Everything You Always Wanted to Know About Deep Learning Frameworks *

(*But Were Afraid to Ask)

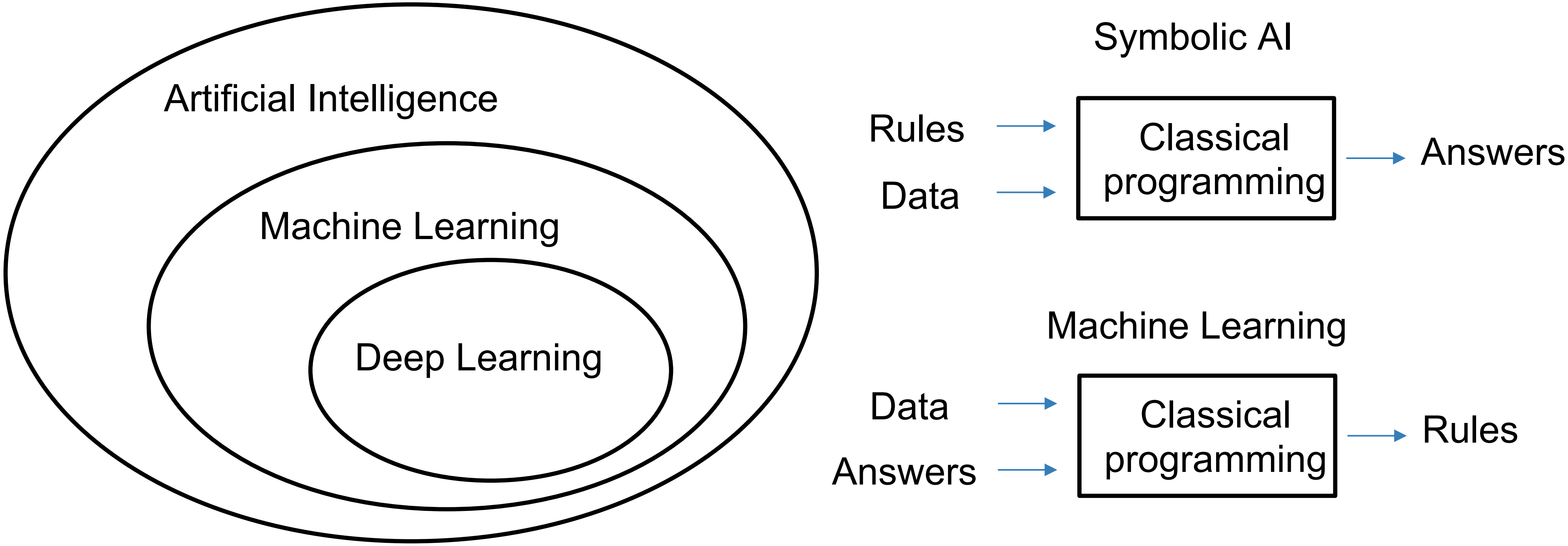
Konstantina Dritsa

PhD Candidate, Business Analytics Laboratory
Athens University of Economics & Business

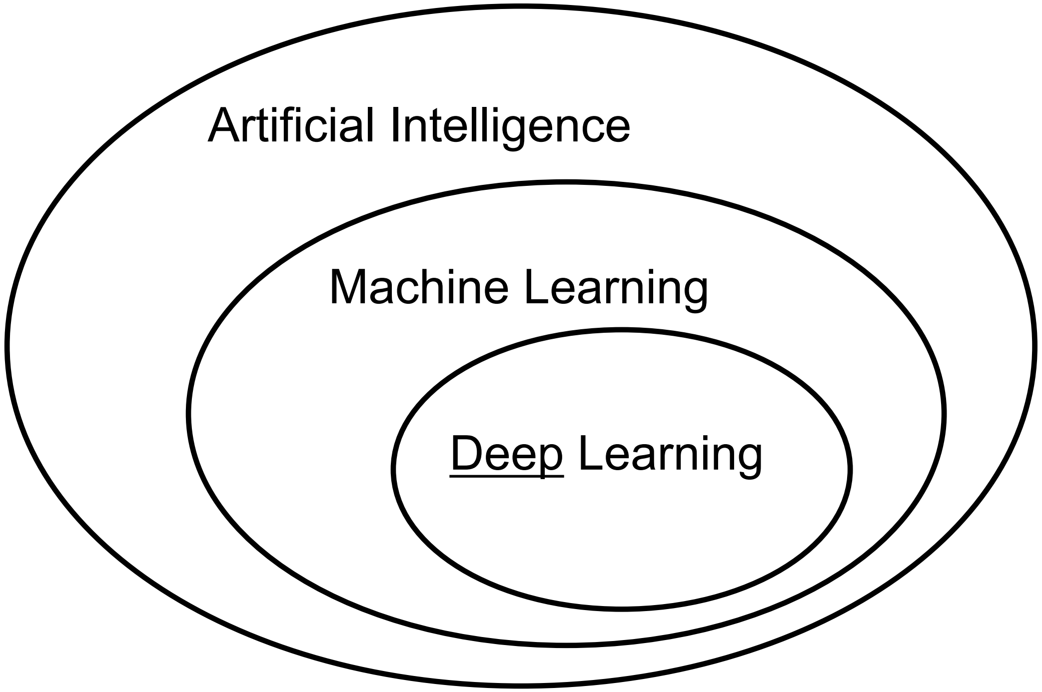
Roadmap

- Introduction to Deep Learning
- Theoretical introduction to Deep Learning Frameworks
- Toy implementation on Keras
- Practical comparison of deep learning implementations between Keras, Tensorflow, Pytorch

AI vs ML vs DL


Artificial Intelligence
Machine Learning
Deep Learning
Symbolic AI
Rules
Data
Classical programming
Answers
Machine Learning
Data
Answers
Classical programming
Rules

AI vs ML vs DL



Hierarchical Feature Abstraction

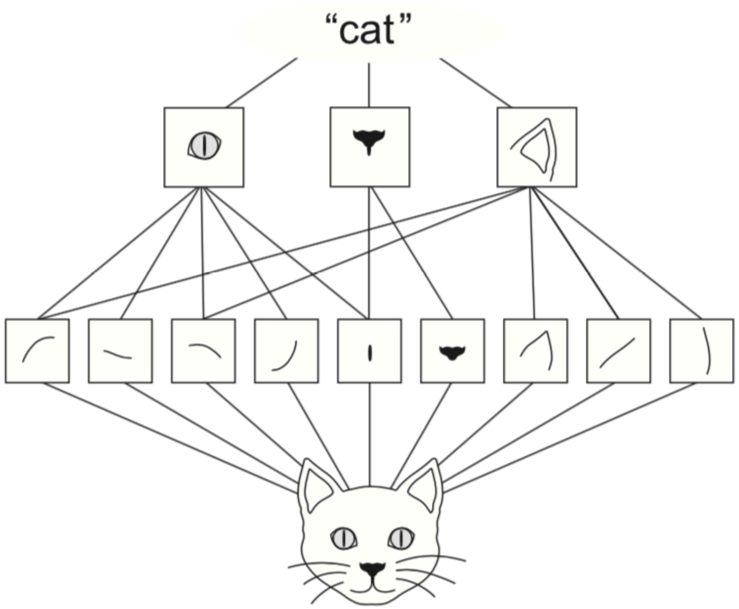
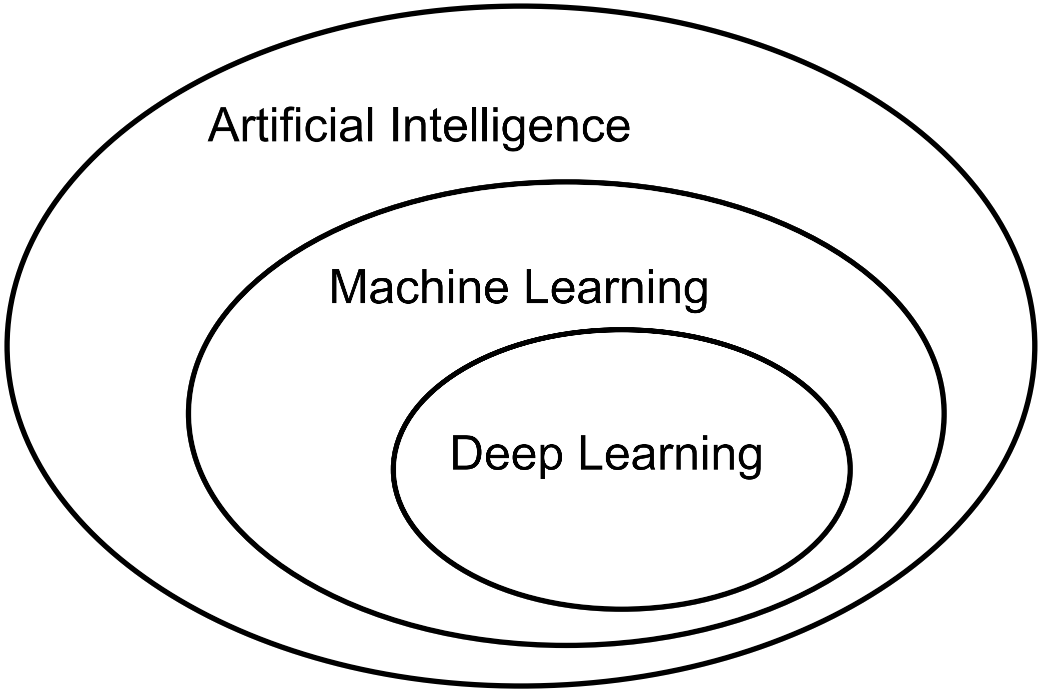
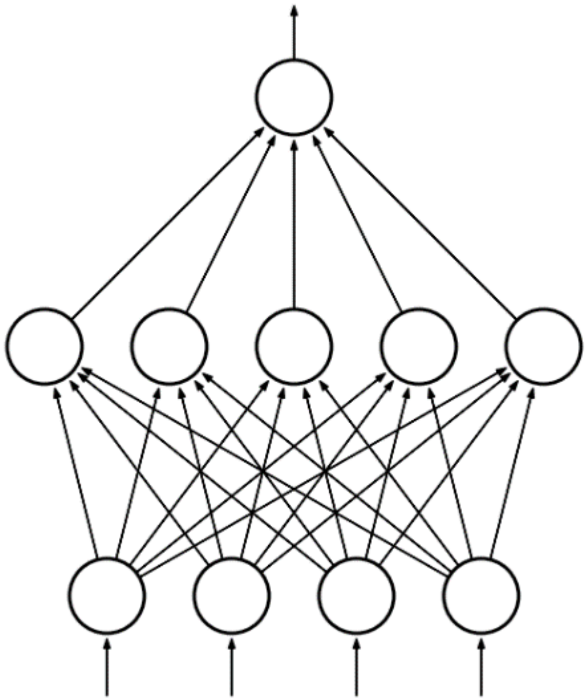


Photo from Francois Chollet's book "Deep Learning with Python"

