哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

Big data technology and Practice

李春山

2020年9月30日

内容提要

Chapter 12 Neural Networks

Chapter 13 Building ANNs with Tensorflow and Keras

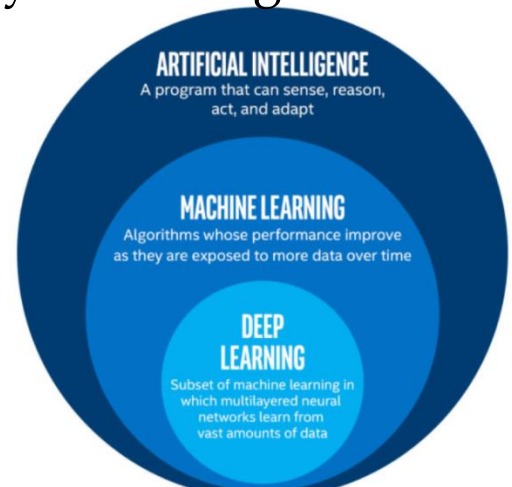
Chapter 12 Neural Networks

- We are going to study Deep Learning technology in the rest of five Sessions.
- Briefly, let's understand a number of popular terms or concepts and their relationships:
  - Artificial Intelligence (AI)
  - Machine Learning (ML)
  - Deep Learning (DL)
  - Big data
  - Cloud Computing

Neural Networks

- **Many definitions of AI, one of them: "It is the study of how to train the computers so that they can do things which at present human can do better."**
- DL is just a subset of ML. Technically, they are similar, but their capabilities are different.
- A DL model is designed to continually analyze data with a logic structure similar to how a human would draw conclusions. To achieve this, it uses a layered structure of algorithms called an artificial neural network (ANN), which is inspired by the biological neural network of the human brain.
- **A great example of deep learning**

is Google's AlphaGo



Neural Networks

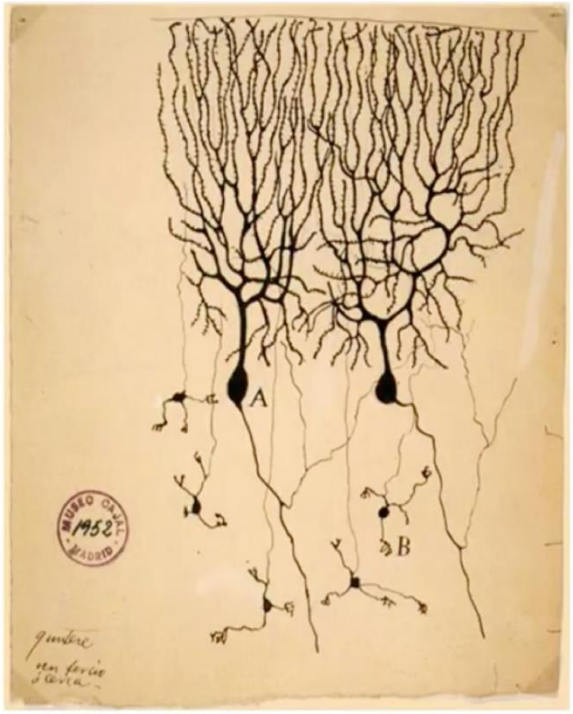
- **How to build an intelligent machine, by looking at the brain's architecture- the key idea that inspired artificial neural networks (ANNs), since 1943.**
- ANNs have gradually become quite different from their biological cousins, similar to building airplanes.
- **ANNs are at the very core of Deep Learning. They are versatile, powerful, and scalable, making them ideal to tackle large and highly complex Machine Learning tasks, such as**
  - Classifying billions of images (e.g., Google Images),
  - Recognition services (e.g., Apple's Siri),
  - Recommending the best videos to hundreds of millions of users every day (e.g.,YouTube)
  - Learning to beat the world champion (DeepMind's AlphaGo).

Neural Networks

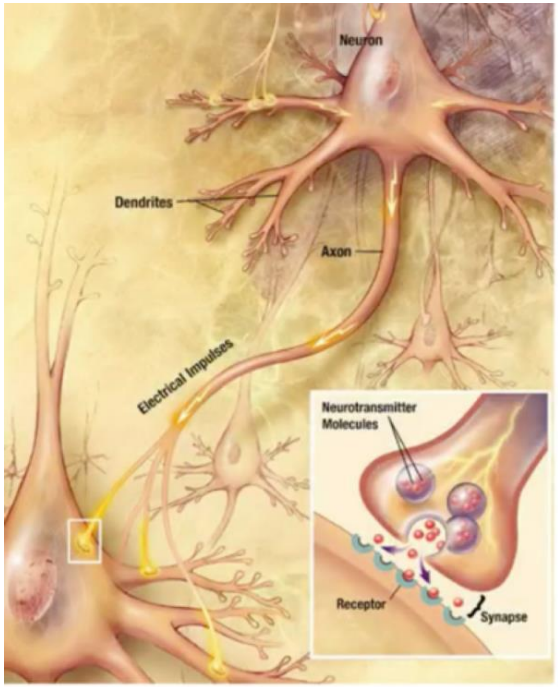
- **ANNs have been around for quite a while: they were first introduced back in 1943. But, why they become so powerful**
  - Today, a huge quantity of data available to train neural networks, and ANNs frequently outperform other ML techniques on very large and complex problems.
  - The tremendous increase in computing power since the 1990s, possible to train very large neural networks in a reasonable amount of time.
  - The training algorithms have been improved, since 1990s. Some theoretical limitations of ANNs have turned out to be benign in practice.
  - ANNs seem to have entered a virtuous circle of funding and progress.

1. Biological Neurons

- It is an unusual-looking cell mostly found in animal cerebral cortexes (e.g., your brain).
- Biological neurons receive short electrical impulses called signals from other neurons via these synapses.

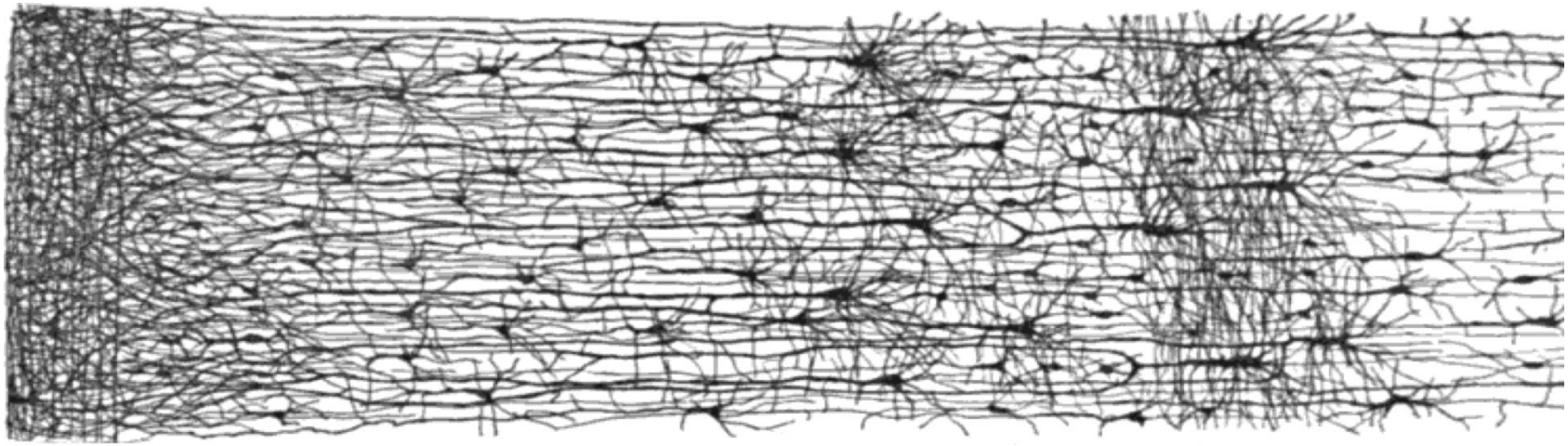


- **When a neuron receives a sufficient number of signals from other neurons within a few milliseconds, it fires its own signals**



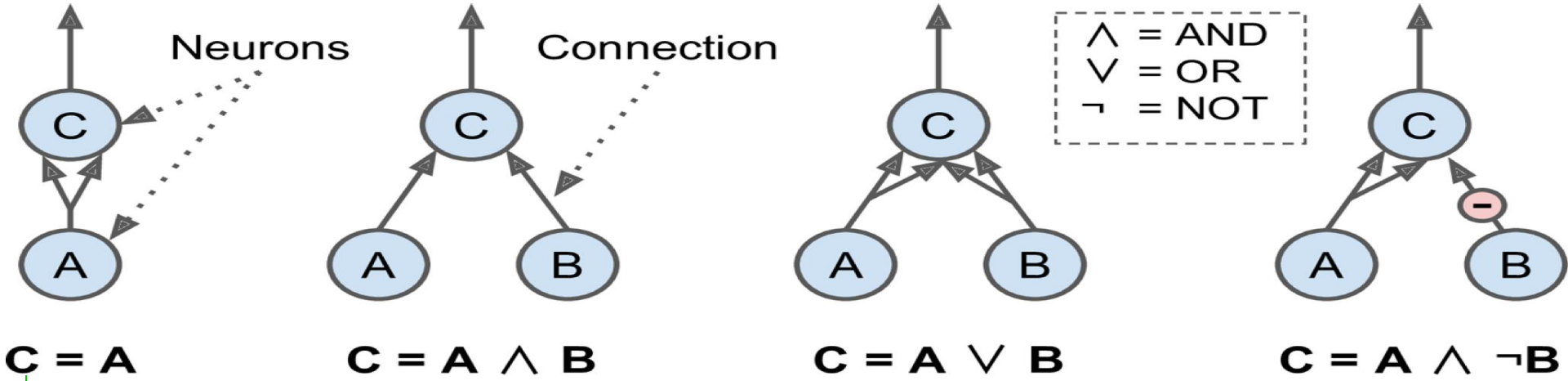
Biological Neurons

- Multiple layers in a biological neural network, human cortex



- **They are organized in a vast network of billions of neurons, each neuron typically connected to thousands of other neurons.**
- Highly complex computations can be performed by a vast network of fairly simple neurons, much like a complex anthill can emerge from the combined efforts of simple ants.
- It seems that neurons are often organized in consecutive layers.

