

TensorFlow



LEARN IN 1 DAY

KRISHNA RUNGTA

TensorFlow in 1 Day: Make your own Neural Network

By Krishna Rungta

Copyright 2018 - All Rights Reserved – Krishna Rungta

ALL RIGHTS RESERVED. No part of this publication may be reproduced or transmitted in any form whatsoever, electronic, or mechanical, including photocopying, recording, or by any informational storage or retrieval system without express written, dated and signed permission from the author.

Table Of Content

Chapter 1: What is Deep learning?

1. [What is Deep learning?](#)
2. [Deep learning Process](#)
3. [Classification of Neural Networks](#)
4. [Types of Deep Learning Networks](#)
5. [Feed-forward neural networks](#)
6. [Recurrent neural networks \(RNNs\)](#)
7. [Convolutional neural networks \(CNN\)](#)

Chapter 2: Machine Learning vs Deep Learning

1. [What is AI?](#)
2. [What is ML?](#)
3. [What is Deep Learning?](#)
4. [Machine Learning Process](#)
5. [Deep Learning Process](#)
6. [Automate Feature Extraction using DL](#)
7. [Difference between Machine Learning and Deep Learning](#)

8. [When to use ML or DL?](#)

Chapter 3: What is TensorFlow?

1. [What is TensorFlow?](#)
2. [History of TensorFlow](#)
3. [TensorFlow Architecture](#)
4. [Where can Tensorflow run?](#)
5. [Introduction to Components of TensorFlow](#)
6. [Why is TensorFlow popular?](#)
7. [List of Prominent Algorithms supported by TensorFlow](#)

Chapter 4: Comparison of Deep Learning Libraries

1. [8 Best Deep learning Libraries /Framework](#)
2. [MICROSOFT COGNITIVE TOOLKIT\(CNTK\)](#)
3. [TensorFlow Vs Theano Vs Torch Vs Keras Vs infer.net Vs CNTK Vs MXNet Vs Caffe: Key Differences](#)

Chapter 5: How to Download and Install TensorFlow Windows and Mac

1. [TensorFlow Versions](#)
2. [Install Anaconda](#)
3. [Create .yml file to install Tensorflow and dependencies](#)

4. [Launch Jupyter Notebook](#)
5. [Jupyter with the main conda environment](#)

[Chapter 6: Jupyter Notebook Tutorial](#)

1. [What is Jupyter Notebook?](#)
2. [Jupyter Notebook App](#)
3. [How to use Jupyter](#)

[Chapter 7: Tensorflow on AWS](#)

1. [PART 1: Set up a key pair](#)
2. [PART 2: Set up a security group](#)
3. [Launch your instance \(Windows users\)](#)
4. [Part 4: Install Docker](#)
5. [Part 5: Install Jupyter](#)
6. [Part 6: Close connection](#)

[Chapter 8: TensorFlow Basics: Tensor, Shape, Type, Graph, Sessions & Operators](#)

1. [What is a Tensor?](#)
2. [Representation of a Tensor](#)
3. [Types of Tensor](#)

4. [Shape of tensor](#)
5. [Type of data](#)
6. [Creating operator](#)
7. [Variables](#)

Chapter 9: Tensorboard: Graph Visualization with Example

Chapter 10: NumPy

1. [What is NumPy?](#)
2. [Why use NumPy?](#)
3. [How to install NumPy?](#)
4. [Mathematical Operations on an Array](#)
5. [Shape of Array](#)
6. [np.zeros and np.ones](#)
7. [Reshape and Flatten Data](#)
8. [hstack and vstack](#)

Chapter 11: Pandas

1. [What is Pandas?](#)
2. [Why use Pandas?](#)
3. [How to install Pandas?](#)

4. [What is a data frame?](#)

5. [What is a Series?](#)

6. [Concatenation](#)

[Chapter 12: Scikit-Learn](#)

1. [What is Scikit-learn?](#)

2. [Download and Install scikit-learn](#)

3. [Machine learning with scikit-learn](#)

4. [Step 1\) Import the data](#)

5. [Step 2\) Create the train/test set](#)

6. [Step 3\) Build the pipeline](#)

7. [Step 4\) Using our pipeline in a grid search](#)

[Chapter 13: Linear Regression](#)

1. [Linear regression](#)

2. [How to train a linear regression model](#)

3. [How to train a Linear Regression with TensorFlow](#)

4. [Pandas](#)

5. [Numpy Solution](#)

6. [Tensorflow solution](#)

Chapter 14: Linear Regression Case Study

1. [Summary statistics](#)

2. [Facets Overview](#)

3. [Facets Deep Dive](#)

4. [Install Facet](#)

5. [Overview](#)

6. [Graph](#)

7. [Facets Deep Dive](#)

Chapter 15: Linear Classifier in TensorFlow

1. [What is Linear Classifier?](#)

2. [How Binary classifier works?](#)

3. [How to Measure the performance of Linear Classifier?](#)

4. [Linear Classifier with TensorFlow](#)

Chapter 16: Kernel Methods

1. [Why do you need Kernel Methods?](#)

2. [What is a Kernel in machine learning?](#)

3. [Type of Kernel Methods](#)

4. [Train Gaussian Kernel classifier with TensorFlow](#)

Chapter 17: TensorFlow ANN (Artificial Neural Network)

1. [What is Artificial Neural Network?](#)
2. [Neural Network Architecture](#)
3. [Limitations of Neural Network](#)
4. [Example Neural Network in TensorFlow](#)
5. [Train a neural network with TensorFlow](#)

Chapter 18: ConvNet(Convolutional Neural Network): TensorFlow Image Classification

1. [What is Convolutional Neural Network?](#)
2. [Architecture of a Convolutional Neural Network](#)
3. [Components of Convnets](#)
4. [Train CNN with TensorFlow](#)

Chapter 19: Autoencoder with TensorFlow

1. [What is an Autoencoder?](#)
2. [How does Autoencoder work?](#)
3. [Stacked Autoencoder Example](#)
4. [Build an Autoencoder with TensorFlow](#)

Chapter 20: RNN(Recurrent Neural Network) TensorFlow

1. [What do we need an RNN?](#)
2. [What is RNN?](#)
3. [Build an RNN to predict Time Series in TensorFlow](#)

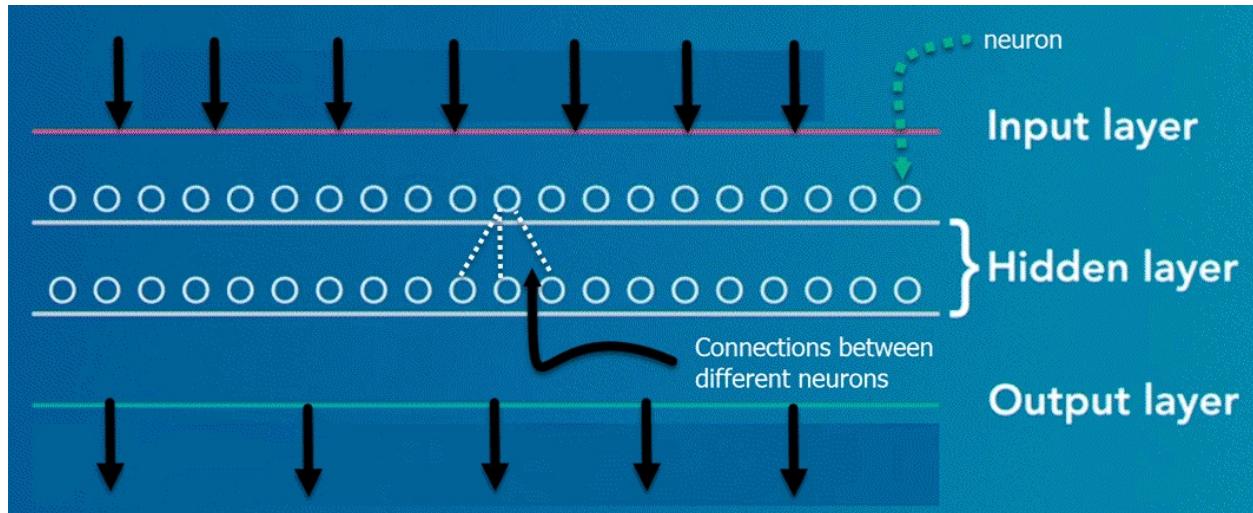
Chapter 1: What is Deep learning?

What is Deep learning?

Deep learning is a computer software that **mimics the network of neurons in a brain**. It is a subset of machine learning and is called deep learning because it makes use of deep **neural networks**.

Deep learning algorithms are constructed with connected layers.

- The first layer is called the Input Layer
- The last layer is called the Output Layer
- All layers in between are called Hidden Layers. The word deep means the network join neurons in more than two layers.



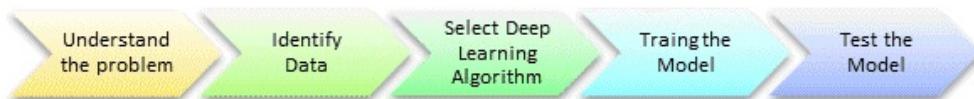
Each Hidden layer is composed of neurons. The neurons are connected to each other. The neuron will process and then propagate the input signal it receives the layer above it. The strength of the signal given the neuron in the next layer depends on the weight, bias and activation function.

The network consumes large amounts of input data and operates them through

multiple layers; the network can learn increasingly complex features of the data at each layer.

Deep learning Process

A deep neural network provides state-of-the-art accuracy in many tasks, from object detection to speech recognition. They can learn automatically, without predefined knowledge explicitly coded by the programmers.



To grasp the idea of deep learning, imagine a family, with an infant and parents. The toddler points objects with his little finger and always says the word 'cat.' As its parents are concerned about his education, they keep telling him 'Yes, that is a cat' or 'No, that is not a cat.' The infant persists in pointing objects but becomes more accurate with 'cats.' The little kid, deep down, does not know why he can say it is a cat or not. He has just learned how to hierarchies complex features coming up with a cat by looking at the pet overall and continue to focus on details such as the tails or the nose before to make up his mind.

A neural network works quite the same. Each layer represents a deeper level of knowledge, i.e., the hierarchy of knowledge. A neural network with four layers will learn more complex feature than with that with two layers.

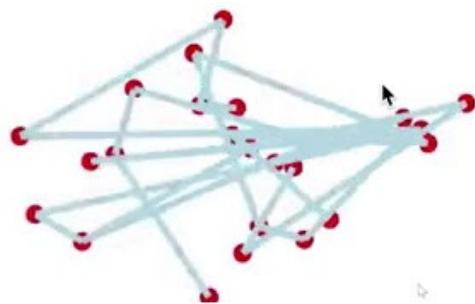
The learning occurs in two phases.

