Porto Business School
/ University of Porto

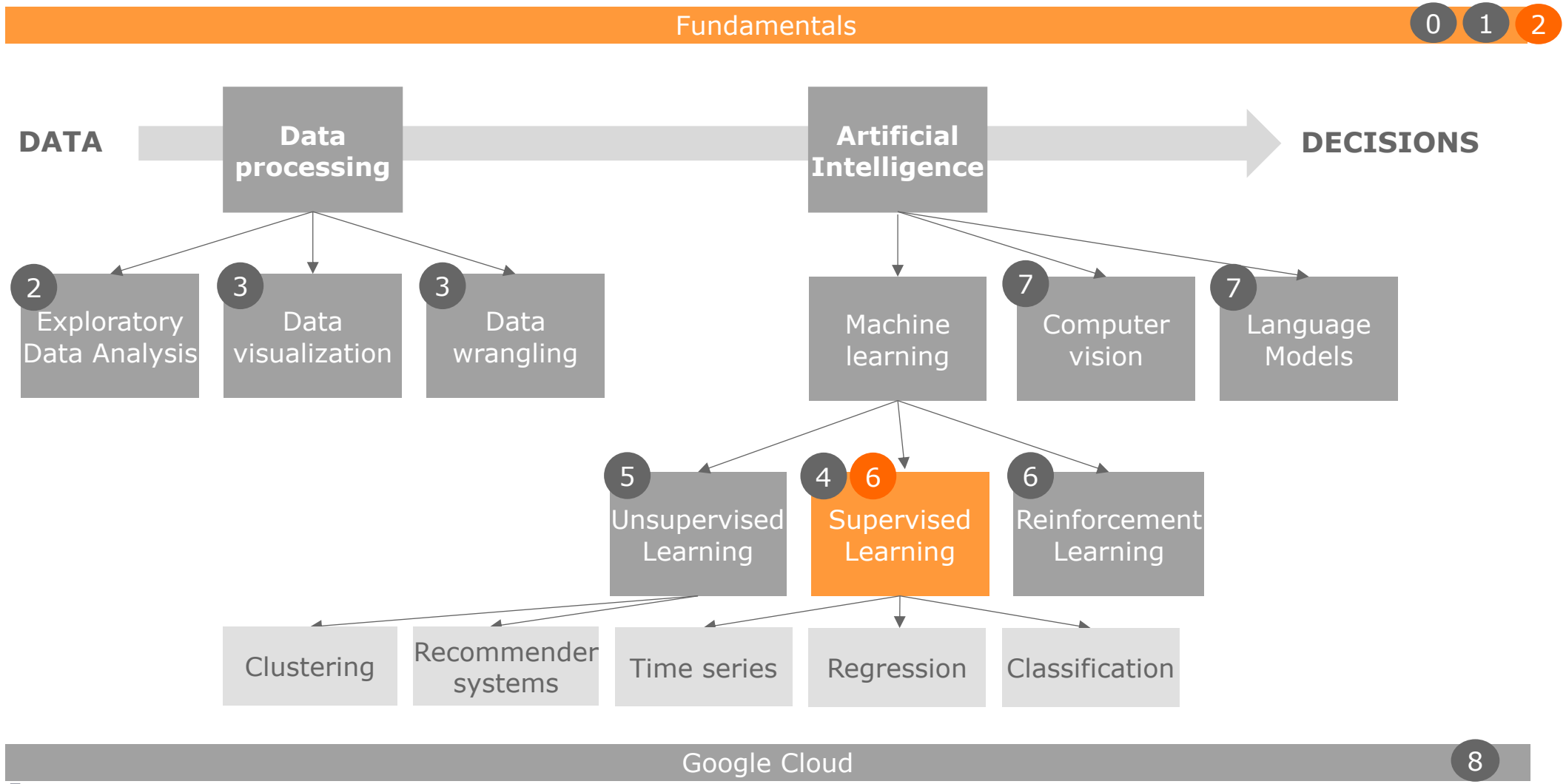
BPI Grupo CaixaBank

Advanced Learning models

Neural Networks

Tiago Cunha | Lisboa, 25/10/2023

Academia DTI



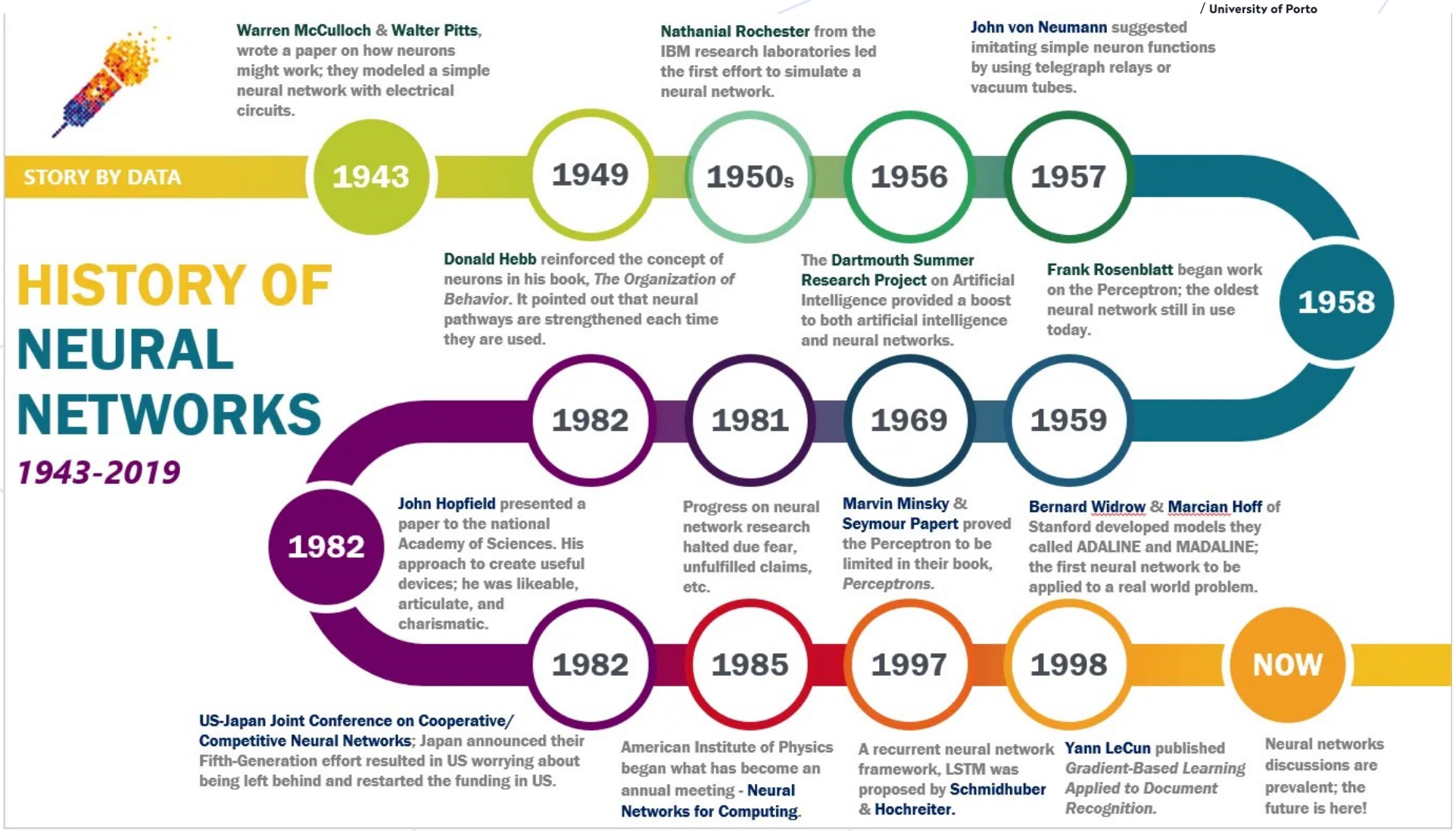
Advanced Learning models

Agenda

09h30 – 11h00	Introduction to Neural Networks
	Neural Network from scratch
	Keras
11h00 – 11h15	**Break**
11h15 – 13h00	Exercises
	Famous Neural Architectures
	Final remarks

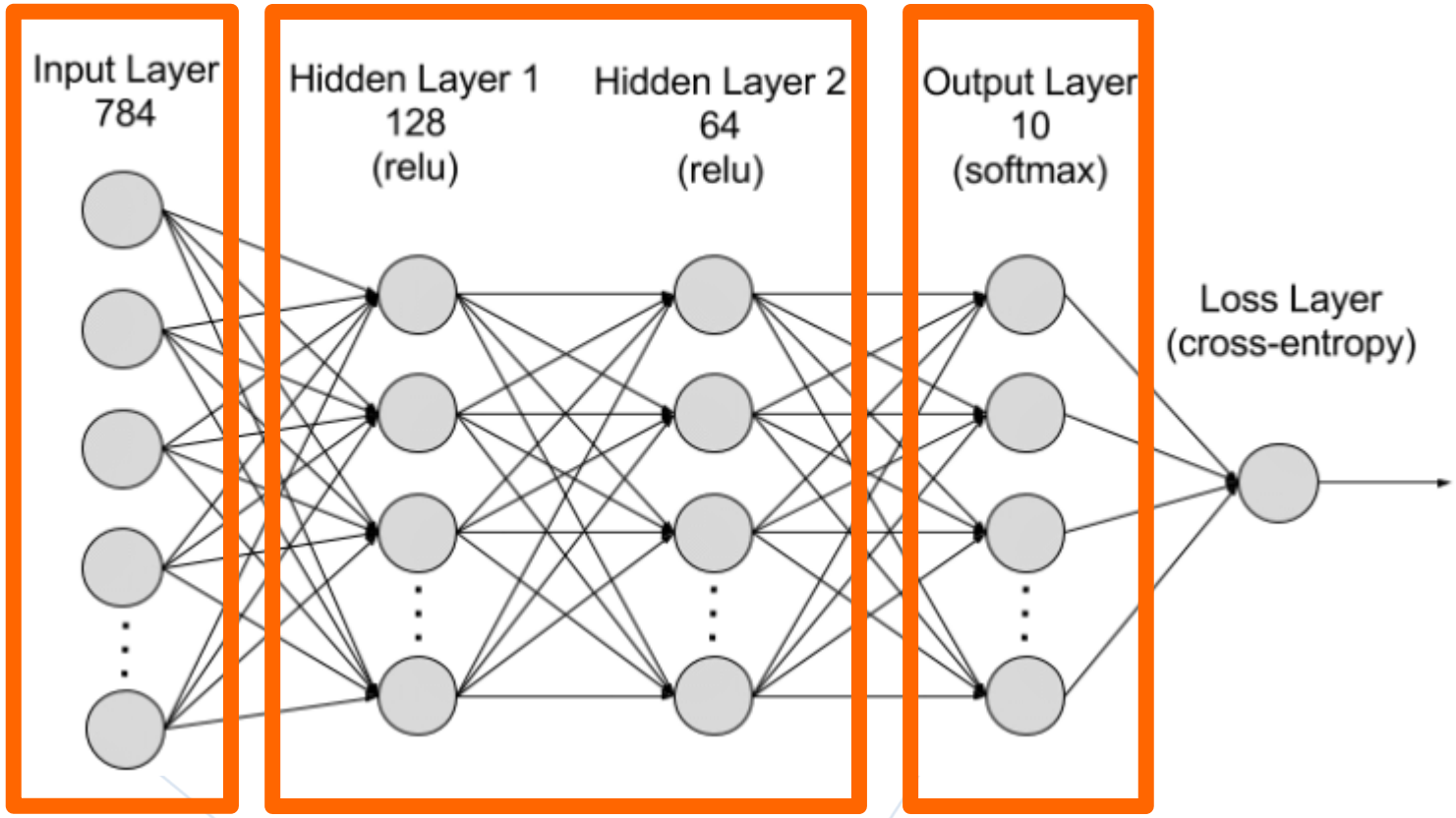
Introduction to Neural Networks

Neural Networks



