Neural Networks for Supervised Learning

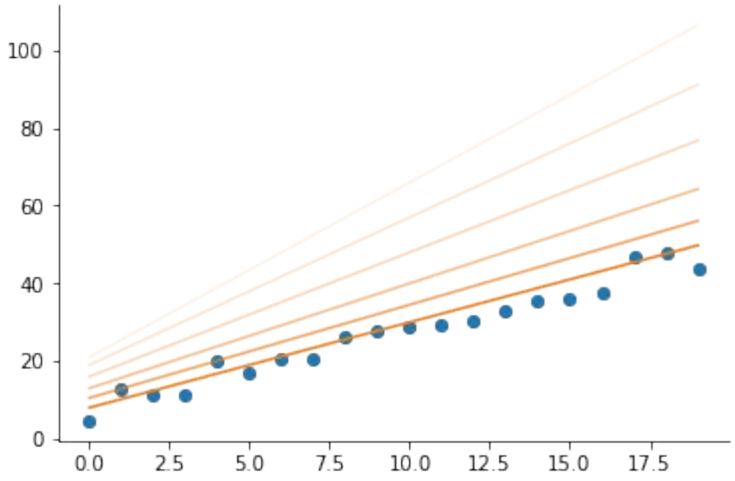
Supervised Learning task

The task is essentially to learn the association between the training examples ***X*** and the labels ***y***.

$$X \rightarrow y$$

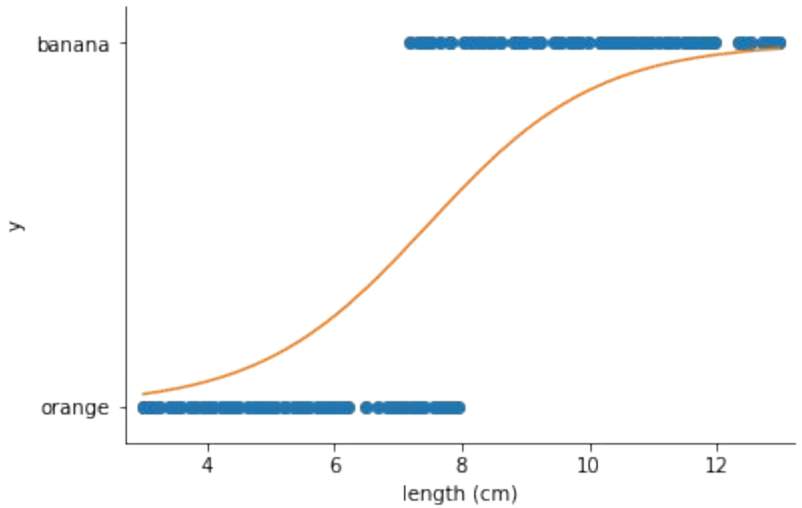
Linear Regression

1. Random initialization W, b.
2. Generate line $\hat{y} = Wx + b$.
3. Calculate total cost J.
4. Compute gradients $\frac{dJ}{dW}$ and $\frac{dJ}{db}$.
5. Update W and b.
6. Repeat steps 2 – 5 until cost stops dropping.



Logistic Regression

- Linear classifier
- Use of activation function (sigmoid) to output "probability"
- Loss function, again, checks distance from predicted probability to actual value
- Other than that training occurs same as in linear regression



Case study 1: linear classifier

Let's now see how well a logistic regression classifier works on two different datasets:

- <u>One that is linearly separable</u>

- <u>One that is not</u>

Logistic Regression to Neural Networks

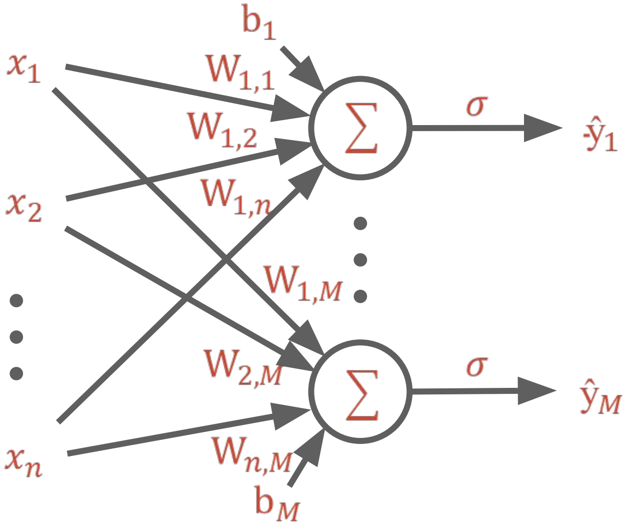
- Simple logistic regression function:

$$\hat{y} = \sigma(Wx + b)$$

- Or in the case of multiple input features or output classes:

$$\hat{Y}_i = f\left(\sum_{j=1}^{M}(w_{i,j}x_i + b_i)\right)$$

- We can imagine the above case as multiple simple regressors:

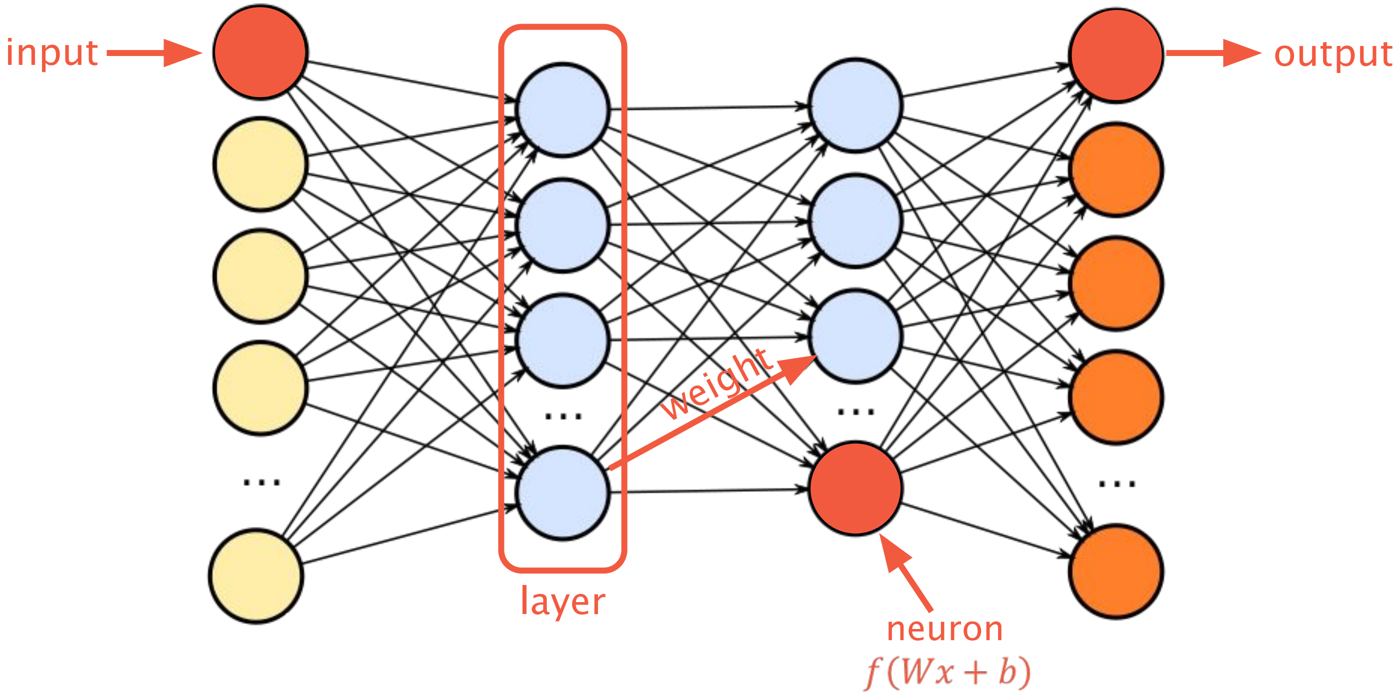


- This is essentially a **neural network**.