- The first phase consists of applying a nonlinear transformation of the input and create a statistical model as output.
- The second phase aims at improving the model with a mathematical method known as derivative.

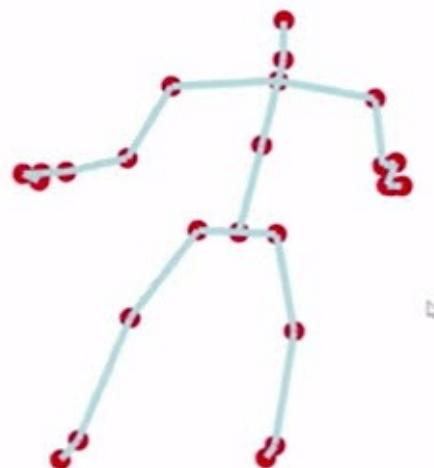
The neural network repeats these two phases hundreds to thousands of time until it has reached a tolerable level of accuracy. The repeat of this two-phase is called

an iteration.

To give an example, take a look at the motion below, the model is trying to learn how to dance. After 10 minutes of training, the model does not know how to dance, and it looks like a scribble.



After 48 hours of learning, the computer masters the art of dancing.



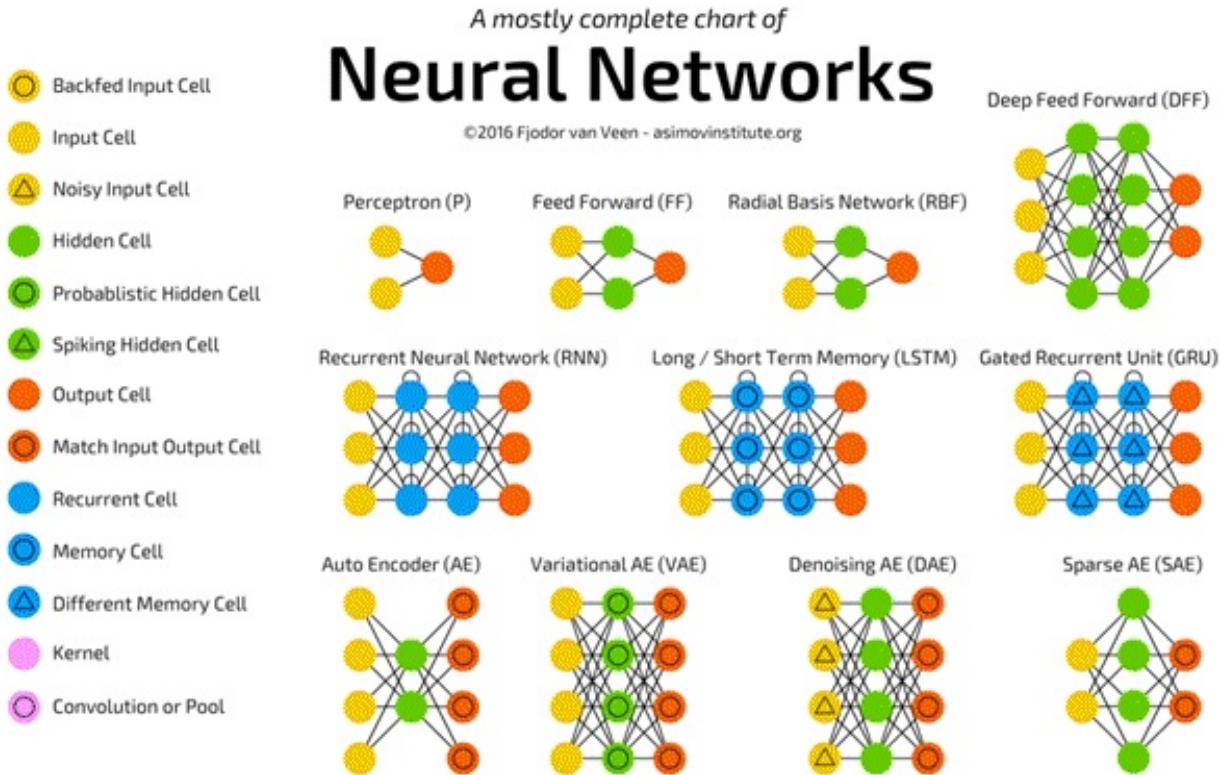
Classification of Neural Networks

Shallow neural network: The Shallow neural network has only one hidden layer between the input and output.

Deep neural network: Deep neural networks have more than one layer. For instance, Google LeNet model for image recognition counts 22 layers.

Nowadays, deep learning is used in many ways like a driverless car, mobile phone, Google Search Engine, Fraud detection, TV, and so on.

Types of Deep Learning Networks



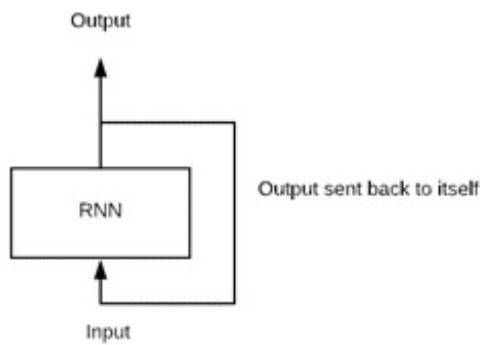
Feed-forward neural networks

The simplest type of artificial neural network. With this type of architecture, information flows in only one direction, forward. It means, the information's flows starts at the input layer, goes to the "hidden" layers, and end at the output layer. The network

does not have a loop. Information stops at the output layers.

Recurrent neural networks (RNNs)

RNN is a multi-layered neural network that can store information in context nodes, allowing it to learn data sequences and output a number or another sequence. In simple words it is an Artificial neural networks whose connections between neurons include loops. RNNs are well suited for processing sequences of inputs.



Example, if the task is to predict the next word in the sentence "Do you want a.....?

- The RNN neurons will receive a signal that point to the start of the sentence.
- The network receives the word "Do" as an input and produces a vector of the number. This vector is fed back to the neuron to provide a memory to the network. This stage helps the network to remember it received "Do" and it received it in the first position.
- The network will similarly proceed to the next words. It takes the word "you" and "want." The state of the neurons is updated upon receiving each word.
- The final stage occurs after receiving the word "a." The neural network will

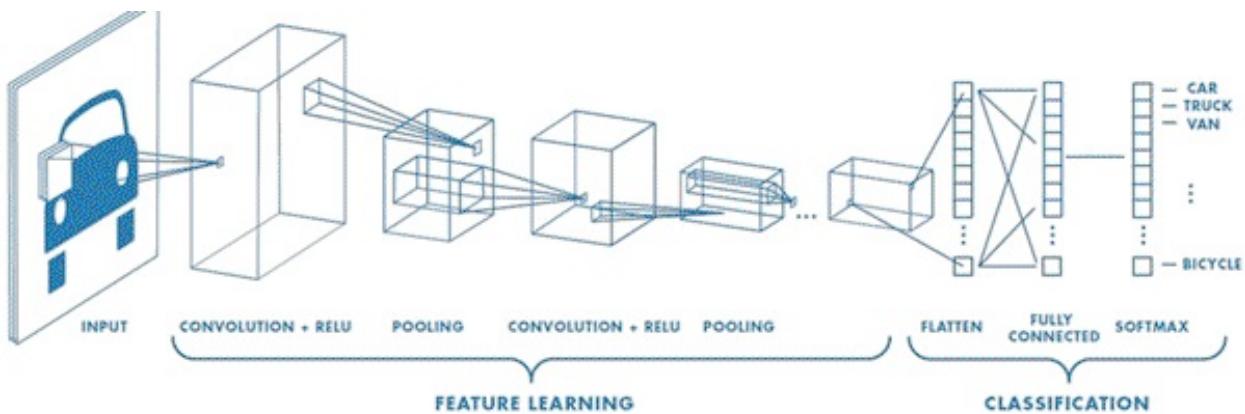
provide a probability for each English word that can be used to complete the sentence. A well-trained RNN probably assigns a high probability to "café," "drink," "burger," etc.

Common uses of RNN

- Help securities traders to generate analytic reports
- Detect abnormalities in the contract of financial statement
- Detect fraudulent credit-card transaction
- Provide a caption for images
- Power chatbots
- The standard uses of RNN occur when the practitioners are working with time-series data or sequences (e.g., audio recordings or text).

Convolutional neural networks (CNN)

CNN is a multi-layered neural network with a unique architecture designed to extract increasingly complex features of the data at each layer to determine the output. CNN's are well suited for perceptual tasks.



CNN is mostly used when there is an unstructured data set (e.g., images) and the practitioners need to extract information from it

For instance, if the task is to predict an image caption:

- The CNN receives an image of let's say a cat, this image, in computer term, is a collection of the pixel. Generally, one layer for the greyscale picture and three layers for a color picture.
- During the feature learning (i.e., hidden layers), the network will identify unique features, for instance, the tail of the cat, the ear, etc.
- When the network thoroughly learned how to recognize a picture, it can provide a probability for each image it knows. The label with the highest probability will become the prediction of the network.

Reinforcement Learning

Reinforcement learning is a subfield of machine learning in which systems are trained by receiving virtual "rewards" or "punishments," essentially learning by trial and error. Google's DeepMind has used reinforcement learning to beat a human champion in the Go games. Reinforcement learning is also used in video games to improve the gaming experience by providing smarter bot.

One of the most famous algorithms are:

- Q-learning
- Deep Q network
- State-Action-Reward-State-Action (SARSA)
- Deep Deterministic Policy Gradient (DDPG)

Applications/ Examples of deep learning applications

AI in Finance: The financial technology sector has already started using AI to save time, reduce costs, and add value. Deep learning is changing the lending industry by using more robust credit scoring. Credit decision-makers can use AI for robust credit lending applications to achieve faster, more accurate risk assessment, using machine intelligence to factor in the character and capacity of applicants.

Underwrite is a Fintech company providing an AI solution for credit makers company. underwrite.ai uses AI to detect which applicant is more likely to pay back a loan. Their approach radically outperforms traditional methods.

AI in HR: Under Armour, a sportswear company revolutionizes hiring and modernizes the candidate experience with the help of AI. In fact, Under Armour Reduces hiring time for its retail stores by 35%. Under Armour faced a growing popularity interest back in 2012. They had, on average, 30000 resumes a month. Reading all of those applications and begin to start the screening and interview process was taking too long. The lengthy process to get people hired and onboarded impacted Under Armour's ability to have their retail stores fully staffed, ramped and ready to operate.

