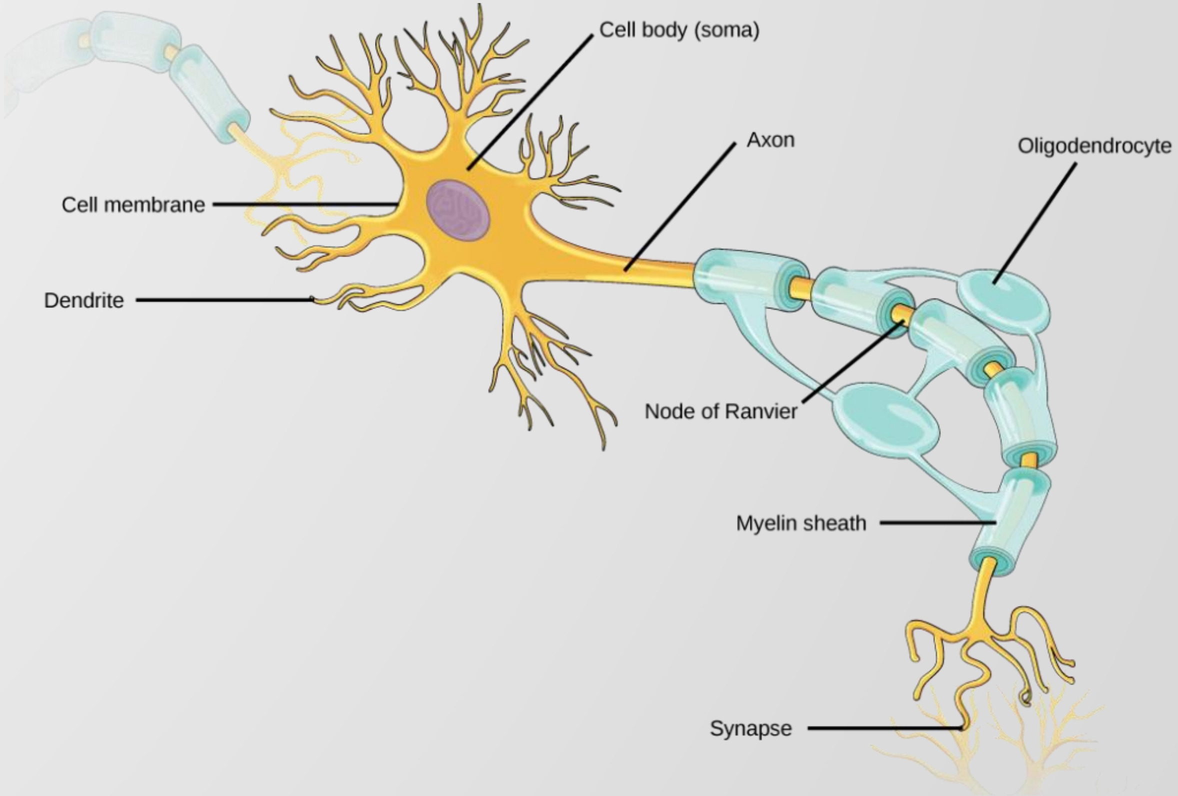
Artificial neural networks



> “*Artificial neural networks paradigm is inspired by the way the biological nervous system processes information. It is composed of large number of highly interconnected processing elements working in unison to solve a specific problem.*”

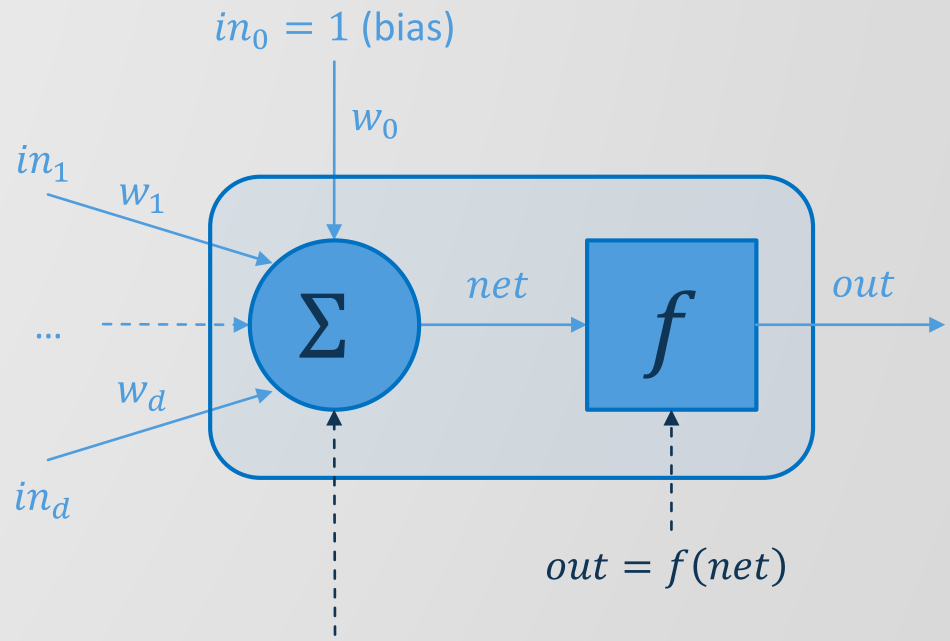
Biological neurons

- Biological neurons are the fundamental units of the brain and nervous system.
- A neuron is formed from four basic parts:
  - the dendrites collect incoming signals (inputs);
  - the soma processes the incoming signals over time and converts the processed value into an output;
  - the axon works as a transmission line;
  - at the end of the axon there are the synapses that are connected to other neurons to transmit the output signal.



Artificial neuron

- The artificial neuron model, firstly proposed by McCulloch and Pitts in 1943, has been designed to mimic the behavior of biological neurons:
  - it receives one or more inputs and sums them to produce an output (or activation);
  - each input is separately weighted, and the sum is passed through a non-linear function (called activation function).

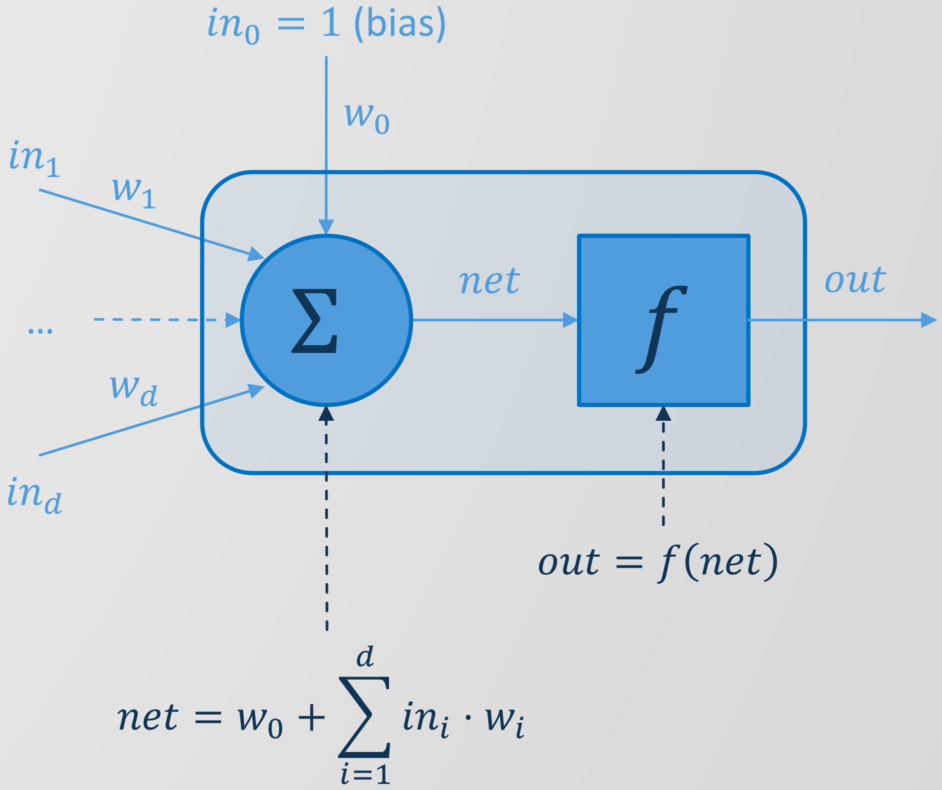


$$out = f(net)$$

$$net = w_0 + \sum_{i=1}^{d} in_i \cdot w_i$$

Artificial neuron (2)

- $in_1, \cdots, in_d$ are the neuron inputs.
- $w_1, \cdots, w_d$ are weights assigned to each input.
- w_0 (called bias) allows to shift the activation function by adding a constant to the input. Bias is used to delay the triggering of the activation function.
- As first step, the neuron computes a weighted sum (net) of all its inputs.
- net is passed into the activation function (f) to compute the output (or activation) of the neuron (out).



$$net = w_0 + \sum_{i=1}^{d} in_i \cdot w_i$$

Brain

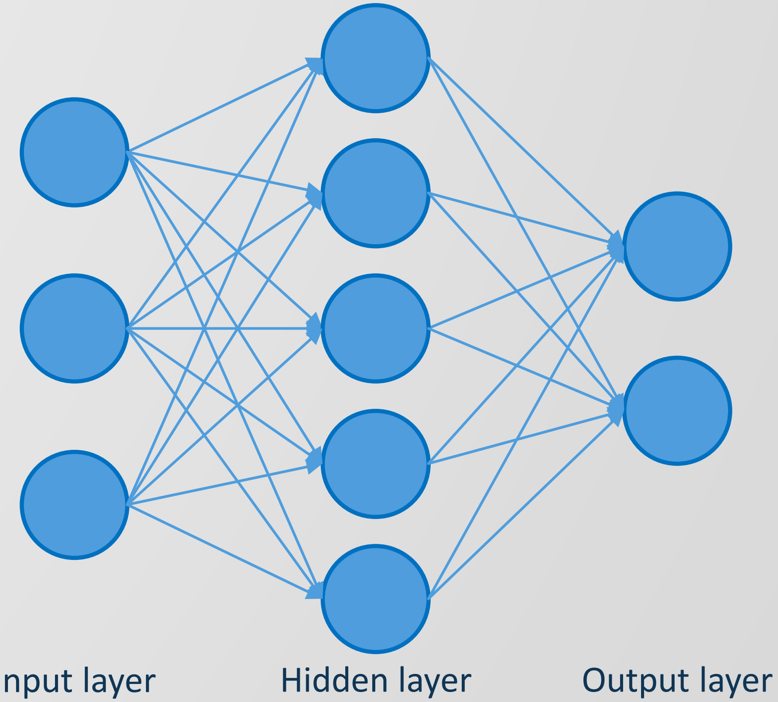
- A single biological neuron is a weak element but connected with billions of other neurons become a powerful network called brain.
- The human brain contains about 100 billion (10^{11}) neurons that communicate by electric and chemical signals through more than a 100 trillion (10^{14}) synapses (connections).

Artificial neural networks

- Similar to the brain, an Artificial Neural Network (ANN) is made up of artificial neurons connected to each other.
- Each connection (called edge), like the synapses in a biological brain, can transmit a signal to other neurons.
- The weight associated to each connection increases or decreases the strength of the signal.
- Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs.
- Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

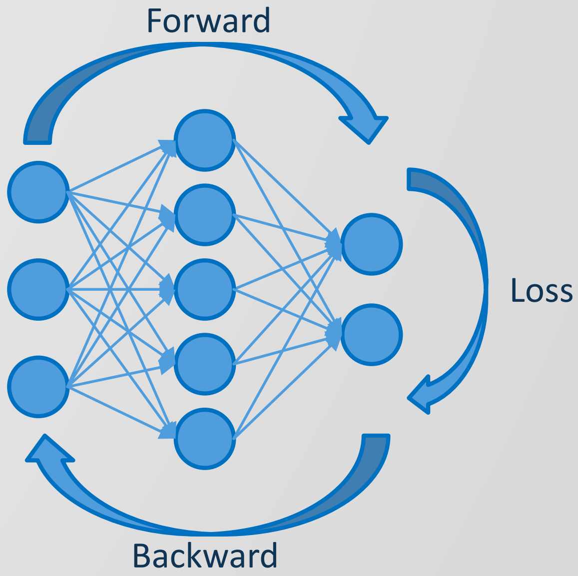
Artificial neural networks (2)

- A Feed-Forward Neural Network (FFNN) is an ANN where connections between neurons do not form a cycle.
- Multi-Layer Perceptron (MLP) is the most common FFNN consisting of three or more layers:
  - an input layer;
  - one or more hidden layers;
  - an output layer.
- MLPs are fully-connected:
  - each neuron in one layer is connected with every neuron in the following layer.

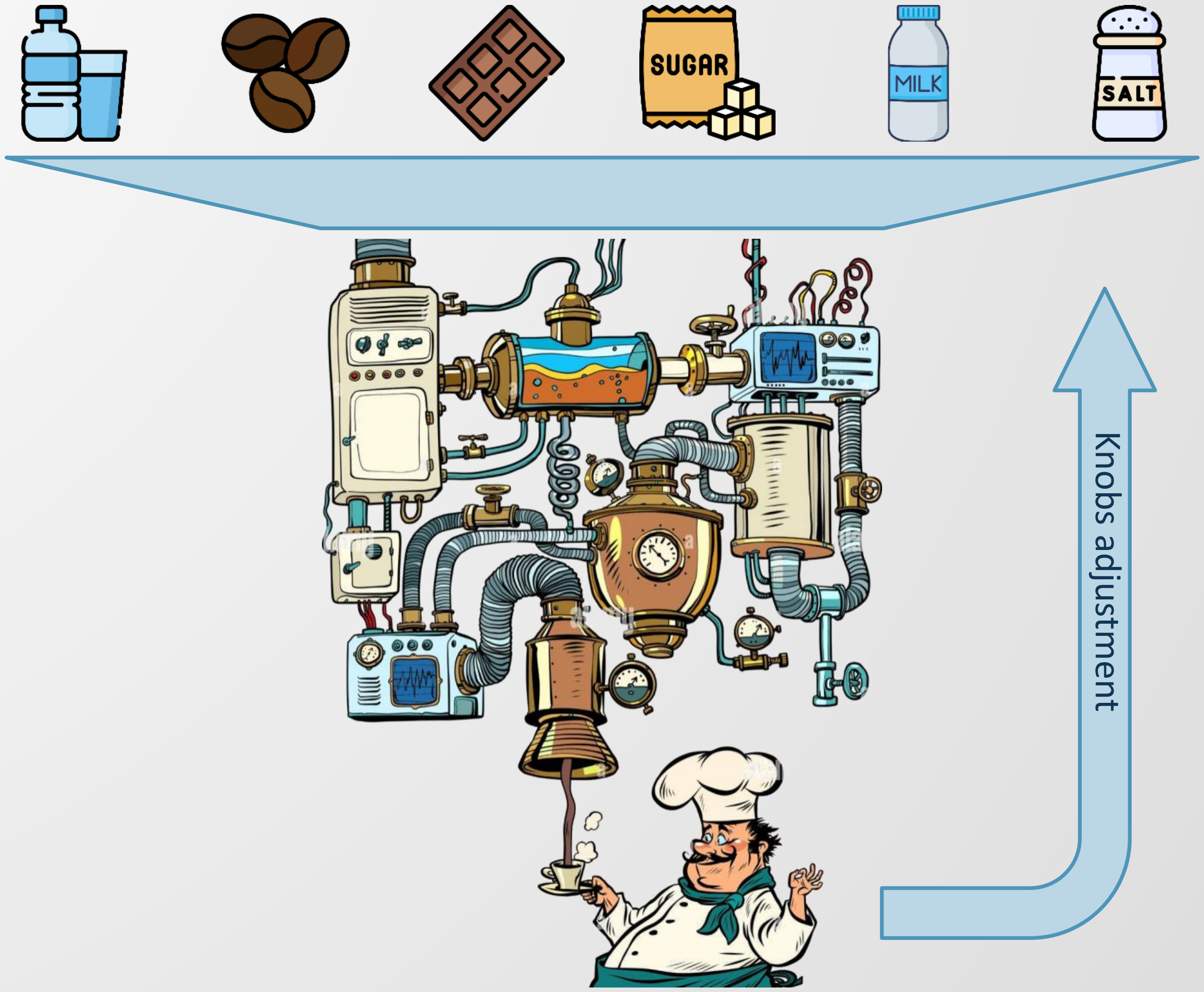


The learning process

- The learning process is a key feature of ANNs and it is closely related to how the human brain learn.
- Iteratively, the training data are presented to the network (forward), then the weights are adjusted (backward) on the basis of how similar the values returned by the network are compared to the desired ones (loss).
  - After all cases are presented, the process often starts over again.
  - During the learning phase, the weights are adjusted to improve the performance on the training data.

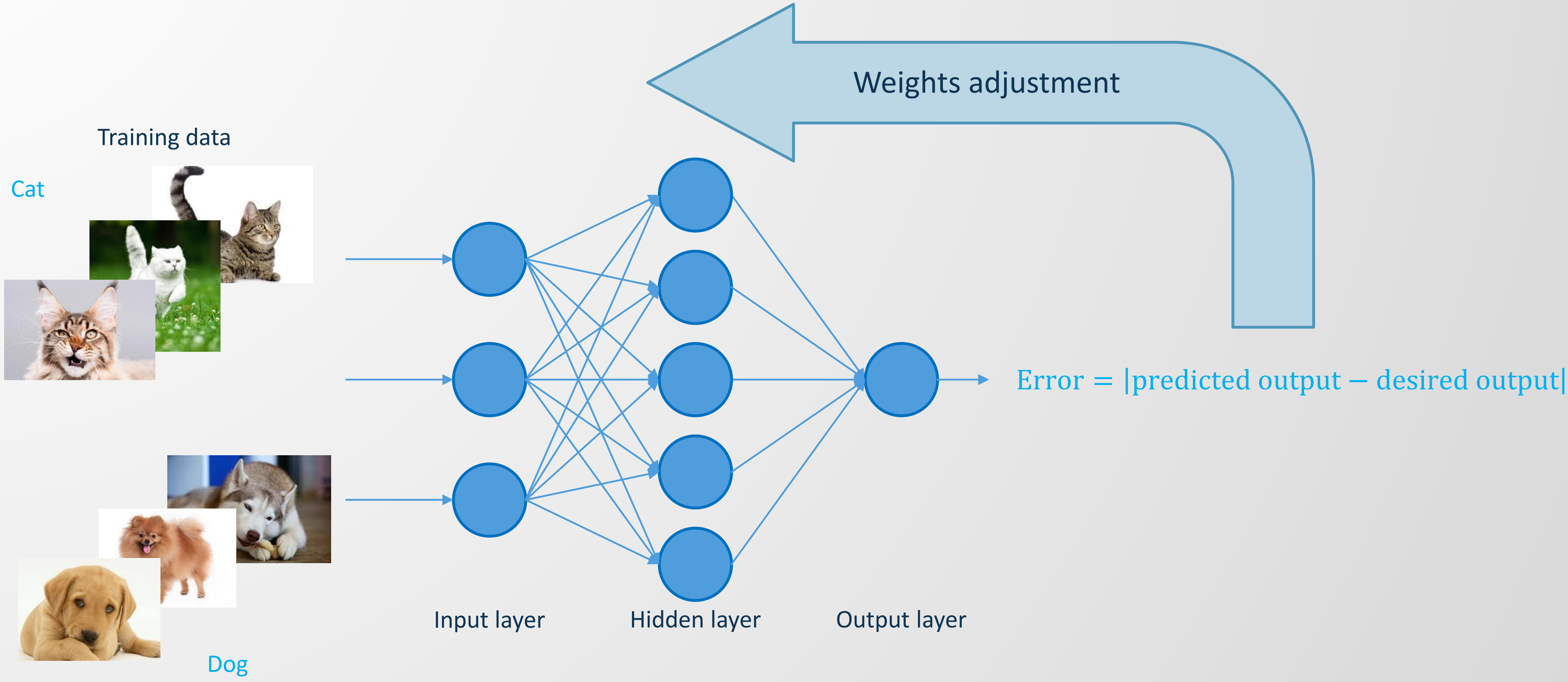


The learning process (2)



Knobs adjustment

The learning process (3)



Weights adjustment

Training data

Cat

Dog

Input layer

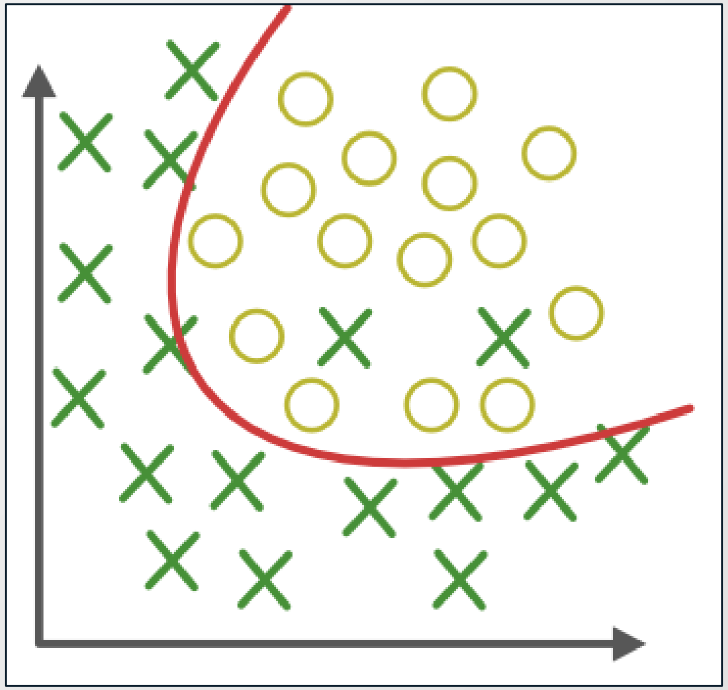
Hidden layer

Output layer

$$\text{Error} = |\text{predicted output} - \text{desired output}|$$

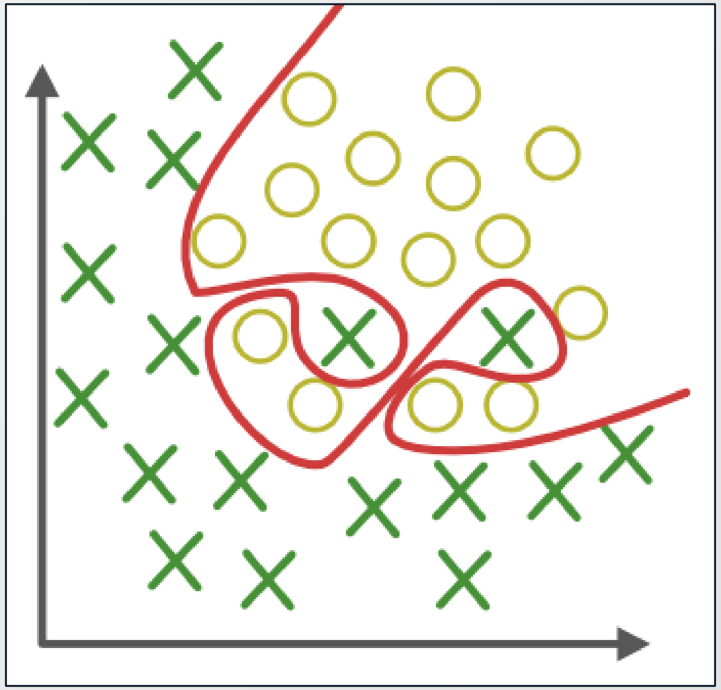
Overfitting and underfitting

- The goal of a good machine learning model is to generalize well from the training data to any data from the problem domain. This allows to make predictions on data the model has never seen.



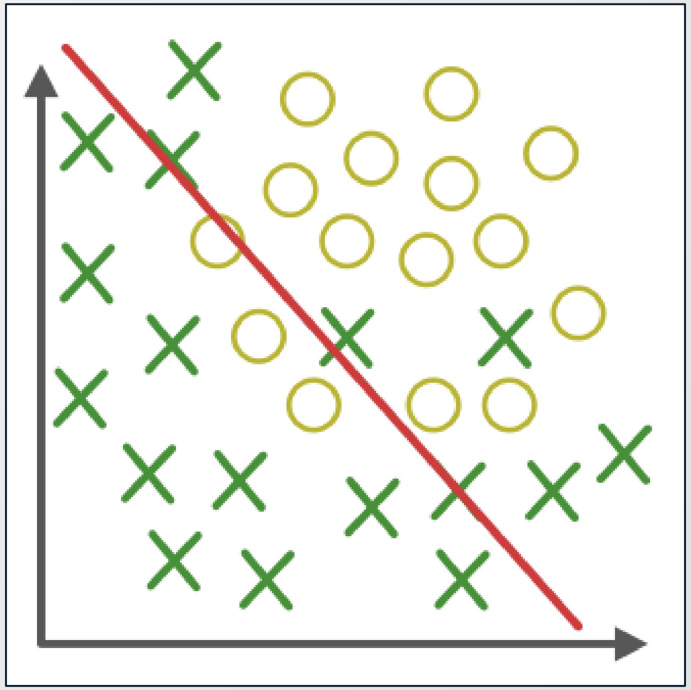
Overfitting and underfitting (2)

- The danger when working with finite training samples is to discover apparent associations not present in the underlying population from which our training set was drawn.
- The phenomenon of fitting the training data more closely than the underlying distribution is called *overfitting*.



Overfitting and underfitting (3)

- Vice versa, *underfitting* refers to a network that can neither model the training data nor generalize to new data.



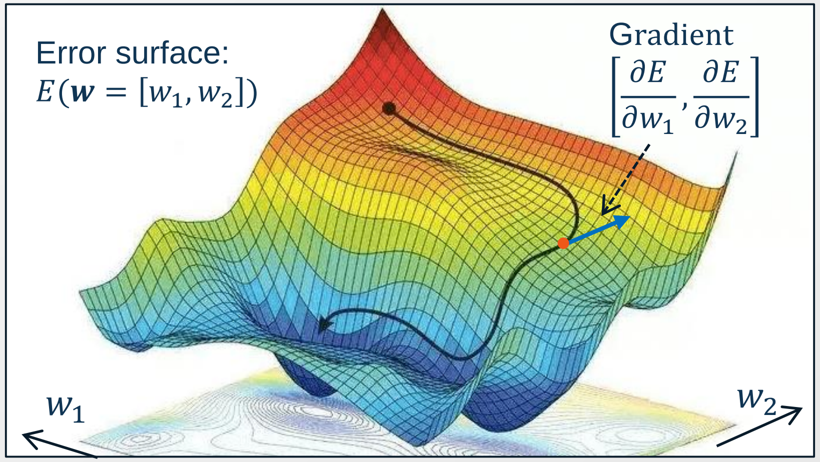
Overfitting and underfitting – solutions

- There are two ways to approach overfitting:
  - increase the size of the training set;
  - reduce the complexity of the network.
- Underfitting can be avoided by:
  - increasing the complexity or the type of the model;
  - increasing the training time to minimize the cost function.

Backpropagation

Backpropagation

- Backpropagation algorithm is probably the most fundamental component of an Artificial Neural Network (ANN).
- After each forward pass through a network, backpropagation performs a backward pass while adjusting the model's parameters (weights and biases).
- The parameters are updated by computing gradients of expressions through automatic differentiation and recursive application of chain rule.



Old weight

$$w' = w - \eta \cdot \frac{\partial E}{\partial w}$$

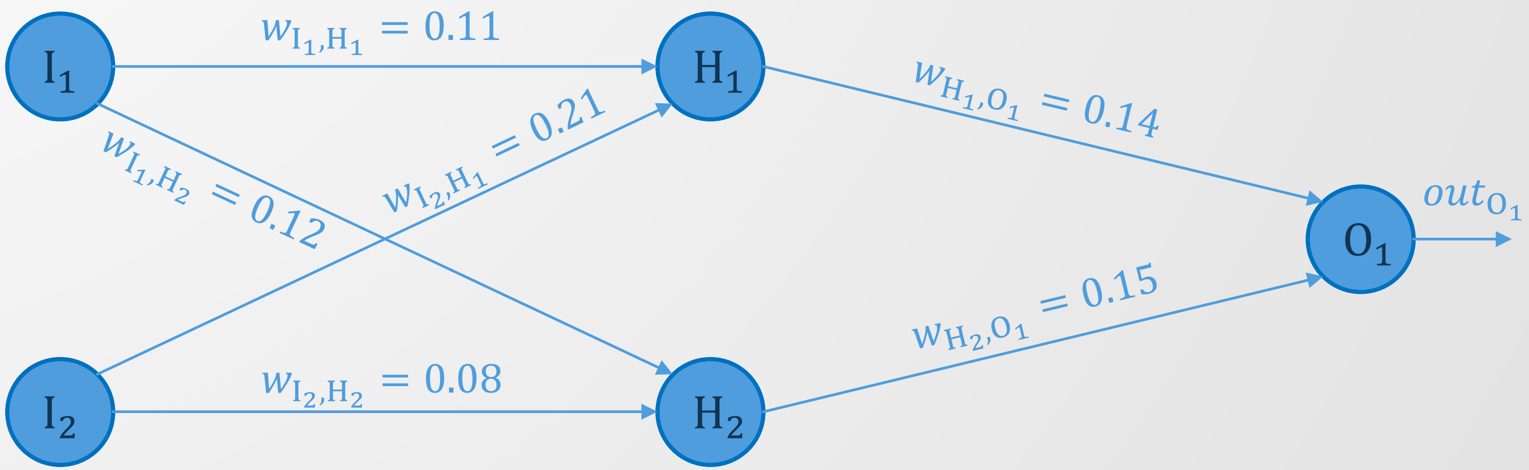
Partial derivative of the loss function with respect to weight w

New weight

Learning rate

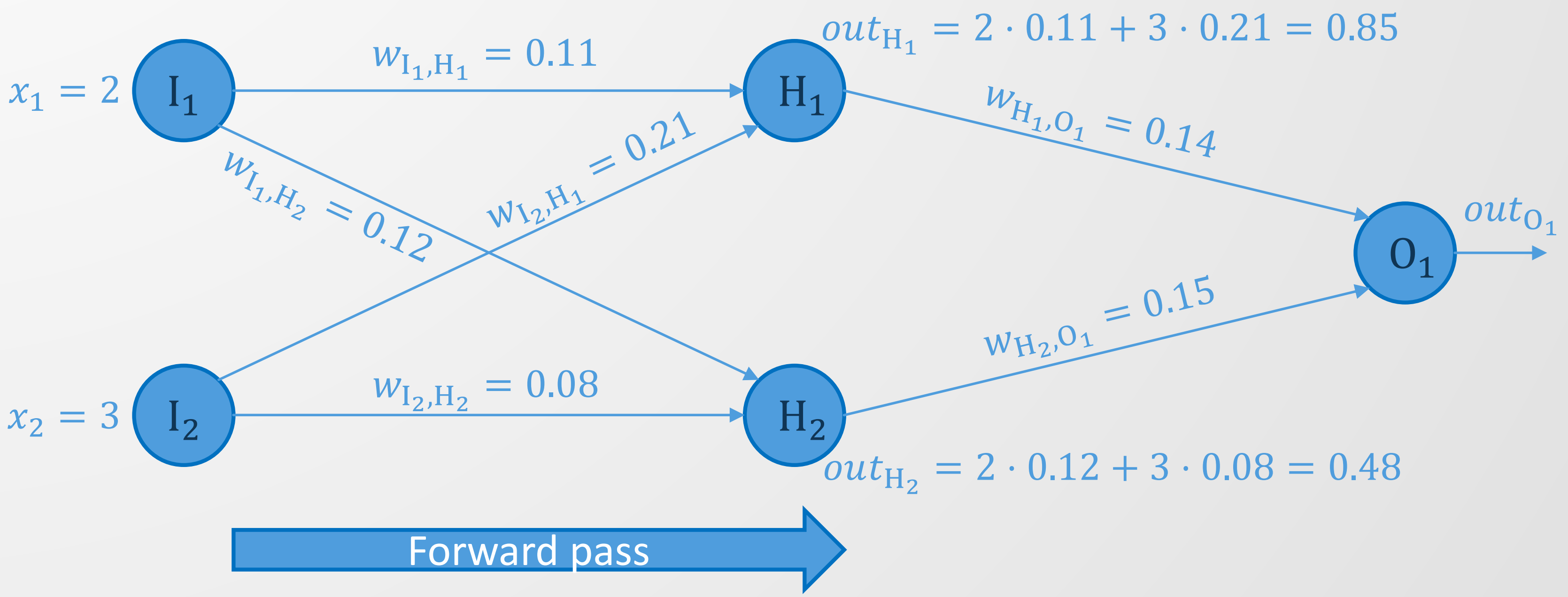
Backpropagation on an ANN – an example

- Given the following ANN:

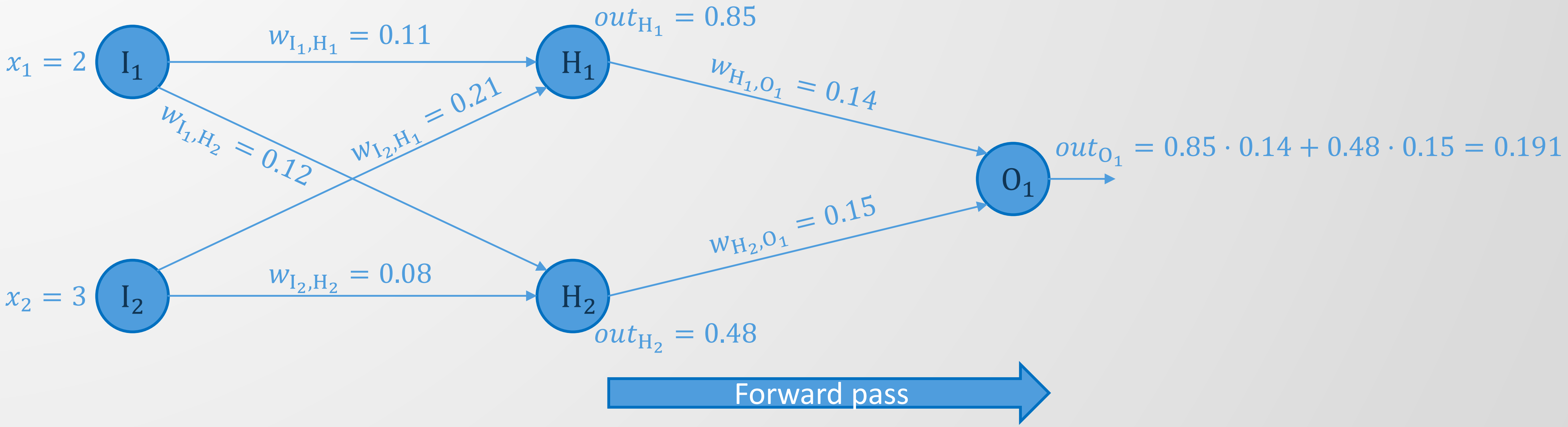


Backpropagation on an ANN – an example (2)

- Consider the inputs $x_1 = 2$, $x_2 = 3$, the desired output $y = 1$, and the identity function as activation function.



Backpropagation on an ANN – an example (3)



$$out_{\mathrm{O}_1} = 0.85 \cdot 0.14 + 0.48 \cdot 0.15 = 0.191$$

Backpropagation on an ANN – an example (4)

- We have to reduce the error (E) between the desired and the predicted outputs.
- The most used loss function for the regression task is the Square Error (SE):

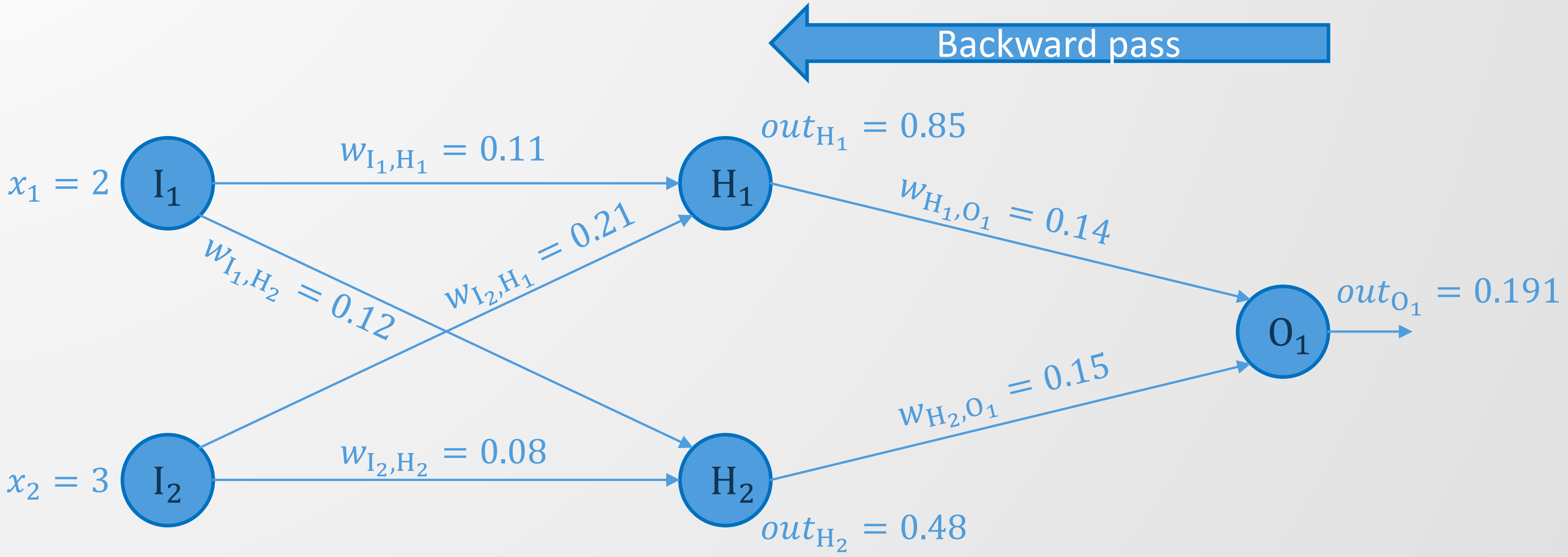
$$\mathrm{SE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$$

where $\mathbf{y}$ and $\hat{\mathbf{y}}$ are the desired and the predicted outputs, respectively.

$$\mathrm{SE} = (1 - 0.191)^2 = 0.654$$

- We use backpropagation formula to update weights: $w' = w - \eta \cdot \frac{\partial E}{\partial w}$ with $\eta = 0.05$.