Neural Networks

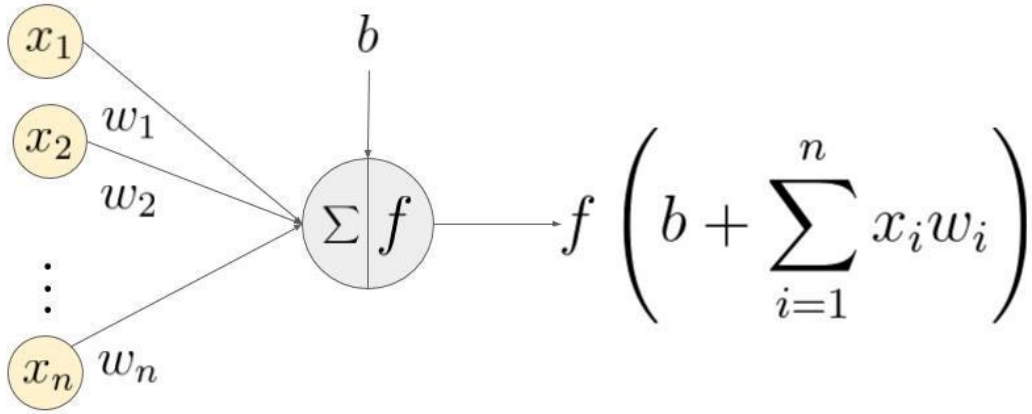
Building blocks



Neural Networks

Multi-Layer Perceptron ("vanilla" NN)

- Fully connected layers
  - each neuron applies a linear transformation to the input vector through a weight matrix
- Non-linear activation function
  - Hyperbolic tangent
  - Sigmoid



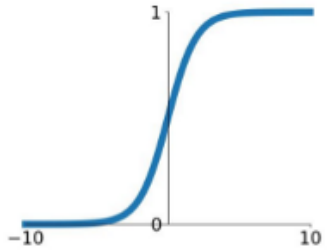
An example of a neuron showing the input (x_1 - x_n), their corresponding weights (w_1 - w_n), a bias (b) and the activation function f applied to the weighted sum of the inputs.