At that time, Under Armour had all of the 'must have' HR technology in place such as transactional solutions for sourcing, applying, tracking and onboarding but those tools weren't useful enough. Under armour choose **HireVue**, an AI provider for HR solution, for both on-demand and live interviews. The results were bluffing; they managed to decrease by 35% the time to fill. In return, the hired higher quality staffs.

AI in Marketing: AI is a valuable tool for customer service management and

personalization challenges. Improved speech recognition in call-center management and call routing as a result of the application of AI techniques allows a more seamless experience for customers.

For example, deep-learning analysis of audio allows systems to assess a customer's emotional tone. If the customer is responding poorly to the AI chatbot, the system can reroute the conversation to real, human operators that take over the issue.

Apart from the three examples above, AI is widely used in other sectors/industries.

Why is Deep Learning Important?

Deep learning is a powerful tool to make prediction an actionable result. Deep learning excels in pattern discovery (unsupervised learning) and knowledge-based prediction. Big data is the fuel for deep learning. When both are combined, an organization can reap unprecedented results in term of productivity, sales, management, and innovation.

Deep learning can outperform traditional method. For instance, deep learning algorithms are 41% more accurate than machine learning algorithm in image classification, 27 % more accurate in facial recognition and 25% in voice recognition.

Limitations of deep learning

Data labeling

Most current AI models are trained through "supervised learning." It means that humans must label and categorize the underlying data, which can be a sizable and error-prone chore. For example, companies developing self-driving-car technologies are hiring hundreds of people to manually annotate hours of video feeds from prototype vehicles to help train these systems.

Obtain huge training datasets

It has been shown that simple deep learning techniques like CNN can, in some cases, imitate the knowledge of experts in medicine and other fields. The current wave of machine learning, however, requires training data sets that are not only labeled but also sufficiently broad and universal.

Deep-learning methods required thousands of observation for models to become relatively good at classification tasks and, in some cases, millions for them to perform at the level of humans. Without surprise, deep learning is famous in giant tech companies; they are using big data to accumulate petabytes of data. It allows them to create an impressive and highly accurate deep learning model.

Explain a problem

Large and complex models can be hard to explain, in human terms. For instance, why a particular decision was obtained. It is one reason that acceptance of some AI tools are slow in application areas where interpretability is useful or indeed required.

Furthermore, as the application of AI expands, regulatory requirements could also drive the need for more explainable AI models.

Summary

Deep learning is the new state-of-the-art for artificial intelligence. Deep learning architecture is composed of an input layer, hidden layers, and an output layer. The word deep means there are more than two fully connected layers.

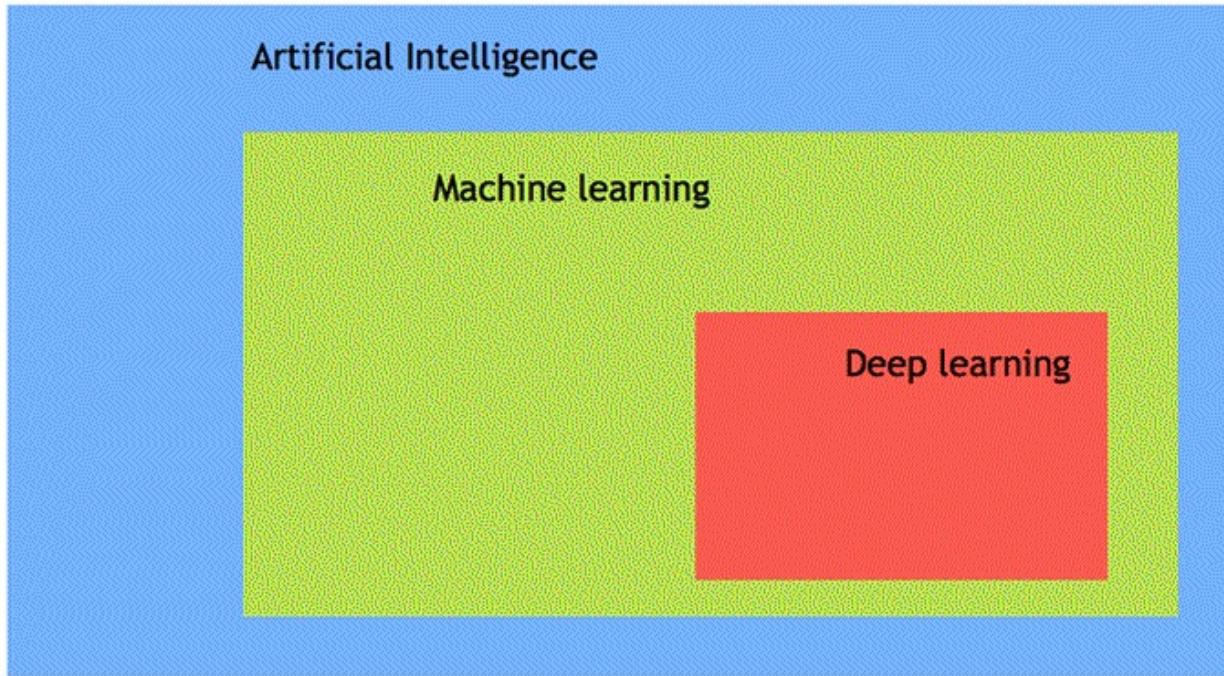
There is a vast amount of neural network, where each architecture is designed to perform a given task. For instance, CNN works very well with pictures, RNN provides impressive results with time series and text analysis.

Deep learning is now active in different fields, from finance to marketing, supply chain, and marketing. Big firms are the first one to use deep learning because they have already a large pool of data. Deep learning requires to have an extensive training dataset.

Chapter 2: Machine Learning vs Deep Learning

What is AI?

Artificial intelligence is imparting a cognitive ability to a machine. The benchmark for AI is the human intelligence regarding reasoning, speech, and vision. This benchmark is far off in the future.



AI has three different levels:

1. **Narrow AI:** A artificial intelligence is said to be narrow when the machine can perform a specific task better than a human. The current research of AI is here now
2. **General AI:** An artificial intelligence reaches the general state when it can perform any intellectual task with the same accuracy level as a human

would

3. **Active AI:** An AI is active when it can beat humans in many tasks

Early AI systems used pattern matching and expert systems.

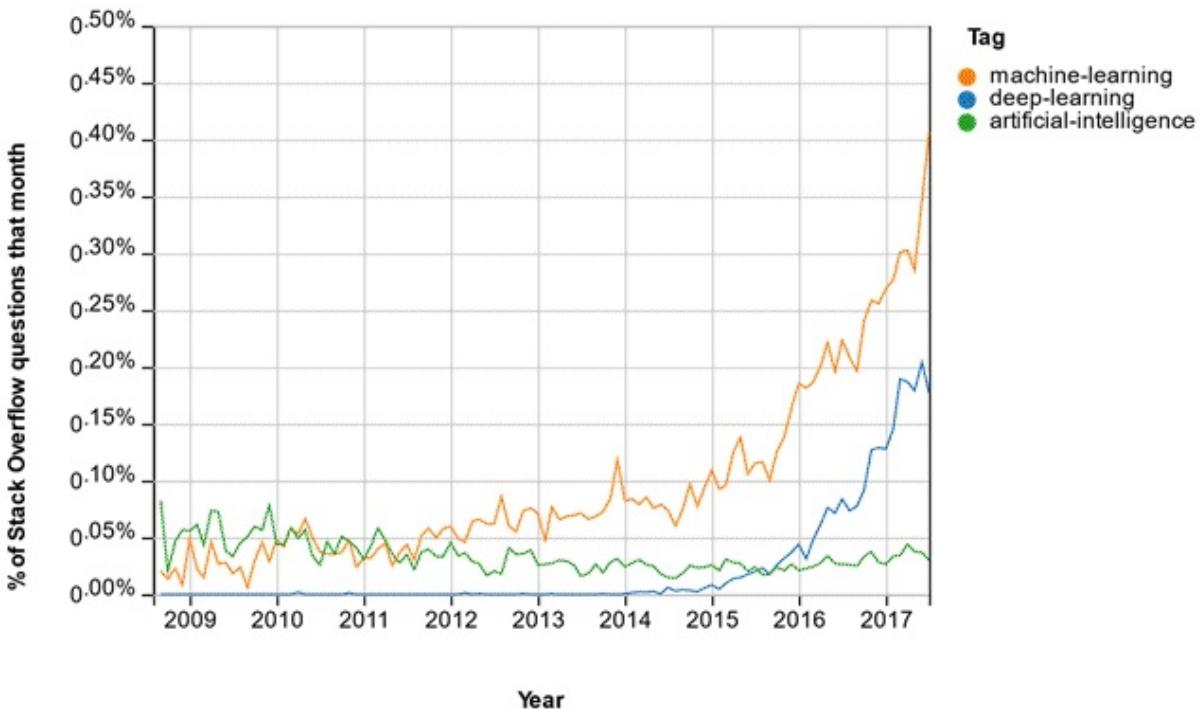
What is ML?

Machine learning is the best tool so far to analyze, understand and identify a pattern in the data. One of the main ideas behind machine learning is that the computer can be trained to automate tasks that would be exhaustive or impossible for a human being. The clear breach from the traditional analysis is that machine learning can take decisions with minimal human intervention.

Machine learning uses data to feed an algorithm that can understand the relationship between the input and the output. When the machine finished learning, it can predict the value or the class of new data point.

What is Deep Learning?

Deep learning is a computer software that mimics the network of neurons in a brain. It is a subset of machine learning and is called deep learning because it makes use of deep neural networks. The machine uses different layers to learn from the data. The depth of the model is represented by the number of layers in the model. Deep learning is the new state of the art in term of AI. In deep learning, the learning phase is done through a neural network. A neural network is an architecture where the layers are stacked on top of each other



Machine Learning Process

Imagine you are meant to build a program that recognizes objects. To train the model, you will use a **classifier**. A classifier uses the features of an object to try identifying the class it belongs to.

In the example, the classifier will be trained to detect if the image is a:

- Bicycle
- Boat
- Car
- Plane

The four objects above are the class the classifier has to recognize. To construct a classifier, you need to have some data as input and assigns a label to it. The algorithm will take these data, find a pattern and then classify it in the corresponding class.

This task is called **supervised learning**. In supervised learning, the training data you feed to the algorithm includes a label.