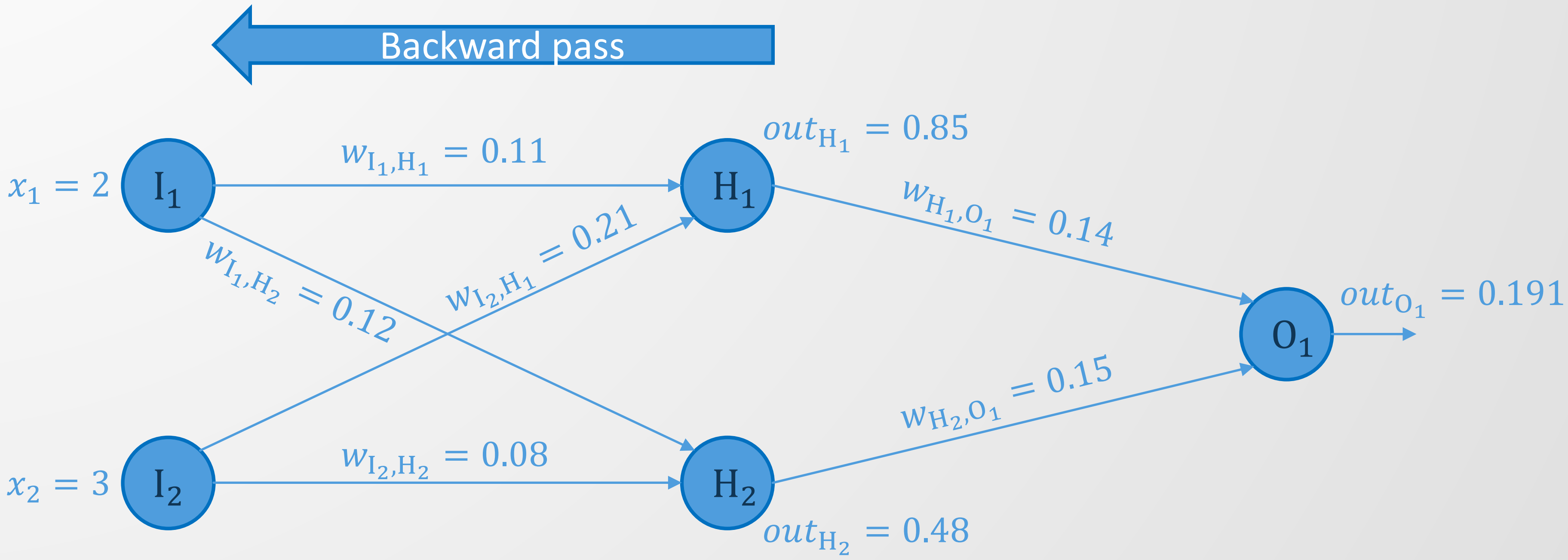
Backpropagation on an ANN – an example (5)



$$w'_{H_1,O_1} = 0.14 - 0.05 \cdot (-1.38) = 0.14 + 0.07 = 0.21$$

$$w'_{H_2,O_1} = 0.15 - 0.05 \cdot (-0.78) = 0.15 + 0.04 = 0.19$$

Backpropagation on an ANN – an example (6)



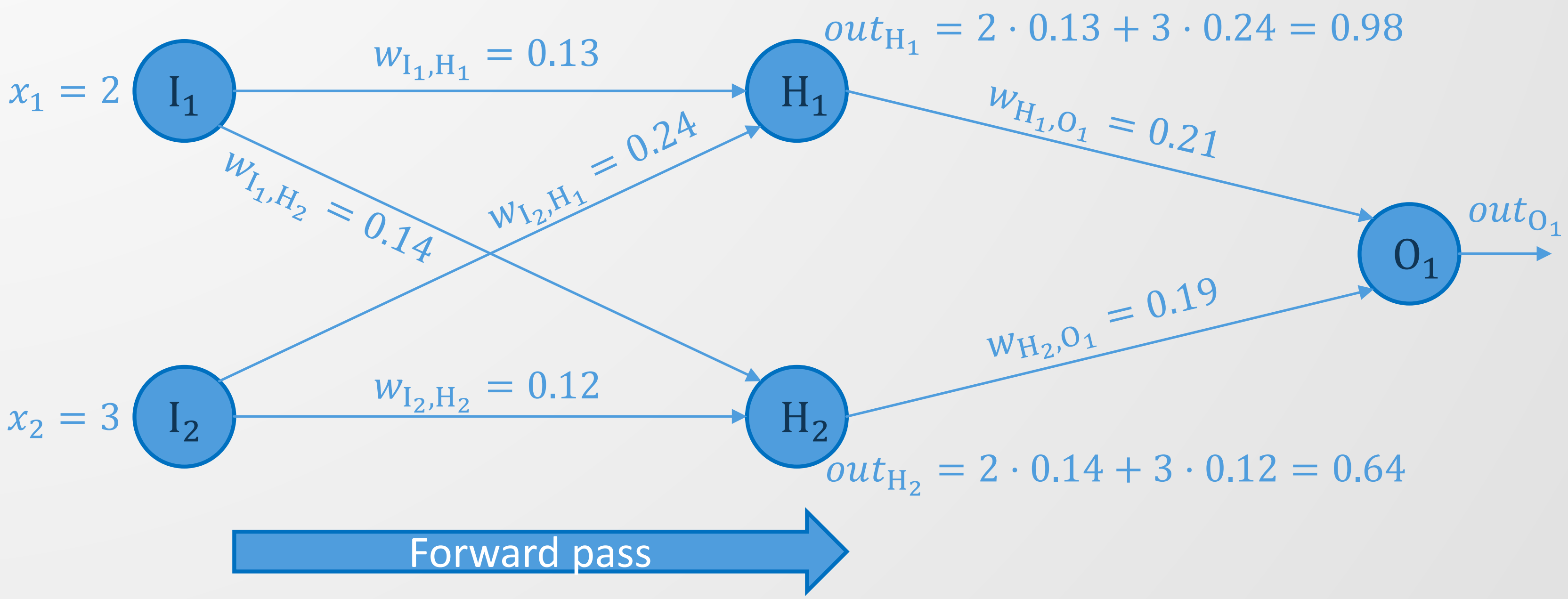
$$w'_{\mathrm{I}_1,\mathrm{H}_1} = 0.11 - 0.05 \cdot (-0.45) = 0.11 + 0.02 = 0.13$$

$$w'_{\mathrm{I}_2,\mathrm{H}_1} = 0.21 - 0.05 \cdot (-0.68) = 0.21 + 0.03 = 0.24$$

$$w'_{\mathrm{I}_1,\mathrm{H}_2} = 0.12 - 0.05 \cdot (-0.49) = 0.12 + 0.02 = 0.14$$

$$w'_{\mathrm{I}_2,\mathrm{H}_2} = 0.08 - 0.05 \cdot (-0.73) = 0.08 + 0.04 = 0.12$$

Backpropagation on an ANN – an example (7)



$x_1 = 2$

$x_2 = 3$

$w_{I_1,H_1} = 0.13$

$w_{I_1,H_2} = 0.14$

$w_{I_2,H_1} = 0.24$

$w_{I_2,H_2} = 0.12$

$out_{H_1} = 2 \cdot 0.13 + 3 \cdot 0.24 = 0.98$

$out_{H_2} = 2 \cdot 0.14 + 3 \cdot 0.12 = 0.64$

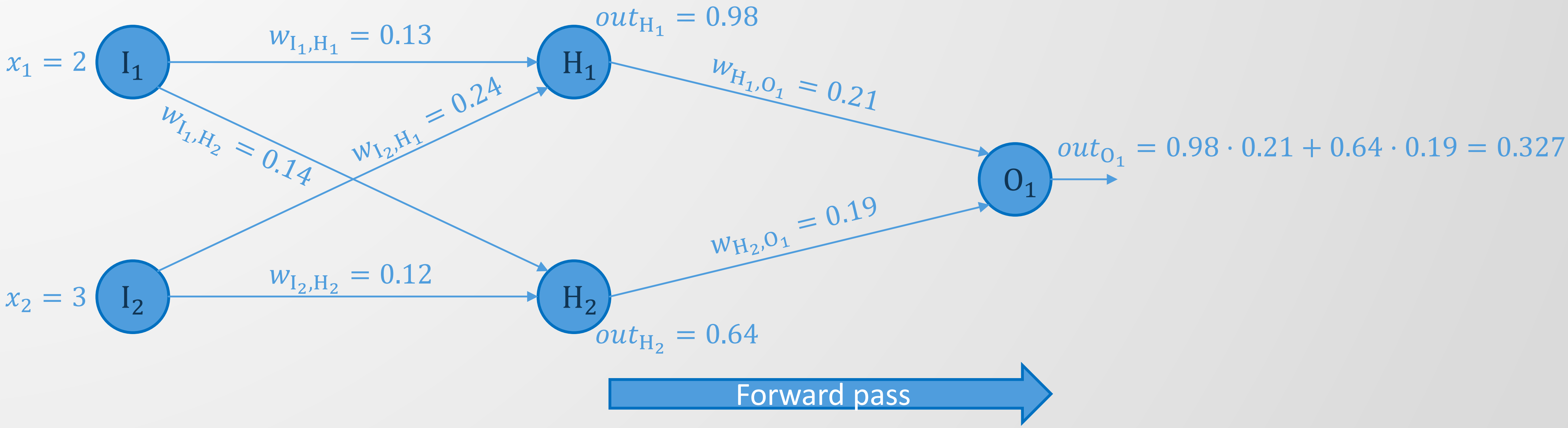
$w_{H_1,O_1} = 0.21$

$w_{H_2,O_1} = 0.19$

out_{O_1}

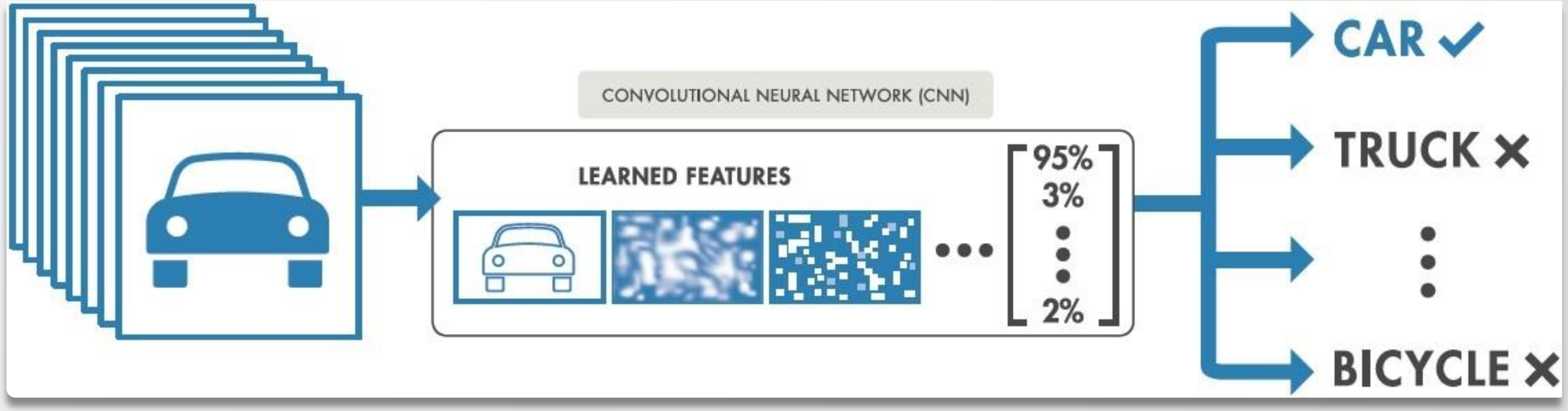
Forward pass

Backpropagation on an ANN – an example (8)



$$\mathrm{SE} = (1 - 0.327)^2 = 0.453\ (< 0.654)$$

Convolutional neural networks



> *A Convolutional Neural Network (CNN) is a deep learning neural network designed for processing structured arrays of data such as images.*

From DeepAI

Applications

- Nowadays, CNNs are used to solve several computer vision problems including:
  - Identity recognition
  - Image classification
  - Object detection
  - Scene labeling
  - Visual search
  - Action recognition
  - Document analysis
  - Anomaly detection
  - Video analysis

Problem with traditional neural networks

- There are several drawbacks when common Artificial Neural Networks (ANNs), such as MultiLayer Perceptron (MLP), are used for image processing:
  - MLPs use one neuron for each input. The amount of weights rapidly becomes unmanageable for large images.

Flattened into
$600 \cdot 400 \cdot 3 = 720000$
input neurons

5 hidden
neurons

600x400 pixels
3 color channels (RGB)

$720000 \cdot 5 = 3.6$
millions weights
(for a single layer)

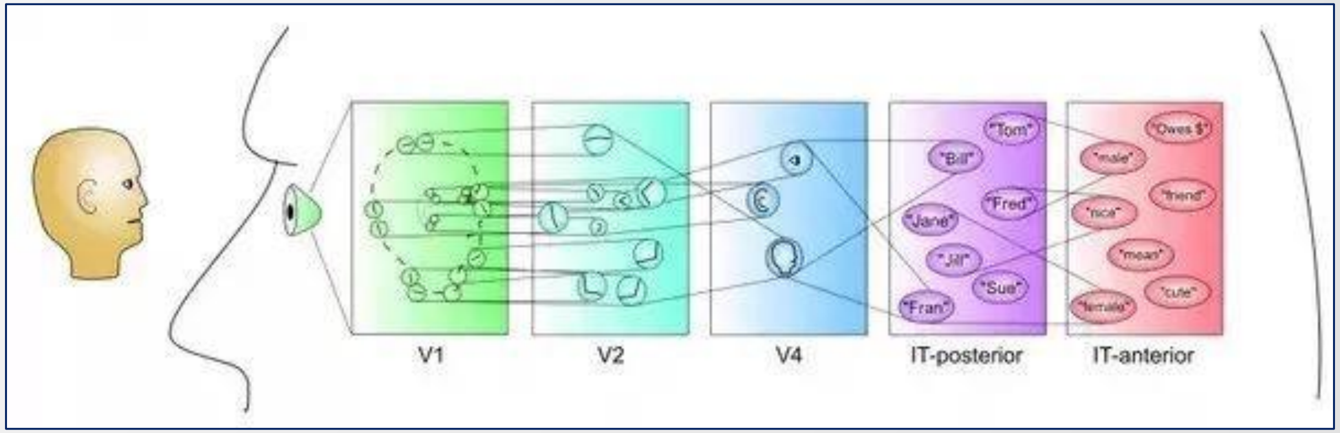
Problem with traditional neural networks (2)

- MLPs react differently to an image and its shifted version because they are not translation invariant.

- <u>Most important:</u> spatial information is lost when the image is flattened into an MLP. Pixels that are close together are important because they help to define the features of an image.

Visual cortex

- In 1968, D. H. Hubel and T. N. Wiesel demonstrated that mammals visually perceive the world around them using a layered architecture of neurons in the brain.

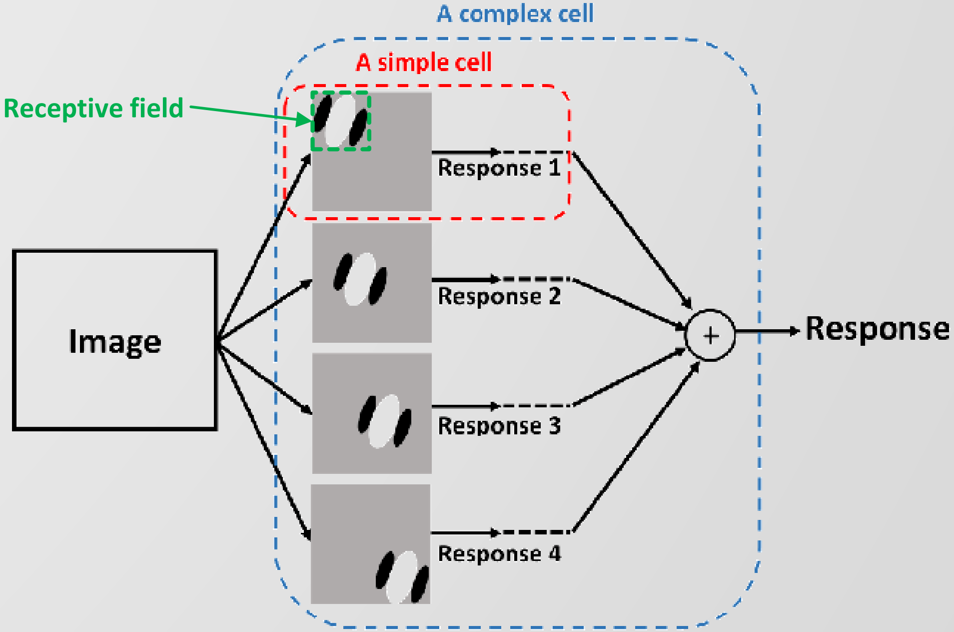


- The structure of the visual cortex is in layers. As information is passed from our eyes to the brain, higher and higher order representation are formed.

Visual cortex (2)

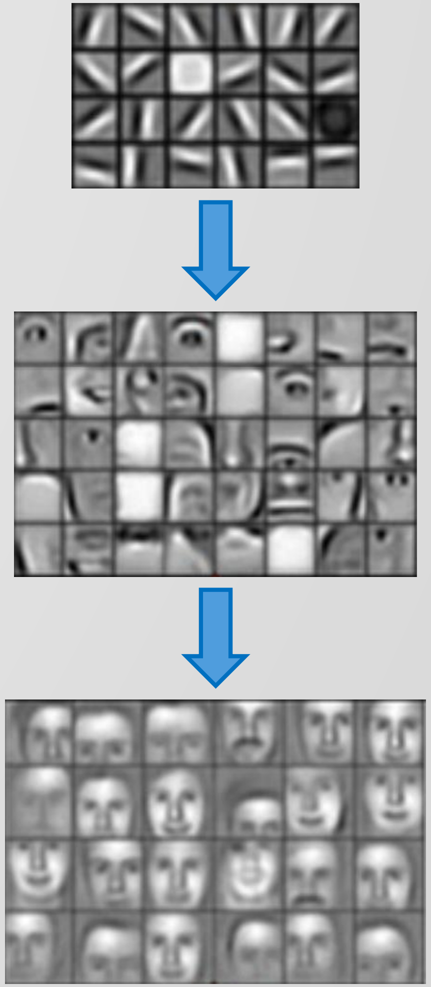
- Within the visual cortex, complex functional responses generated by "complex cells" are constructed by combining more simplistic responses from "simple cells".
  - Simple cells respond to edges with a specific orientation in a particular position (called *receptive field*).
  - Complex cells respond to edges with a specific orientation regardless of the position where they are located (obtaining spatial invariance).

- Spatial invariance is obtained by "summing" the contribution of simple cells responding to the same orientation but with different receptive fields.



The architecture of CNNs

- This is the inspiration behind CNNs. Higher and higher representations are formed through the layers:
  - the early layers, taking in the raw pixels, find edges;
  - then more abstract features are found by combining these edges;
  - finally, the last layers find higher order semantic meaning.

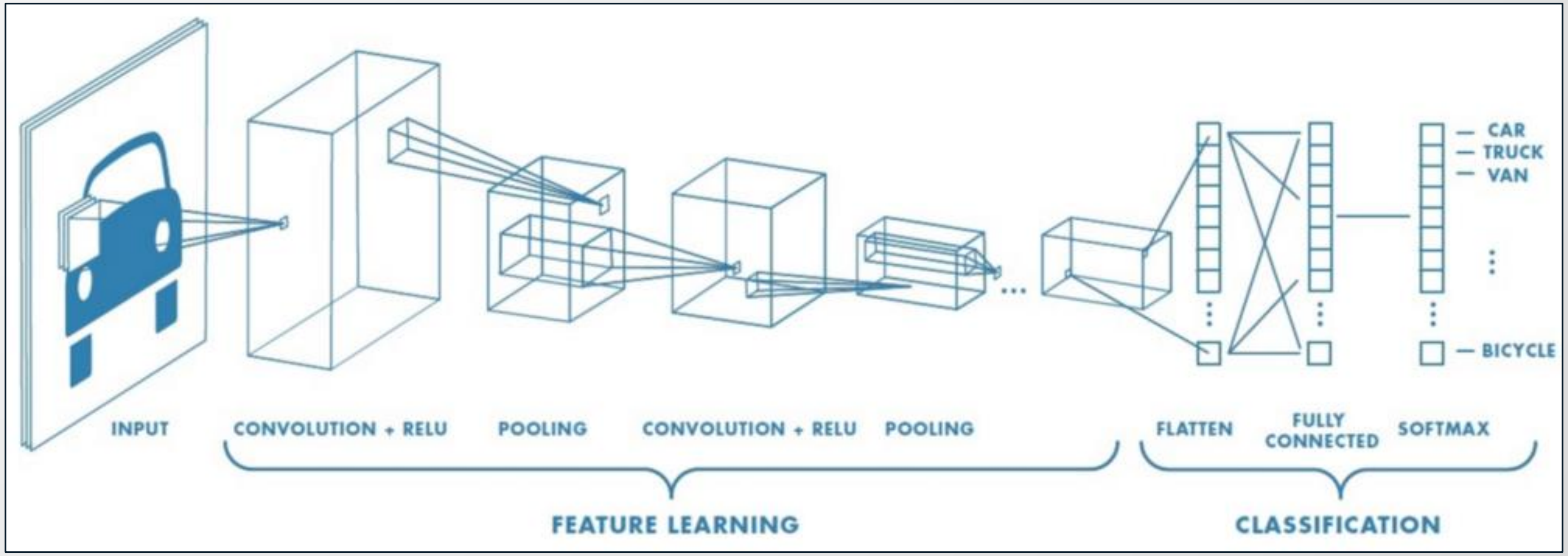


The architecture of CNNs (2)

- CNNs have been introduced in 1998 by Y. LeCun and Y. Bengio.
- Their architecture is based on:
  - local connections;
  - layering;
  - spatial invariance.
- The main differences compared to an MLP are:
  - local connections - neurons are only locally connected to neurons of the previous level with a strong reduction of the number of connections;
  - shared weights - different neurons of the same level perform the same operation on different portions (receptive field) of the input with a strong reduction of the number of weights;
  - alternation of feature extraction and pooling layers (this is no longer true for the most recent CNNs).

The architecture of CNNs (3)

- A CNN is a combination of two basic building parts:
  - The convolutional part — it consists of convolutional and pooling layers. This part forms the essential component of feature extraction.
  - The fully-connected part — it consists of a fully-connected neural network architecture. This part performs the task of classification based on the input from the convolutional part.

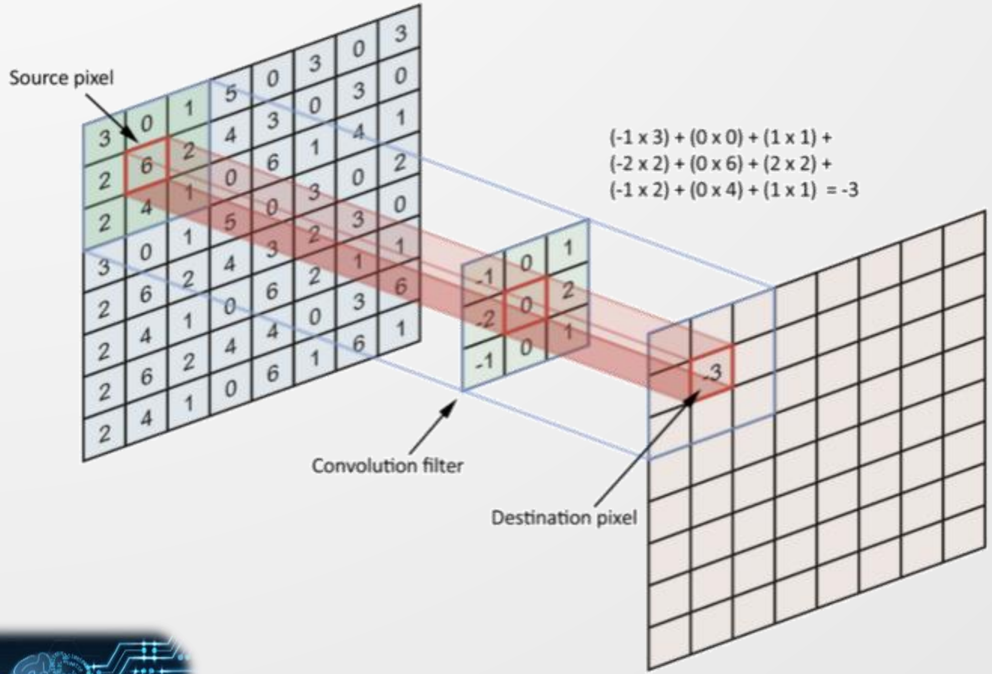


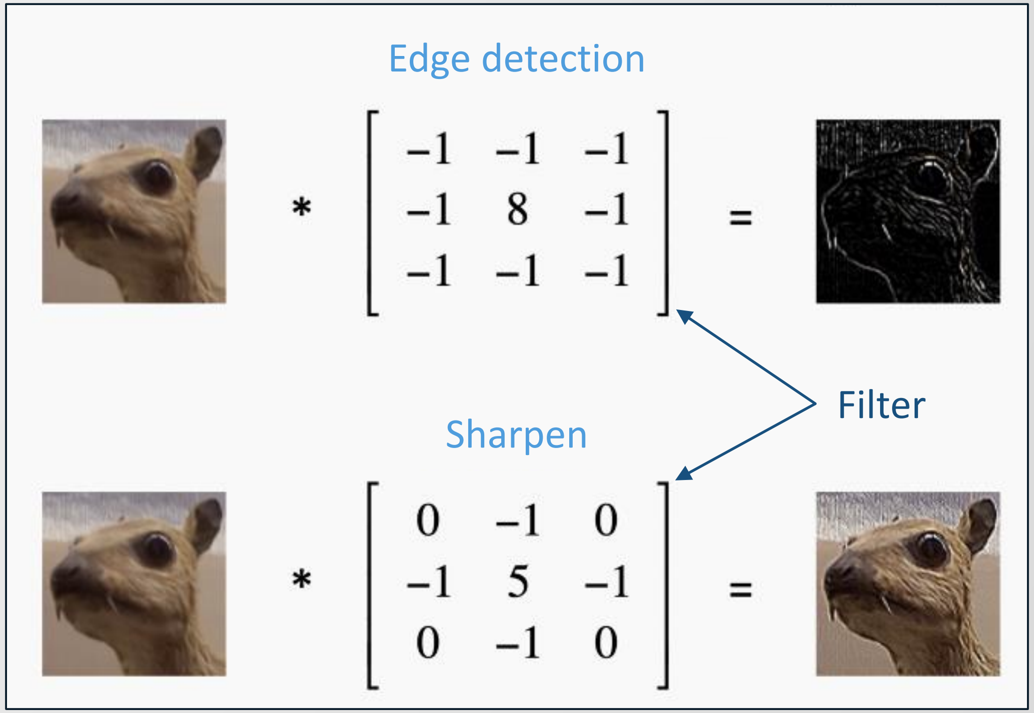
The architecture of CNNs (4)

- A CNN is a sequence of layers, and every layer transforms one volume of activations to another through a differentiable function.
- Three main types of layers are used to build CNN architectures:
  - Convolutional layer – it contains a set of learnable filters. The width and height of the filters are smaller than those of the input volume. The filter slides across the input and the dot products between the input and filter are computed at every spatial position.
  - Pooling layer- it reduces the number of parameters and computation by down-sampling the representation.
  - Fully-connected layer - neurons in a fully-connected layer have full connections to all activations in the previous layer, as seen in traditional ANNs.

Convolution

- Convolution is one of the most important image processing operations.
- A filter strides across the width and height of the input and the dot product between the filter and the input is computed at each position.



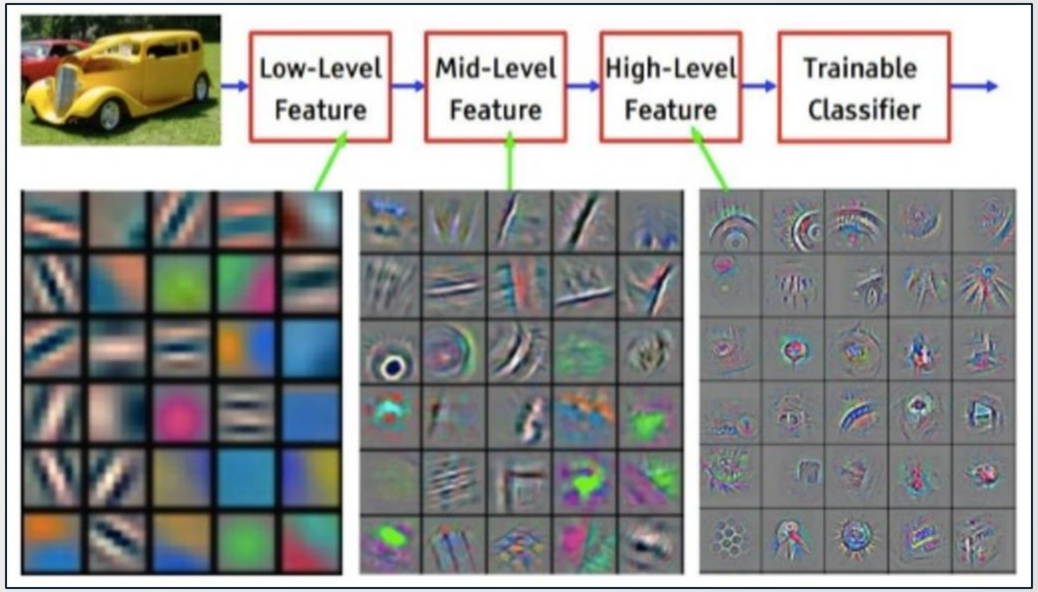


$$* \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$

$$* \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} =$$

Convolutional layer

- A CNN autonomously learns the kernel weights during the training process.
- Usually, the first convolutional layer is responsible to identify low-level features such as edges, color, gradient orientation, etc.
- The subsequent layers allow to distinguish increasingly higher-level features, giving the network a more detailed understanding of the image.

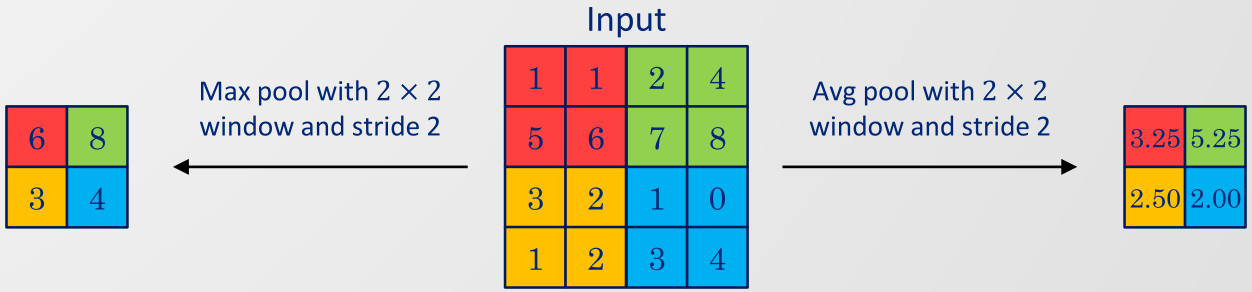


Pooling layer

- Often our ultimate task asks some global question about the image, so typically the units of our final layer should be sensitive to the entire input.
- Moreover, when detecting lower-level features, such as edges, we often want our representations to be invariant to translation.
- A pooling layer serves the dual purposes of:
    - aggregating information to reduce the spatial resolution and maintaining dominant features (reducing the amount of parameters, the computational power required and the risk of overfitting);
    - mitigating the sensitivity of convolutional layers to location (obtaining an approximate translation invariance).

Pooling layer (2)

- Pooling operator consists of a fixed-shape window that slides over all regions in the input and computing a single output for each location.
- Unlike convolutional layers, the pooling layer contains no trainable parameters.
- It is not a trainable layer but deterministic, typically calculating either the maximum or the average value of the elements in the pooling window.



Input

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

Max pool with 2 × 2 window and stride 2

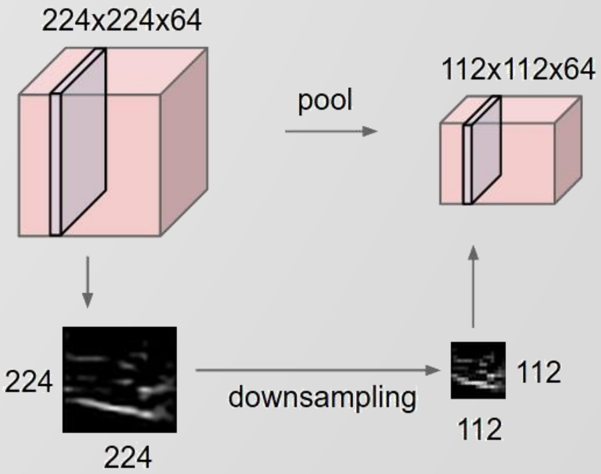
6	8
3	4

Avg pool with 2 × 2 window and stride 2

3.25	5.25
2.50	2.00

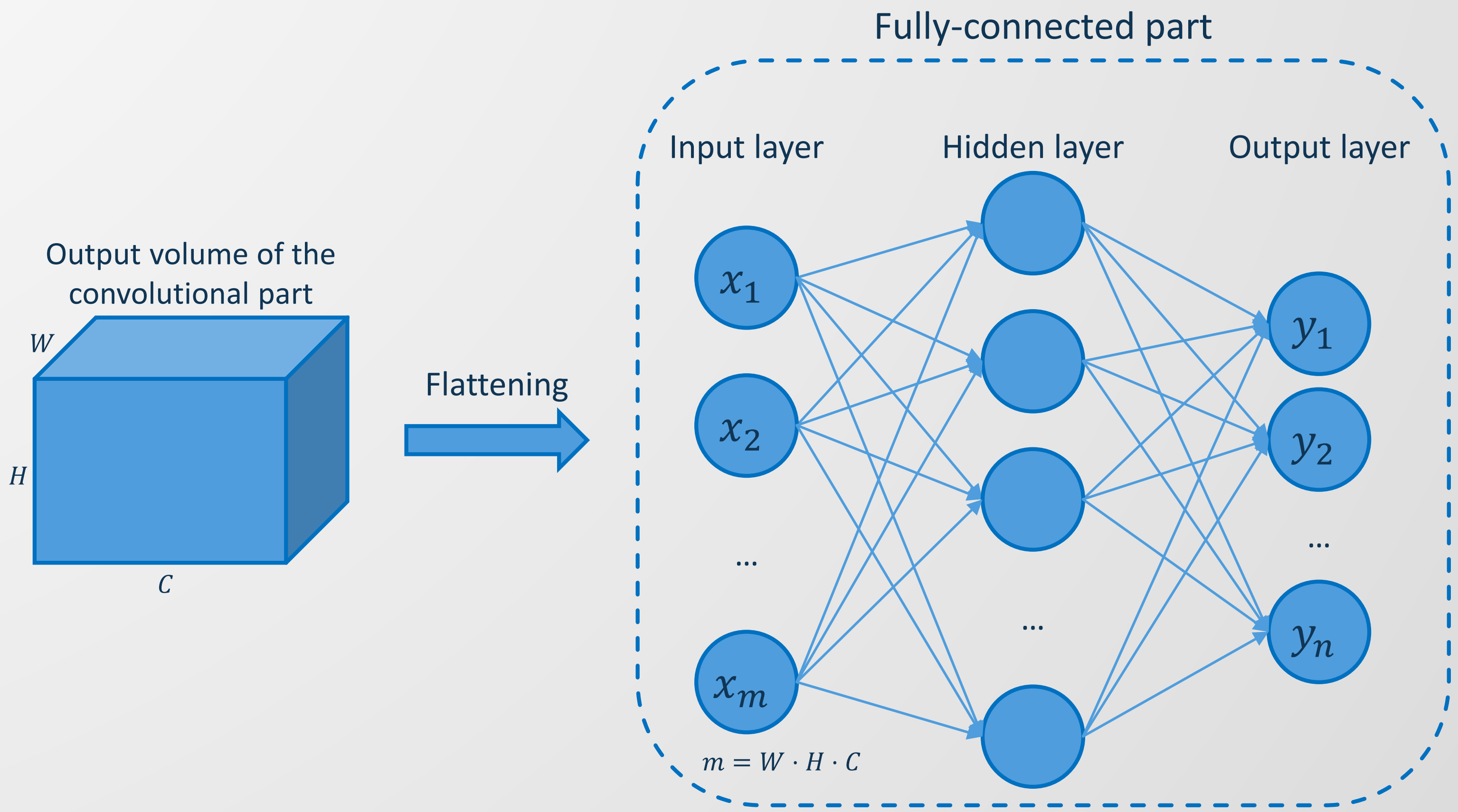
Pooling layer (3)

- Max pooling also performs as de-noising by discarding the noisy activations.
- Average pooling simply performs dimensionality reduction as a noise suppressing mechanism.
- Usually, max pooling performs better than average pooling.
- The pooling layer operates independently on every channel of the input volume keeping the depth size unchanged.
- As with convolutional layers, pooling layers can change the output shape by padding the input and adjusting the stride.



