Deep learning with R

Dr. Iegor Rudnytskyi
<iegor.rudnytskyi@gmail.com>

R-lunches
"Remote" Geneva
as.Date("2020-12-01")

I, Robot (2004)

Spooner: So, Dr. Calvin, what exactly do you do around here?

Dr. Calvin: My general fields are advanced robotics and psychiatry. Although, I specialize in hardware-to-wetware interfaces [...]

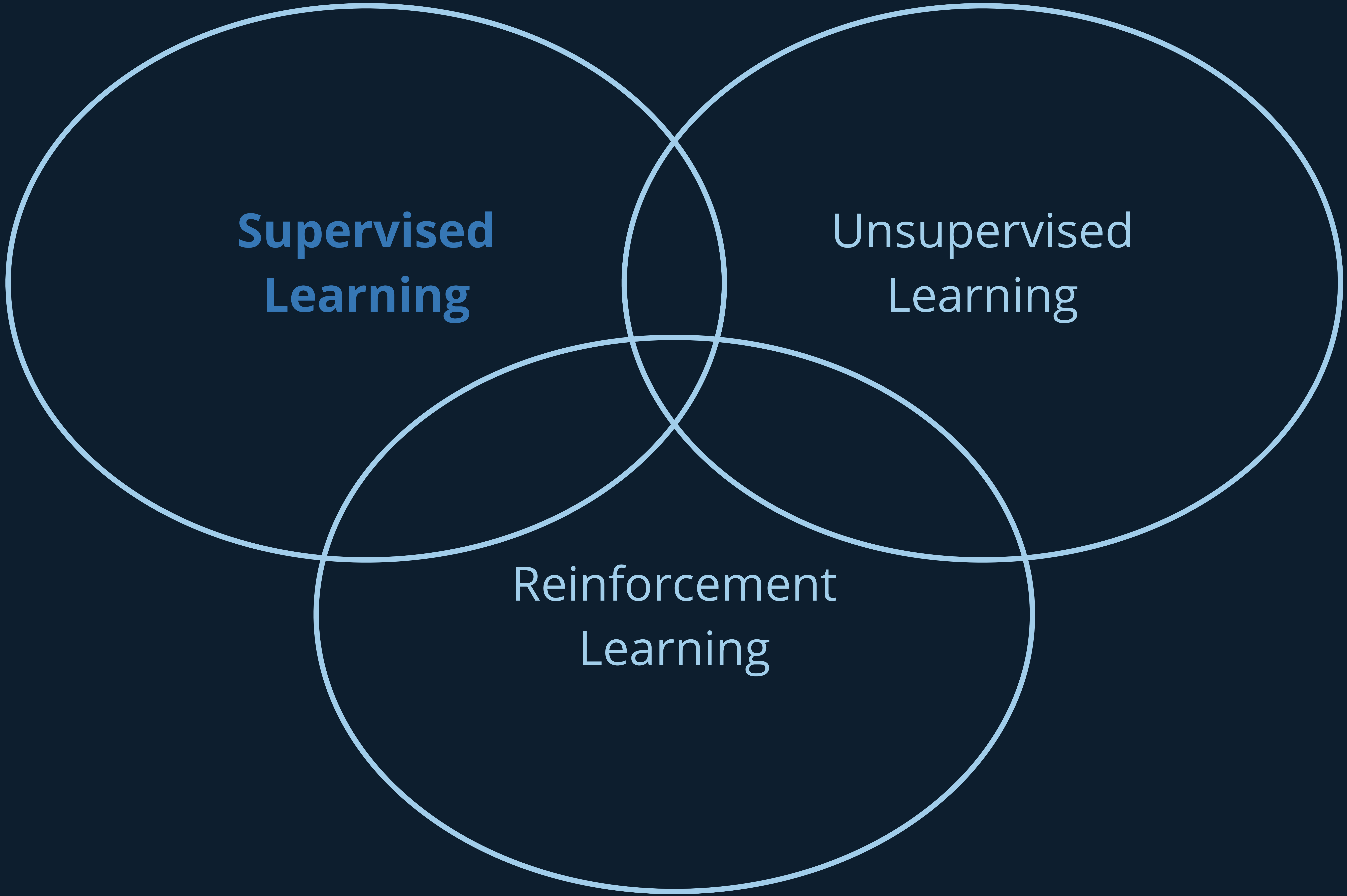
Spooner: So, what exactly do you do around here?

Dr. Calvin: I make the robots seem more human.

Spooner: Now wasn't that easier to say?

Dr. Calvin: Not really. No.

Machine Learning



Machine Learning

Deep learning is a machine learning method based on artificial neural networks (ANNs).

Major three branches of machine learning are:

- **supervised learning**: mapping features to known outcome, it also includes self-supervised learning
  - classification (categorical outcome)
  - regression (numerical outcome)
- unsupervised learning
- (semi-supervised learning)
- reinforcement learning

Supervised Learning

$$\begin{tikzcd}
\begin{array}{c}\text{Inputs/features}\\ \mathbf{x}\end{array} \arrow[r] & \boxed{\text{“Black box”/mapping}} \arrow[r] & \begin{array}{c}\text{Output/outcome}\\ y\end{array}
\end{tikzcd}$$

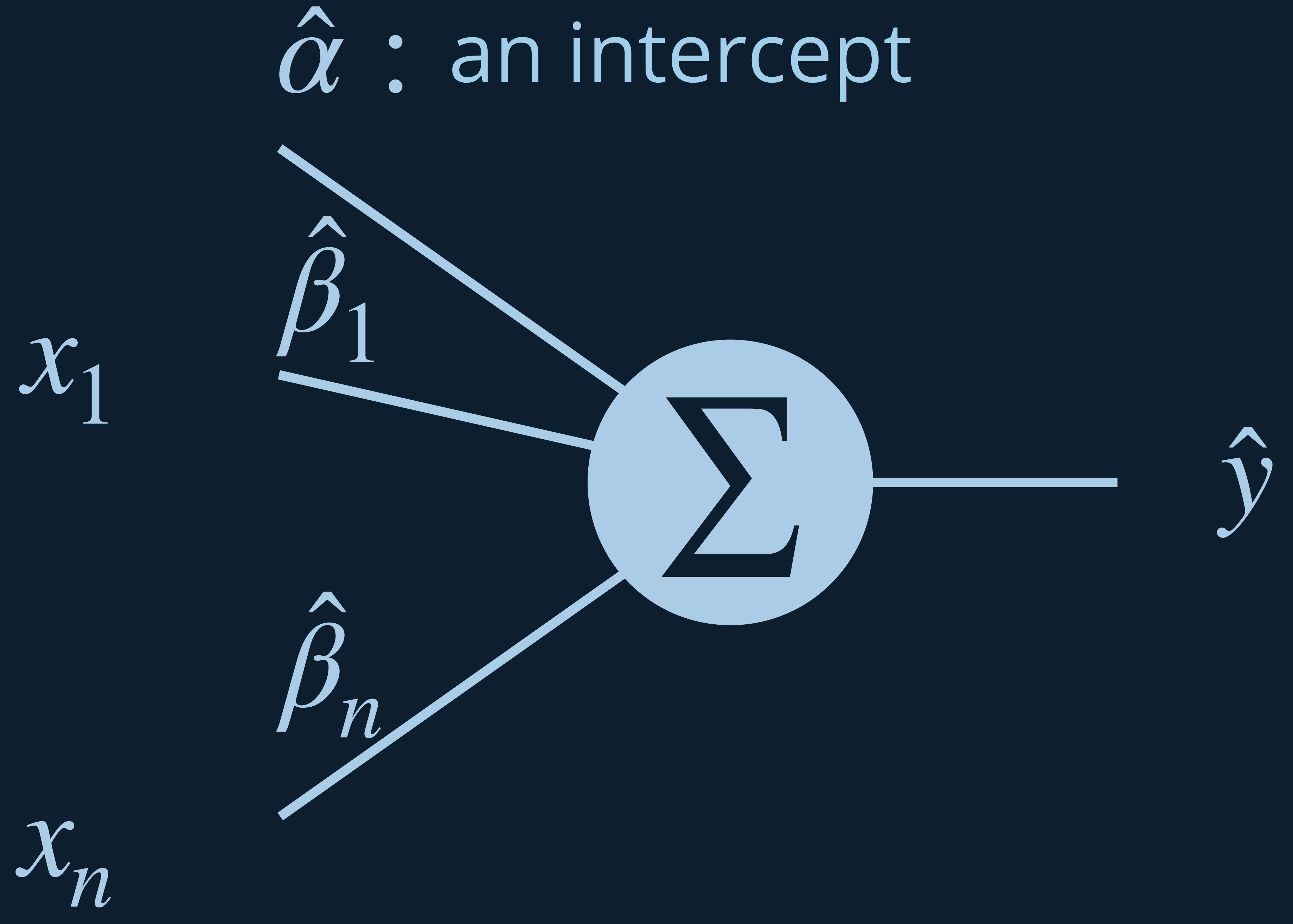
Supervised Learning

The task of supervised learning is to discover a function that maps an input to an output. Example: predicting the shoe size of a human by their height and weight.

