Universidade do Minho
Escola de Engenharia
Departamento de Informática

Mestrado Integrado em Engenharia Informática
Computação Natural
2018/2019

Paulo Novais, Cesar Analide, Filipe Gonçalves

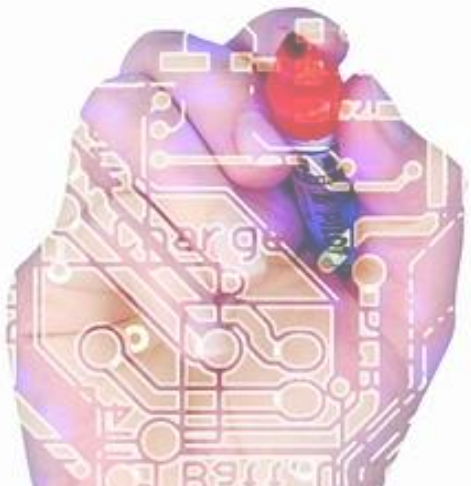
ISLab
Synthetic Intelligence Lab

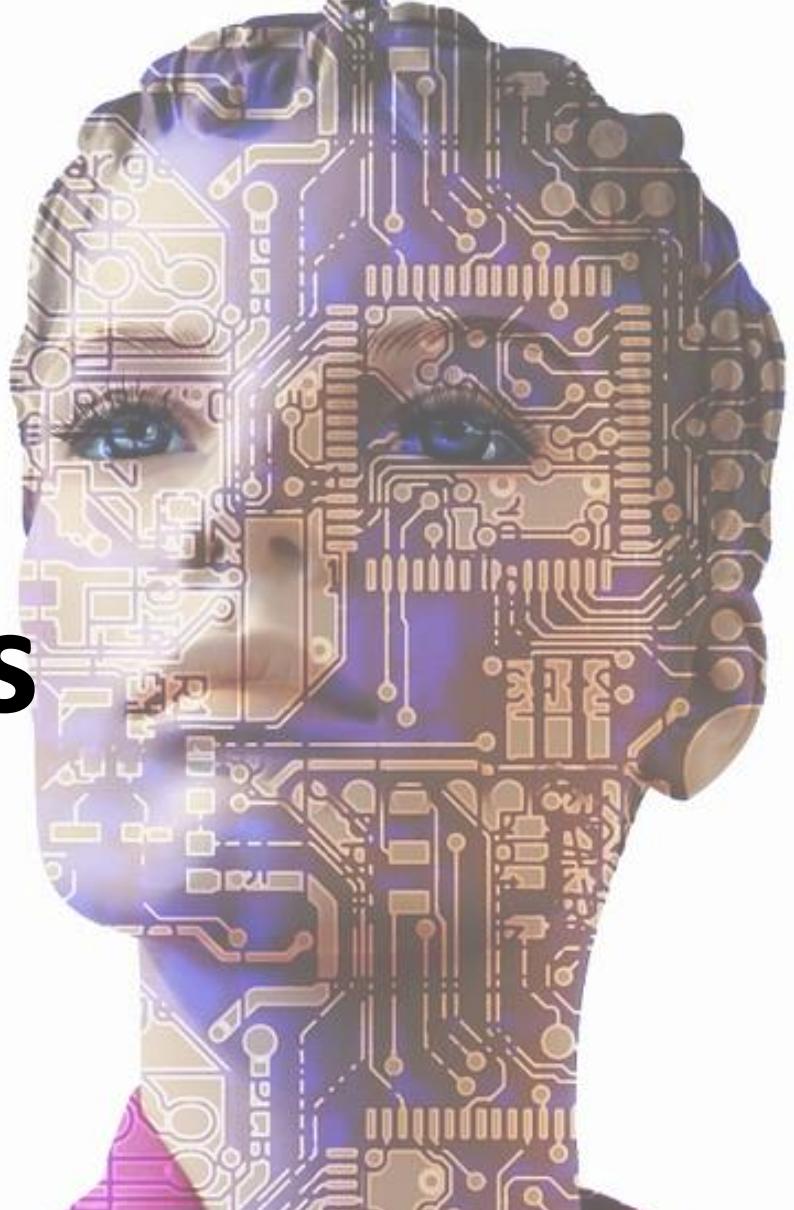
Computação Natural@ 2018/2019

- Paulo Novais – pjon@di.uminho.pt
- Cesar Analide – cesar.analide@di.uminho.pt
- Filipe Gonçalves – fgoncalves@algoritmi.uminho.pt

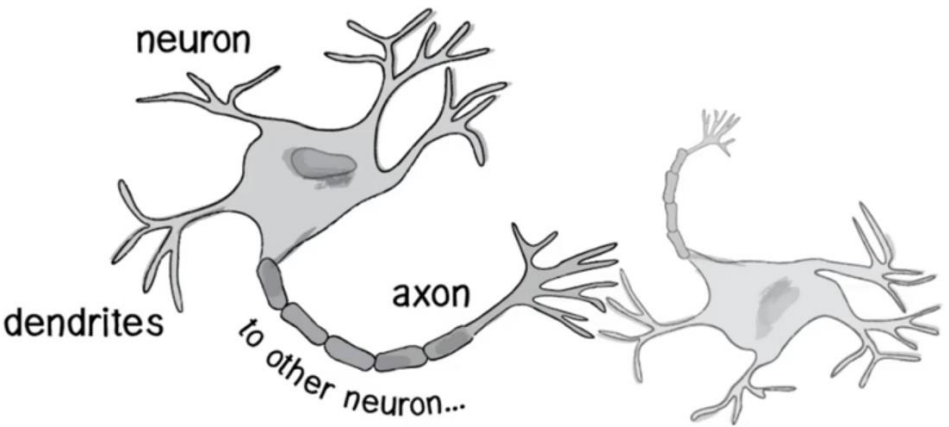
- Departamento de Informática
  Escola de Engenharia
  Universidade do Minho
- Grupo ISLab – (Synthetic Intelligence Lab)
- Centro ALGORITMI
  Universidade do Minho

Introduction to Neural Networks





- Artificial neural networks are modeled after biological neural networks and attempt to allow computers to learn in a similar manner to humans – reinforcement learning.
- The human brain has interconnected neurons that receive inputs, and then based on those inputs, produce and electrical signal output through the axon
- Use cases:
  - Pattern Recognition
  - Time Series Predictions
  - Signal Processing
  - Anomaly Detection
  - Control



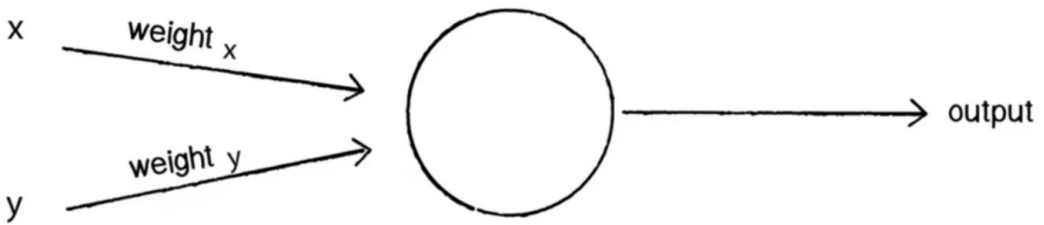
- There are problems that are difficult for humans but easy for computers
  - E.g. calculating large arithmetic problems
- Then there are problems easy for humans, but difficult for computers
  - E.g. recognizing a picture of a person from the side
- Neural networks attempt to solve problems that would normally be easy for humans but hard for computers!
- Let's start by looking at the simples neural network possible – the perceptron

- A perceptron consists of one or more inputs, a processor, and a single output

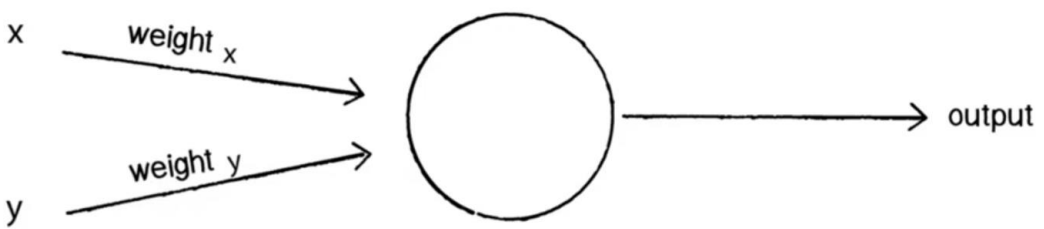
- A perceptron follows the “feed-forward” model, meaning inputs are sent into the neuron, are processed, and result in an output

- A perceptron process follows 4 main tasks:

1. Receive Inputs
2. Weight Inputs
3. Sum Inputs
4. Generate Output

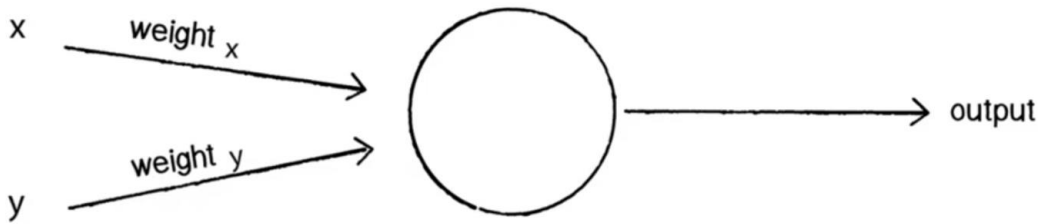


- Say we have a perceptron with two inputs:
  - Input 0: x1 = 12
  - Input 1: x2 = 4
- Each input that is sent into the neuron must first be weighted, i.e. multiplied by some value (often a number between [-1,1])
- When creating a perceptron, we'll typically begin by assigning random weights
  - Weight 0: 0,5
  - Weight 1: -1

