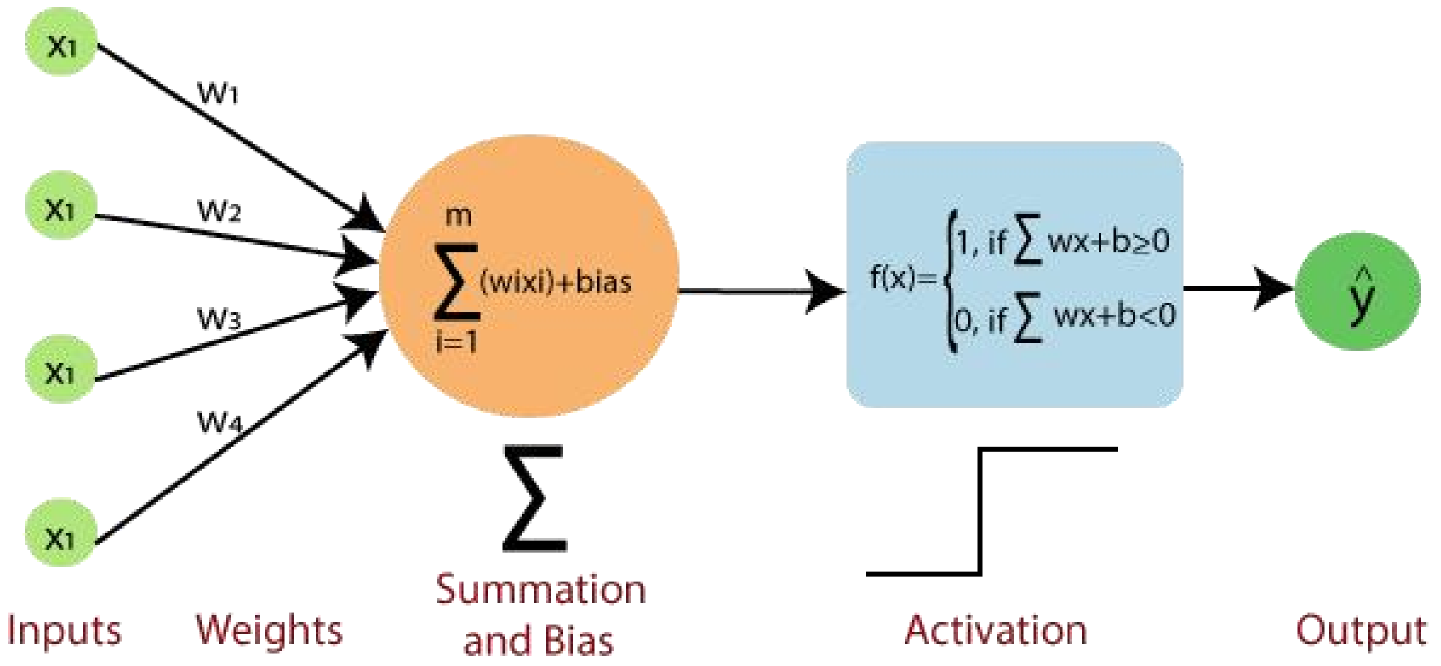
How to Build your First Neural Network

Keras & TensorFlow

Hichem Felouat
hichemfel@gmail.com

Perceptron

- The inputs and output are numbers and each input connection is associated with a weight.
- Compute the **weighted sum of its inputs** then applies an **activation function** to that sum and **outputs the result**.



Algorithm for Training a Perceptron

Input :($\mathcal{D}$, $\mathbf{w}^{(0)}$).