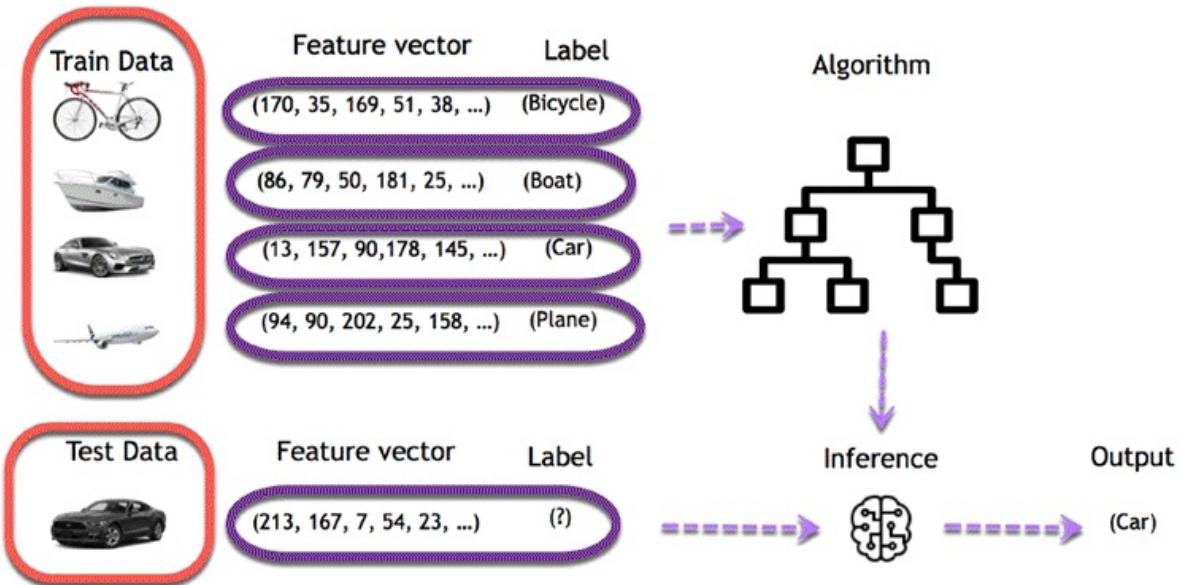
Training an algorithm requires to follow a few standard steps:

- Collect the data
- Train the classifier
- Make predictions

The first step is necessary, choosing the right data will make the algorithm success or a failure. The data you choose to train the model is called a **feature**. In the object example, the features are the pixels of the images.

Each image is a row in the data while each pixel is a column. If your image is a 28x28 size, the dataset contains 784 columns (28x28). In the picture below, each

picture has been transformed into a feature vector. The label tells the computer what object is in the image.



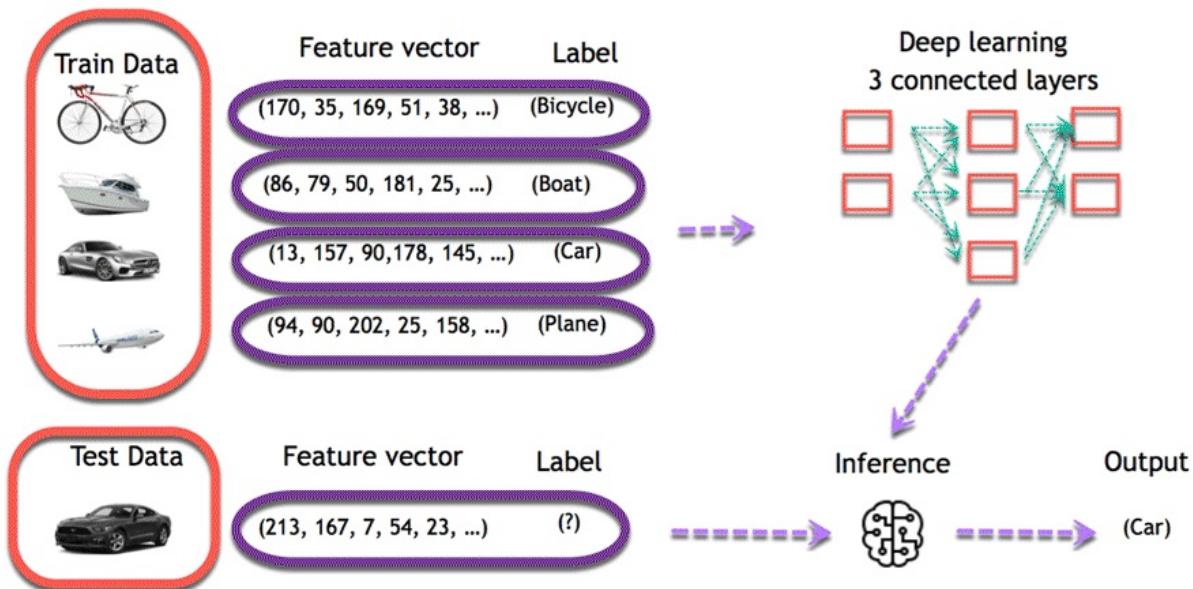
The objective is to use these training data to classify the type of object. The first step consists of creating the feature columns. Then, the second step involves choosing an algorithm to train the model. When the training is done, the model will predict what picture corresponds to what object.

After that, it is easy to use the model to predict new images. For each new image feeds into the model, the machine will predict the class it belongs to. For example, an entirely new image without a label is going through the model. For a human being, it is trivial to visualize the image as a car. The machine uses its previous knowledge to predict as well the image is a car.

Deep Learning Process

In deep learning, the learning phase is done through a neural network. A neural network is an architecture where the layers are stacked on top of each other.

Consider the same image example above. The training set would be fed to a neural network. Each input goes into a neuron and is multiplied by a weight. The result of the multiplication flows to the next layer and becomes the input. This process is repeated for each layer of the network. The final layer is named the output layer; it provides an actual value for the regression task and a probability of each class for the classification task. The neural network uses a mathematical algorithm to update the weights of all the neurons. The neural network is fully trained when the value of the weights gives an output close to the reality. For instance, a well-trained neural network can recognize the object on a picture with higher accuracy than the traditional neural net.



Automate Feature Extraction using DL

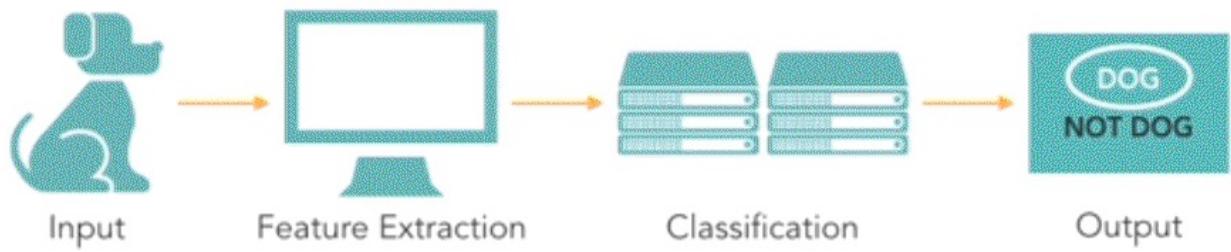
A dataset can contain a dozen to hundreds of features. The system will learn from the relevance of these features. However, not all features are meaningful for the algorithm. A crucial part of machine learning is to find a relevant set of features to make the system learns something.

One way to perform this part in machine learning is to use feature extraction. Feature extraction combines existing features to create a more relevant set of features. It can be done with PCA, T-SNE or any other dimensionality reduction algorithms.

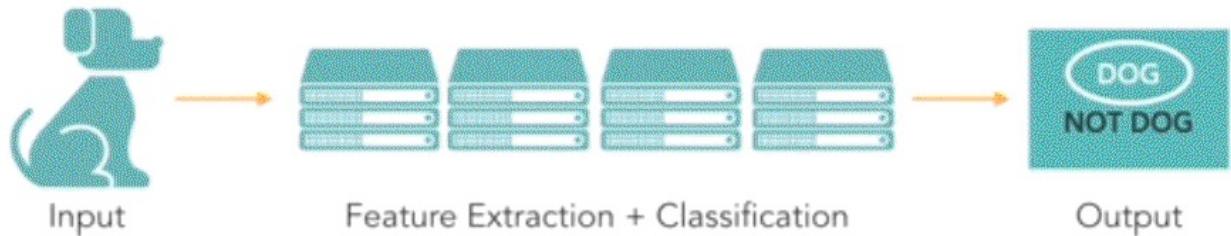
For example, in image processing, the practitioner needs to extract the feature manually in the image like the eyes, the nose, lips and so on. Those extracted features are feed to the classification model.

Deep learning solves this issue, especially for a convolutional neural network. The first layer of a neural network will learn small details from the picture; the next layers will combine the previous knowledge to make more complex information. In the convolutional neural network, the feature extraction is done with the use of the filter. The network applies a filter to the picture to see if there is a match, i.e., the shape of the feature is identical to a part of the image. If there is a match, the network will use this filter. The process of feature extraction is therefore done automatically.

TRANDITIONAL MACHINE LEARNING



DEEP LEARNING



Difference between Machine Learning and Deep Learning

	Machine Learning	Deep Learning
Data Dependencies	Excellent performances on a small/medium dataset	Excellent performance on a big dataset
Hardware dependencies	Work on a low-end machine.	Requires powerful machine, preferably with GPU: DL performs a significant amount of matrix multiplication
Feature engineering	Need to understand the features that represent the data	No need to understand the best feature that represents the data
Execution time	From few minutes to hours	Up to weeks. Neural Network needs to compute a significant number of weights
Interpretability	Some algorithms are easy to interpret (logistic, decision tree), some are almost impossible (SVM, XGBoost)	Difficult to impossible

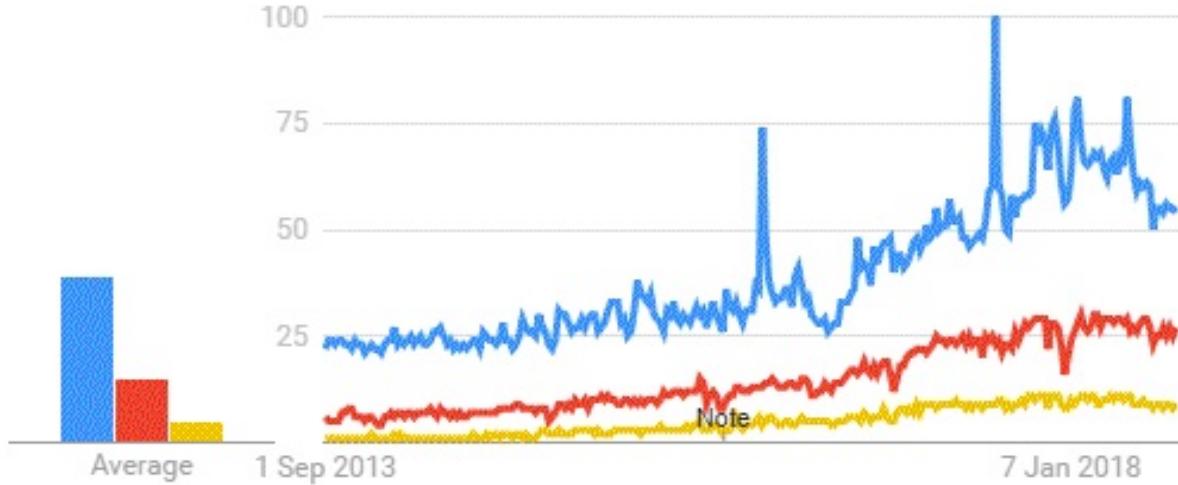
When to use ML or DL?