Supervised learning is the most common case of machine learning from the business perspective. Almost all applications falls into this category, and therefore, we will focus on it.

```
> lm()
```

$$y = \alpha + \boldsymbol{\beta}^T \mathbf{x} + \epsilon = \alpha + \beta_1 x_1 + \ldots + \beta_n x_n + \epsilon$$



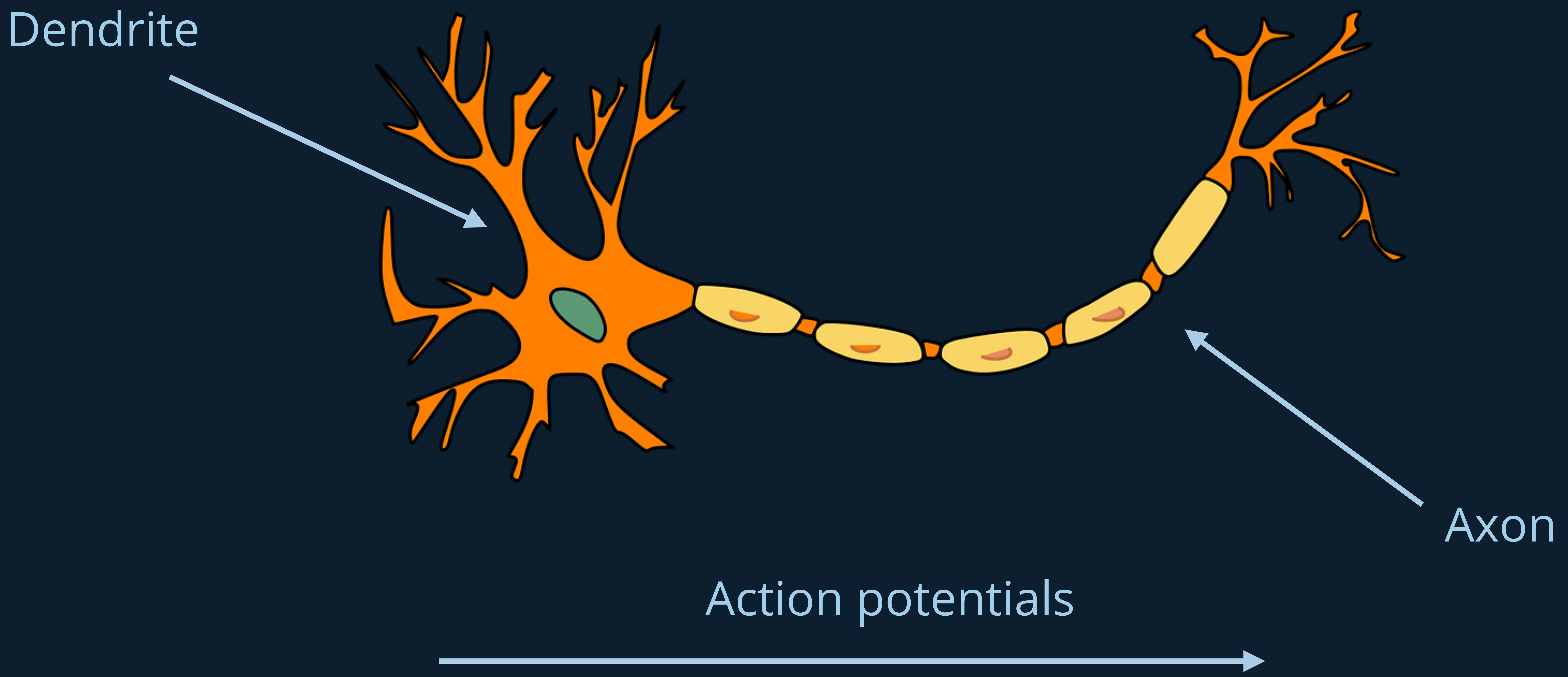
$$\hat{y} = \hat{\alpha} + \hat{\boldsymbol{\beta}}^T \mathbf{x} = \hat{\alpha} + \hat{\beta}_1 x_1 + \ldots + \hat{\beta}_n x_n$$

> lm()

The very first machine learning/statistical model that you have learned is linear regression. In the framework of this model, the outcome variable is computed as a linear combination of its features (sum of features times coefficients).

Despite its number of advantages, it has a crucial drawback — **linearity of the relationship between the outcome and features**. In the context of pattern recognition, this drawback outweighs all the strengths.

Biological neuron



Biological neuron

The biological neuron consists of branching extensions called **dendrites** and one very long extension called **axon**. The axon is connected to dendrites of other neurons.

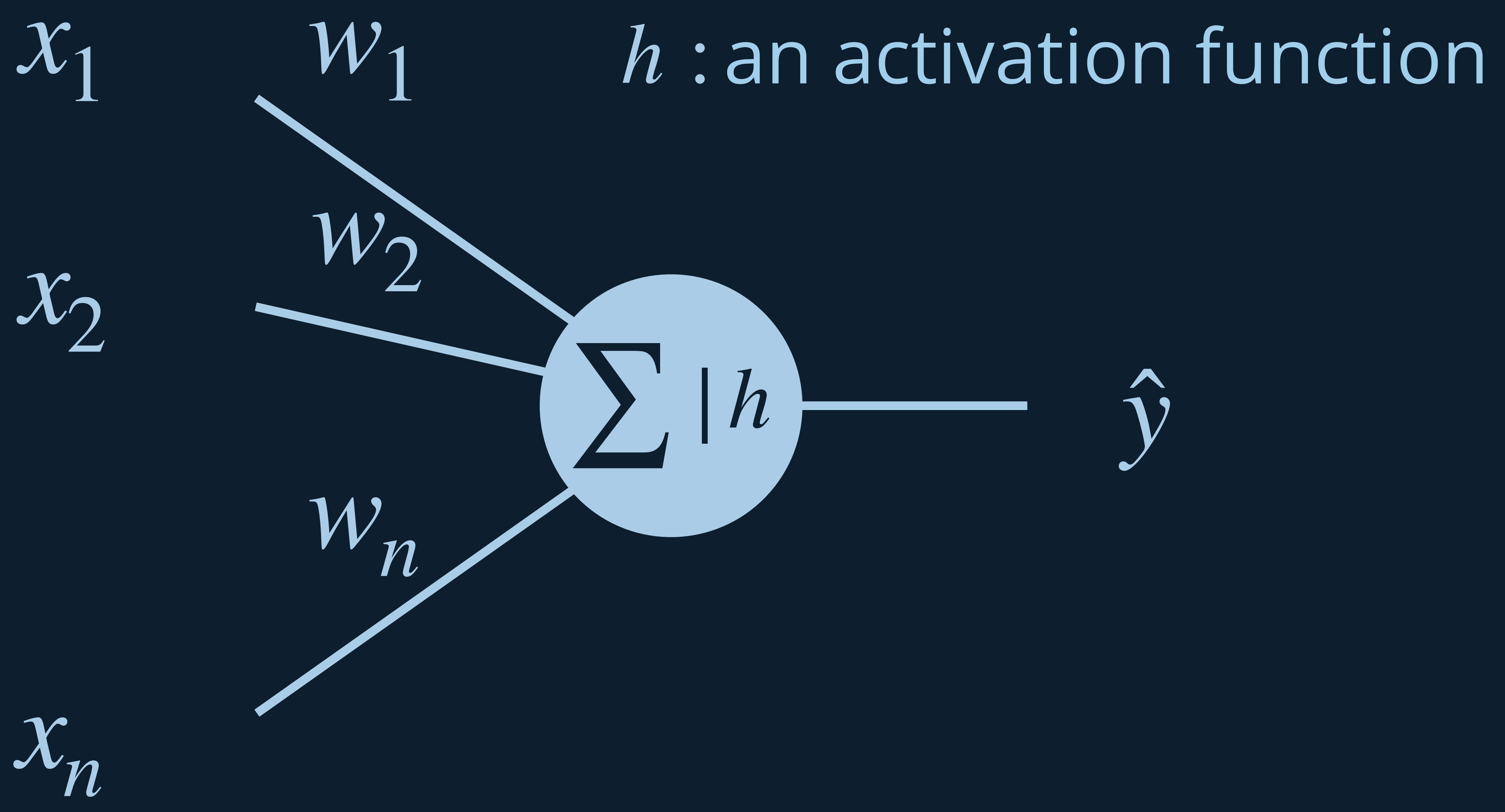
The neuron produce short electrical impulse (an action potential) and sends chemical signals to other neurons via axon. When the other neurons receive a sufficient number of these chemical signals, they fire their own electrical impulse and send chemical signals to the next neurons.