Neural Networks

Activation functions

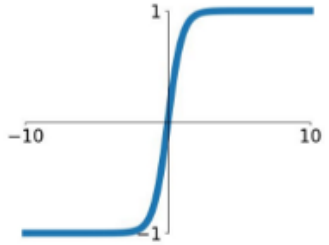
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



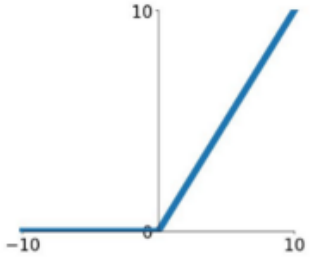
tanh

$$\tanh(x)$$



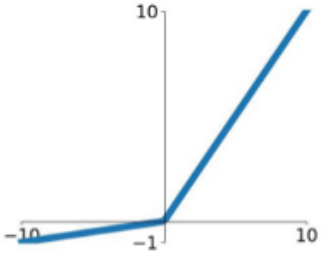
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

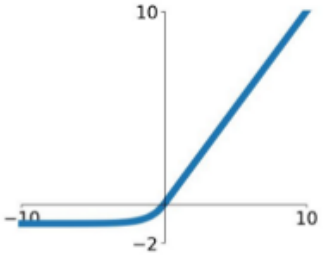


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Neural Networks

Activation functions in Output Layer

Problem	Activation Function
Regression	Linear
Binary Classification	Sigmoid
Multi-class Classification	Softmax

Sigmoid

2 classes

$$out = P(Y=class1|X)$$

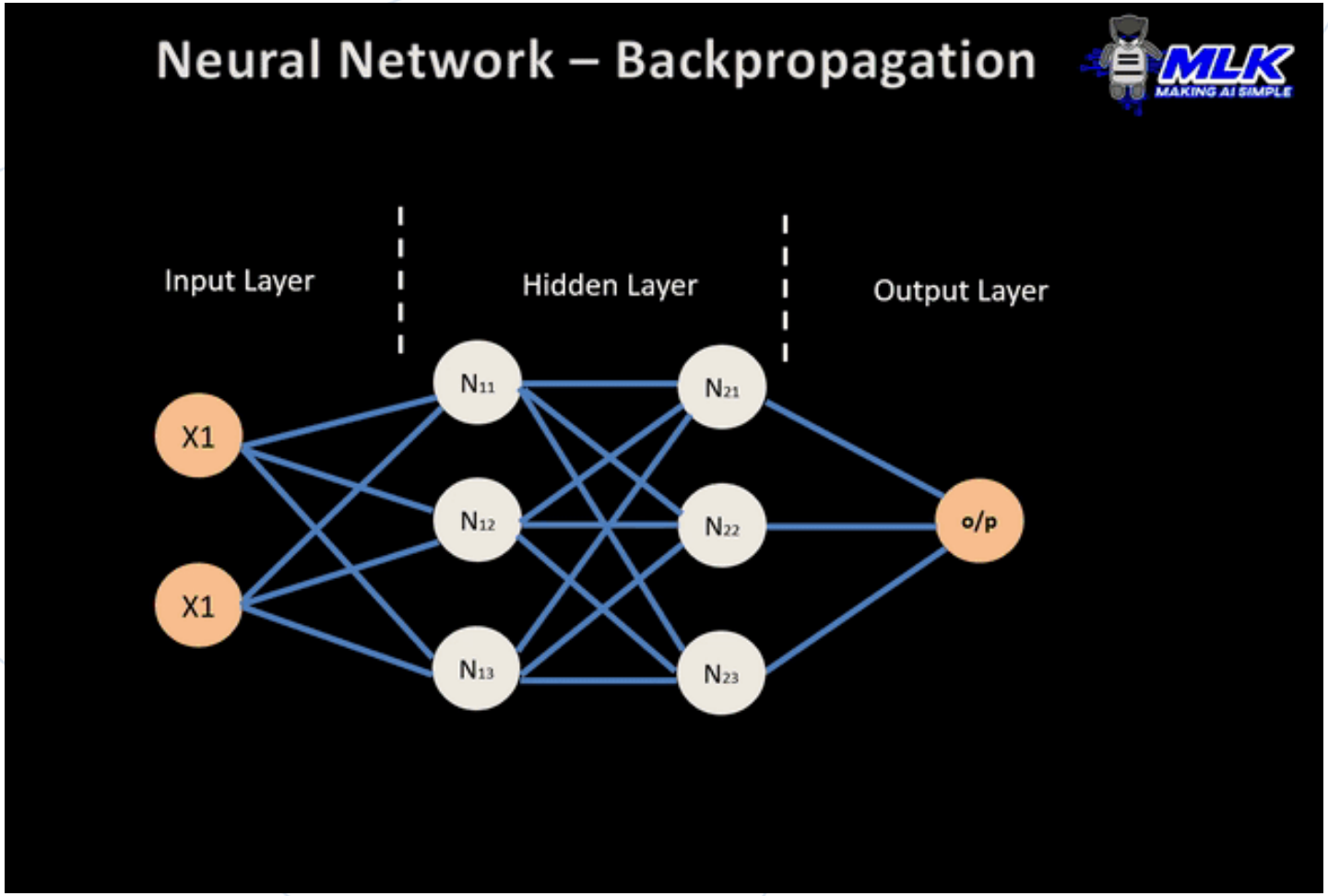
SoftMax

k>2 classes

$$out = \begin{bmatrix} P(Y=class1|X) \\ P(Y=class2|X) \\ P(Y=class3|X) \\ \vdots \\ P(Y=classk|X) \end{bmatrix}$$

Neural Networks

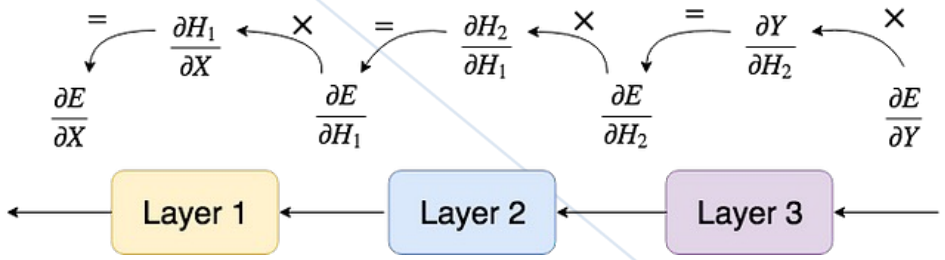
Feed-forward and Backpropagation



Neural Networks

Backpropagation

1. Calculate gradients
   - Chain Rule
2. Update weights
   - Start at last layer, move backward
3. Repeat until terminal status



Algorithm Backpropagation learning algorithm

Input:

A set of training examples $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$
A multilayer network with L layers, weights w_{ij}^{l}, and activation function f
Loss function $J(y, o)$
Learning rate $0 < \alpha < 1$
Number of epochs $epochs$

for each weight w_{ij}^{l} in the network **do**
  $w_{ij}^{l} \leftarrow$ a small random number
for $i = 1$ to $epochs$ **do**
  for each training example $(\mathbf{x}, y) \in D$ **do**
    /* Propagate the inputs forward to compute the outputs */
    for each neuron i in the input layer **do**
      $a_i^0 \leftarrow x_i$
    for $l = 2$ to L **do**
      for each neuron i in layer l **do**
        $z_i^l \leftarrow \sum_j w_{ij}^l a_j^{l-1}$
        $a_i^l \leftarrow f(z_i^l)$
    /* Propagate deltas backward from the output layer to the input layer */
    for each neuron i in the output layer **do**
      $\delta_i^L \leftarrow \frac{\partial J(y_i, o_i)}{\partial o_i} f'(z_i^L)$
    for $l = L - 1$ to 1 **do**
      for each neuron i in layer l **do**
        $\delta_i^l \leftarrow f'(z_i^l) \sum_j \left(w_{ji}^{l+1} \delta_j^{l+1}\right)$
    /* Update the weights using the deltas */
    for each weight w_{ij}^{l} in the network **do**
      $w_{ij}^{l} \leftarrow w_{ij}^{l} - \alpha \delta_i^l a_j^{l-1}$

BPI Grupo CaixaBank

Neural Networks

Gradient

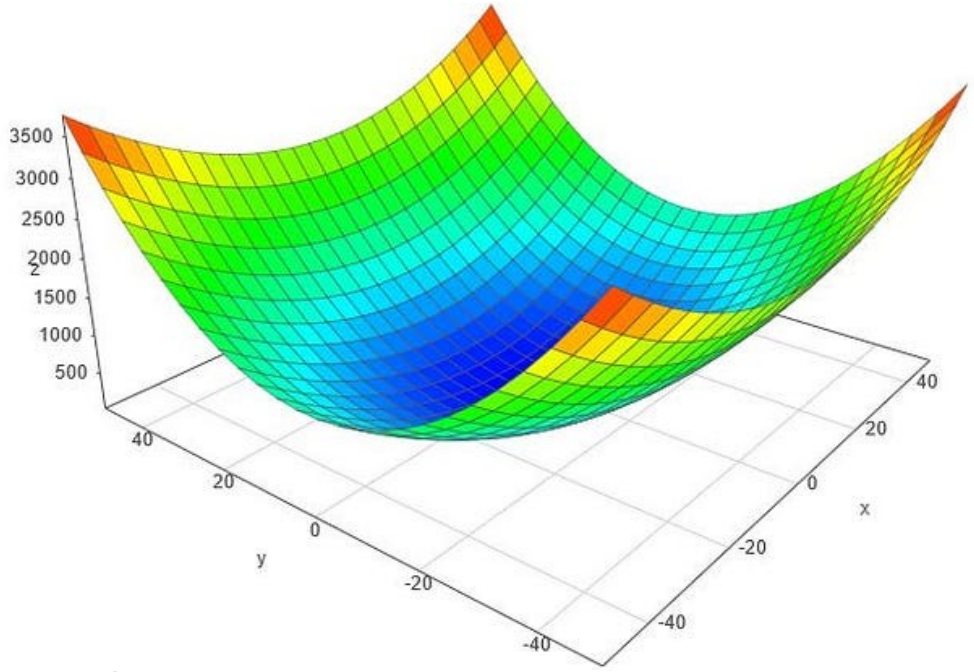
- Univariate function
  - First derivative at selected point
- Multivariate function
  - Vector of derivatives in each axis

- Partial derivatives
- Gradient
- Gradient at specific point

$$\frac{\partial f(x,y)}{\partial x} = x, \quad \frac{\partial f(x,y)}{\partial y} = 2y$$

$$\nabla f(x,y) = \begin{bmatrix} x \\ 2y \end{bmatrix}$$

$$\nabla f(10,10) = \begin{bmatrix} 10 \\ 20 \end{bmatrix}$$



$$f(x) = 0.5x^2 + y^2$$

Neural Networks

Gradient Descent

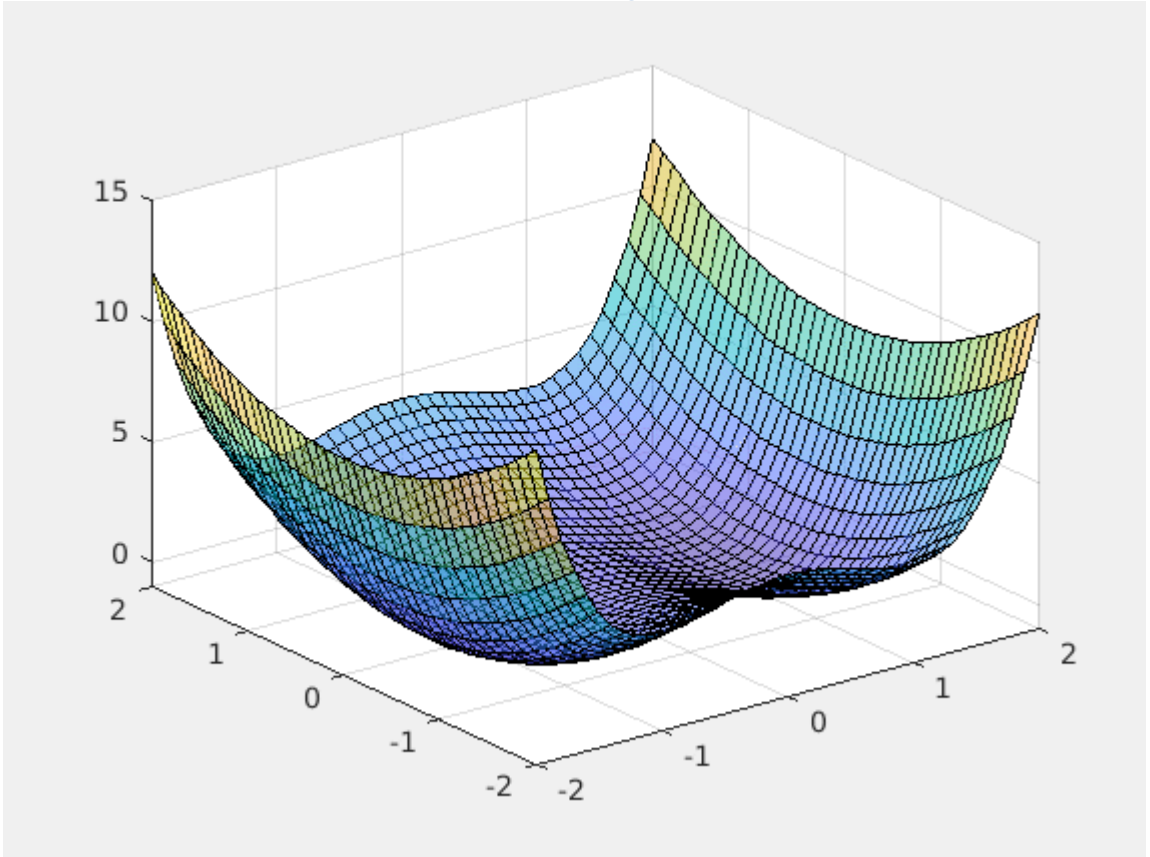
- Used to find local minimum of a differentiable and convex function
  - Important limitation in loss function