In the table below, we summarize the difference between machine learning and deep learning.

	Machine learning	Deep learning
Training dataset	Small	Large
Choose features	Yes	No
Number of algorithms	Many	Few
Training time	Short	Long

With machine learning, you need fewer data to train the algorithm than deep learning. Deep learning requires an extensive and diverse set of data to identify the underlying structure. Besides, machine learning provides a faster-trained model. Most advanced deep learning architecture can take days to a week to train. The advantage of deep learning over machine learning is it is highly accurate. You do not need to understand what features are the best representation of the data; the neural network learned how to select critical features. In machine learning, you need to choose for yourself what features to include in the model.

● Artificial intelligence ● machine learning ● deep learning



Summary

Artificial intelligence is imparting a cognitive ability to a machine. Early AI systems used pattern matching and expert systems.

The idea behind machine learning is that the machine can learn without human intervention. The machine needs to find a way to learn how to solve a task given the data.

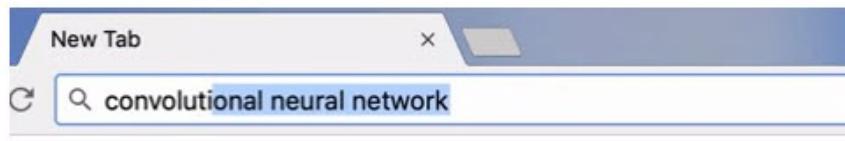
Deep learning is the breakthrough in the field of artificial intelligence. When there is enough data to train on, deep learning achieves impressive results, especially for image recognition and text translation. The main reason is the feature extraction is done automatically in the different layers of the network.

Chapter 3: What is TensorFlow?

What is TensorFlow?

Currently, the most famous deep learning library in the world is Google's TensorFlow. Google product uses machine learning in all of its products to improve the search engine, translation, image captioning or recommendations.

To give a concrete example, Google users can experience a faster and more refined the search with AI. If the user types a keyword a the search bar, Google provides a recommendation about what could be the next word.



Google wants to use machine learning to take advantage of their massive datasets to give users the best experience. Three different groups use machine learning:

- Researchers
- Data scientists
- Programmers.

They can all use the same toolset to collaborate with each other and improve their efficiency.

Google does not just have any data; they have the world's most massive computer, so TensorFlow was built to scale. TensorFlow is a library developed by the Google Brain Team to accelerate machine learning and deep neural network research.

It was built to run on multiple CPUs or GPUs and even mobile operating systems, and it has several wrappers in several languages like Python, C++ or Java.

History of TensorFlow

A couple of years ago, deep learning started to outperform all other machine learning algorithms when giving a massive amount of data. Google saw it could use these deep neural networks to improve its services:

- Gmail
- Photo
- Google search engine

They build a framework called **Tensorflow** to let researchers and developers work together on an AI model. Once developed and scaled, it allows lots of people to use it.

It was first made public in late 2015, while the first stable version appeared in 2017. It is open source under Apache Open Source license. You can use it, modify it and redistribute the modified version for a fee without paying anything to Google.

TensorFlow Architecture

Tensorflow architecture works in three parts:

- Preprocessing the data
- Build the model
- Train and estimate the model

It is called Tensorflow because it takes input as a multi-dimensional array, also known as **tensors**. You can construct a sort of **flowchart** of operations (called a Graph) that you want to perform on that input. The input goes in at one end, and then it flows through this system of multiple operations and comes out the other end as output.

This is why it is called TensorFlow because the tensor goes in it flows through a list of operations, and then it comes out the other side.

Where can Tensorflow run?

TensorFlow can hardware, and software requirements can be classified into

Development Phase: This is when you train the mode. Training is usually done on your Desktop or laptop.

Run Phase or Inference Phase: Once training is done Tensorflow can be run on many different platforms. You can run it on

- Desktop running Windows, macOS or Linux
- Cloud as a web service
- Mobile devices like iOS and Android

You can train it on multiple machines then you can run it on a different machine, once you have the trained model.

The model can be trained and used on GPUs as well as CPUs. GPUs were initially designed for video games. In late 2010, Stanford researchers found that GPU was also very good at matrix operations and algebra so that it makes them very fast for doing these kinds of calculations. Deep learning relies on a lot of matrix multiplication. TensorFlow is very fast at computing the matrix multiplication because it is written in C++. Although it is implemented in C++, TensorFlow can be accessed and controlled by other languages mainly, Python.

Finally, a significant feature of TensorFlow is the TensorBoard. The TensorBoard enables to monitor graphically and visually what TensorFlow is doing.

Introduction to Components of TensorFlow

Tensor

Tensorflow's name is directly derived from its core framework: **Tensor**. In Tensorflow, all the computations involve tensors. A tensor is a **vector** or **matrix** of n-dimensions that represents all types of data. All values in a tensor hold identical data type with a known (or partially known) **shape**. The shape of the data is the dimensionality of the matrix or array.

A tensor can be originated from the input data or the result of a computation. In TensorFlow, all the operations are conducted inside a **graph**. The graph is a set of computation that takes place successively. Each operation is called an **op node** and are connected to each other.

The graph outlines the ops and connections between the nodes. However, it does not display the values. The edge of the nodes is the tensor, i.e., a way to populate the operation with data.

Graphs

TensorFlow makes use of a graph framework. The graph gathers and describes all the series computations done during the training. The graph has lots of advantages:

- It was done to run on multiple CPUs or GPUs and even mobile operating system
- The portability of the graph allows to preserve the computations for immediate or later use. The graph can be saved to be executed in the future.
- All the computations in the graph are done by connecting tensors together
 - A tensor has a node and an edge. The node carries the mathematical operation and produces an endpoints outputs. The edges the edges

explain the input/output relationships between nodes.

Why is TensorFlow popular?

TensorFlow is the best library of all because it is built to be accessible for everyone. Tensorflow library incorporates different API to built at scale deep learning architecture like CNN or RNN. TensorFlow is based on graph computation; it allows the developer to visualize the construction of the neural network with Tensorboard. This tool is helpful to debug the program. Finally, Tensorflow is built to be deployed at scale. It runs on CPU and GPU.

Tensorflow attracts the largest popularity on GitHub compare to the other deep learning framework.

List of Prominent Algorithms supported by TensorFlow

Currently, TensorFlow 1.10 has a built-in API for:

- Linear regression: `tf.estimator.LinearRegressor`
- Classification: `tf.estimator.LinearClassifier`
- Deep learning classification: `tf.estimator.DNNClassifier`
- Deep learning wipe and deep: `tf.estimator.DNNLinearCombinedClassifier`
- Booster tree regression: `tf.estimator.BoostedTreesRegressor`
- Boosted tree classification: `tf.estimator.BoostedTreesClassifier`

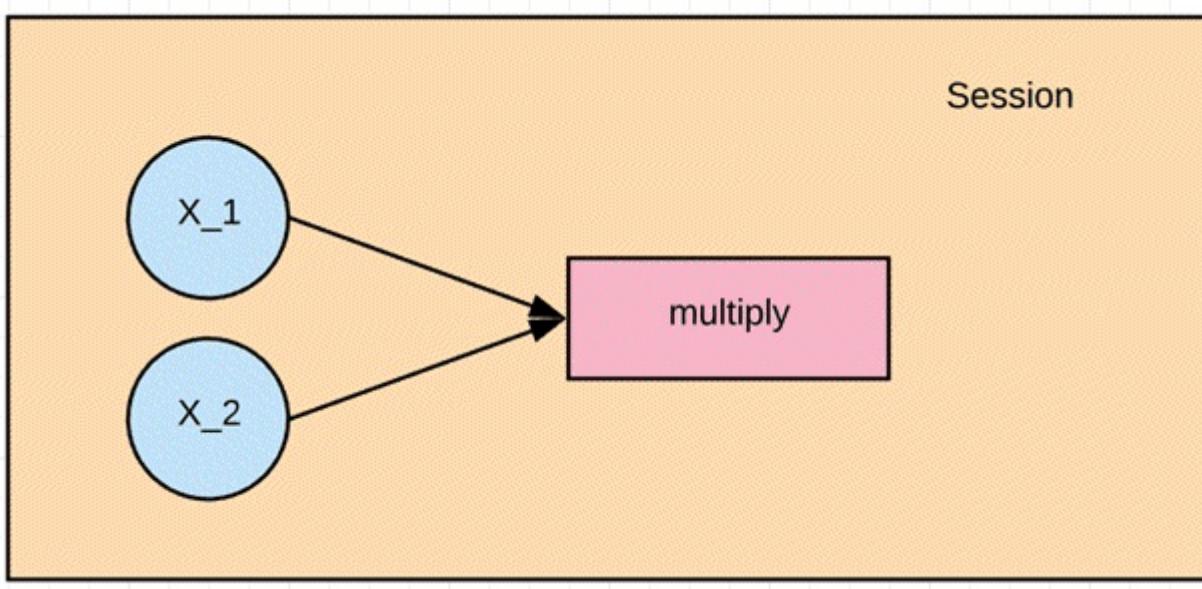
Simple TensorFlow Example

```
import numpy as np  
import tensorflow as tf
```

In the first two line of code, we have imported tensorflow as tf. With Python, it is a common practice to use a short name for a library. The advantage is to avoid to type the full name of the library when we need to use it. For instance, we can import tensorflow as tf, and call tf when we want to use a tensorflow function

Let's practice the elementary workflow of Tensorflow with a simple example. Let's create a computational graph that multiplies two numbers together.

During the example, we will multiply X_1 and X_2 together. Tensorflow will create a node to connect the operation. In our example, it is called multiply. When the graph is determined, Tensorflow computational engines will multiply together X_1 and X_2.



Finally, we will run a TensorFlow session that will run the computational graph with the values of X_1 and X_2 and print the result of the multiplication.

Let's define the X_1 and X_2 input nodes. When we create a node in Tensorflow, we have to choose what kind of node to create. The X1 and X2 nodes will be a placeholder node. The placeholder assigns a new value each time we make a calculation. We will create them as a TF dot placeholder node.

