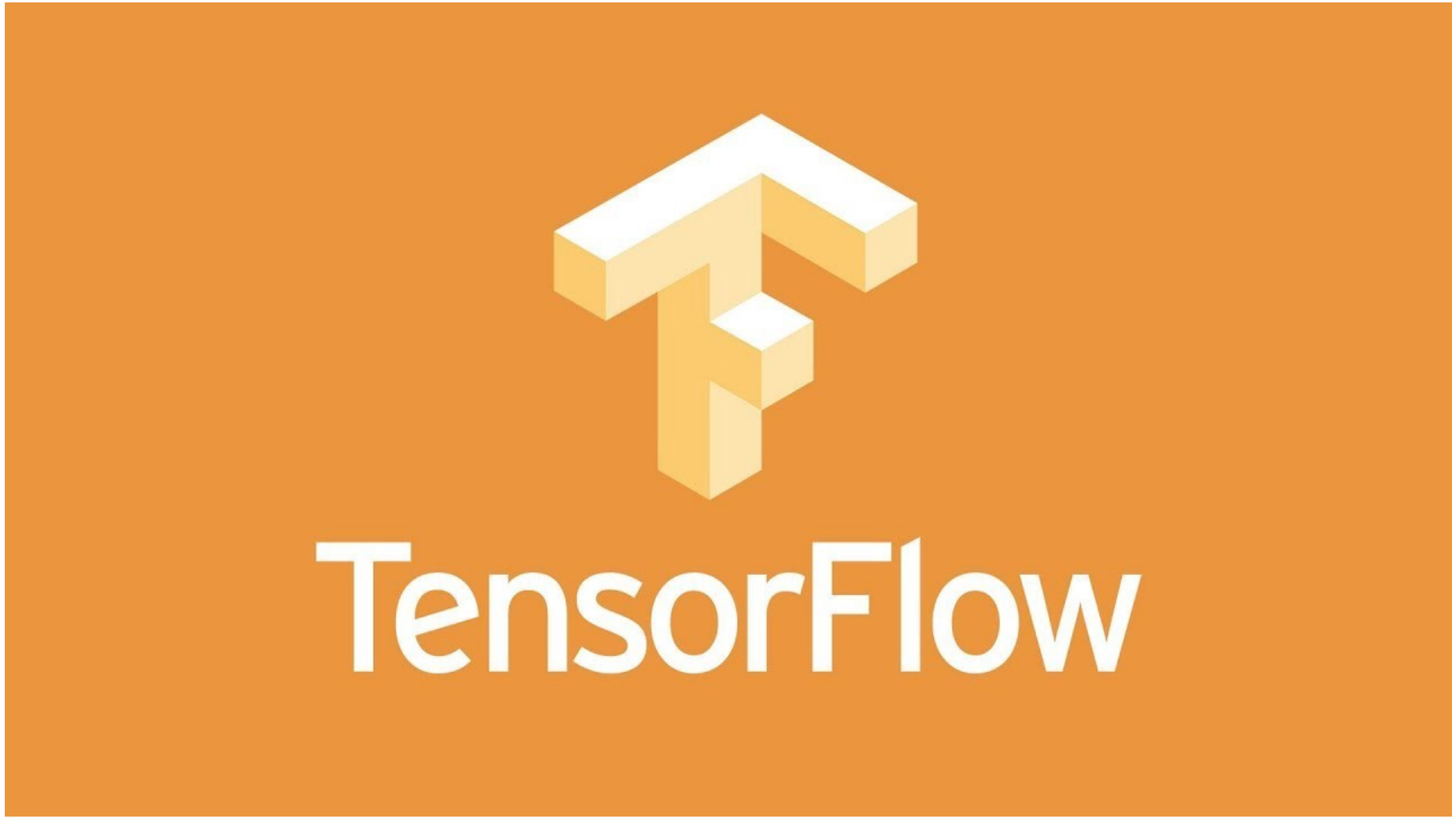
AI Deep Learning: Multilayer Perceptrons (MLPs) Fully-Connected Neural Networks with Keras

Thuan L Nguyen, PhD

AI Deep Learning Frameworks: TensorFlow & Keras

1. Keras: What is It? & Why Now?
2. Multilayer Perceptrons (MLPs): Fully Connected Neural Networks
3. Multi-Class Classification: Iris Dataset
4. Multi-Class Classification: One-Hot Coding
5. MLPs (Fully Connected Neural Networks): with Keras
6. MLPs (Fully Connected Neural Networks): with Keras: Dense Layer
7. MLPs (Fully Connected Neural Networks): with Keras: Dense Layer: Examples
8. AI Deep Learning: Activation Functions: ReLU (Rectified Linear Unit) and Softmax

AI Deep Learning Frameworks: TensorFlow & Keras



AI Deep Learning Frameworks: TensorFlow & Keras

AI Deep Learning Frameworks: TensorFlow & Keras

Keras: What is It?

- Keras is an open source neural network library written in Python.
  - It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, and other AI framework.
  - It is designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.
  - It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System).
  - Its primary author and maintainer is François Chollet, a Google engineer.
    - Chollet also is the author of the XCeption deep neural network model.

- In 2017, Google's TensorFlow team decided to support Keras in TensorFlow's core library.
- Chollet explained that:
  - Keras was conceived to be an interface rather than a standalone machine-learning framework.
  - It offers a higher-level, more intuitive set of abstractions that make it easy to develop deep learning models regardless of the computational backend used.

AI Deep Learning Frameworks: TensorFlow & Keras

Keras: What is It?

- Keras contains numerous implementations of commonly used neural network building blocks:
  - Such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier.
  - The code is hosted on GitHub, and community support forums include the GitHub issues page, and a Slack channel.

- In addition to standard neural networks,
  - Keras has support for convolutional and recurrent neural networks.
  - It supports other common utility layers like dropout, batch normalization, and pooling.

- Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine.
  - It also allows the use of distributed training of deep learning models on clusters of Graphics Processing Units (GPU) and Tensor processing units (TPU).