- Follow the direction of negative gradient to decrease value fast

GRADIENT

$$p_{n+1} = p_n - \gamma \nabla F(p_n)$$

STEP SIZE



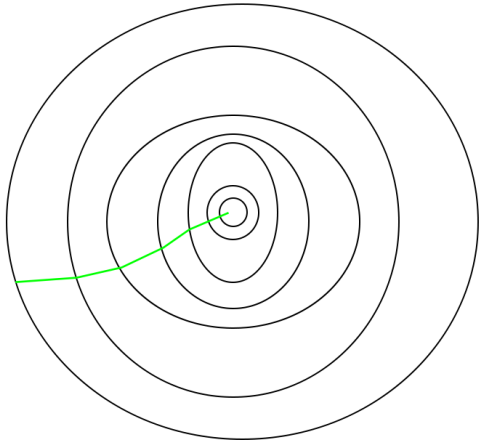
Gradient Descent algorithm

1. Choose a starting point (initialization)
2. Calculate gradient at this point
3. Make a scaled step in the opposite direction to the gradient (objective: minimize)
4. Repeat points 2 and 3 until one of the criteria is met:
   - maximum number of iterations reached
   - step size is smaller than the tolerance

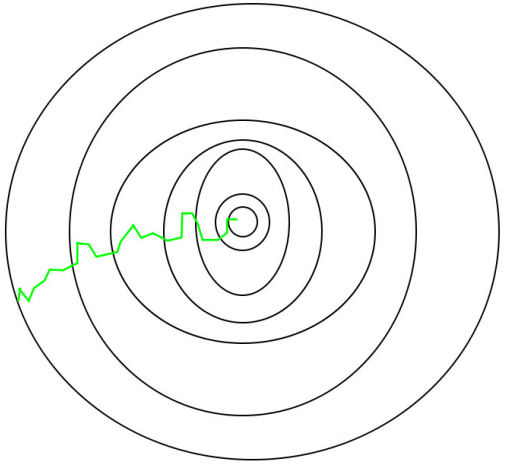
Neural Networks

Stochastic Gradient Descent (SGD)

- Gradient descent does not work well in large datasets
  - Too many calculations
- SGD: choose one example at random
  - Update all values based on the gradient at a single point
- Mini-batch SGD: select a random sample of points
  - Balance between speed and effectiveness



GRADIENT DESCENT



STOCHASTIC GRADIENT DESCENT

Neural Networks

Loss Functions

Regression

- Mean Absolute Error

$$\mathcal{L}_{\mathcal{MAE}} = \frac{1}{N}\sum_{i=1}^{N}|y_i - f(x_i)|$$

- Mean Squared Error

$$\mathcal{L}_{\mathcal{MSE}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2$$

- Root mean Squared Error

$$\mathcal{L}_{\mathcal{RMSE}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2}$$

Classification

- Binary cross entropy

$$\mathcal{L}_{\mathcal{BCE}} = \frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(x_i)) + (1 - y_i) \cdot log(1 - p(x_i))$$

- Cross entropy Loss

$$\mathcal{L}_{\mathcal{CE}} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}.log(f(x_{ij}))$$

$$N : samples; M : classes$$

- KL Divergence

$$\mathcal{L}_{\mathcal{KL}} = \sum_{i=1}^{N} y_i \cdot log(\frac{y_i}{f(x_i)})$$

Neural Networks

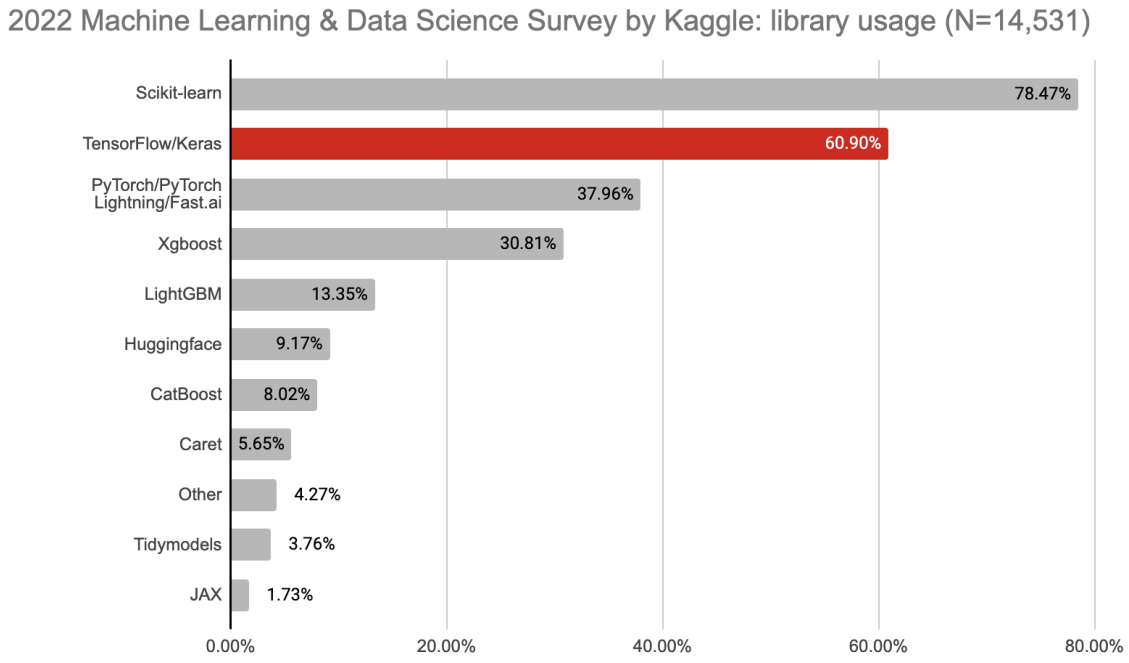
Neural Network from scratch:

- https://colab.research.google.com/drive/1ldgyRVenf6fs-Hlqe6sUFzsDr6xdtrfc#scrollTo=oTrTMpTwtLXd

Keras

Neural Networks

Keras



Advantages:

- Consistent and extensible API
- User friendly framework
- It supports multiple platforms and backends
- It runs on both CPU and GPU
- Highly scalability of computation

Neural Networks

Sequential Model API

- Sequence of Layers
  - Covers most NN architectures

Layers (https://keras.io/api/layers/)

- Dense
- Embedding
- Convolution
- Pooling
- Recurrent
- Dropout
- …

```
model = keras.Sequential(
    [
        layers.Dense(2, activation="relu"),
        layers.Dense(3, activation="relu"),
        layers.Dense(4),
    ]
)
```

```
model = keras.Sequential()
model.add(layers.Dense(2, activation="relu"))
model.add(layers.Dense(3, activation="relu"))
model.add(layers.Dense(4))
```

Neural Networks

Sequential Model API (https://keras.io/api/models/model_training_apis/)

- model.compile(optimizer='adam',loss='binary_crossentropy', metrics=['accuracy'])
- model.fit(X_train, y_train, batch_size=32, epochs=50, validation_split=0.2)
- model.evaluate(X_test, y_test)
- model.predict(X_test.iloc[0:4])