Step 1: Define the variable

```
X_1 = tf.placeholder(tf.float32, name = "X_1")
X_2 = tf.placeholder(tf.float32, name = "X_2")
```

When we create a placeholder node, we have to pass in the data type will be adding numbers here so we can use a floating-point data type, let's use tf.float32. We also need to give this node a name. This name will show up when we look at the graphical visualizations of our model. Let's name this node X_1 by passing in a parameter called name with a value of X_1 and now let's define X_2 the same way. X_2.

Step 2: Define the computation

```
multiply = tf.multiply(X_1, X_2, name = "multiply")
```

Now we can define the node that does the multiplication operation. In Tensorflow we can do that by creating a tf.multiply node.

We will pass in the X_1 and X_2 nodes to the multiplication node. It tells tensorflow to link those nodes in the computational graph, so we are asking it to pull the values from x and y and multiply the result. Let's also give the multiplication node the name multiply. It is the entire definition for our simple computational graph.

Step 3: Execute the operation

To execute operations in the graph, we have to create a session. In Tensorflow, it is done by tf.Session(). Now that we have a session we can ask the session to run operations on our computational graph by calling session. To run the

computation, we need to use run.

When the addition operation runs, it is going to see that it needs to grab the values of the X_1 and X_2 nodes, so we also need to feed in values for X_1 and X_2. We can do that by supplying a parameter called feed_dict. We pass the value 1,2,3 for X_1 and 4,5,6 for X_2.

We print the results with print(result). We should see 4, 10 and 18 for 1x4, 2x5 and 3x6

```
X_1 = tf.placeholder(tf.float32, name = "X_1")
X_2 = tf.placeholder(tf.float32, name = "X_2")

multiply = tf.multiply(X_1, X_2, name = "multiply")

with tf.Session() as session:
    result = session.run(multiply, feed_dict={X_1:[1,2,3], X_2:
[4,5,6]})
    print(result)
```

```
[ 4. 10. 18.]
```

Options to Load Data into TensorFlow

The first step before training a machine learning algorithm is to load the data. There are two common ways to load data:

1. Load data into memory: It is the simplest method. You load all your data into memory as a single array. You can write a Python code. These lines of code are unrelated to Tensorflow.
2. Tensorflow data pipeline. Tensorflow has built-in API that helps you to load the data, perform the operation and feed the machine learning algorithm easily. This method works very well especially when you have a large dataset. For instance, image records are known to be enormous and do not fit into memory. The data pipeline manages the memory by itself

What solution to use?

Load data in memory

If your dataset is not too big, i.e., less than 10 gigabytes, you can use the first method. The data can fit into the memory. You can use a famous library called Pandas to import CSV files. You will learn more about pandas in the next tutorial.

Load data with Tensorflow pipeline

The second method works best if you have a large dataset. For instance, if you have a dataset of 50 gigabytes, and your computer has only 16 gigabytes of memory then the machine will crash.

In this situation, you need to build a Tensorflow pipeline. The pipeline will load the data in batch, or small chunk. Each batch will be pushed to the pipeline and be ready for the training. Building a pipeline is an excellent solution because it

allows you to use parallel computing. It means Tensorflow will train the model across multiple CPUs. It fosters the computation and permits for training powerful neural network.

You will see in the next tutorials on how to build a significant pipeline to feed your neural network.

In a nutshell, if you have a small dataset, you can load the data in memory with Pandas library.

If you have a large dataset and you want to make use of multiple CPUs, then you will be more comfortable to work with Tensorflow pipeline.

Create Tensorflow pipeline

In the example before, we manually add three values for X_1 and X_2. Now we will see how to load data to Tensorflow.

Step 1) Create the data

First of all, let's use numpy library to generate two random values.

```
import numpy as np  
x_input = np.random.sample((1,2))  
print(x_input)
```

[[0.8835775 0.23766977]]

Step 2: Create the placeholder

Like in the previous example, we create a placeholder with the name X. We need to specify the shape of the tensor explicitly. In case, we will load an array with only two values. We can write the shape as shape=[1,2]

```
# using a placeholder  
x = tf.placeholder(tf.float32, shape=[1,2], name = 'X')
```

Step 3: Define the dataset method

next, we need to define the Dataset where we can populate the value of the placeholder x. We need to use the method tf.data.Dataset.from_tensor_slices

```
dataset = tf.data.Dataset.from_tensor_slices(x)
```

Step 4: Create the pipeline

In step four, we need to initialize the pipeline where the data will flow. We need to create an iterator with make_initializer_iterator. We name it iterator. Then

we need to call this iterator to feed the next batch of data, `get_next`. We name this step `get_next`. Note that in our example, there is only one batch of data with only two values.

```
iterator = dataset.make_initializable_iterator()  
get_next = iterator.get_next()
```

Step 5: Execute the operation

The last step is similar to the previous example. We initiate a session, and we run the operation iterator. We feed the `feed_dict` with the value generated by numpy. These two value will populate the placeholder `x`. Then we run `get_next` to print the result.

```
with tf.Session() as sess:  
    # feed the placeholder with data  
    sess.run(iterator.initializer, feed_dict={ x: x_input })  
    print(sess.run(get_next)) # output [ 0.52374458  0.71968478]
```

```
[0.8835775  0.23766978]
```

Summary

TensorFlow is the most famous deep learning library these recent years. A practitioner using TensorFlow can build any deep learning structure, like CNN, RNN or simple artificial neural network.

TensorFlow is mostly used by academics, startups, and large companies. Google uses TensorFlow in almost all Google daily products including Gmail, Photo and Google Search Engine.

Google Brain team's developed TensorFlow to fill the gap between researchers and products developers. In 2015, they made TensorFlow public; it is rapidly growing in popularity. Nowadays, TensorFlow is the deep learning library with the most repositories on GitHub.

Practitioners use Tensorflow because it is easy to deploy at scale. It is built to work in the cloud or on mobile devices like iOS and Android.

Tensorflow works in a session. Each session is defined by a graph with different computations. A simple example can be to multiply two numbers. In Tensorflow, three steps are required:

1. Define the variable

```
x_1 = tf.placeholder(tf.float32, name = "X_1")
x_2 = tf.placeholder(tf.float32, name = "X_2")
```

2. Define the computation

```
multiply = tf.multiply(x_1, x_2, name = "multiply")
```

3. Execute the operation

```
with tf.Session() as session:
    result = session.run(multiply, feed_dict={x_1:[1,2,3], x_2:[4,5,6]})
    print(result)
```

One common practice in Tensorflow is to create a pipeline to load the data. If you follow these five steps, you'll be able to load data to TensorFlow

1. Create the data

```
import numpy as np
x_input = np.random.sample((1,2))
print(x_input)
```

2. Create the placeholder

```
x = tf.placeholder(tf.float32, shape=[1,2], name = 'X')
```

3. Define the dataset method

```
dataset = tf.data.Dataset.from_tensor_slices(x)
```

4. Create the pipeline

```
iterator = dataset.make_initializable_iterator() get_next =  
iteraror.get_next()
```

5. Execute the program

```
with tf.Session() as sess:  
sess.run(iterator.initializer, feed_dict={ x: x_input })  
print(sess.run(get_next))
```

Chapter 4: Comparison of Deep Learning Libraries

Artificial intelligence is growing in popularity since 2016 with, 20% of the big companies using AI in their businesses (McKinsey report, 2018). As per the same report AI can create substantial value across industries. In banking, for instance, the potential of AI is estimated at \$ 300 billion, in retail the number skyrocket to \$ 600 billion.

To unlock the potential value of AI, companies must choose the right deep learning framework. In this tutorial, you will learn about the different libraries available to carry out deep learning tasks. Some libraries have been around for years while new library like TensorFlow has come to light in recent years.

8 Best Deep learning Libraries /Framework

In this list, we will compare the top Deep learning frameworks. All of them are open source and popular in the data scientist community. We will also compare popular ML as a service providers

Torch

Torch is an old open source machine learning library. It is first released was 15 years ago. Its primary programming language is LUA, but has an implementation in C. Torch supports a vast library for machine learning algorithms, including deep learning. It supports CUDA implementation for parallel computation.

Torch is used by most of the leading labs such as Facebook, Google, Twitter, Nvidia, and so on. Torch has a library in Python named Pytorch.

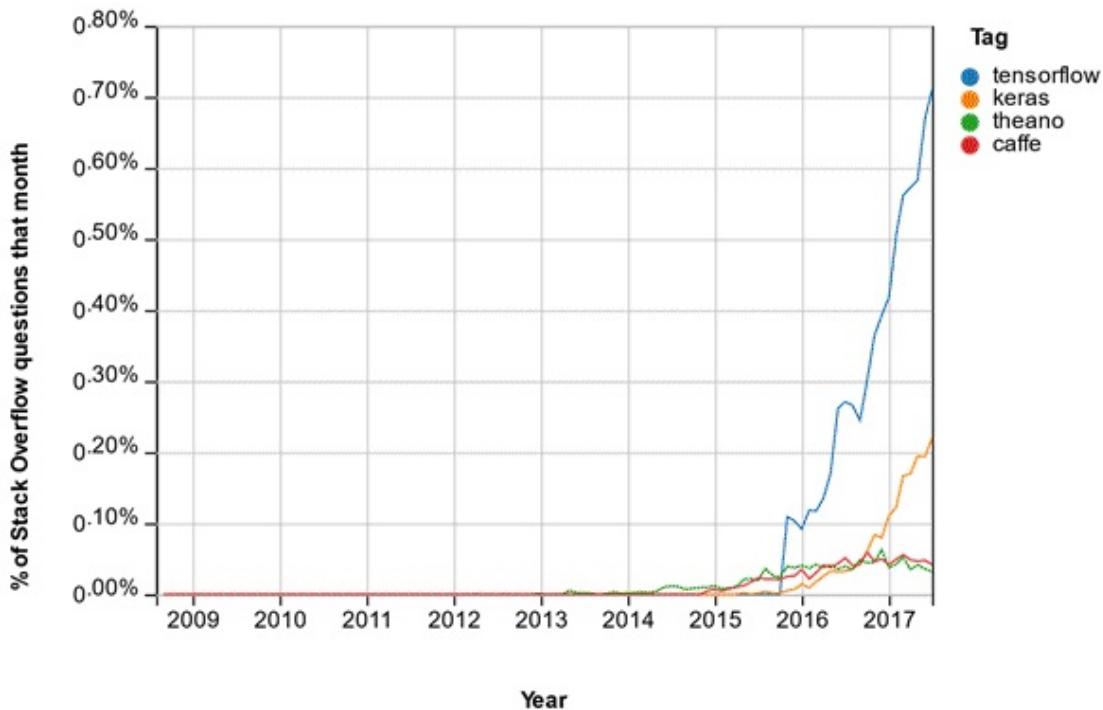
Infer.net

Infer.net is developed and maintained by Microsoft. Infer.net is a library with a primary focus on the Bayesian statistic. Infer.net is designed to offer practitioners state-of-the-art algorithms for probabilistic modeling. The library contains analytical tools such as Bayesian analysis, hidden Markov chain, clustering.