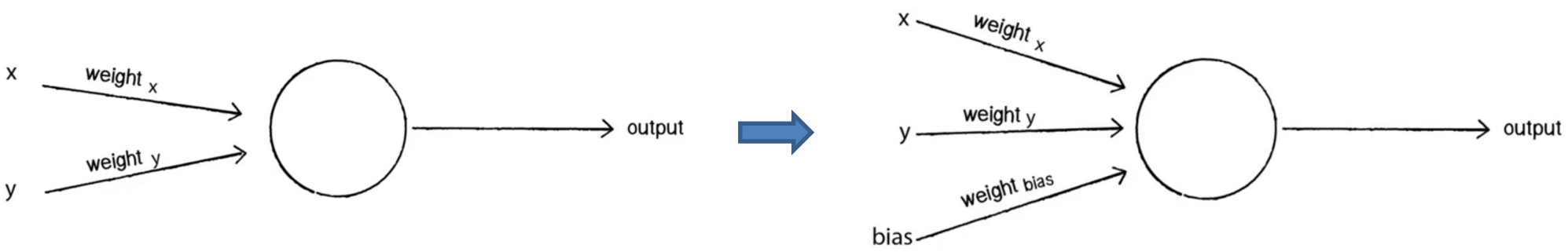


- We take each input and multiply it by its weight
  - Input 0 x Weight 0 = 12 x 0,5 = 6
  - Input 1 x Weight 1 = 4 x (-1) = -4

  → 6+(-4) = 2 [Since 2 >= 0 (binary activation function), perceptron outputs the value 2]
- The output of a perceptron is generated by passing that sum through na **activation function**. In the case of a simple binary output, the activation function is what tells the perceptron whether to "fire" or not
- Many activation functions to choose from (Logistic, Trigonometric, Step, etc..). Let's make the activation function the sign of the sum. In other words: if the sum is a positive number, the output is 1; if it is negative, the output is -1



- One more thing to consider is Bias. Imagine that both inputs were equal to zero, then any sum no matter what multiplative weight would also be zero!
- To avoid this problem, we add a third input known as a bias input with a value of 1. This avoids the zero issue!



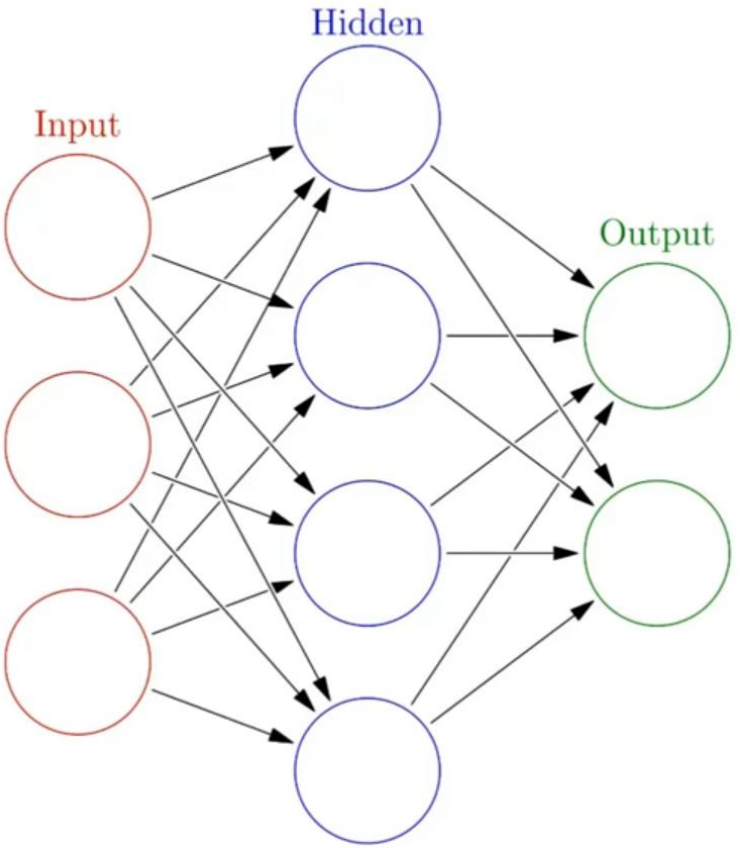
- To actually train the perceptron we use the following steps:

1. Provide the perceptron with inputs for which there is a known answer
2. Ask the perceptron to guess an answer
3. Compute the error (how far off from the correct answer?)
4. Adjust all the weights according to the error
5. Return to Step1 and repeat!

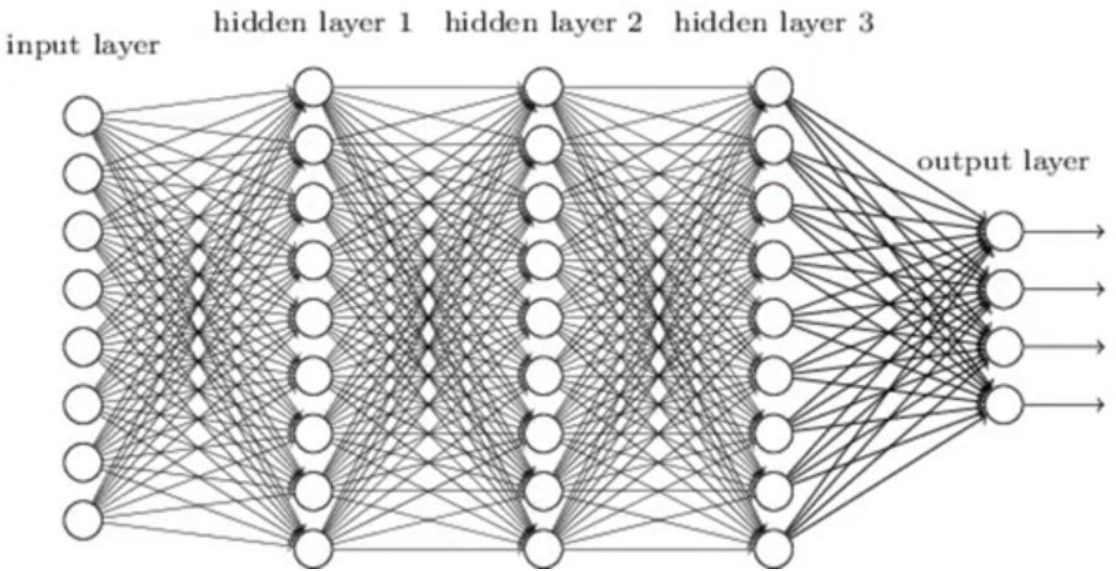
- We repeat this until we reach an error we are satisfied with (we set this before hand)

- That is how a single perceptron would work, now to create a neural network all you have to do is link many perceptrons together in layers!

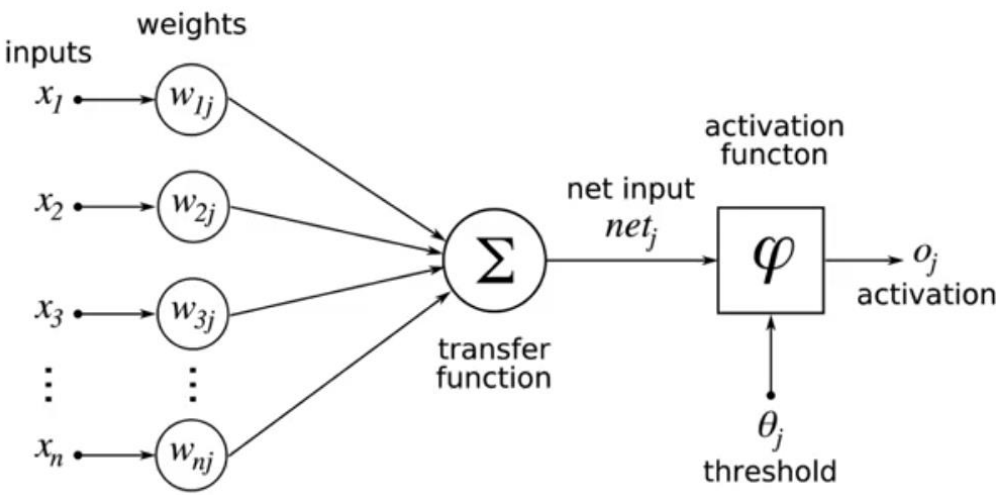
- You'll have na input layer and an output layer
- Any layers in between are known as hidden layers, because you don't directly "see" anything but the input or output
  - Debugging tools are useful to analyse info. between layers

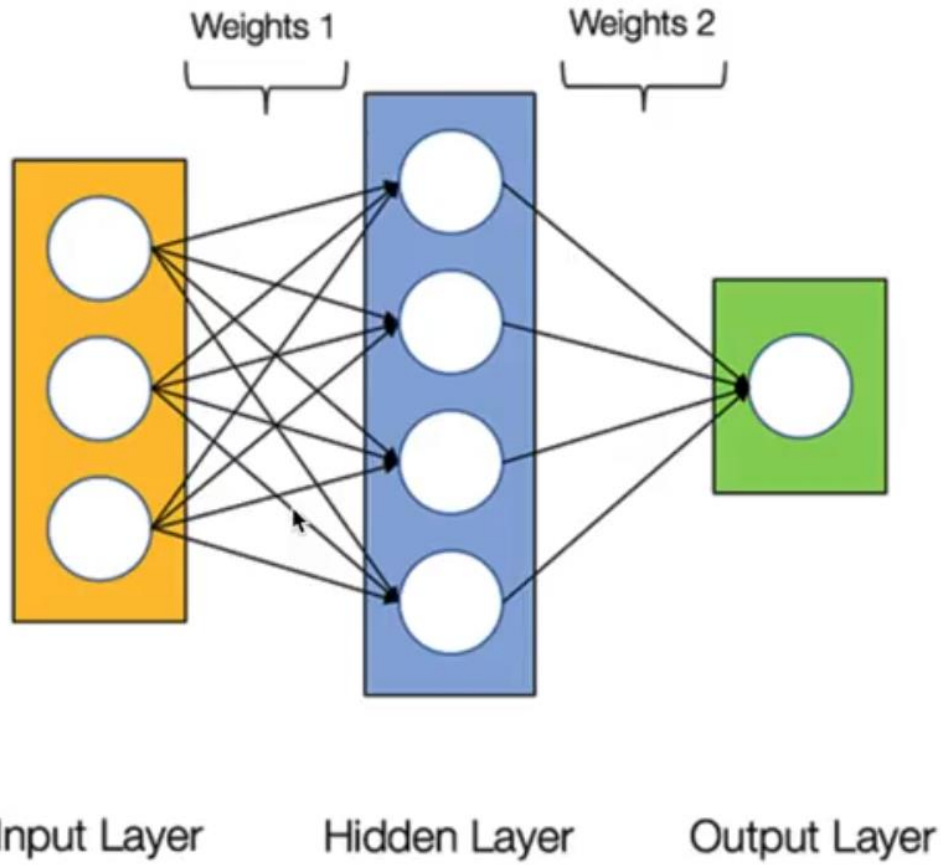


- You may have heard of the term “Deep Learning”
- Deep Learning is a neural network with many hidden layers, causing it to be “deep”
  - E.g. Microsoft’s state of the art vision recognition uses 152 layers