Neural Networks

Sequential Model API

- Activation functions https://keras.io/api/layers/activations/
- Losses https://keras.io/api/losses/
- Optimizer https://keras.io/api/optimizers/
- Regularizers https://keras.io/api/layers/regularizers/

  https://keras.io/api/layers/regularization_layers/

Neural Networks

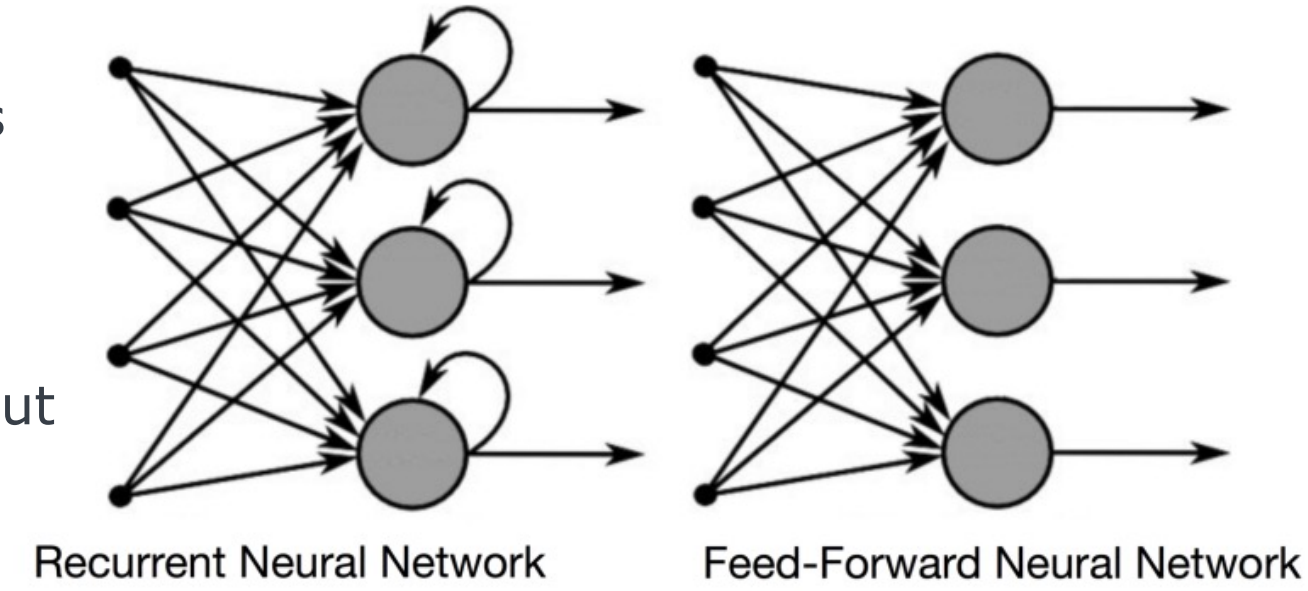
- Regression:
  - Example:
    - https://colab.research.google.com/drive/1P85WeIBnY5DIL17XEtLHoV7N75dtcStW#scrollTo=FkCBbJ2S8szj
- Classification:
  - Exercise:
    - https://colab.research.google.com/drive/11RFXWEsl8GoV6lHx_I1U7bYSV4h4nvXW
  - Solution:
    - https://colab.research.google.com/drive/1Sm1ZfW91EtKrkkEhJrCWwB5mcSuBi7GO

Famous Neural architectures

Neural Networks

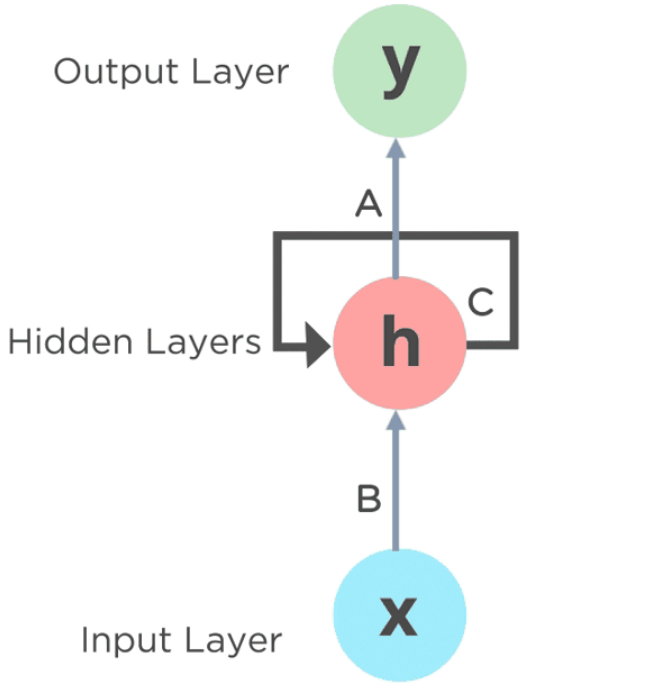
Recurrent Neural Network

- Feedback loops
  - Signals travelling in both directions
- Ability to memorize
  - Features derived from previous input are used in current calculations
- Useful in sequence data
  - Text, audio, video, etc.