Output :$\mathbf{w}$.

for each data point $\mathbf{x}^{(j)}, j = 1, \dots n$, **do**

  if $\left(y^{(j)} = +1 \text{ and } f(\mathbf{x}^{(j)}) \leq 0\right)$ **then**

    $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \alpha\,\mathbf{x}^{(j)}$

  else **if** $\left(y^{(j)} = -1 \text{ and } f(\mathbf{x}^{(j)}) \geq 0\right)$ **then**

      $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha\,\mathbf{x}^{(j)}$

    else

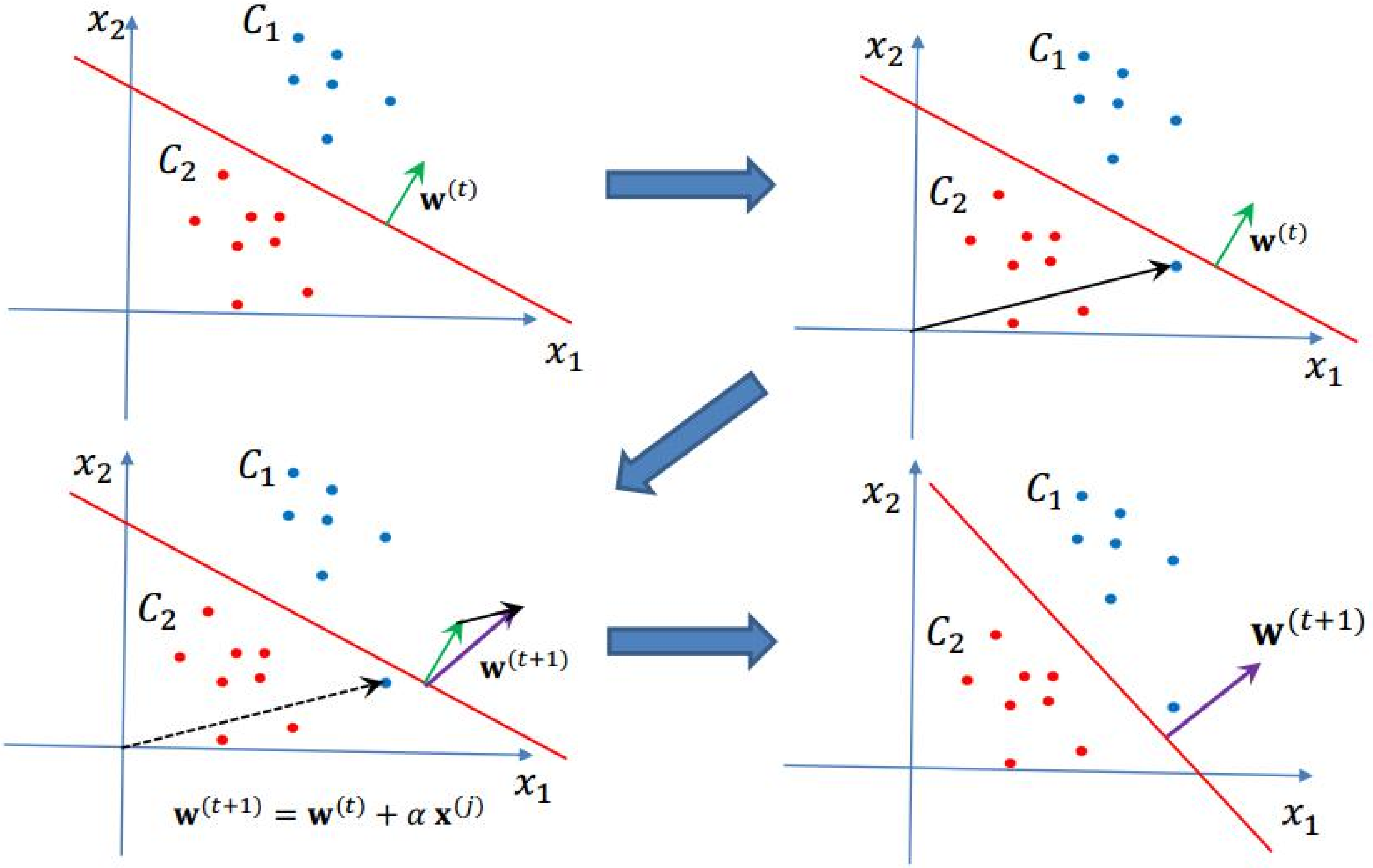
      $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)}$.