AI vs ML vs DL



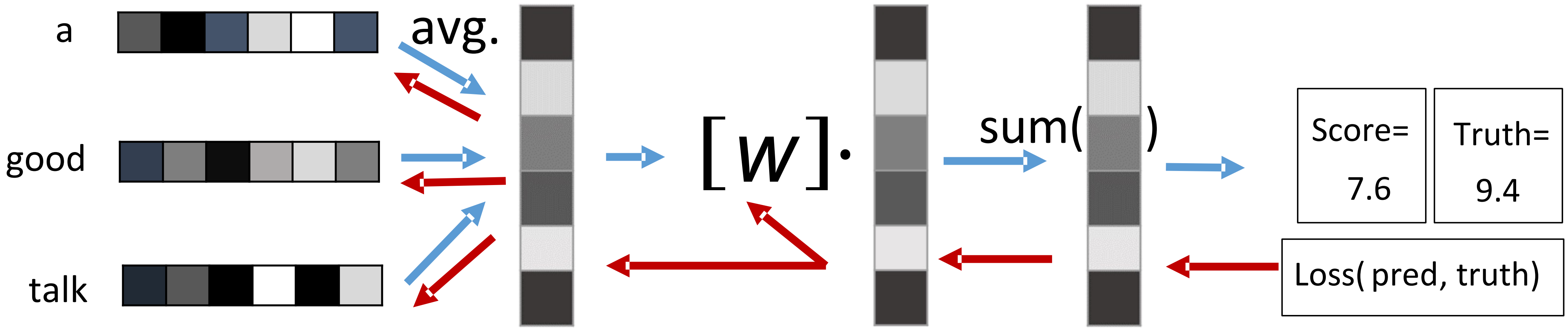
Hierarchical Feature Abstraction



Neural network: structured sequence of algebraic operations on vectors and matrices

Deep Learning

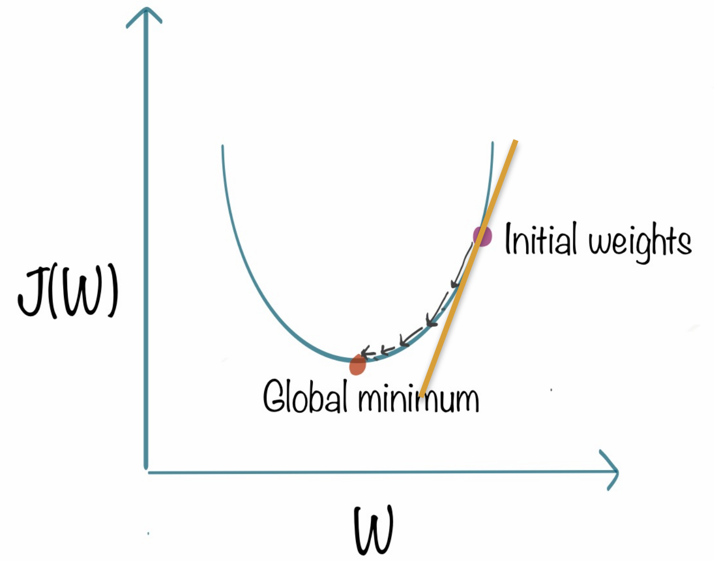
- A simple example : Predict how positive is a given sentence



$$vec \leftarrow vec - \alpha \frac{\partial loss}{\partial vec} \qquad w \leftarrow w - \alpha \frac{\partial loss}{\partial w}$$

Gradient Descent for Backpropagation

Optimization algorithm that minimizes a loss function, moving repeatedly to the steepest descent.



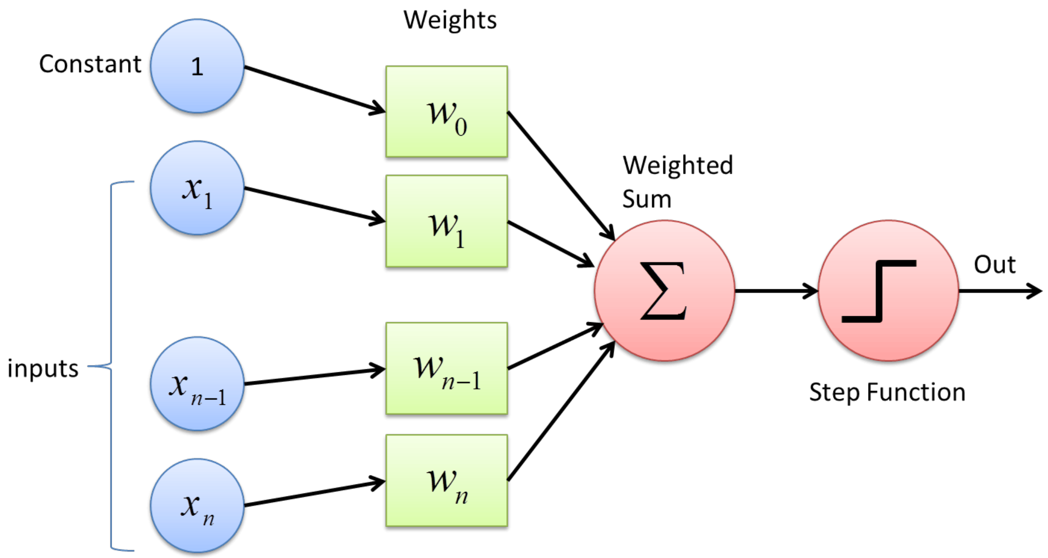
$$w \leftarrow w - \alpha \frac{\partial loss}{\partial w}$$

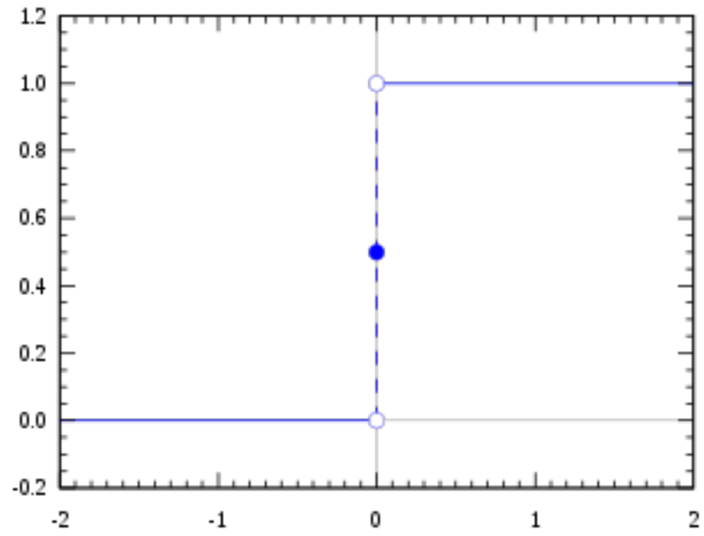
Perceptron

- With step function -> linear classifier (binary)
- Produces a single binary output (0 or 1)
- Divides the space with a straight line in two segments

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j + b \le 0 \\ 1 & \text{if } \sum_j w_j x_j + b > 0 \end{cases}$$

$$w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

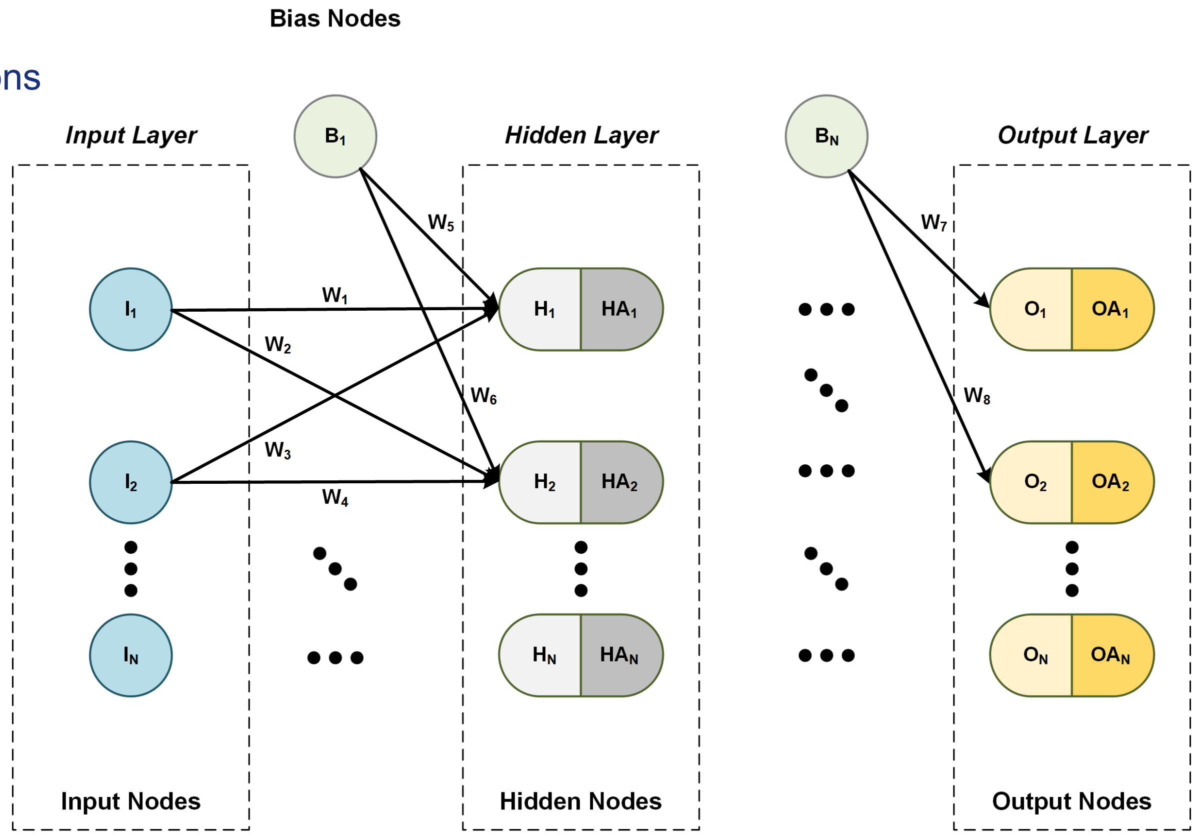




Deep Learning: Neural network anatomy

- Layers: collection of neurons
- Neurons: nodes of mathematical computations
- Connection: weighted relationship between nodes of subsequent layers
- Weights of the connections

- H1: hidden node
- HA1: value of H1 passed through the activation function
- Accordingly O1, OA1, B1…



Deep Learning: Hyperparameter tuning

- Size of neural network
- Loss function
- Activation function
- Optimizer
- Learning rate
- Epochs
- Batch size
- Dropout
- Architecture

Deep Learning: Hyperparameter tuning

- Model capacity: How many layers and nodes
- The more the parameters, the larger the memorization capacity

Rule-of-thumb methods to choose size:

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

- Size of neural network
- Loss function
- Activation function
- Optimizer
- Learning rate
- Epochs
- Batch size
- Dropout
- Architecture

The real challenge: generalization

"Travel is fatal to prejudice, bigotry, and narrow-mindedness"
- Mark Twain

Generalization refers to your model's ability to make valid predictions on new, previously unseen data.

The real challenge: generalization

Underfitting:

Too few neurons to learn complex representations in a complicated data set.