This idea inspired building an artificial neuron to create an intelligent machine.

From lm() to the perceptron

Threshold logic unit (TLU), an artificial neuron

$$\hat{y} = h(\mathbf{w}^T\mathbf{x}) = \begin{cases} 1 & if \quad \mathbf{w}^T\mathbf{x} \geq 0 \\ 0 & if \quad \mathbf{w}^T\mathbf{x} < 0 \end{cases}$$



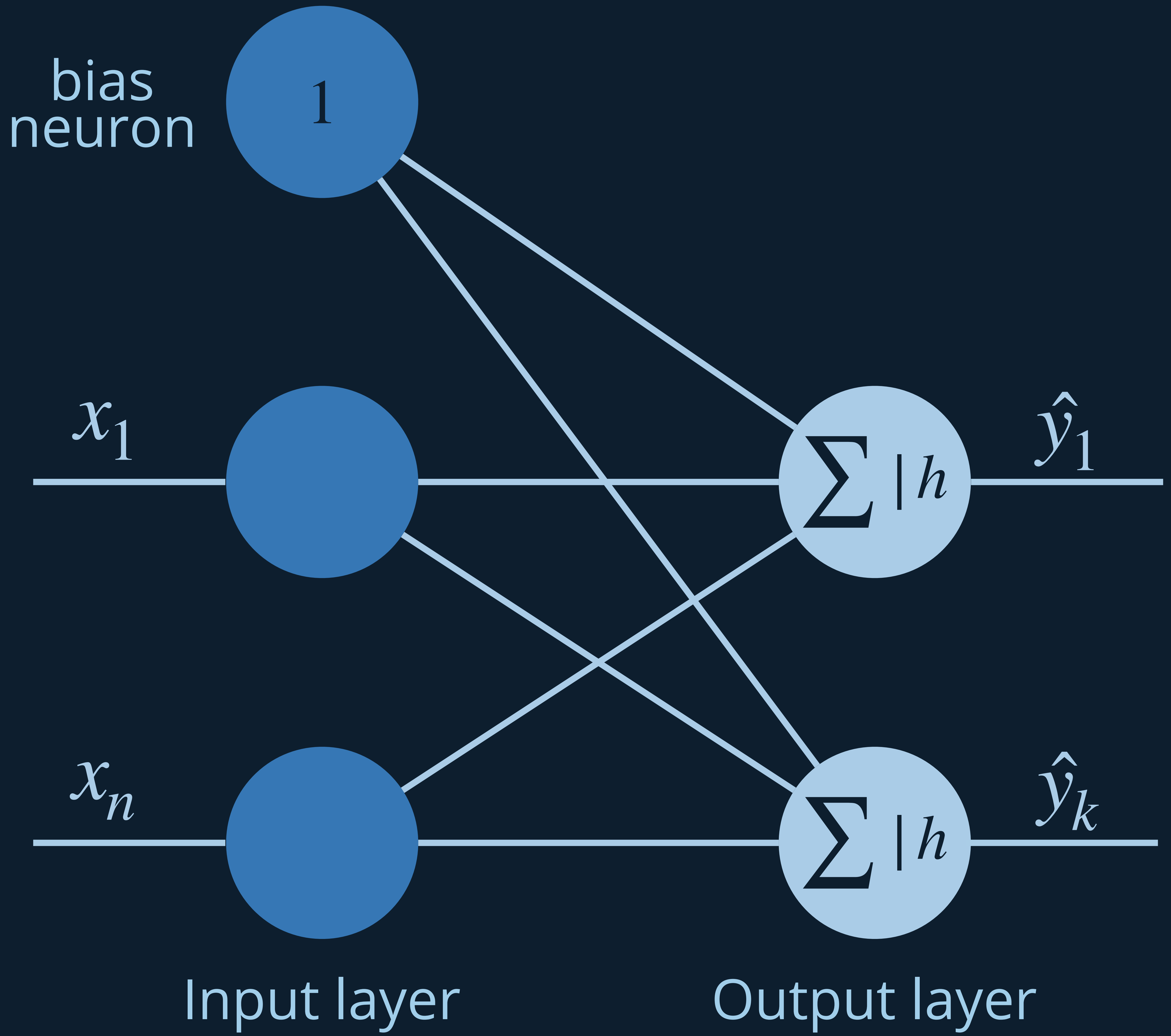
$$\mathbf{w}^T\mathbf{x} = w_1x_1 + w_2x_2 + \ldots + w_nx_n$$

From `lm()` to the perceptron

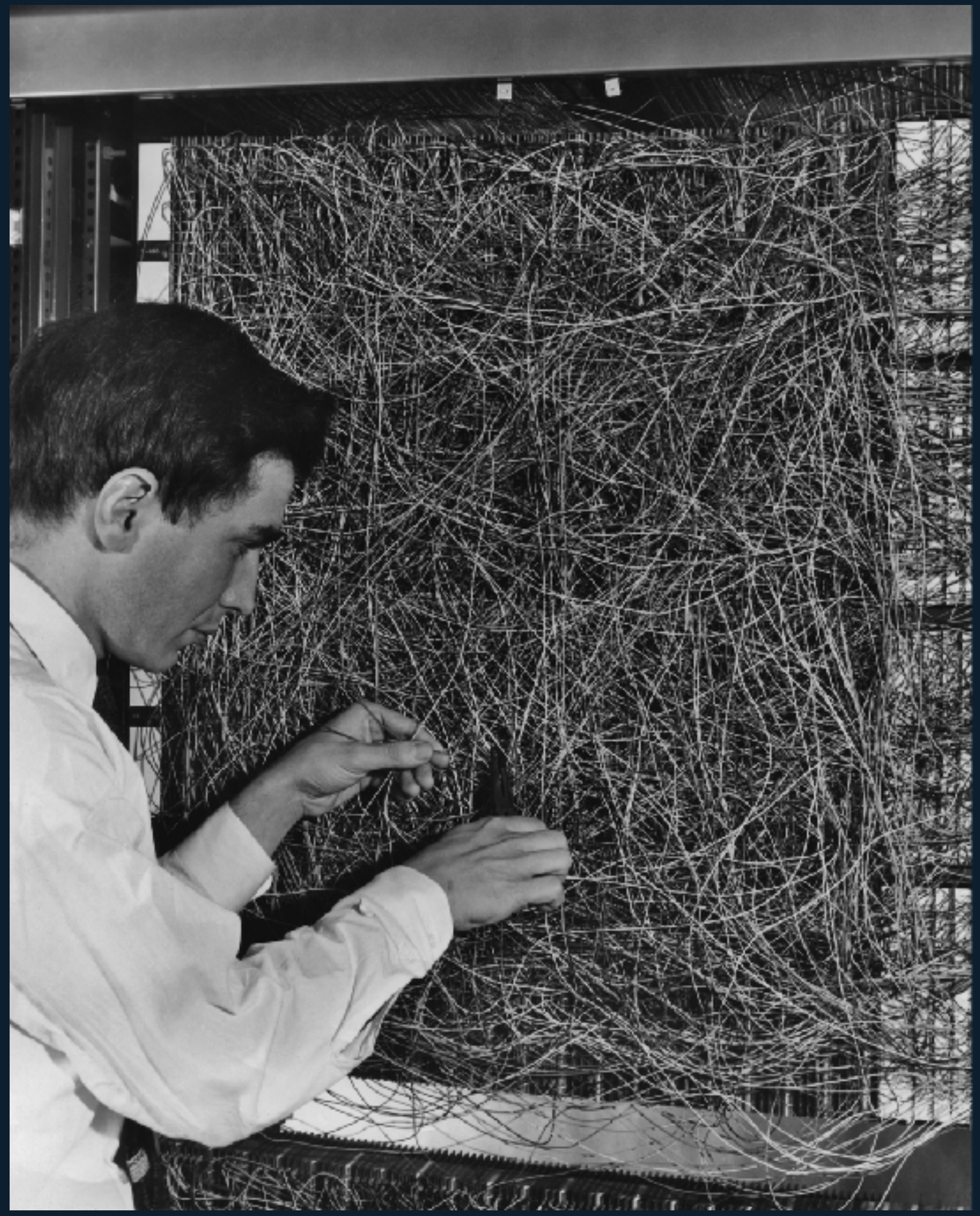
One of the first artificial neurons is TLU, or sometimes, a linear threshold unit (LTU). This type of neurons is similar to linear regression: it sums up its weighted inputs (again, we can set a first feature to 1, see equivalent formulation of `lm()`). However, it also applies so-called activation function (which is in the case of TLU is a step function) to the sum: if the sum is higher than a certain threshold, it returns 1, otherwise, 0.