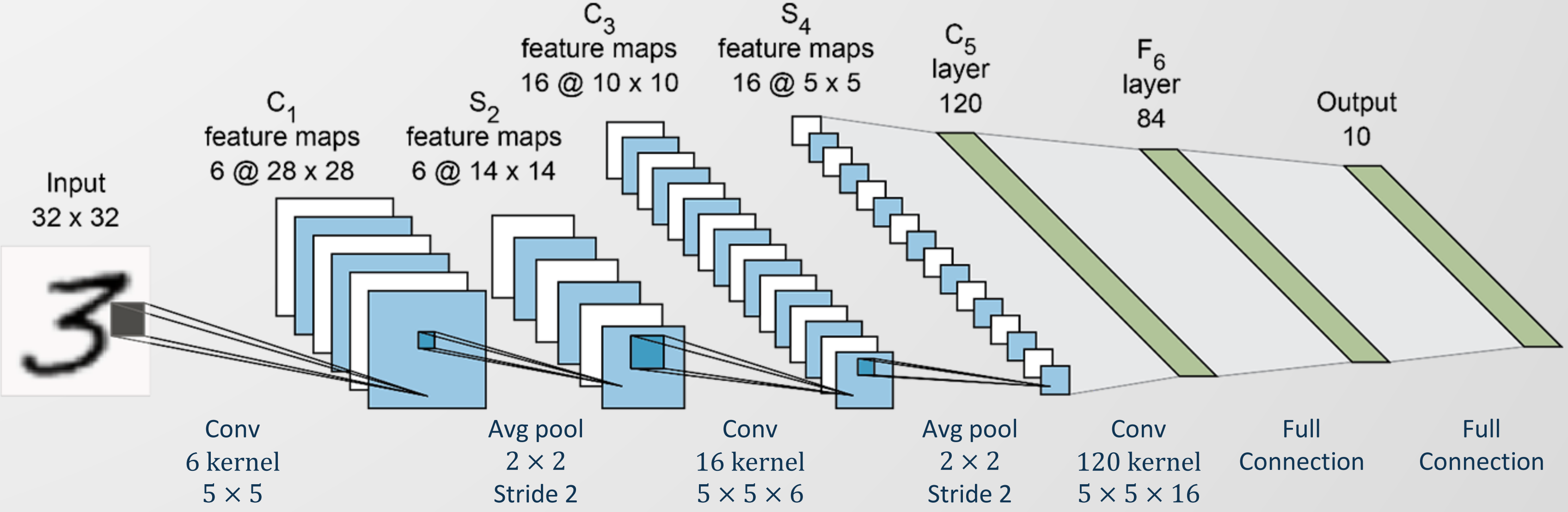
Fully-connected part

- Usually the fully-connected part is a simple MLP, consisting of two or three hidden layers and an output layer, that performs the classification among a large number of categories.



LeNet-5

- In 1998, LeNet-5 is introduced to recognize handwritten digits in images.
- It consists of:
  - three convolutional layers (C1, C3 and C5);
  - two average pooling layers (S2 and S4);
  - two fully-connected layers (F6 and Output).

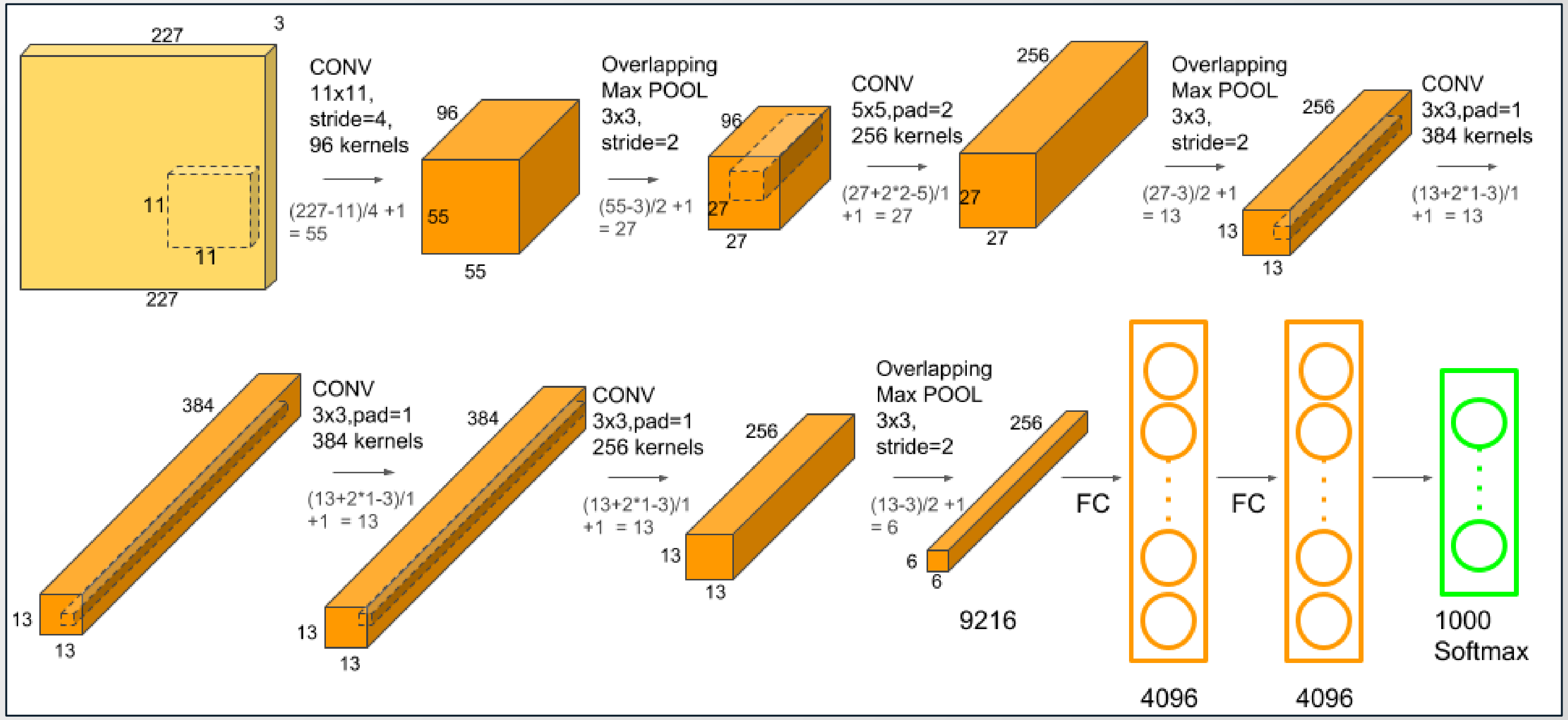


LeNet-5 (2)

- It is the first ANN to use convolutional and pooling layers to extract spatial features.
- Tanh has been used as activation function in C_1, C_3, C_5 and F_6 layers.
- It contains about 60K trainable parameters.
- It achieved 99% accuracy on the MNIST database containing handwritten characters divided into 10 classes.

AlexNet

- In 2012, AlexNet is introduced, a CNN with an architecture similar to LeNet-5.
- It consists of:
  - five convolutional layers;
  - three max pooling layers;
  - three fully-connected layers.

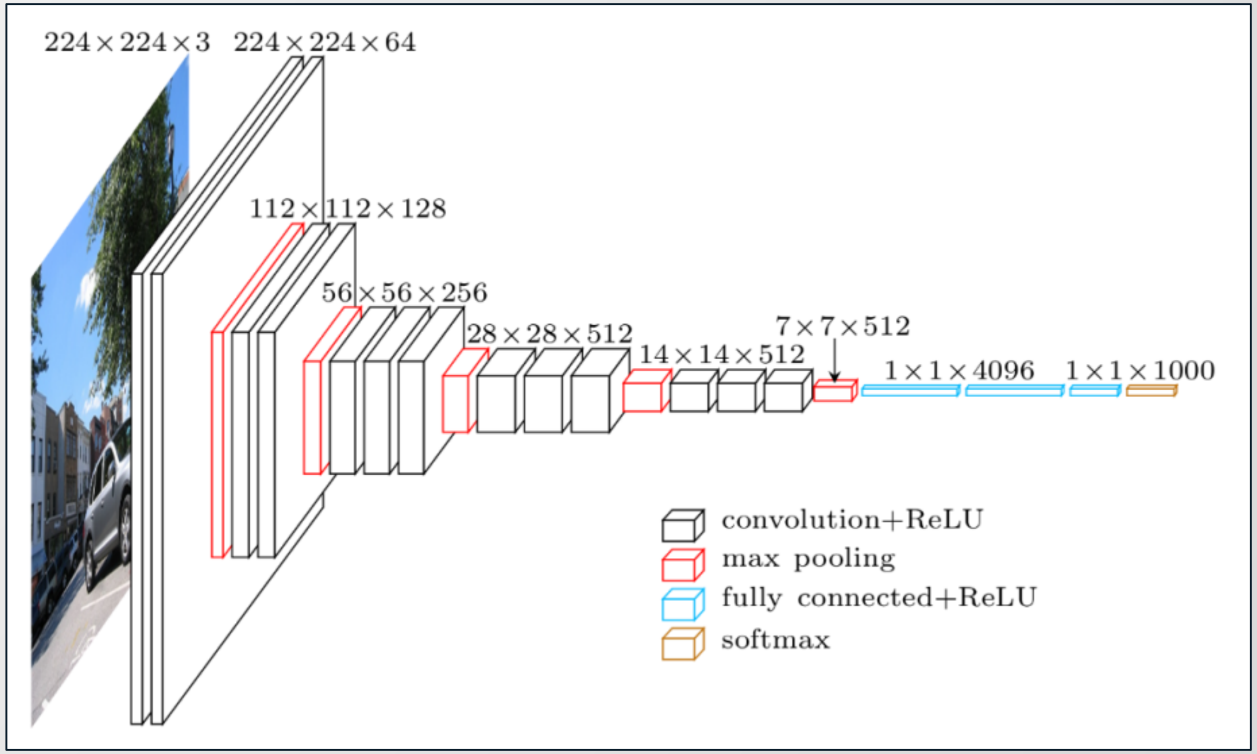


AlexNet (2)

- It won the ImageNet *Large Scale Visual Recognition Challenge* (LSVRC) 2012 by a very large margin with a top-5 error rate of 15.3% (compared to 26.2% of the runner-up).
- It showed, for the first time, that learned features can overcome hand-crafted features.
- There are also significant differences with respect to LeNet-5:
  - AlexNet is much deeper than LeNet-5;
  - ReLU is used as activation function;
  - Local response normalization is applied after the first two convolutional layers;
  - Dropout ($p = 0.5$) is used in the first two fully-connected layers to reduce overfitting;
  - Data augmentation is used to artificially enlarge the training set.
- It contains about 60M trainable parameters.

VGGNet

- In 2014, the *Visual Geometry Group* (VGG) proposes a novel CNN called VGGNet.
- It consists of:
  - 13 convolutional layers (VGG-16 version);
  - five max pooling layers;
  - three fully-connected layers.

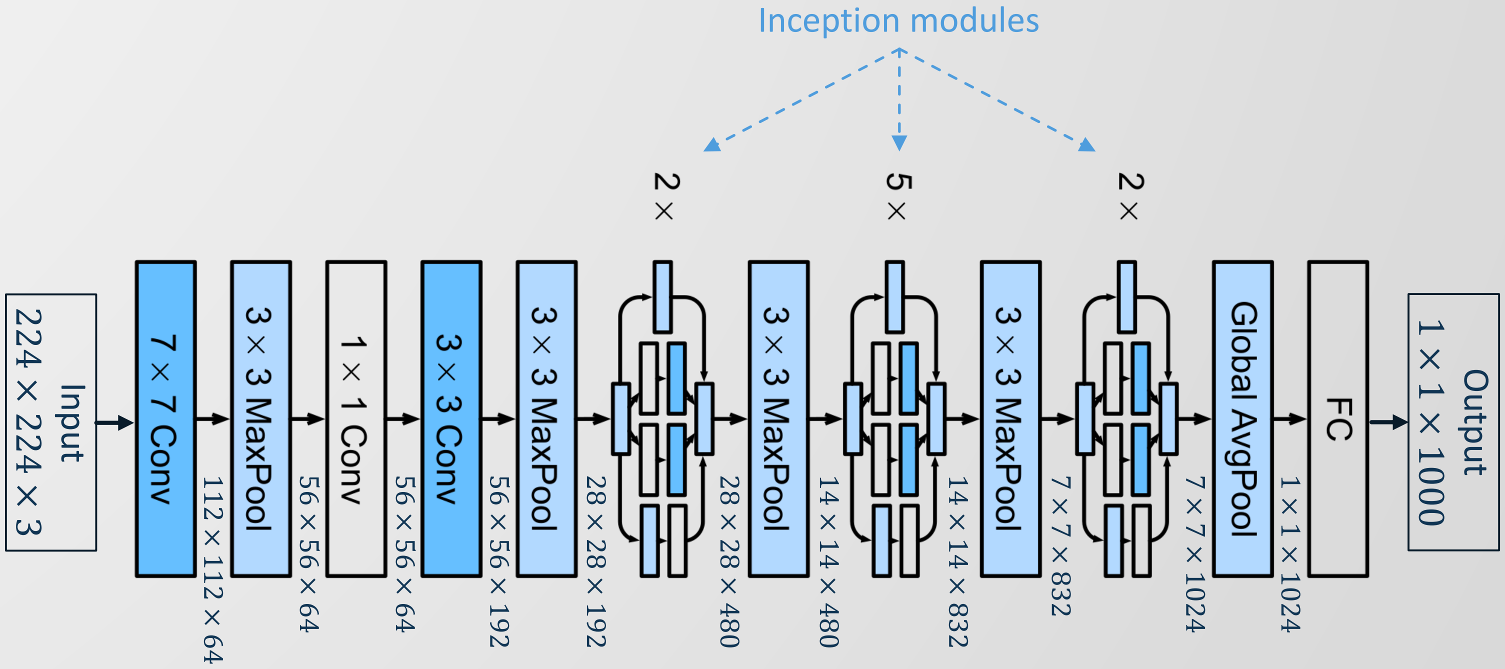


VGGNet (2)

- There are four versions sharing the number of pooling and fully-connected layers but with a different number of convolutional layers:
  - VGG-11 (8);
  - VGG-13 (10);
  - VGG-16 (13);
  - VGG-19 (16).
- VGG-16 reached a top-5 error rate of 7.3% on ImageNet LSVRC-2014.
- VGG-16 contains about 138M trainable parameters.

GoogLeNet

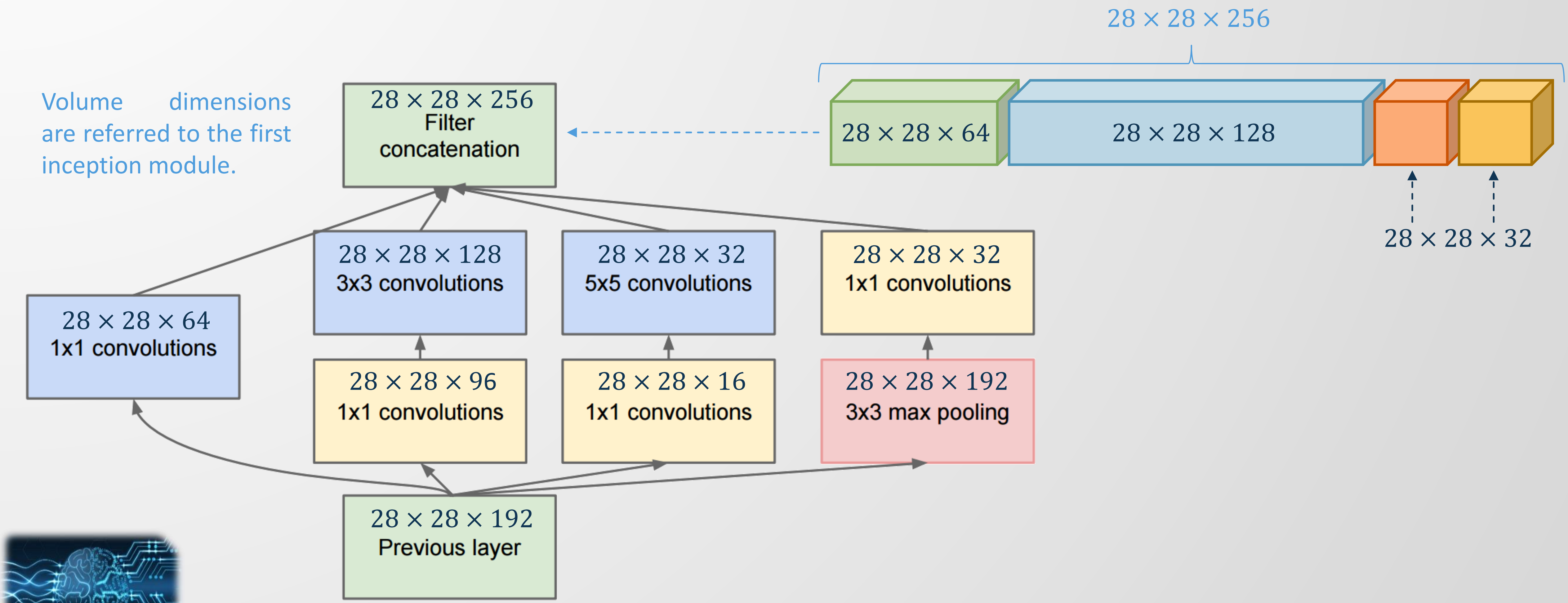
- In 2014, GoogLeNet won the ImageNet LSVRC-2014 challenge obtaining a top-5 error rate of 6.7% with about 7M trainable parameters.
- It consists of:
  - three convolutional layers;
  - four max pooling layers;
  - nine inception modules;
  - a global average pooling layer;
  - a fully-connected layer.



GoogLeNet – inception module

- The basic block in GoogLeNet is called inception module:

Volume dimensions are referred to the first inception module.



$28 \times 28 \times 256$

$28 \times 28 \times 64$

$28 \times 28 \times 128$

$28 \times 28 \times 32$

$28 \times 28 \times 256$
Filter concatenation

$28 \times 28 \times 64$
1x1 convolutions

$28 \times 28 \times 128$
3x3 convolutions

$28 \times 28 \times 32$
5x5 convolutions

$28 \times 28 \times 32$
1x1 convolutions

$28 \times 28 \times 96$
1x1 convolutions

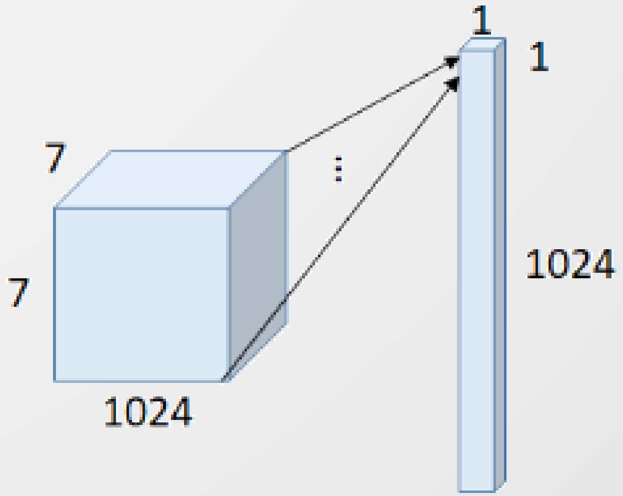
$28 \times 28 \times 16$
1x1 convolutions

$28 \times 28 \times 192$
3x3 max pooling

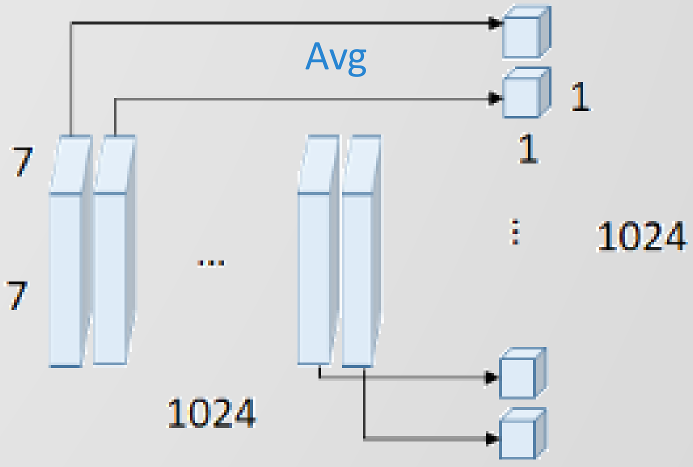
$28 \times 28 \times 192$
Previous layer

GoogLeNet – global average pooling

- In previous CNNs, fully-connected layers are used at the end of the network and all inputs are connected to each output.
- In GoogLeNet, global average pooling is used at the end of the network by averaging each feature map from 7×7 to 1×1 to drastically reduce the number of weights.



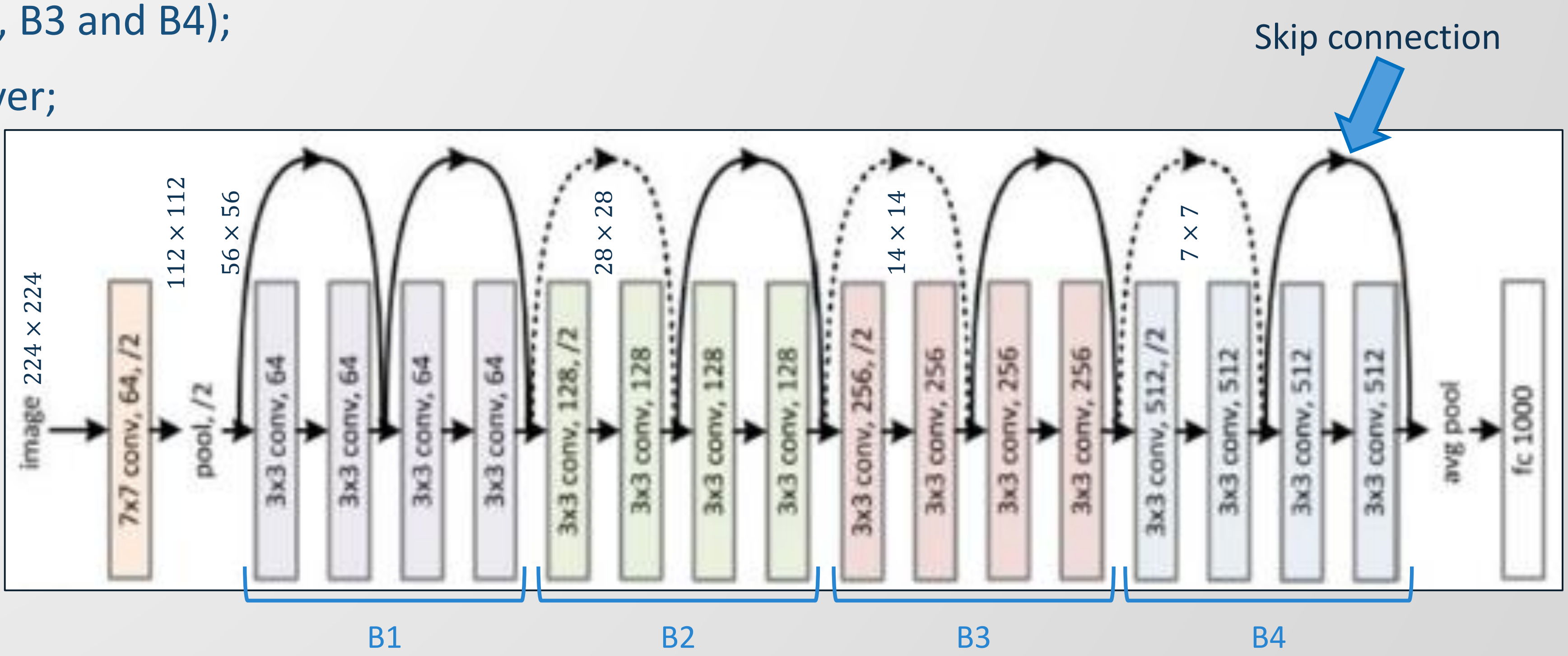
Fully-connected
weights (connections) = $1024 \cdot (7 \cdot 7 \cdot 1024) = 51.4\text{M}$



Global average pooling
weights (connections) = 0

ResNet

- In 2015, ResNet (Residual neural Network), proposed by Microsoft Research, won the ImageNet LSVRC-2015 challenge. It consists of:
  - a convolutional layer;
  - a max pooling layer;
  - four residual blocks (B1, B2, B3 and B4);
  - a global average pooling layer;
  - a fully-connected layer.

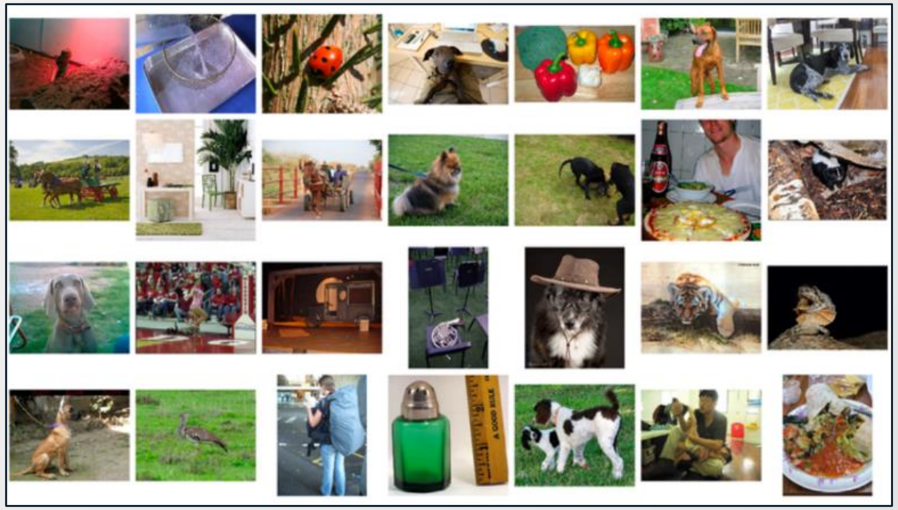


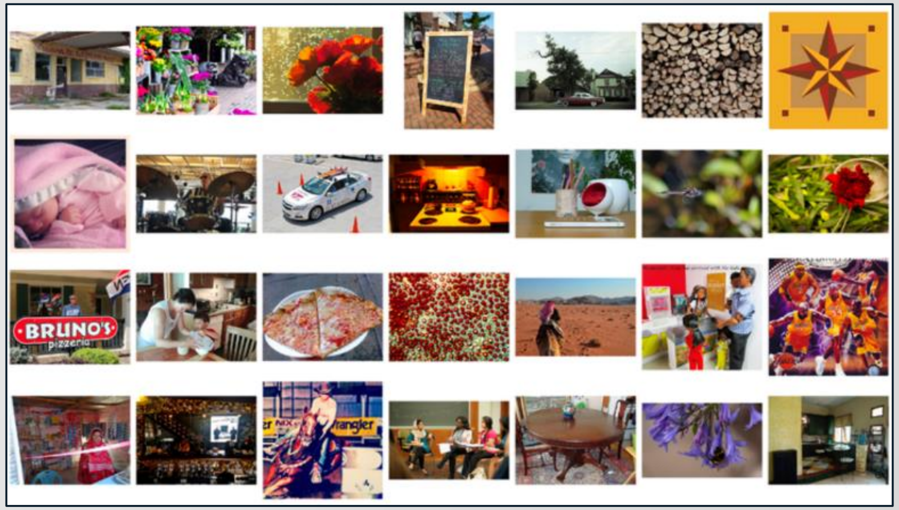
ResNet (2)

- There are five versions sharing the overall structure but with residual blocks presenting a different architecture and number of sub-blocks: ResNet-18, ResNet-34, ResNet-50, ResNet-101 e ResNet-152.
- ResNet won ImageNet LSVRC-2015 overtaking for the first time a human expert with a top-5 error rate of:
  - human expert – 5.1%;
  - ResNet-50 – 5.3%;
  - ResNet-152 – 4.5%.
- ResNet-50 contains about 25M trainable parameters.

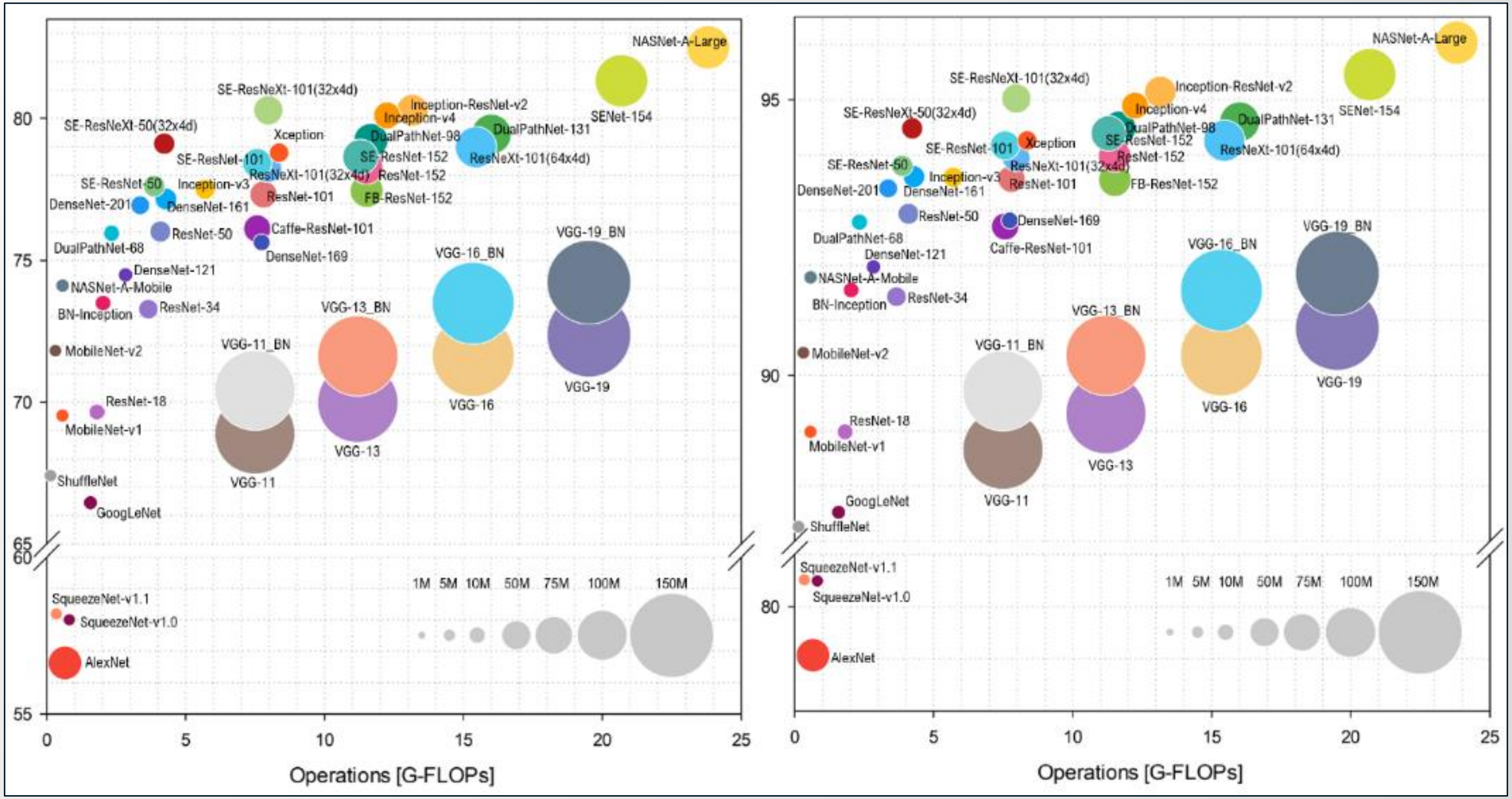
CNNs comparison

- ImageNet is a large visual database designed for use in visual object recognition software containing more than 14 million images collected from the web and labeled by humans.
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual international competition to evaluate image classification and object detection algorithms using the ImageNet database:
  - 1000 classes;
  - 1.2M training images;
  - 50K validation images;
  - 100K test images.