AI Deep Learning Frameworks: TensorFlow & Keras

Keras: Why?

- Keras is an API designed for human beings, not machines.
  - Keras follows best practices for reducing cognitive load
  - It offers consistent & simple APIs
  - It minimizes the number of user actions required for common use cases.
  - It provides clear and actionable feedback upon user error.

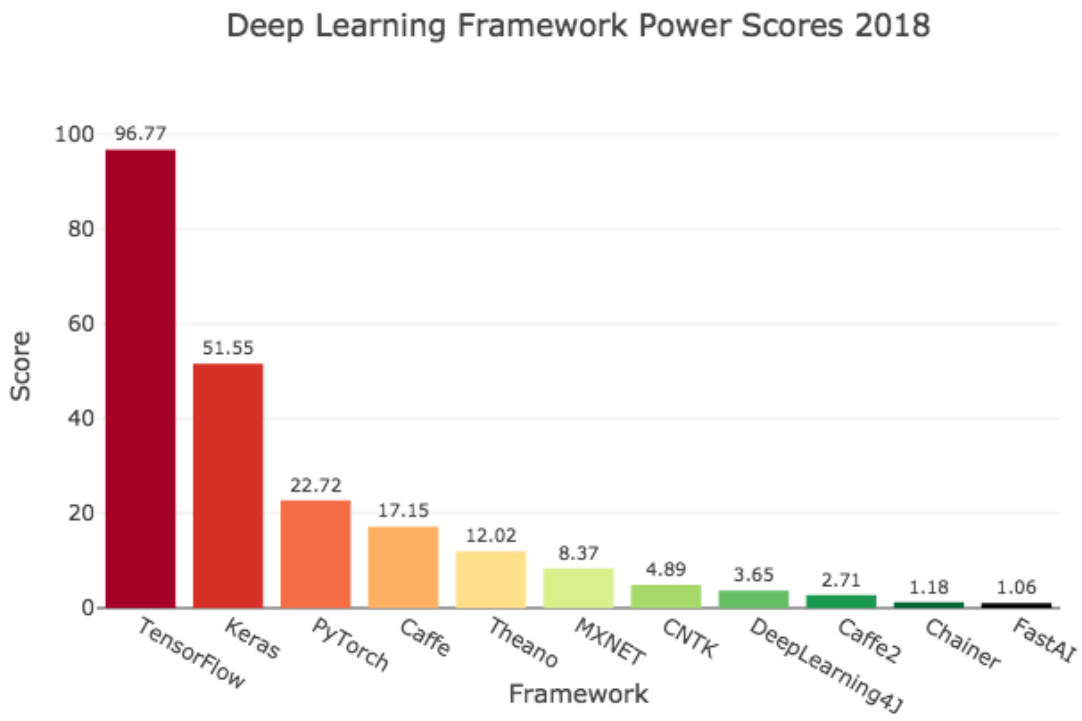
- This makes Keras easy to learn and easy to use.

- This ease of use does not come at the cost of reduced flexibility:
  - Because Keras integrates with lower-level deep learning languages (in particular TensorFlow), it enables researchers to implement anything they could have built in the base language.
  - In particular, as **tf.keras**, the Keras API integrates seamlessly with the **TensorFlow** workflows.

AI Deep Learning Frameworks: TensorFlow & Keras

Keras: Why?

- Keras has broad adoption in the industry and the research community

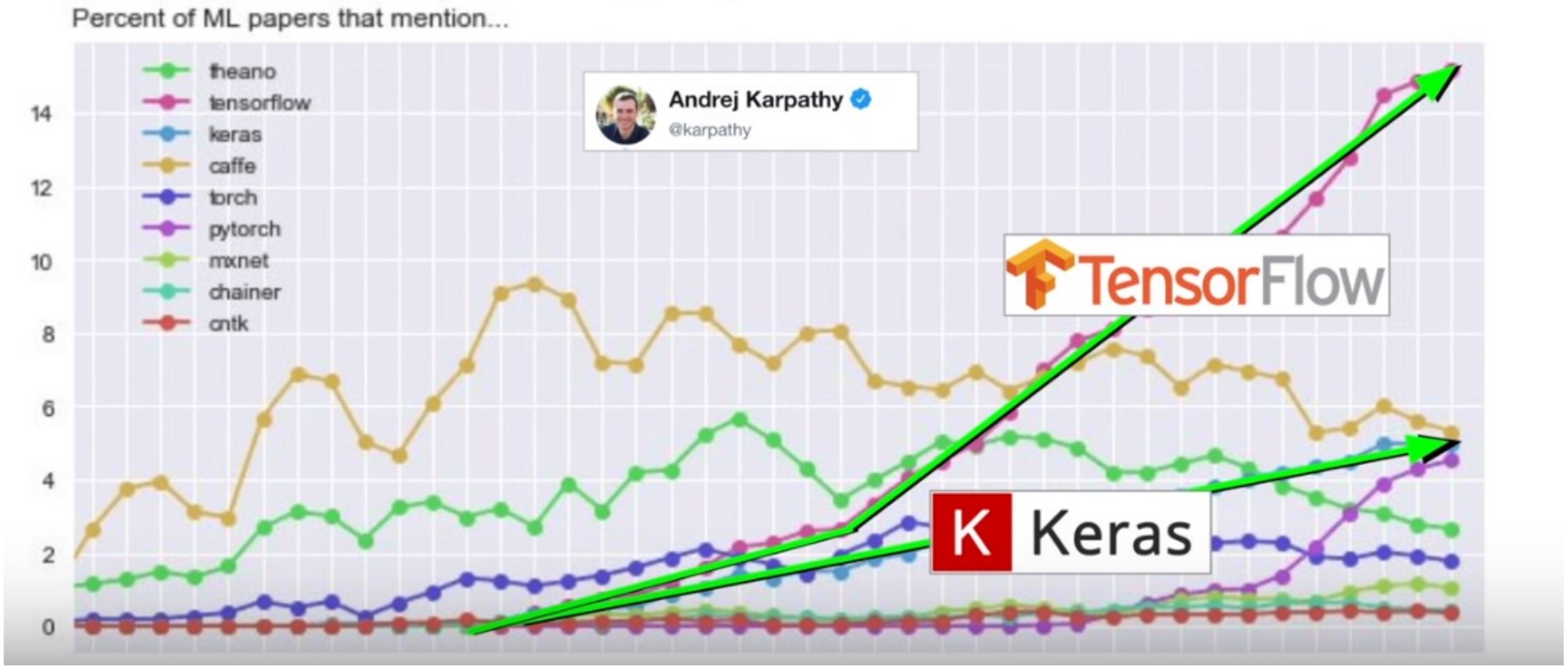


Sources: Jeff Hale – Towards Data Science

AI Deep Learning Frameworks: TensorFlow & Keras

TensorFlow and Keras: Why?