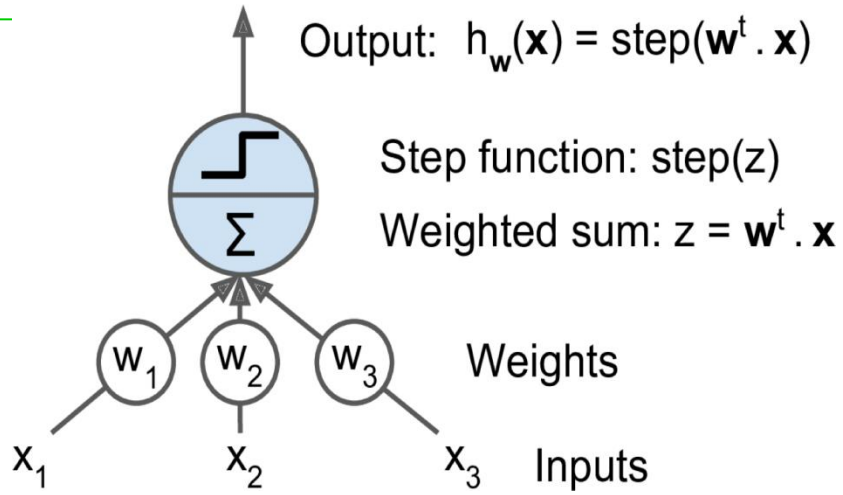
2. Logical Computations with Neurons



- **The first network is simply the identity function: if neuron A is activated, then neuron C gets activated as well； if neuron A is off, then neuron C is off as well.**
- The second performs a logical AND: neuron C is activated only when both neurons A and B are activated
- The third performs a logical OR: neuron C gets activated if either neuron A or neuron B is activated (or both).
- **Imagine: how these networks can be combined to compute complex**
- **logical expressions.**

3. A Linear Threshold Unit(LTU)

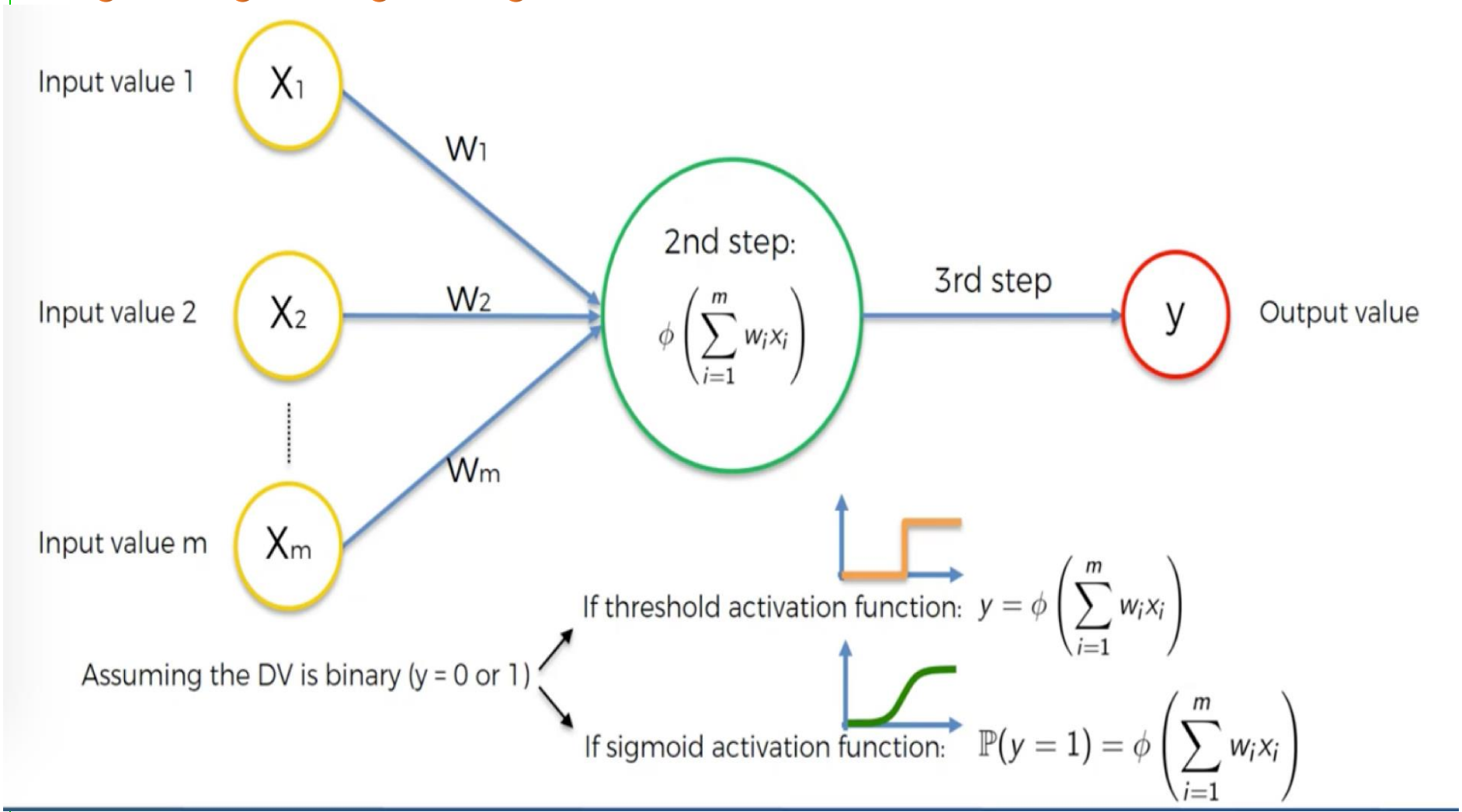
For a single neuro, the inputs and output are any numbers and each input connection is associated with a weight.



- The LTU computes a weighted sum of its inputs (z = w1x1 + w2x2 + ⋯ + wnxn = wT . x), then applies an activate function, say "step function ", to that sum z.

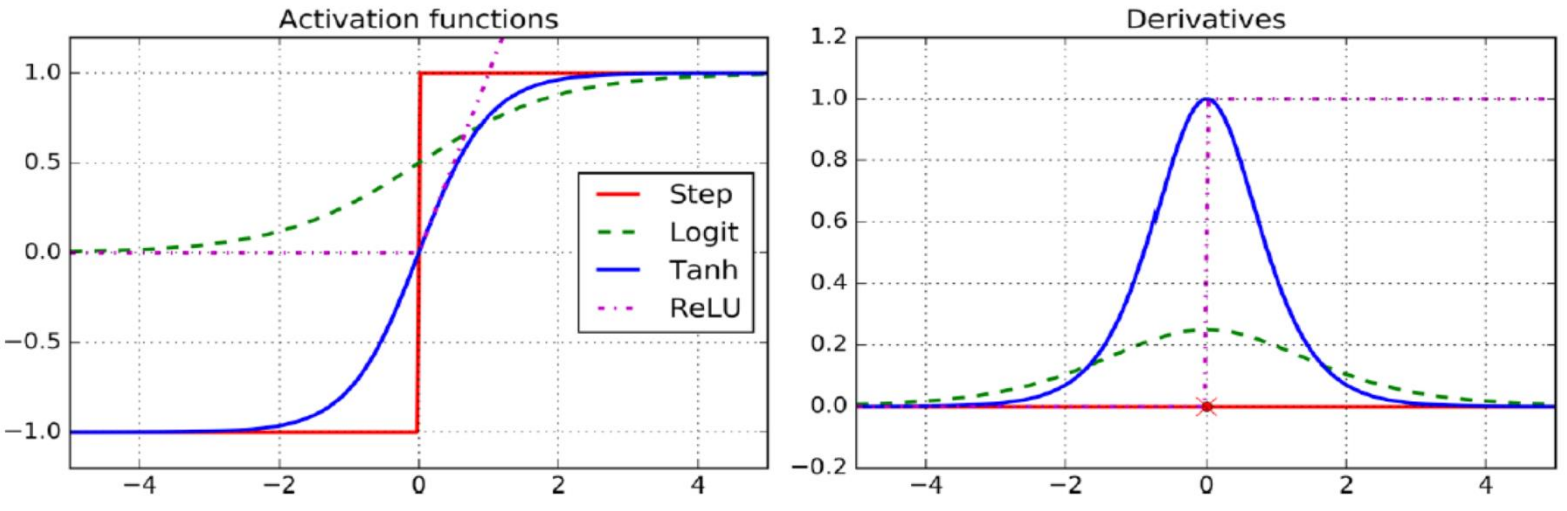
- It outputs the result: hw(x) = step (z) = step (wT .x).

An example of activation functions



4. Typical Activation Functions

- Let's see four activation functions, (such as "Step function" early) and their derivatives:

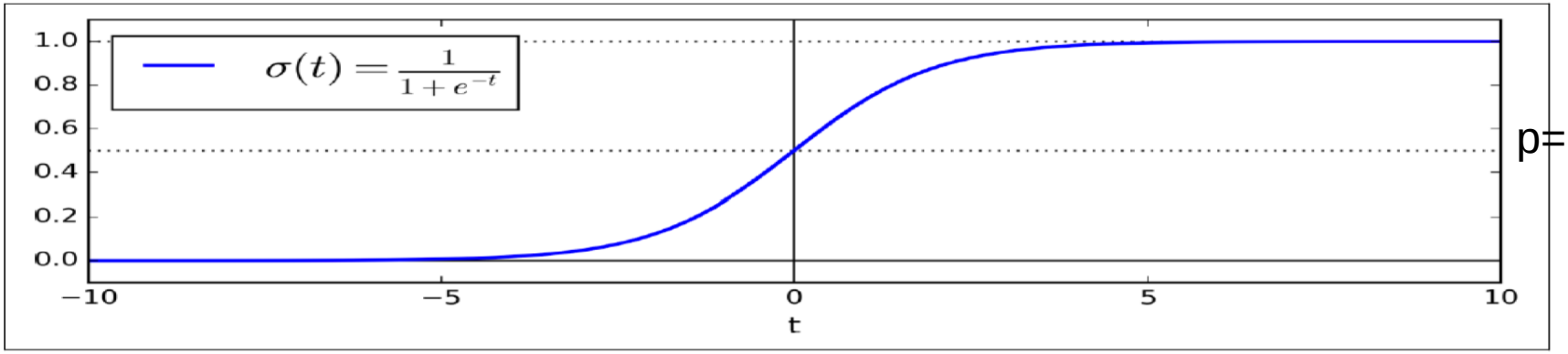


- 1). Step function or Threshold function (solid red)

the step function contains only flat segments, so there is no gradient to work with.