CNNs comparison (2)

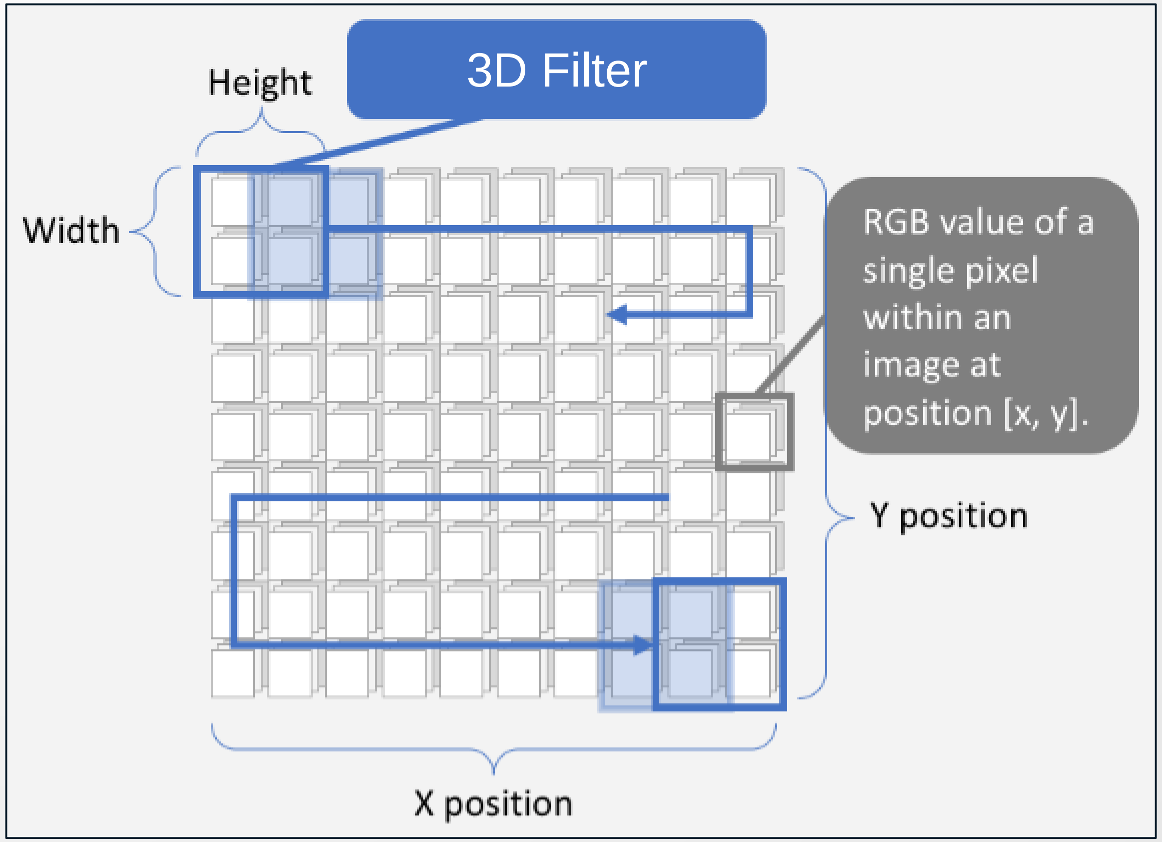


Top-1 accuracy [%]

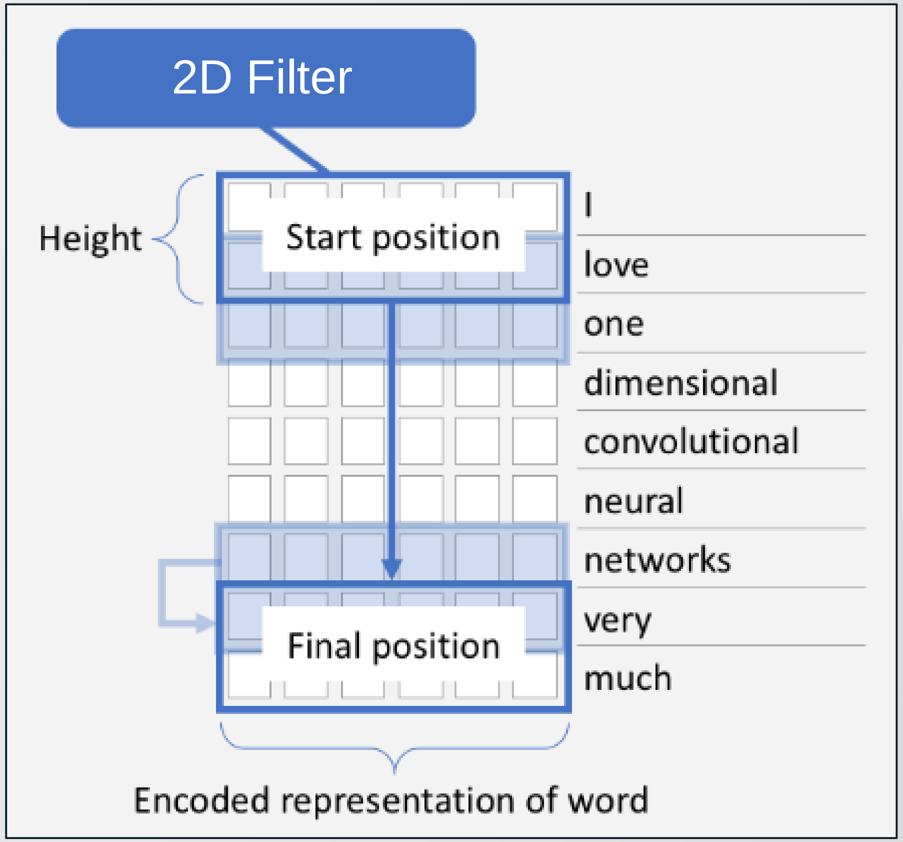
Top-5 accuracy [%]

1D CNNs for time sequences

2D convolutional layer



1D convolutional layer



Recurrent neural networks

> “*A Recurrent Neural Network (RNN) is a type of neural network that contains loops, allowing information to be stored within the network.*”

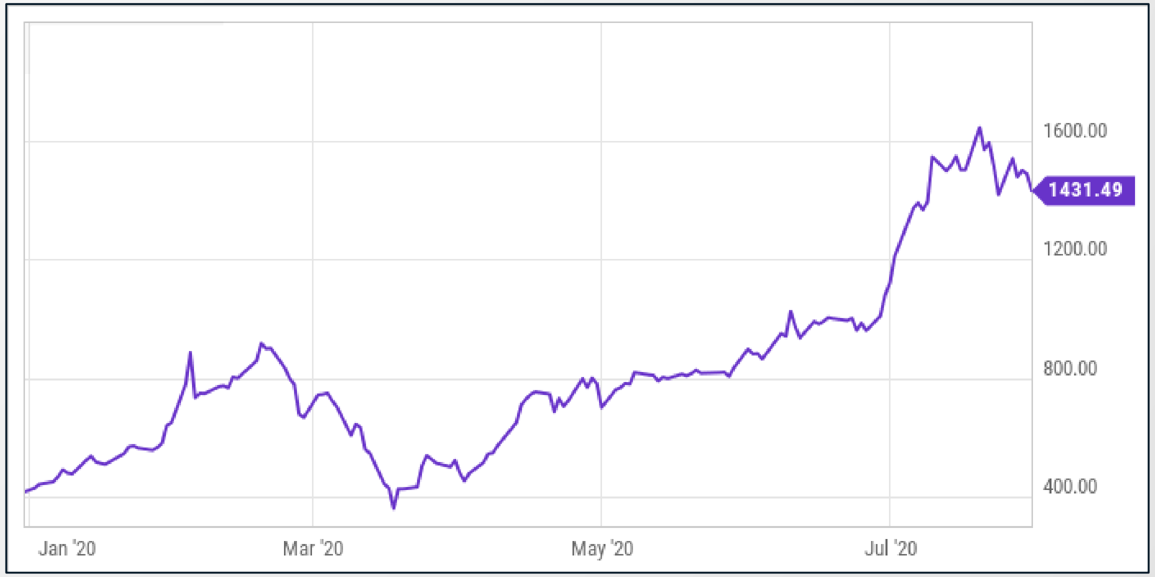
From DeepAI

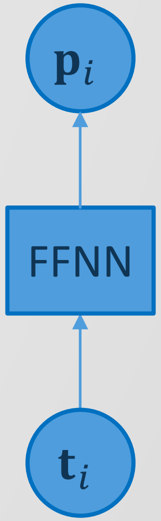
What is a recurrent neural network?

- Feed-Forward Neural Networks (FFNNs) are really good at learning a pattern between a set of inputs and outputs assuming that all inputs (and outputs) are independent of each other.
- FFNNs accept a fixed-sized vector as input (e.g., an image) and produce a fixed-sized vector as output (e.g., probabilities of different classes).
- FFNNs are not well suited to tasks which require previous context for making future predictions.
- In other words, FFNNs are not designed to take a series of input with no predetermined limit on size.

What is a recurrent neural network? (2)

- For example, if we have to predict the price of a stock, a FFNN can make a prediction ($\mathbf{p}$) based on the current time ($\mathbf{t}$).

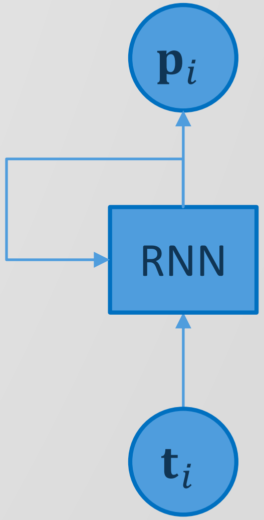




- This is not sufficient to make an accurate prediction because the current stock price depends on the stock trend and not only on the current time.

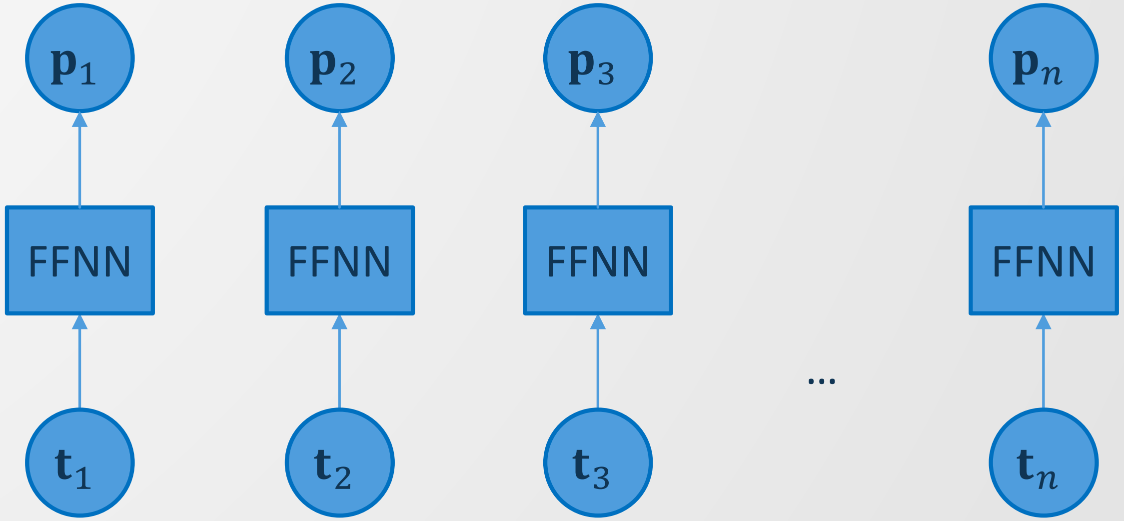
What is a recurrent neural network? (3)

- RNNs are a class of neural networks which not just looks at the current input but uses sequential data or time series data.
- The output of any layer not only depends on the current input but also on the sequence of inputs that have came before.
- This special feature provides it a significant advantage over FFNNs by taking help of inputs obtained before to predict outputs at the later stage.
- Another way to think about RNNs is that they have a "memory" which captures information about what has been previously calculated.



What is a recurrent neural network? (4)

- Why not repeatedly call a FFNN?



- Because each input item from the series is related to the others and it has an influence on its neighbors. Otherwise it is not a series but only many inputs.
- RNNs are able to capture this relationship across inputs meaningfully.

Applications

- Machine translation
- Natural language processing
- Robot control
- Time series prediction
- Speech recognition
- Speech synthesis
- Time series anomaly detection
- Sentiment analysis
- Rhythm learning
- Music composition
- Grammar learning
- Handwriting recognition
- Human action recognition
- Image captioning
- Video tagging
- Text summarization

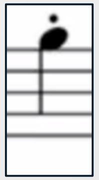
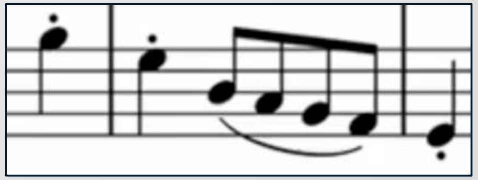
Examples of sequence data

- Speech recognition:

→ *“The brown fox jumped over the lazy dog.”*

- Sentiment analysis: *“There is nothing to like in this movie.”* → ★☆☆☆☆

- Music composition:



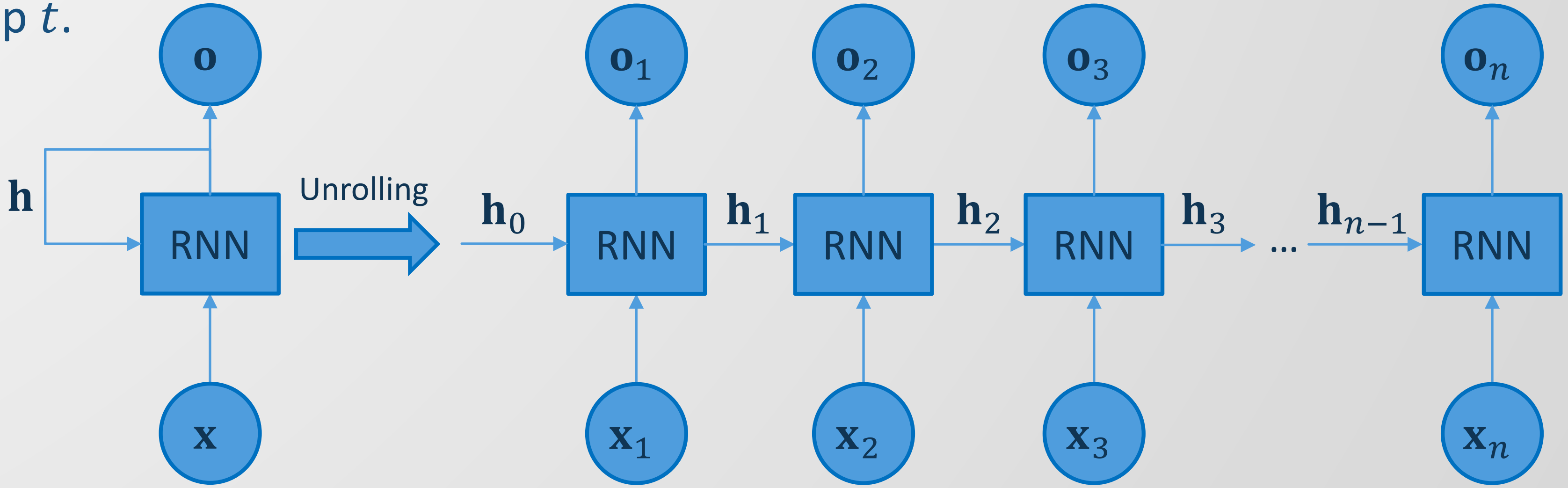
→

- DNA analysis: AGCCCCTGTGAGGAACTAG → AGCCCCTGTGAGGAACTAG

- Machine translation: *“Do you want to dance with me?”* → *“Vuoi ballare con me?”*

RNNs architecture

- The diagram shows a RNN being *unrolled* (or unfolded) into a full network.
- Unrolling means that the network is written out for the complete sequence.
- Where:
  - $\mathbf{x}_t$ is the input at time step t;
  - $\mathbf{h}_t$ is the hidden state at time step t;
  - $\mathbf{o}_t$ is the output at time step t.



Types of recurrent neural networks

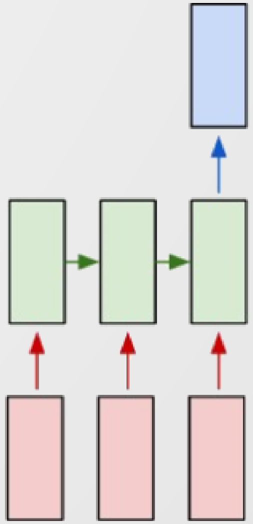
- FFNNs map one input to one output while RNNs inputs and outputs can vary in length.
- RNNs are of different types based on the number of their inputs and outputs.

One-to-one
$T_x = T_y = 1$

Traditional FFNNs

Types of recurrent neural networks (2)

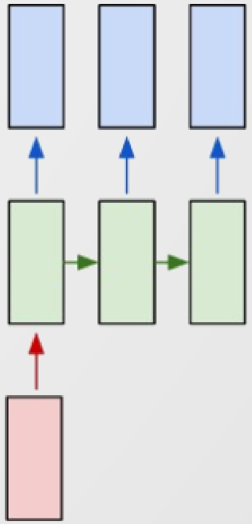
Many-to-one
$T_x > 1, T_y = 1$



Sentiment analysis
Movie rating
Video activity recognition

Types of recurrent neural networks (3)

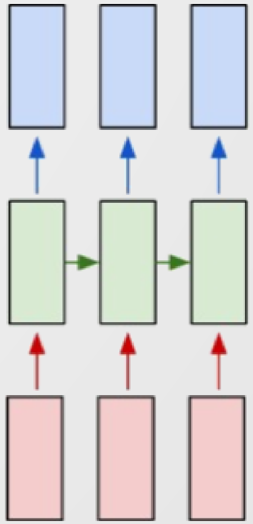
One-to-many
$T_x = 1, T_y > 1$



Music composition
Image captioning

Types of recurrent neural networks (4)

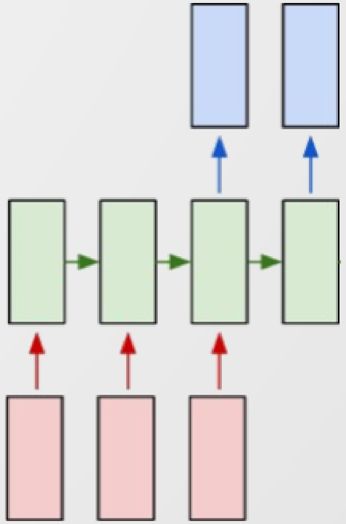
Many-to-many
$T_x = T_y > 1$



Named-entity recognition
Video classification of each frame

Types of recurrent neural networks (5)

Many-to-many
$T_x \neq T_y, T_x > 1, T_y > 1$



Machine translation
Speech recognition

Types of recurrent neural networks (6)

One-to-one $T_x = T_y = 1$		→	Cat
Many-to-one $T_x > 1, T_y = 1$	*“This café is great, the staff is really friendly and the coffee is delicious”*	→	Positive
One-to-many $T_x = 1, T_y > 1$		→	*“Dog is running through the water”*
Many-to-many $T_x = T_y > 1$	*“Luke joined Google as a data scientists in Mountain view”*	→	*“*$\underline{\text{Luke}}^{\text{PERSON}}$ *joined* $\underline{\text{Google}}^{\text{ORG}}$ *as a data scientists in* $\underline{\text{Mountain view}}^{\text{PLACE}}$*”*
Many-to-many $T_x \neq T_y, T_x > 1, T_y > 1$	*“Do you want to dance with me?”*	→	*“Vuoi ballare con me?”*

Advantage and drawbacks of RNNs

- Advantages
  - Possibility of processing input of any length.
  - Model size not increasing with size of input.
  - Computation takes into account historical information.
  - Weights are shared across time.

- Drawbacks
  - Computation being slow.
  - Exploding and vanishing gradient.
  - Difficulty of accessing information from a long time ago (short-term memory).
  - Cannot consider any future input for the current state.

Advantage and drawbacks of RNNs (2)

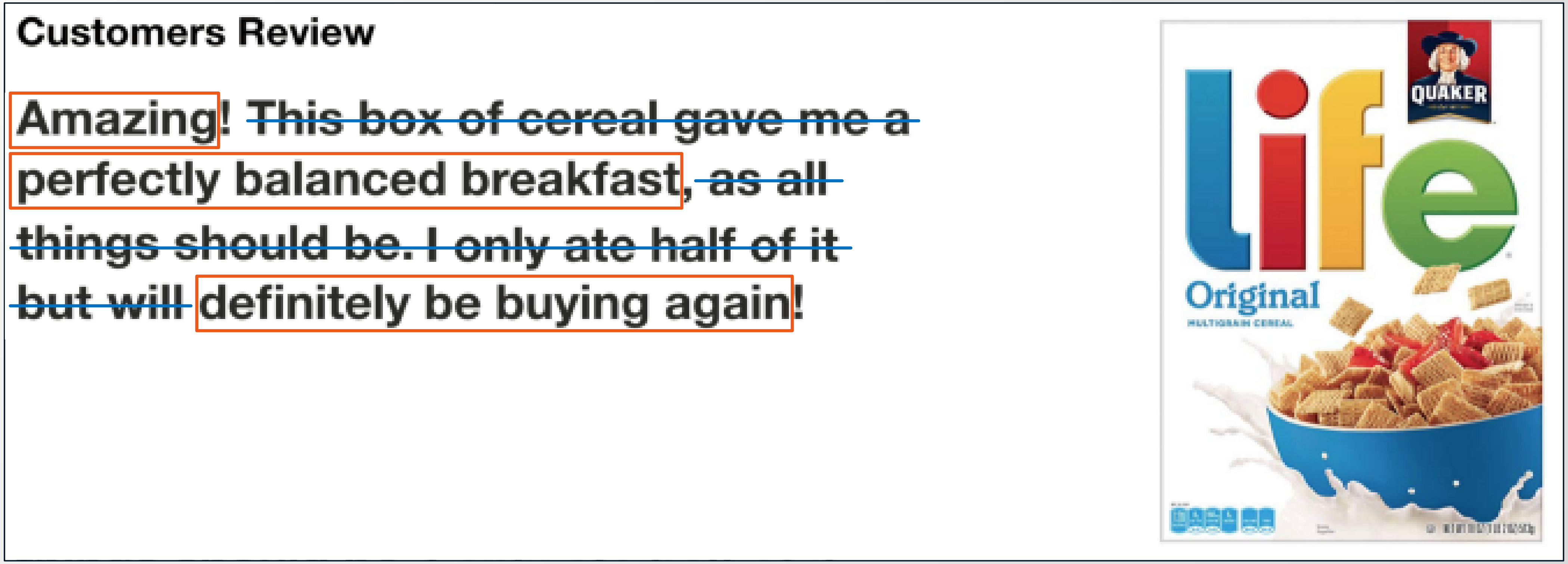
- Because of the vanishing/exploding gradient problems, RNNs suffer from short-term memory: they are not able to memorize data for long time and begins to forget its previous inputs.
- Consider trying to predict the last word in:

"I grew up in France … I speak fluent French"

- Recent information suggests that the next word is probably the name of a language.
- To narrow down which language, we need the context of *France*, from further back.
- Unfortunately, as the gap between the relevant information and the point where it is needed grows, RNNs become unable to learn to connect the information.

Advantage and drawbacks of RNNs (4)

- Moreover, there are situations where some tokens are irrelevant because carry no pertinent observation.



Customers Review

Amazing! ~~This box of cereal gave me a~~ perfectly balanced breakfast, ~~as all things should be. I only ate half of it but will~~ definitely be buying again!

Advantage and drawbacks of RNNs (5)

Possible solutions:

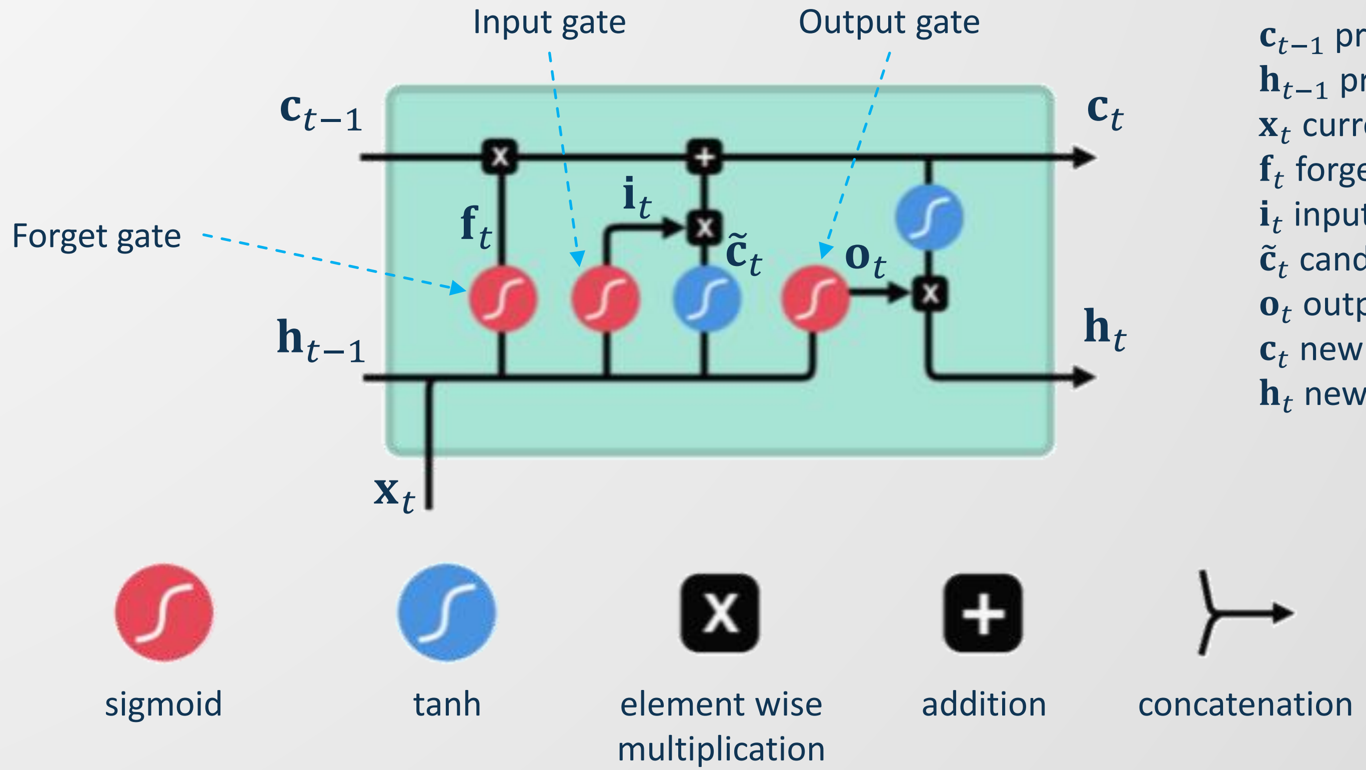
- Difficulty of accessing information from a long time ago (short-term memory)
  - long short-term memory
  - gated recurrent units
- Cannot consider any future input for the current state
  - bidirectional recurrent neural networks

Long short-term memory

- Long Short-Term Memory networks (LSTMs) are a special kind of RNN, capable of learning long-term dependencies.
- LSTM contains internal mechanisms (called gates) to regulate the information flow.
- These gates can learn which data in a sequence is important to keep or throw away.
- By doing that, it passes relevant information down the long chain of sequences to make predictions and discards non relevant data.

Long short-term memory (2)

- An LSTM has a similar control flow as a RNN. It processes data, passing on information as it propagates forward. The differences are the operations within the LSTM's cells.



$\mathbf{c}_{t-1}$ previous cell state
$\mathbf{h}_{t-1}$ previous hidden state
$\mathbf{x}_t$ current input
$\mathbf{f}_t$ forget gate output
$\mathbf{i}_t$ input gate output
$\tilde{\mathbf{c}}_t$ candidate
$\mathbf{o}_t$ output gate output
$\mathbf{c}_t$ new cell state
$\mathbf{h}_t$ new hidden state

Long short-term memory (3)

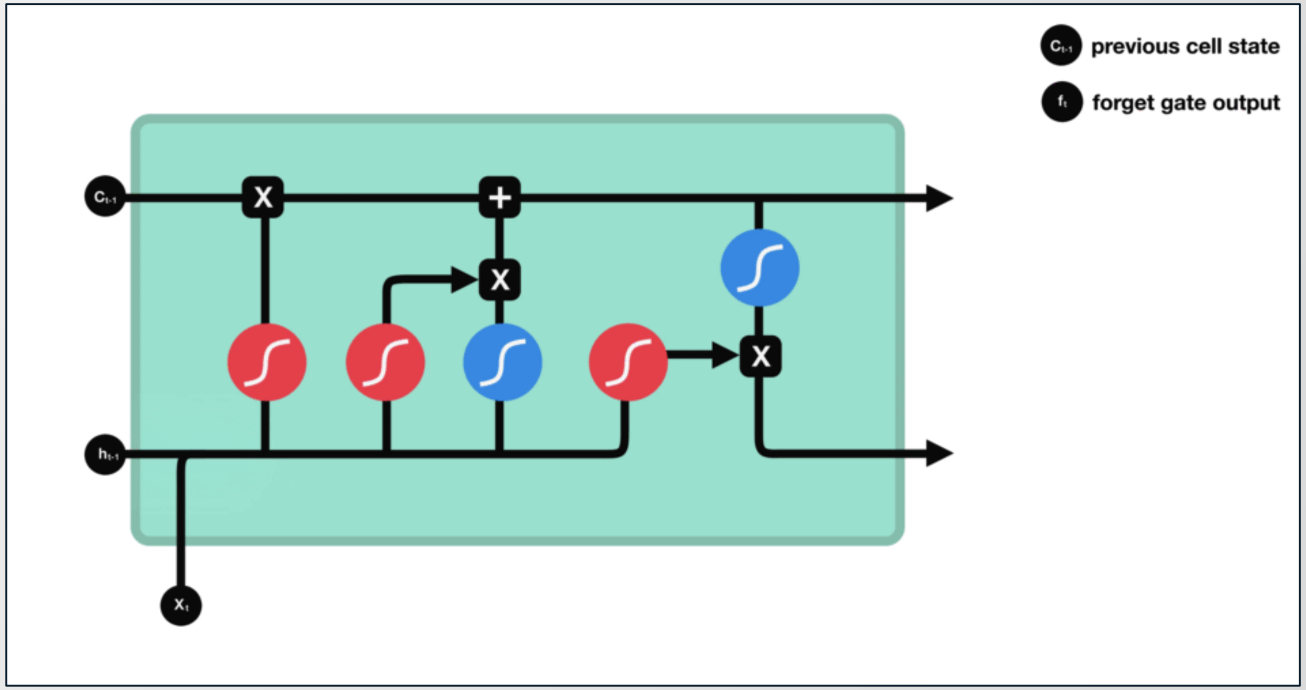
- The core concept of LSTM is the cell state $\mathbf{c}_t$, and its various gates.
- The cell state act as a transport highway that transfers relative information all the way down the sequence chain. You can think of it as the "memory" of the network.
- The cell state can carry relevant information throughout the processing of the sequence.
- Even information from the earlier time steps can make its way to later time steps, reducing the effects of short-term memory.
- As the cell state goes on its journey, information are added or removed to the cell state via gates.
- The gates are different neural networks that decide which information is allowed on the cell state.

Long short-term memory (4)

- The LSTM have the ability to remove or add information to the cell state, carefully regulated by structures called gates.
- Gates are a way to optionally let information through. They are composed by a sigmoid neural network layer.
- A gate returns values between 0 and 1, describing how much of each component should be let through. A value of 0 means “forget information” while a value of 1 means “kept information as is”.
- An LSTM has three of these gates, to protect and control the cell state. In particular:
  - the forget gate decides what is relevant to keep from prior steps;
  - the input gate decides what information are relevant to add from the current step;
  - the output gate determines what the next hidden state should be.

Long short-term memory (5)

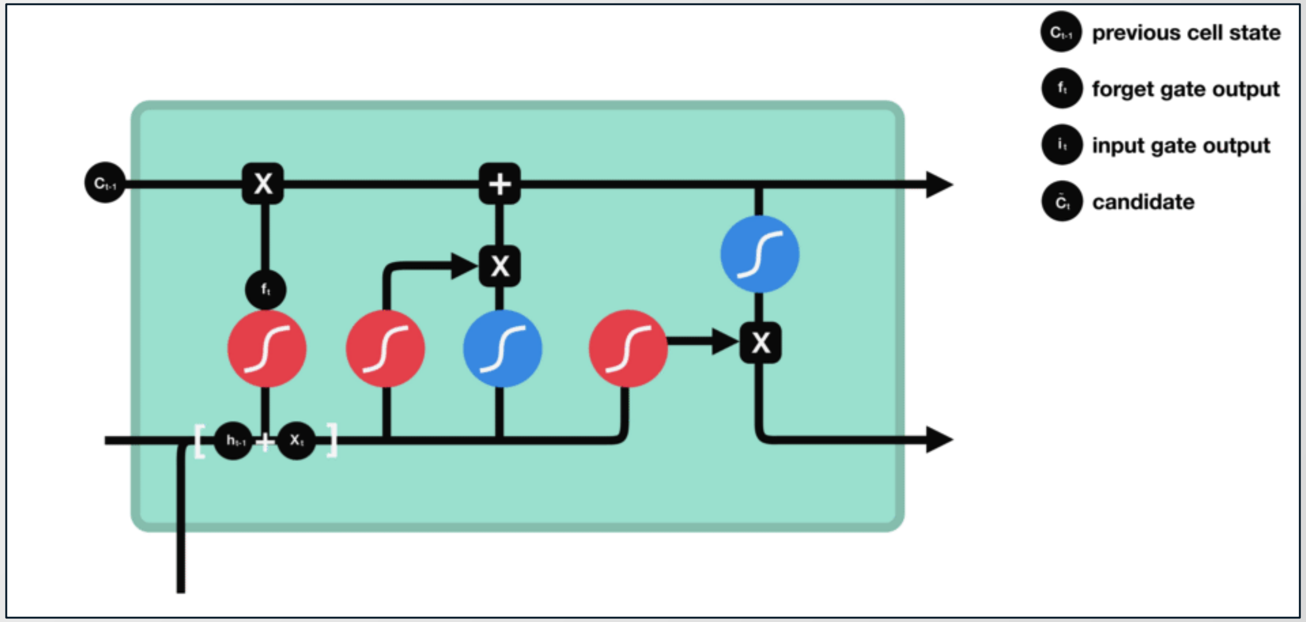
- The first step is to decide what information from the previous cell state $\mathbf{c}_{t-1}$ should be kept.
- This decision is made by the forget gate: it looks at the previous hidden state $\mathbf{h}_{t-1}$ and the current input $\mathbf{x}_t$, and outputs a value between 0 and 1 for each element of the previous cell state $\mathbf{c}_{t-1}$.



Source

Long short-term memory (6)

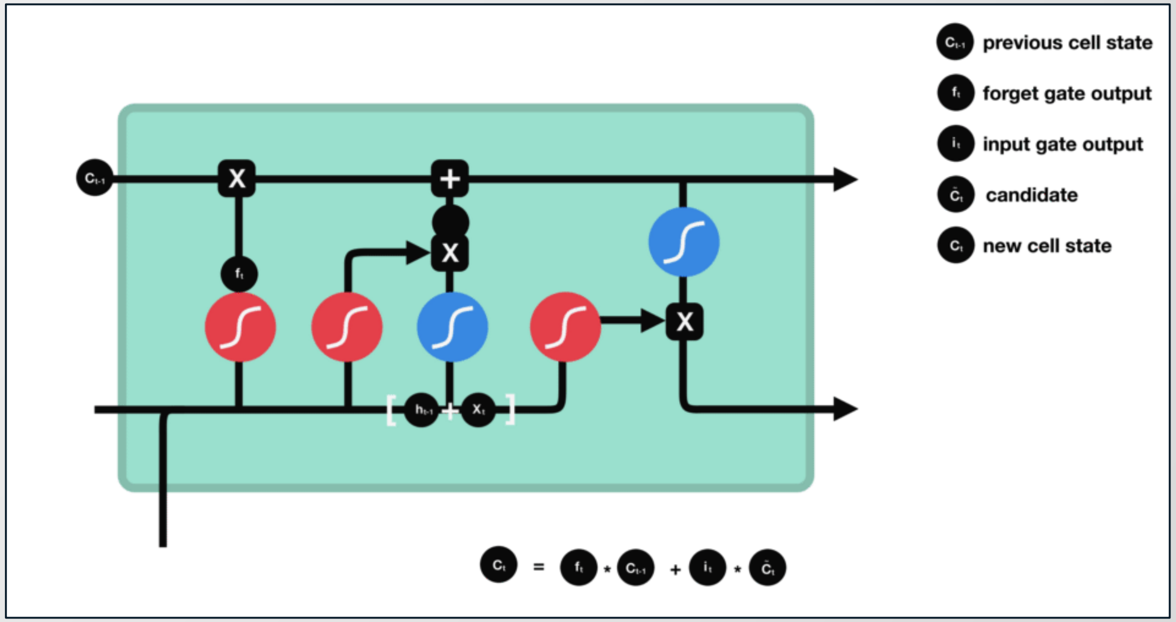
- The next step is to decide what new information will be stored in the new cell state $\mathbf{c}_t$.
- First, the input gate decides which values will be updated given the previous hidden state $\mathbf{h}_{t-1}$ and the current input $\mathbf{x}_t$.
- Next, a tanh layer creates a vector of new candidate values ($\tilde{\mathbf{c}}_t$) that could be added to the new cell state ($\mathbf{c}_t$).



Source

Long short-term memory (7)

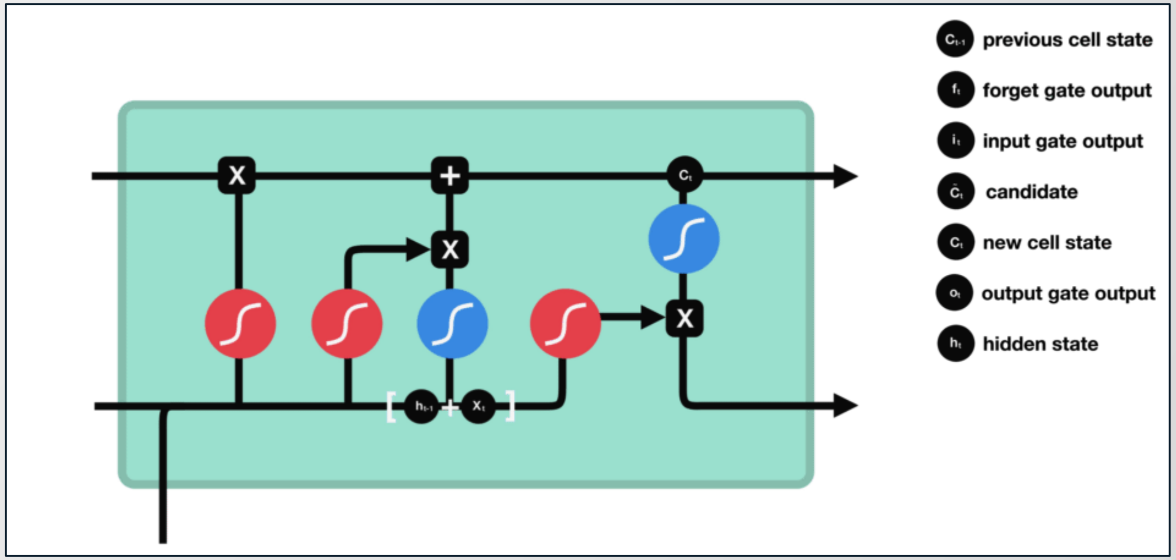
- To calculate the new cell state $\mathbf{c}_t$, the previous cell state $\mathbf{c}_{t-1}$ is multiplied by the forget output $\mathbf{f}_t$ to remove the things we decided to forget earlier.
- Then the new weighted candidate values $(\mathbf{i}_t \odot \tilde{\mathbf{c}}_t)$ are added.



Source

Long short-term memory (8)

- Finally, the new hidden state $\mathbf{h}_t$ will be computed from a filtered version of the new cell state.
- The previous hidden state $\mathbf{h}_{t-1}$ and the current input $\mathbf{x}_t$ are passed into the output gate while the new cell state $\mathbf{c}_t$ is passed through a tanh activation function.
- Then the two outputs are multiplied together to decide what information will be carried by the new hidden state $\mathbf{h}_t$.