Keras

Keras is a Python framework for deep learning. It is a convenient library to construct any deep learning algorithm. The advantage of Keras is that it uses the same Python code to run on CPU or GPU. Besides, the coding environment is pure and allows for training state-of-the-art algorithm for computer vision, text recognition among other.

Keras has been developed by François Chollet, a researcher at Google. Keras is used in prominent organizations like CERN, Yelp, Square or Google, Netflix, and Uber.



Theano

Theano is deep learning library developed by the Université de Montréal in 2007. It offers fast computation and can be run on both CPU and GPU. Theano has been developed to train deep neural network algorithms.

MICROSOFT COGNITIVE TOOLKIT(CNTK)

Microsoft toolkit, previously known as CNTK, is a deep learning library developed by Microsoft. According to Microsoft, the library is among the fastest on the market. Microsoft toolkit is an open-source library, although Microsoft is using it extensively for its products like Skype, Cortana, Bing, and Xbox. The toolkit is available both in Python and C++.

MXNet

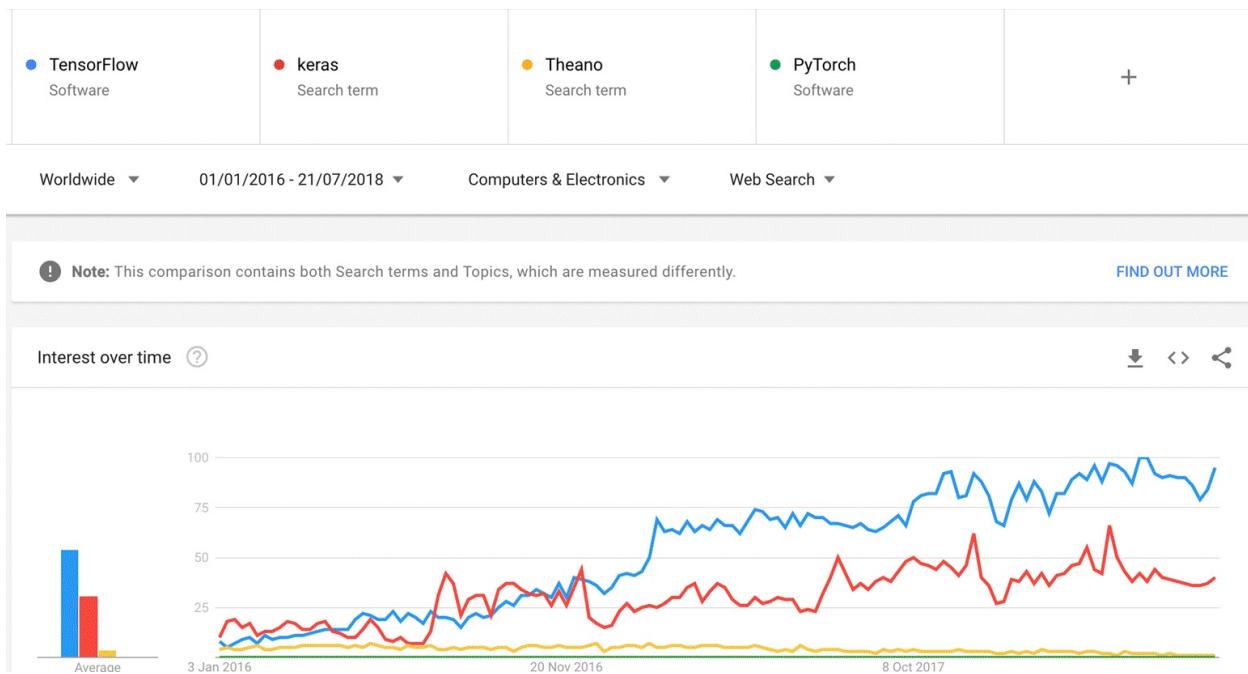
MXNet is a recent deep learning library. It is accessible with multiple programming languages including C++, Julia, Python and R. MXNet can be configured to work on both CPU and GPU. MXNet includes state-of-the-art deep learning architecture such as Convolutional Neural Network and Long Short-Term Memory. MXNet is built to work in harmony with dynamic cloud infrastructure. The main user of MXNet is Amazon.

Caffe

Caffe is a library built by Yangqing Jia when he was a PhD student at Berkeley. Caffe is written in C++ and can perform computation on both CPU and GPU. The primary uses of Caffe is Convolutional Neural Network. Although, In 2017, Facebook extended Caffe with more deep learning architecture, including Recurrent Neural Network. Caffe is used by academics and startups but also some large companies like Yahoo!.

TensorFlow

TensorFlow is Google's open source project. TensorFlow is the most famous deep learning library these days. It was released to the public in late 2015.



TensorFlow is developed in C++ and has convenient Python API, although C++ APIs are also available. Prominent companies like Airbus, Google, IBM and so on are using TensorFlow to produce deep learning algorithms.

TensorFlow Vs Theano Vs Torch Vs Keras Vs infer.net Vs CNTK Vs MXNet Vs Caffe: Key Differences

Library	Platform	Written in	Cuda support	Parallel Execution	Has trained models	RNN	CNN
Torch	Linux, MacOS, Windows	Lua	Yes	Yes	Yes	Yes	Yes
Infer.Net	Linux, MacOS, Windows	Visual Studio	No	No	No	No	No
Keras	Linux, MacOS, Windows	Python	Yes	Yes	Yes	Yes	Yes
Theano	Cross-platform	Python	Yes	Yes	Yes	Yes	Yes
TensorFlow	Linux, MacOS, Windows, Android	C++, Python, CUDA	Yes	Yes	Yes	Yes	Yes
MICROSOFT COGNITIVE TOOLKIT	Linux, Windows, Mac with Docker	C++	Yes	Yes	Yes	Yes	Yes
Caffe	Linux, MacOS, Windows	C++	Yes	Yes	Yes	Yes	Yes
MXNet	Linux, Windows, MacOs, Android, iOS, Javascript	C++	Yes	Yes	Yes	Yes	Yes

Verdict:

TensorFlow is the best library of all because it is built to be accessible for everyone. Tensorflow library incorporates different API to built at scale deep learning architecture like CNN or RNN. TensorFlow is based on graph computation, it allows the developer to visualize the construction of the neural network with Tensorboard. This tool is helpful to debug the program. Finally, Tensorflow is built to be deployed at scale. It runs on CPU and GPU.

Tensorflow attracts the largest popularity on GitHub compare to the other deep

learning framework.

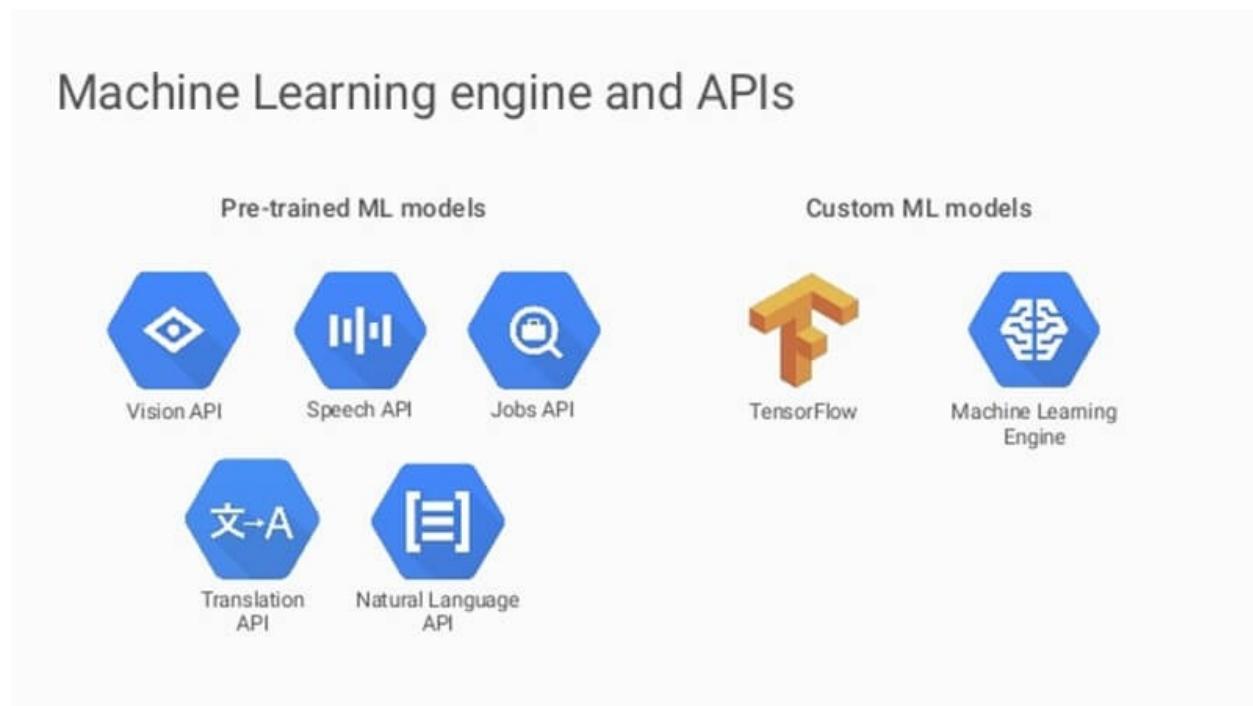
Comparing Machine Learning as a Service

Following are 4 popular DL as a service providers

Google Cloud ML

Google provides for developer pre-trained model available in Cloud AutoML. This solution exists for a developer without a strong background in machine learning. Developers can use state-of-the-art Google's pre-trained model on their data. It allows any developers to train and evaluate any model in just a few minutes.

Google currently provides a REST API for computer vision, speech recognition, translation, and NLP.