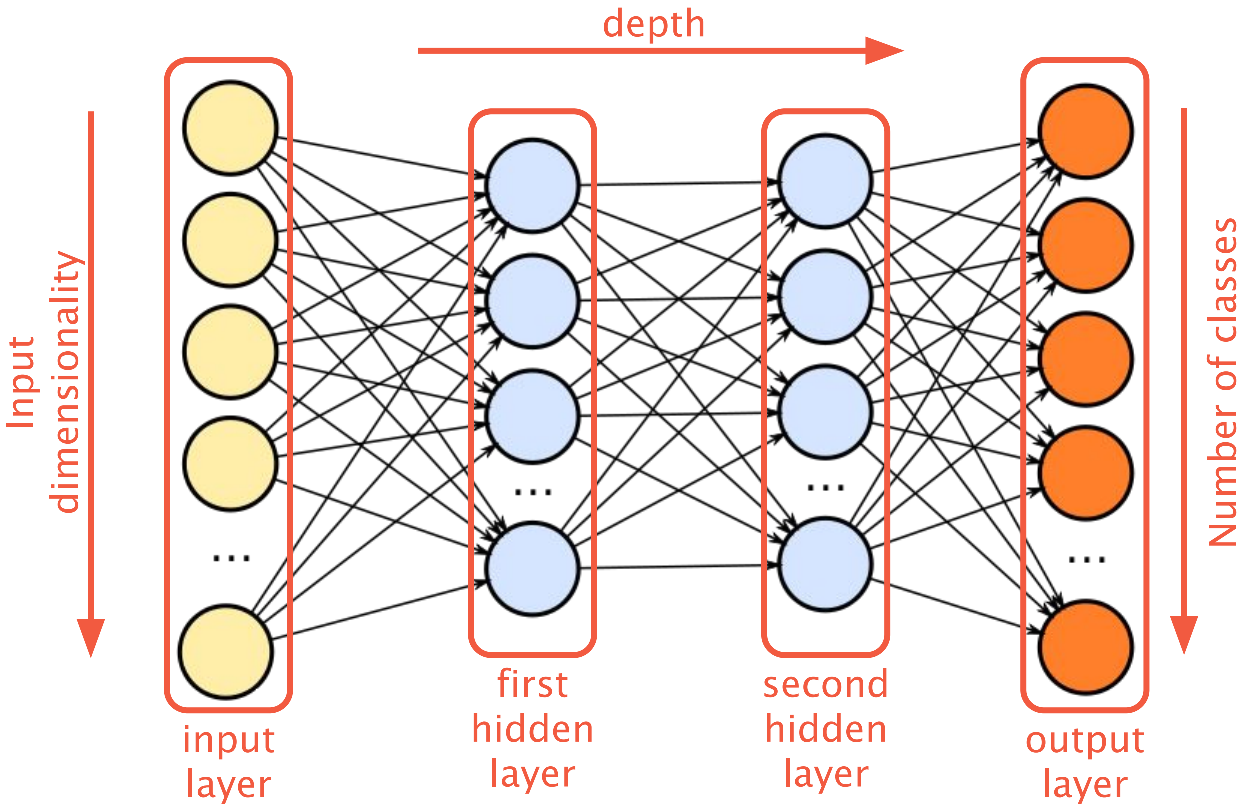
Neural Networks – terminology (1)

- Each logistic regression unit ($\hat{y} = \sigma(WX + b)$) is called a **neuron**.
- The matrices W and b are called **weights** and **biases**.
- Neurons sharing the same inputs X are called a **layer**.
- Using this definition, logistic regression is essentially a neural network with one layer.
- By adding a second layer, we give the network the capacity of modelling **non-linear problems**.
- The layer whose neurons constitute the network's output is called the **output layer**.
- The layers whose outputs are fed into another layer are referred to as **hidden layers**.

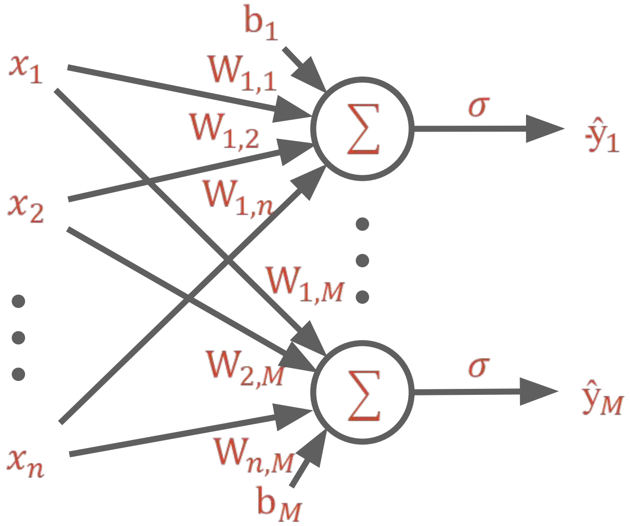
Neural Networks – terminology (2)