- In review: Neural Networks are composed by:
  - Input Layer
  - Hidden Layer
  - Output Layer
  - Weights and Biases between Layers
  - Activation Function

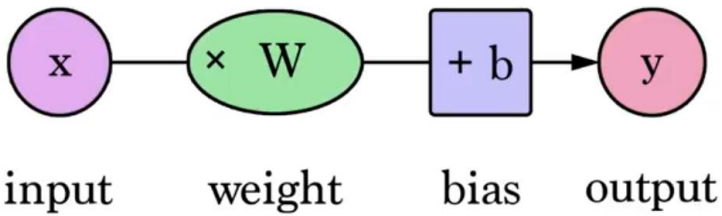



Weights 1
Weights 2
Input Layer
Hidden Layer
Output Layer

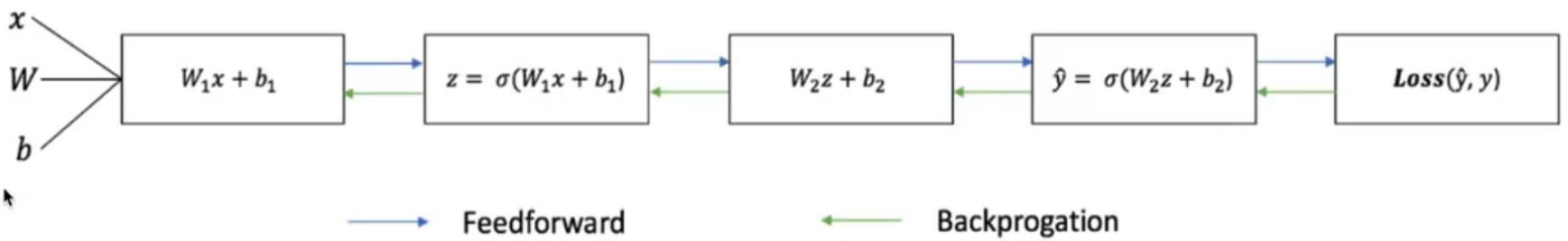
- Neural Network Training
  - Output is designated by the following function:

$$Y = \sigma(W_2\sigma(W_1x+b_1)+b_2)$$

    - Weights are represented by W_1 and W_2
    - Biases represented by b_1 and b_2



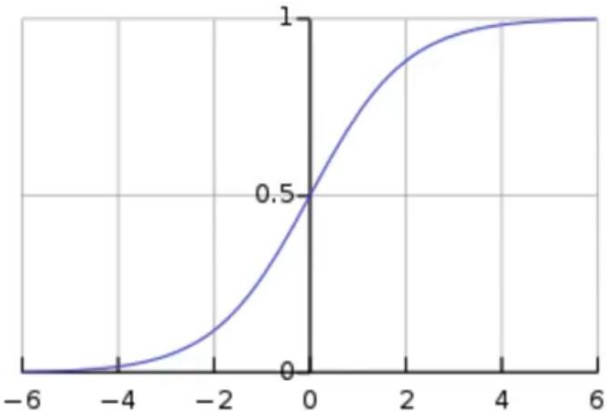
- Two steps in training:
  - Feedforward
  - Backpropagation



- Feedforward Implementation

```
def feedforward(self):
    self.layer1 = sigmoid(np.dot(self.input, self.weights1))
    self.output = sigmoid(np.dot(self.layer1, self.weights2))
```

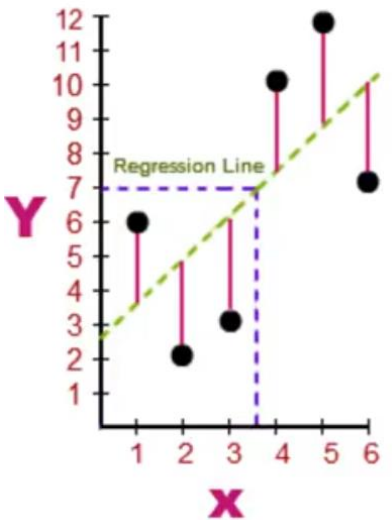
- Sigmoid Activation Function



- Loss Function
  - Many diferente possibilities
  - We will choose: Sum of Squares Error

$$\sum_{i=1}^{n}(y - \hat{y})^2$$

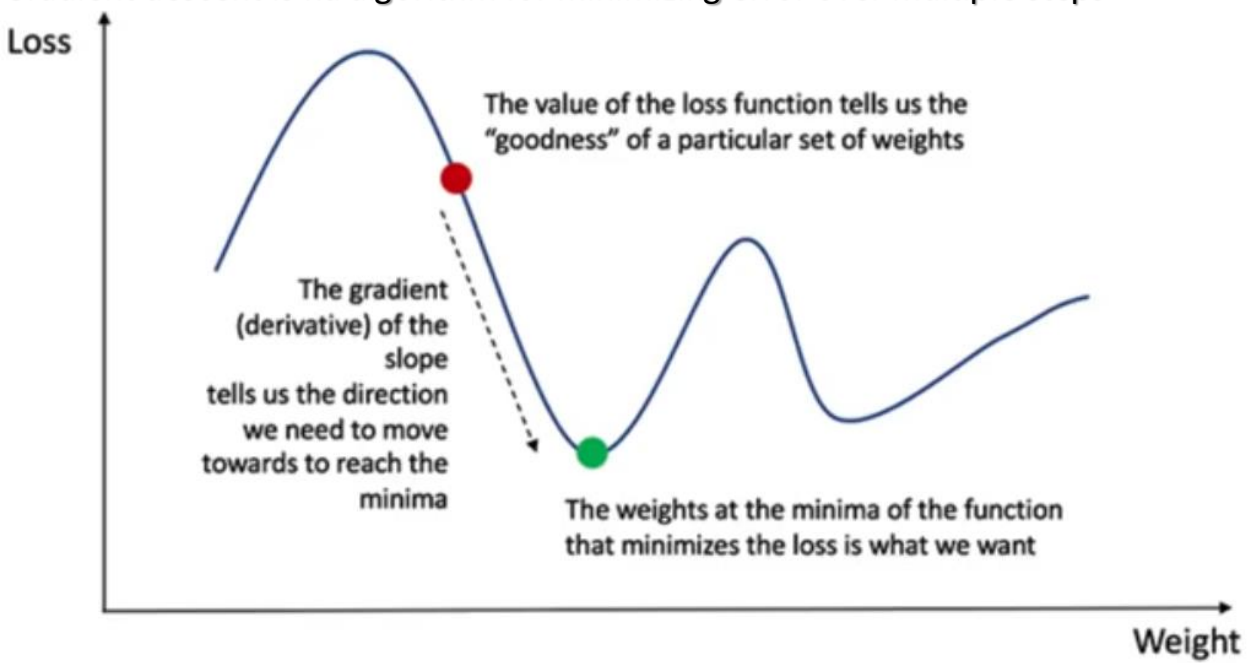
  - **Goal: Minimize the Loss Function**



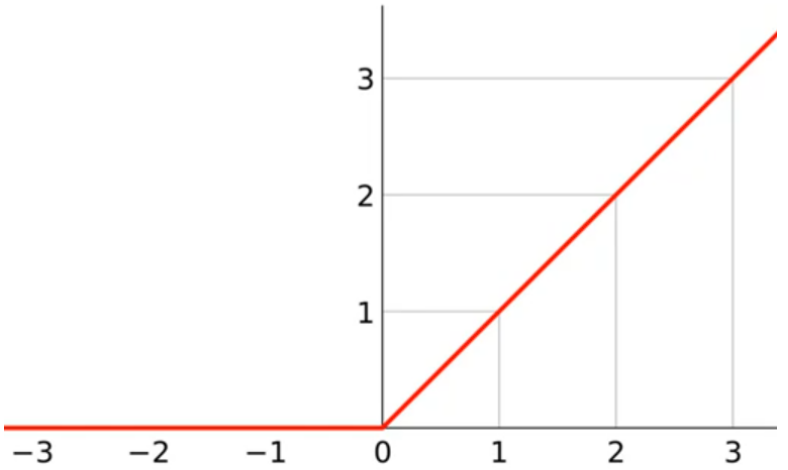
- Backpropagation
  - Propagate the error back and update our weights and biases
  - Finds the extent to which we need to adjust the weights and biases

```
def backprop(self):

    d_weights2 = np.dot(self.layer1.T, (2*(self.y - self.output) *
sigmoid_derivative(self.output)))

    d_weights1 = np.dot(self.input.T,  (np.dot(2*(self.y - self.output) *
sigmoid_derivative(self.output), self.weights2.T) * sigmoid_derivative(self.layer1)))

    self.weights1 += d_weights1

    self.weights2 += d_weights2
```

- Gradient Descent
  - Derivative of the Loss Function with respect to weights and biases
  - Gradient descent is na algorithm for minimizing error over multiple steps



- Activation functions (aka rectifier)
  - Step functions don't work with gradient descent
  - Mathematically, they have no useful derivative
- Alternatives
  - Logistic function
  - Hyperbolic Tangente function
  - Exponential Linear Unit (ELU)
  - ReLU function (Rectified Linear Unit)
- ReLU is common. Fast to compute and works well
  - Also: "Leaky ReLU", "Noisy ReLU"
  - ELU can sometimes lead to faster learning though