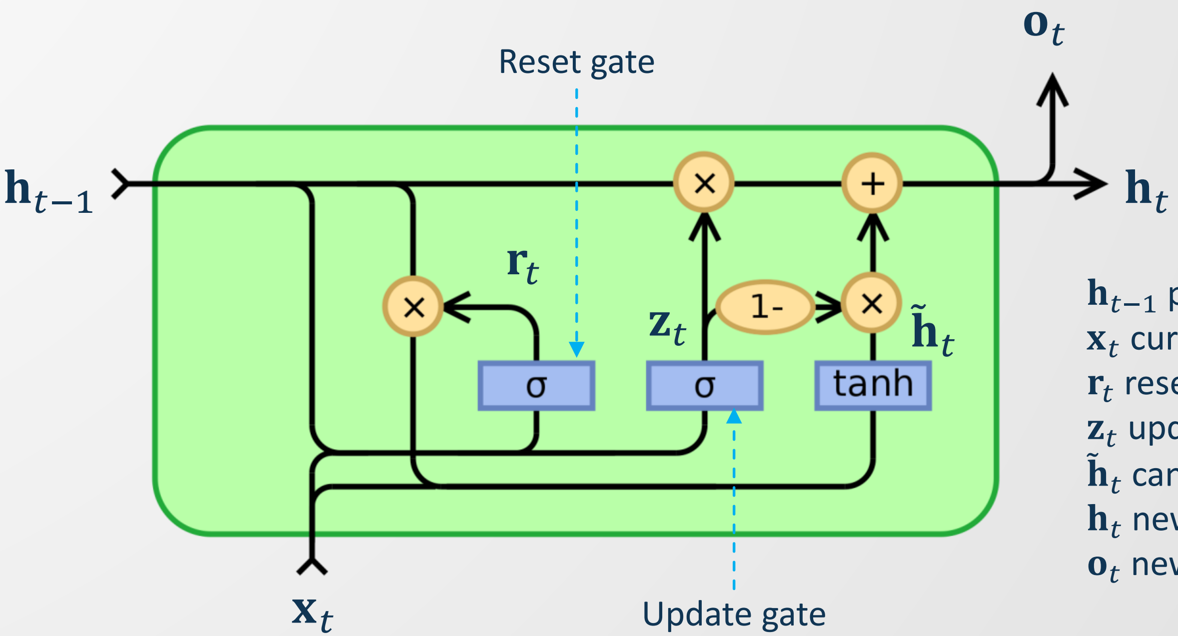


Source

Gated recurrent units

- Gated Recurrent Units (GRU) belongs to the newer generation of RNNs and it is pretty similar to an LSTM but with fewer parameters as it lacks an output gate.
- GRU has only two gates:
  - the reset gate decides how much past information to forget;
  - the update gate acts similar to the forget and input gates of an LSTM. It decides what information to throw away and what new information to add.
- GRU's performance on certain tasks was found to be similar or even better than that of LSTM.
- GRU uses less memory and is faster than LSTM.
- In general LSTM outperforms GRU especially when using datasets with longer sequences.

Gated recurrent units (2)



$\mathbf{h}_{t-1}$ previous hidden state
$\mathbf{x}_t$ current input
$\mathbf{r}_t$ reset gate output
$\mathbf{z}_t$ update gate output
$\tilde{\mathbf{h}}_t$ candidate hidden state
$\mathbf{h}_t$ new hidden state
$\mathbf{o}_t$ new output

- Note that, in GRU the new hidden state and the new output are the same.

Bidirectional recurrent neural networks

- The objective in a typical sequence learning scenario is to model the next output given a sequence of past information.
- Sometimes it is not enough to learn from the past to predict the future, but it is also important to look into the future to fix the past.
- Consider the task of filling in the blank in a text sequence:

“I am ___”

“I am ___ hungry”

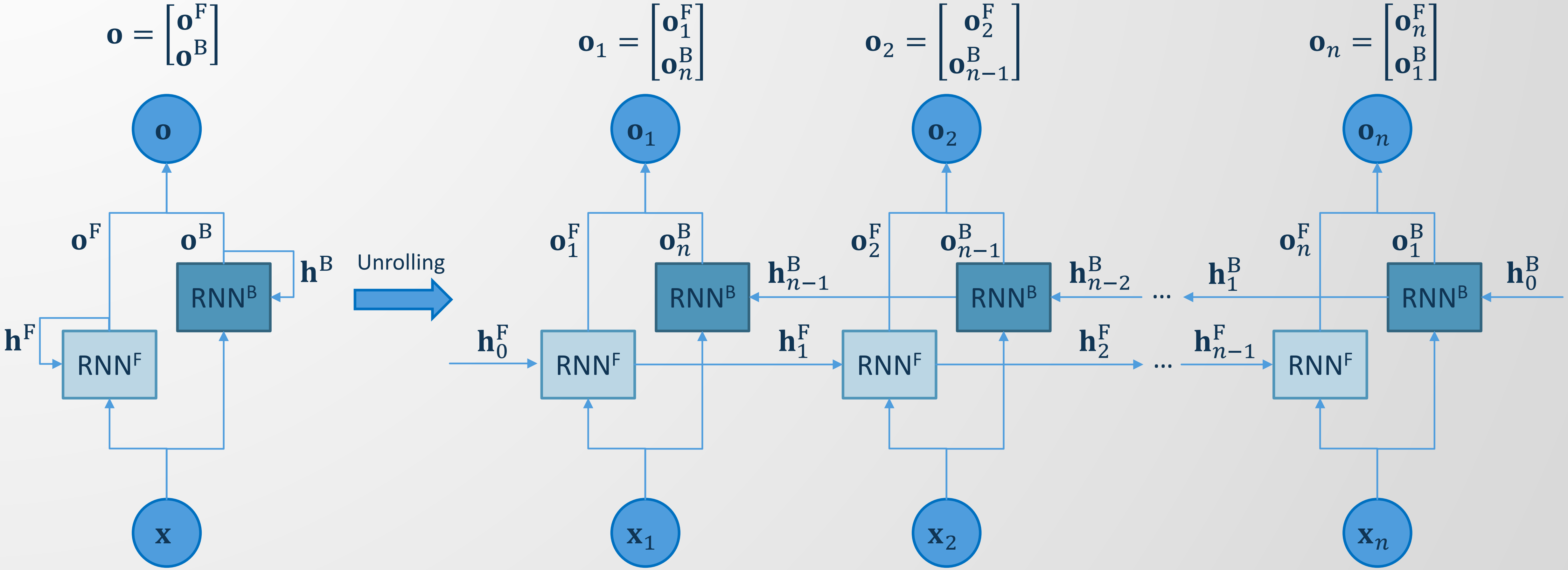
“I am ___ hungry, and I can eat half a pig”

- Depending on the amount of information available, we might fill in the blanks with very different words such as “happy”, “not”, and “very”.
- In such cases, a sequence model (such as RNNs) that is unable of taking advantage of future information will perform very poorly.

Bidirectional recurrent neural networks (2)

- Bidirectional Recurrent Neural Networks (BRNNs) are modified RNNs with ability to look both back and forth at every time step.
- A BRNN is composed of two RNNs running in opposite directions allowing them to receive information from both past and future states:
  - the input sequence of the first RNN is fed in normal time order;
  - the input sequence of the second RNN is fed in reverse time order;
  - the outputs of the two RNNs are concatenated at each time step.
- BRNNs are trained with similar algorithms as RNNs, since the two RNNs do not interact each other.

Bidirectional recurrent neural networks (3)



Autoencoders

> “*An autoencoder is a type of artificial neural network used to learn efficient data representations in an unsupervised manner.*”

From Wikipedia

What is an autoencoder?

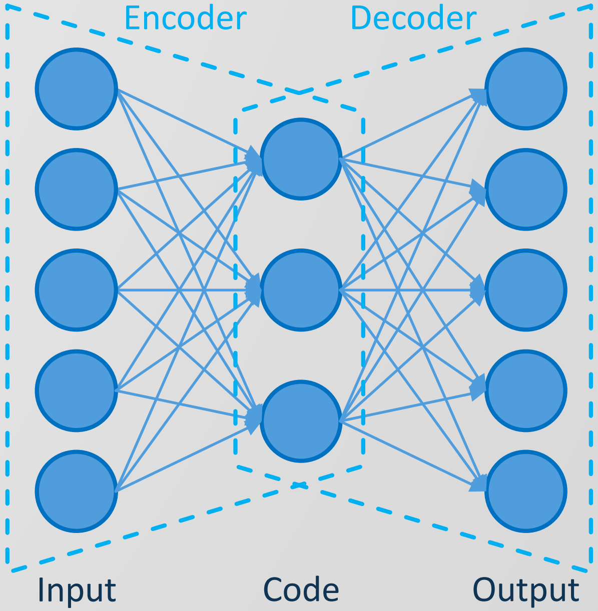
- The aim of an AutoEncoder (AE) is to learn a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore signal “noise”.
- AEs transform the input into a new representation (called *code* or *latent-space representation*) and then reconstruct the output from this representation.
- An AE is composed by:
  - an encoding function $E(\mathbf{x}): \mathbb{R}^n \longrightarrow \mathbb{R}^k$ outputting a latent representation $\mathbf{s}$;
  - a decoding function $D(\mathbf{s}): \mathbb{R}^k \longrightarrow \mathbb{R}^n$ computing the reconstructed output $\mathbf{o}$.

Applications

- Clustering
- Dimensionality reduction
- Classification
- Data generation
- Information retrieval
- Anomaly detection
- Data denoising
- Data reconstruction
- Machine translation
- Recommendation systems

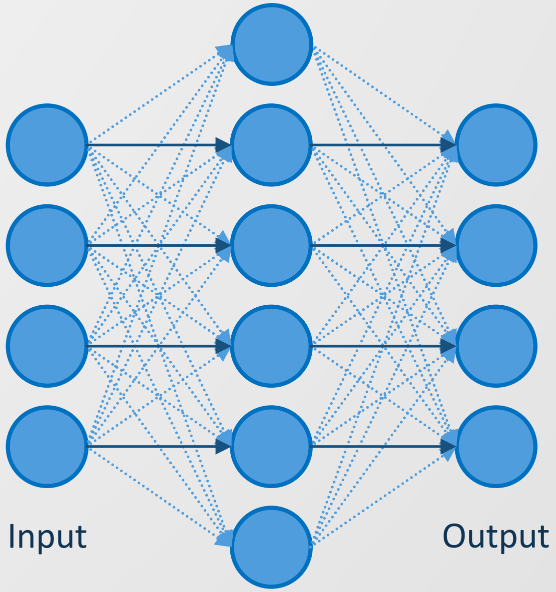
Autoencoder architecture

- The simplest form of an AE is a feed-forward neural network having an input layer and an output layer with the same number of neurons and one or more hidden layers connecting them.
- The purpose is to minimize the difference between the input and the output.
- An AE consists of two parts:
  - the encoder (E) – it compresses the input ($\mathbf{x}$) and produces the code ($\mathbf{s}$);
  - the decoder (D) – it reconstructs the input ($\mathbf{o}$) starting from the code ($\mathbf{s}$).
- An AE can be trained by minimizing the *reconstruction error*, $\mathcal{L}(\mathbf{x}, \mathbf{o})$, which measures the difference between the input and its reconstruction.



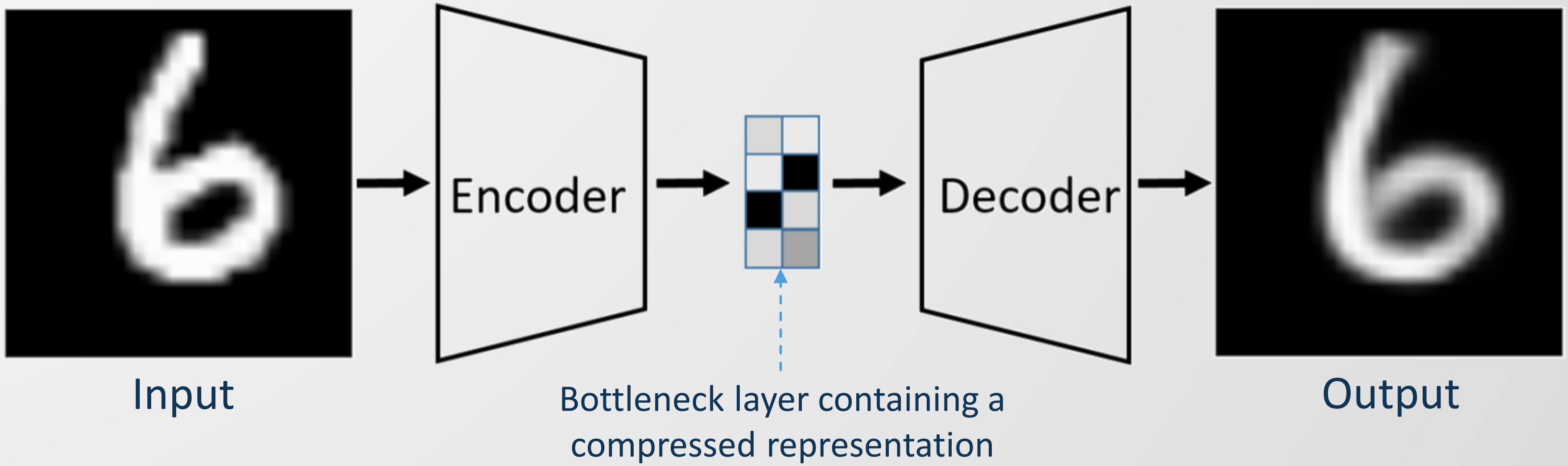
The risk of trivial identity

- If the only purpose of AEs is to copy the input to the output, they would be useless.
- The hope is that during training the latent representation will take on useful properties.
- The risk is the AE could learn the so-called *identity function*, so the output equals the input, and does not perform any useful representation learning or dimensionality reduction.



Undercomplete autoencoders

- To avoid this risk, the simplest solution is to use a bottleneck layer which forces a compressed knowledge representation of the original input constraining the amount of information that can traverse the full network ($k < n$).

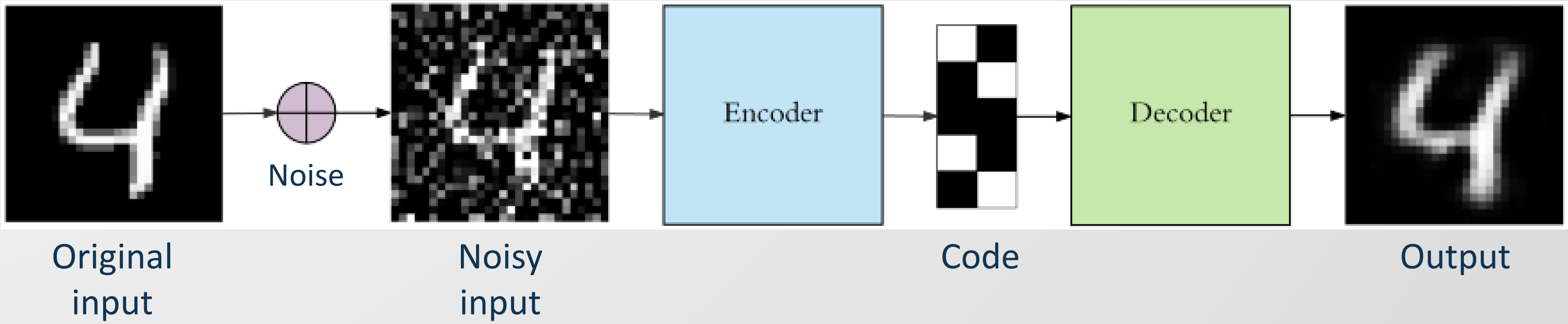


Different types of autoencoders

- Several variants exist to the basic model, with the aim of forcing the learned representations to assume useful properties:
  - denoising autoencoders;
  - sparse autoencoders;
  - variational autoencoders;
  - conditional variational autoencoders.

Denoising autoencoders

- Denoising AEs prevent the network learning the identity function by corrupting the input data on purpose (adding noise or masking some of the input values) and making it recover the original noise-free data.
- The AE cannot simply copy the input to its output, but it is forced to extract useful features that constitute better higher-level representations of the input.
- The input corruption is performed only during the training phase. Once the model has learnt the optimal parameters, in order to extract the representations from the original data no corruption is added.



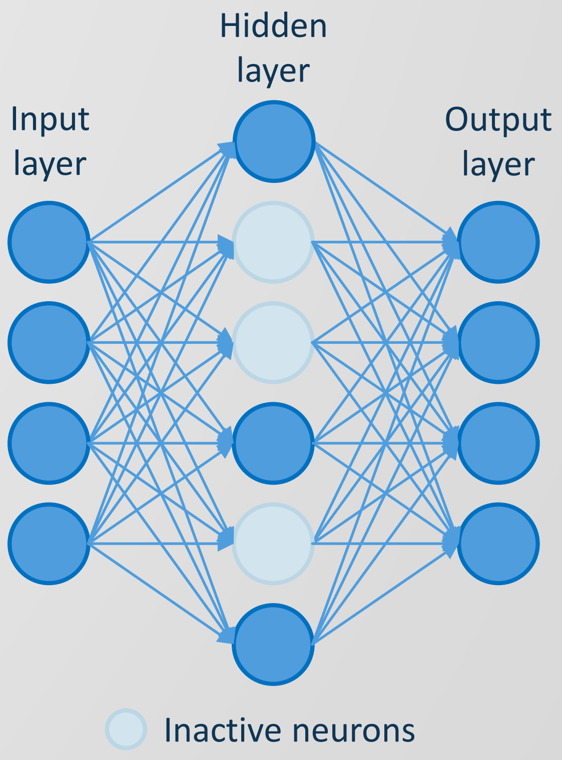
Sparse autoencoders

- Sparse AEs represent an alternative method to avoid that the model learns the identity function, without a reduction in the number of neurons, by using a *sparsity constraint*.
- A penalty term is added to the loss function such that only a fraction of the neurons become active:

$$\mathcal{L}(\mathbf{x}, \mathbf{o}) + \Omega(\mathbf{h})$$

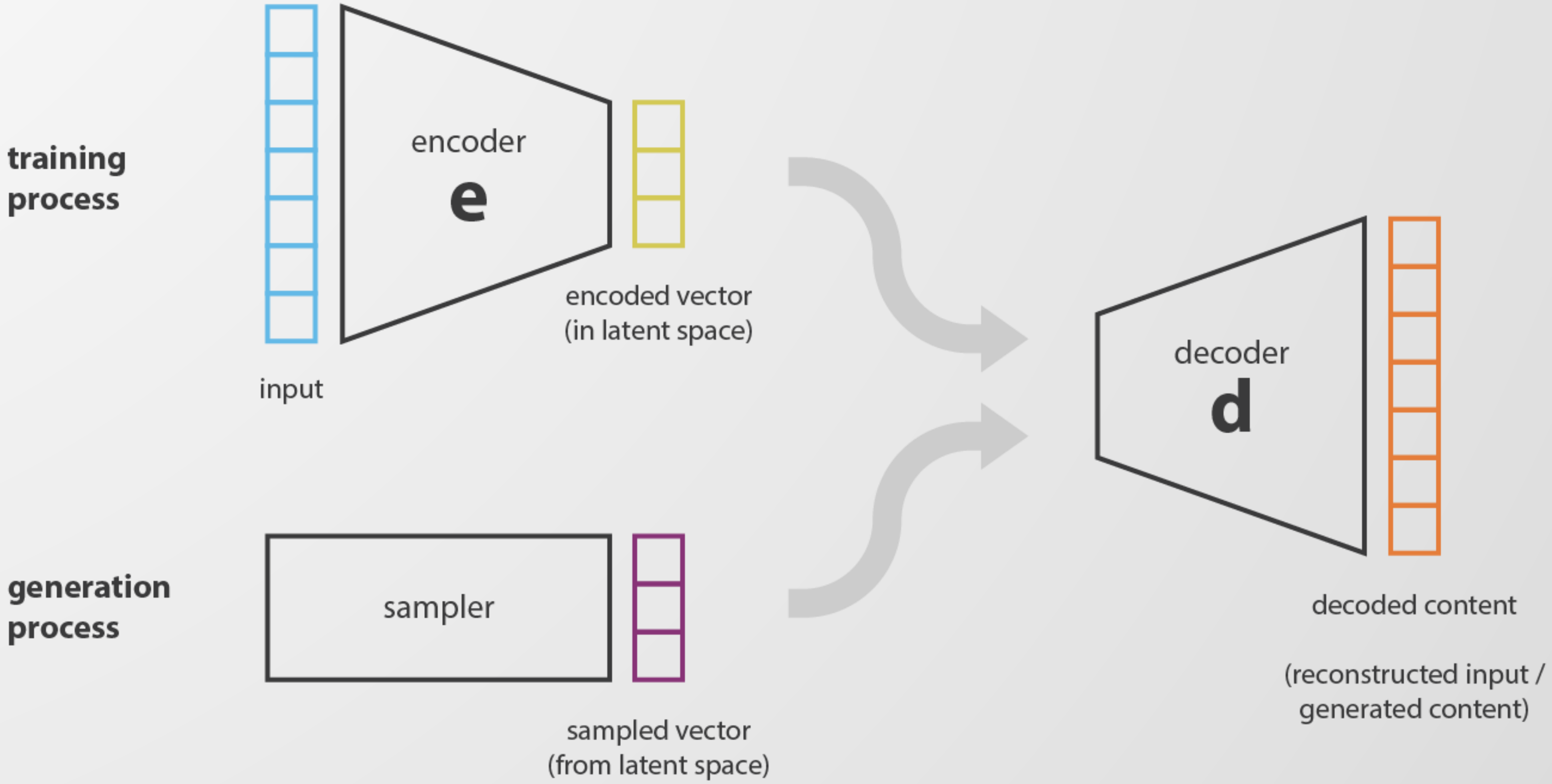
where Ω is a penalty function on hidden layer activations ($\mathbf{h}$).

- This forces the AE:
  - to represent each input as a combination of small number of neurons;
  - to discover interesting structure in the data.



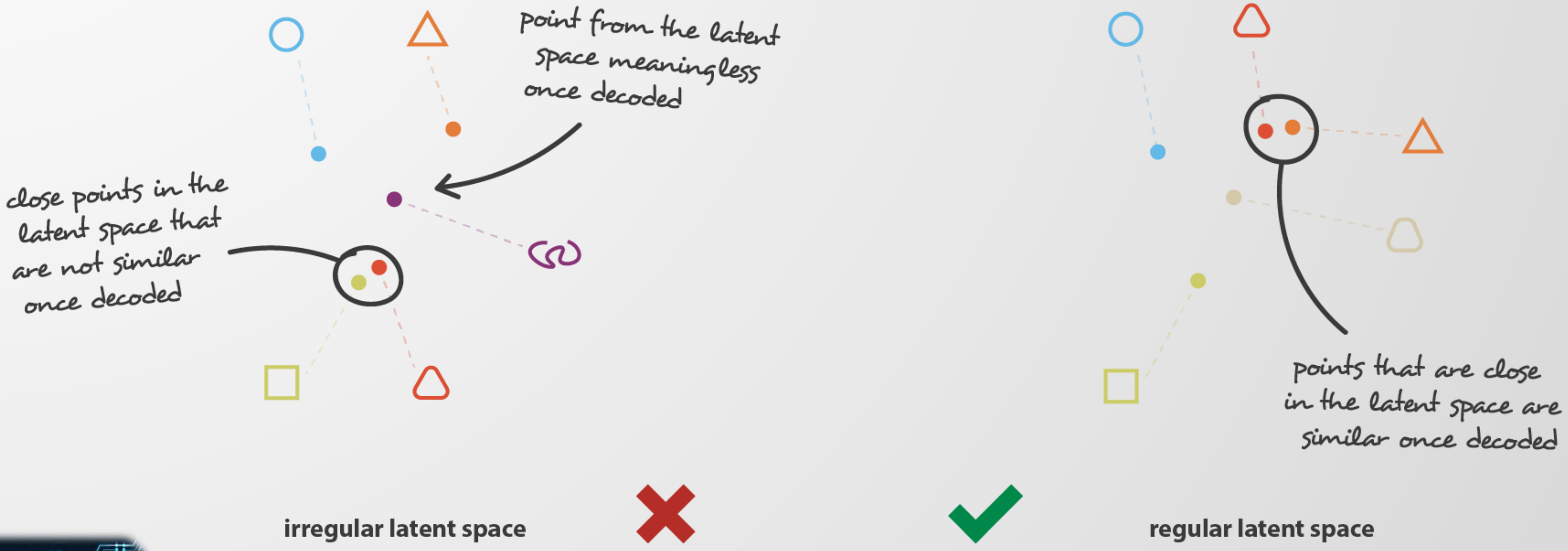
Can AEs be used to generate new data?

- Given an AE, can new data be generated by decoding points that are randomly sampled from the latent space?



Can AEs be used to generate new data? (2)

- The quality and relevance of generated data depend on the regularity of the latent space.

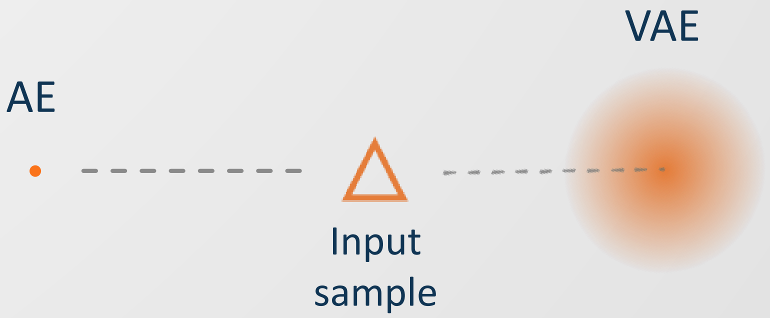


Can AEs be used to generate new data? (3)

- To make generative process possible, the latent space must satisfy two requirements:
    - *continuity* - two close points in the latent space should not give two completely different contents once decoded;
    - *completeness* - points in the latent space should give meaningful content once decoded.
- Unfortunately, it is very difficult (if not impossible) to ensure, a priori, that the latent space, created by the encoder, satisfies these requirements.

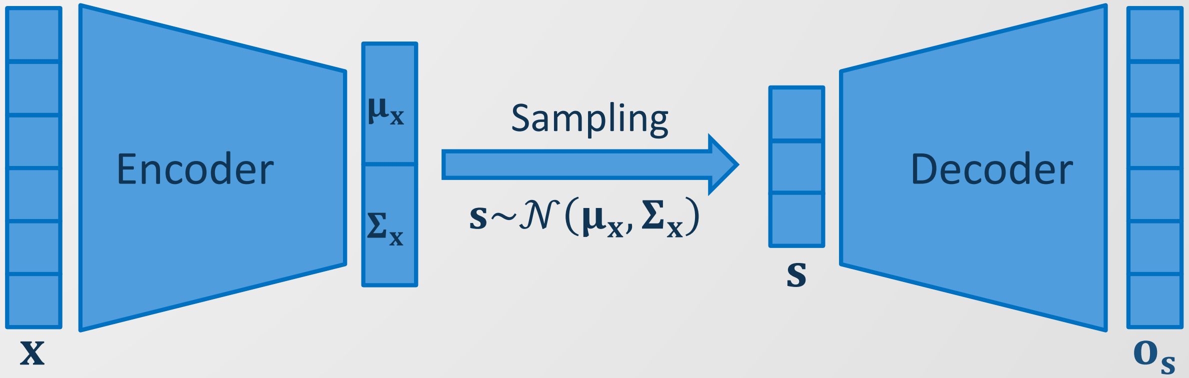
Variational autoencoders

- A Variational AutoEncoder (VAE) is an AE whose training is regularized to avoid overfitting and ensure that the latent space satisfies continuity and completeness requirements to enable generative process.
- To regularize the latent space, instead of encoding an input as a single point, each input is encoded into the parameters of a k-dimensional multivariate normal distribution (mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$) over the latent space of size k.



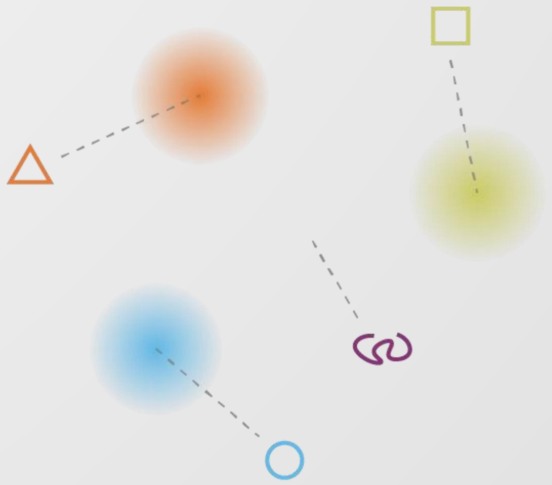
Variational autoencoders (2)

- The model is trained as follows:
  1. the input ($\mathbf{x}$) is encoded as distribution over the latent space ($\boldsymbol{\mu}_{\mathbf{x}}, \boldsymbol{\Sigma}_{\mathbf{x}}$);
  2. a point ($\mathbf{s}$) in the latent space is sampled from the normal distribution ($\mathcal{N}(\boldsymbol{\mu}_{\mathbf{x}}, \boldsymbol{\Sigma}_{\mathbf{x}})$);
  3. the sampled point ($\mathbf{s}$) is decoded ($\mathbf{o}_{\mathbf{s}}$), and the reconstruction error is computed;
  4. the reconstruction error is backpropagated through the network.



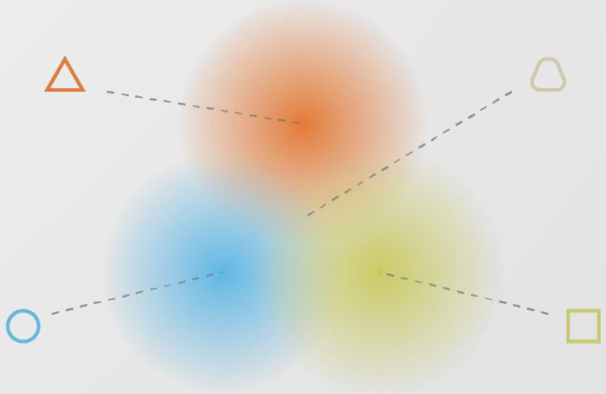
Variational autoencoders (3)

- The only fact that VAEs encode inputs as distributions instead of simple points is not sufficient to ensure continuity and completeness.
- Without a well-defined regularization term, the model can learn to ignore distributions and acting like almost classic AEs by returning:
  - distributions with very small variances (like punctual distributions);
  - distributions with very different means (far from each other in the latent space).



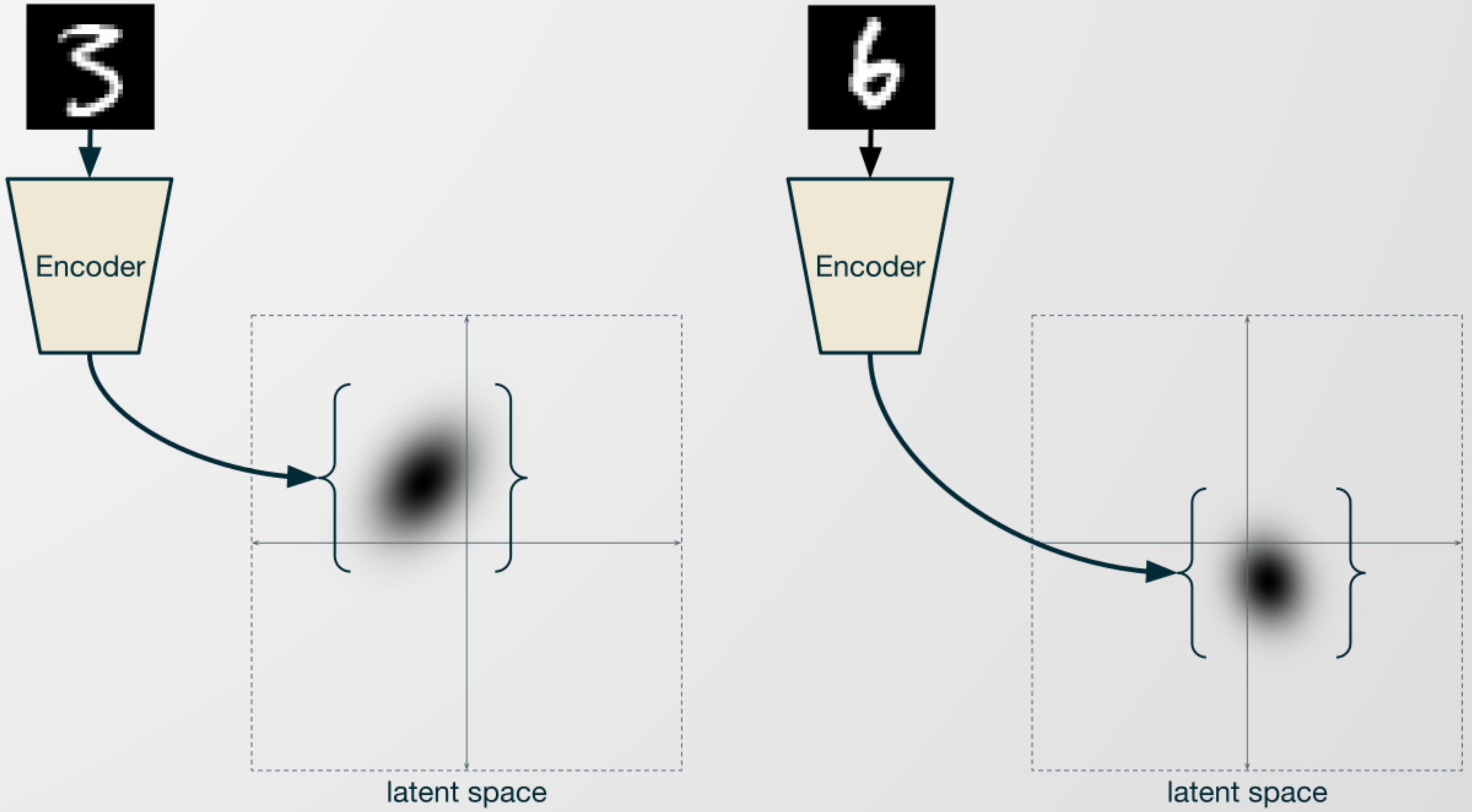
Variational autoencoders (4)

- To avoid these effects, both the covariance matrix and the mean of the distributions returned by the encoder need to be regularized.
- This regularization is done by enforcing distributions to be close to a standard normal distribution (with mean zero and covariance matrix equals to the identity matrix).
- In this way:
  - the covariance matrices will be close to the identity, preventing punctual distributions;
  - the mean will be close to zero, preventing distributions to be too far apart from each other.



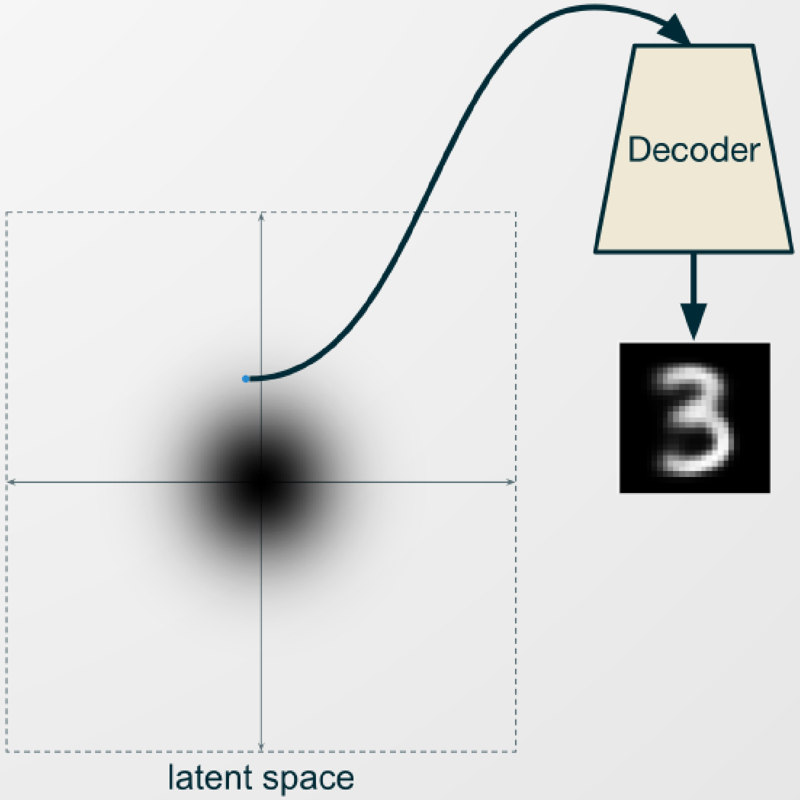
Variational autoencoders – an example

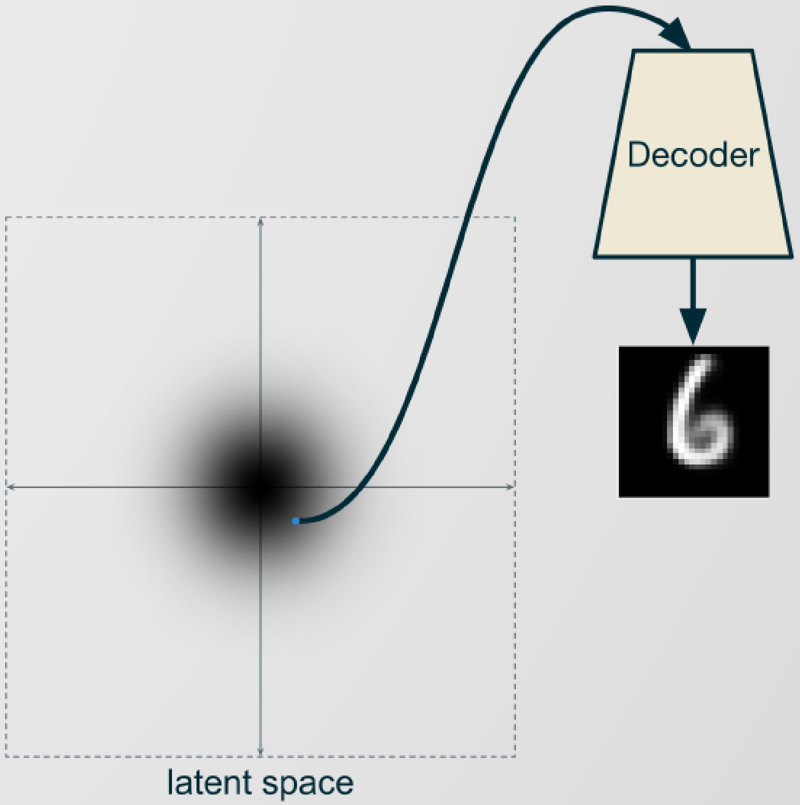
- The encoder takes in handwritten digit images and produces probability distributions in the latent space.