$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

2). Logistic function or sigmoid function (dashed green)

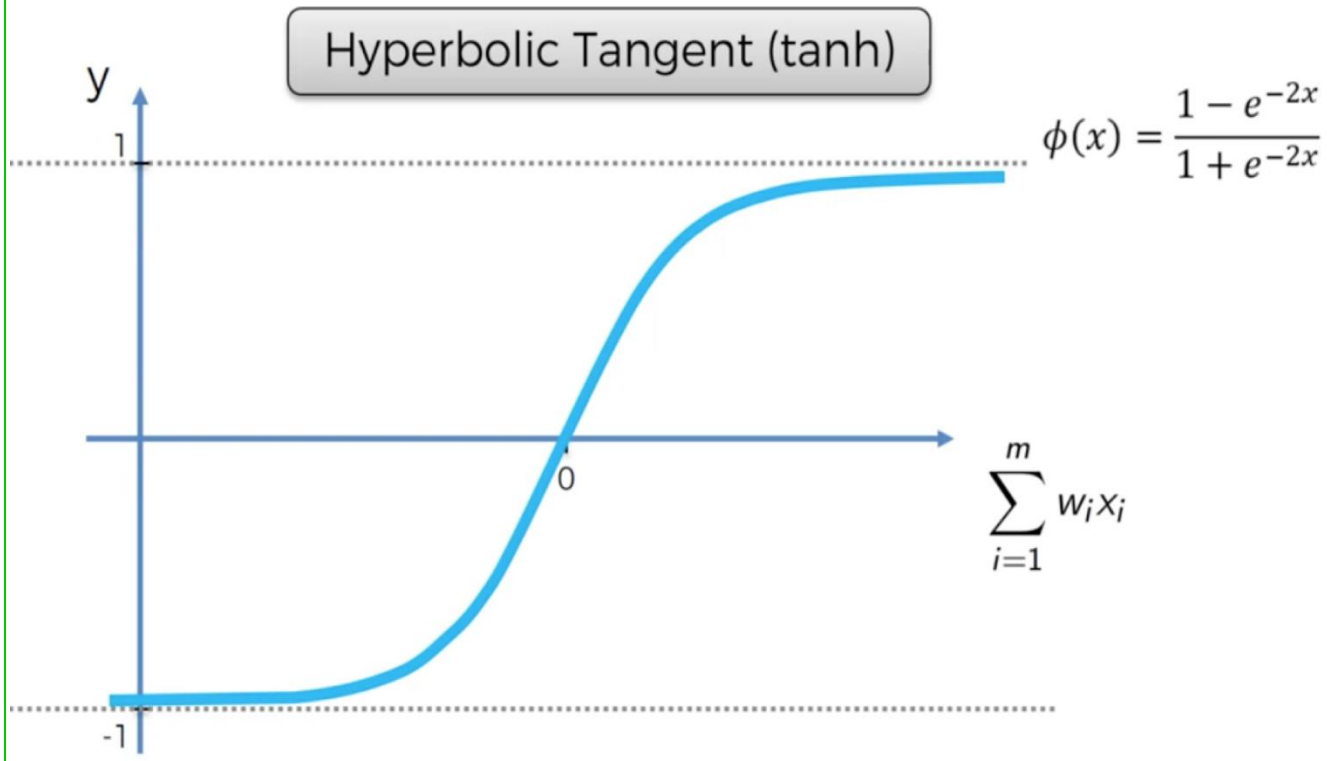


p= σ (t)

- the logistic function has a well-defined nonzero derivative everywhere, allowing Gradient Descent to make some progress at every step
- Its output values, p , from 0 to 1:
  - If t = 0, p = 0
  - If t > 0, 1> p >0.5
  - If t < 0, 0.5 > p > 0
- **Here, t represents z in Step function, as discussed in Session 3.**

3). The hyperbolic tangent function

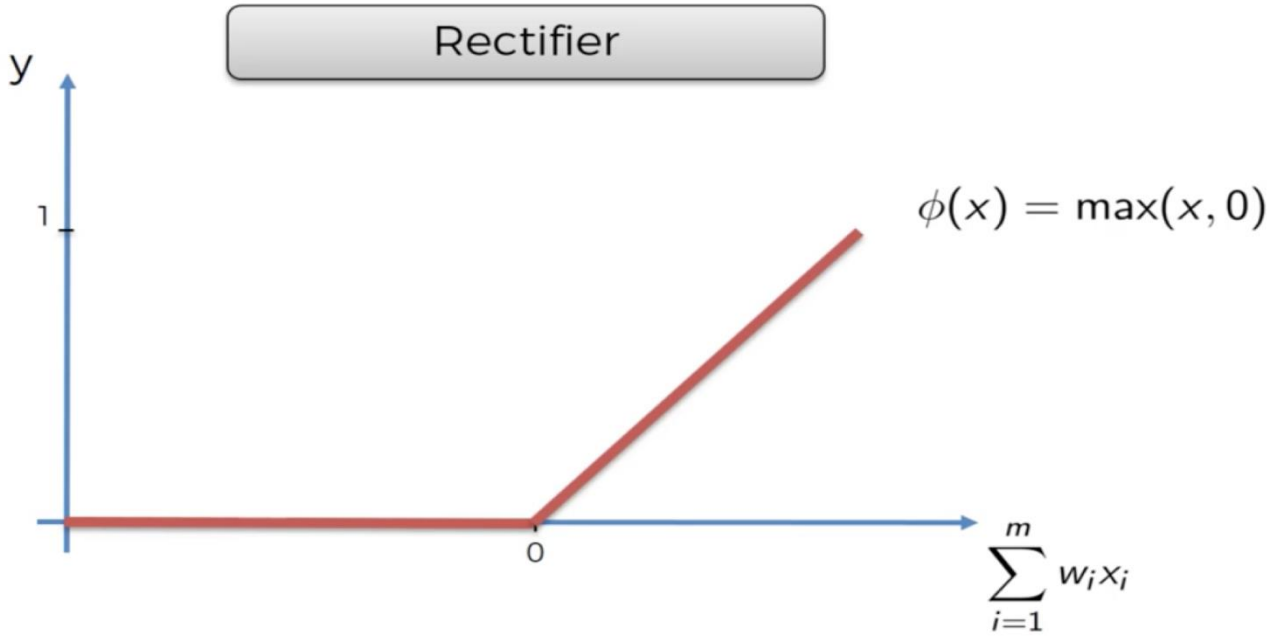
- **Just like the logistic function it is S-shaped, continuous, and differentiable, but its output value ranges from –1 to 1, which tends to make each layer's output more or less normalized (i.e., centered around 0) at the beginning of training. This often helps speed up convergence.**



$\tanh(z) = 2\sigma(2z) - 1$

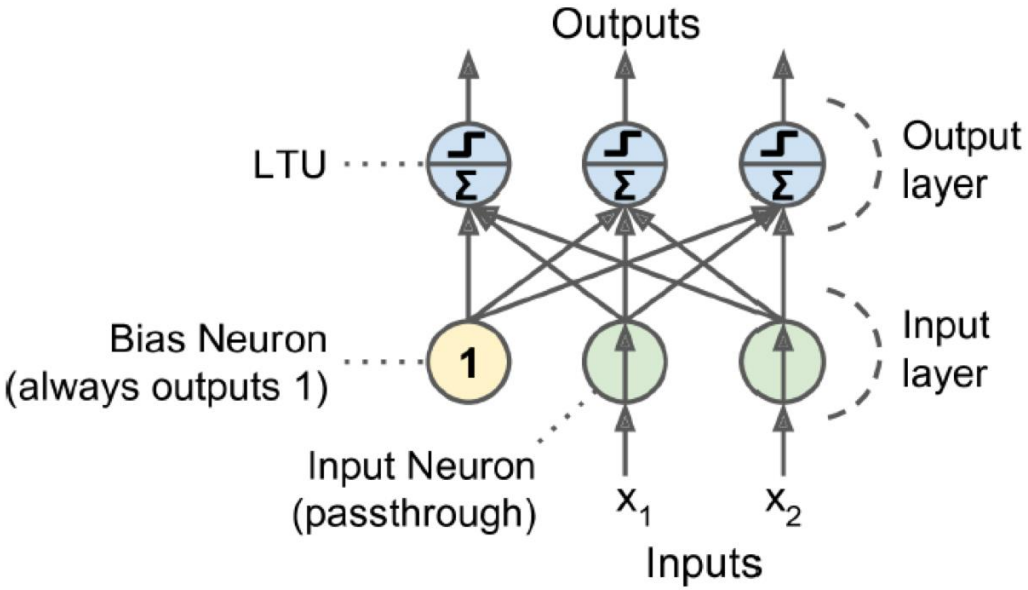
4). The ReLU (Rectified Linear Units) function (dotted red)

- A ReLU computes a linear function of the inputs, and outputs the result if it is positive, and 0 otherwise. ReLU (z) = max (0, z).
- It is continuous but unfortunately not differentiable at z = 0.
- However, in practice it works very well and has the advantage of being fast to compute. Most importantly, the fact that it does not have a maximum output value also helps reduce some issues during Gradient Descent.