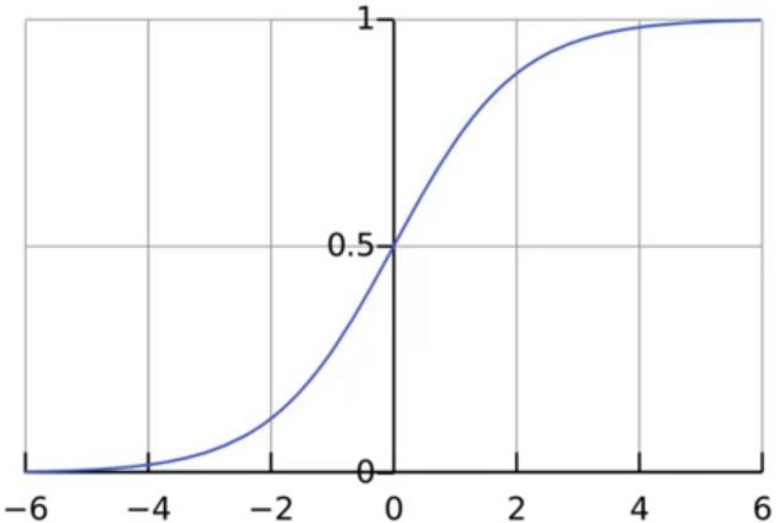
ReLU function

softmax

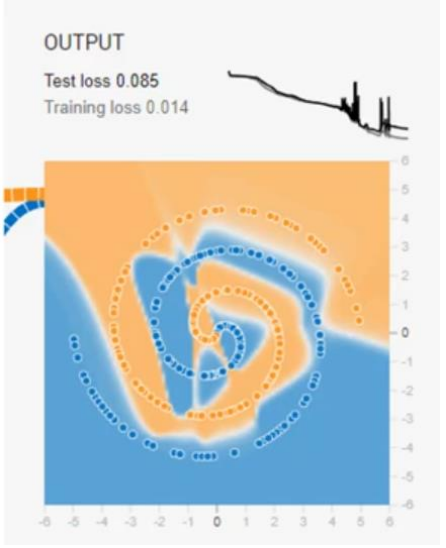
- Used for classification
  - Chooses the most probable classification given several input values
  - It produces a probability for each class
  - The class with the highest probability is the “answer” you get

$$h_\theta(x) = \frac{1}{1 + \exp(-\theta^T x)},$$

x is a vector of input values
theta is a vector of weights

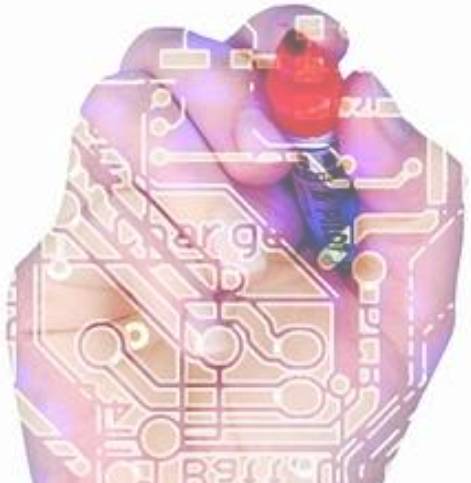


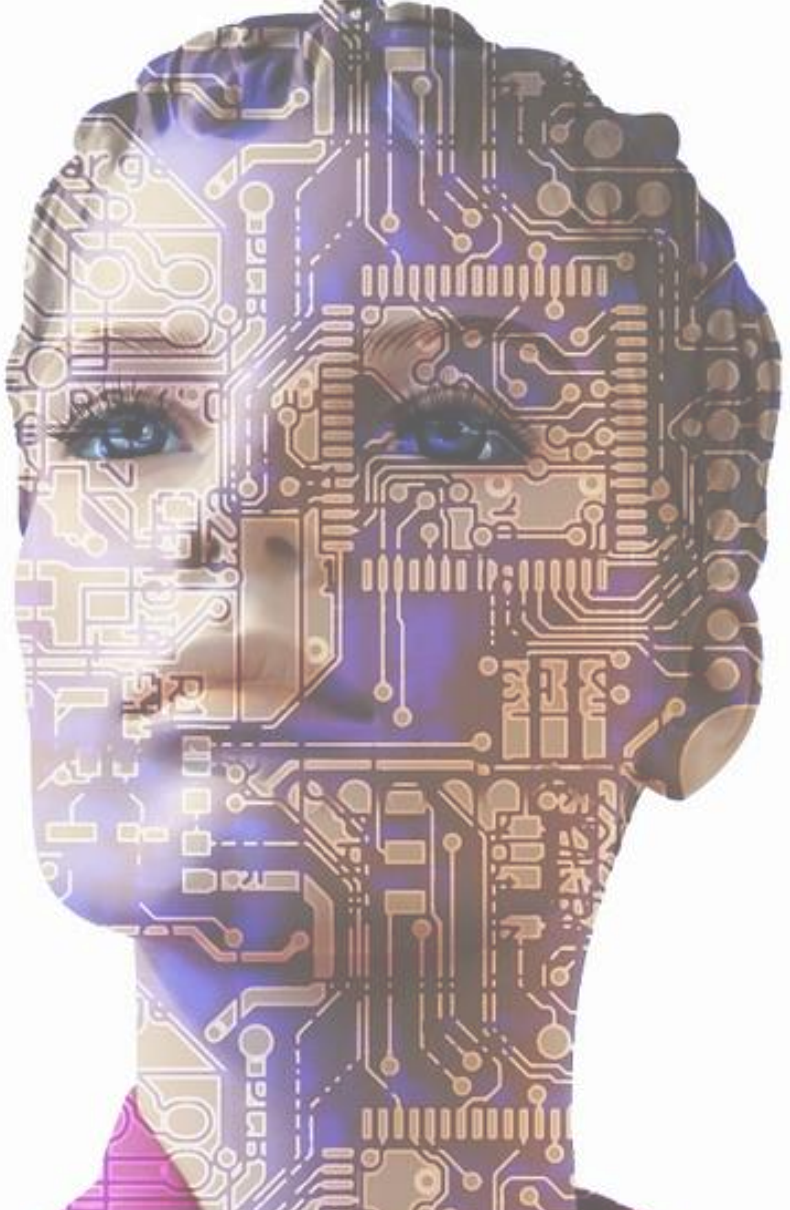
- Avoiding Overfitting
  - With thousands of weights to tune, overfitting is a problem
  - Early stopping (when performance starts dropping)
  - Regularization terms added to cost function during training
  - Dropout – ignore say 50% of all neurons randomly at each training step
    - Works surprisingly well
    - Forces your model to spread out its learning



- Tuning your topology
  - Trial & error is one way
    - Evaluate a smaller network with less neurons in the hidden layers
    - Evaluate a larger network with more layers – try reducing the size of each layer as you progress (form a funnel)
  - More layers can yield faster learning
  - (Alternative) just use more layers and neurons than you need, and let the early stop mechanism do its work

TensorFlow Overview





- TensorFlow is an open source software library developed by Google
- It's not specifically for neural networks – it's more generally an architecture for executing a graph of numerical operations / data flow graphs
- These graphs have nodes and edges, just as we saw in any typical neural network
- The arrays (data) passed along from layer of nodes to layer of nodes is known as a tensor
- Tensorflow can optimize the processing of that graph, and distribute its processing across a network

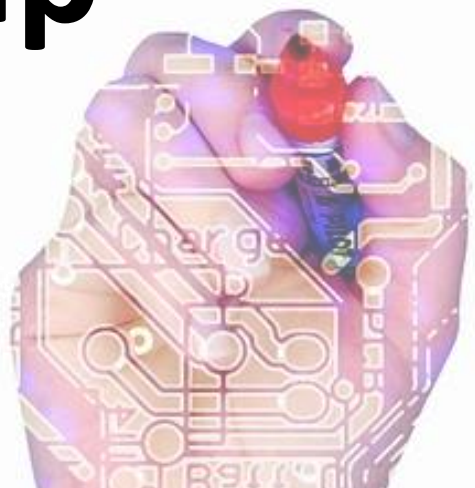
- It can also distribute work across GPU's - can handle massive scale
- Runs on about anything - Windows, Linux/macOS, Raspberry Pi, Android

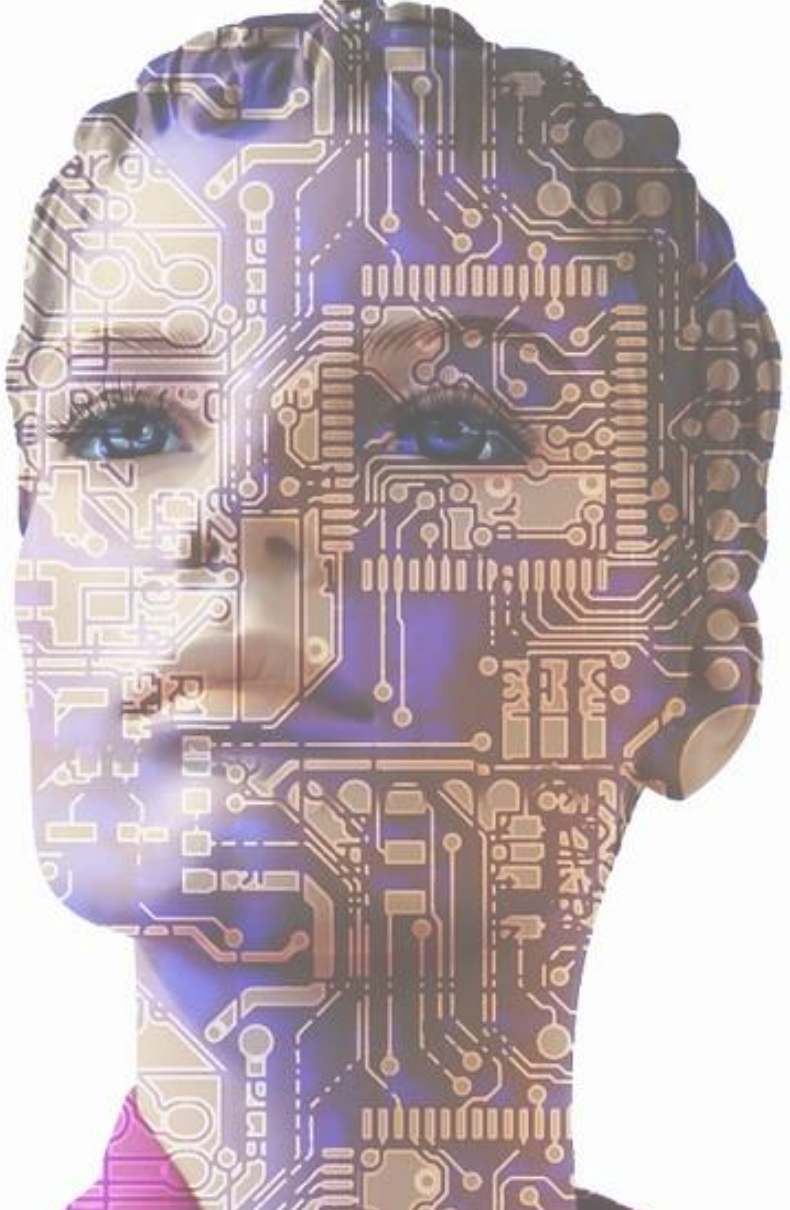
- There are two ways to use TensorFlow:
  - Customizable Graph Session
  - SciKit-Learn type interface with Contrib.Learn