Variational autoencoders – an example (2)

- The decoder can produce reasonable handwritten digit images given sampled points from the latent distribution.



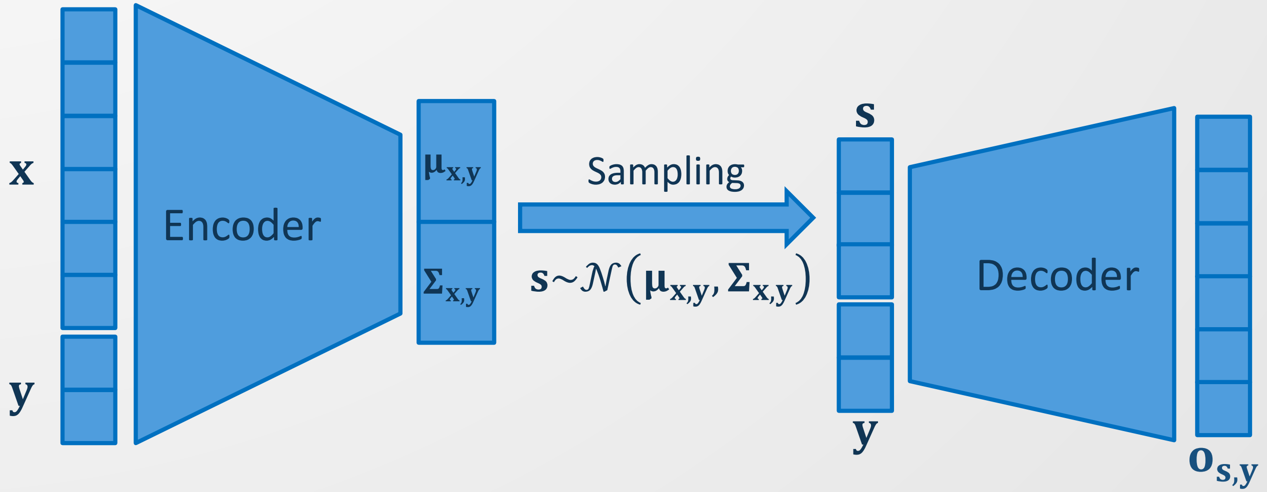


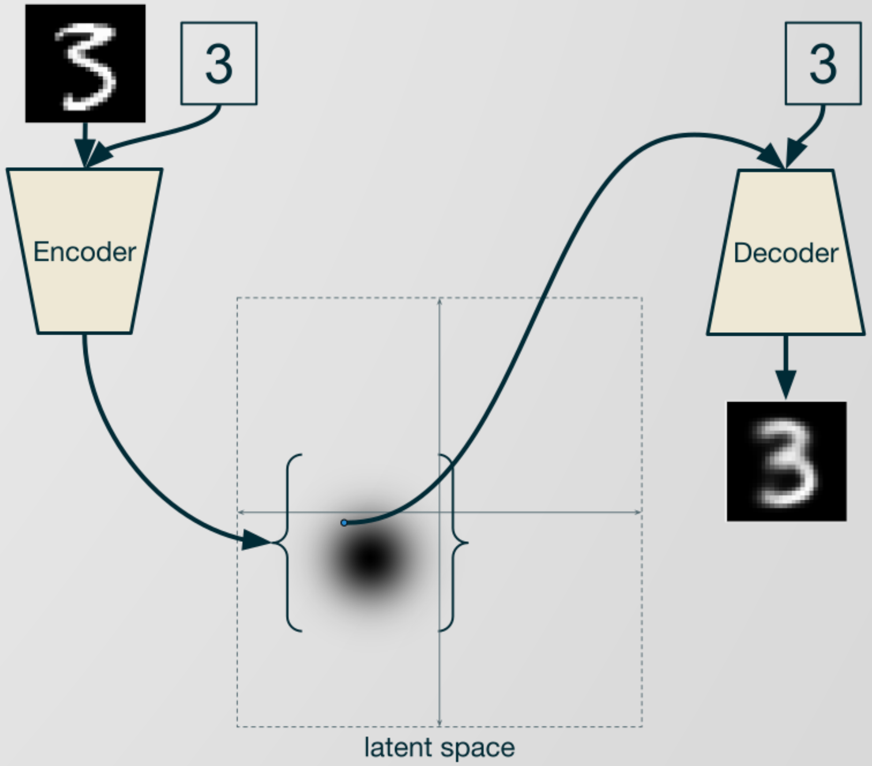
Can VAEs be used to generate specific data?

- A VAE cannot generate specific data (e.g., a *particular number on demand*) by decoding a point randomly sampled from the latent distribution.
- It is because the encoder models the latent space directly based on the input not caring about its type.
- Similarly, the decoder models the output based only on the point sampled from the latent distribution.

Conditional variational autoencoders

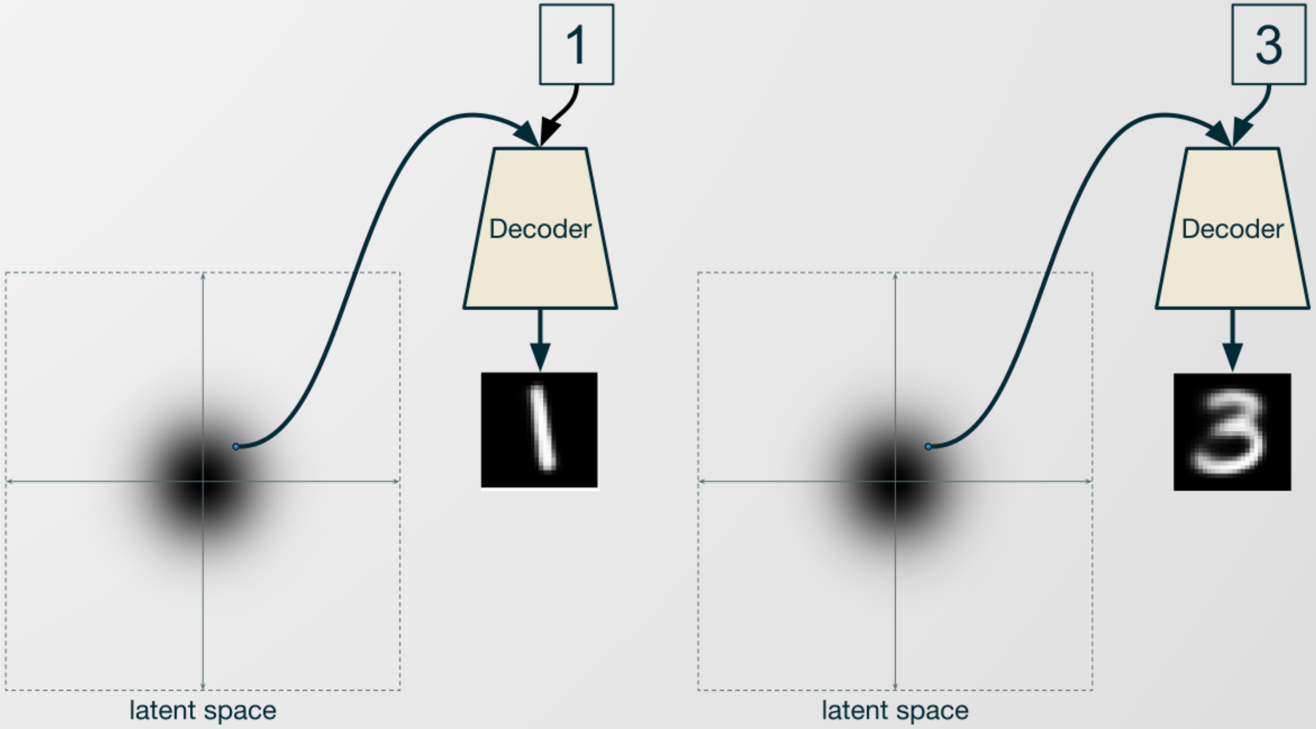
- A Conditional Variational AutoEncoder (CVAE) is a VAE with an extra input to both the encoder and the decoder to shape the entire generative process on a specific input.
- At training time, the input type $\mathbf{y}$ (i.e., the class, label or category) is provided to both the encoder and decoder.





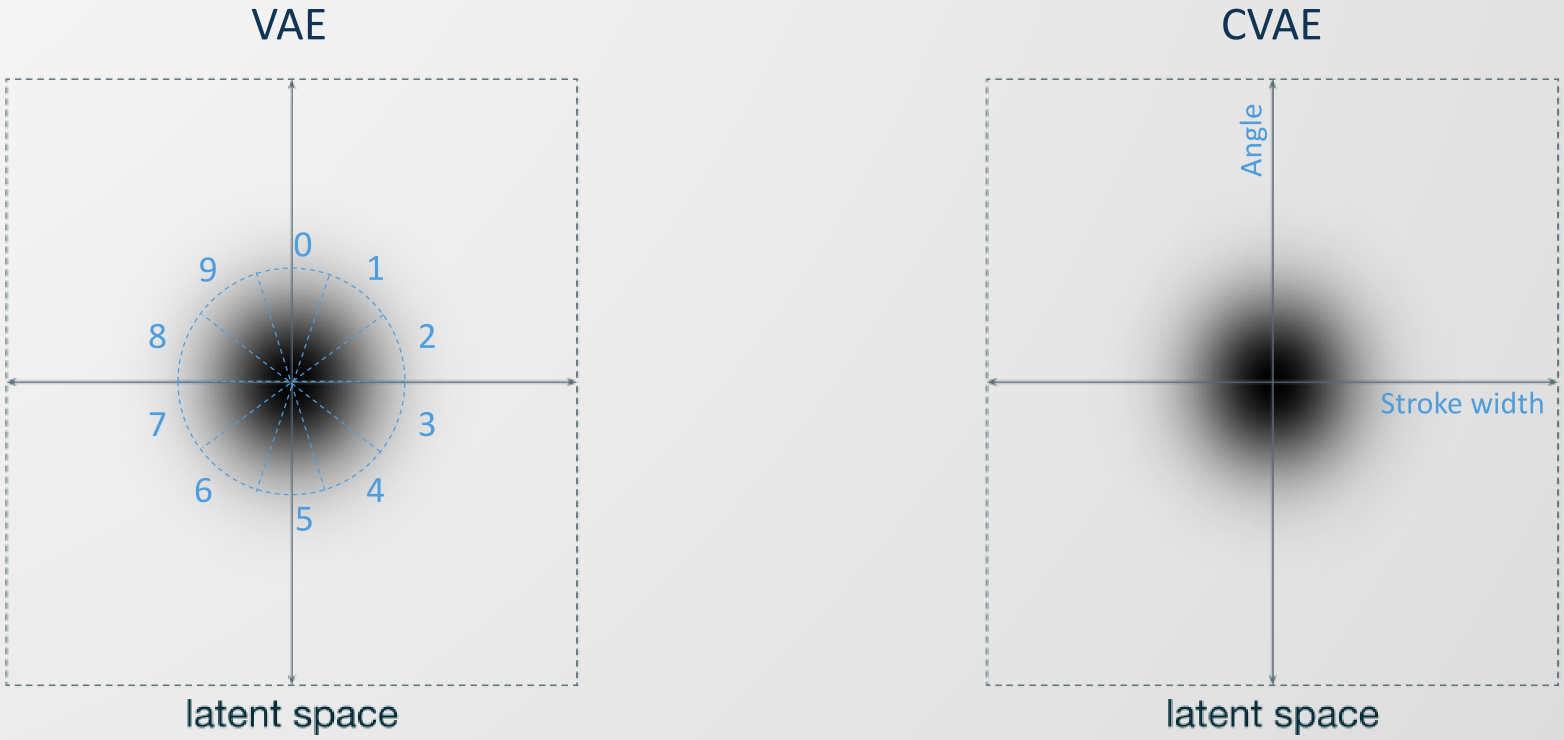
Conditional variational autoencoders (2)

- To generate a specific output, the desired type is fed into the decoder along with a random point sampled from the latent distribution.
- If the same latent point is fed in to produce two different outputs, the process will work correctly, since the system no longer relies on the latent space to encode the type.



Conditional variational autoencoders (3)

- In a CVAE, the latent space encodes other information.
- In the handwritten digit example, it could encode information such as stroke width or the angle at which the number is written.



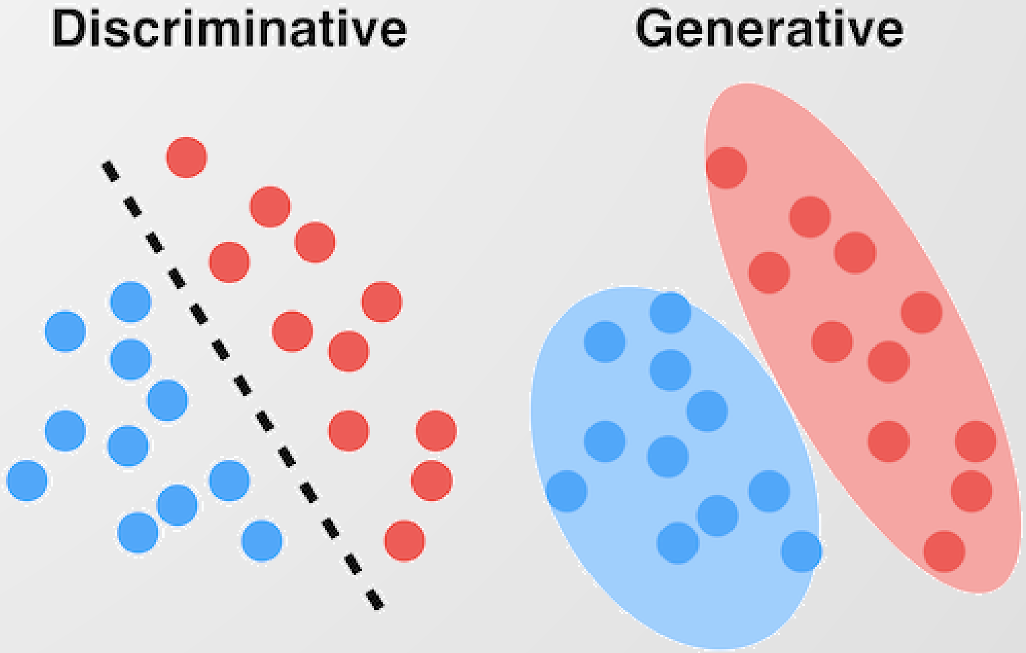
Generative models

> *“Approaches that explicitly or implicitly model the distribution of inputs as well as outputs are known as generative models, because by sampling from them it is possible to generate synthetic data points in the input space.”*

From Pattern Recognition and Machine Learning

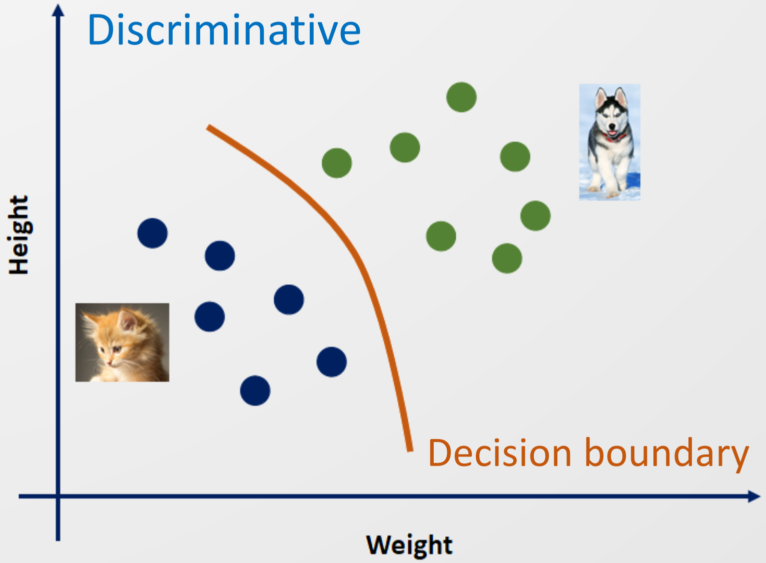
Discriminative vs generative modeling

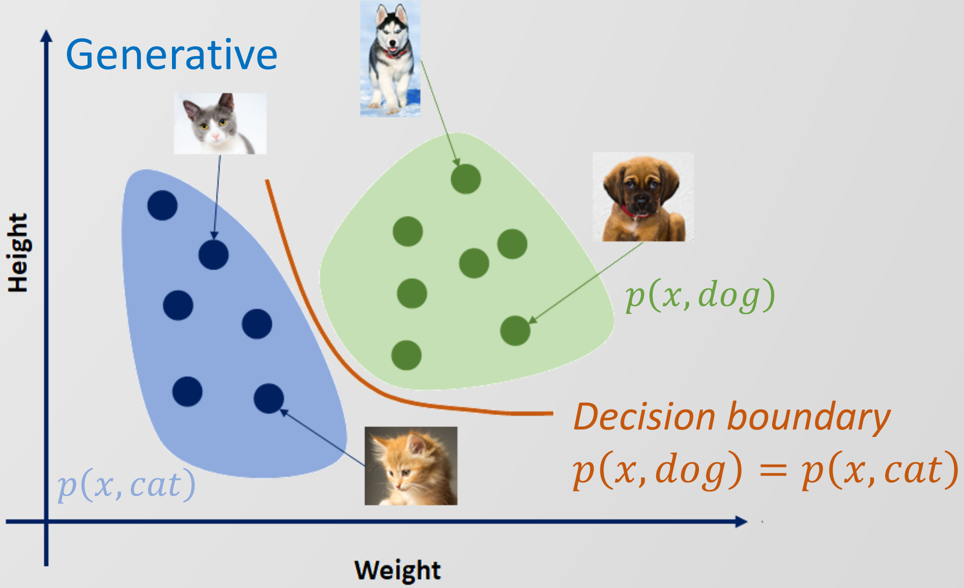
- The fundamental difference between discriminative and generative models is:
  - discriminative models learn the boundary between classes;
  - generative models explicitly/implicitly model the distribution of individual classes.



Discriminative vs generative modeling (5)

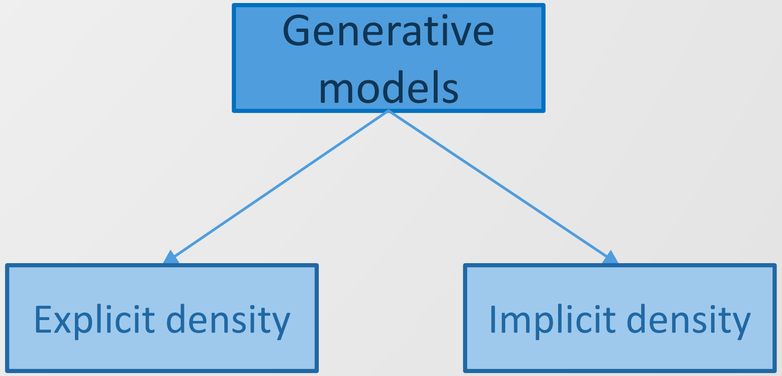
- Example - classify an animal as a cat or a dog based on weight and height:
  - a discriminative approach finds a decision boundary that separates cats and dogs and checks on which side of the decision boundary the new observation falls.
  - a generative approach builds models of what cats and dogs like and compares the new observation against the two models.





Classes of generative models

- Generative models can be divided into two categories:
  - explicit density models define an explicit density function $p_{model}(x)$ similar to $p_{data}(x)$;
  - Implicit density models define a stochastic process that, after training, aims to draw samples from the underlying data distribution $p_{data}(x)$ without explicitly defining it.



Generative models

Explicit density

Implicit density

Classes of generative models (2)

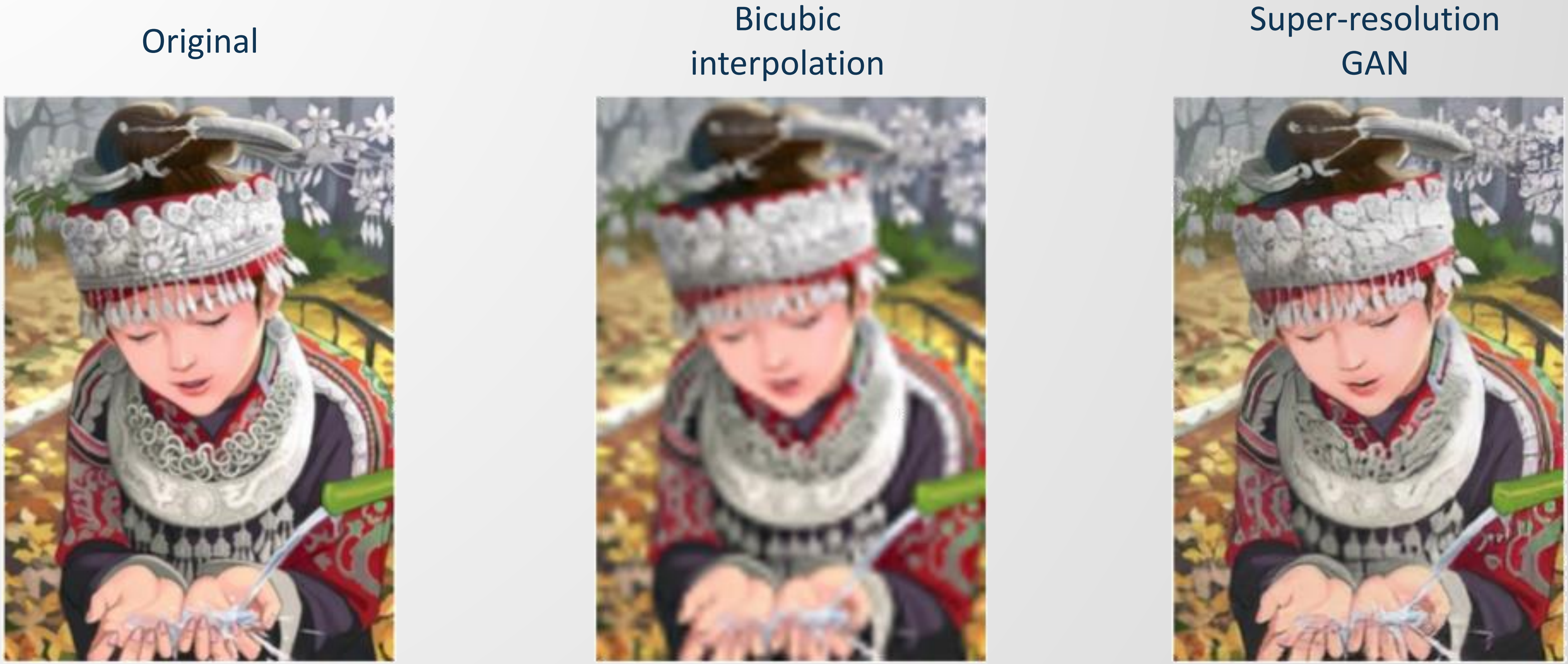
- The main difficulty in designing an explicit model is to capture all of the complexity of the data to be generated while still maintaining computational tractability.
- What if we want to explicitly model the distribution of horse images in order to generate new full HD imaginary horses?
  - Full HD (1920×1080) RGB images have more than 6M dimensions. It is impossible to deal with functions in such high-dimensional space.
- Implicit models do not care about the data distribution, their objective is to produce outputs as similar as possible to the real ones.

Applications

- Image denoising
- Image inpainting
- Image super-resolution
- Image generation
- Image-to-image translation
- Text-to-image synthesis
- Exploration in reinforcement learning
- Neural network pretraining
- Language generation
- Text-to-speech
- Imitation learning
- Classification

Applications – image inpainting

Applications – image super-resolution



Applications – image generation

Applications – image-to-image translation

edges → shoes

horse → zebra

orange → apple

Deep generative models

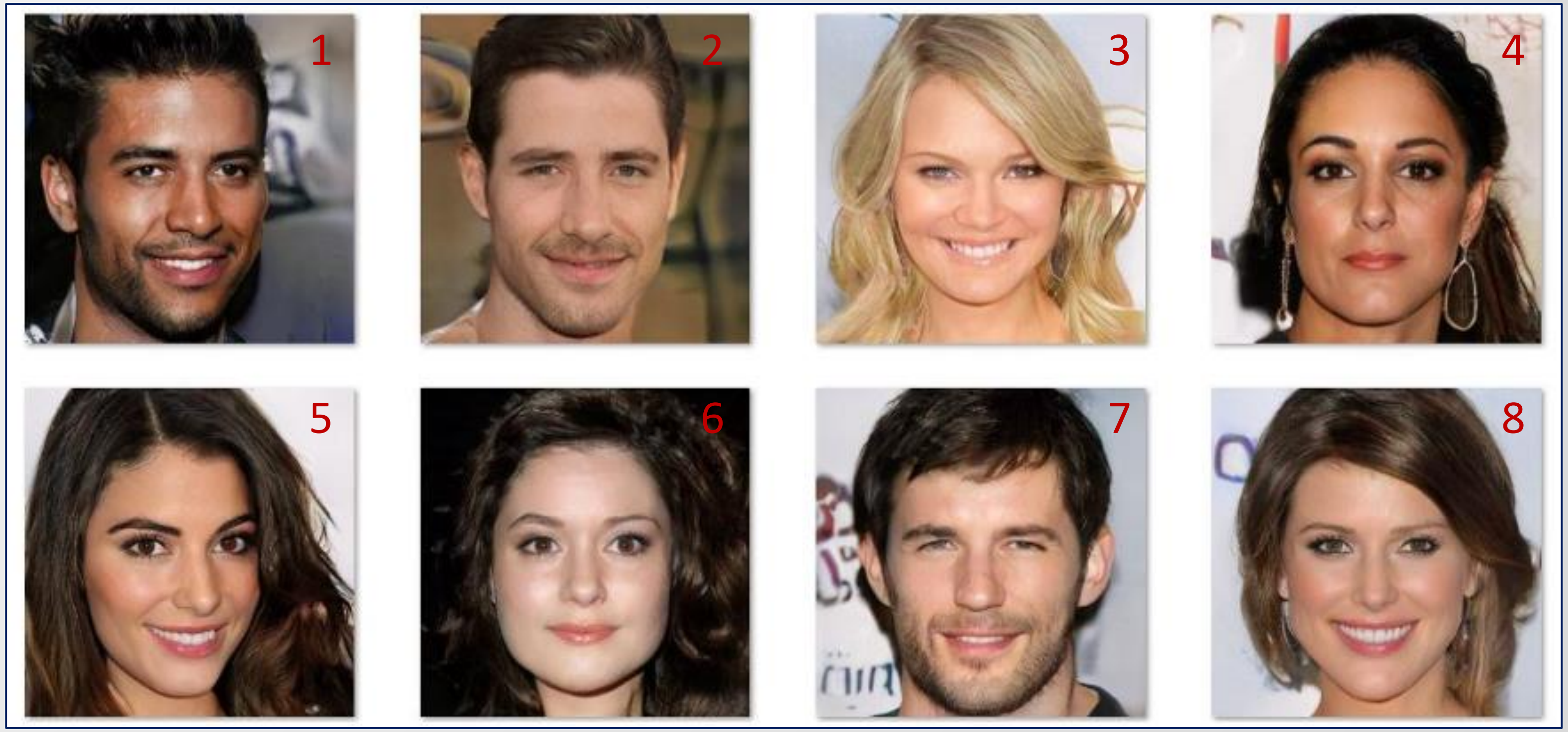
- Deep generative models are formed through the combination of generative models and deep neural networks.
- The basic idea is to force the model to discover and efficiently internalize the essence of the data in order to generate it.
- The most popular deep generative models are:
  - variational autoencoders (explicit density models);
  - generative adversarial networks (implicit density models).

> “*Generative adversarial networks are the most interesting idea in the last 10 years in machine learning.*”
>
> Yann LeCun

Generative adversarial networks

- Generative Adversarial Networks (GANs) are a model architecture for training a generative model designed by I. J. Goodfellow and his colleagues in 2014.
- GANs rely on the idea that a generator is good if we cannot tell fake data apart from real data.

Which of these photos is a fake?



Generative adversarial networks – architecture

- The GAN model architecture consists of two sub-models:
  - the generator G is trained to produce plausible data (fake);
  - the discriminator D is trained to distinguish the generator's fake data from real examples.
- The two models are trained together, in an adversarial game, until the discriminator model is no longer able to distinguish a real from a fake example.
- When training begins, the generator produces obviously fake data, and the discriminator quickly learns to tell that it is fake.

Generated data

Discriminator

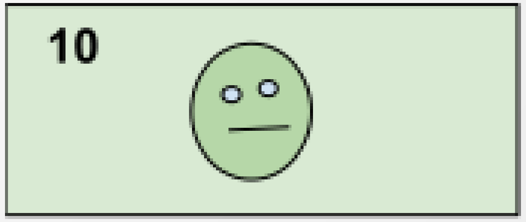
FAKE REAL

Real data

Generative adversarial networks – architecture (2)

- As training progresses, the generator gets closer to producing output that can fool the discriminator.

Generated data



Discriminator

FAKE REAL

Real data

- Finally, the discriminator starts to classify fake data as real, and its accuracy decreases.

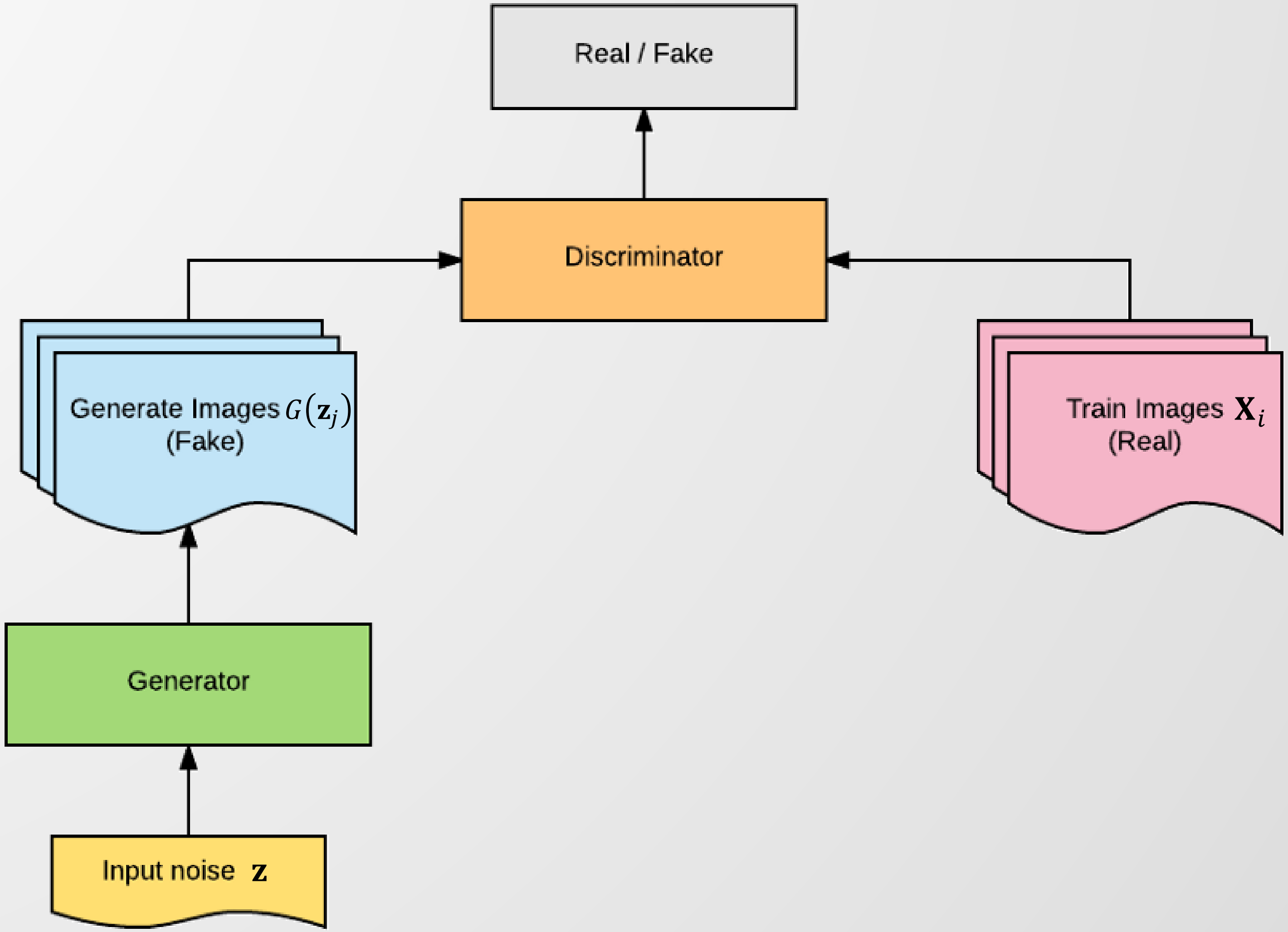
Generated data

Discriminator

REAL REAL

Real data

Generative adversarial networks – architecture (3)



Generative adversarial networks – training

- GAN training proceeds in alternating periods:
  1. the discriminator D trains for one or more epochs (keeping the generator constant);
  2. the generator G trains for one or more epochs (keeping the discriminator constant);
  3. repeat steps 1 and 2 to continue training the discriminator and generator.
- As the generator improves with training, the discriminator performance gets worse because the discriminator cannot easily tell the difference between real and fake.
- If the generator succeeds perfectly, then the discriminator has a 50% accuracy.

Generative adversarial networks – training (2)

- The discriminator feedback gets less meaningful over time. If the GAN continues training past the point when the discriminator is giving completely random feedback, then the generator starts to train on junk feedback, and its own quality may collapse.
- For this reason, it is important to monitor the quality of the generated output and stop training once the discriminator has lost the game to the generator.