Just like the biological neuron: it fires up (outputs 1 instead of 0) when it receives enough of chemical signals (the sum is higher than threshold).

From lm() to the perceptron



Mark I Perceptron: 20x20 image



Source: Ehre für die «Deep Learning Mafia», http://tiny.cc/cvdhhz

From lm() to the perceptron

The perceptron is simply a layer of TLUs with each TLU connected to all the inputs.

Input layer consists of **input neurons** (dark blue). They output whatever they are fed. A special case of input neurons is **the bias neuron** that outputs 1 all the time.

When all neurons in the layer are connected to all neurons in the previous layer, the layer is called a **fully connected layer**, or a **dense layer**.

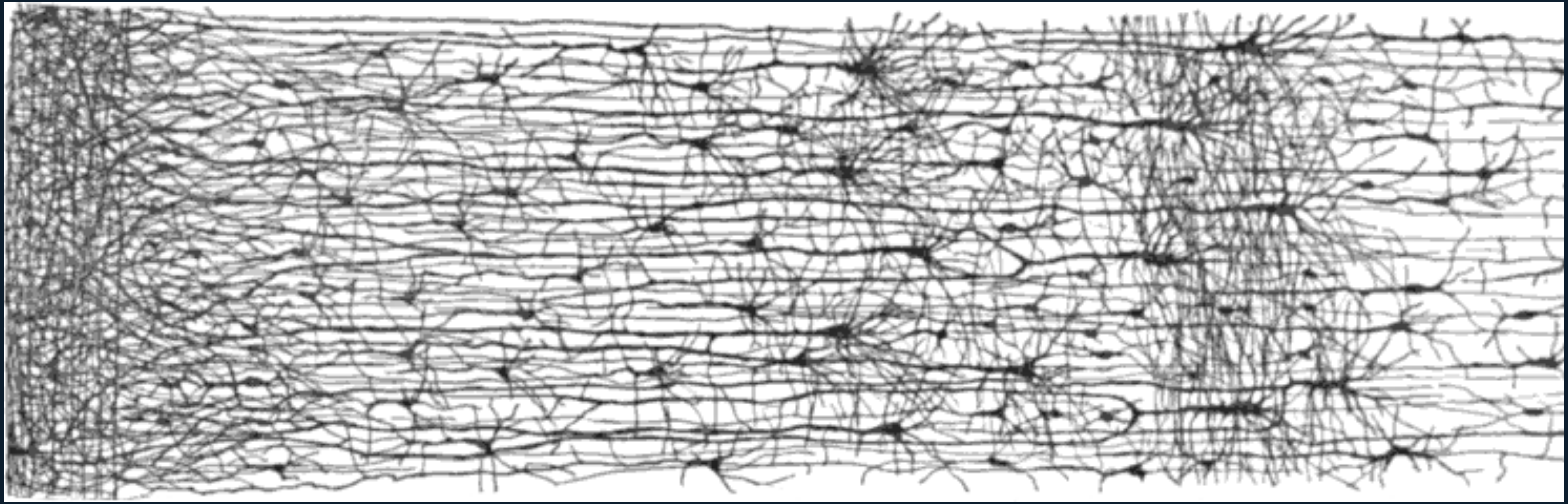
From lm() to the perceptron

A single TLU can be used as a binary classifier, while the perceptron is a multioutput classifier.

The perceptron is good enough for linearly separable patterns, however, the perceptron cannot solve even simplest problems, e.g., XOR.

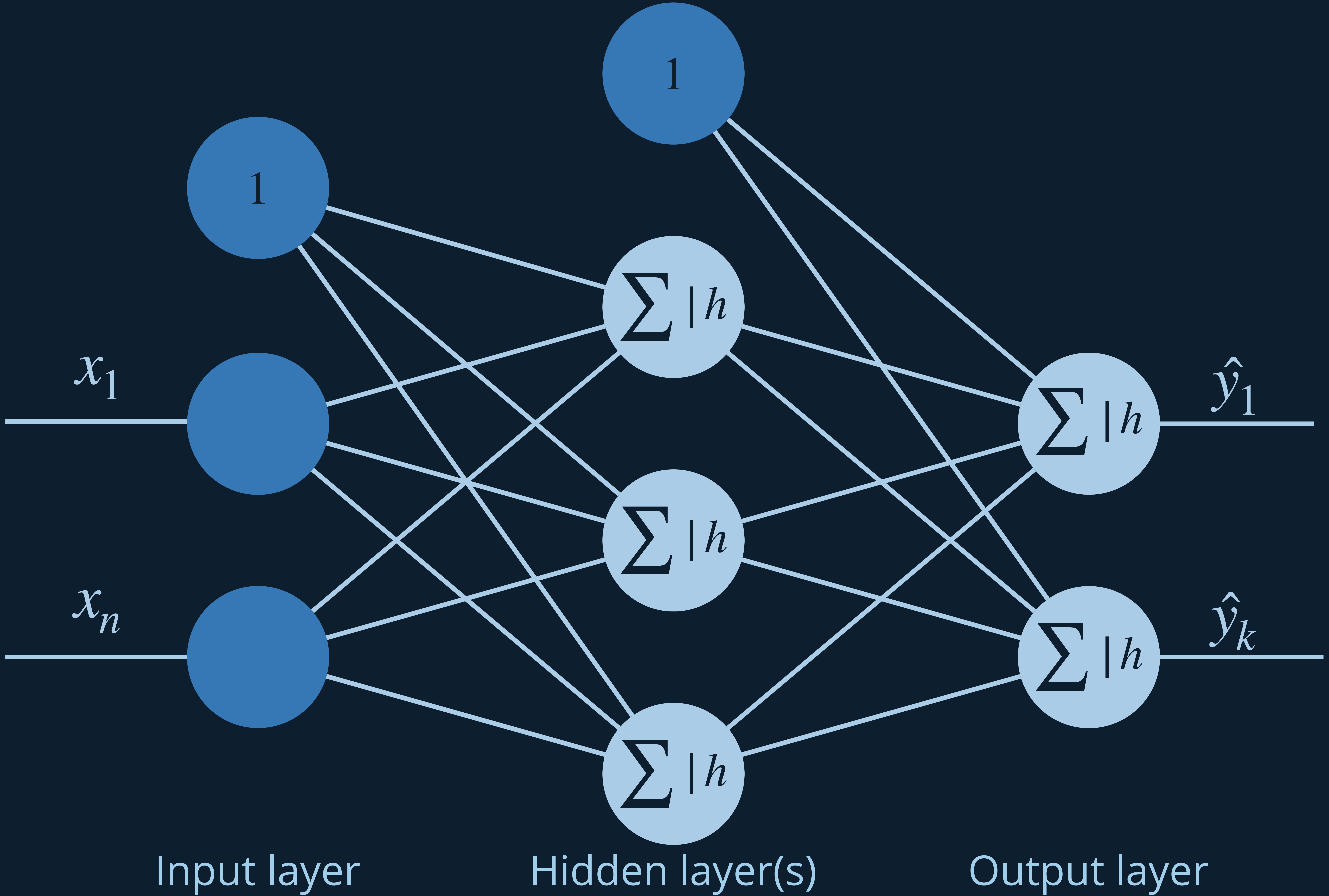
It turns out that this issue can be solved by stacking multiple perceptrons, just like in biological neural networks: neurons are often organized in consecutive layers.

Multiple layers in a biological neural network (BNN)



Source: Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, Second Edition. O'Reilly Media, Inc.

The multilayer perceptron



Why deep learning?

The presented model is called the multilayer perceptron (MLP).

The layers between the input and the output layers are called **hidden layers**. Every layer except the output layer includes a bias neuron (that always yields 1) and is fully connected to the next layer (dense layers).

When an ANN contains a deep stack of hidden layers (from two to even hundreds), it is called a **deep neural network** (DNN). The field of **Deep Learning** studies DNNs.