- Keep in mind the Session method can feel very overwhelming if you don't have a strong background in the mathematics behind these topics

- Explore the documentation!
  - www.tensorflow.org

Environment Setup





- This course will use Jupyter Notebooks for teaching and the notes.
- All the notes can be downloaded as .py files that are compatible with any Python IDE or texto editor
- **Note**: you are free to use **whatever development environment you prefer**

- We will be using the latest version of Python 3 for this course through the Anaconda Distribution
- All notes and syntax are compatible with Python 2
- Now let’s go over your installation options for Jupyter Notebook!

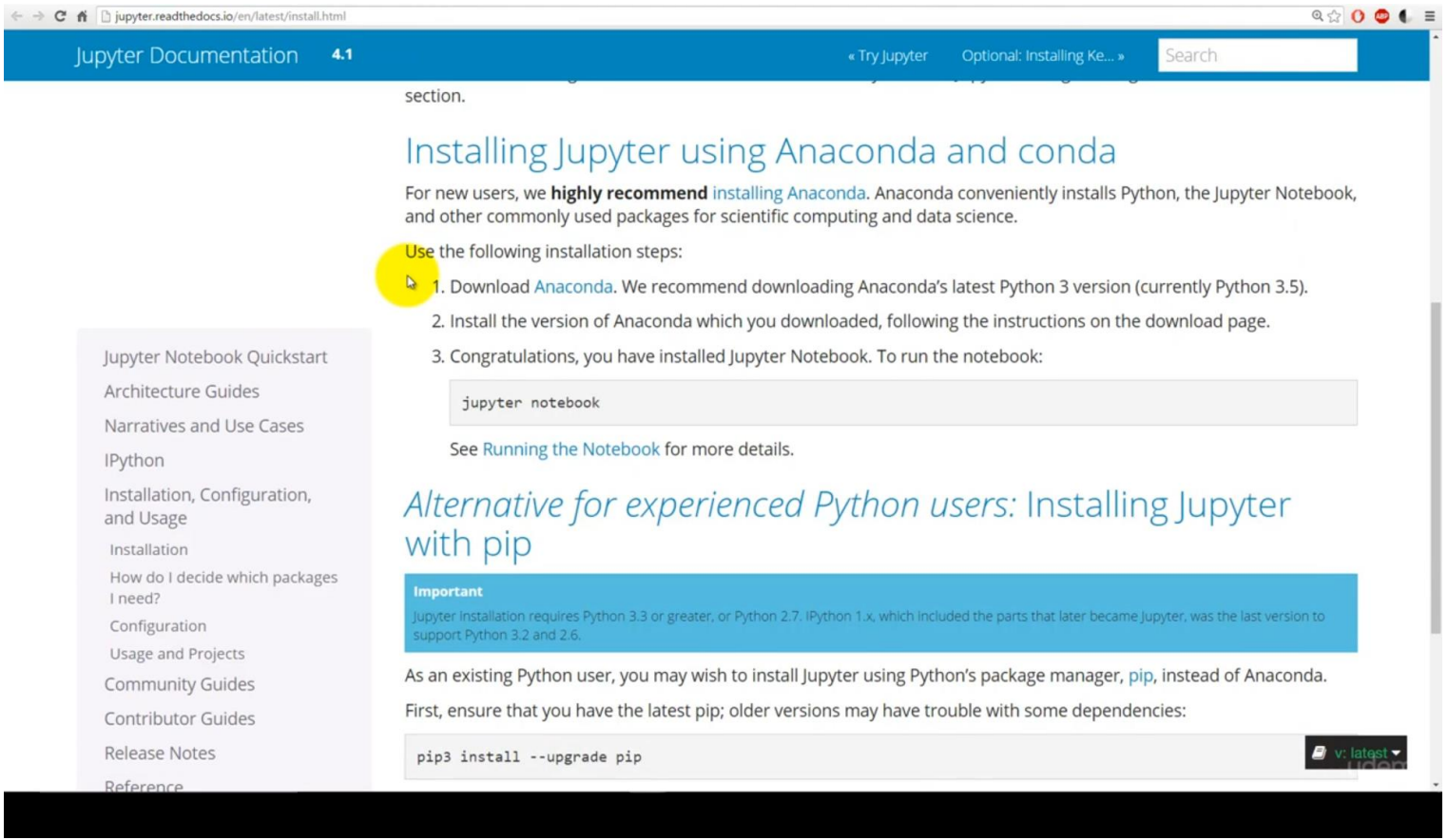
- For experienced users who already have Python
  - As na existing Python user, you may wish to install Jupyter using Python's package manager, pip, instead of Anaconda
  - Just go to your command prompt or terminal and use:

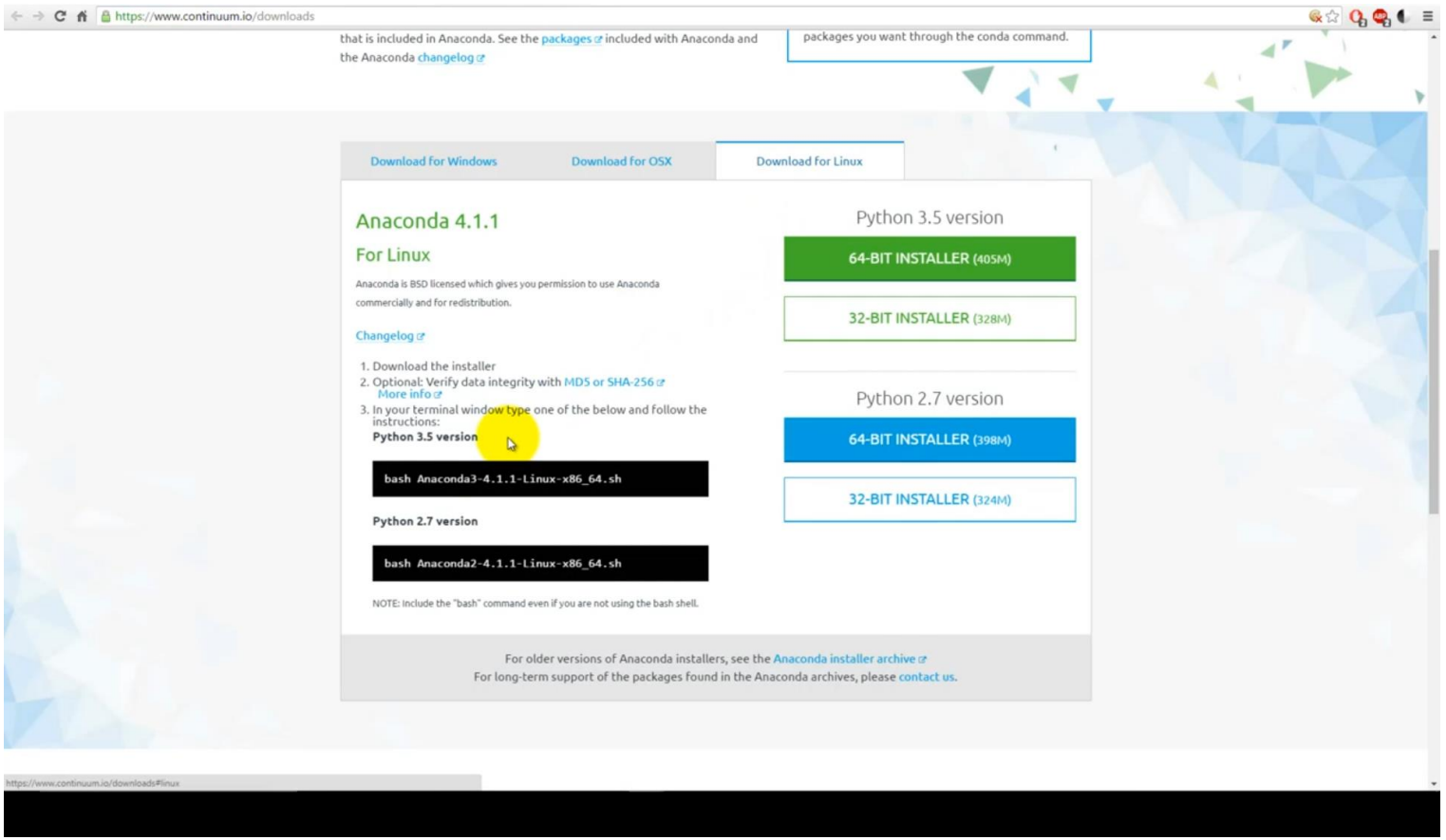
Pip3 install jupyter

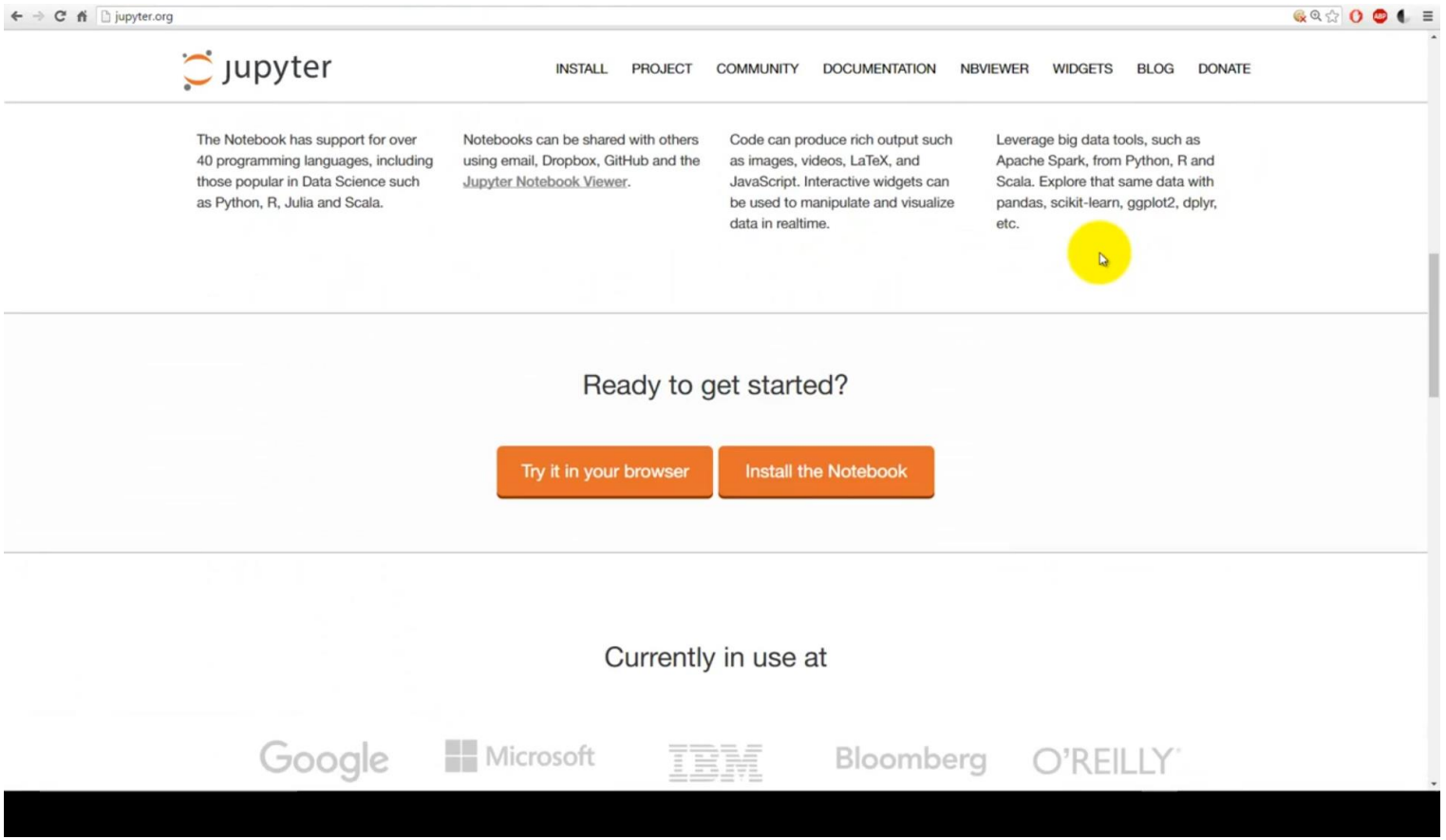
Or

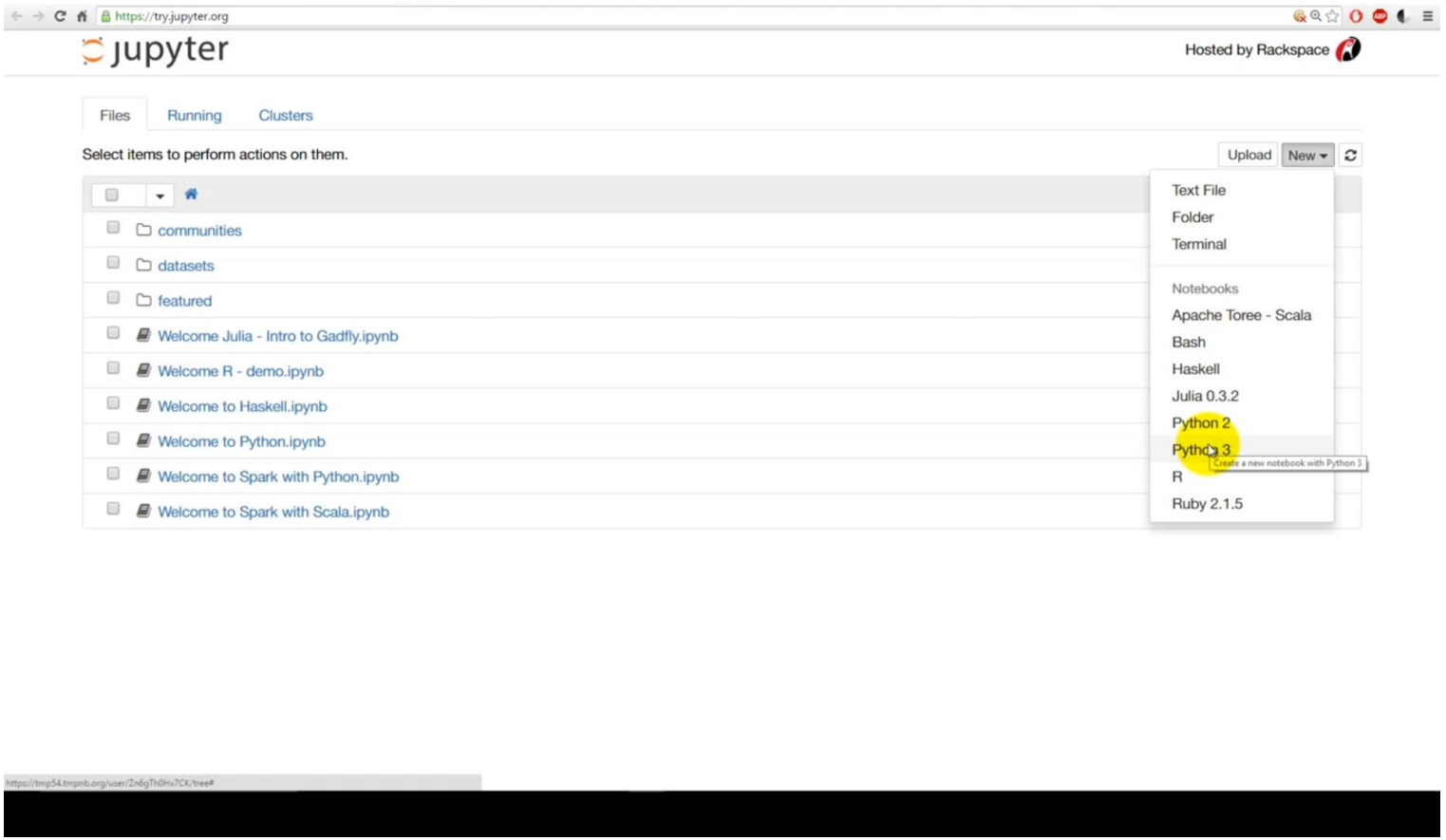
Pip install jupyter

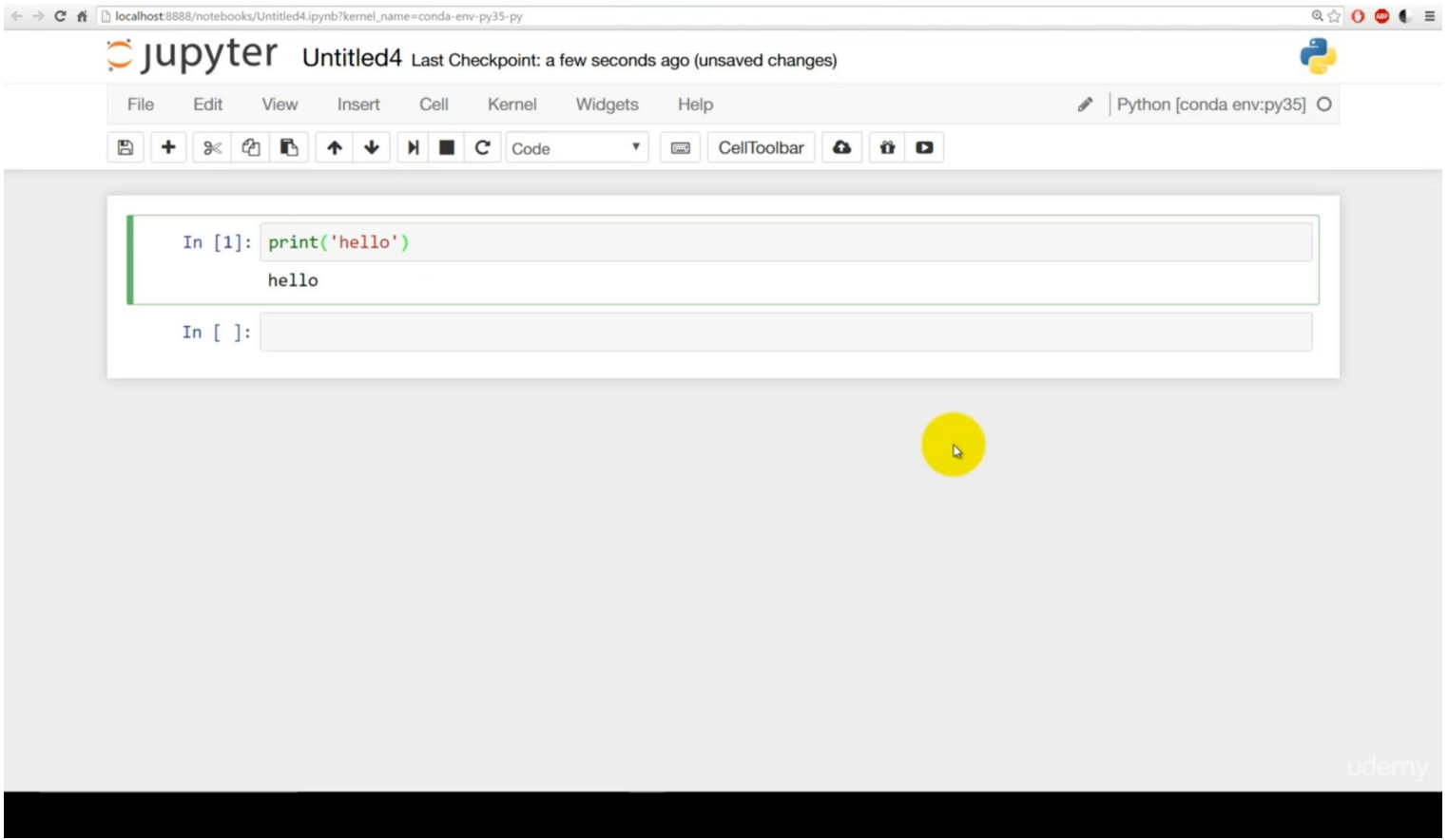
- For new users, we highly recommend installing Anaconda
  - Anaconda conveniently installs Python, the Jupyter Notebook, and other commonly used packages for scientific computing and data science
  - Let's go to www.jupyter.org to walkthrough the installation steps!