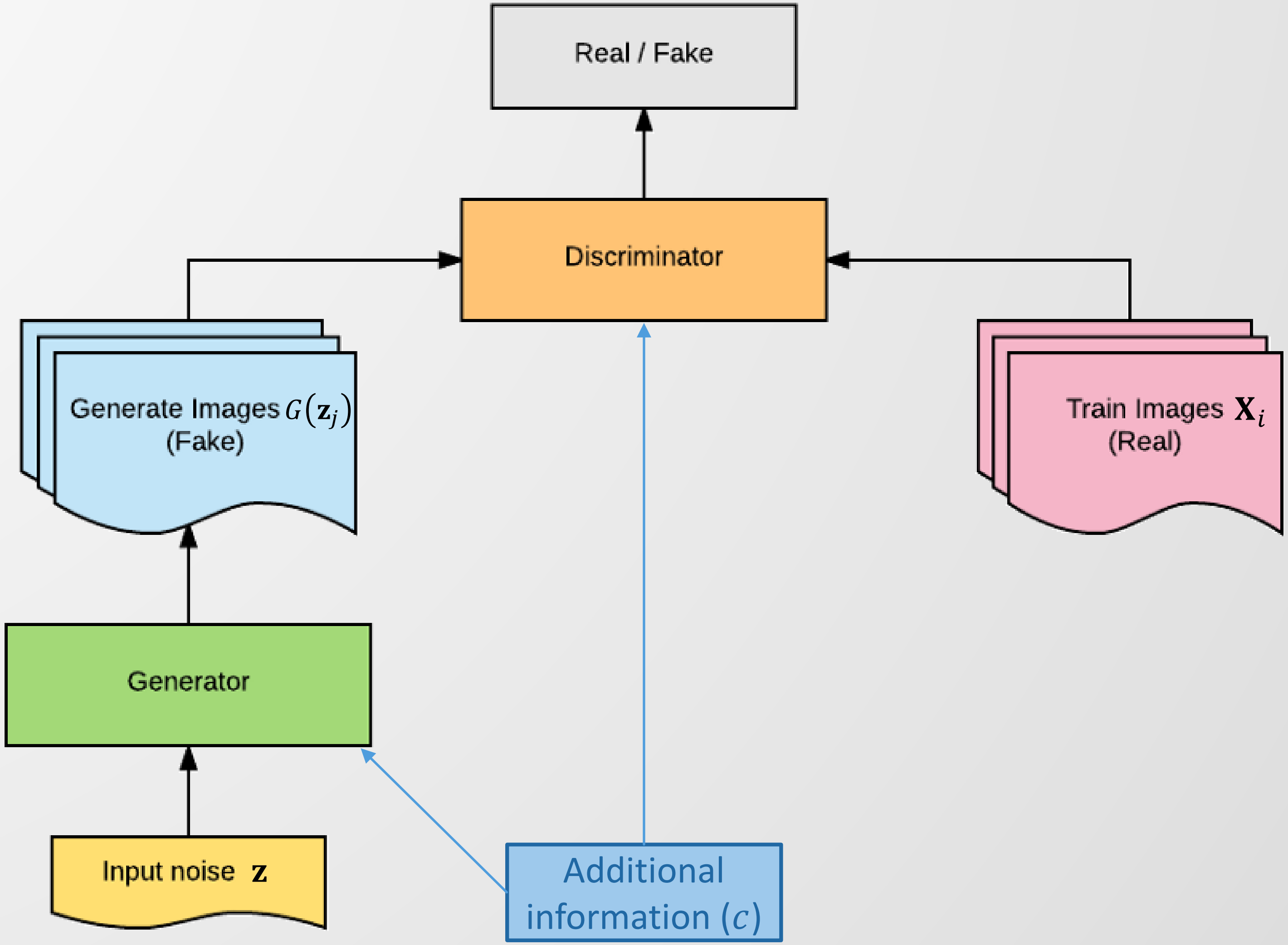
Generative adversarial networks – variations

- Several variations of original GAN architecture have been proposed in literature to solve specific problems including:
  - Conditional GANs;
  - Deep Convolutional GANs;
  - Pix2Pix;
  - CycleGANs.

Conditional GANs

- Is there any way to provide extra information to the model about what type of output we want to generate?
- In 2014, M. Mirza and S. Osindero proposed an extension to GAN architecture (called conditional GAN - cGAN) allowing to condition the data generation process by providing additional information to both generator and discriminator.
- The additional input (c) can be any kind of auxiliary information, such as class labels or data from other modalities.
- cGANs are used in a variety of tasks such as:
  - text-to-image generation;
  - image-to-image translation.

Conditional GANs – architecture

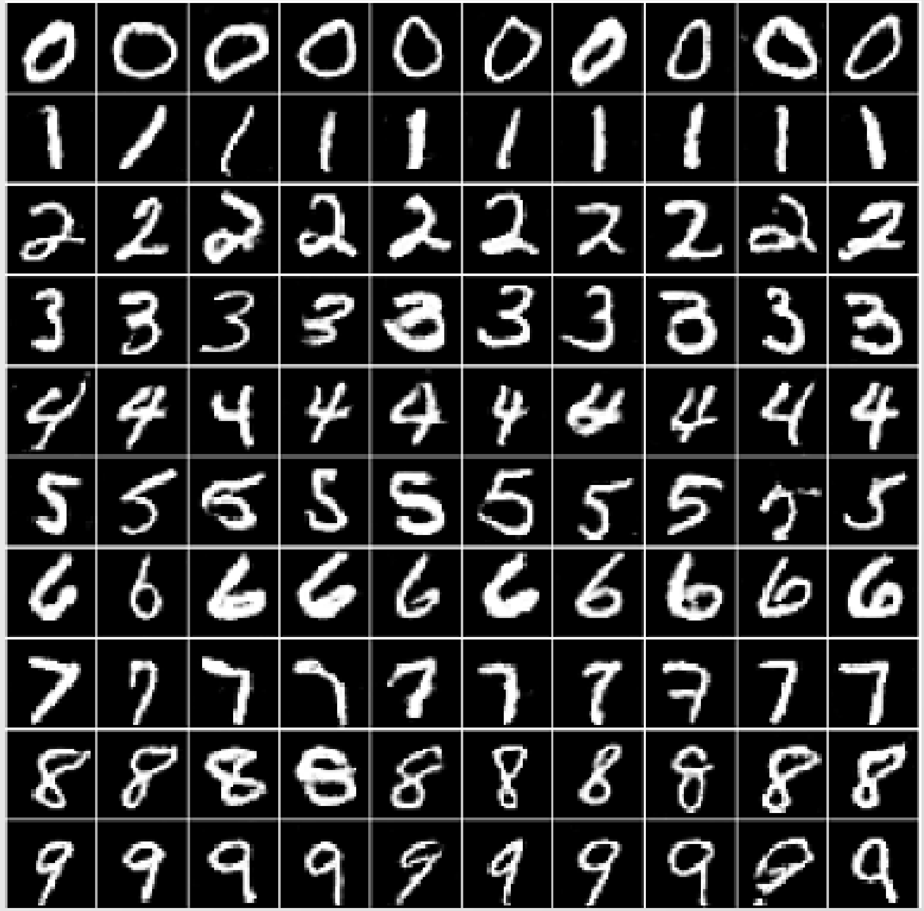


Additional information (c)

Conditional GANs – results

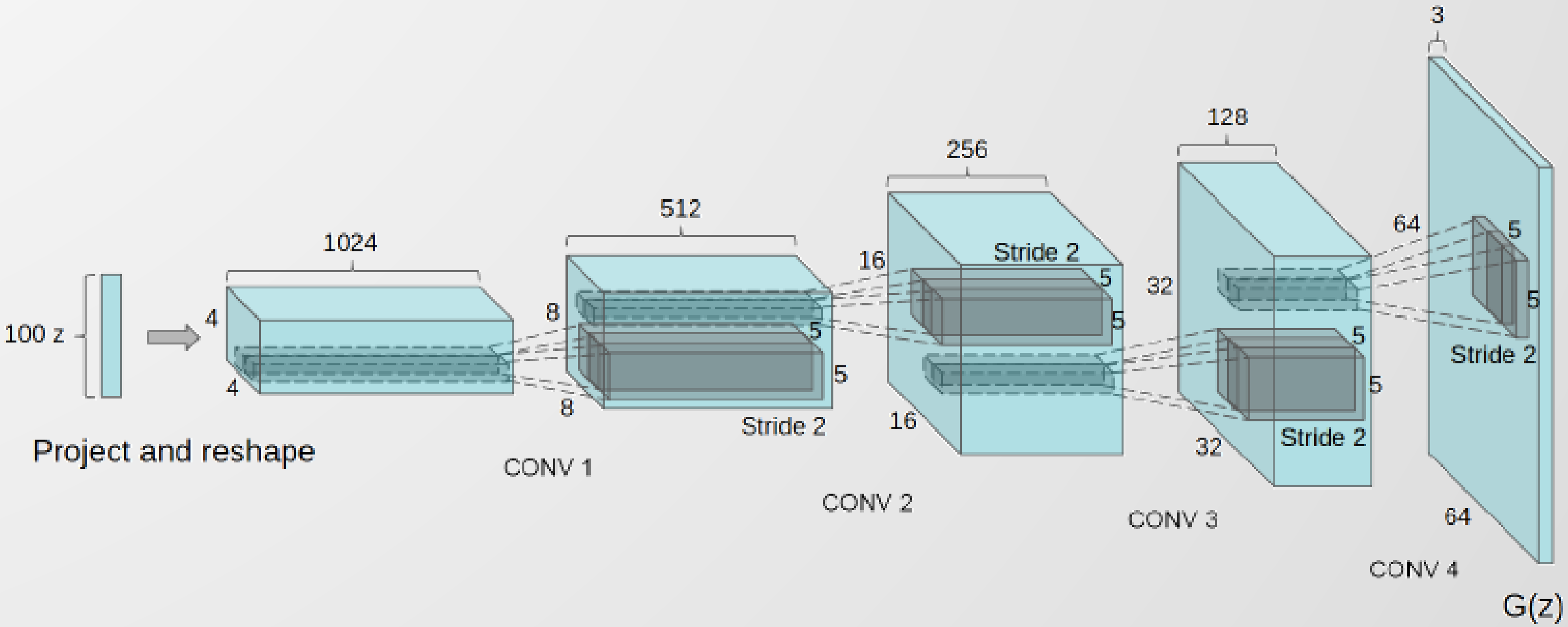
GAN

cGAN



Deep convolutional GANs

- Deep Convolutional GAN (DCGAN) is a generative adversarial network architecture, designed in 2015 by A. Radford, L. Metz and S. Chintala, for unsupervised learning.
- Before the introduction of DCGANs, several attempts to improve GANs to model images using CNNs have been unsuccessful primarily due to training instability.

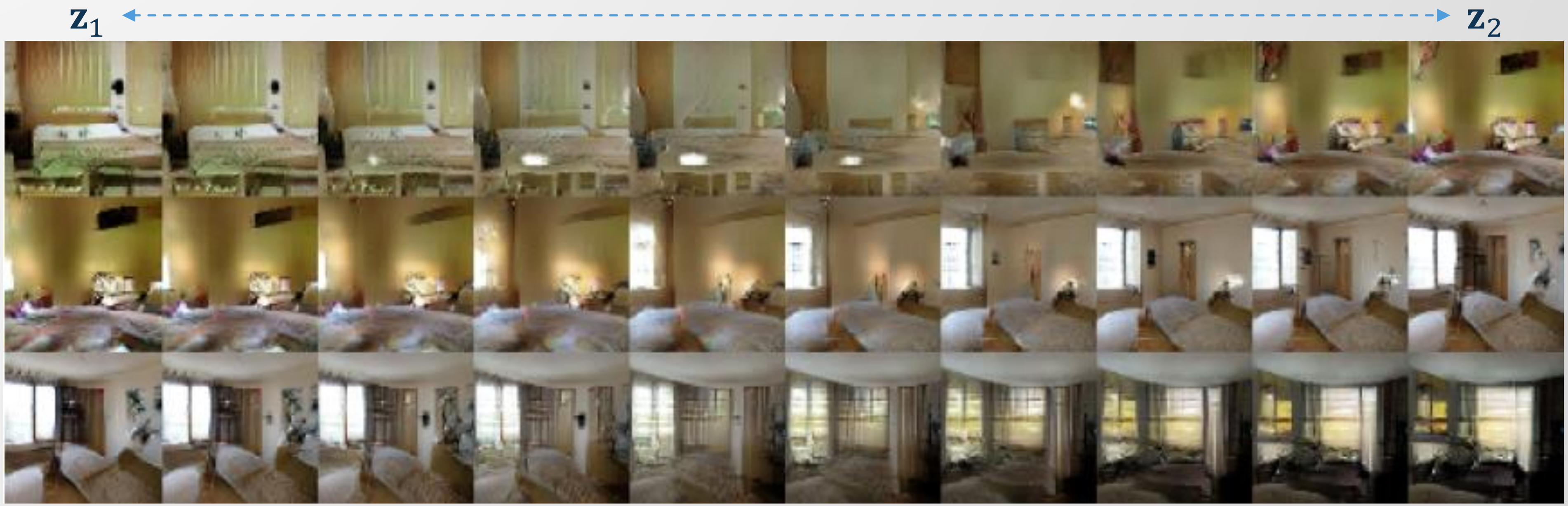


Deep convolutional GANs – results

- Some generated images of bedrooms after training on the LSUN bedrooms dataset.

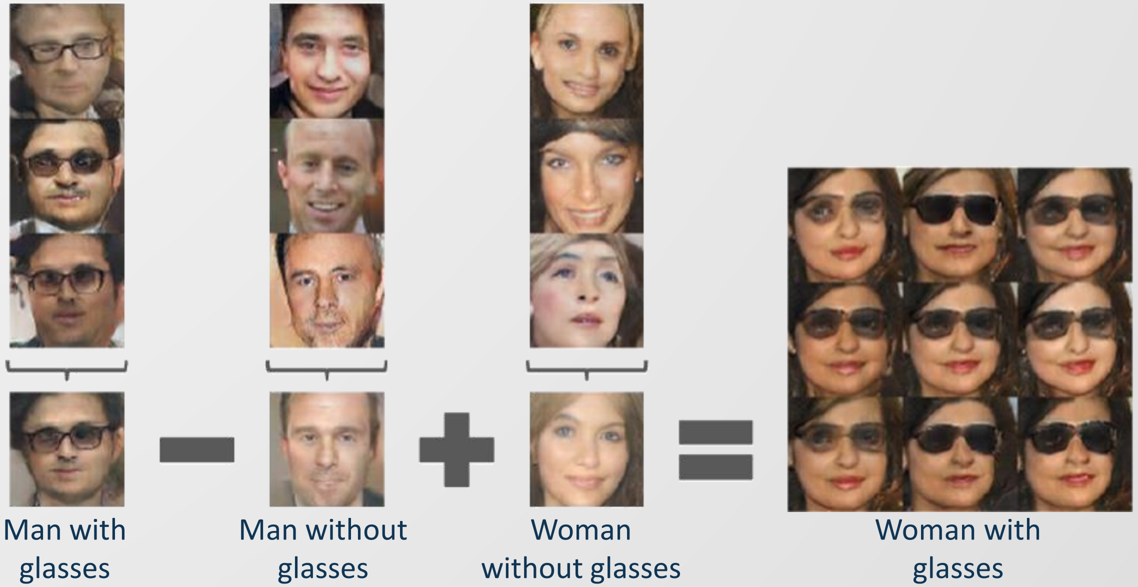
Deep convolutional GANs – results (2)

- The authors showed that the generator has interesting vector arithmetic properties using which the generated images can be manipulated:
  - Input vector interpolation
    - The input vector $\mathbf{z}$ is a n dimensional vector in a n dimensional space.
    - If interpolation is performed between two input vectors $\mathbf{z}_1$ and $\mathbf{z}_2$, a gradual change can be seen.



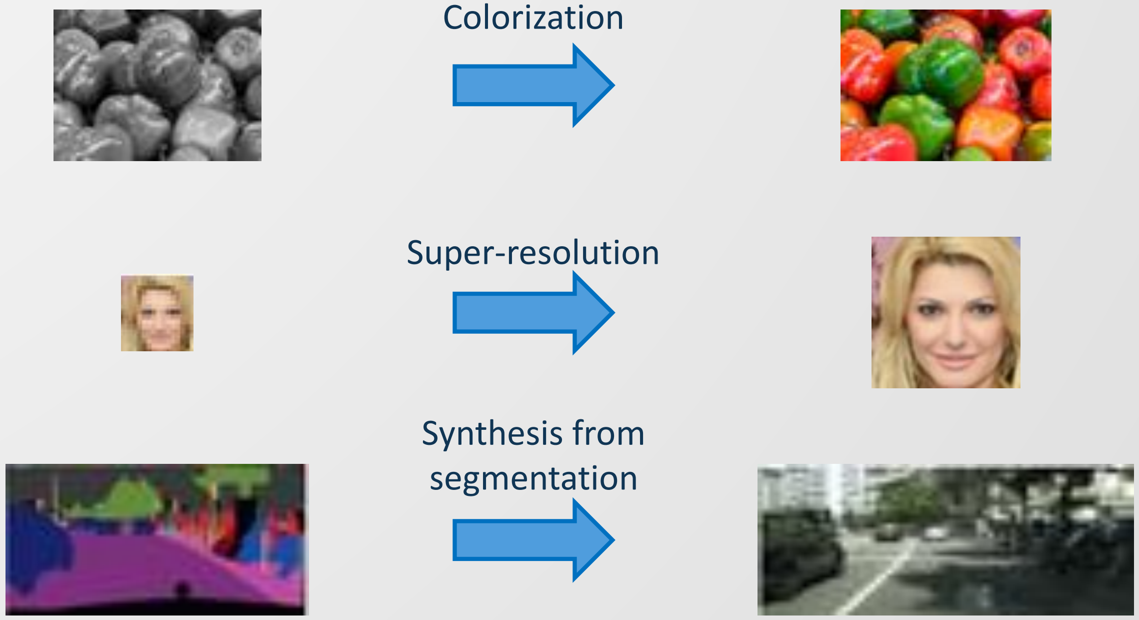
Deep convolutional GANs – results (3)

- Vector arithmetic
  - Simple arithmetic operations revealed rich linear structure in representation space.
  - e.g.: $\left(\mathbf{z}_{man\ with\ glasses} - \mathbf{z}_{man\ without\ glasses}\right) + \mathbf{z}_{woman\ without\ glasses}$ can result in a vector whose nearest neighbor was the vector for $woman\ with\ glasses$.
  - Experiments working on only single samples were unstable. Averaging the $\mathbf{z}$ vector for multiple exemplars showed consistent and stable generations that semantically obeyed the arithmetic.



Pix2Pix

- Pix2Pix is a cGAN model proposed in 2016 for general purpose image-to-image translation by P. Isola, J. Zhu, T. Zhou and A. A. Efros.
- Image-to-image translation is the task of taking images from one domain and transforming them so they have the style (or characteristics) of images from another domain.



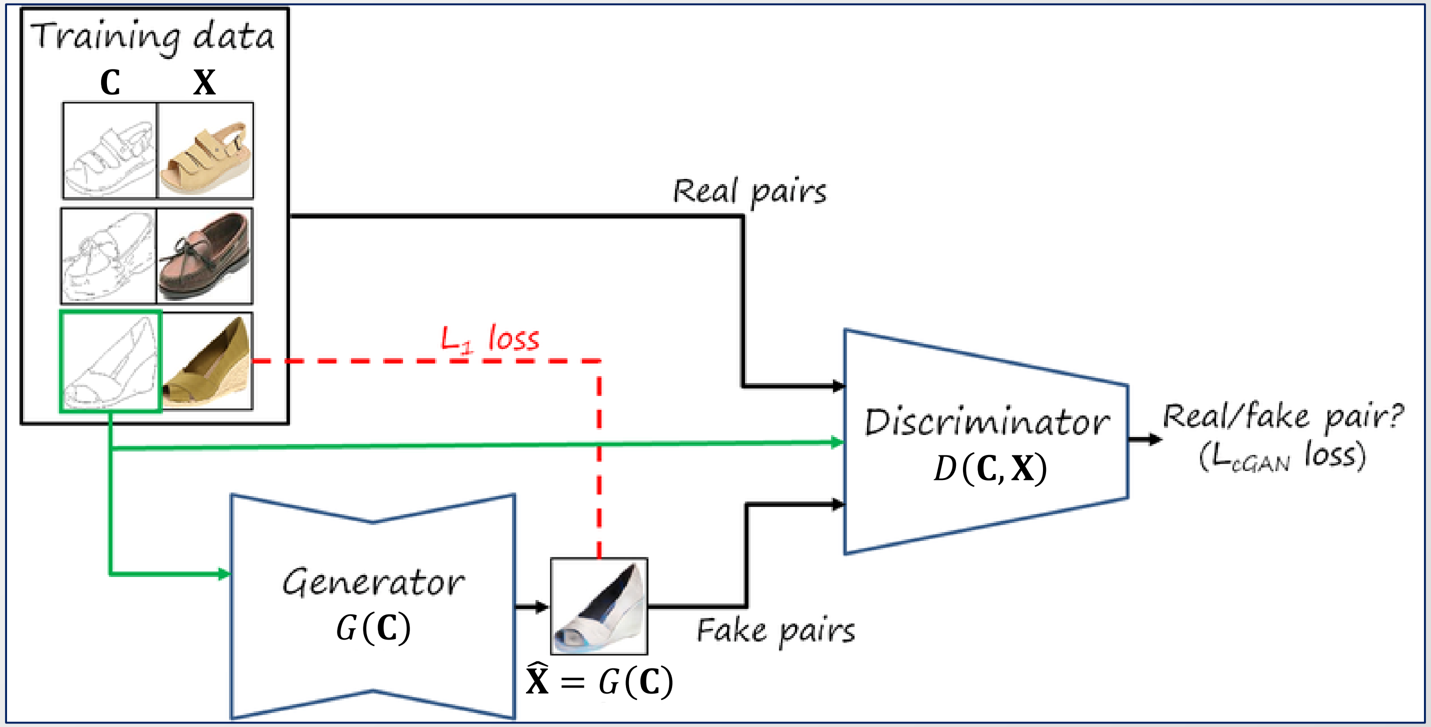
Colorization

Super-resolution

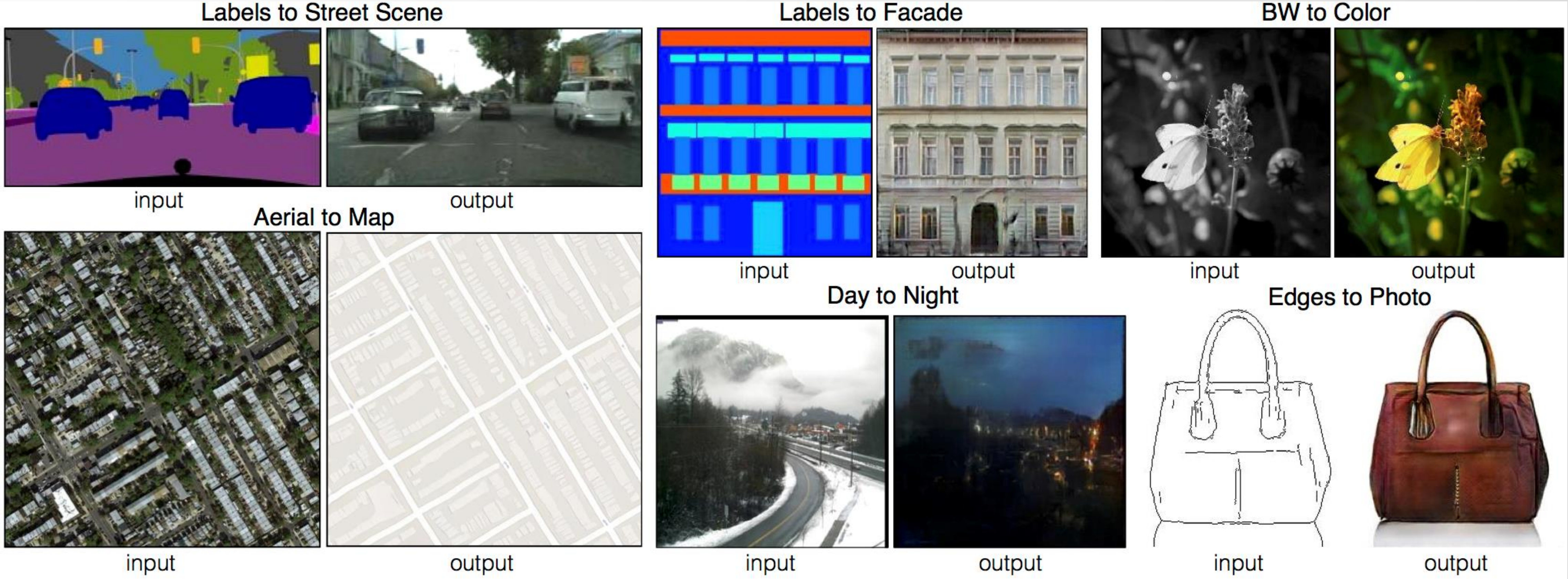
Synthesis from segmentation

Pix2Pix – architecture

- The architectures employed for the generator and discriminator closely follow DCGAN:
  - an U-Net as generator with batch normalization, LeakyReLU and skip connections;
  - a PatchGAN as discriminator that only penalizes structure at the scale of patches.

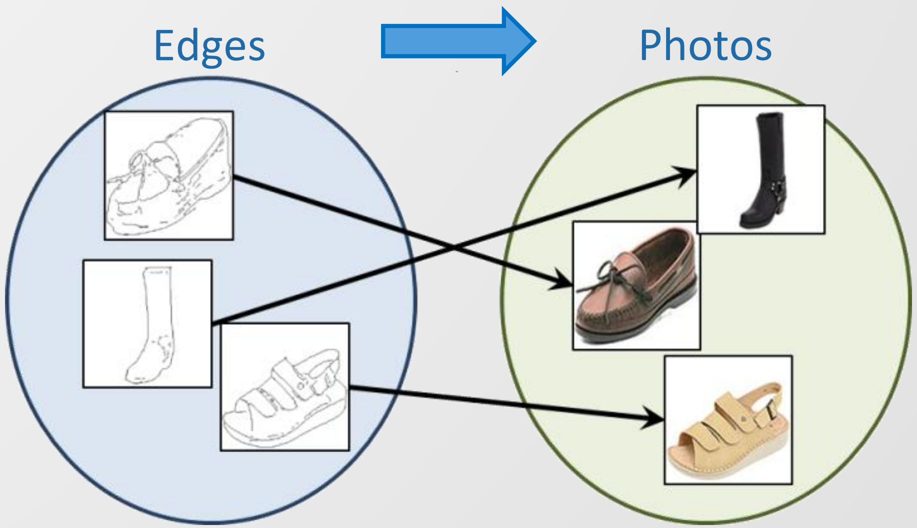


Pix2Pix – results



Paired vs unpaired image-to-image translation

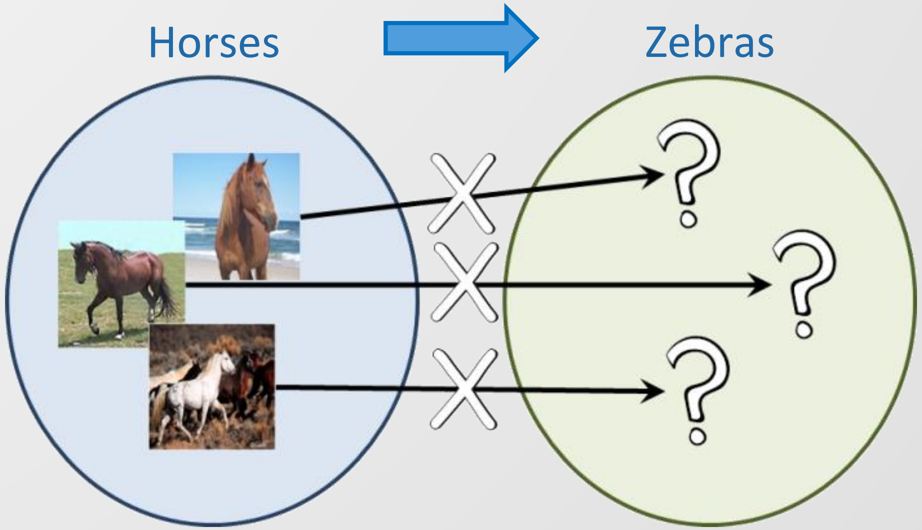
- Training an image-to-image translation model typically requires a large dataset of paired examples of source and target domain images.
- For example, we can get edge images from photos (e.g., applying an edge detector), to solve the more challenging problem of reconstructing photo images from edge images.



Paired vs unpaired image-to-image translation (2)

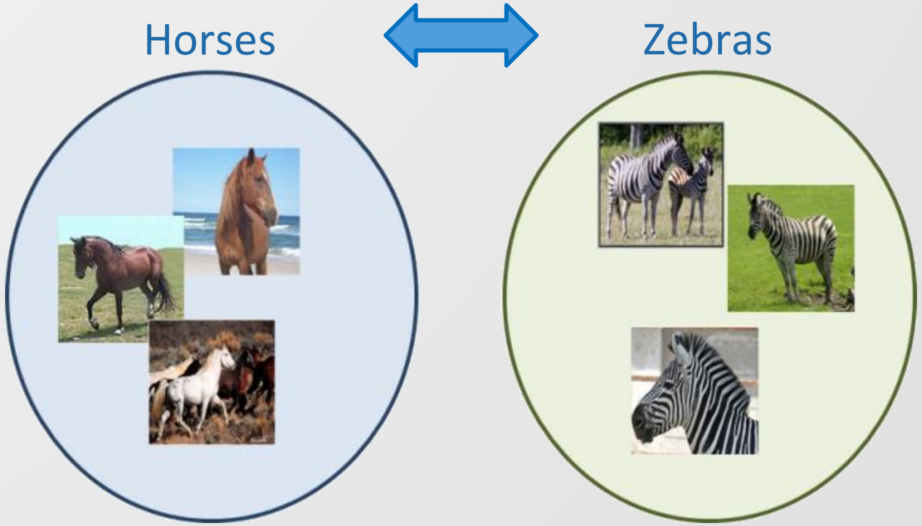
- The requirement for a paired training dataset is a limitation. These datasets are challenging and expensive to prepare: for instance, photos of different scenes under different conditions.

- In many cases, the datasets simply do not exist, such as famous paintings and their respective photographs.



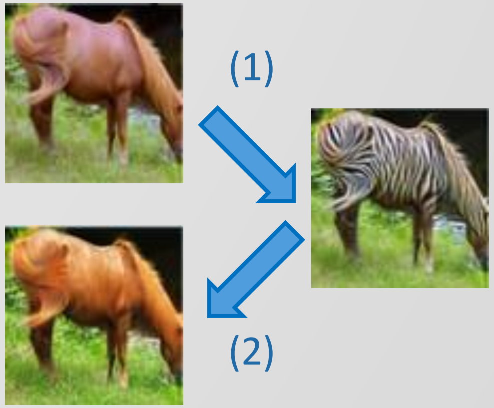
Paired vs unpaired image-to-image translation (3)

- Is it possible to train an image-to-image translation system without a paired dataset?
- In other word, can general characteristics be extracted from two collections of unrelated images and used in the image translation process?
- For example, take two collections of horse and zebra photos with unrelated scenes and locations and translate specific photos from one group to the other.
- This is called the problem of unpaired image-to-image translation.



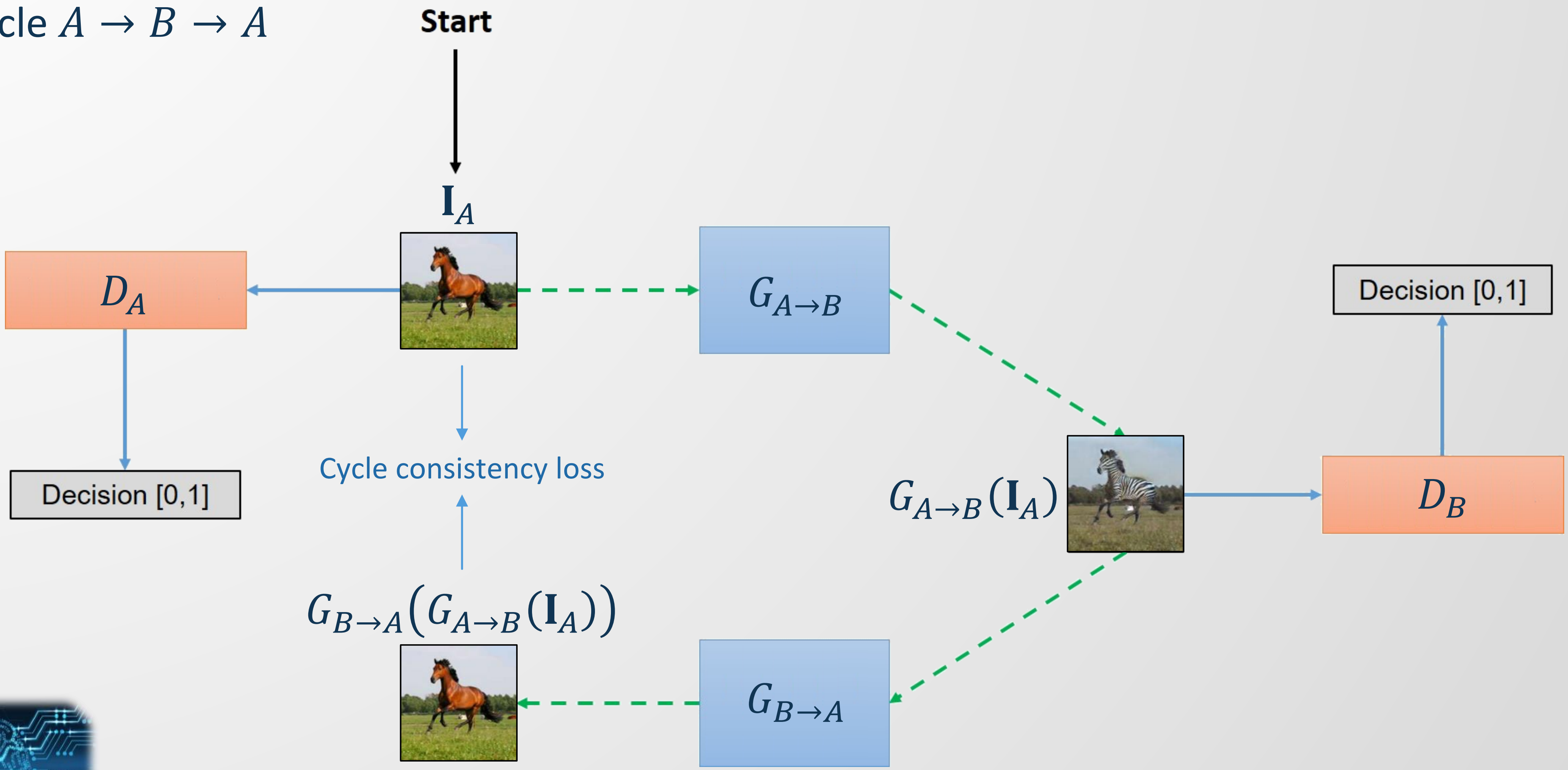
CycleGANs

- In 2017, J. Zhu, T. Park, P. Isola, and A. A. Efros proposed the Cycle Consistent Adversarial Network (CycleGAN) model to perform image-to-image translation without paired examples.
- Since in unpaired dataset there is no predefined transformation that can be learned by the generator, the idea behind CycleGANs is to create such transformation.
- To ensure a meaningful relation between input and generated images, the generator is enforced to preserve those features useful to map a generated image back to the input image by making a two-step transformation (forming a cycle):
  1. the input image is mapped from source (A) to target (B) domain;
  2. then the obtained image is transformed back from target (B) to the source (A) domain.