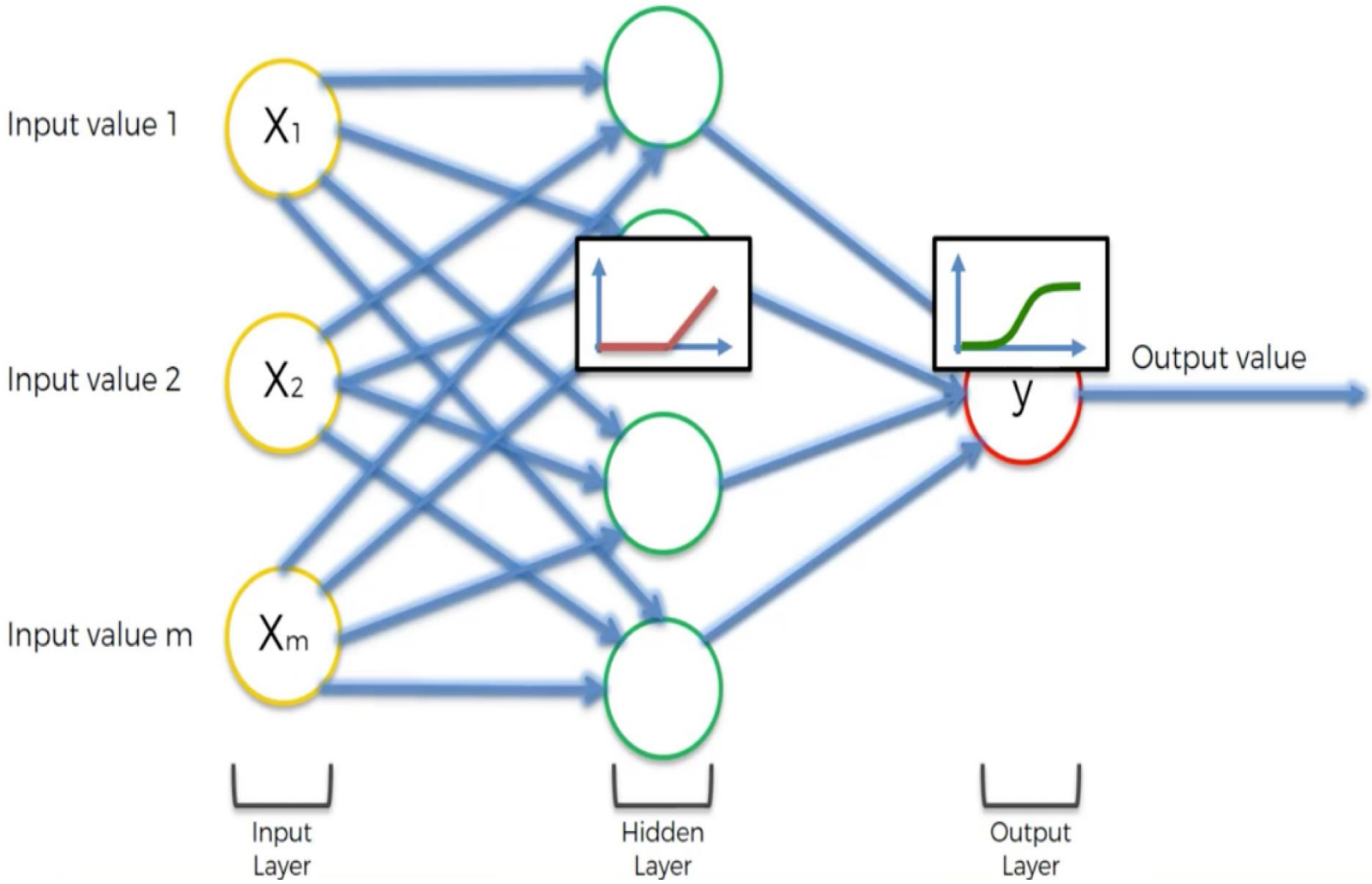
5. Perceptron

- A Perceptron is composed of a single layer of LTUs, with each neuron connected to all the inputs.



- **These connections are often represented using special pass through neurons called input neurons: they just output whatever input they are fed.**
- **An extra bias feature is generally added (x0 = 1), typically represented using a special type of neuron called a bias neuron, which just outputs 1 all the time.**
- **This perceptron can classify instances simultaneously into three different binary classes, which makes it a multioutput classifier.**

Perceptron


Input value 1
Input value 2
Input value m
X_1
X_2
X_m
y
Output value
Input Layer
Hidden Layer
Output Layer

6. Perceptron learning rule

- How is a Perceptron trained?
- More specifically, the Perceptron is fed one training instance at a time, and for each instance it makes its predictions or outputs.
- For every output neuron that produced a wrong prediction, it reinforces the connection weights from the inputs that would have contributed to the correct prediction.
- It means that such weights should be update. How?
- The rule is shown in the following equation (Perceptron learning rule for weight update):

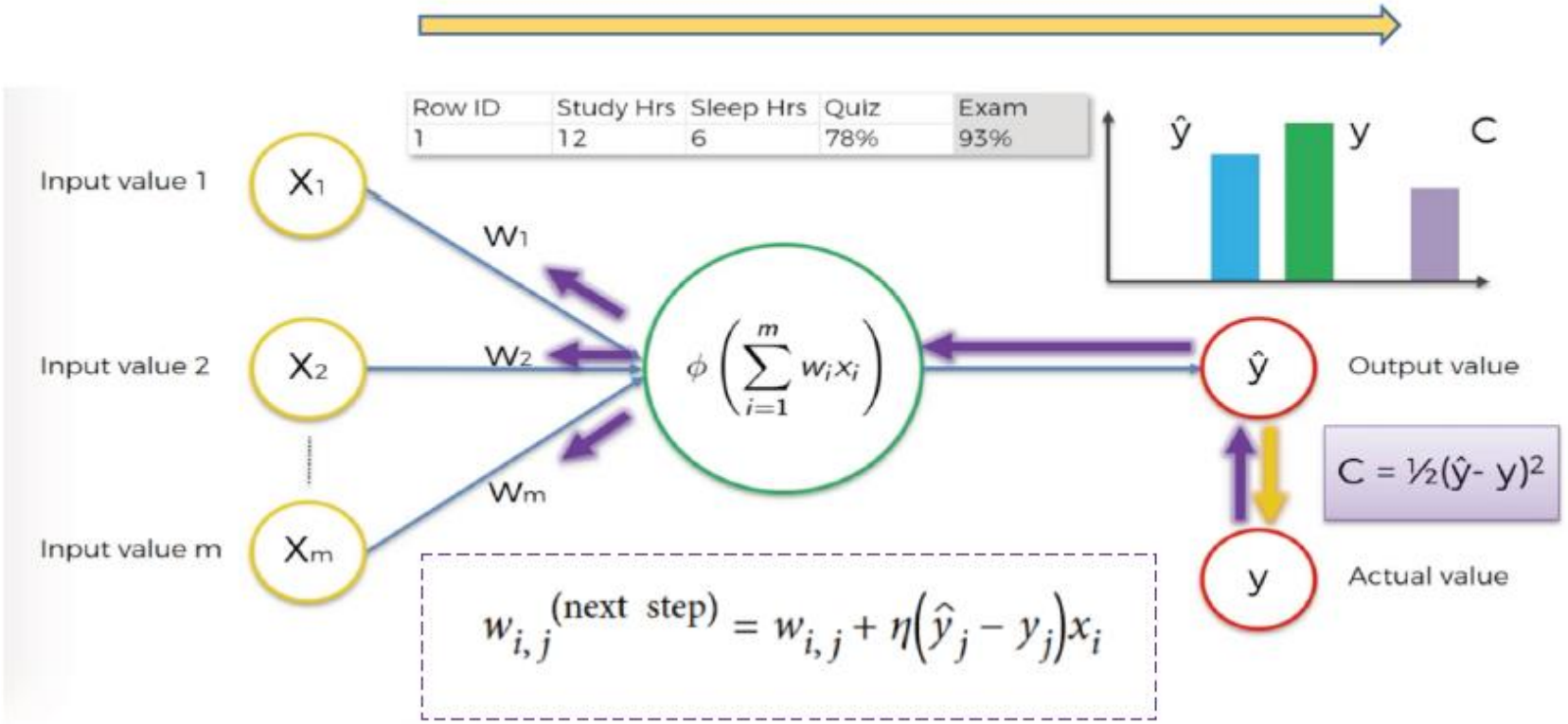
$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta\left(\hat{y}_j - y_j\right)x_i$$

- $w_{i,j}$ is the connection weight between the i^{th} input neuron and the j^{th} output neuron.
- x_i is the i^{th} input value of the current training instance.
- $\hat{y}_j$ is the output of the j^{th} output neuron for the current training instance.
- y_j is the target output of the j^{th} output neuron for the current training instance.
- η is the learning rate.

- The decision boundary of each output neuron is linear, so Perceptions are incapable of learning complex patterns (just like Logistic Regression classifiers).
- However, if the training instances are linearly separable, Rosenblatt demonstrated that this algorithm would converge to a solution.
- This is called the Perceptron convergence theorem.
- Scikit-Learn provides a Perceptron class that implements a single LTU network.
- See the sample in Lab time.

Perceptron

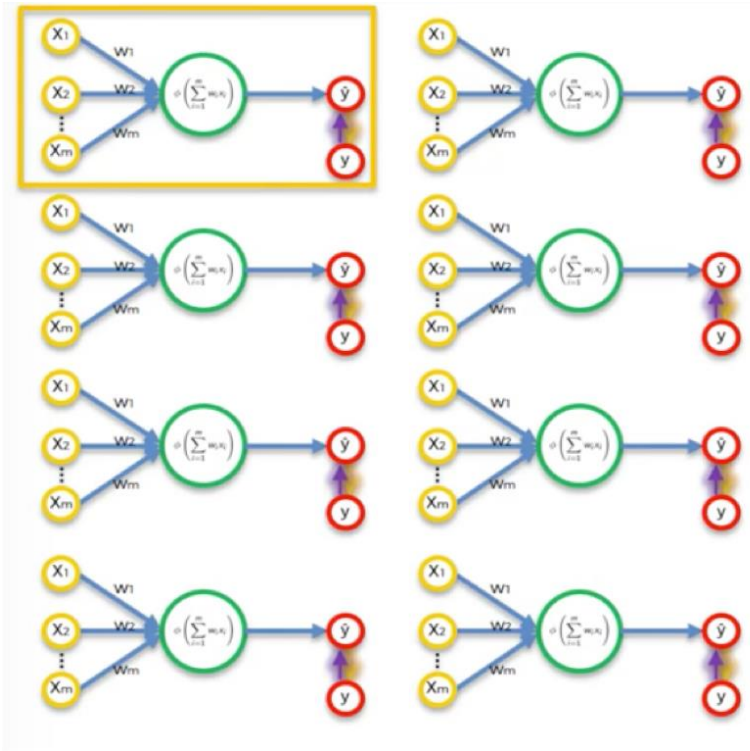
For Example:



- or other cost function
- or using Gradient Decent, rather than this rule

Perceptron

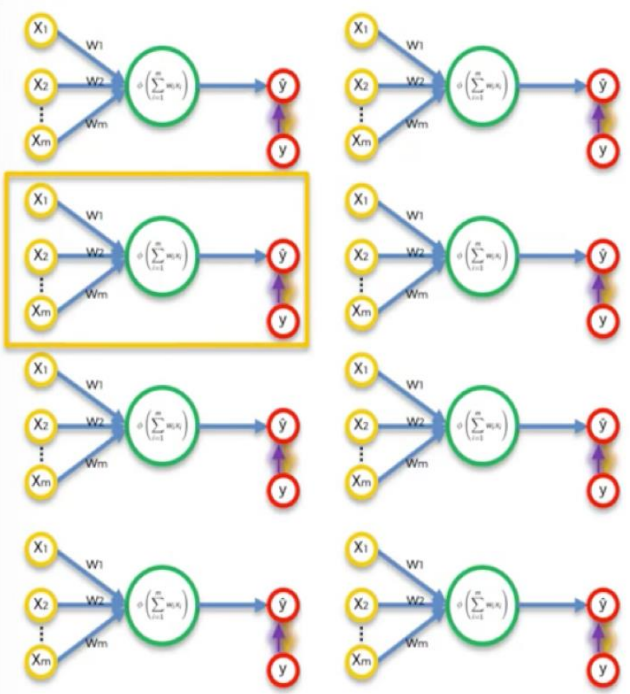
- Step 1:



Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

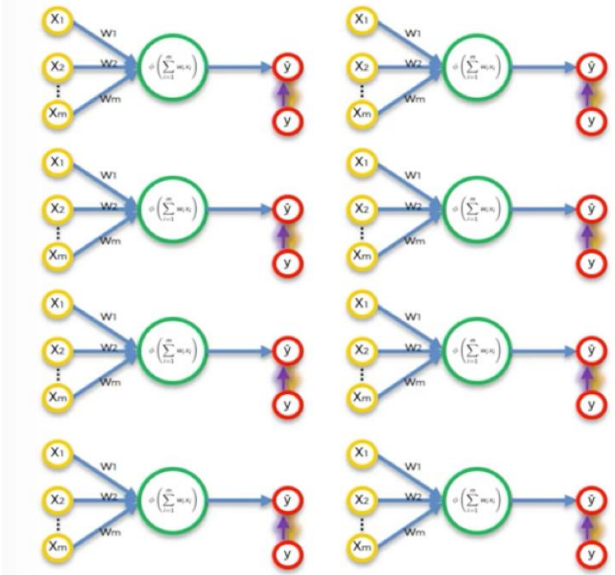
- Predict the exam score of a student, using the dataset collected in history

Step 2:



Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

repeatedly

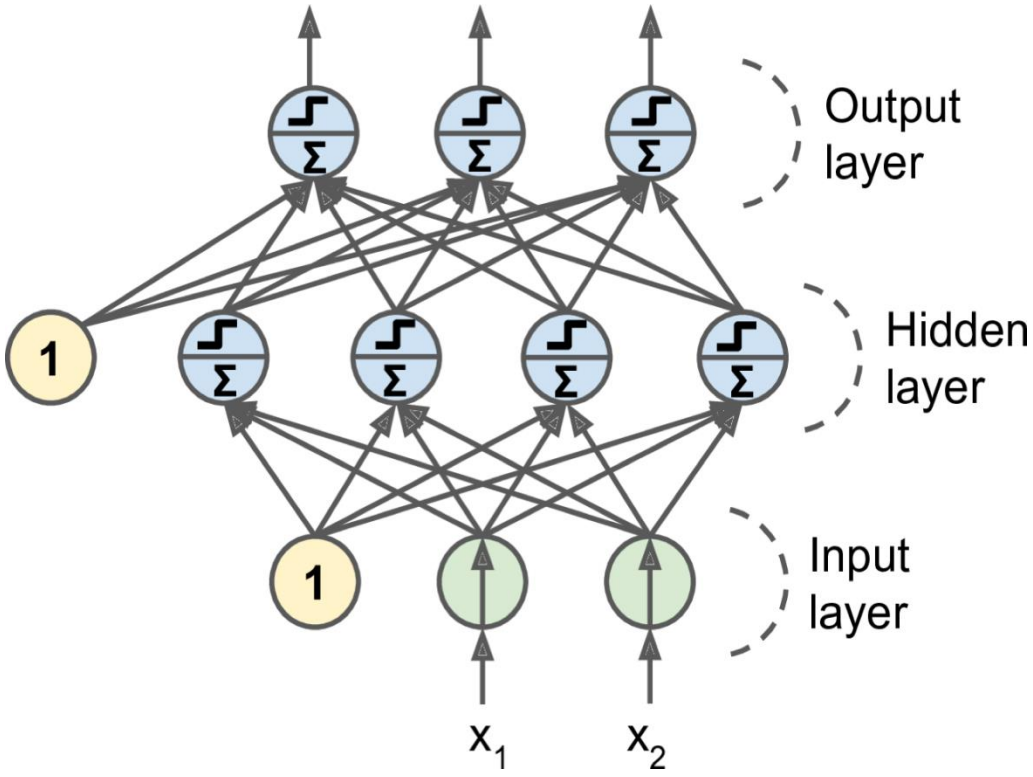


Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

$C = \sum \frac{1}{2}(\hat{y} - y)^2$

7. Multi-Layer Perceptron (MLP) and Backpropagation

- An MLP is composed of one input layer, one or more hidden layers and one final layer of LTUs called the output layer.
- They are fully connected to the next layer. When an ANN has two or more hidden layers, it is called a Deep Neural Network (DNN).



Multi-Layer Perceptron (MLP)

- For many years researchers struggled to find a way to train MLPs, without success. But in 1986, D. E. Rumelhart et al. published a groundbreaking article introducing the backpropagation training algorithm.

- Today we would describe it as Gradient Descent using reverse-mode autodiff (Gradient Descent and autodiff were discussed in previous Chapters).

Multi-Layer Perceptron (MLP)

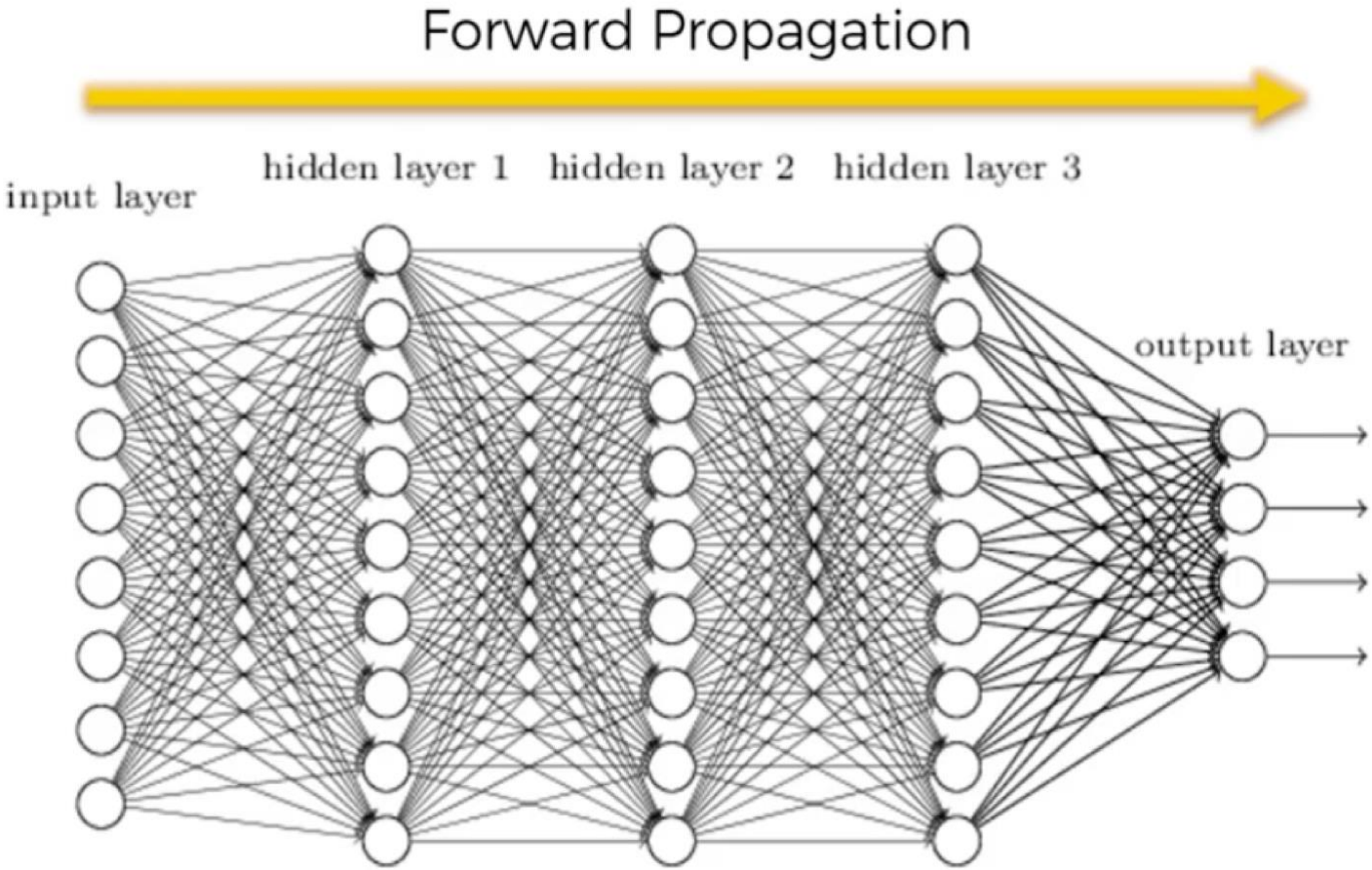


Image Source: neuralnetworksanddeeplearning.com

Multi-Layer Perceptron (MLP)