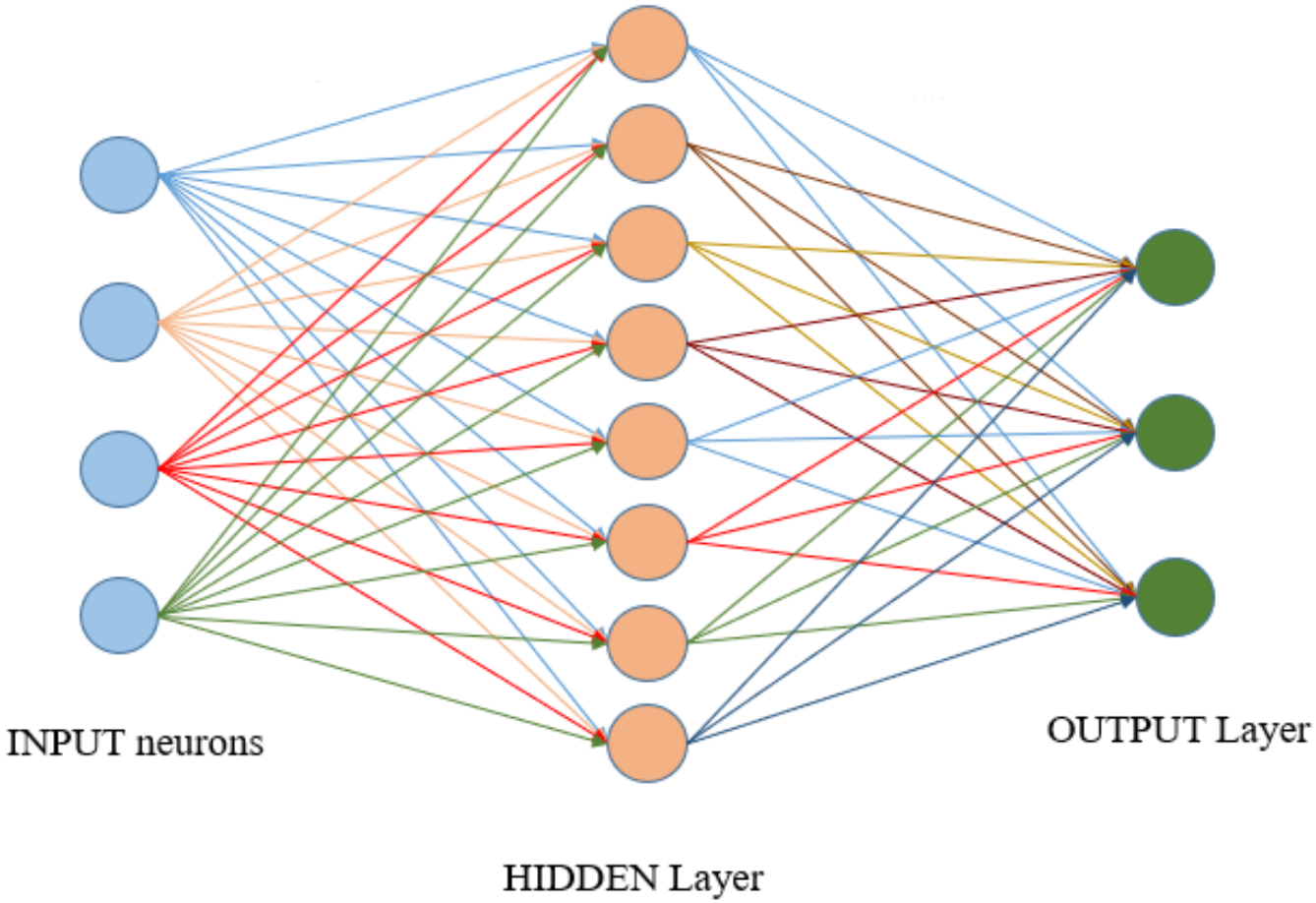
- TensorFlow and Keras are widely used for deep learning projects.
  - They are now the top deep learning frameworks in research and production.



Sources: Andrej Karpathy and Kiwisoft

AI Deep Learning: Multilayer Perceptrons (MLPs) with Keras

Multilayer Perceptrons (MLPs): a.k.a. Fully Connected Neural Networks



AI Deep Learning: Multilayer Perceptrons (MLPs) with Keras

Multilayer Perceptrons (MLPs): A Bit of History of Perceptron

- A perceptron is a linear classifier:
  - It is an algorithm that classifies input by separating two categories with a straight line.
  - Input is typically a feature vector x multiplied by weights w and added to a bias b:
    - **y = w * x + b.**

- A perceptron produces a single output based on several real-valued inputs
  - By forming a linear combination using its input weights (and sometimes passing the output through a nonlinear activation function).