    end

  end

end

Algorithm for Training a Perceptron



Limitations of Perceptron

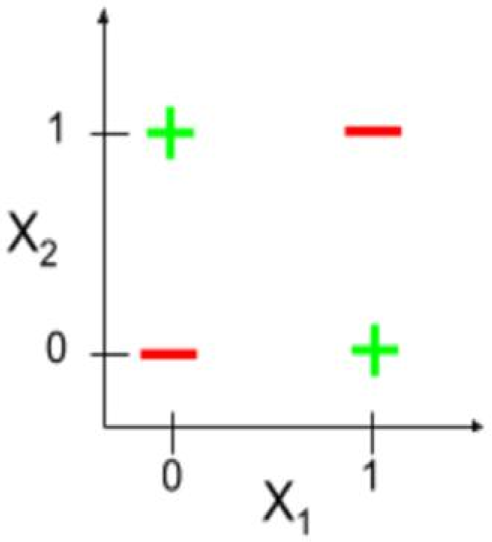
A perceptron can only separate **linearly separable classes**, but it is unable to separate **non-linear class boundaries**.

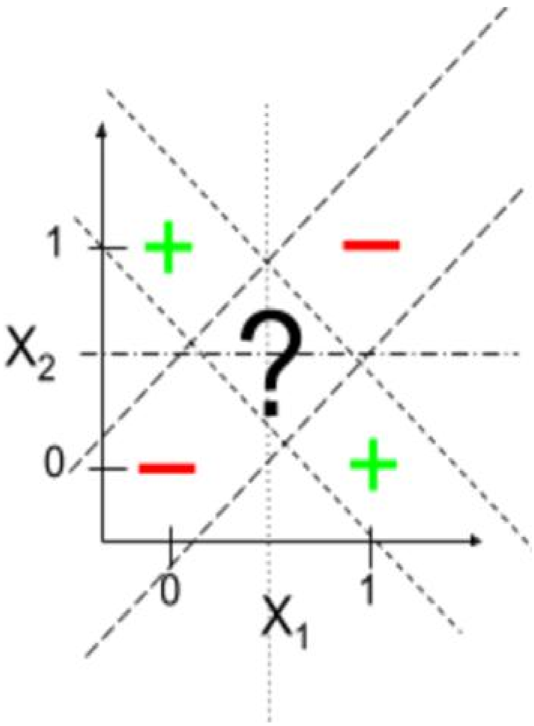
Example: Let the following problem of binary classification(problem of the **XOR**).

Clearly, no line can separate the two classes!

Solution :

- **Use two lines instead of one!**
- **Use an intermediary layer of neurons in the NN.**