CycleGANs – architecture

Cycle $A \rightarrow B \rightarrow A$



Start

$\mathbf{I}_A$

D_A

$G_{A \to B}$

Decision [0,1]

Decision [0,1]

Cycle consistency loss

$G_{A \to B}(\mathbf{I}_A)$

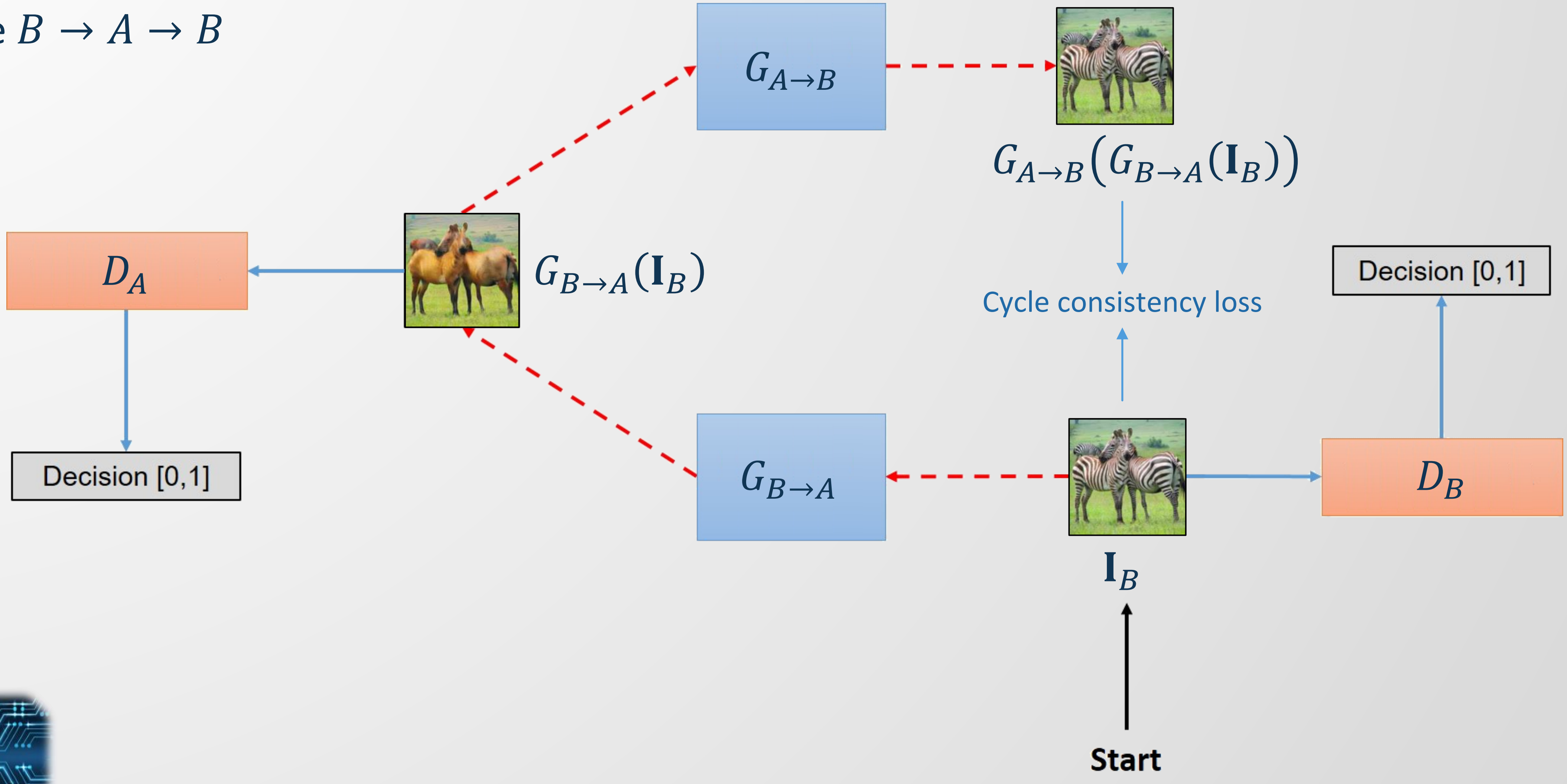
D_B

$G_{B \to A}\left(G_{A \to B}(\mathbf{I}_A)\right)$

$G_{B \to A}$

CycleGANs – architecture (2)

Cycle $B \rightarrow A \rightarrow B$



$G_{A \rightarrow B}$

$G_{A \rightarrow B}\left(G_{B \rightarrow A}(\mathbf{I}_B)\right)$

D_A

$G_{B \rightarrow A}(\mathbf{I}_B)$

Cycle consistency loss

Decision [0,1]

Decision [0,1]

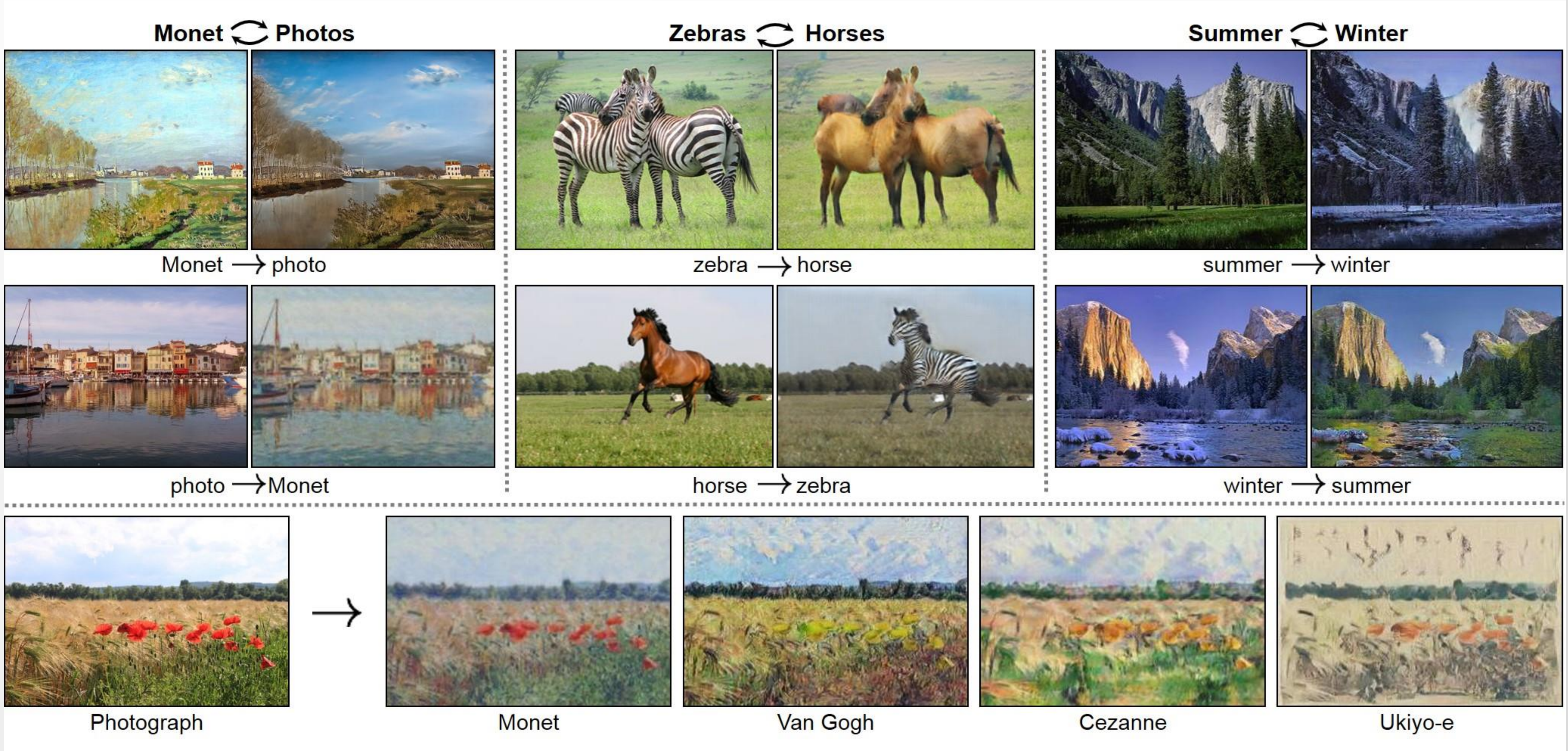
$G_{B \rightarrow A}$

D_B

$\mathbf{I}_B$

Start

CycleGANs – results

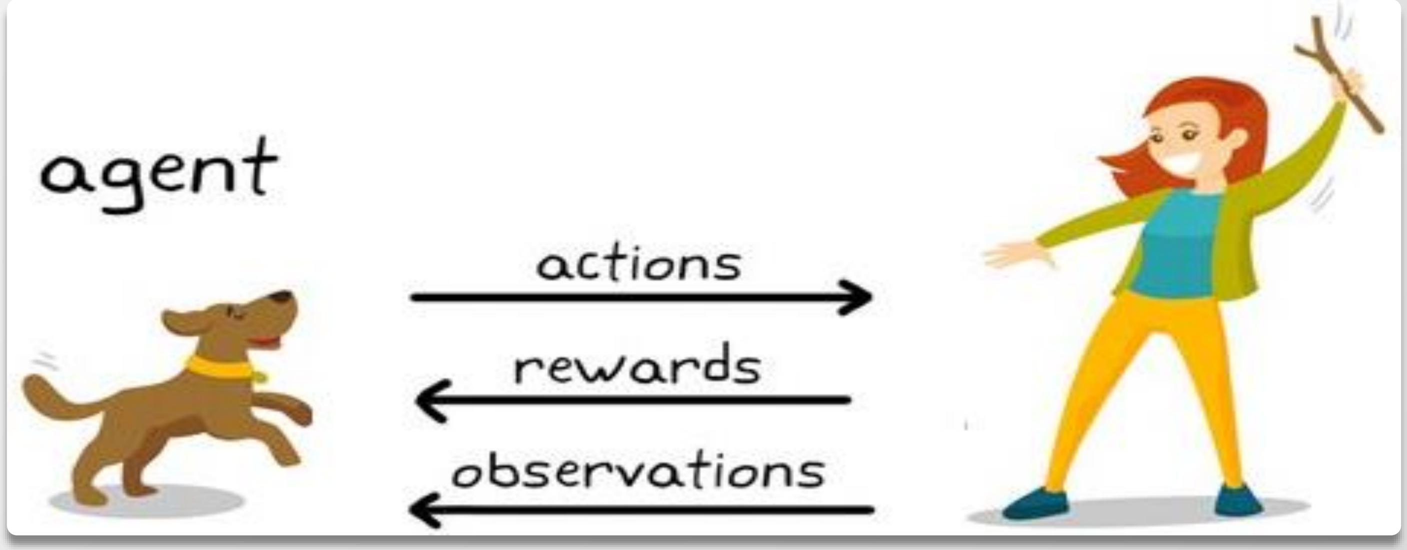


Recent generative models

BigGAN

VQ-VAE

Reinforcement learning

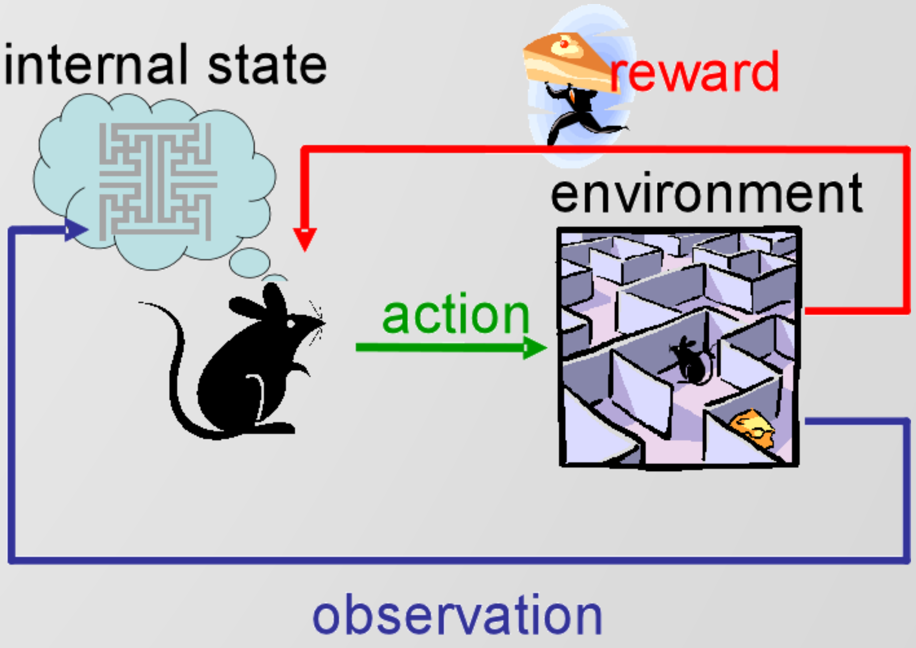


> “*Reinforcement learning is the science of decision making. It is about learning the optimal behavior in an environment to obtain maximum reward.*”

Source

What is reinforcement learning?

- Reinforcement Learning (RL) is a general framework in which an agent learns to behave in an environment by performing the actions and seeing the results of actions.
- For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty.
- The agent learns with the process of hit and trial, and based on the experience, it learns to perform the task in a better way using feedbacks without any labeled data.
- RL is an important model of how we (and all animals in general) learn. Praise from our parents, grades in school, salary at work – these are all examples of rewards.



Applications

Game playing

Adaptive control

Robotics

Finance sector

Chemistry

Business strategy

Manufacturing

Education

Healthcare

Terms used in reinforcement learning

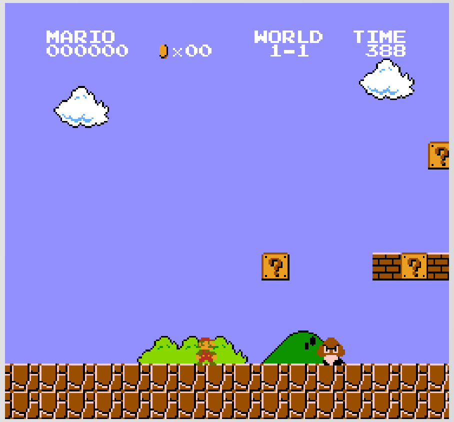
- Agent: an entity that can perceive/explore the environment and act upon it.
- Environment: where the agent learns and decides what actions to perform.
  - Anything that the agent cannot change arbitrarily is considered to be part of the environment.
  - Usually in RL, the environment is stochastic, which means the next state may be somewhat random.
- Action: actions are the moves taken by an agent within the environment.

Terms used in reinforcement learning (2)

- State: state is a situation returned by the environment after each action taken by the state agent.
  - This is how the environment changes in response to the agent's action.
  - If only a partial description of the state is available to the agent, it is called observation.

State

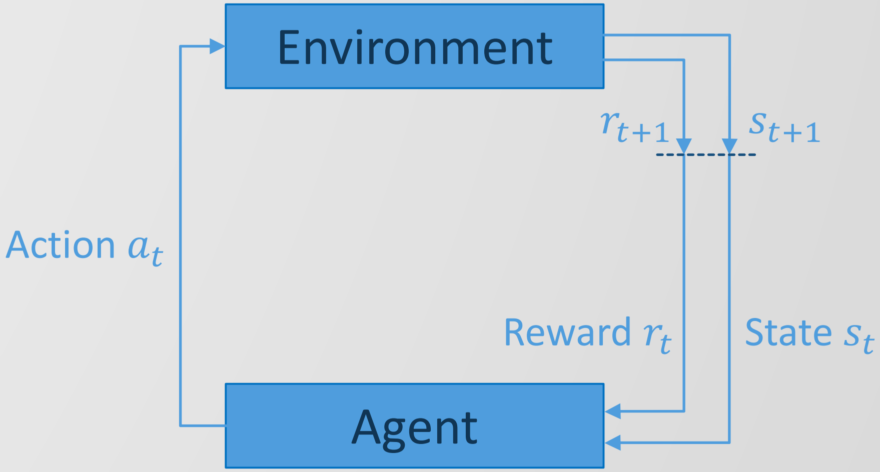
Observation



- Reward: a scalar feedback returned to the agent from the environment when it performs specific actions.
- Policy: policy is a strategy applied by the agent for the next action based on the current state. It defines the agent behavior at a given time by mapping state to action.

Reinforcement learning process

- At each time step t
  - The agent:
    1. analyzes current environment state s_t;
    2. out of possible actions it chooses and executes action a_t.
  - The environment:
    3. receives action a_t;
    4. emits new state s_{t+1};
    5. return scalar reward r_{t+1}.
  - The agent:
    6. updates its knowledge with the reward r_{t+1} given by the environment.
- The goal of the agent is to maximize the reward in the long run.

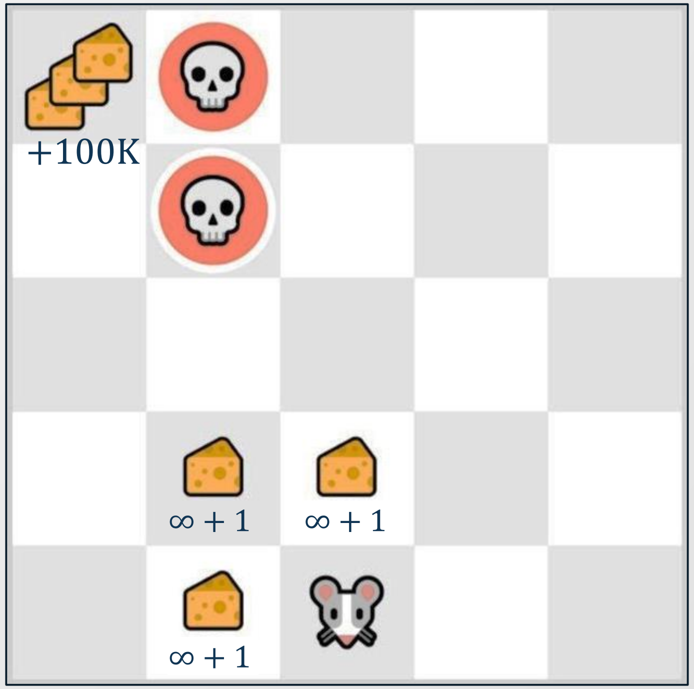


Reinforcement learning – example

- RL can be easily explained using the game of PacMan as example.
  - The goal of PacMan (the agent) is to eat the food in the grid while avoiding the ghosts on its way.
  - The grid is the interactive environment for PacMan.
  - PacMan can make four different actions: up, down, left and right.
  - PacMan receives:
    - rewards for eating food;
    - punishments if it gets killed by a ghost (loses the game).
  - The state is represented by the locations of PacMan, ghosts and food in the grid world.
  - The total cumulative reward is PacMan winning the game.

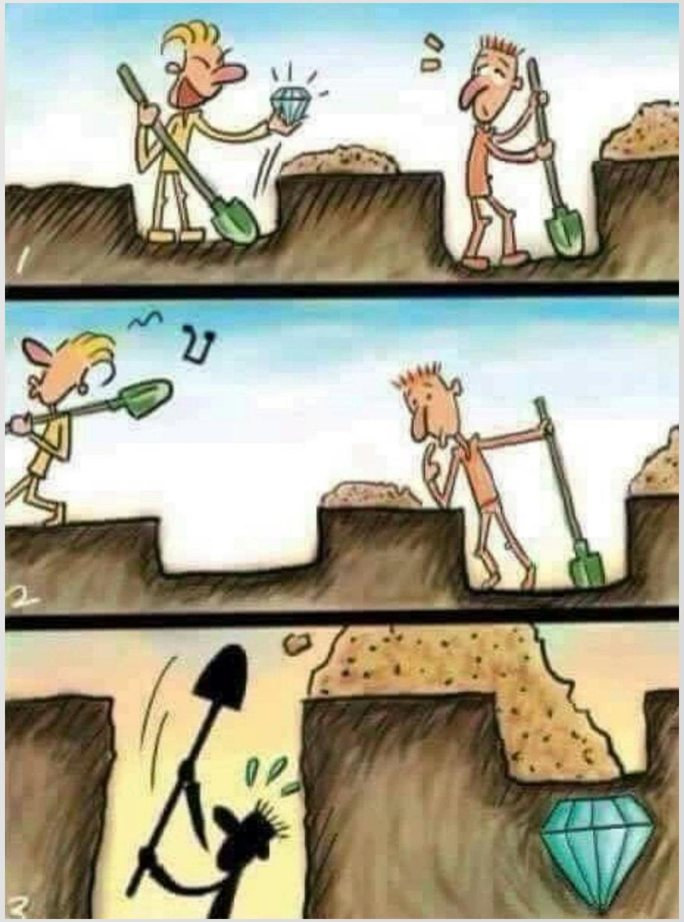
Exploration vs exploitation

- An agent needs to explore the environment in order to assess its reward structure. After some exploration, the agent might have found a set of apparently rewarding actions.
- How can the agent be sure that the found actions are actually the best?
- When should an agent continue to explore or else, when should it just exploit its existing knowledge?



Exploration vs exploitation (2)

- In order to build an optimal policy, the agent faces the dilemma of exploring new states while maximizing its reward at the same time.
- This is called *exploration vs exploitation dilemma*:
  - exploration means exploring and capturing more information about the environment in the hope of finding better actions;
  - exploitation involves using the already known information to maximize the rewards.



> “*There are only* 10^{15} *total hairs on all the human heads in the world,* 10^{23} *grains of sand on Earth, and about* 10^{81} *atoms in the known, observable universe. The number of chess games (estimated around* 10^{100K}*) is many times as great as all those numbers multiplied together — an impressive feat for 32 wooden pieces lined up on a board.*”

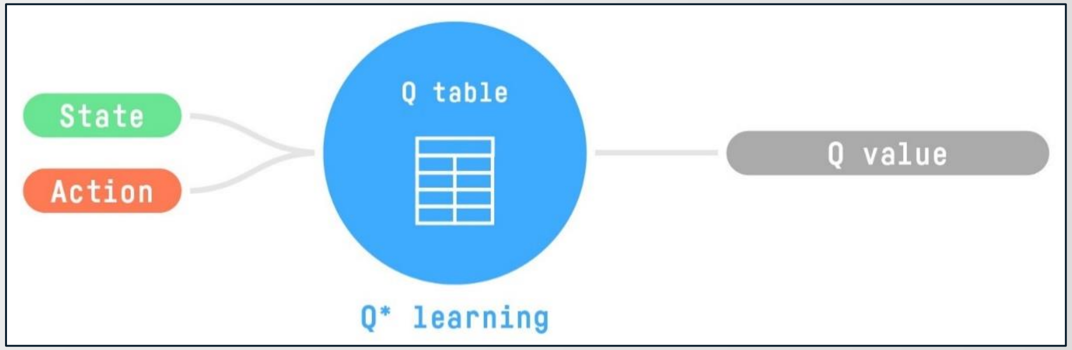
Source

Deep reinforcement learning

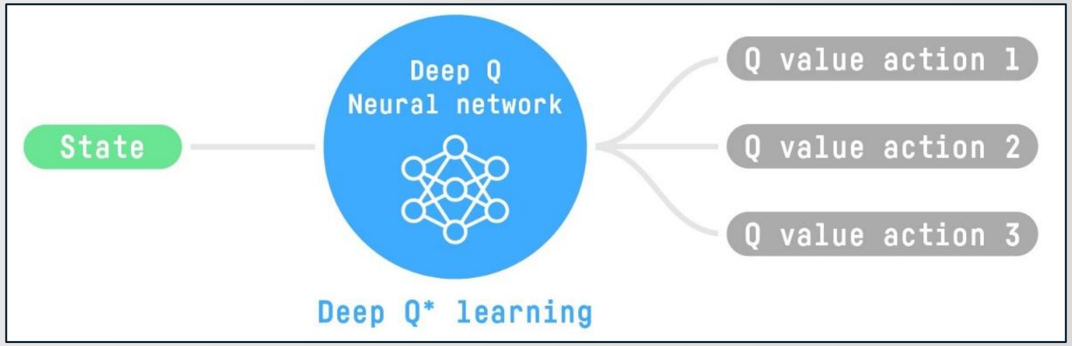
- In many practical decision-making problems, the states s are high-dimensional (e.g., images from a camera or raw sensor stream from a robot) and cannot be solved by traditional RL algorithms.
- Moreover, the amount of time required to explore each state to create the required Q-table would be unrealistic.
- Deep RL combines deep neural networks and RL to solve such problems, representing the policy π or other learned functions as a deep neural network.
- Deep RL algorithms can take in very large inputs (e.g., every pixel rendered to the screen in a video game) and decide what actions to perform to optimize an objective (e.g., maximizing the game score) without manual engineering the state space.

Deep Q-learning

- One of the fundamental problems involving the use of Q-learning is that the amount of memory required to store data rapidly expands as the number of states increases.

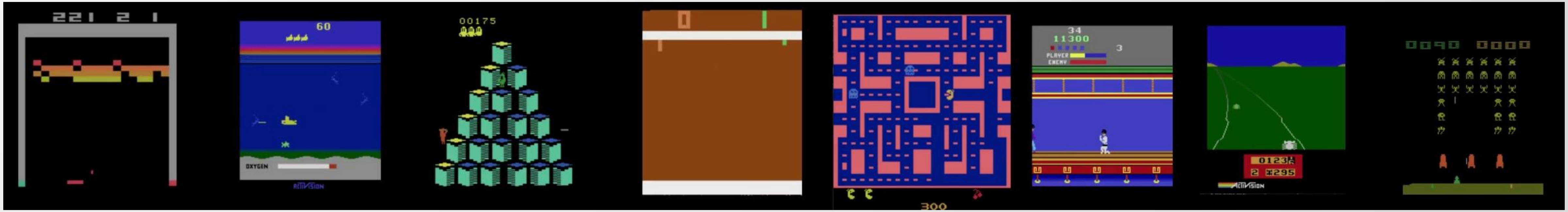


- With deep Q-learning, the Q-values are estimated with neural networks. The neural network takes the state as input, and outputs Q-values for all different actions the agent might take.



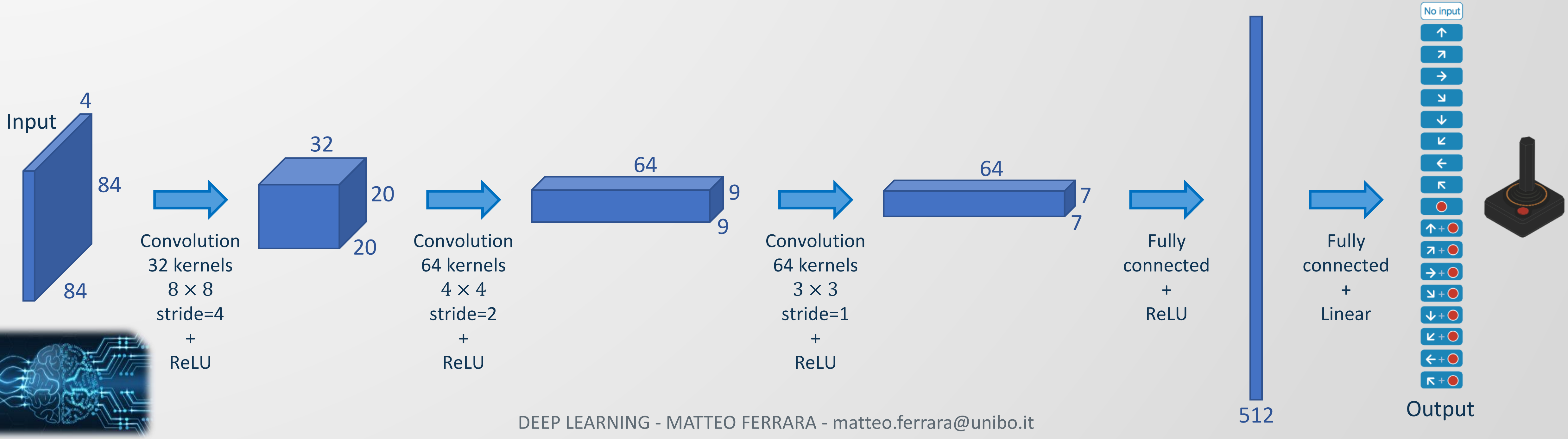
Deep Q-network

- In 2013, a small company, called DeepMind (immediately bought by Google), developed Deep Q-Network (DQN).
- DQN learned to play Atari video games by observing just the screen pixels and receiving a reward when the game score increased.
- DQN has been trained on 49 different Atari games using the same algorithm, architecture and hyper-parameters and it reached human-level performance on 29 of them.



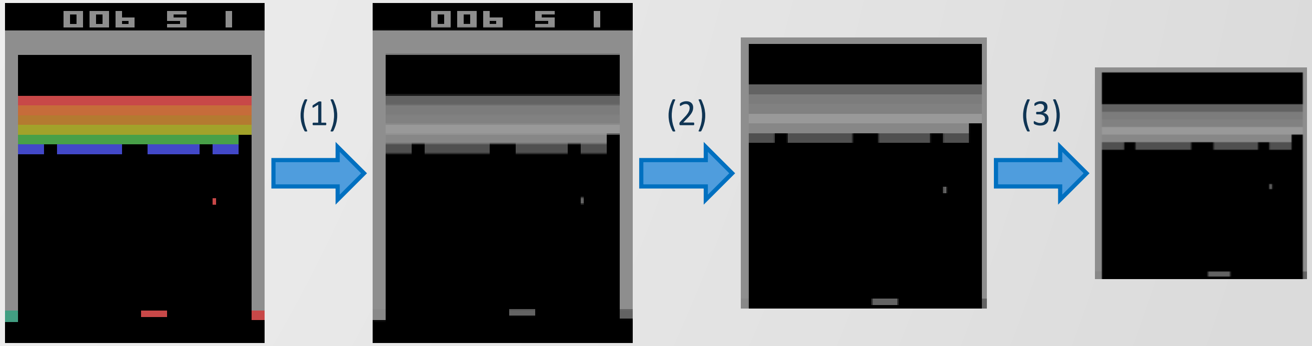
Deep Q-network – architecture

- The DQN consists of:
  - three convolutional layers;
  - two fully-connected layers.
- Note that, there are no pooling layers because they introduce translation invariance, and the network would become insensitive to the location of an object in the image.



Deep Q-network – input

- DQN takes the image of the screen as input state. To reduce the state complexity, and consequently the computation time, each frame is:
  1. transformed in grayscale;
  2. cropped to select the region of interest;
  3. resized to 84×84.



- To solve the problem of temporal limitation and give the network the sense of motion, DQN takes a stack of four frames as input.

Deep Q-network – training algorithm

```
initialize network $Q$ with random weights
for $m$ episodes
    $t = 0$
    repeat
        with probability $\epsilon$ select a random action $a_t$ otherwise select $a_t = \text{argmax}_a Q(s_t, a)$
        execute action $a_t$ and observe reward $r_{t+1}$ and new state $s_{t+1}$
        estimate the target value $y_t = r_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a)$
        perform a gradient descent step to update $Q$ weights by minimizing $l = \left(y_t - Q(s_t, a_t)\right)^2$
        $t = t + 1$
    until terminated
end for
```

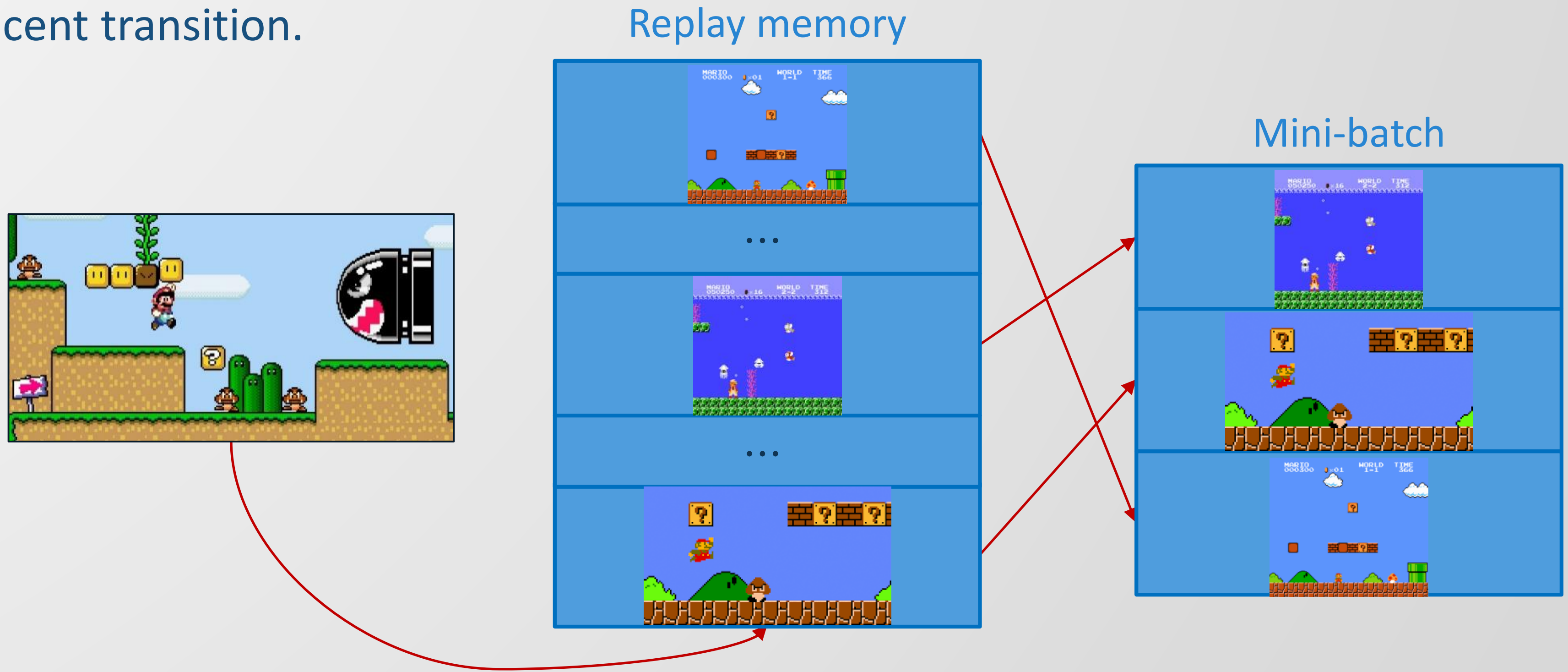
Deep Q-network – experience replay

- There is an issue when using neural network as Q approximator: the transitions are very correlated since they are all extracted from the same episode reducing the overall variance.
- As a result, the network tends to forget the previous transitions as it overwrites them with new ones resulting in a network overfitted on the current episode.

For instance, if we are in the first level and then in the second (which is totally different), the Mario agent can forget how to behave in the first level.

Deep Q-network – experience replay (2)

- To remove correlations and make the DQN training more stable, the *experience replay* technique can be used:
  - during training, all transitions are stored in a replay memory;
  - when updating the network, mini-batches are randomly sampled from the replay memory and used instead of the most recent transition.



Replay memory

…

…

Mini-batch

Deep Q-network – examples

[Source]

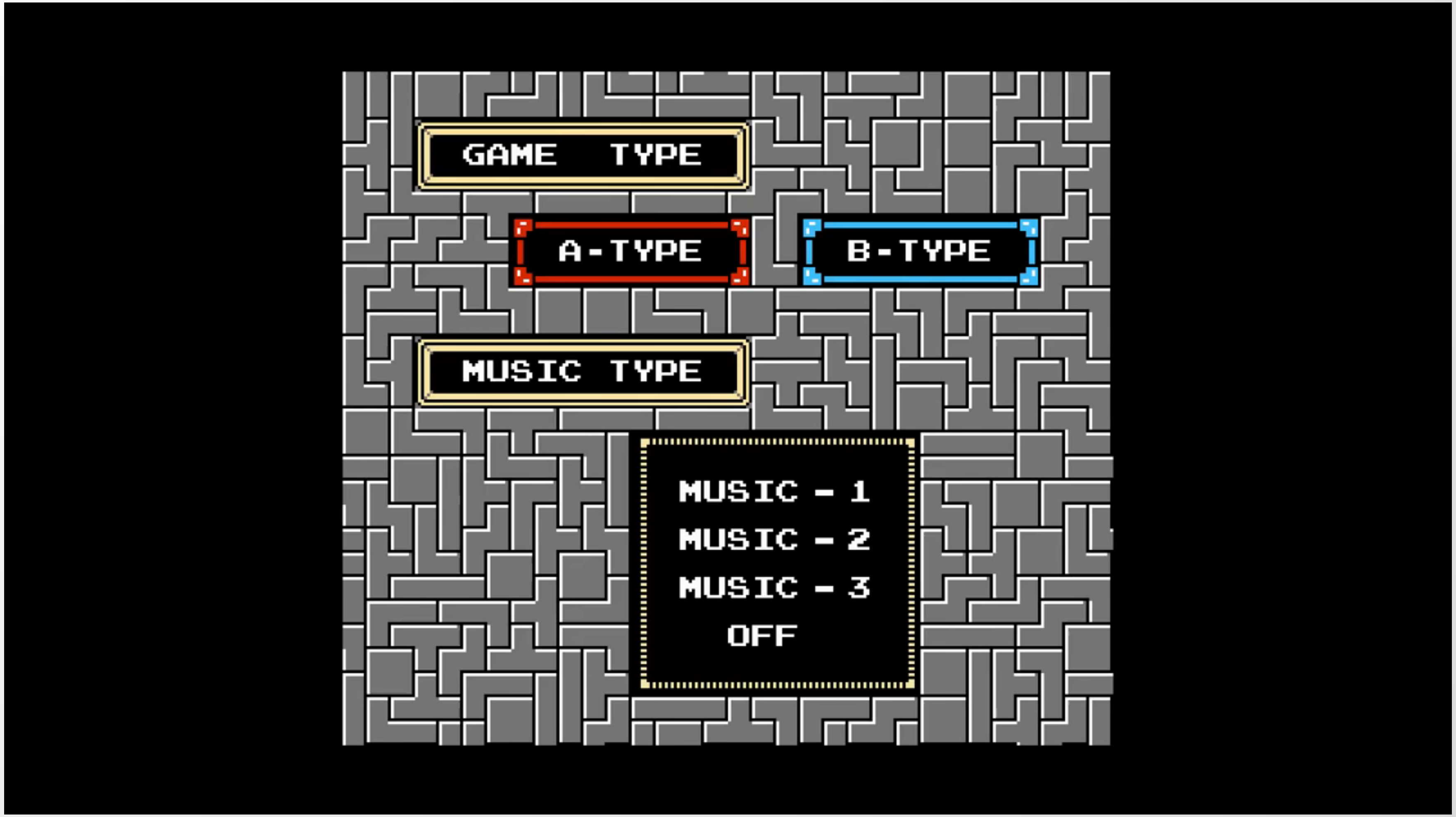
Deep Q-network – examples (2)

Google Deepmind DQN playing
Atari Pacman

Setup:
NVIDIA GTX 690
i7-3770K - 16 GB RAM
Ubuntu 16.04 LTS
Google Deepmind DQN

Source

Deep Q-network – examples (3)



Source

Suggested readings

- F. Chollet, *"Deep Learning with Python (2nd edition)"*, Manning Pubblications Co., USA, 2021.
- A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *"Dive into Deep Learning"*, 2020.
- M. Elgendy, *"Deep Learning for Vision Systems"*, Manning Publications Co., USA, 2020.
- A. Geron, *“Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems”*, O'Reilly Media, Inc., USA, 2019.
- M. Nielsen, *"Neural Networks and Deep Learning"*, 2019.
- I. Goodfellow, Y. Bengio, and A. Courville, *"Deep Learning"*, MIT Press, 2016.