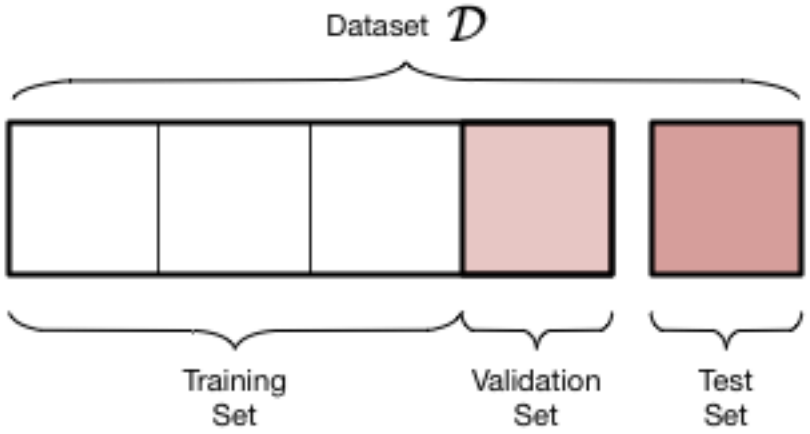
Overfitting:

Too many neurons or too many epochs that lead to the memorization of the training data.

Pareto split principal:

- 20% test set
- 80% for training and validation

(from which 80% for training and 20% for validation)



Deep Learning: Hyperparameter tuning

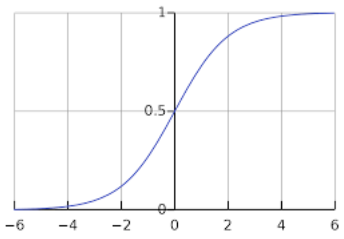
- Loss function: distance between prediction and ground truth
- Multiple loss functions.
- Choose the right loss function for the task:

Problem	Loss Functions
Regression	Mean Squared Error Loss, Mean Squared Logarithmic Error Loss, Mean Absolute Error Loss
Binary Classification	Binary Cross-Entropy, Hinge Loss, Squared Hinge Loss
Multi-Class Classification	Multi-Class Cross-Entropy Loss, Sparse Multiclass Cross-Entropy Loss, Kullback Leibler Divergence Loss

- Size of neural network
- Loss function
- Activation function
- Optimizer
- Learning rate
- Epochs
- Batch size
- Dropout
- Architecture

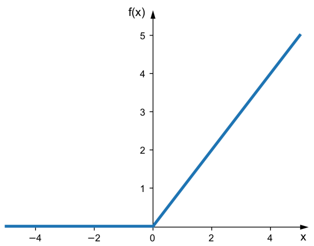
Deep Learning: Hyperparameter tuning

- **Sigmoid.** S-shaped curve that ranges between 0 and 1. Squashes arbitrary values into the [0,1] interval, something that can be interpreted as a probability.



- **Tanh:** S-shaped curve that ranges between -1 and 1.

- **Rectified Linear Unit (ReLU):** Zero for negative x values. More computationally effective.



- **Softmax:** outputs probabilities. Ideal for classification. The outputs should sum to 1.

- Size of neural network
- Loss function
- Activation function
- Optimizer
- Learning rate
- Epochs
- Batch size
- Dropout
- Architecture

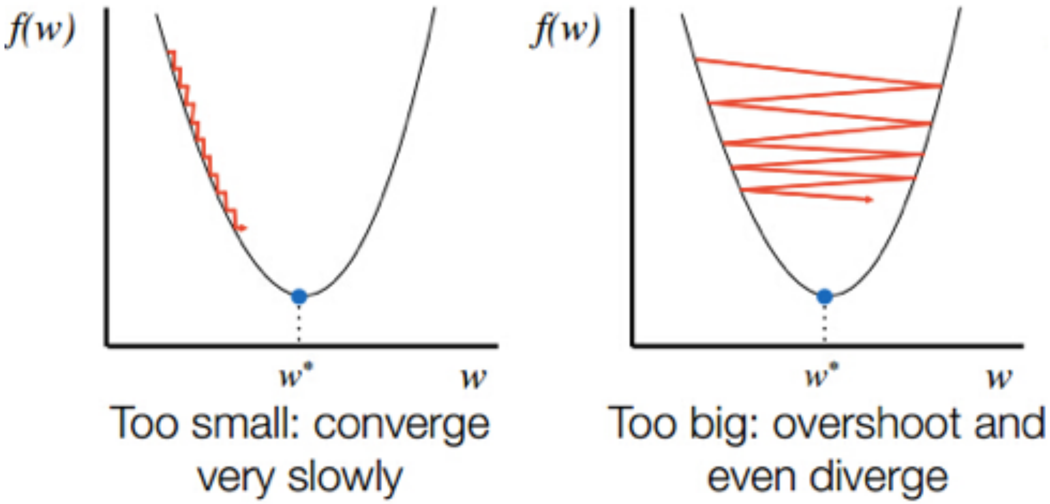
Deep Learning: Hyperparameter tuning

- Optimizer: Determines how the network will be updated based on the loss function.
- It implements a specific variant of stochastic gradient descent (SGD).

- Popular gradient descent optimization algorithms:
  - Adam — Adaptive Moment Estimation
  - AdaMax — variant of adam
  - RMSProp — Root Mean Square Propagation
  - Adagrad — Adaptive Gradient Algorithm
  - Adadelta — extension of Adagrad
  - Nesterov accelerated gradient
  - etc.

- Size of neural network
- Loss function
- Activation function
- Optimizer
- Learning rate
- Epochs
- Batch size
- Dropout
- Architecture

Deep Learning: Hyperparameter tuning



- Size of neural network
- Loss function
- Activation function
- Optimizer
- Learning rate
- Epochs
- Batch size
- Dropout
- Architecture

Deep Learning: Hyperparameter tuning

- Epoch: Each iteration over all the training data.
- Too many epochs -> Overfitting
- Take into consideration the loss in the validation dataset

- Size of neural network
- Loss function
- Activation function
- Optimizer
- Learning rate
- Epochs
- Batch size
- Dropout
- Architecture

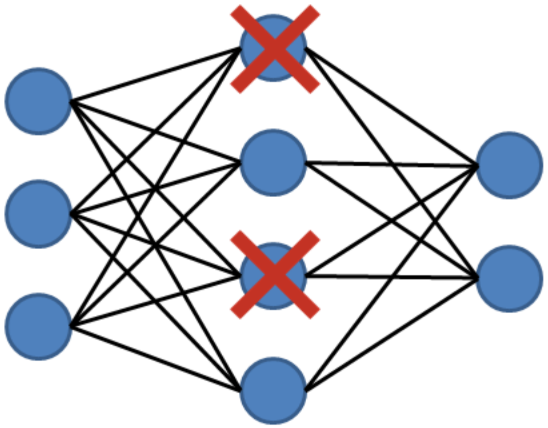
Deep Learning: Hyperparameter tuning

- Training step: Each epoch consists of training steps. A training step is each forward propagation & backpropagation (parameter update)
- **Batch Gradient Descent: All** the training data are presented to the model at once. Each epoch = one training step.
  Computationally inefficient for large datasets.
- **Stochastic Gradient Descent:** Only **one** random sample of the training data is presented to the model at each training step. Each epoch = many training steps.
- **Mini-batch Stochastic Gradient Descent:** At each training step, present to the model a **batch** of the data.
- Parameter updates are made once for each batch.

- Size of neural network
- Loss function
- Activation function
- Optimizer
- Learning rate
- Epochs
- Batch size
- Dropout
- Architecture

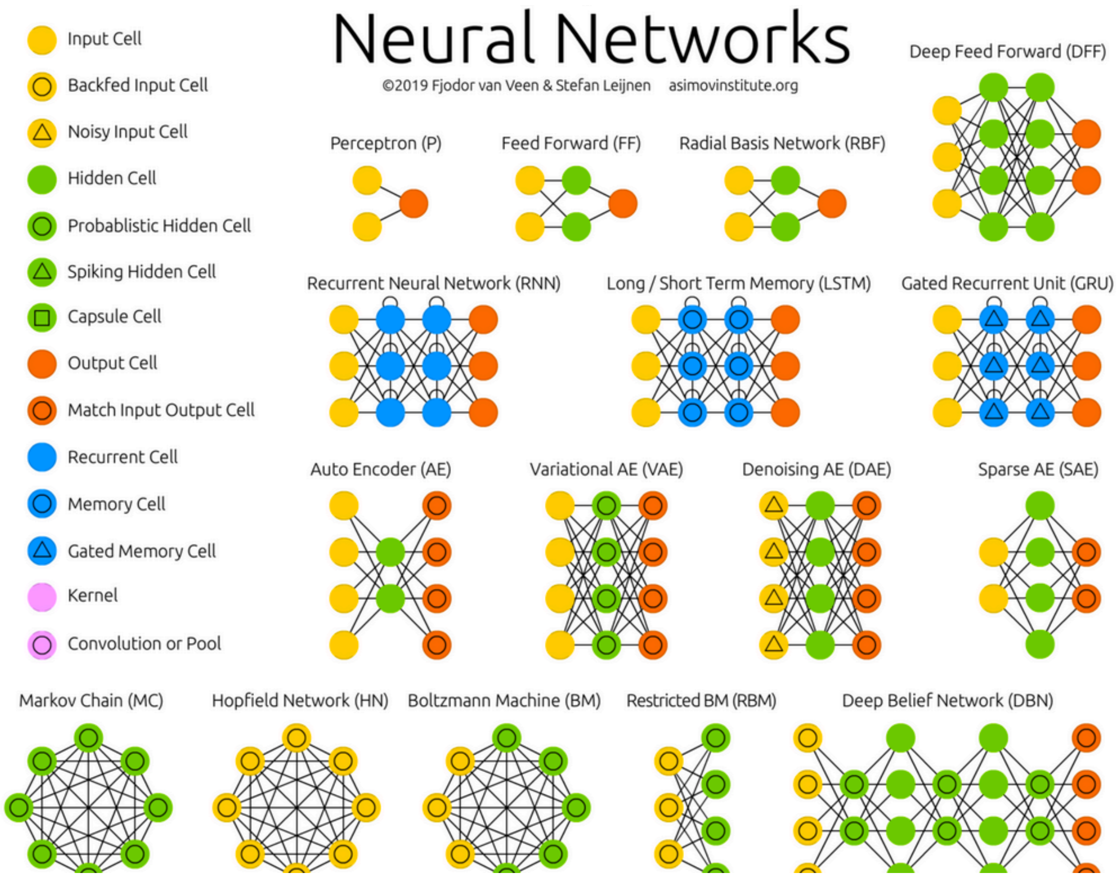
Deep Learning: Hyperparameter tuning

- Dropout: applied to a layer, it consists of randomly dropping out (setting to zero) a number of output features of the layer during training.
- It forces the neural net to “not rely” on any specific node, by making the training process noisy.
- Thus, it helps reduce overfitting.
- Usually set between 0.2 and 0.5.