- The algorithm can be modeled in a mathematic expression:
  - Where **w**: the vector of weights, **x**: the vector of inputs, **b**: the bias, **phi**: the non-linear activation function.

$$y = \varphi(\sum_{i=1}^{n} w_i x_i + b) = \varphi(\mathbf{w}^T \mathbf{x} + b)$$

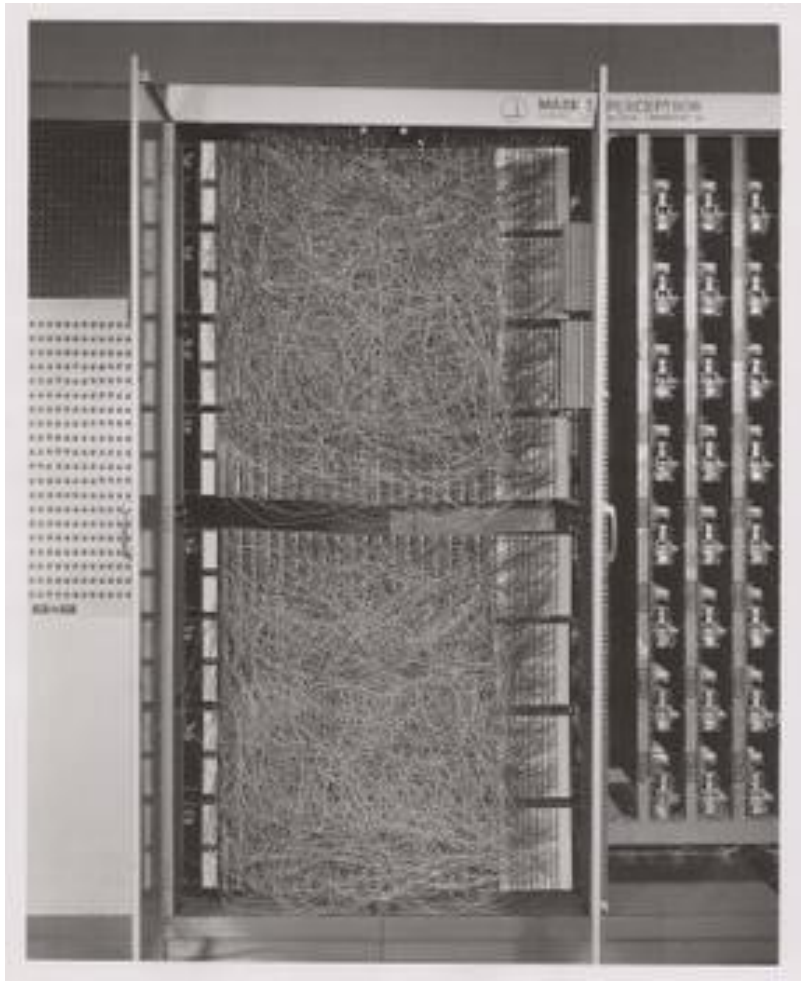
AI Deep Learning: Multilayer Perceptrons (MLPs) with Keras

Multilayer Perceptrons (MLPs): A Bit of History of Perceptron

- The perceptron, the simplest neural network, is a simple algorithm intended to perform binary classification: yes or no: approved or rejected, etc.

- Frank Rosenblatt, the godfather of the perceptron, make it well-known as a computing device rather than an algorithm.

- The perceptron first entered the world as hardware.
  - Rosenblatt, a psychologist who studied and later lectured at Cornell University, received funding from the U.S. Office of Naval Research to build a machine that could learn.
  - His machine, the Mark I perceptron, is pictured in the figure to the right.



AI Deep Learning: Multilayer Perceptrons (MLPs) with Keras

Multilayer Perceptrons (MLPs): A Bit of History of Perceptron

- Rosenblatt built a single-layer perceptron.
  - His hardware-algorithm did not include multiple layers, which allow neural networks to model a feature hierarchy.

  - It was, therefore, a shallow neural network:
    - That prevented his perceptron from performing non-linear classification, such as the XOR function
    - An XOR (exclusive OR) operator trigger when input exhibits either one trait or another, but not both,
      - *As Minsky and Papert showed in their research paper (Minsky and Papert, 1969)*

XOR

Input x_1	Input x_2	Output
0	0	0
0	1	1
1	0	1
1	1	0

AI Deep Learning: Multilayer Perceptrons (MLPs) with Keras

Artificial Neural Networks: Limitations of Simple Neural Networks

Let's do the same for the XOR logic operation. The training data can lead to four inequalities:

in_1	in_2	out
0	0	0
0	1	1
1	0	1
1	1	0

$\Rightarrow$

$$
\begin{aligned}
w_1 0 + w_2 0 - \theta &< 0 \\
w_1 0 + w_2 1 - \theta &\geq 0 \\
w_1 1 + w_2 0 - \theta &\geq 0 \\
w_1 1 + w_2 1 - \theta &< 0
\end{aligned}
$$

$\Rightarrow$

$$
\begin{aligned}
\theta &> 0 \\
w_2 &\geq \theta \\
w_1 &\geq \theta \\
w_1 + w_2 &< \theta
\end{aligned}
$$

- Clearly the first, second and third inequality are incompatible with the fourth.
  - So, there is no solution.
- It's necessary to use more complex networks – Multi-Layered Neural Networks (MLP).
  - The networks that combine together many simple networks.
- With more complex neural networks:
  - It is much more difficult to determine all the weights and thresholds by hand.
  - It is necessary to use AI software frameworks such as TensorFlow, Keras, etc.

AI Deep Learning: Multilayer Perceptrons (MLPs) with Keras

Multilayer Perceptrons (MLPs): a.k.a. Fully Connected Neural Networks

- A multilayer perceptron (MLP) is a deep, artificial neural network.

- An **MLP** is composed of:
  - Input neurons (some books label them as "input layer") to receive the signal
  - An output layer that makes a decision or prediction about the input
  - And an arbitrary number of hidden layers (between the input layer and output layer) that are considered as the computational engine of the MLP.

- **Multilayer perceptrons** are often used for **supervised** learning problems
  - They train on a set of input-output pairs and learn to model the correlation (or dependencies) between those inputs and outputs
  - Training involves adjusting the parameters, or the weights and biases, of the model in order to minimize error.

AI Deep Learning: Multilayer Perceptrons (MLPs) with Keras

Multi-Class Classification on Iris Dataset

- The Iris flower data set or Fisher's Iris data set:
  - Is introduced by the British statistician and biologist Ronald Fisher in his 1936 paper.
    - "The use of multiple measurements in taxonomic problems as an example of linear discriminant analysis."