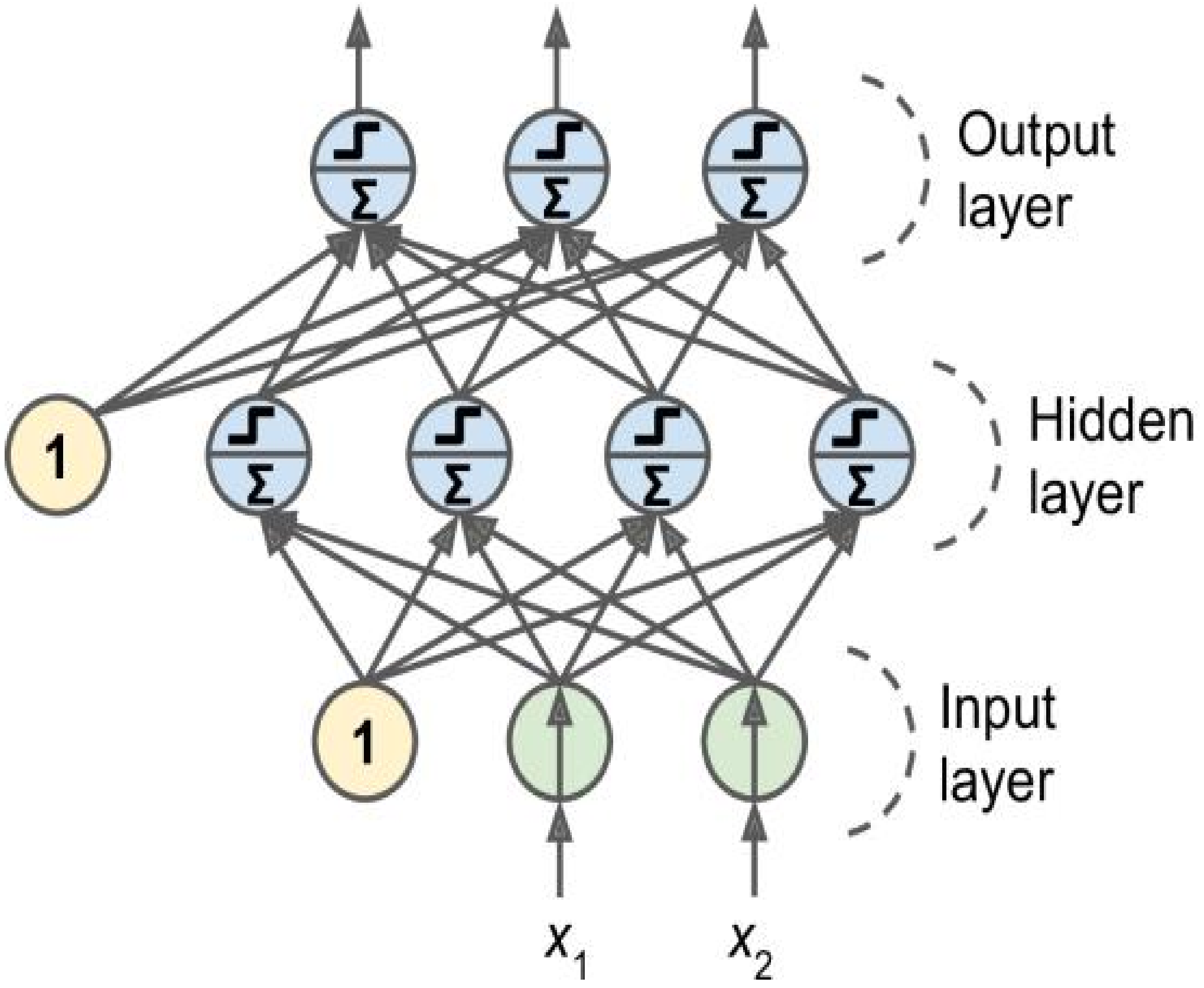




The Multilayer Perceptron MLP

- The signal flows only in one direction (from the inputs to the outputs), so this architecture is an example of a **feedforward neural network (FNN)**.

- When an ANN contains a deep stack of hidden layers, it is called a **deep neural network (DNN)**.