Neural Networks

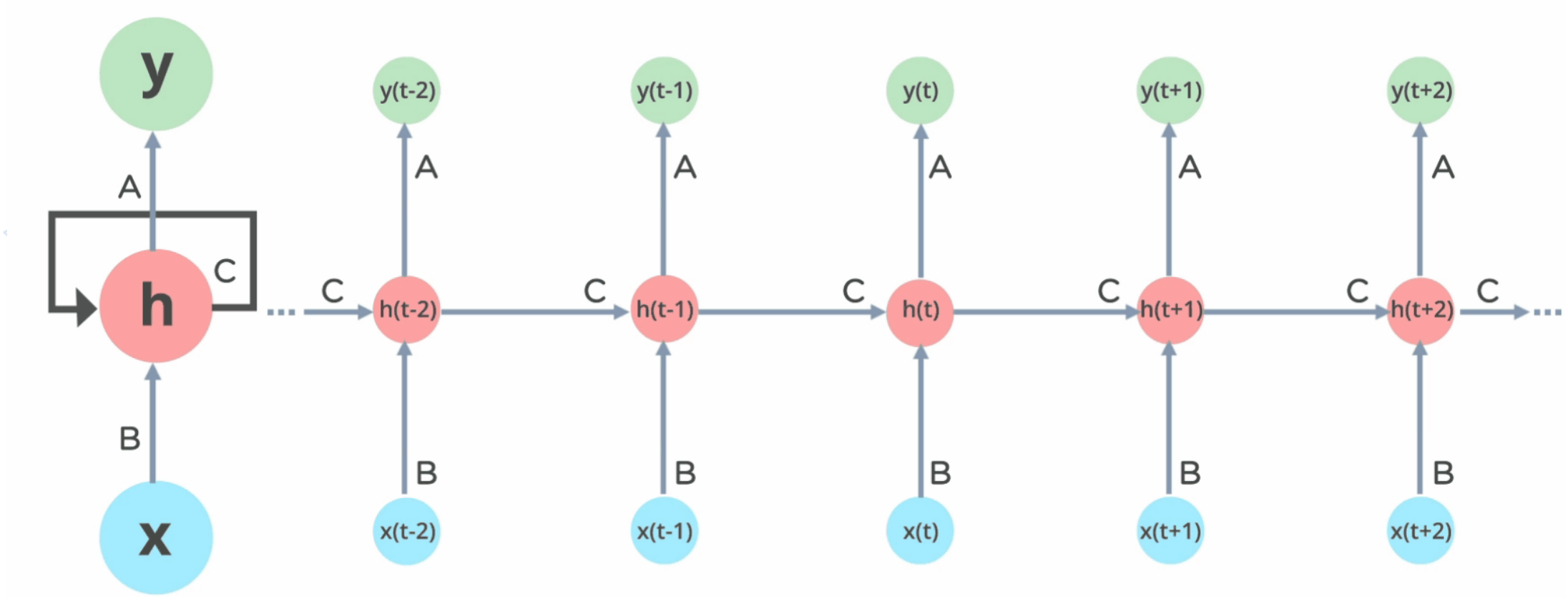
RNN - unfolding



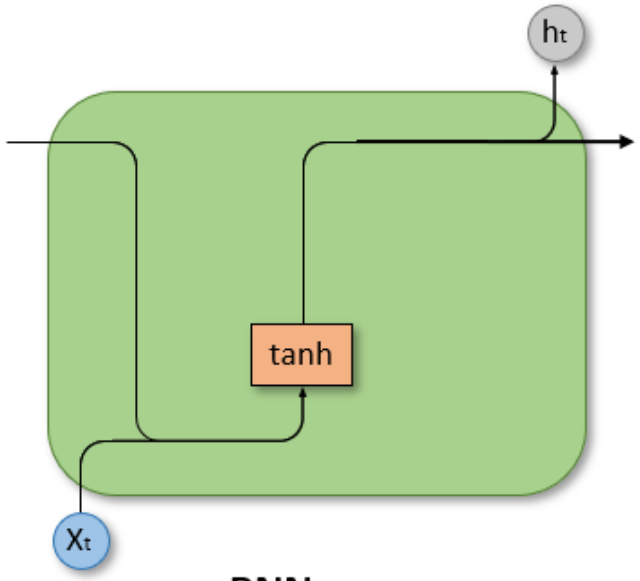
A, B and C are the parameters

Neural Networks

RNN – unfolding



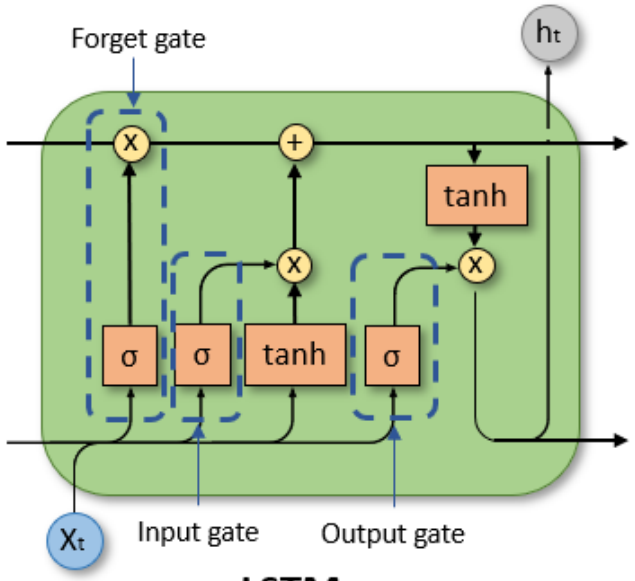
RNN vs Long Short-Term memory (LSTM) vs Gated Recurrent Unit (GRU)



RNN

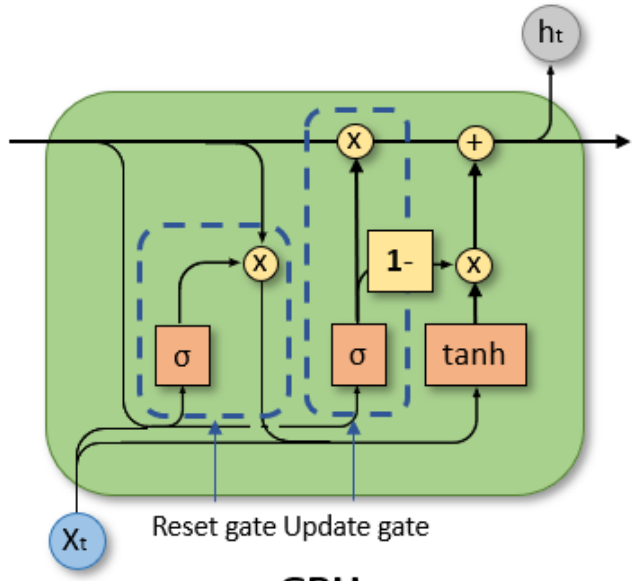
VANISHING GRADIENTS

it's too difficult for RNN to learn to preserve information over many timesteps.



LSTM

(1) Decide what to forget (state)
(2) Decide what to remember (state)
(3) The actual "forgetting" and update of the state
(4) Production of the output