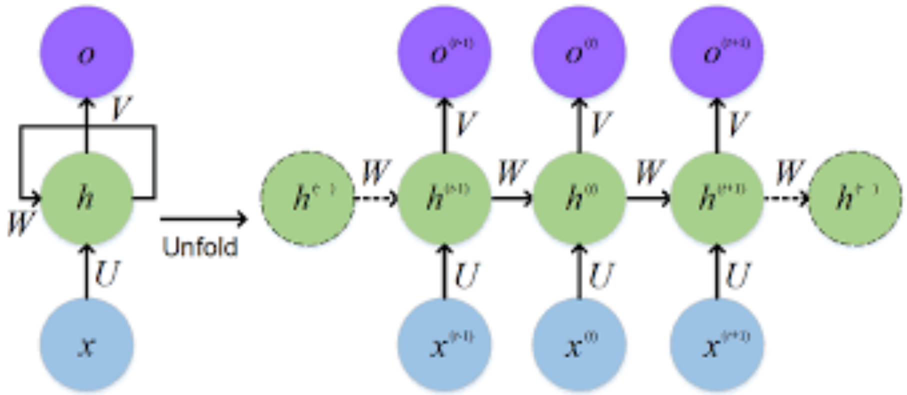
- Size of neural network
- Loss function
- Activation function
- Optimizer
- Learning rate
- Epochs
- Batch size
- Dropout
- Architecture

Deep Learning: Hyperparameter tuning

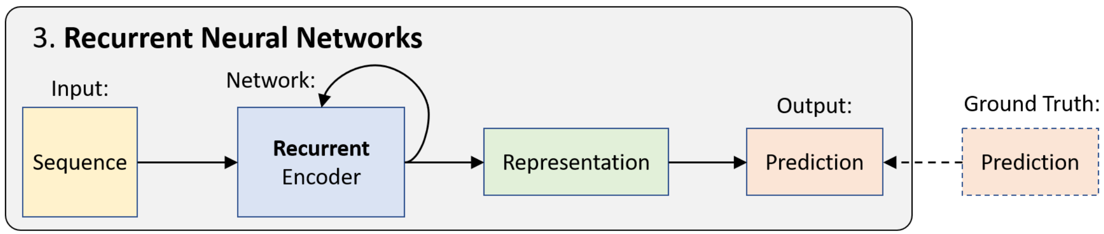


- Size of neural network
- Loss function
- Activation function
- Optimizer
- Learning rate
- Epochs
- Batch size
- Dropout
- Architecture

Recurrent neural networks (RNN)

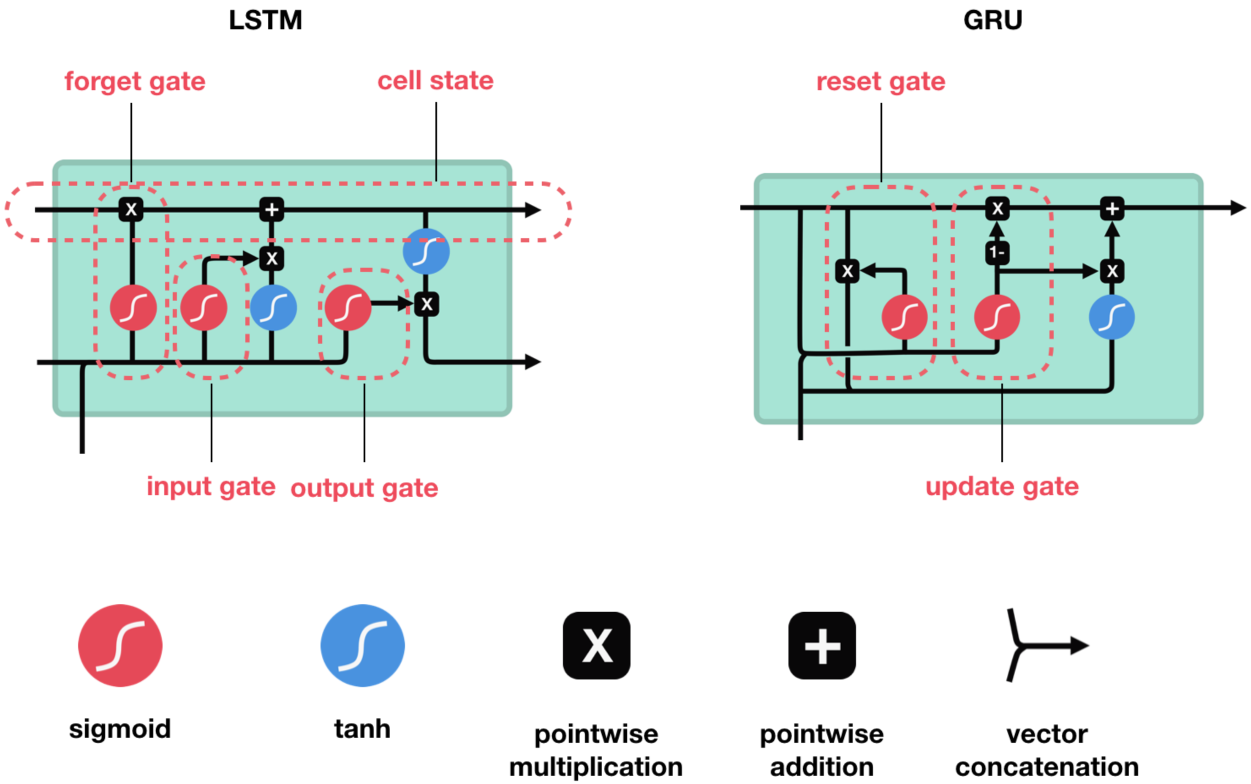


- When data order matters - sequential input
- Certain pathways are cycled
- Neurons are fed information:
  - from the previous layer and
  - from themselves from the previous pass
- Vanishing (or exploding) gradient problem
- Tasks: language modeling, speech recognition/generation etc.



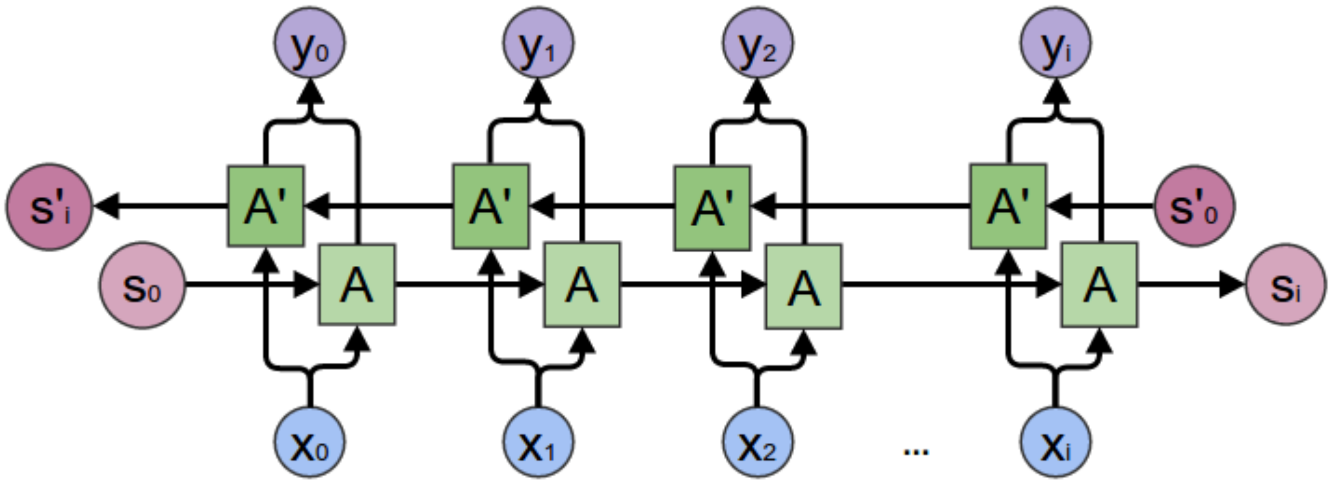
LSTM & GRU

- Solution to short-term memory
- Use a more complex recurrent unit
- Gates to control what information is passed through
- LSTM: Forget – Input from previous layers, Update cell state, Output part of cell state
- GRU: Reset (=forget), Update (=input+previous cell state)
- GRU is faster but less expressive

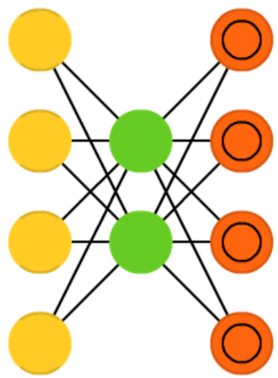


Bidirectional networks (BiRNN, BiLSTM and BiGRU)

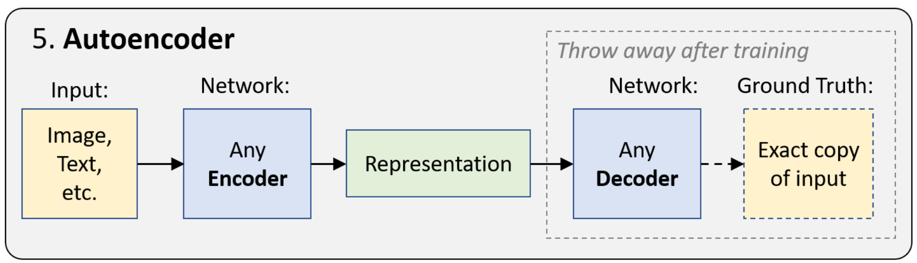
- Connected to the past, but also to the future
- Tasks: fill in gaps, fill in missing parts of images

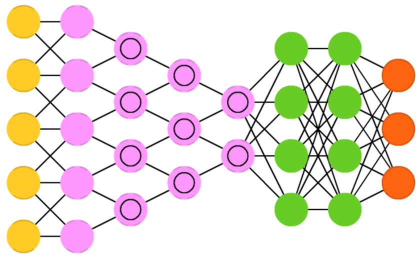


Encoder Decoder Architectures



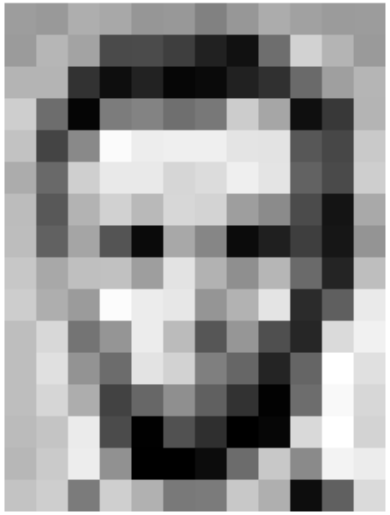
- Input = target,
- Learns efficient data representations (encoding
- Encode information (as in compress, not encrypt)
- Up to the middle: encoding part
- In the middle the information is most compressed
- From middle till the end: decoding part
- Dimensionality reduction and reconstruction
- Tasks: remove noise from audio, image, signal

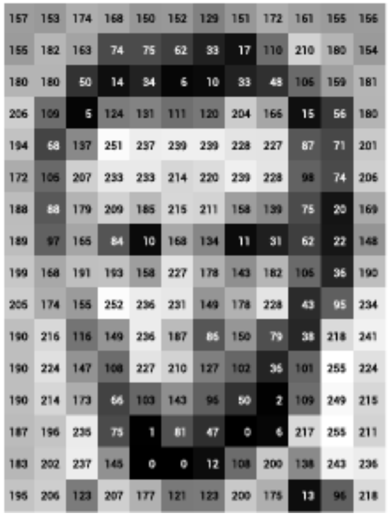




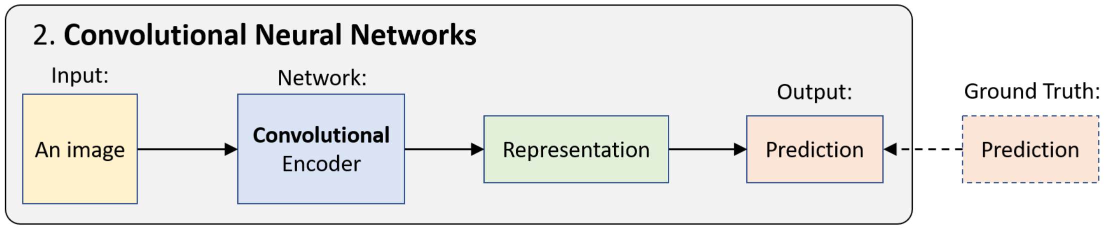
Convolutional neural networks (CNN)

- Tasks: image classification, object detection, video action recognition etc.