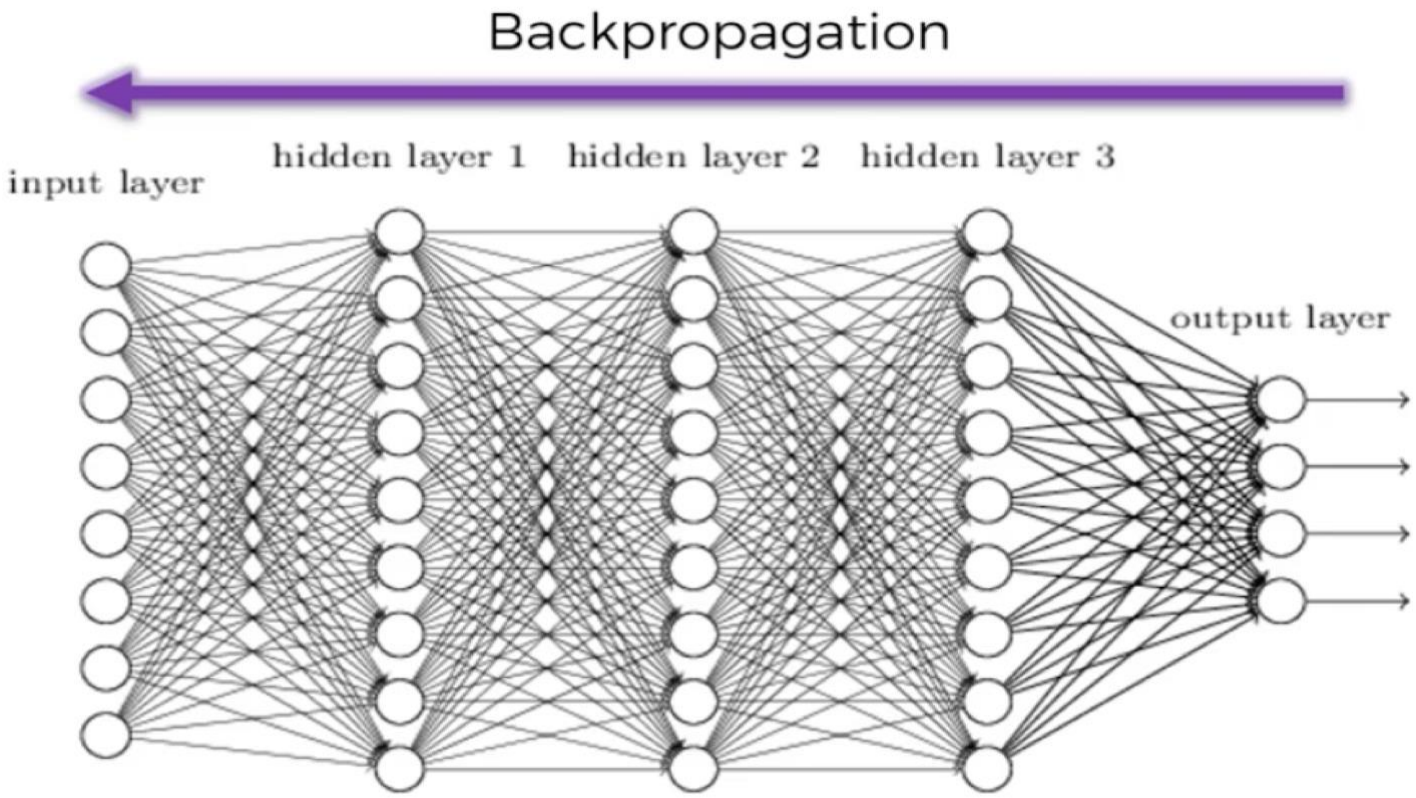


Image Source: neuralnetworksanddeeplearning.com

8. Training an ANN with Stochastic Gradient Decent

- 1) Randomly initialize the weights to small numbers close to 0 (but not 0).
- 2) Input the first instance of the dataset in the input layer; each feature in one input node (or neuro)
- 3) Forward-propagation: from left to right, the neuros are activated in a way that impact of each neuro's activation is limited by the weights. Propagate the activations until getting the predicted result y.

8. Training an ANN with Stochastic Gradient Decent

- 4) Compare the predicted result to the actual results; Measure the generated error.

- 5) Back-Propagation: from right to left. The error is back-propagated. Update the weights according to how much they responsible for the error. The learning rate decided by how much we update the weights.

- 6). Repeat steps 1) to 5) and update the weights after each instance (“ reinforcement learning”). Or: repeat steps 1) to 5), but update the weights only after a batch of instances (“batch learning”)

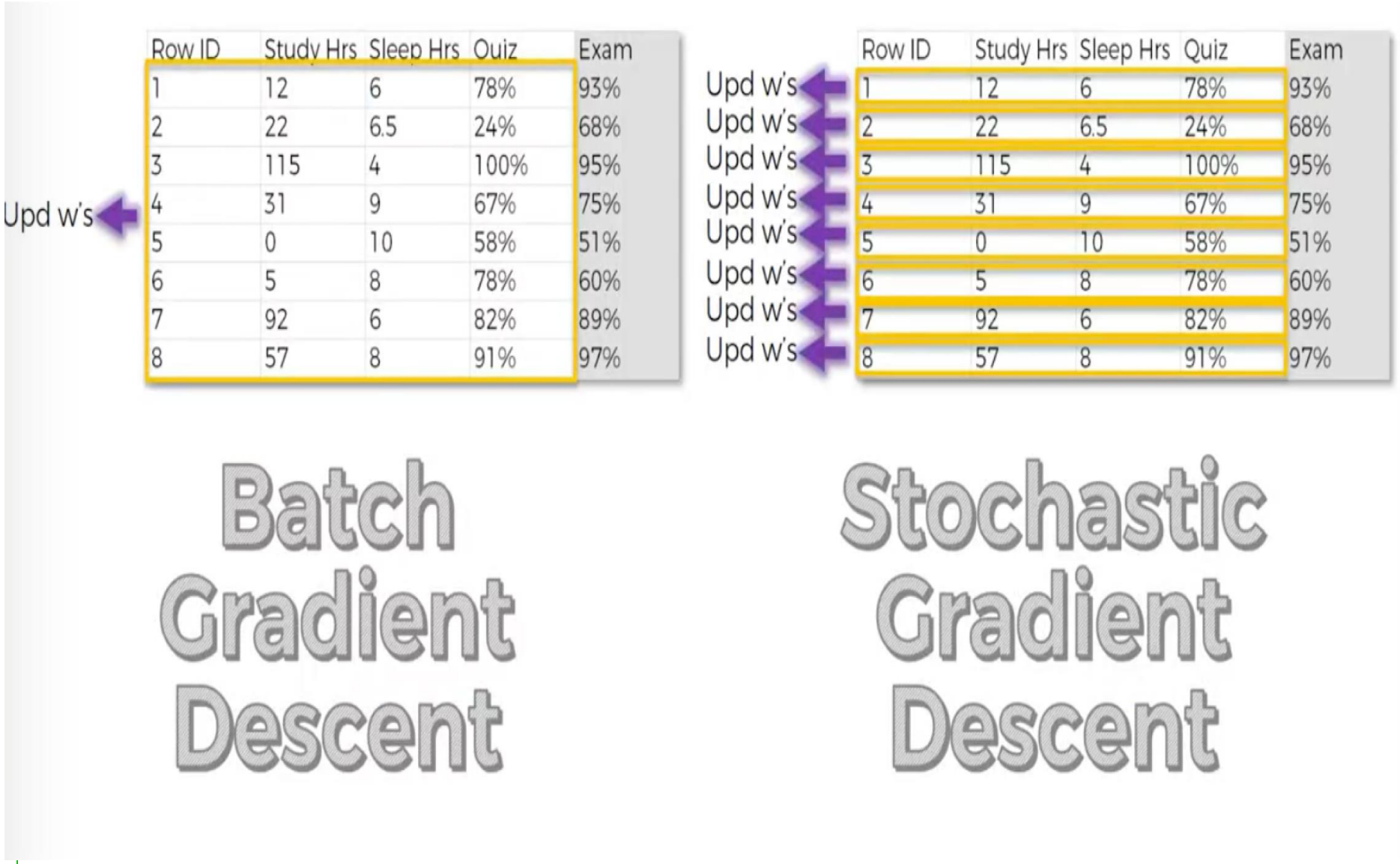
8. Training an ANN with Stochastic Gradient Decent

- 7.) When the whole training set passed through the ANN, that makes an epoch. Redo more epochs.

- **Remarks:**
- During the reverse pass, slightly modifies the connection weights to reduce the error, where Stochastic gradient decent are applied.

- The data set can be divided into “batches” to reduce the computation.

- One important thing here is that back propagation is an advanced algorithm driven by very interesting and sophisticated mathematics which allows us to adjust the weights. All the weights can be adjusted simultaneously (say, by parallel computing on cloud).



One of them may be used for updating the weights.