Photo by houroumono, some rights reserved.

AI Deep Learning: Multilayer Perceptrons (MLPs) with Keras

Multi-Class Classification on Iris Dataset

- It is sometimes called Anderson's Iris data set because Edgar Anderson collected the data to quantify the morphologic variation of Iris flowers of three related species.
  - Two of the three species were collected in the Gaspé Peninsula "all from the same pasture, and picked on the same day and measured at the same time by the same person with the same apparatus".

- The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor).
  - Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters.

AI Deep Learning: Multilayer Perceptrons (MLPs) with Keras

Multi-Class Classification: Encoding Classes with One-Hot Coding

- One of the major problems with Machine Learning:
  - It is a daunting task to work directly with categorical data.
- Computers are powerful and efficient when dealing with numbers.
  - So, it would be much better to convert our input data (if it is not in numeric formats) to numbers.

AI Deep Learning: Multilayer Perceptrons (MLPs) with Keras

Multi-Class Classification: Encoding Classes with One-Hot Coding

- What is category data?
  - Categorical data are variables that contain label values rather than numeric values.
  - The number of possible values is often limited to a fixed set.
  - Categorical variables are often called nominal.

- Some examples include:
  - A “pet” variable with the values: “dog” and “cat“.
  - A “color” variable with the values: “red“, “green” and “blue“.
  - A “place” variable with the values: “first”, “second” and “third“.

- Each value represents a different category.

- Some categories may have a natural relationship to each other, such as a natural ordering.
  - The “place” variable above does have a natural ordering of values.
  - This type of categorical variable is called an ordinal variable.

AI Deep Learning: Multilayer Perceptrons (MLPs) with Keras

Multi-Class Classification: Encoding Classes with One-Hot Coding

What is the problem with category data?

- Some algorithms can work with categorical data directly.
  - For example, a decision tree can be learned directly from categorical data with no data transform required (this depends on the specific implementation).

- Many machine learning algorithms cannot operate on label data directly.
  - They require all input variables and output variables to be numeric.
  - In general, this is mostly a constraint of the efficient implementation of machine learning algorithms rather than hard limitations on the algorithms themselves.

- This means that categorical data must be converted to a numerical form.
  - If the categorical variable is an output variable, it is desirable to convert predictions by the model back into a categorical form in order to present them or use them in some application.

AI Deep Learning: Multilayer Perceptrons (MLPs) with Keras

Multi-Class Classification: Encoding Classes with One-Hot Coding

How to Convert Categorical Data to Numerical Data?

- The conversion is done in two steps:
  - Integer coding
  - One-hot coding

First Step: Integer Coding

- First, each unique category value is assigned an integer value.
  - For example:
    - "Yes" = 1, "No" = 0
    - "*red*" = 0, "*green*" = 1, "*blue*" = 2.
    - "sedan" = 1, "truck" = 2; "sport-utility" = 3.

- This is called a label encoding or an integer encoding and is easily reversible.
- The integers have a natural ordered relationship that is good for ordinal and boolean variables
  - In this case, integer coding is good enough, e.g., "Yes" = 1; "No" = 0.

AI Deep Learning: Multilayer Perceptrons (MLPs) with Keras

Multi-Class Classification: Encoding Classes with One-Hot Coding

How to Convert Categorical Data to Numerical Data?

Second Step: One-Hot Coding

- For categorical variables where no such ordinal relationship exists:
  - The integer encoding is not enough.
- In fact, using the integer encoding and allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results
  - Like predictions halfway between categories
- In this case, a one-hot encoding can be applied to the integer representation.
  - A vector (1D array) of binary values is used to represent each unique integer value.
  - For example, for three colors, there are 3 categories that needs 3 vectors of binary values:
    - "Red" = 0: [1, 0, 0] → the value at the index 0 = 1; all the values at other indices = 0
    - "Green" = 1: [0, 1, 0] → the value at the index 1 = 1; all the values at other indices = 0
    - "Blue" = 2: [0, 0, 1] → the value at the index 2 = 1; all the values at other indices = 0

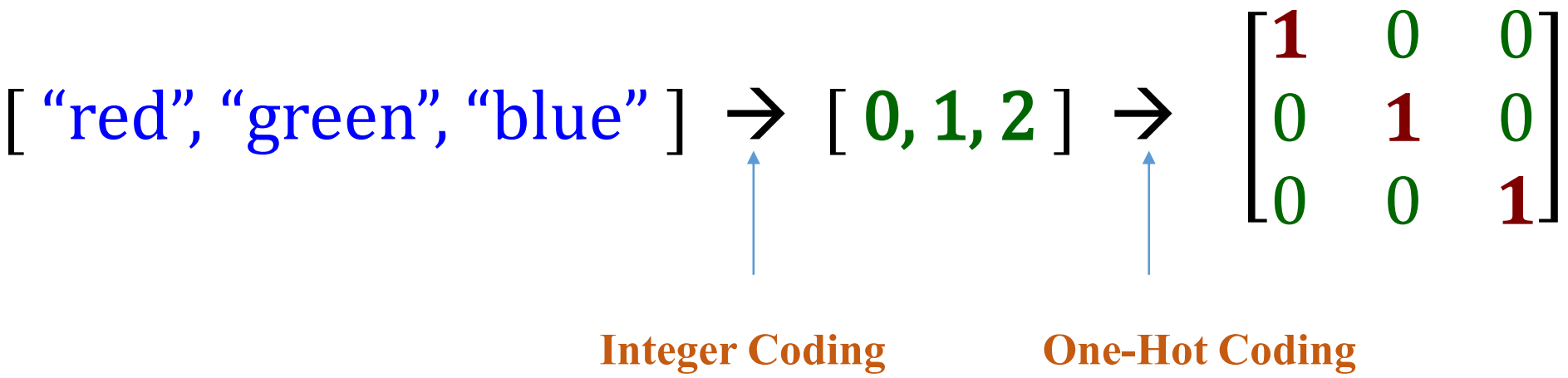
AI Deep Learning: Multilayer Perceptrons (MLPs) with Keras

Multi-Class Classification: Encoding Classes with One-Hot Coding

How to Convert Categorical Data to Numerical Data?