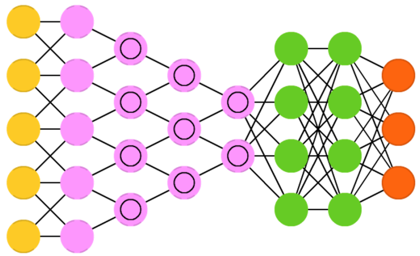


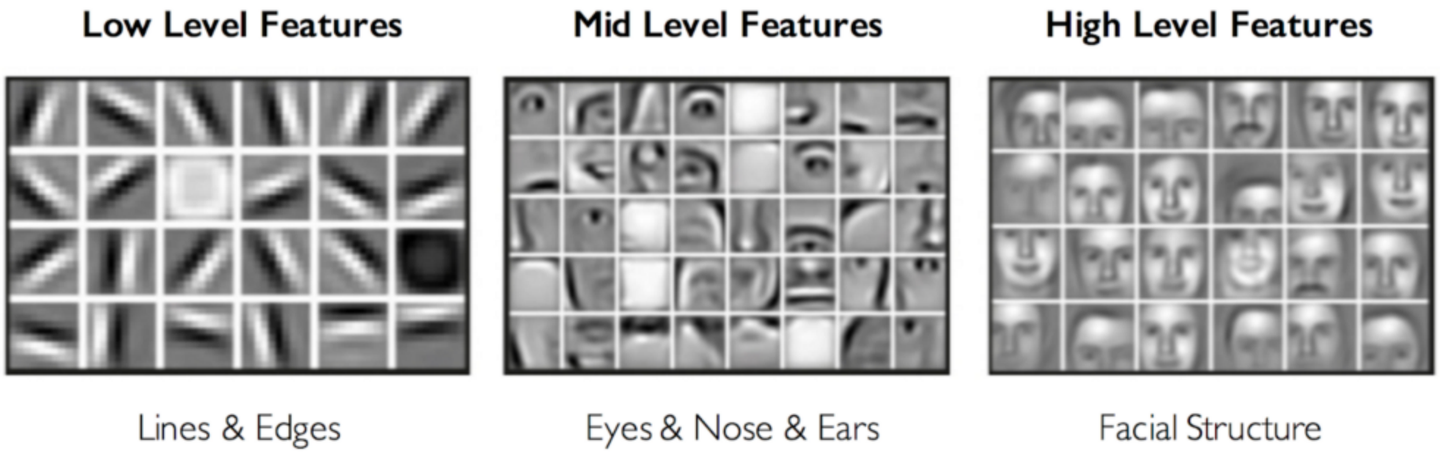


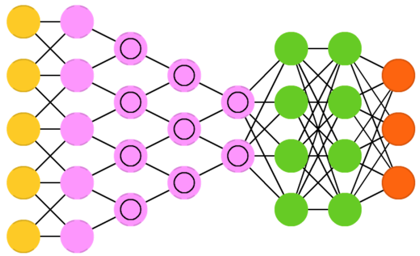
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218



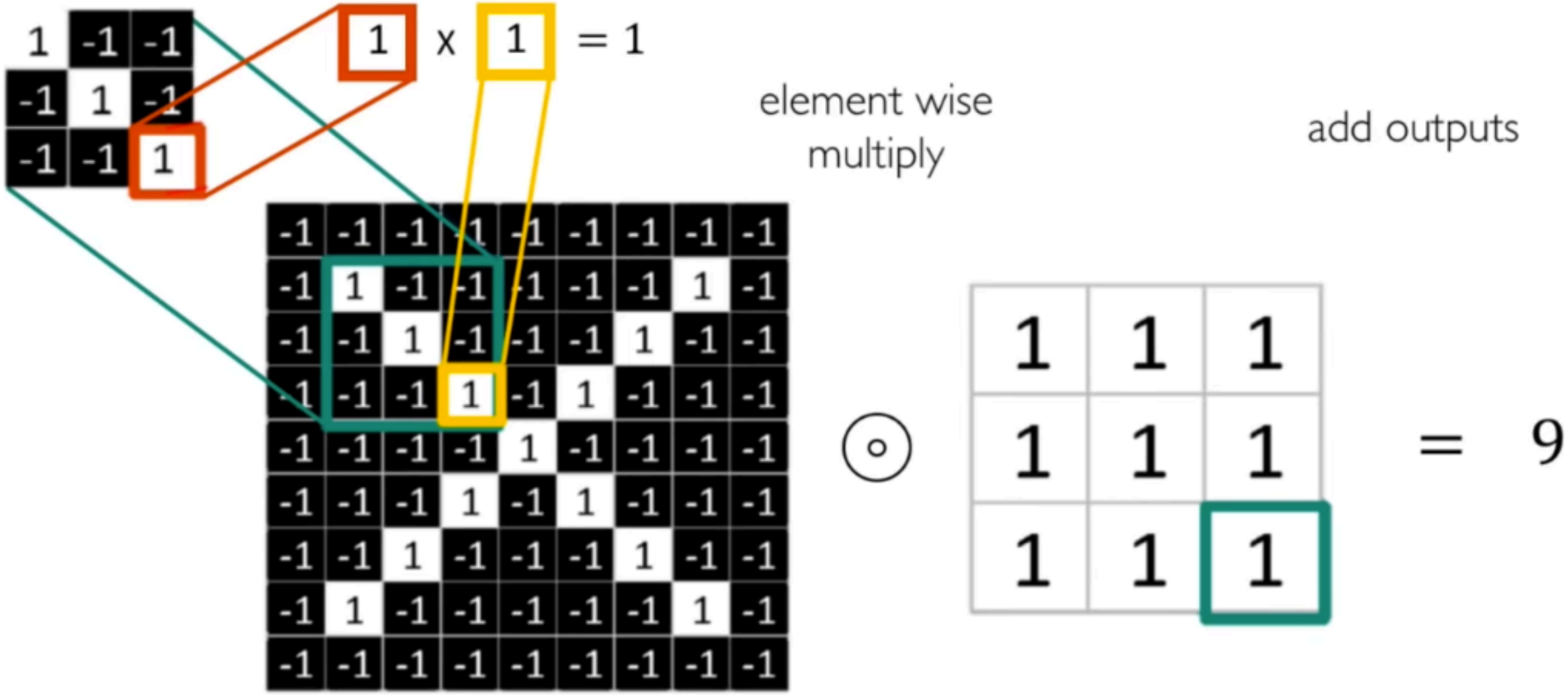
Convolutional neural networks (CNN)

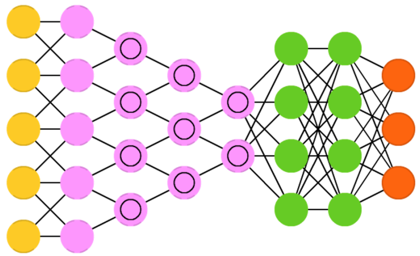




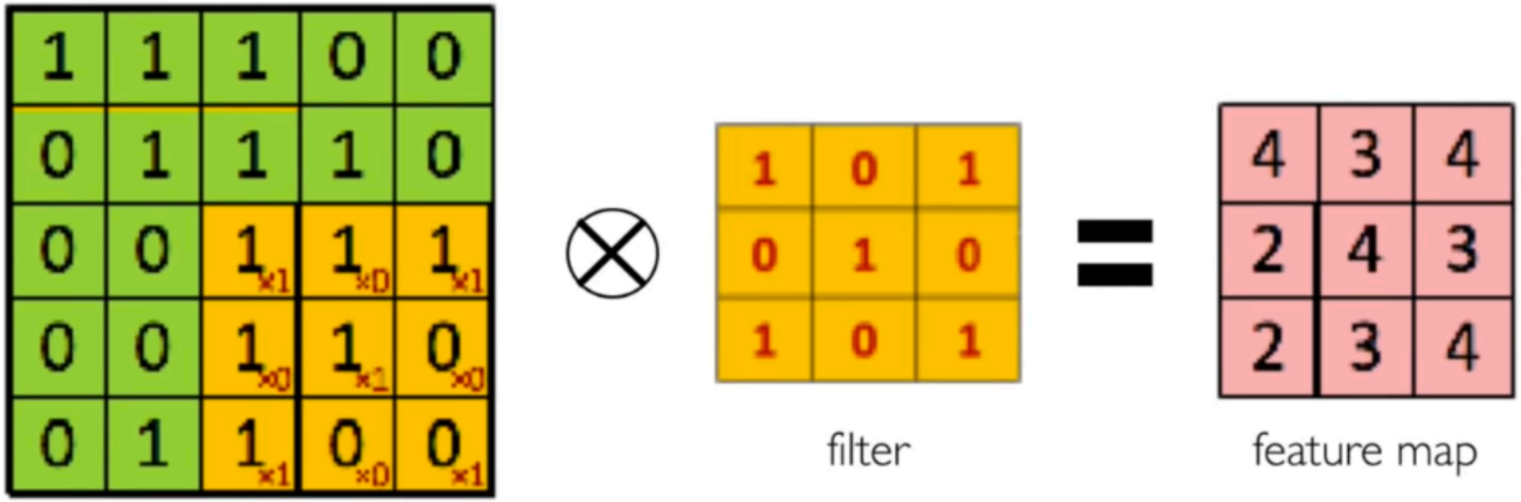


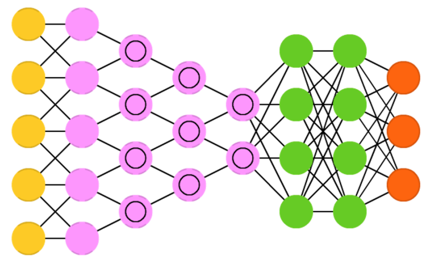
Convolutional neural networks (CNN)