The Multilayer Perceptron MLP

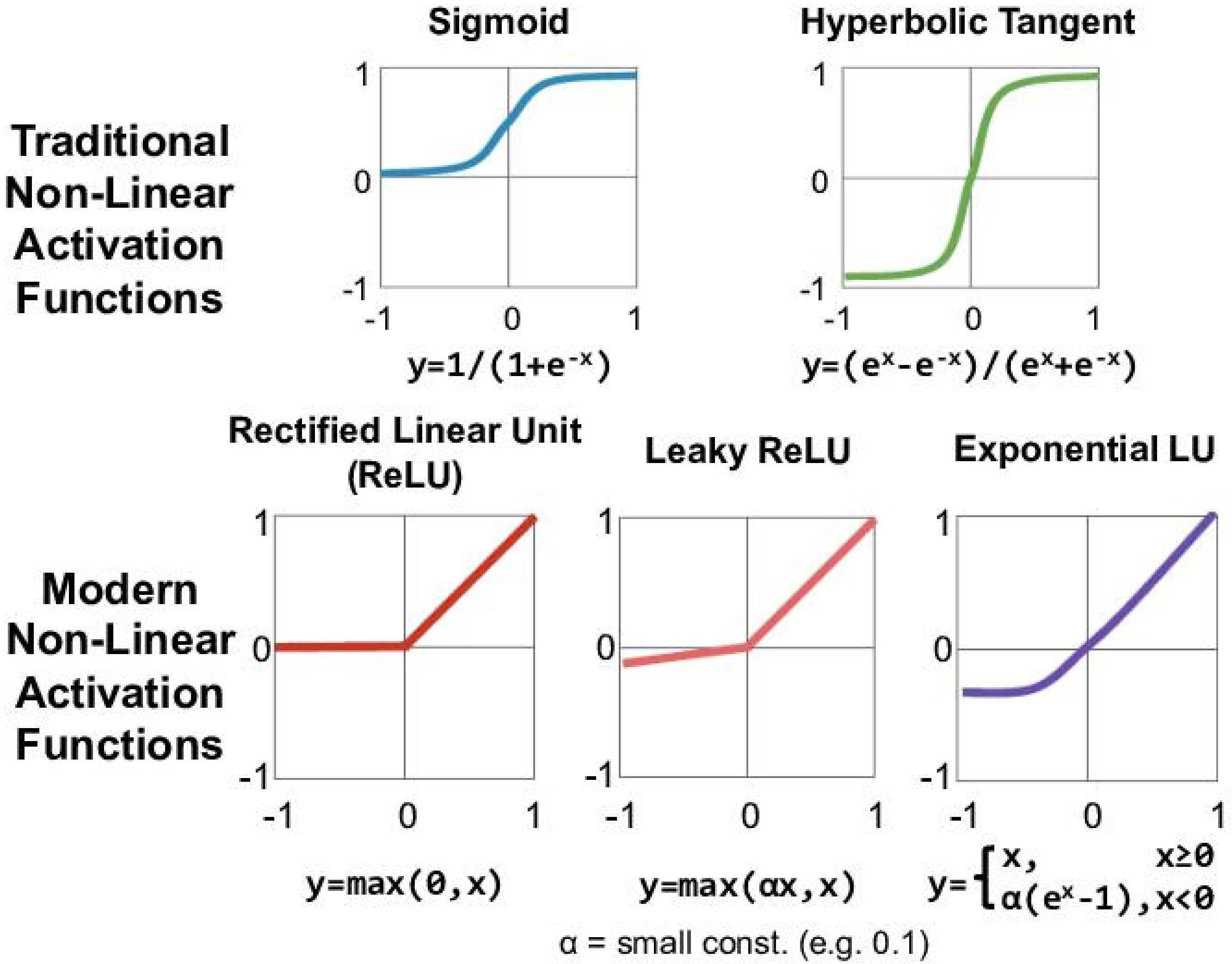
The Multilayer Perceptron MLP

- In 1986, the **backpropagation** training algorithm was introduced, which is still used today.

- **The backpropagation** consists of only two passes through the network **(one forward, one backward)**, the backpropagation algorithm is able to compute the gradient of the **network's error** with regard to every single model parameter. In other words, it can find out how each **connection weight** and each **bias** term should be tweaked in order to reduce the error.

David Rumelhart et al. "Learning Internal Representations by Error Propagation," (Defense Technical Information Center technical report, September 1985).

Popular Activation functions for MLP



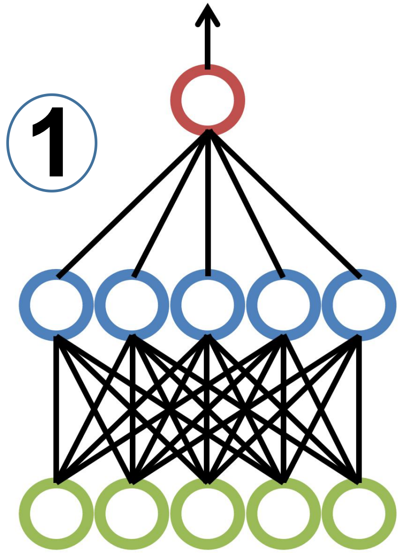
Neural Network vocabulary

1) Cost Function
2) Gradient Descent
3) Learning Rate
4) Backpropagation
5) Batches
6) Epochs