jupyter.readthedocs.io/en/latest/install.html
Jupyter Documentation 4.1
« Try Jupyter
Optional: Installing Ke... »
Search
section.
Installing Jupyter using Anaconda and conda
For new users, we highly recommend installing Anaconda. Anaconda conveniently installs Python, the Jupyter Notebook, and other commonly used packages for scientific computing and data science.
Use the following installation steps:
1. Download Anaconda. We recommend downloading Anaconda's latest Python 3 version (currently Python 3.5).
2. Install the version of Anaconda which you downloaded, following the instructions on the download page.
3. Congratulations, you have installed Jupyter Notebook. To run the notebook:
jupyter notebook
See Running the Notebook for more details.
Alternative for experienced Python users: Installing Jupyter with pip
Important
Jupyter Installation requires Python 3.3 or greater, or Python 2.7. IPython 1.x, which included the parts that later became Jupyter, was the last version to support Python 3.2 and 2.6.
As an existing Python user, you may wish to install Jupyter using Python's package manager, pip, instead of Anaconda.
First, ensure that you have the latest pip; older versions may have trouble with some dependencies:
pip3 install --upgrade pip
v: latest
Jupyter Notebook Quickstart
Architecture Guides
Narratives and Use Cases
IPython
Installation, Configuration, and Usage
Installation
How do I decide which packages I need?
Configuration
Usage and Projects
Community Guides
Contributor Guides
Release Notes
Reference


https://www.continuum.io/downloads
that is included in Anaconda. See the packages included with Anaconda and the Anaconda changelog
packages you want through the conda command.
Download for Windows
Download for OSX
Download for Linux
Anaconda 4.1.1
For Linux
Anaconda is BSD licensed which gives you permission to use Anaconda commercially and for redistribution.
Changelog
1. Download the installer
2. Optional: Verify data integrity with MD5 or SHA-256
More info
3. In your terminal window type one of the below and follow the instructions:
Python 3.5 version
bash Anaconda3-4.1.1-Linux-x86_64.sh
Python 2.7 version
bash Anaconda2-4.1.1-Linux-x86_64.sh
NOTE: Include the "bash" command even if you are not using the bash shell.
Python 3.5 version
64-BIT INSTALLER (405M)
32-BIT INSTALLER (328M)
Python 2.7 version
64-BIT INSTALLER (398M)
32-BIT INSTALLER (324M)
For older versions of Anaconda installers, see the Anaconda installer archive
For long-term support of the packages found in the Anaconda archives, please contact us.


jupyter.org
jupyter
INSTALL PROJECT COMMUNITY DOCUMENTATION NBVIEWER WIDGETS BLOG DONATE
The Notebook has support for over 40 programming languages, including those popular in Data Science such as Python, R, Julia and Scala.
Notebooks can be shared with others using email, Dropbox, GitHub and the Jupyter Notebook Viewer.
Code can produce rich output such as images, videos, LaTeX, and JavaScript. Interactive widgets can be used to manipulate and visualize data in realtime.
Leverage big data tools, such as Apache Spark, from Python, R and Scala. Explore that same data with pandas, scikit-learn, ggplot2, dplyr, etc.
Ready to get started?
Try it in your browser
Install the Notebook
Currently in use at
Google
Microsoft
IBM
Bloomberg
O'REILLY


https://try.jupyter.org
jupyter
Hosted by Rackspace
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
Text File
Folder
Terminal
Notebooks
Apache Toree - Scala
Bash
Haskell
Julia 0.3.2
Python 2
Python 3
R
Ruby 2.1.5
Create a new notebook with Python 3
communities
datasets
featured
Welcome Julia - Intro to Gadfly.ipynb
Welcome R - demo.ipynb
Welcome to Haskell.ipynb
Welcome to Python.ipynb
Welcome to Spark with Python.ipynb
Welcome to Spark with Scala.ipynb


localhost:8888/notebooks/Untitled4.ipynb?kernel_name=conda-env-py35-py
jupyter
Untitled4
Last Checkpoint: a few seconds ago (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Python [conda env:py35]
Code
CellToolbar
In [1]:
print('hello')
hello
In [ ]:
udemy

Optional: Anaconda Virtual Environments

- Virtual Environments allow you to set up virtual installations of Python and libraries on your computer
- You can have multiple versions of Python or libraries and easily activate or deactivate these environments
- Let’s see some exemples of why you may want to do this

- Sometimes you'll want to program in diferente versions of a library
- For example:
  - You develop a program with SciKit-Learn 0.17
  - SciKit-Learn 0.18 is released
  - You want to explore 0.18 but don't want your old code to break

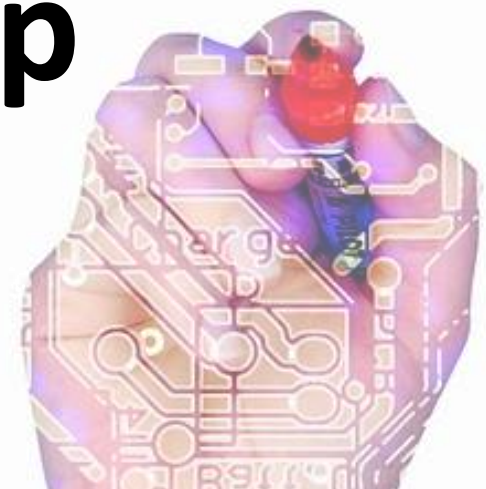
- Sometimes you'll want to make sure your library installations are in the correct location
- For exemple:
  - You want multiple versions of Python on your computer
  - You want one environment with Python 2.7 and another with Python 3.5

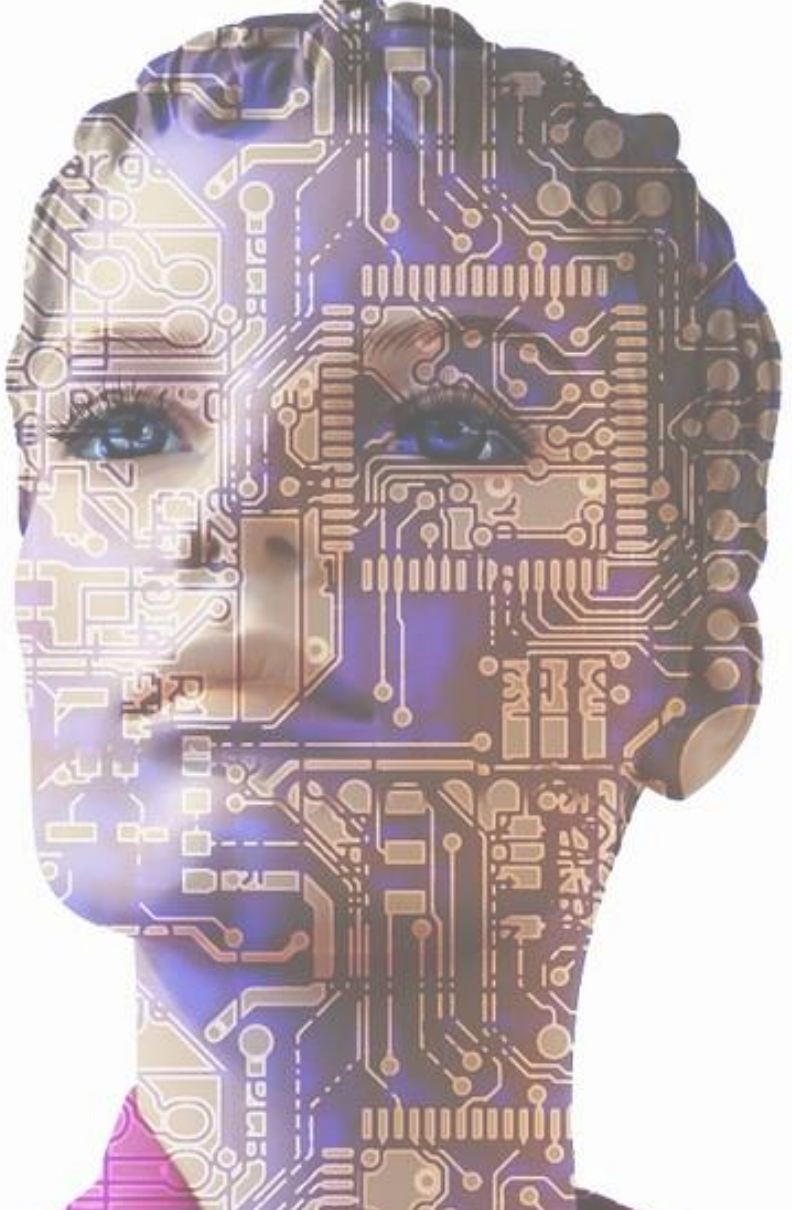
- There is the **virtualenv** library for normal Python distributions
- Anaconda has a built-in virtual environment manager that makes the whole process really easy
- Check out the resource link for the oficial documentation:
  - http://conda.pydata.org/docs/using/envs.html