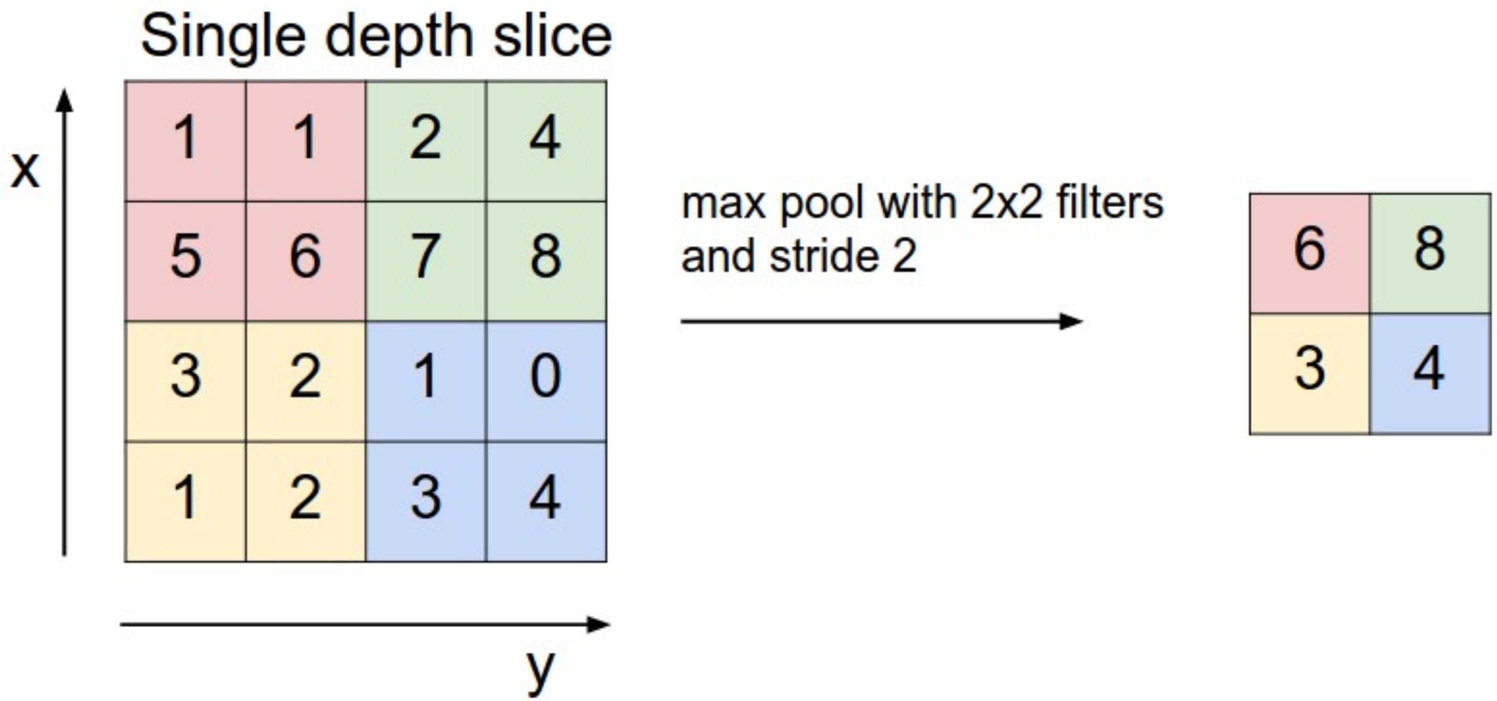


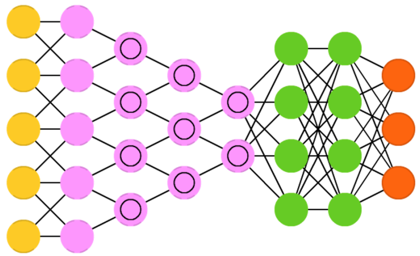
Convolutional neural networks (CNN)





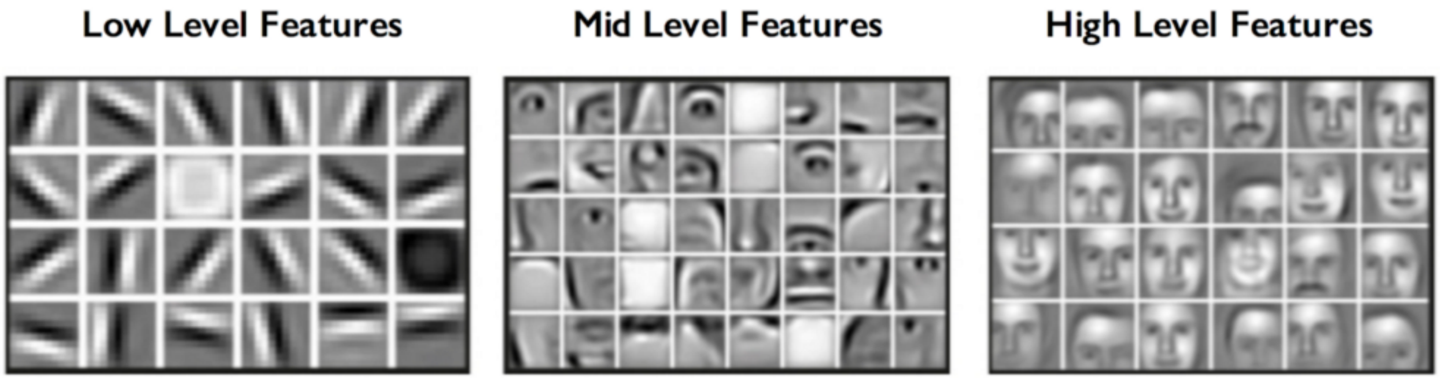
Convolutional neural networks (CNN)





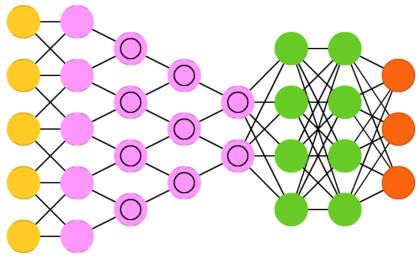
Convolutional neural networks (CNN)

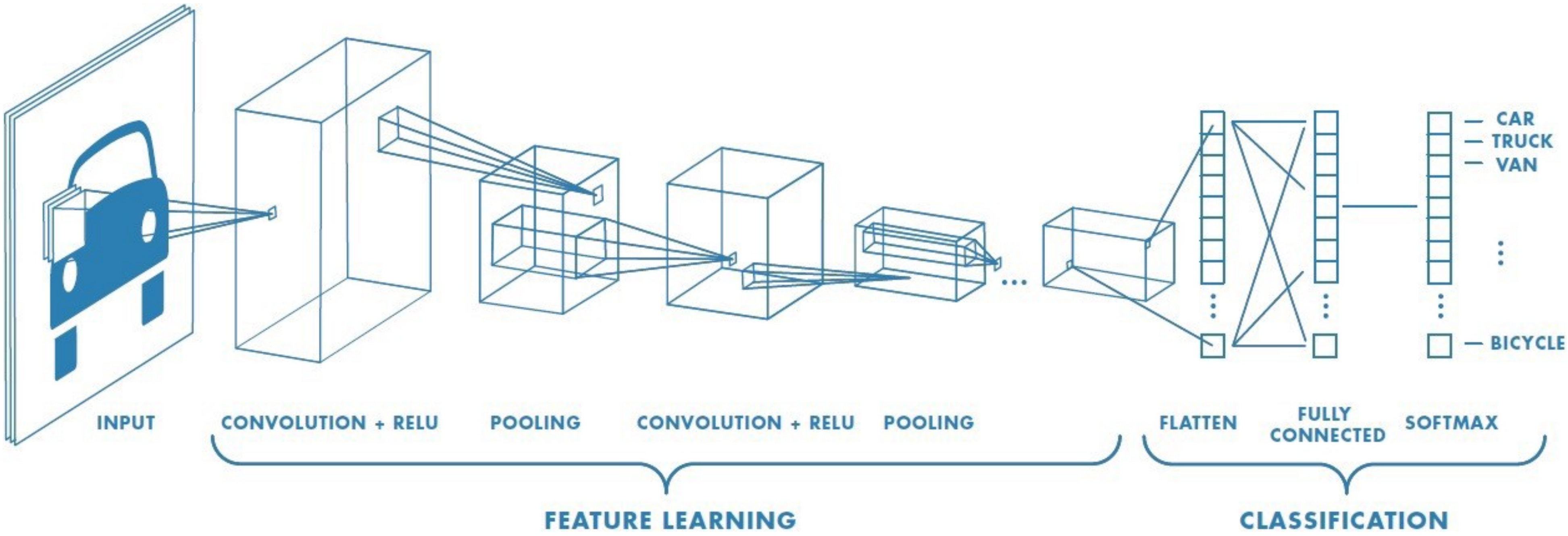
- Tasks: image or audio processing



Conv layer 1 Conv layer 2 Conv layer 3

Convolutional neural networks (CNN)





What is a Deep Learning Framework?

Frameworks: Introduction

- **TensorFlow**
- **Keras**
- **Pytorch**
- **Torch** (Collobert R., Kavukcuoglu K, Farabet C., 2002)
- **Caffe** (Berkeley Vision and Learning Center, 2013)
- **Caffe2** (Facebook, 2017, merged with PyTorch)
- **Theano** (University of Montreal, 20010-2017)
- **Chainer** (Preferred Networks, 2015)
- **Apache MXNet** (Apache Software Foundation , 2015)
- **CNTK** (Microsoft Research, 2016)
- **Deep Sparse Scalable Tensor Network Engine / DSSTNE** (Amazon, 2016)
- **BigDL** (Jason Dai (Intel) , 2016)
- **DyNet** (Carnegie Mellon University, 2017)

Frameworks: Introduction

Keras

Model-level library, as high level API running on top of backends: TensorFlow, CNTK, Theano, MXNet.

Pytorch

Machine learning library based on the Torch library and written in Python.

Tensorflow

Open source software library for numerical computation using data flow graphs.

Nodes: computations

Edges: tensor flows

Frameworks: Introduction

Keras

Initial Release: March 2015

Creator: François Chollet

Platforms: Linux, macOS, Windows

Beginner-friendly, modular, extensible, good for fast prototyping.

Pytorch

Initial Release: October 2016

Creator: Facebook AI Research lab

Platforms: Linux, macOS, Windows

Preferred for academic research and non-standard models. Easy debugging and fast training.