Neural Networks – terminology (3)



Inside each layer…



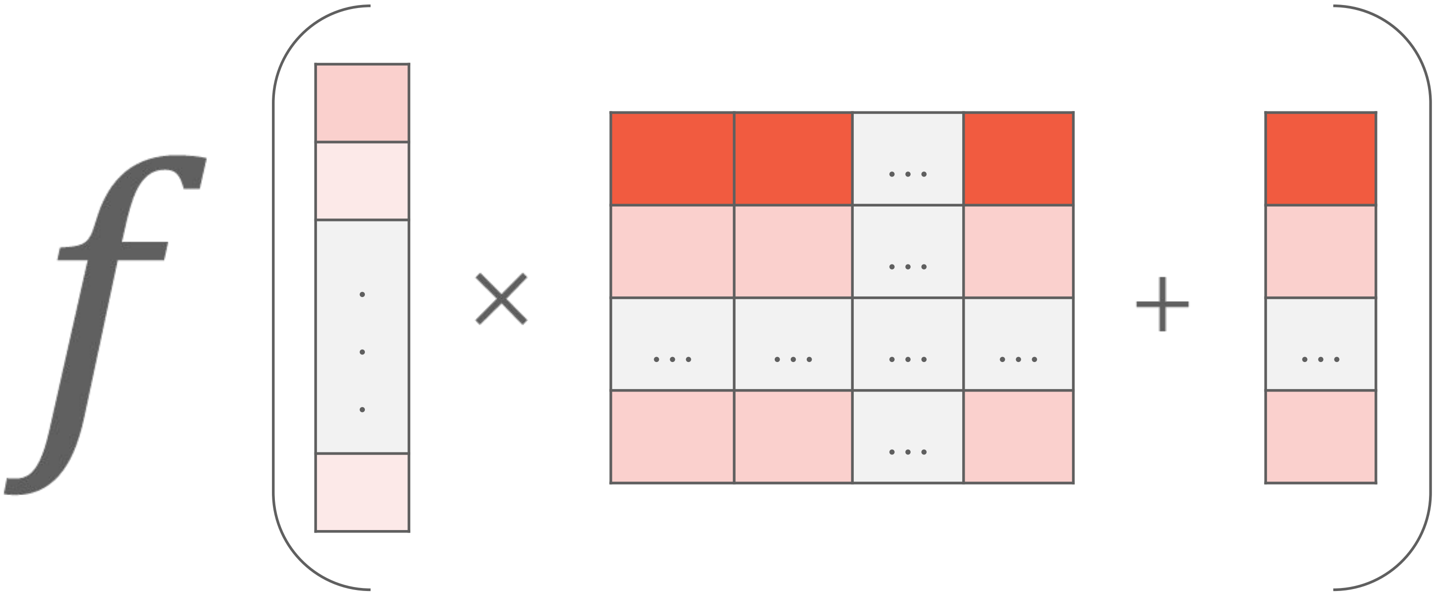
In matrix format

$$\hat{y} = f(XW + b)$$

If we have n input features and M neurons:

- Weight matrix has $M \times n$ values.
- Bias matrix has $M \times 1$ values.

Inside each layer…



$$\hat{y} = f(xW + b)$$

Activation functions

What exactly is f in $\hat{y} = f(xW + b)$?

- f is a non-linear function called an **activation function**

The most popular functions are:

- sigmoid (what is used in logistic regression):