内容提要

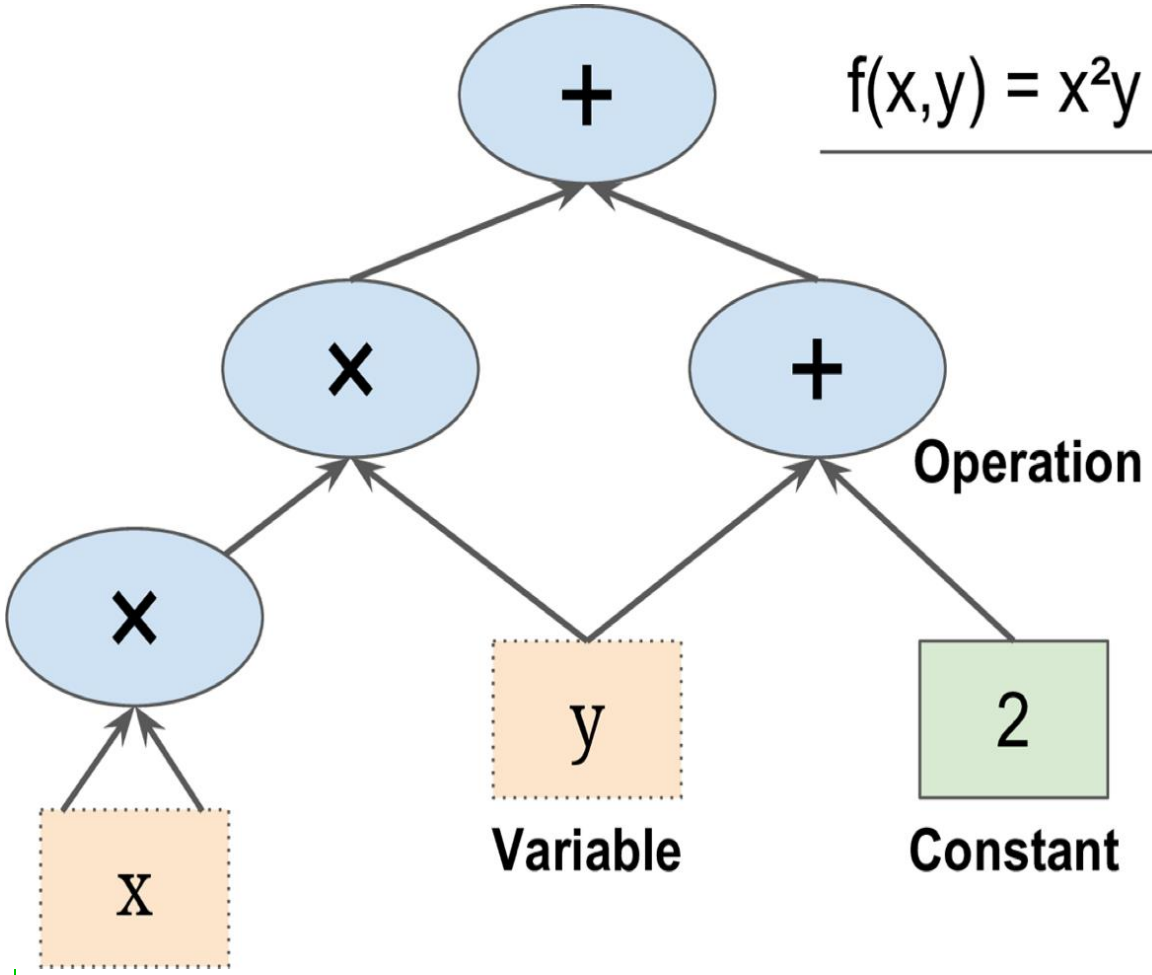
Chapter 12 Neural Networks

Chapter 13 Building ANNs with Tensorflow and Keras

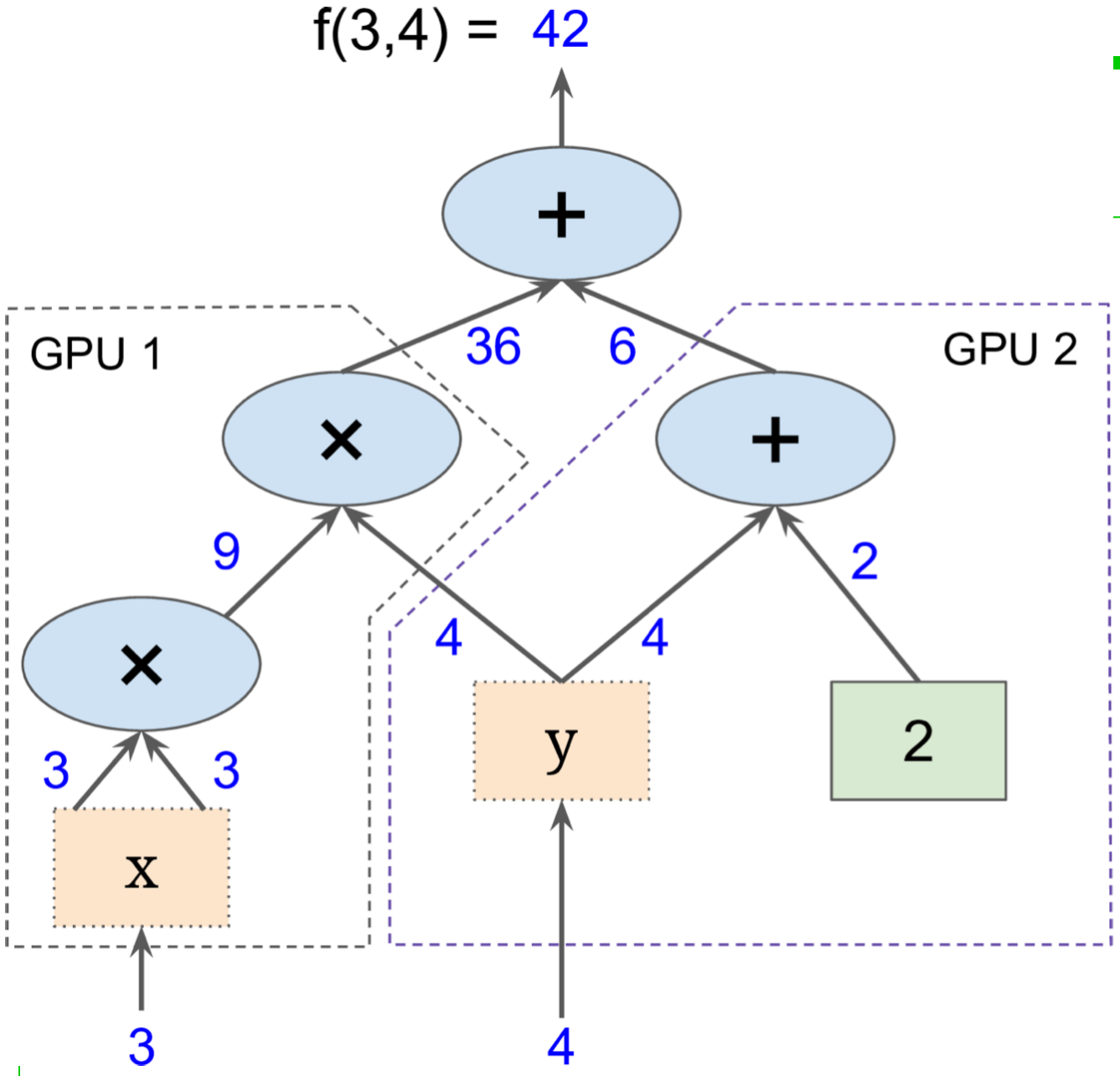
Chapter 13 Building ANNs with Tensorflow and Keras

- **TensorFlow**
- TensorFlow is a powerful open source software library for numerical computation, particularly well suited and fine-tuned for large-scale Machine Learning and neural networks.
- Originally developed by Google (2015)

- Its basic principle is simple:
  - (i). define in Python a graph of computations to perform and then
  - (ii). Run the graph.
- Most importantly, it is able to break up the graph into several chunks and run them in parallel across multiple CPUs or GPUs



Simple Computation Graph



Parallel Computing
on multiple CPUs
and
Servers

TensorFlow

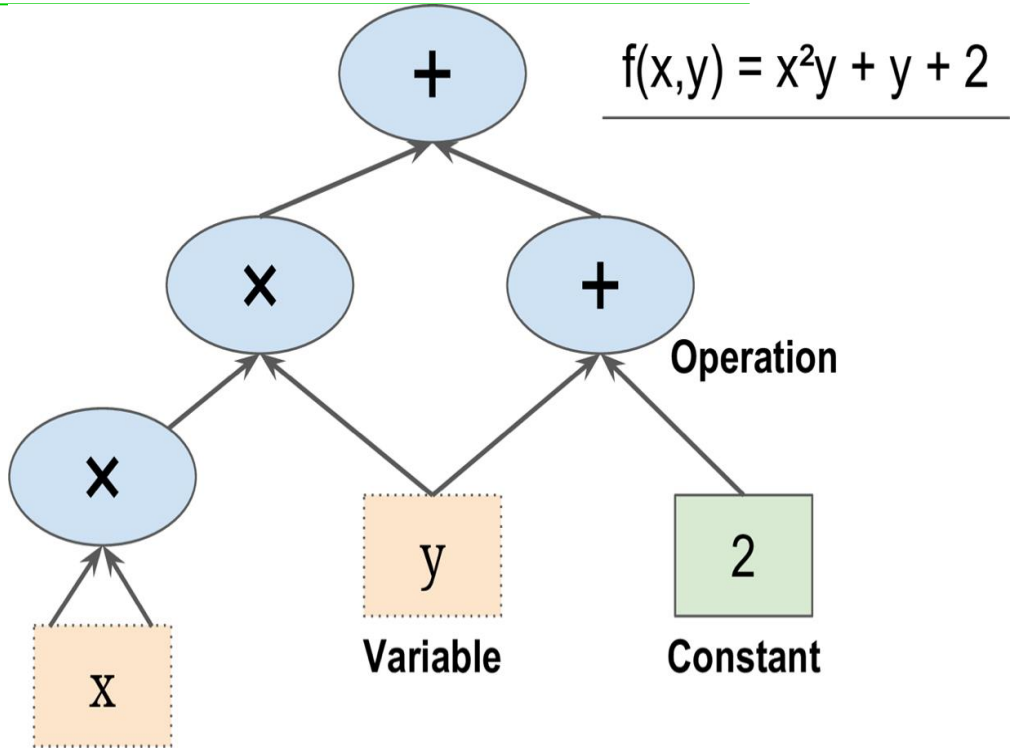
- TensorFlow supports distributed computing, able to train colossal neural networks on humongous training sets in a reasonable amount of time by splitting the computations across hundreds of servers.
- TensorFlow can train a network with millions of parameters on a training set composed of billions of instances with millions of features each.
- Since TensorFlow was developed, it powers many of Google's large-scale services, such as Google Cloud Speech, Google Photos, and Google Search

Table 9-1. Open source Deep Learning libraries (not an exhaustive list)

Library	API	Platforms	Started by	Year
Caffe	Python, C++, Matlab	Linux, macOS, Windows	Y. Jia, UC Berkeley (BVLC)	2013
Deeplearning4j	Java, Scala, Clojure	Linux, macOS, Windows, Android	A. Gibson, J.Patterson	2014
H2O	Python, R	Linux, macOS, Windows	H2O.ai	2014
MXNet	Python, C++, others	Linux, macOS, Windows, iOS, Android	DMLC	2015
TensorFlow	Python, C++	Linux, macOS, Windows, iOS, Android	Google	2015
Theano	Python	Linux, macOS, iOS	University of Montreal	2010
Torch	C++, Lua	Linux, macOS, iOS, Android	R. Collobert, K. Kavukcuoglu, C. Farabet	2002