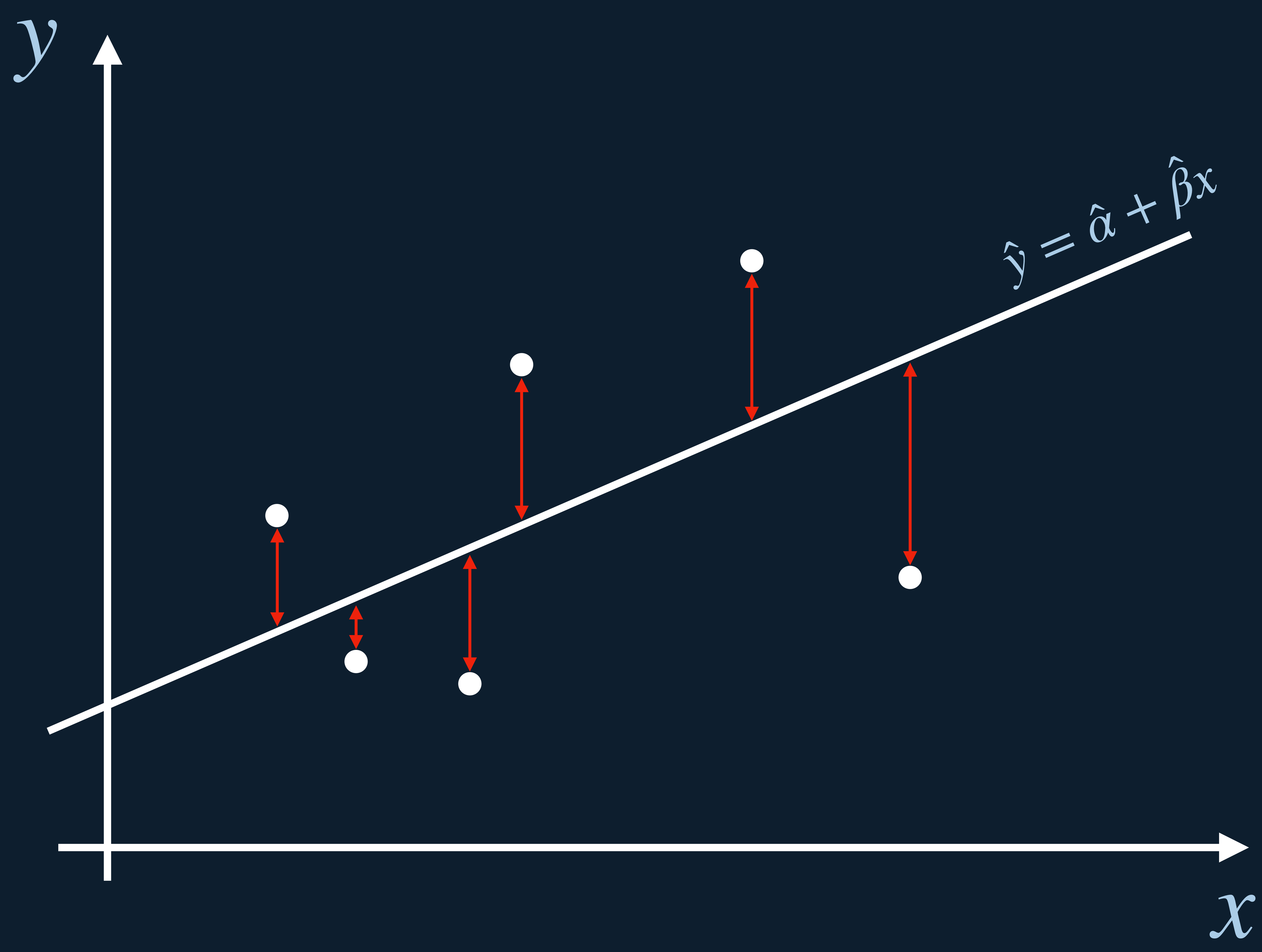
Training models

Recall, training (in statistical context is calibrating) a model means setting its parameters so that the model best fits the training data. For instance, to train a linear model means to find such parameters (coefficients), so that predicted values are “close” to the observed ones. To measure how “close” they are, we use various **cost functions** (sometimes, referred to as **loss functions**).

Therefore, the idea of training is to optimize the parameters with respect to cost function: finding such combination of parameters that yields the minimum value of the loss function.

Training models

Consider `lm()` and the mean squared errors MSE as the cost function.



$$MSE = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{\alpha} + \hat{\boldsymbol{\beta}}^T\mathbf{x}_i - y_i\right)^2$$

Training models

- Using closed-from analytical solution that minimizes the cost function (OLS, equivalent to MLE):

$$\hat{\beta} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

- Using iterative optimization approach, e.g., gradient descent

Training models

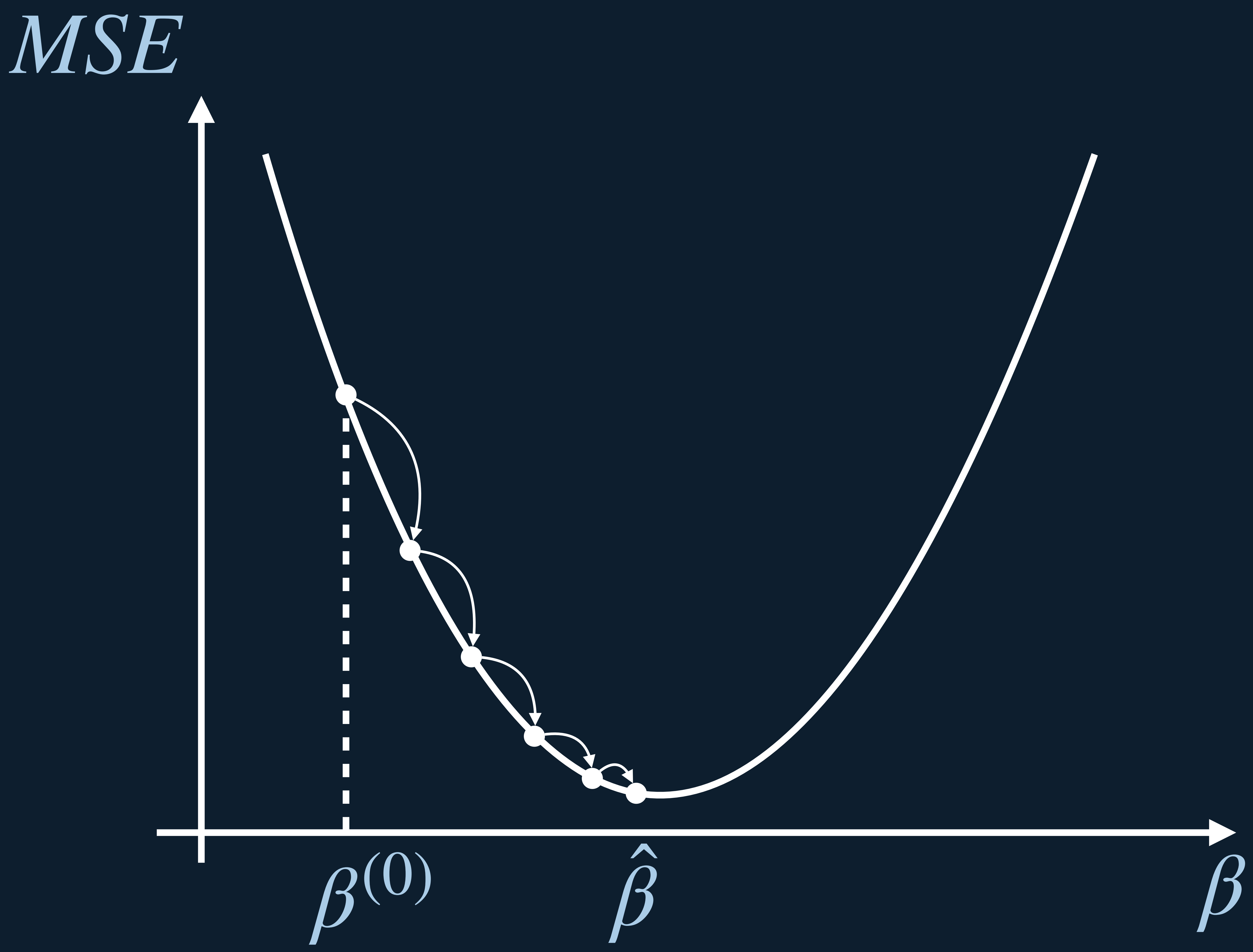
The closed-form analytical solution (i.e., the formula) is a very convenient and precise. However, (1) it does not always exist (i.e., impossible to derive), (2) is not always able to handle too many instances or too many features (matrices are too big to be allocated, it takes too much time to inverse a matrix).

To overcome this issue, we can use iterative methods. The idea of these methods is to tweak parameters iteratively in order to minimize the cost function.