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

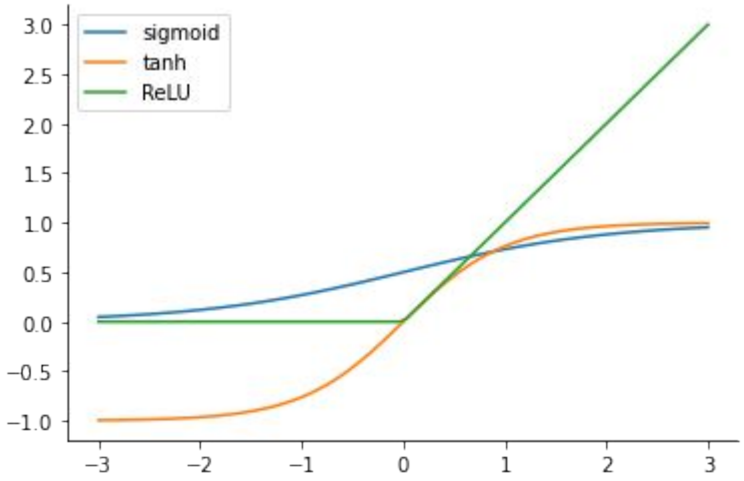
- tanh

$$\tanh(\mathrm{z}) = \frac{e^{2z} - 1}{e^{2z} + 1}$$

- ReLU

$$ReLU(\mathrm{z}) = \max(0, z)$$

Activation functions

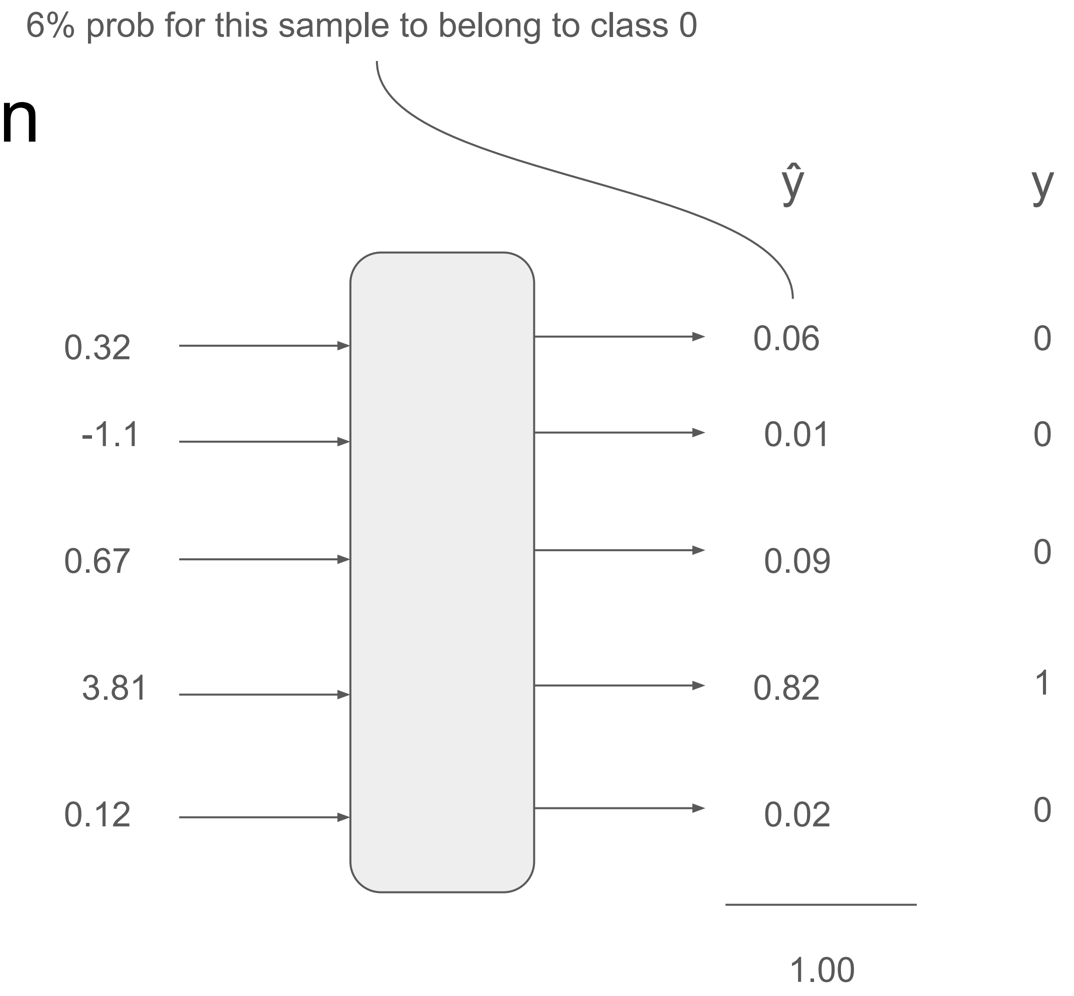


Output

- **Regression** (single value):
  - One output neuron
  - No activation function for the final layer
  - The output will be a single real number
- **Classification**:
  - Binary:
    - One output neuron
    - Sigmoid or softmax activation
    - The output will be a real number in [0, 1] (probability of belonging to class 1)
  - Multiclass:
    - Output neurons as many as classes
    - Softmax activation
    - The output will be a vector with one real number per class. All numbers are in [0, 1] and must sum to 1.