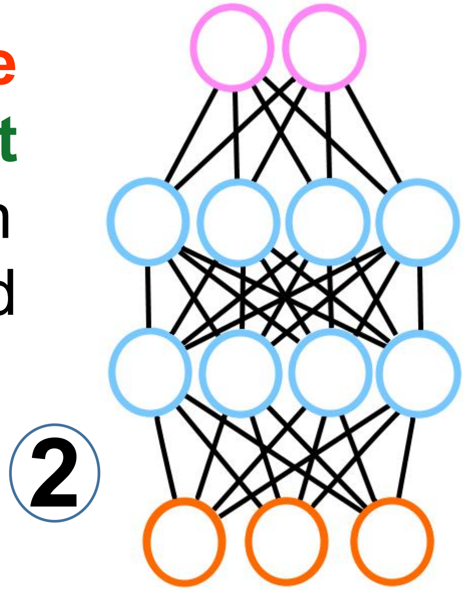
Regression MLPs

Regression MLPs

1) If you want to **predict a single value** (e.g., the price of a house, given many of its features), then you just need a **single output neuron**, its output is the predicted value.



2) For **multivariate regression** (i.e., **to predict multiple values at once**), you need **one output neuron per output dimension**. For example, to locate the center of an object in an image, you need to predict 2D coordinates, so you need two output neurons.



Regression MLPs

Typical regression MLP architecture:

Hyperparameter	Typical value
input neurons	One per input feature
hidden layers	Depends on the problem, but typically 1 to 5
neurons per hidden layer	Depends on the problem, but typically 10 to 100
output neurons	1 per prediction dimension
Hidden activation	ReLU (relu)
Output activation	None, or ReLU/softplus (if positive outputs) or logistic/tanh (if bounded outputs)
Loss function	MSE (mean_squared_error) or MAE/Huber (if outliers)

Classification MLPs

Classification MLPs

Binary classification:

- We just need a **single output neuron** using the logistic activation function **0 or 1**.

Multilabel Binary Classification:

- We need **one output neuron for each positive class**.

For example: you could have an email classification system that predicts whether each incoming email is **ham or spam** and simultaneously predicts whether it is an **urgent or nonurgent** email. In this case, you would need **two output neurons**, both using the logistic activation function.