Second Step: One-Hot Coding

- With the one-hot coding, the set of categorical values (“red”, “green”, “blue”) is transformed from a **vector** (1D array) into a **matrix** (2D array = vector of vectors):



AI Deep Learning: Multilayer Perceptrons (MLPs) with Keras

Multi-Class Classification: Encoding Classes with One-Hot Coding

How to Convert Categorical Data to Numerical Data?

Second Step: One-Hot Coding

- With the one-hot coding, the set of categorical values ("red", "green", "blue") is transformed from a **vector** (1D array) into a **matrix** (2D array = vector of vectors).

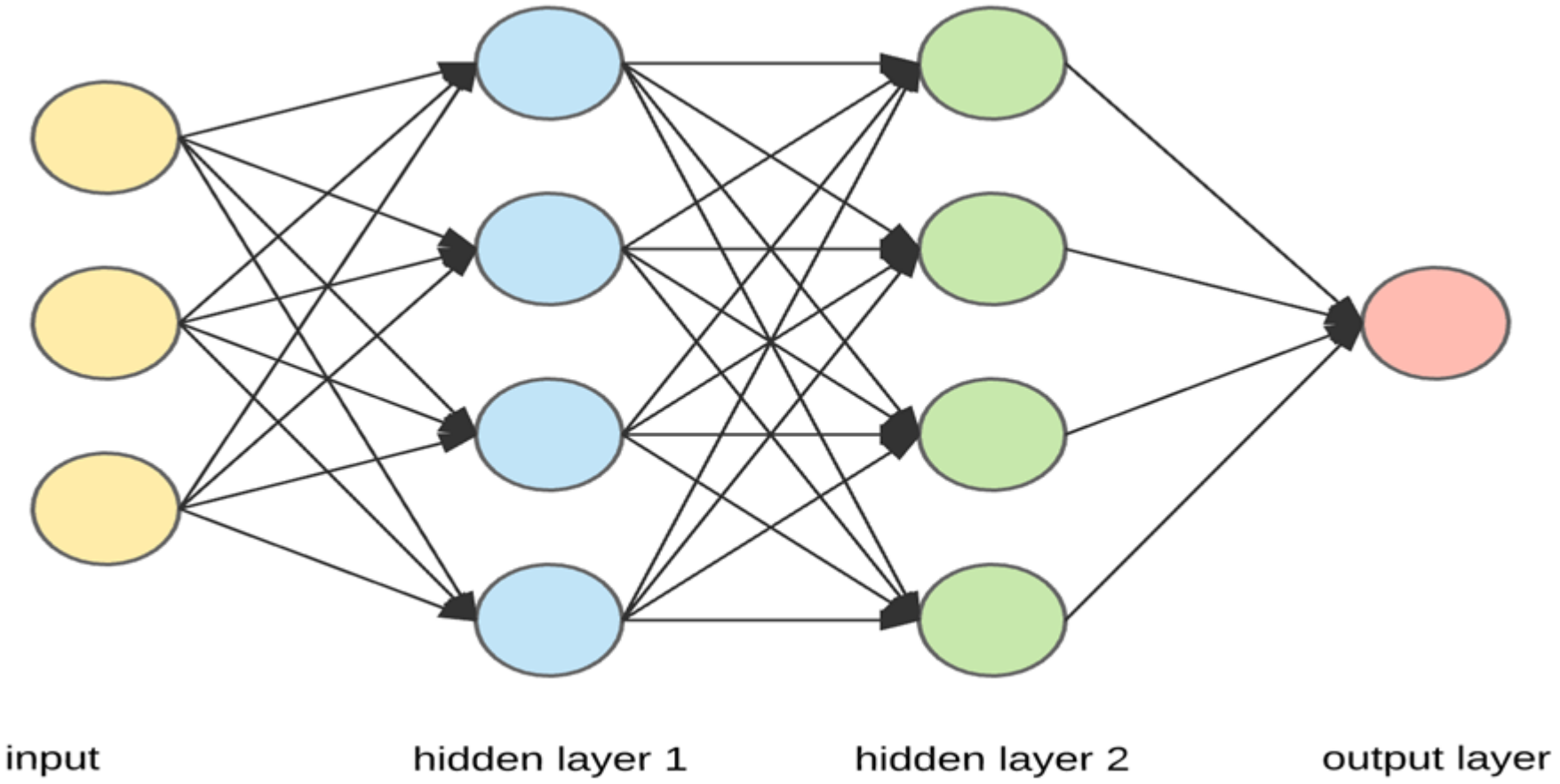
IMPORTANT NOTES:

--) If the class attribute of a dataset has only two categorical values, e.g., "**Yes**" and "**No**", "Approved" and "Rejected", etc., in most cases, the **integer coding is enough**.

--) If the class attribute of a dataset has more than two categorical values, **even if** that they are already in numeric formats like the **integers**, it should be better to **encode them using one-hot coding** to get better performance in AI machine learning and deep learning.

AI Deep Learning: Multilayer Perceptrons (MLPs) with Keras

MLPs (Fully Connected Neural Networks) with Keras



AI Deep Learning Frameworks: TensorFlow & Keras

MLPs (Fully Connected Neural Networks) with Keras

- There are two ways to build Keras models: **sequential** and **functional**.

- The sequential API
  - It allows you to create models layer-by-layer for most problems.
  - It is limited in that it does not allow you to create models that share layers or does not help you much in defining complex models.