Softmax activation

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$



Case study 2: MLP

Let’s now see how well a Neural Network with one hidden layer (also referred to as a Multi-Layer Perceptron) performs on the same two datasets:

- Linearly separable dataset

- Not linearly separable dataset

Training: cost function

- The cost function is essentially a distance metric showing *how close* our predictions are to the labels.
- Cost functions **always** compare continuous values (even for classification)

- Regression:

  - Most commonly Mean Squared Error

$$J(y, \hat{y}) = \sum (y - \hat{y})^2$$

- Classification:

  - Most commonly cross-entropy

$$J(y, \hat{y}) = -\sum y \log \hat{y}$$

Softmax activation

$\hat{y}$	y
0.06	0
0.01	0
0.09	0
0.82	1
0.02	0

$\Sigma\, y \log\hat{y} =$

$0 * \log 0.06 + 0 * \log 0.01 + \ldots + 1 * \log 0.82$

$= \ldots$

Training: backpropagation

- The cost function shows us how good or bad our model is doing.
- We need a way of knowing how much each parameter of the network contributed to the total loss:

$$\frac{\partial J}{\partial w} \; \forall \, w \in \theta$$

- **Backpropagation**: an algorithm that computes the partial derivatives of each parameter in the network efficiently
- It does this by starting from the output layer and moving backwards.
- The vector containing all partial derivatives is called the **gradient** and it is key to training the network:

$$\nabla J = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \dots\right]$$

Training: optimization

- Besides the contribution of each parameter to the total error, the gradient shows us the **direction** we need to move each parameter to **decrease** the error.

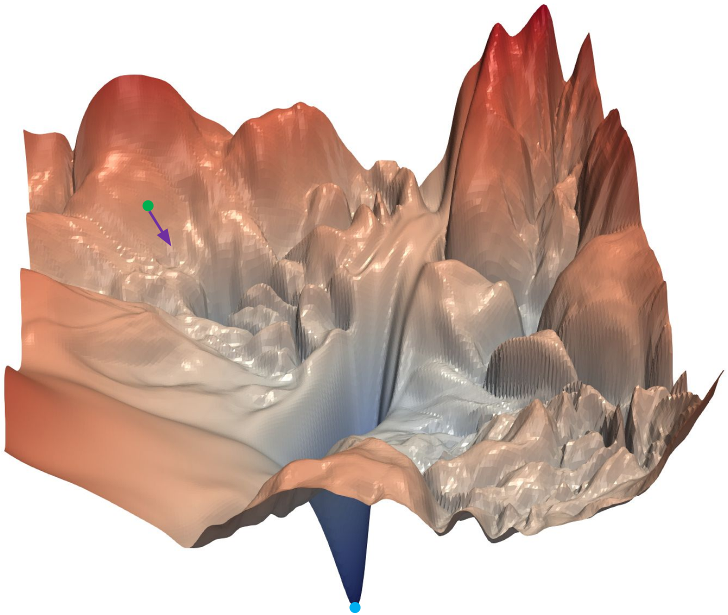
- The optimizer is in charge of updating the parameters.

- Different optimizers have different strategies in updating their parameters.

- Ideally we'd want an optimizer that besides being good (i.e. **reducing the loss**), also **converges fast**.

Training: intuition and challenges

- Remember the loss landscape of linear regression?
- Here things are a bit more complicated…
- Consider this 2D representation of a Neural Network loss landscape:
- We start at one random point (random weight initialization) and want to reach the global minimum.
- The gradient shows us the slope of the surface at our current position.
- Problem: we can get stuck in **local minima**, **ridges** and **plateaus**!
- Even worse, the loss depicts performance on the training set, which does not always correlate with performance on the test set.



Case study 3: Even harder problems…

Let’s now see how well a Neural Network with one hidden layer (also referred to as a Multi-Layer Perceptron) performs on the same two datasets:

- Circles
- Spiral