Imagine you are on top of a mountain and trying to descent. There is a very dense fog and the only indicator is the direction of the slope under your feet. A good strategy to get to the bottom of the valley is to go downhill in the direction of the steepest slope step by step.

(Batch) Gradient descent

`lm()` with n = 1 (and without an intercept)



- Generate the random initial value $\beta^{(0)}$
- Perform the gradient descent step a certain number of times (the number of iterations)

$$\beta^{(k+1)} = \beta^{(k)} - \underbrace{\eta}_{\text{learning rate}} \cdot \underbrace{\frac{\partial}{\partial \beta} MSE \bigg|_{\beta=\beta^{(k)}}}_{\text{derivative of } MSE \text{ evaluated at } \beta^{(k)}}$$

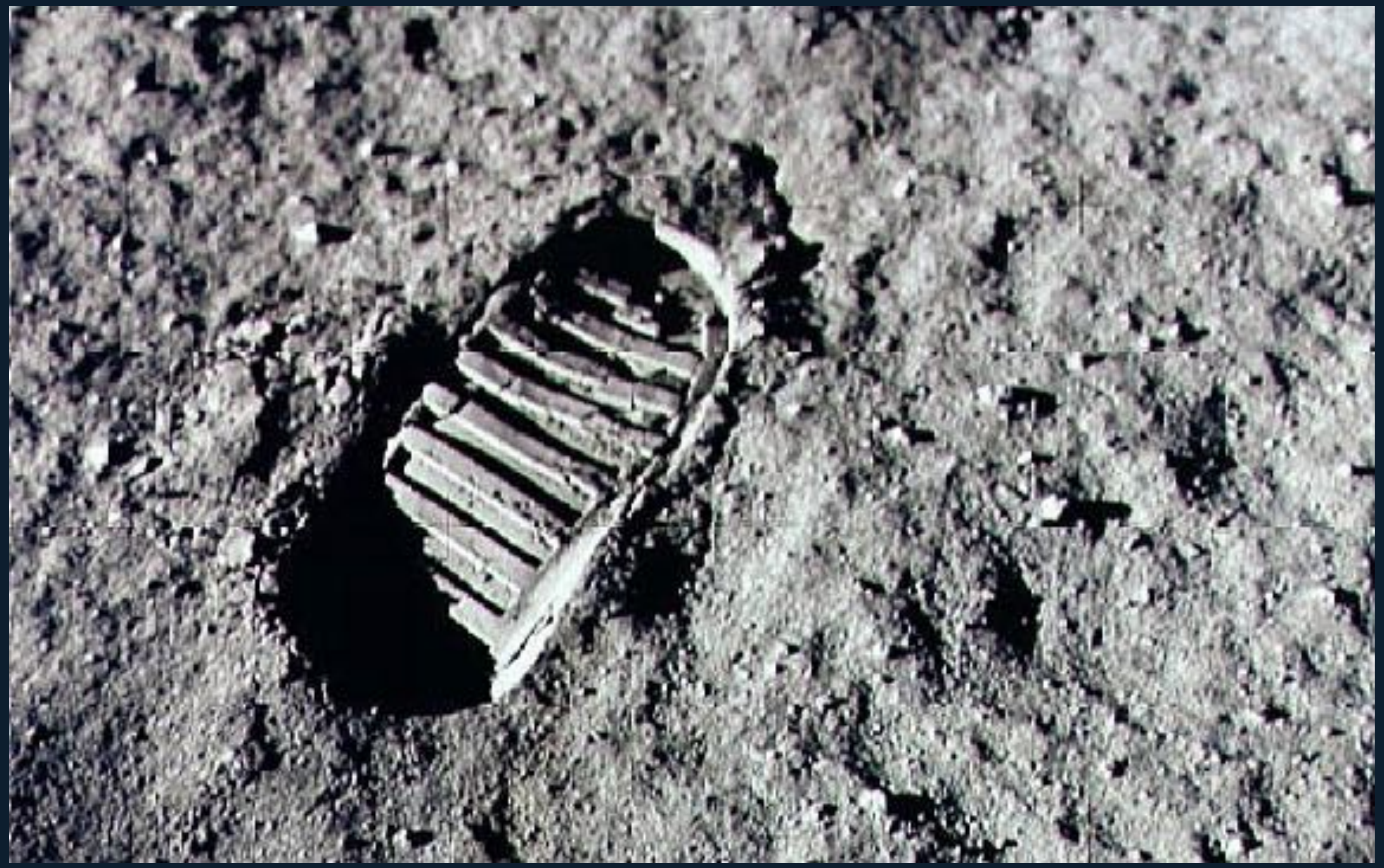
* gradient ~ derivative

(Batch) Gradient descent

In the case when we do not consider the intercept and have only one feature, we have only one parameter to estimate. Therefore, the direction of the slope can be assessed by the derivative (the sensitivity to change of the MSE value with respect to a change in the value of the parameter).

The positive sign of the derivative means that if we increase the value of the parameter, the MSE will be also increased: in this case, we have to decrease the value of the parameter. And vice versa, the negative sign means that if the value of the parameter is increased, MSE is decreased. Therefore, we need to increase the value of parameter.

A small step for a human but a big step for mankind



So far, we considered the linear regression model, the gradient of which can be expressed analytically and easy to compute.

In case of MLPs, the analytical form of the gradient is generally not known, or very difficult to compute. In 1986, David Rumelhart, Geoffrey Hinton, and Ronald Williams proposed several breakthrough innovations:

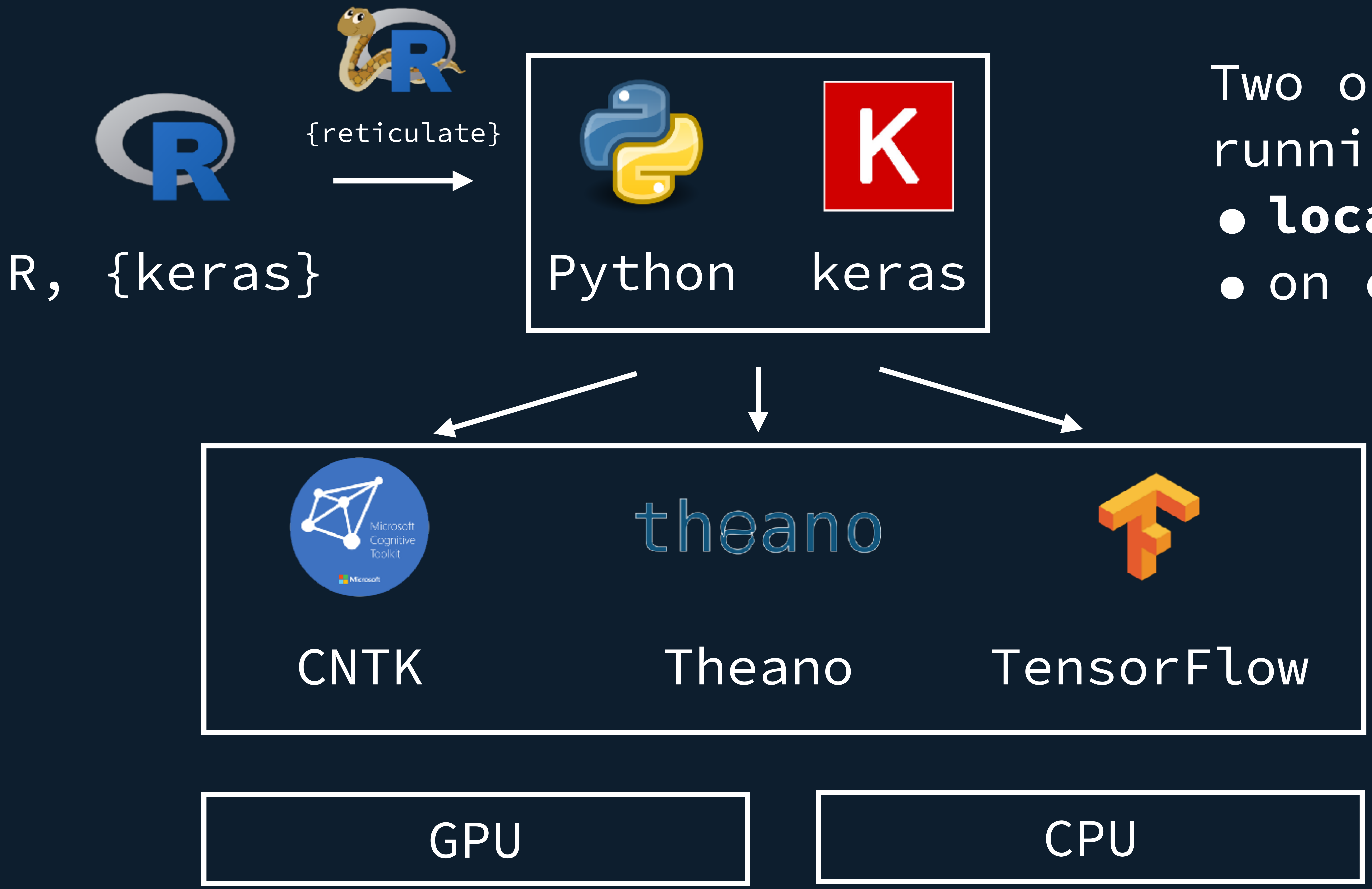
- Backpropagation algorithm
- Usage of sigmoid activation function instead of the step function

`{keras}`

Obviously, there are dozens libraries and frameworks that allow manipulating deep learning models, and one of them is a Python library `{keras}`.