Tensorflow

Initial Release: November 2015

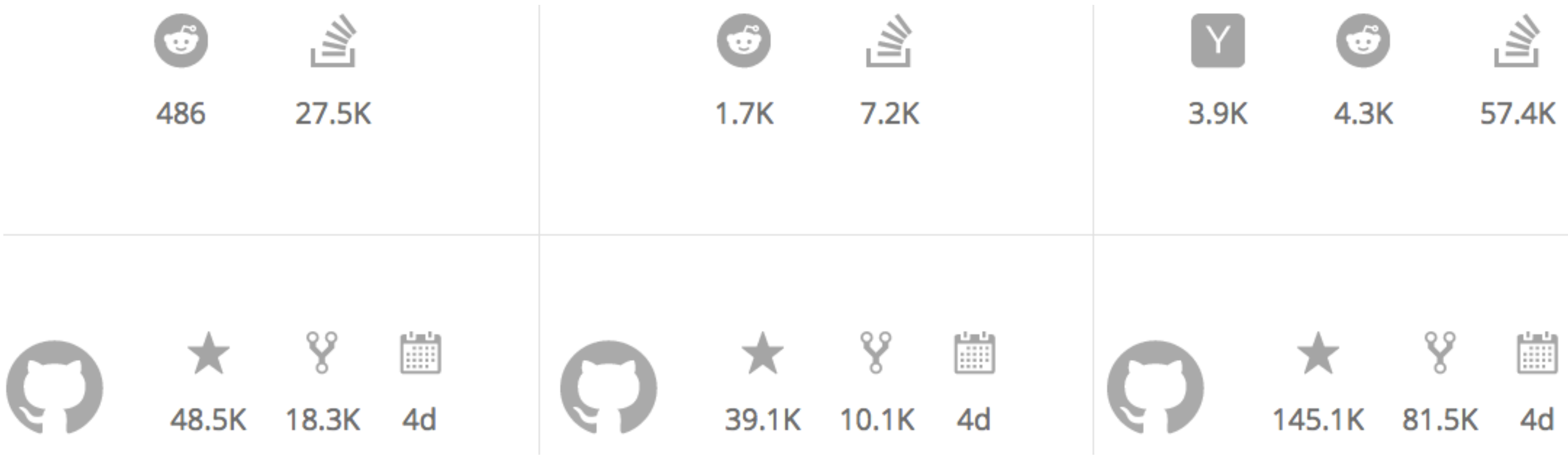
Creator: Google Brain

Platforms: Linux, macOS, Windows, Android, JavaScript

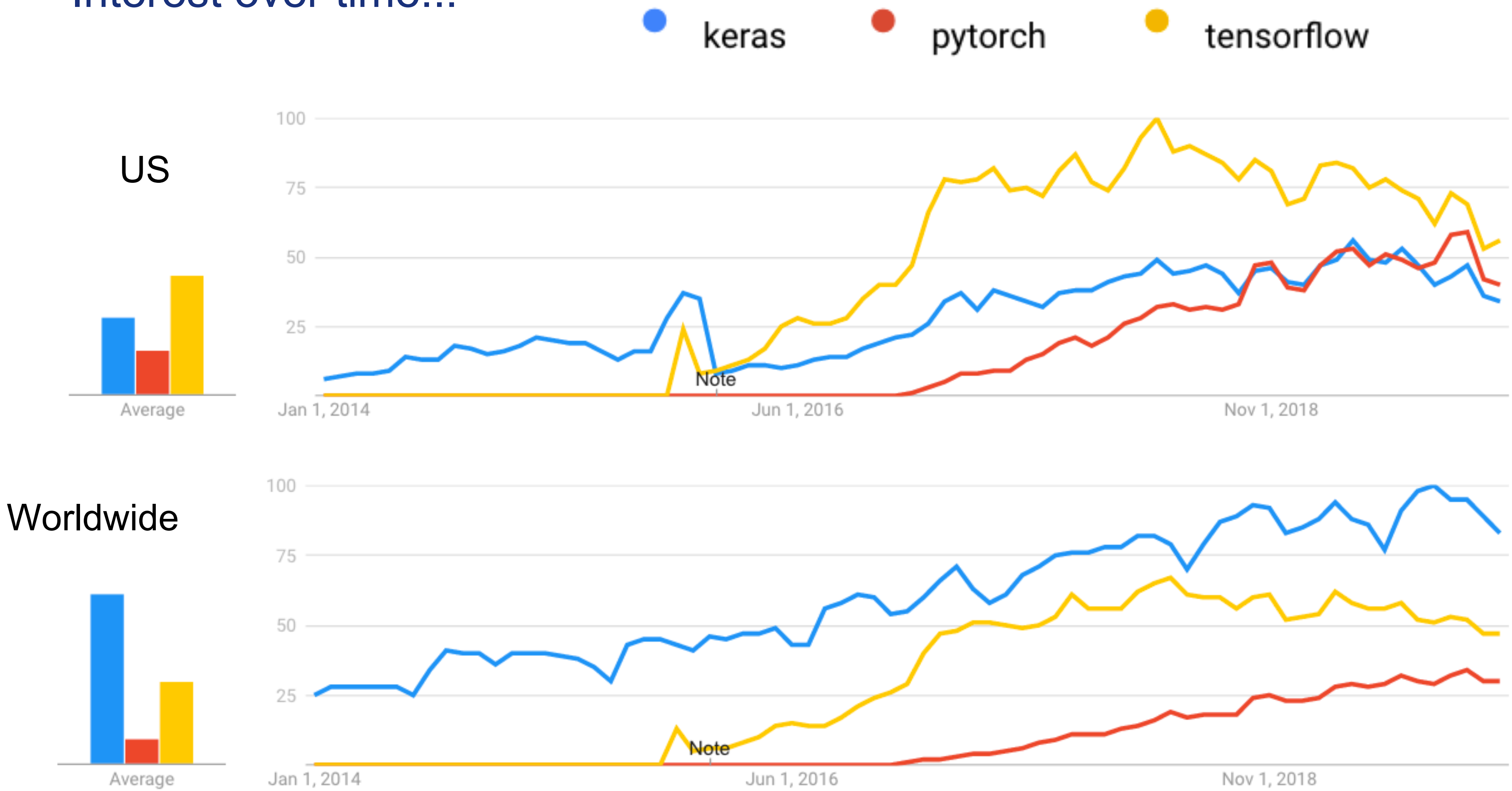
Preferred for industry. Very low-level, fast training, ideal for deployment to production and with great community support.

Frameworks: Introduction

Keras Pytorch Tensorflow



Interest over time...



Tensorflow Installation

TensorFlow Version 2 with CPU and GPU

- Python 3.5–3.7
- Ubuntu 16.04 or later
- Windows 7 or later
- macOS 10.12.6 (Sierra) or later
- Raspbian 9.0 or later

Older versions of TensorFlow

- Version 1.15 and older:

  CPU and GPU packages are separate:

  pip install tensorflow==1.15 # CPU

  pip install tensorflow-gpu==1.15 # GPU

PIP

```
pip install tensorflow
```

ANACONDA

```
conda install tensorflow
conda install tensorflow-gpu
```

Keras Installation

Keras version 2.3

- Python 2.7-3.6 in documentation (+3.7)
- First release to support TensorFlow 2.0
- Last major release of multi-backend Keras. Keras will be fully integrated in TF.
- With TensorFlow 2.0, you should be using tf.keras rather than the separate Keras package. Multi-backend Keras is superseded by tf.keras

~~from keras... import ...~~

from tensorflow.keras ... import ...

```
pip install keras
```

```
conda install keras
```

PyTorch installation

PyTorch Build	Stable (1.4)	Preview (Nightly)		
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python	C++ / Java		
CUDA	9.2	10.1	None	
Run this Command:	pip install torch torchvision			

pip

```
pip install torch torchvision
```

ANACONDA

```
conda install pytorch torchvision -c pytorch
```

Comparison table

#	Keras	Tensorflow	PyTorch
Ease of use	Easy and syntactically simple	Difficult without keras	Medium difficulty
Level of API	High	High and low	Low
Speed	Slower	Faster	Faster
Architecture	Simple	Complex	Complex
Debugging	Less frequent need to debug But hard debugging	Hard to debug	Best debugging capabilities
Community support	Larger	Larger	Smaller
Dataset	Smaller	Larger	Larger

Keras

Guiding principles

- **User friendliness:** minimize actions in common uses, clear user errors, simple API
- **Modularity:** model as sequence of fully configurable modules e.g. neural layers, cost functions, optimizers, activation functions etc.
- **Easy extensibility: n**ew modules are simple to add (as new classes and functions e.g. custom loss function, custom layers)

Keras

Pros:

- It has all the advantages that Tensorflow has to offer.
- Prototyping is fast and easy.
- You can run the same code with different backend engines.
- It is harder to make mistakes.

Keras

Cons:

- It is harder to pin down trouble line
- It does not handle low-level operations such as tensor products, convolutions and so on itself. It relies on its backend.
- It is consistently slower.
- It is much less configurable that Tensorflow or Pytorch.

Tensorflow

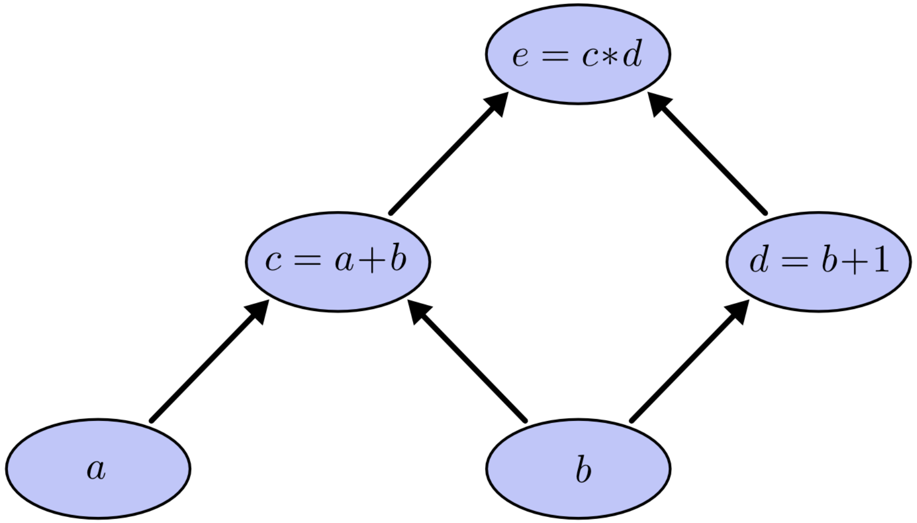
Tensorflow: graph execution engine for ML

- Computations in a neural network are organized in terms of:
  - a forward pass, in which we compute the outputs and
  - a backwards pass, in which we compute the gradients

- Computations are expressed as **dataflow graphs**.

  Graph nodes: mathematical operations

  Graph edges: multidimensional data arrays (tensors) flowing between nodes.



Tensorflow

Why Graphs in the first place?

- Graph as a platform-independent representation

  (deployed to non-pythonic infrastructure e.g. server, phone, GPU, TPU, Raspberry Pi)
- Automatic distribution to 100s machines
- Take advantage of graph-based optimizations
- Easy to differentiate graph (automatic differentiation)

Tensorflow: Dynamic vs Static graph definition

Tensorflow 2 introduced eager execution to add the dynamic graph capability.

Before eager execution:

"Define and Run" - the abstract data structures have to be defined in a Graph, before running the model. To then actually execute the code, a session must be used. In case of changes in the model architecture, you would have to retrain the model.

After eager execution:

- Automatic differentiation available for dynamic code
- Play with your model during building
- Really understand your model
- Improves performance in applications on sequential data e.g. machine translation

Tensorflow: Gradient Tape

With the dynamic computation graph, tensors are evaluated immediately and different operations can occur during each call.

Gradient Tape: Records operations for automatic differentiation.

- First, it records all forward-pass operations on a "tape".
- Next, it computes the gradients by "playing" the tape backwards.
- Then, it discards the tape.

Tensorflow

Other features:

- TensorFlow Extended: End-to-end platform for deploying production ML pipelines
- Tensorflow Serving: Flexible, high-performance serving system for machine learning models, designed for production environments
- TensorFlow has APIs available in several languages e.g. Python, JavaScript, C++, Java, Go, R, Swift (Early Release)
- TensorFlow Lite: Convert a TensorFlow model into a compressed flat buffer and deploy on a mobile.