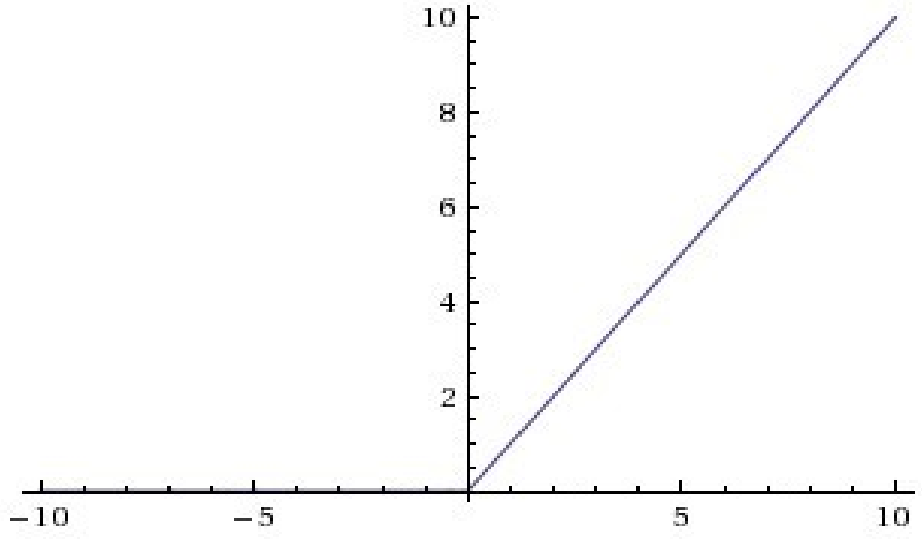
- Alternatively, the functional API – using the class Model – allows you to create models with a lot more flexibility
  - As you can easily define models where layers connect to more than just the previous and next layers.
  - In fact, you can connect layers to (literally) any other layer.
  - As a result, creating complex networks such as siamese networks and residual networks become possible.

AI Deep Learning Frameworks: TensorFlow & Keras

AI: Deep Learning: Activation Functions

- Many activation functions can be used in the layers of a neural networks.
  - In recent years, two activation functions – **ReLU** and **Softmax**, have become very popular thanks to their superior performance in comparison with the alternatives.

- **ReLU (Rectified Linear Unit)** function:
  - The Rectified Linear Unit (ReLU) has become very popular in the last few years.
  - It computes the function **f(x)=max(0,x)**.
    - In other words, the activation is simply thresholded at zero (see image to the right).



AI Deep Learning Frameworks: TensorFlow & Keras

AI: Deep Learning: Activation Functions

- **Softmax** function:
  - In mathematics, the **softmax** function, also known as softargmax or normalized exponential function is a function that takes as **input** a vector of K real numbers, and **normalizes** it into a probability distribution consisting of K probabilities. (Source: Wikipedia)
    - It means that prior to applying softmax, some vector components could be negative, or greater than one; and might not sum to 1; but after applying softmax, each component will be in the interval (0,1), and the components will add up to 1, so that they can be interpreted as probabilities.
    - Furthermore, the larger input components will correspond to larger probabilities. Softmax is often used in neural networks, to map the non-normalized output of a network to a probability distribution over predicted output classes.

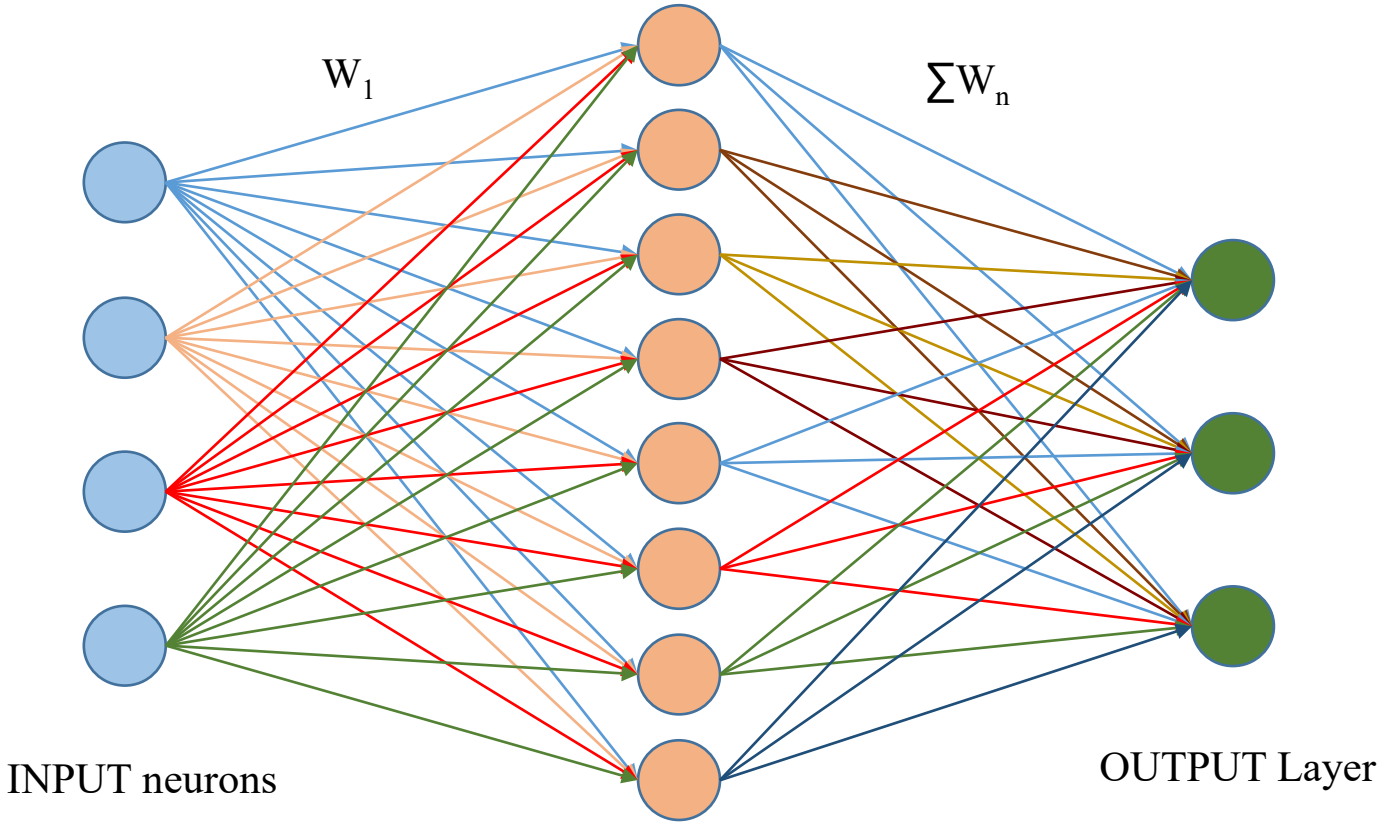
The standard (unit) softmax function $\sigma : \mathbb{R}^K \to \mathbb{R}^K$ is defined by the formula