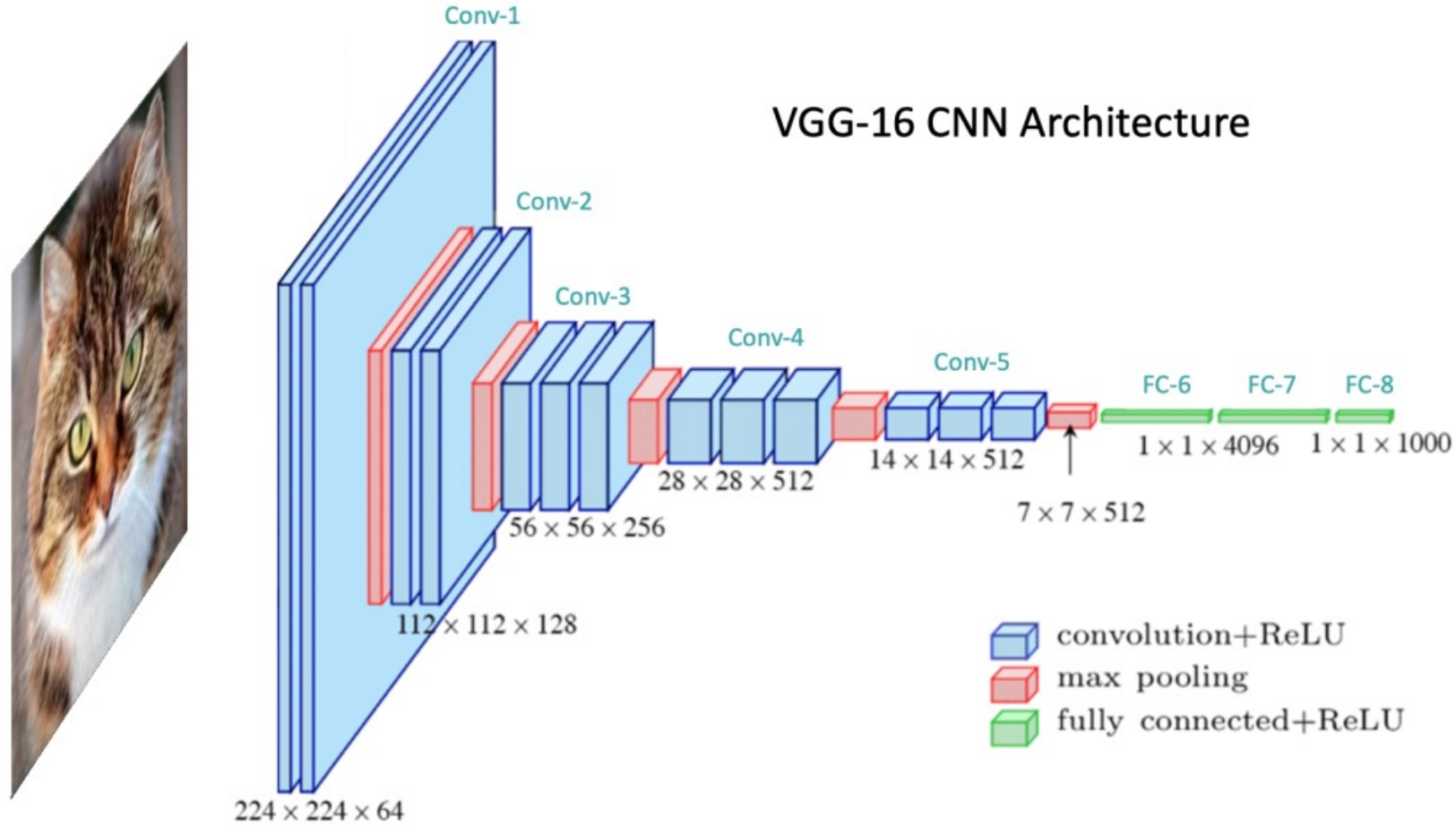


GRU

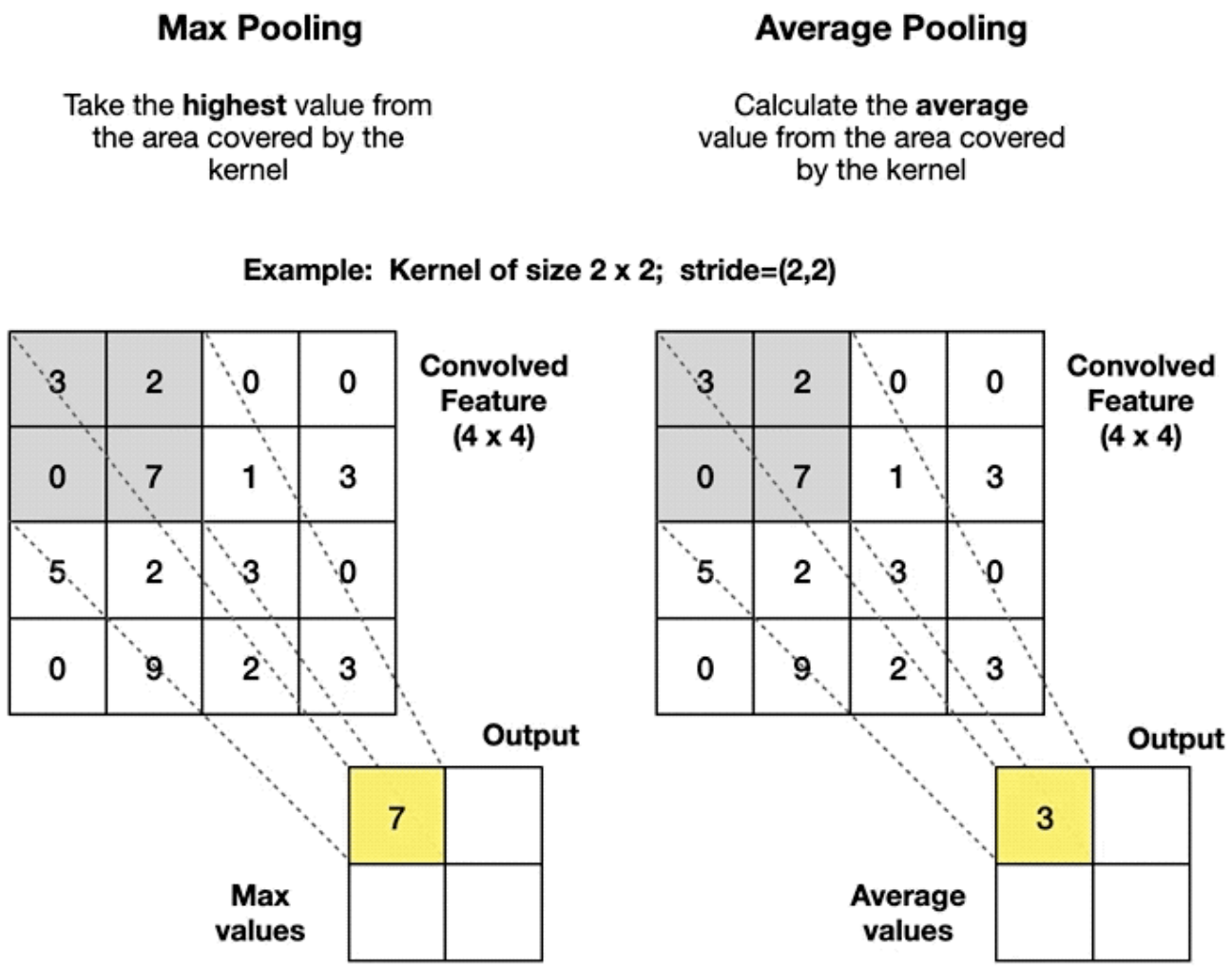
(1) the update gate acts similarly to the forget and input gate of an LSTM, it decides what information to keep and which to throw away, and what new information to add.
(2) the reset gate is used to decide how much of the past information to forget.

Neural Network

Convolutional Neural Network (CNN)



CNN – Convolution and Pooling



Neural Networks

- Image classifier
  - Exercise:
    - https://colab.research.google.com/drive/1HxEsn3OPPdiLek7SeH9Bh-SpUVd7MorU#scrollTo=x6gabqT8gXqZ
  - Solution:
    - https://colab.research.google.com/drive/1ItvsrrBxJz8dOIlsvFKYMRMrJs2_wZ-5

Neural Networks

More resources

- Goodfellow et al. Deep learning https://www.deeplearningbook.org/
- Deep Learning Specialization https://www.deeplearning.ai/courses/deep-learning-specialization/
- More Keras examples: https://keras.io/examples/

Wrap-up

- ✓ Conclusion #1 – Deep learning methods are based on
- ✓ Conclusion #2 – Backpropagation and Gradient Descent are key components to fit Neural Network models
- ✓ Conclusion #3 - The power of Deep Learning methods comes from allowing the model to find the best encoding of information
- ✓ Conclusion #4 – Each Neural architecture is designed for a particular task and always requires tuning to specific problem
- ✓ Conclusion #5 – Keras provides an easy interface to build and test Deep Learning models

Make change happen