Tensorflow

Other features:

- TensorFlow.js: A library for ML in JavaScript and use in the browser or Node.js
- Tensorflow Estimators: A high-level API with built in support for distributed training optimization. Has now integrated in Tensorflow, like Keras.
- Modules tf.image and tf.keras.preprocessing for image preprocessing.
- Tensorboard visualization library.
  - Tracking and visualizing metrics (loss and accuracy), parameters (weights, biases)
  - Visualizing the computational graph (ops and layers).
  - Displaying images, text and audio data.

Tensorflow

Pros:

- Simple built-in high-level API (**Keras**)
- **Eager execution** (dynamic computation graphs).
- Support for multiple languages to create deep learning models
- Visualizing training with **Tensorboard**.
- Scalable production **deployment** options, including on mobile (LITE).
- Good documentation and community support.

Tensorflow

Cons:

- It is very low level with a steep learning curve
- Demands extensive coding
- It is hard to make quick changes
- It is not always the fastest option

PyTorch

- Python version of Torch ML library (computational framework) open-sourced by Facebook
- In terms of high vs low level coding style, PyTorch lies somewhere in between Keras and TensorFlow.
- Autograd package of PyTorch builds dynamic computation graphs from tensors and automatically computes gradients.

PyTorch

Other features:

- TorchServe: model server that uses a RESTful API for both inference (prediction) and management calls (e.g. increase/decrease number of workers for specific model) and brings models to production faster.
- Easy debugging: use Python debugging tools such as pdb, ipdb, PyCharm debugger or old trusty print statements.
- Save model: PyTorch saves models in Pickles, which are Python-based and not portable.
- torch.nn.Module: Base class for all neural network modules. Allows creating reusable code which is very developer friendly.
- Visualizations: Pytorch uses Visdom for graphical represenations. Integration with TensorBoard also exists.
- torchvision.transforms: for common image transformations

PyTorch

Pros:

- Python-like coding
- Debugging is easy
- It is usually as fast as TensorFlow
- It has good documentation
- It provides lots of modular pieces that are easy to combine
- It is easy to write your own layer types
- It uses dynamic computation graphs

PyTorch

Cons:

- You usually write your own training code (Less plug and play)
- It has a smaller online community
- Third-parties are needed for visualization

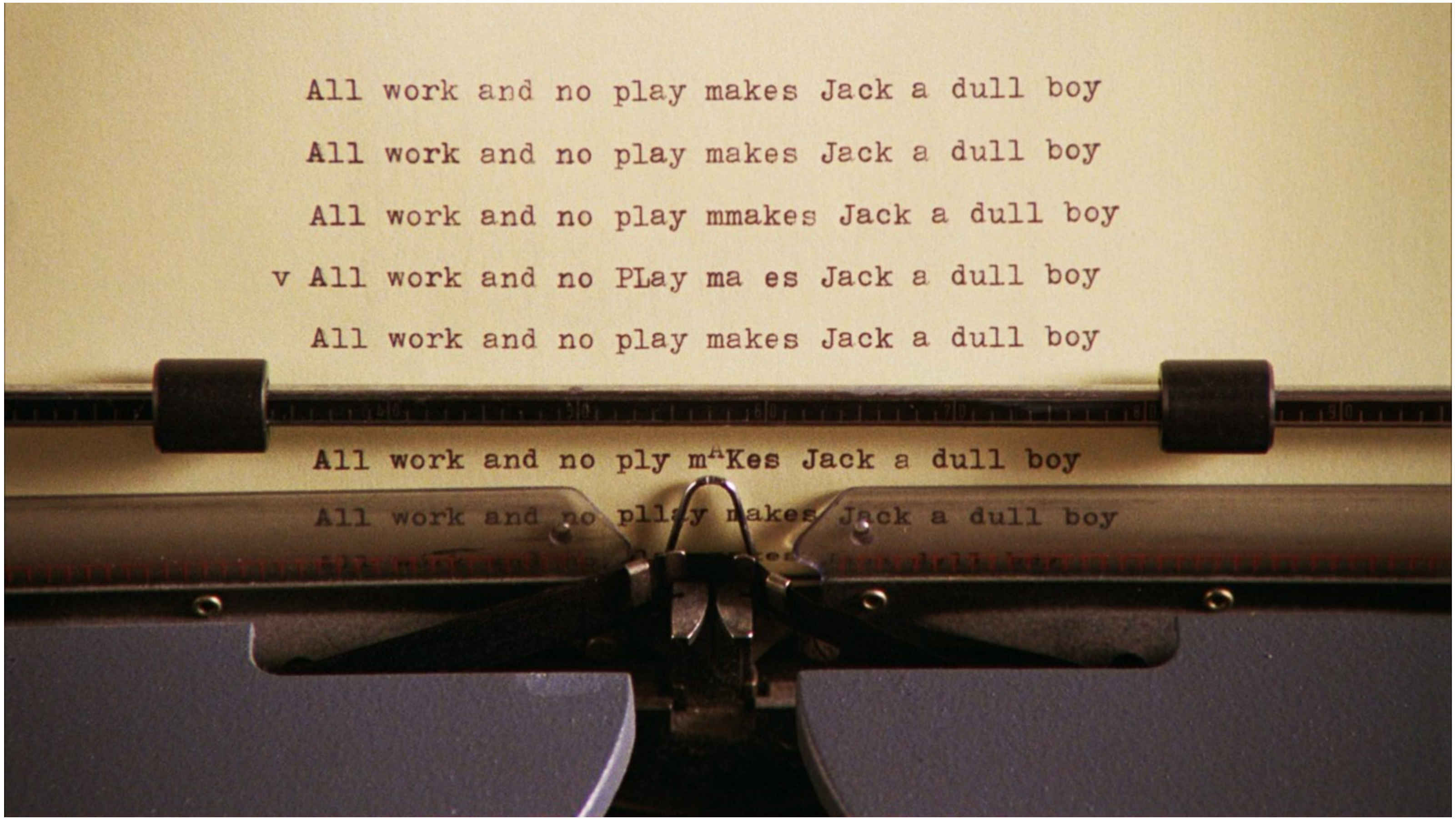
Best use case for each

- **Keras**
  - Beginner
  - Fast development
  - Small Dataset
  - Rapid Prototyping
  - Multiple back-end support

- **PyTorch**
  - Academic research
  - Non-standard implementation
  - Debugging capabilities
  - Pythonic

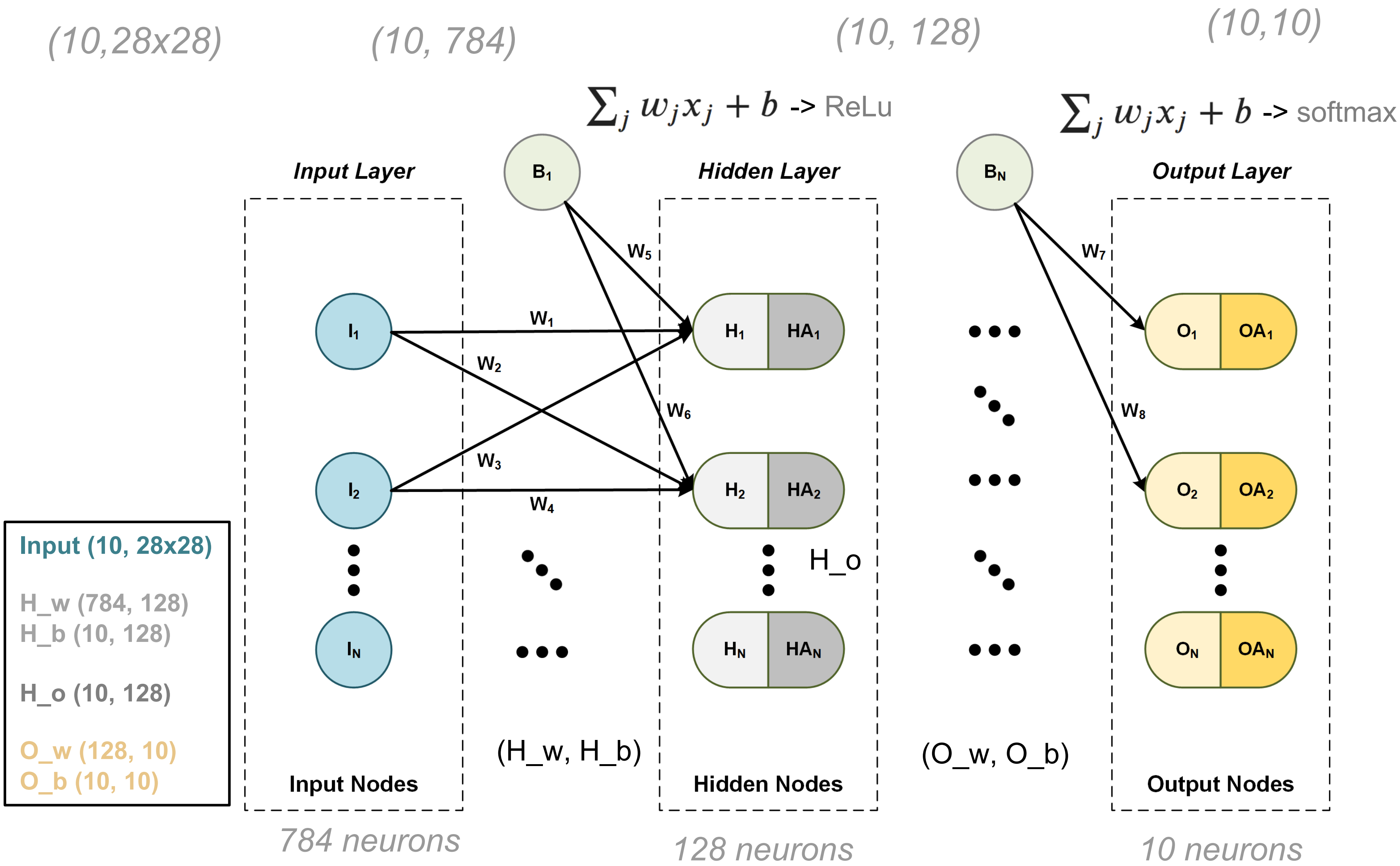
- **Tensorflow**
  - Industry
  - Deploy to production
  - Large Dataset
  - High Performance

Let’s see some code…



THANK YOU FOR YOUR ATTENTION

www.prace-ri.eu



(10,28x28)
(10, 784)
(10, 128)
(10,10)
$\sum_j w_j x_j + b$ -> ReLu
$\sum_j w_j x_j + b$ -> softmax
Input Layer
Hidden Layer
Output Layer
B_1
B_N
W_1
W_2
W_3
W_4
W_5
W_6
W_7
W_8
I_1
I_2
I_N
H_1 HA_1
H_2 HA_2
H_N HA_N
H_o
O_1 OA_1
O_2 OA_2
O_N OA_N
Input (10, 28x28)
H_w (784, 128)
H_b (10, 128)
H_o (10, 128)
O_w (128, 10)
O_b (10, 10)
(H_w, H_b)
(O_w, O_b)
Input Nodes
Hidden Nodes
Output Nodes
784 neurons
128 neurons
10 neurons