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$$

  - The **softmax** function is often used in the final layer of a **neural network**-based classifier.
    - Such networks are commonly trained under a log loss (or cross-entropy) regime, giving a non-linear variant of multinomial logistic regression. (Source: Wikipedia)

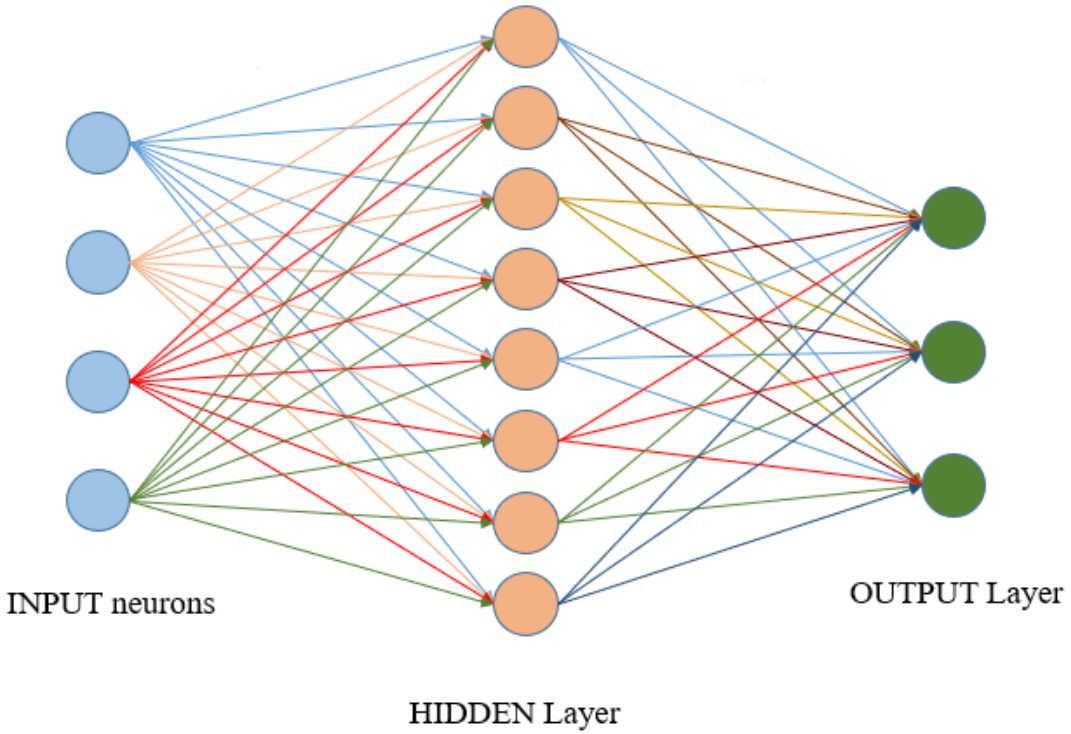
AI Deep Learning: Multilayer Perceptrons (MLPs) with Keras

MLPs (Fully Connected Neural Networks: 2 Layers) with Keras



AI Deep Learning: Multilayer Perceptrons (MLPs) with Keras

MLPs (Fully Connected Neural Networks: 2 Layers) with Keras



IMPORTANT NOTES:

*Some books label the **input neurons** as the "input layer." However, this input layer is **not counted** as a layer of the neural network.*

AI Deep Learning: Multilayer Perceptrons (MLPs) with Keras

MLPs (Fully Connected Neural Networks) with Keras

- Keras: DENSE Layer or Fully Connected Neural Networks
  - It is just a regular layer of neurons in a neural network.
  - Each neuron receives input from all the neurons in the previous layer, thus densely connected.
  - The layer has a weight matrix **W,** a bias vector **b,** and the activation function **a.**

- Examples of building DENSE model with Keras sequential API:
  - from keras.models import Sequential
  - from keras.layers import Dense
  - model = Sequential()
  - model.add(Dense(8, input_dim=4, activation='relu‘)
  - model.add(Dense(3, activation='softmax'))

- In the above example with IRIS dataset: 2-Layered MLP
- Two layers: The hidden layer + OUTPUT layer
  - --) The input neurons**:** 4 neurons for 4 predictors, e.g., with IRIS dataset
  - --) The 1st layer (the hidden layer): 8 neurons that receives data from the 4 input variables.
  - --) The 2nd layer (the output layer): 3 neurons to predict the flower species (three species)

AI Deep Learning: Multilayer Perceptrons (MLPs) with Keras

MLPs (Fully Connected Neural Networks) with Keras

- Examples of building DENSE model with Keras sequential API:
  - Add another hidden layer into the MLP in the previous example
    - from keras.models import Sequential
    - from keras.layers import Dense
    - model = Sequential()
    - model.add(Dense(8, input_dim=4, activation='relu‘)
    - model.add(Dense(4, activation='relu‘))
    - model.add(Dense(3, activation='softmax'))
- In the example:
  - Three layers: The input layer (**not** counted) + 2 hidden layers + the output layer
  - --) The input layer: 4 neurons used for inputs and the ‘relu’ as the activation function
  - --) The 1st hidden layer: 8 neurons that can receive input data from 4 inputs (input_dim = 4)
  - --) The 2nd hidden layer: 4 neurons and the activation function ‘relu’.
  - --) The output layer: 3 neurons to make prediction on three categories and the softmax as the activation function.