`{keras}` is a high-level deep learning API for defining, training and evaluating all sorts of neural networks. Initially, it was implemented as a Python library, but soon the R package has been developed, which is basically an R interface to the Python library.

{keras}



Two options of running:

- **locally**
- on cloud

Backend

{keras}

Why use {keras}?

- User-friendly and flexible design: quick prototyping deep learning models (it was designed for humans, not machines)
- Support of central processing unit (**CPU**) and graphics processing unit (GPU): running the same code on CPU and GPU
- Support of variety of deep learning models: convolutional networks, recurrent networks, and any combination of both
- Support of arbitrary architectures: multi-input or multi-output models, layer sharing, model sharing, etc.
- Multiple backends: **TensorFlow**, Microsoft Cognitive Toolkit (CNTK, and Theano.
- Deployment on platforms: iOS (Apple’s Core ML), Android (TensorFlow Lite), etc.

{keras}

Why do WE use {keras}?

- *User friendly*
  Keras has a simple, consistent interface optimized for common use cases. It provides clear and actionable feedback for user errors.

- *Modular and composable*
  Keras models are made by connecting configurable building blocks together, with few restrictions.

- *Amazing infrastructure*
  - `{tfruns}` provides a suite of tools for tracking, visualizing, and managing TensorFlow training runs
  - `{cloudml}` provides an R interface to Google's AI Platform

{keras}

How to install?*

```r
# 1. Install {reticulate}
install.packages("reticulate")

# 2. Install Miniconda
library(reticulate)
install_miniconda()

# 3. Install {keras} and core Keras and TensorFlow backend
install.packages("keras")
library(keras)
install_keras(method = "conda")
```

* This is a non-canonical installation procedure

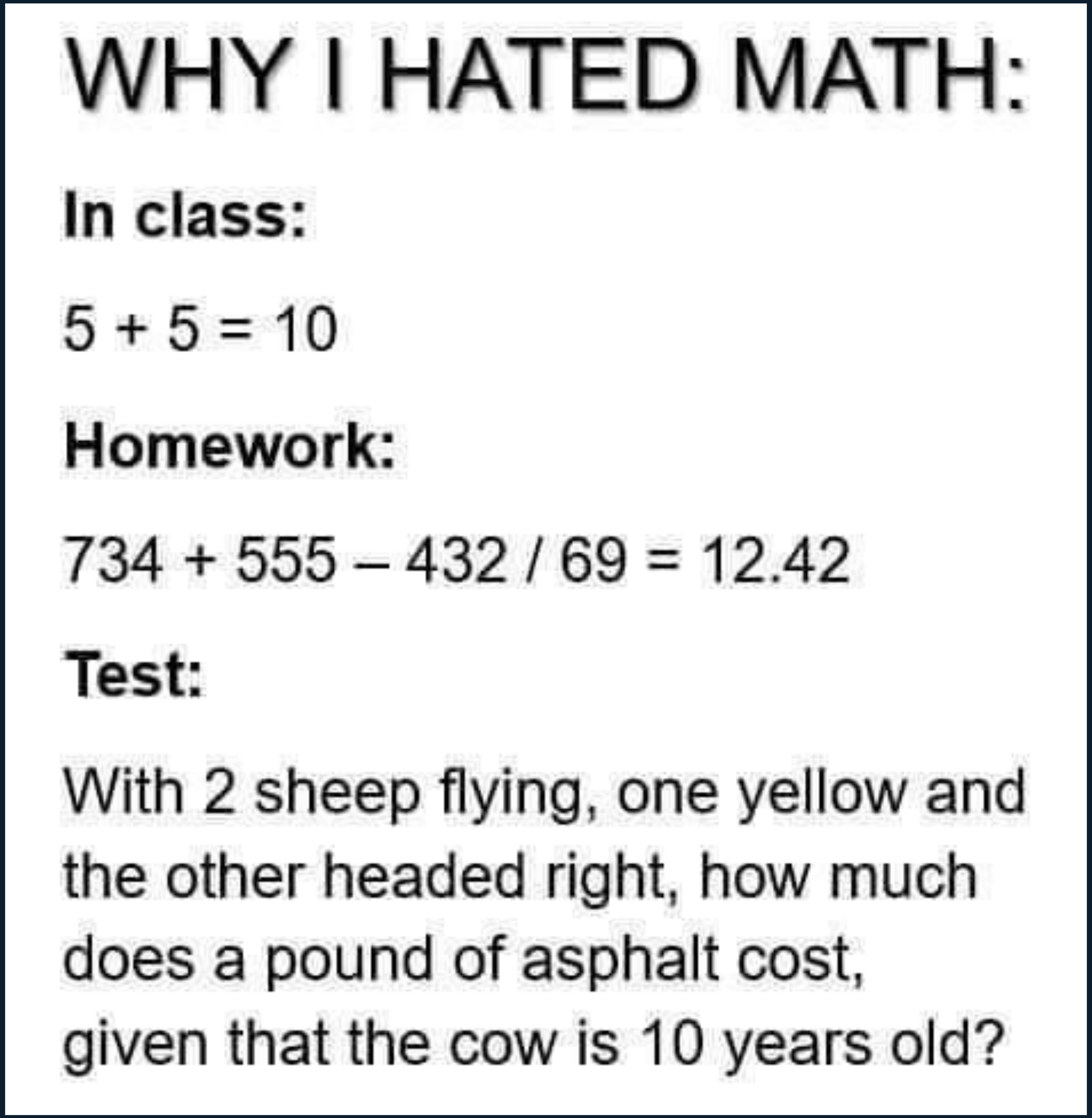
{keras}

How to install?

`{keras}`

`{keras}` workflow:

1. Create/build a desired model: define the architecture of the model in terms of `{keras}` functions
2. Compile: specify and configure optimization details
3. Fit: train the network by specifying input and output data
4. Evaluate: validate the model by measuring the performance of the model based on the test set



WHY I HATED MATH:

In class:

5 + 5 = 10

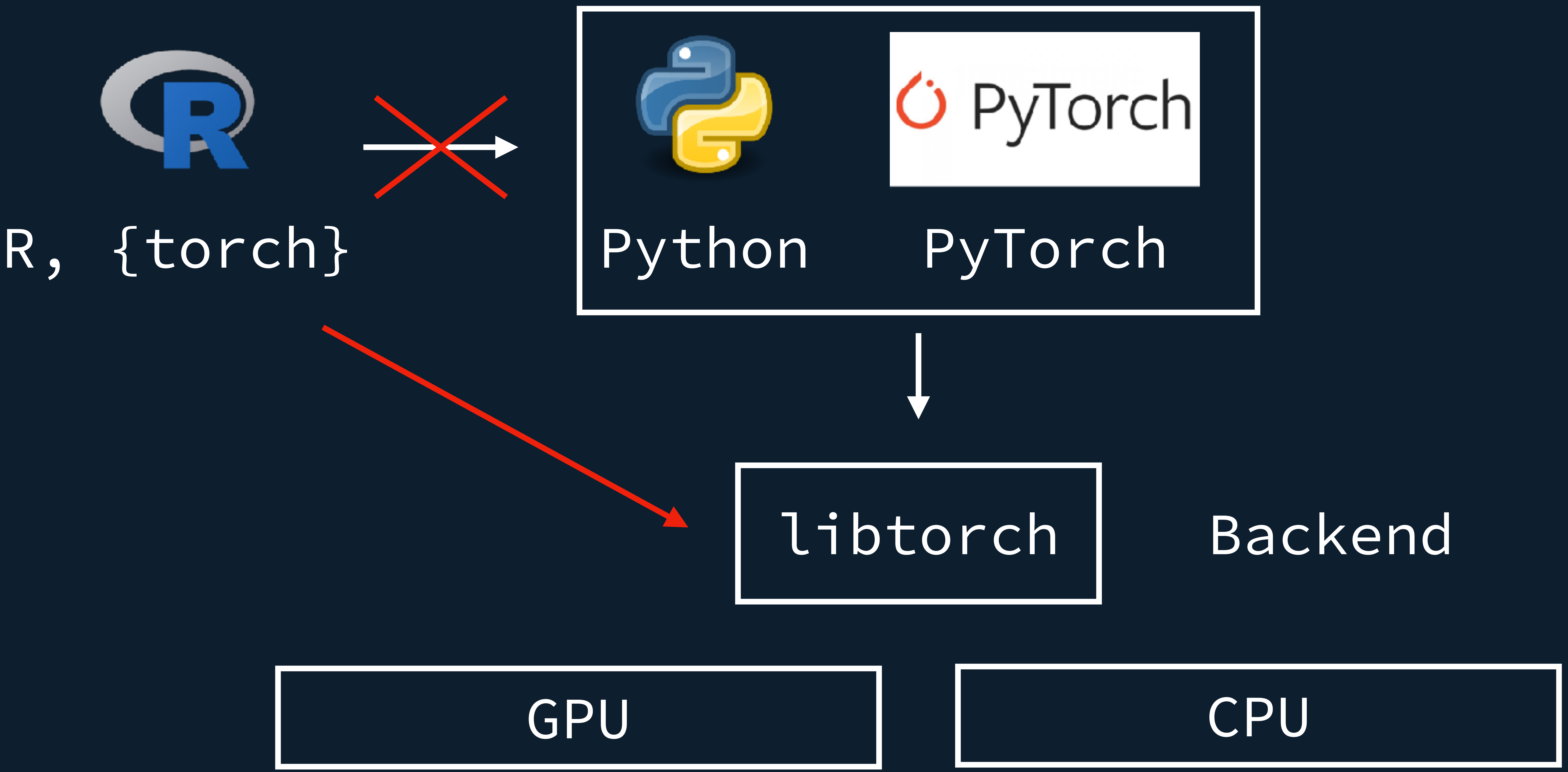
Homework:

734 + 555 – 432 / 69 = 12.42

Test:

With 2 sheep flying, one yellow and the other headed right, how much does a pound of asphalt cost, given that the cow is 10 years old?

New kid in town: {torch}

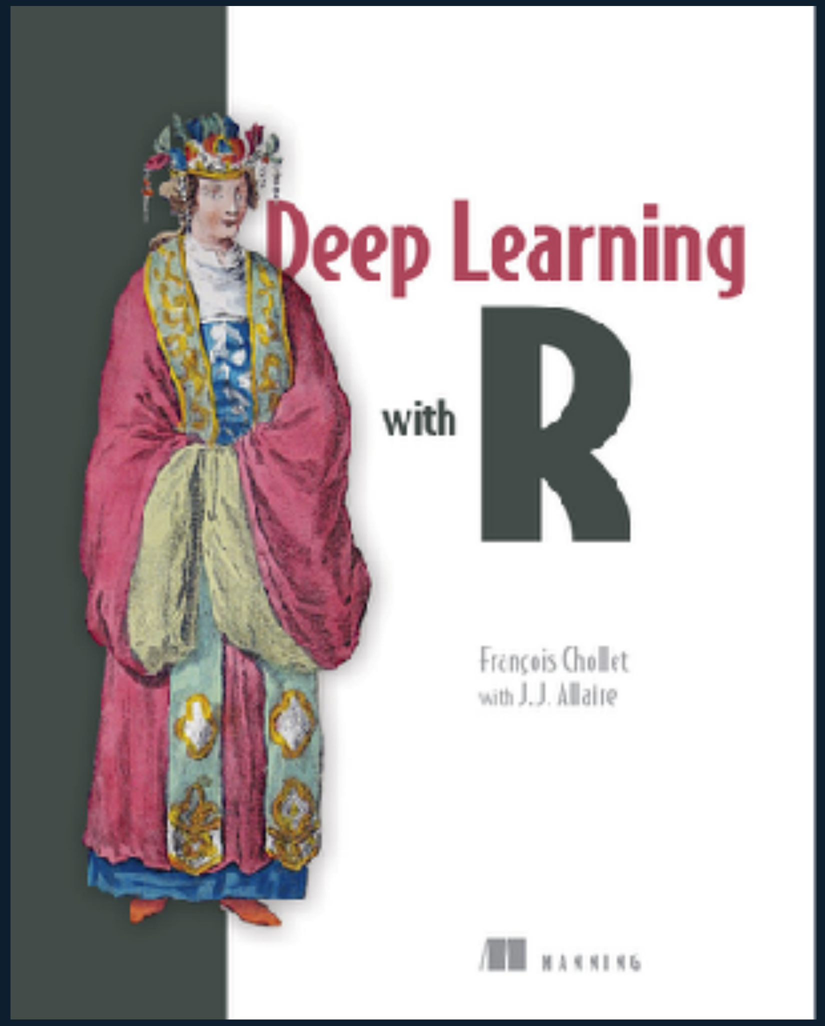


Application of deep learning

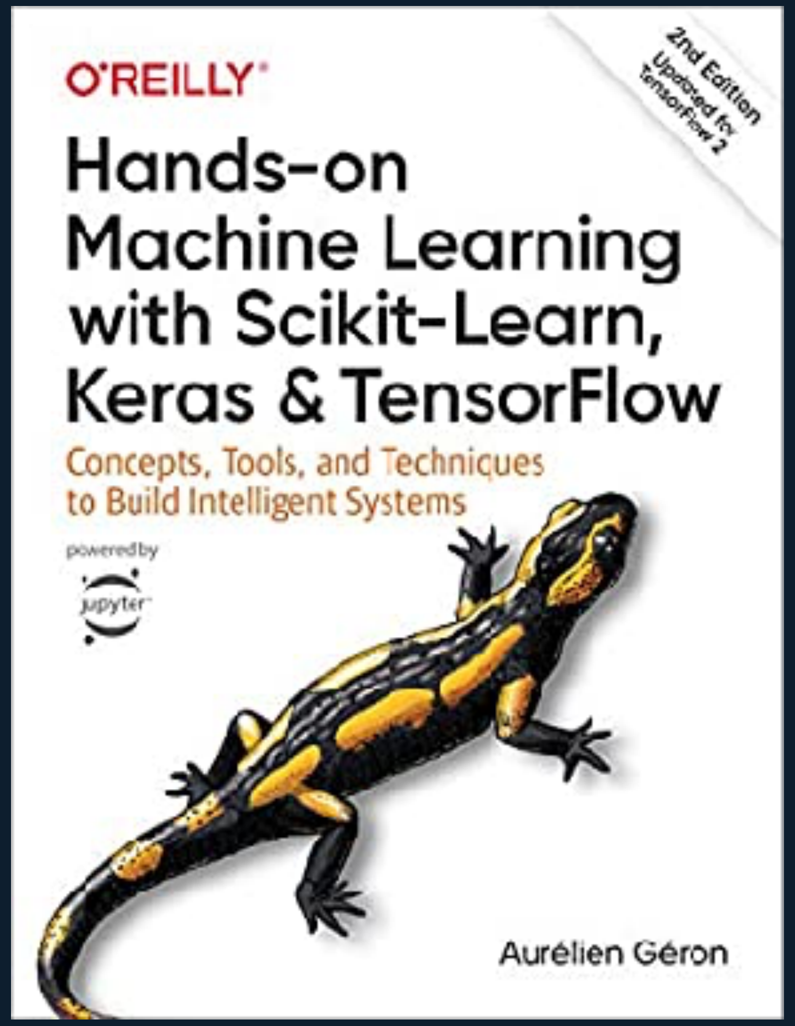
- Self-driving cars: Tesla Autopilot, Uber Advanced Technologies Group
- Virtual Assistants: Apple Siri, Amazon Alexa, Google Assistant
- Chatbots
- Natural translation models: Google Translate

All have a near human level!

Bookshelf



Chollet, F. and Allaire, J.J. (2018)
Deep Learning with R
Manning Publications Co.


O'REILLY
2nd Edition
Updated for TensorFlow 2
Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow
Concepts, Tools, and Techniques to Build Intelligent Systems
powered by jupyter
Aurélien Géron

Géron, A. (2019)
Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems
Second Edition. O'Reilly Media, Inc.

Isaac Asimov
Profession

Deep dive?

EPFL EXTENSION SCHOOL

Unil

UNIL | Université de Lausanne

THANK YOU!