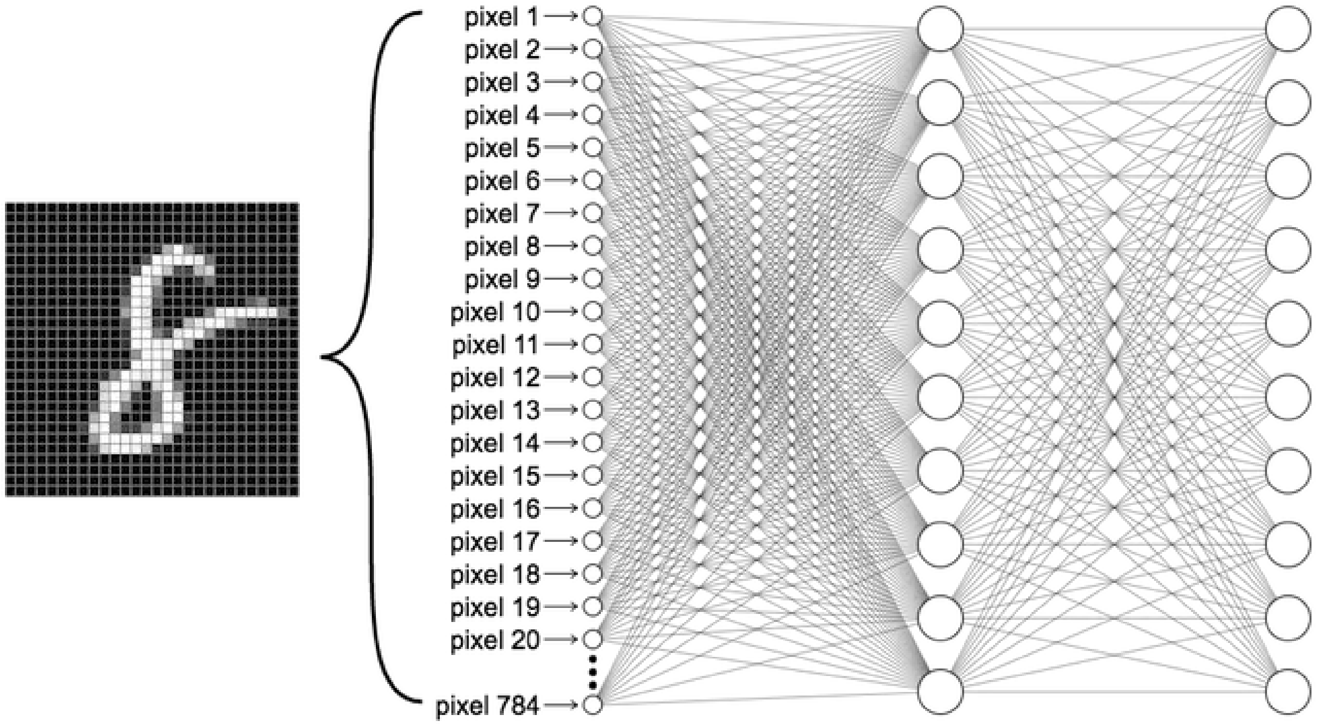
Classification MLPs

Multiclass Classification:

We need to have **one output neuron per class**, and we should use the softmax activation function for the whole output layer.

For example:

classes **0 through 9** for digit image classification [28, 28].



Classification MLPs

Typical classification MLP architecture

Hyperparameter	Binary classification	Multilabel binary classification	Multiclass classification
input neurons	One per input feature	One per input feature	One per input feature
hidden layers neurons per hidden layer	Depends on the problem	Depends on the problem	Depends on the problem
output neurons	1	1 per label	1 per class
Hidden activation	ReLU (relu)	ReLU (relu)	ReLU (relu)
Output layer activation	Logistic (sigmoid)	Logistic (sigmoid)	Softmax (softmax)
Loss function	Cross entropy	Cross entropy	Cross entropy

Binary classification : categorical_crossentropy
Multiclass classification : sparse_categorical_crossentropy

Classification MLPs

If the training set was very skewed, with some classes being **overrepresented and others underrepresented**, it would be useful to set the **class_weight** argument when calling the fit().

sample_weight: Per-instance weights could be useful if some instances were labeled by experts while others were labeled using a crowdsourcing platform: you might want to give more weight to the former.

Classification MLPs

If you are not satisfied with the performance of your model, you should go back and tune the hyperparameters.

- Try another **optimizer**. **Gradient Descent**, **Momentum Optimization**, **Nesterov Accelerated Gradient**, **AdaGrad**, **RMSProp**, **Adam**, **Nadam**
- Try tuning model hyperparameters such as **the number of layers**, **the number of neurons per layer**, and the **types of activation functions** to use for each hidden layer.
- Try tuning other hyperparameters, such as **the number of epochs** and t**he batch size**.

- *Once you are satisfied with your model's validation accuracy, you should evaluate it on the test set to estimate the generalization error before you deploy the model to production.*

How to Save and Load Your Model

Keras use the **HDF5 format** to save both the **model's architecture (including every layer's hyperparameters)** and the values of all **the model parameters** for every layer **(e.g., connection weights and biases)**. It also saves **the optimizer** (including its hyperparameters and any state it may have).

```
model.save("my_keras_model.h5")
```

Loading the model:

```
model = keras.models.load_model("my_keras_model.h5")
```

How to increase your small image dataset

```
trainAug = ImageDataGenerator( rotation_range=40, width_shift_range=0.2,
            height_shift_range = 0.2, shear_range=0.2, zoom_range=0.2, horizontal_flip=True,
            fill_mode='nearest')

model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
H = model.fit_generator( trainAug.flow(trainX, trainY, batch_size=BS),
steps_per_epoch=len(trainX) // BS,validation_data=(testX, testY), validation_steps=len(testX)
// BS, epochs=EPOCHS)
```

Training a Deep DNN

Training a very large deep neural network can be painfully slow. Here are some ways to speed up training (and reach a better solution):

- Applying a good initialization strategy for the connection **weights**.
- Using a good **activation function**.
- Using **Batch Normalization**.
- **Reusing parts** of a pretrained network (possibly built on an auxiliary task or using unsupervised learning).
- Using **faster optimizer**.