- Command Prompt Example (create env. and activate it):

```
conda create –-name mypython3version python=3.5 numpy
conda info --evns
activate mypython3version
python
import numpy as np
import pandas as pd      -> Error
quit()
conda install pandas
deactivate
```

TensorFlow Setup





- Check the documentation for the latest install instructions
- In this course, we'll be using TensorFlow 1.10
  - Ref: https://www.tensorflow.org/install

Conda install:

(optional) conda create –n tensorflow_env python=3.5

source activate tensorflow_env

conda install –c conda-forge tensorflow

Pip install:

Pip3 install –U pip virtualenv

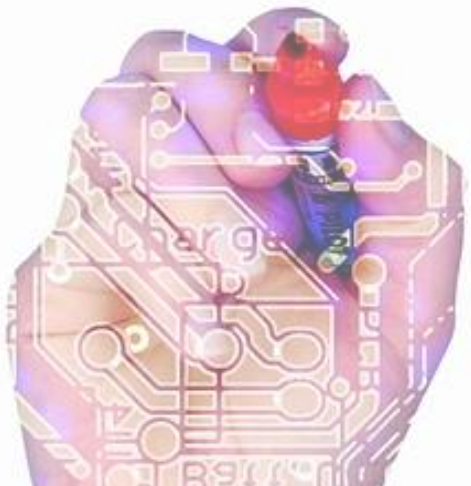
virtualenv --system-site-packages -p python3 ./venv

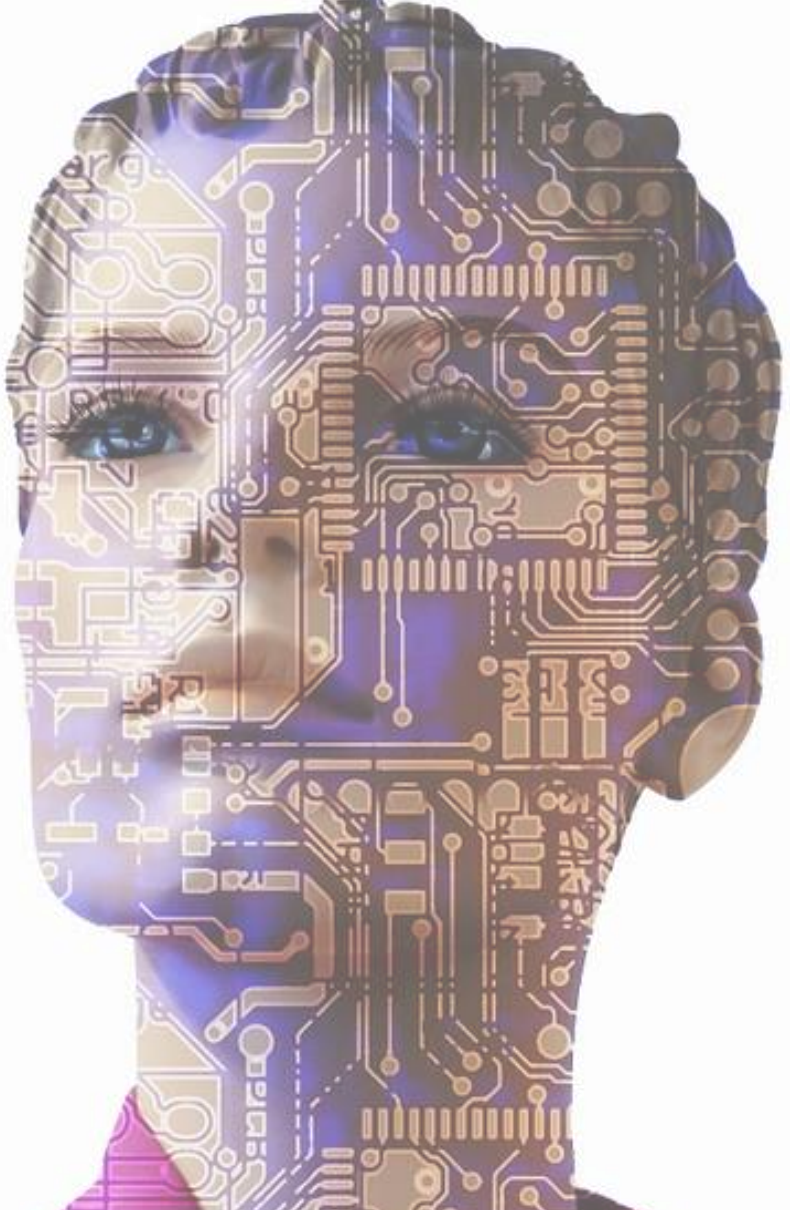
.\venv\Scripts\activate

pip install --upgrade pip

pip install tensorflow==1.10

Tensorflow Basics





- A tensor is just a fancy name for na array or matrix of values

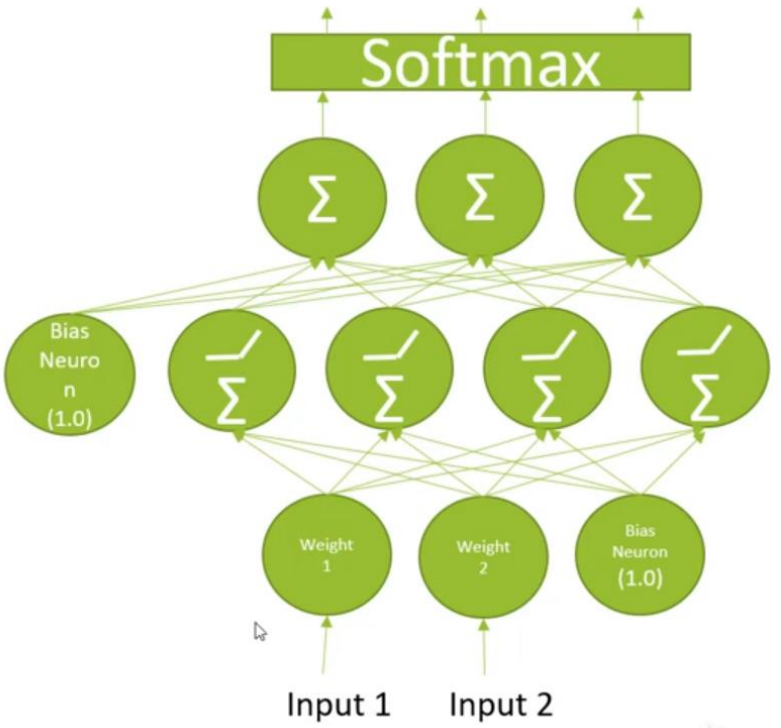
- To use TensorFlow:
  - Construct a graph to compute your tensors
  - Initialize your variables
  - Execute that graph – nothing actually happens until then

World's simplest Tensorflow app:

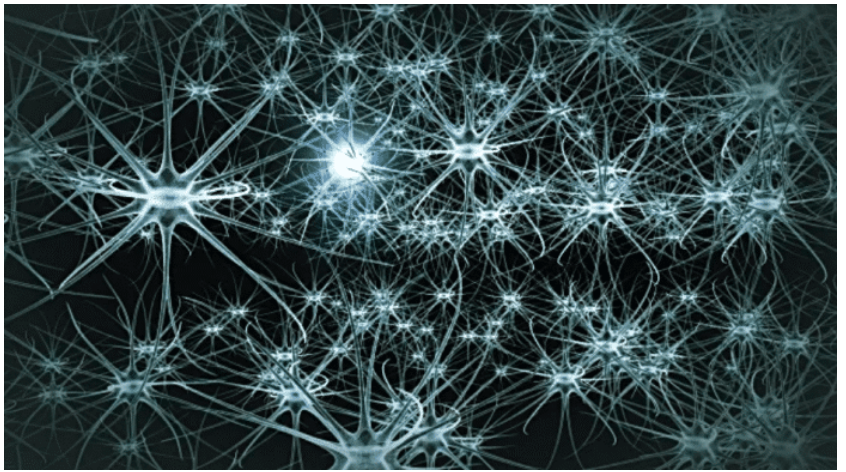
```
import tensorflow as tf

a = tf.Variable(1, name="a")
b = tf.Variable(2, name="b")
f = a + b

init = tf.global_variables_initializer()
with tf.Session() as s:
    init.run()
    print( f.eval() )
```

- Creating a neural network
  - Mathematical insights:
    - All those interconnected arrows multiplying weights can be thought of as a big matrix multiplication
    - The bias term can just be added onto the result of that matrix multiplication
  - In TensorFlow, we can define a layer of a neural network as:

output = tf.matmul (previous_layer, layer_weights) + layer_biases



- Creating a neural network
  1. Load up the training and testing data
  2. Construct a graph describing the neural network
     - Use placeholders for the input data and target labels – this way we can use the same graph for training and testing
     - Use variables for the learned weights for each connection and learned biases for each neuron – variables are preserved across runs within a TensorFlow session
  3. Associate na optimizer (i.e. gradient descent) to the network
  4. Run the optimizer with your training data
  5. Evaluate your trained network with your testing data



- Make sure your features are normalized
  - Neural networks usually work best if your input data is normalized
    - That is, 0 mean and unit variance
    - Every input feature is comparable in terms of magnitude
  - Scikit_learn's StandardScaler can do this for you
  - Many datasets are normalized to begin with

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Mestrado Integrado em Engenharia Informática
Computação Natural
2018/2019

Paulo Novais, Cesar Analide, Filipe Gonçalves