1. Creating A Graph and Running It in a Session

- Creating a graph:
- For example, the codes here create the graph as shown
- Notes:
- this code does not actually perform any computation, even though it looks like it does.
- It just creates a computation graph. In fact, even the variables are not initialized yet.




```
import tensorflow as tf

x = tf.Variable(3, name="x")
y = tf.Variable(4, name="y")
f = x*x*y + y + 2
```

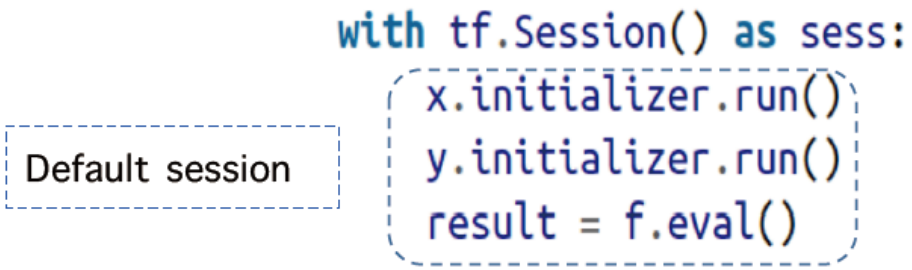
Evaluating a graph

- To evaluate this graph, you need to open a TensorFlow session and use it to initialize the variables and evaluate f.
- A TensorFlow session takes care of placing the operations onto devices such as CPUs and GPUs and running them, and it holds all the variable values.

```
>>> sess = tf.InteractiveSession()
>>> init.run()
>>> result = f.eval()
>>> print(result)
42
>>> sess.close()
```

Evaluating a graph

- There are other ways to evaluate a graph, for examples:



- Instead of manually running the initializer for every single variable, you can use the global_variables_initializer() function. Note that it does not actually perform the initialization immediately, but rather creates a node in the graph that will initialize all variables when it is run:

```
init = tf.global_variables_initializer()  # prepare an init node

with tf.Session() as sess:
    init.run()  # actually initialize all the variables
    result = f.eval()
```

Evaluating a graph

- Inside Jupyter or within a Python shell you may prefer to create an InteractiveSession.
- The only difference is that when an InteractiveSession is created it automatically sets itself as the default session,
- so you don't need a "with" block (but you do need to close the session manually when you are done with it):

```
>>> sess = tf.InteractiveSession()
>>> init.run()
>>> result = f.eval()
>>> print(result)
42
>>> sess.close()
```

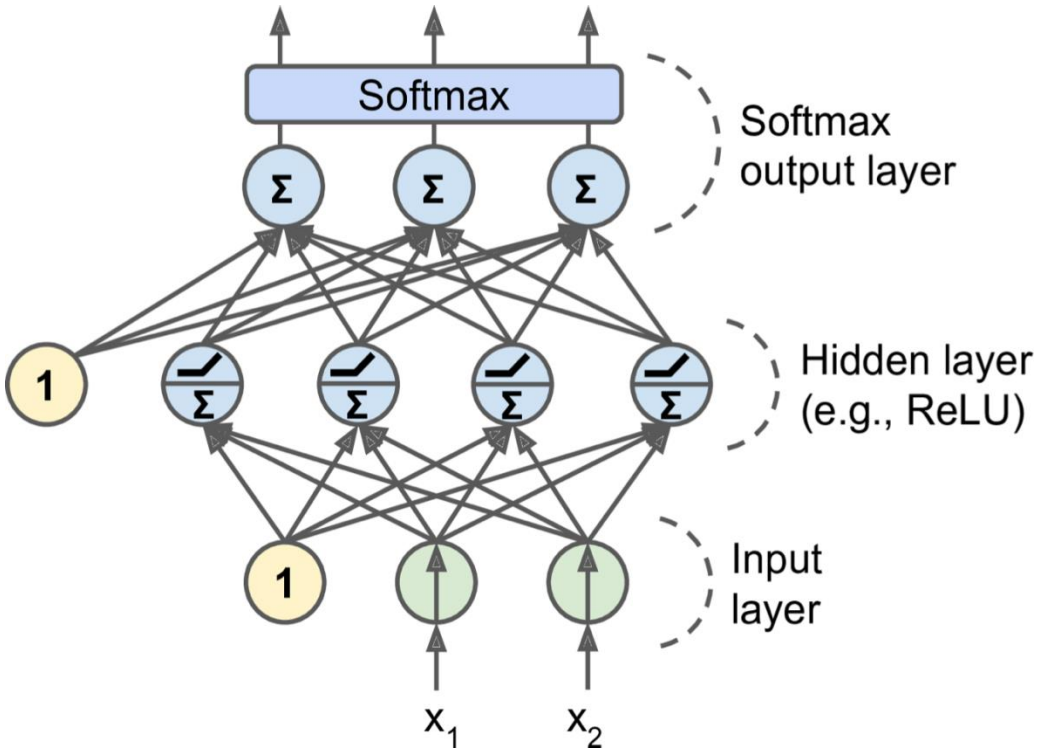
In summary:

- A TensorFlow program is typically split into two parts:
- Computation graph, called “construction phase”, and
- the second part runs, “execution phase”
- The construction phase typically builds a computation graph representing the ML model and the computations required to train it.
- The execution phase generally runs a loop that evaluates a training step repeatedly (for example, one step per mini-batch), gradually improving the model parameters.

- **In the lab class, we will have a example, how to use TensorFlow dealing with Linear Regression.**

Example of MLP's applications

- An MLP is often used for classification, with each output corresponding to a different binary class (e.g., spam/ham, urgent/noturgent, and so on).
- When the classes are exclusive (e.g., classes 0 through 9 for digit image classification, the output layer is typically modified by replacing the individual activation functions by a shared softmax function.



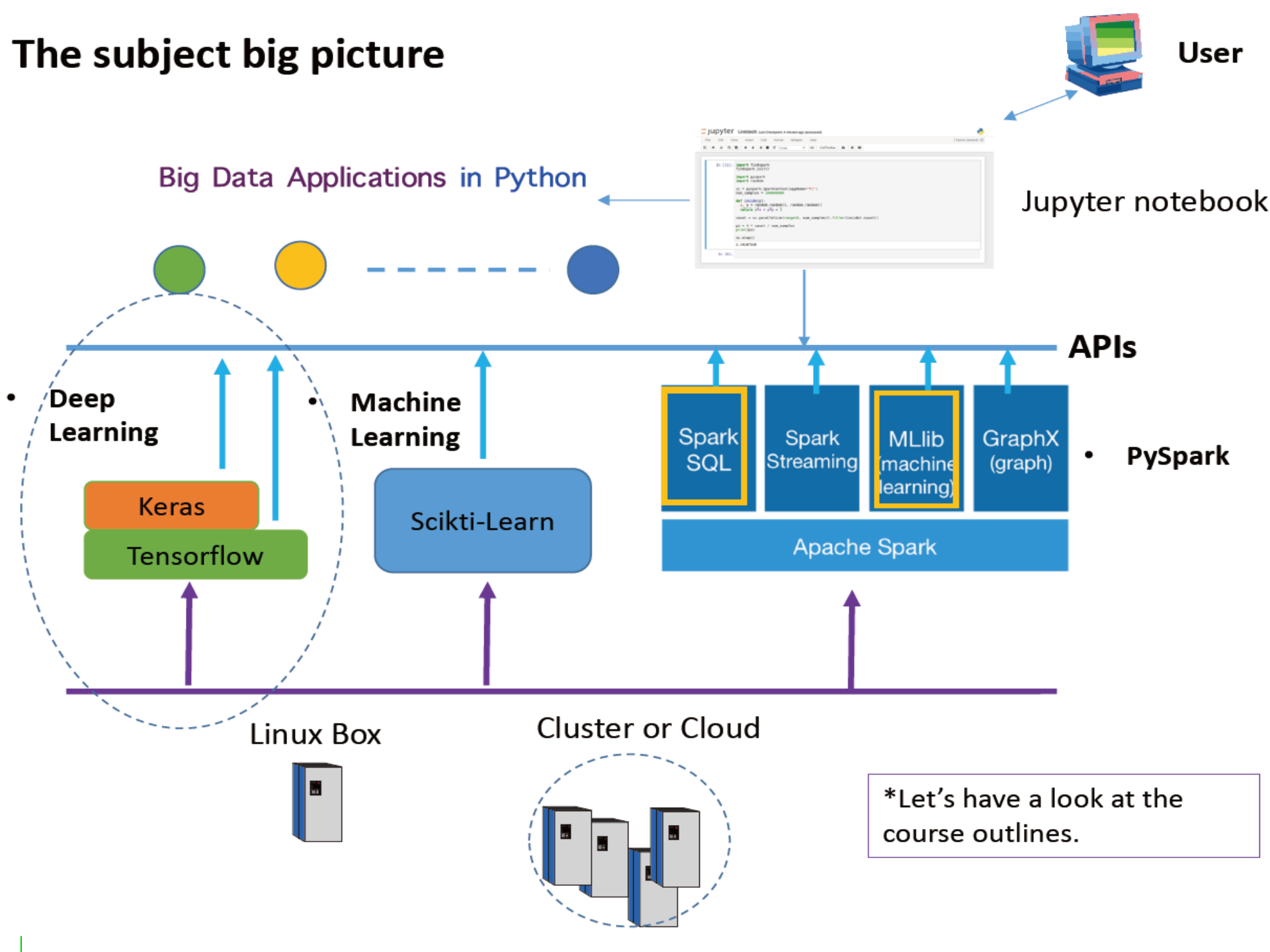
A modern MLP (including ReLU and softmax) for classification

The output of each neuron corresponds to the estimated probability of the corresponding class. Note that 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).

Keras

- Keras is a high-level neural networks API in Python and capable of running on top of TensorFlow, CNTK, or Theano.
- It was developed with a focus on enabling fast experimentation, being able to go from idea to result with the least possible delay. Thus, Keras allows you for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- It supports both convolutional neural networks (CNNs) and recurrent networks (RNNs), as well as combinations of the two, that we will discuss in next two sessions.
- More details presented in "Keras documentation" at https://keras.io
- We will use a sample, ann.ipynb, to demonstrate the basics of programming with Keras in the lab time.

The subject big picture



哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

结束

2020年9月30日