Using Google Cloud, you can train a machine learning framework build on

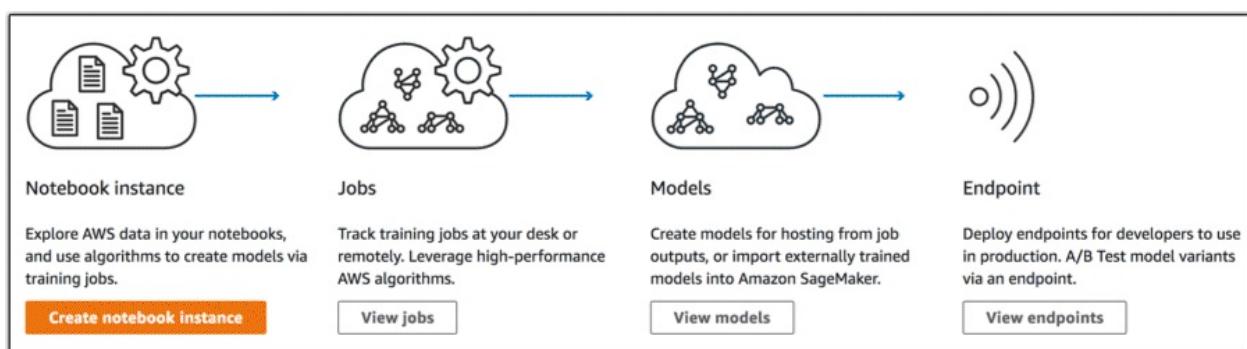
TensorFlow, Scikit-learn, XGBoost or Keras. Google Cloud machine learning will train the models across its cloud.

The advantage to use Google cloud computing is the simplicity to deploy machine learning into production. There is no need to set up Docker container. Besides, the cloud takes care of the infrastructure. It knows how to allocate resources with CPUs, GPUs, and TPUs. It makes the training faster with paralleled computation.

AWS SageMaker

A major competitor to Google Cloud is Amazon cloud, AWS. Amazon has developed Amazon SageMaker to allow data scientists and developers to build, train and bring into production any machine learning models.

SageMaker is available in a Jupyter Notebook and includes the most used machine learning library, TensorFlow, MXNet, Scikit-learn amongst others. Programs written with SageMaker are automatically run in the Docker containers. Amazon handles the resource allocation to optimize the training and deployment.



Amazon provides API to the developers in order to add intelligence to their applications. In some occasion, there is no need to reinventing-the-wheel by building from scratch new models while there are powerful pre-trained models in the cloud. Amazon provides API services for computer vision, conversational

chatbots and language services:

The three major available API are:

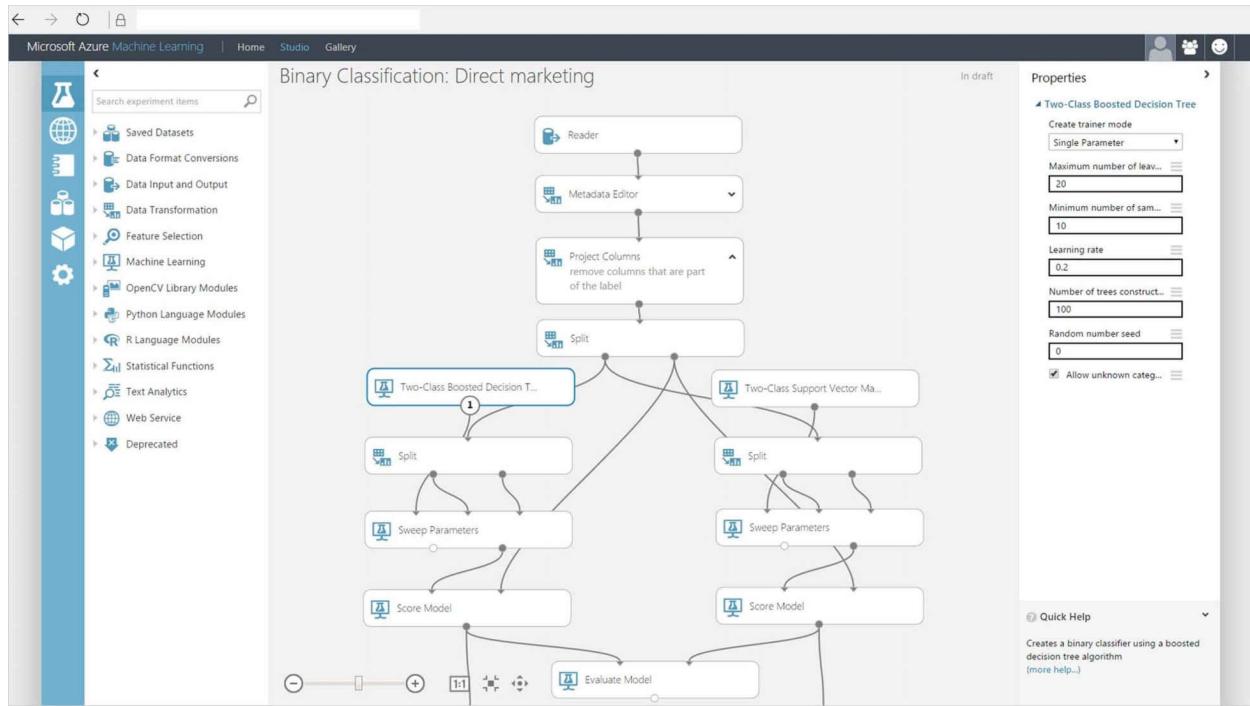
- Amazon Rekognition: provides image and video recognition to an app
- Amazon Comprehend: Perform text mining and neural language processing to, for instance, automatize the process of checking the legality of financial document
- Amazon Lex: Add chatbot to an app

Azure Machine Learning Studio

Probably one of the friendliest approaches to machine learning is Azure Machine Learning Studio. The significant advantage of this solution is that no prior programming knowledge is required.

Microsoft Azure Machine Learning Studio is a drag-and-drop collaborative tool to create, train, evaluate and deploy machine learning solution. The model can be efficiently deployed as web services and used in several apps like Excel.

Azure Machine learning interface is interactive, allowing the user to build a model just by dragging and dropping elements quickly.



When the model is ready, the developer can save it and push it to Azure Gallery or Azure Marketplace.

Azure Machine learning can be integrated into R or Python their custom built-in package.

IBM Watson ML

Watson studio can simplify the data projects with a streamlined process that allows extracting value and insights from the data to help the business to get smarter and faster. Watson studio delivers an easy-to-use collaborative data science and machine learning environment for building and training models, preparing and analyzing data, and sharing insights all in one place. Watson Studio is easy to use with a drag-and-drop code.

The screenshot shows the IBM Cloud interface for managing Watson services. On the left, there's a sidebar with 'Manage', 'Service credentials' (which is selected and highlighted in blue), 'Plan', and 'Connections'. The main area is titled 'Watson / sda-ML-predictiveModel-5-15'. It shows the location as 'US South', org as 'scott.dangelo', and space as 'dev'. Below this, under 'Service credentials', there's a table with one item. The table has columns for 'KEY NAME', 'DATE CREATED', and 'ACTIONS'. The single row shows 'apsx-data' as the key name, created on 'May 15, 2018 - 09:17:42', and a 'View credentials' button. The credential details are shown in a modal window below, containing a JSON object:

```
{  
  "url": "https://ibm-watson-ml.mybluemix.net",  
  "username": "REDACTED",  
  "password": "REDACTED",  
  "instance_id": "REDACTED"  
}
```

Watson studio supports some of the most popular frameworks like Tensorflow, Keras, Pytorch, Caffe and can deploy a deep learning algorithm on to the latest GPUs from Nvidia to help accelerate modeling.

Verdict:

In our point of view, Google cloud solution is the one that is the most recommended. Google cloud solution provides lower prices than AWS by at least 30% for data storage and machine learning solution. Google is doing an excellent job to democratize AI. It has developed an open source language, TensorFlow, optimized data warehouse connection, provides tremendous tools from data visualization, data analysis to machine learning. Besides, Google Console is ergonomic and much more comprehensive than AWS or Windows.

Chapter 5: How to Download and Install TensorFlow Windows and Mac

In this tutorial, we will explain how to install **TensorFlow** with **Anaconda**. You will learn how to use TensorFlow with Jupyter. Jupyter is a notebook viewer.

TensorFlow Versions

TensorFlow supports computations across multiple CPUs and GPUs. It means that the computations can be distributed across devices to improve the speed of the training. With parallelization, you don't need to wait for weeks to obtain the results of training algorithms.

For Windows user, TensorFlow provides two versions:

- **TensorFlow with CPU support only:** If your Machine does not run on NVIDIA GPU, you can only install this version
- **TensorFlow with GPU support:** For faster computation, you can use this version of TensorFlow. This version makes sense only if you need strong computational capacity.

During this tutorial, the basic version of TensorFlow is sufficient.

Note: TensorFlow does not provides GPU support on MacOS.

Here is how to proceed

MacOS User:

- Install Anaconda
- Create a .yml file to install Tensorflow and dependencies
- Launch Jupyter Notebook

For Windows

- Install Anaconda
- Create a .yml file to install dependencies
- Use pip to add TensorFlow
- Launch Jupyter Notebook

To run Tensorflow with Jupyter, you need to create an environment within Anaconda. It means you will install Ipython, Jupyter, and TensorFlow in an appropriate folder inside our machine. On top of this, you will add one essential library for data science: "Pandas". The Pandas library helps to manipulate a data frame.

Install Anaconda

Download Anaconda version 4.3.1 (for Python 3.6) for the appropriate system.

Anaconda will help you to manage all the libraries required either for Python or R. Refer this tutorial to install Anaconda

Create .yml file to install Tensorflow and dependencies

It includes

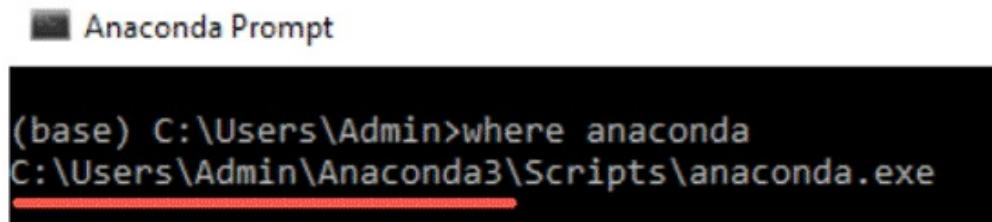
- Locate the path of Anaconda
- Set the working directory to Anaconda
- Create the yml file (For MacOS user, TensorFlow is installed here)
- Edit the yml file
- Compile the yml file
- Activate Anaconda
- Install TensorFlow (Windows user only)

Step 1) Locate Anaconda

The first step you need to do is to locate the path of Anaconda. You will create a new conda environment that includes the necessities libraries you will use during the tutorials about TensorFlow.

Windows

If you are a Windows user, you can use Anaconda Prompt and type: C:\>where anaconda



```
Anaconda Prompt  
(base) C:\Users\Admin>where anaconda  
C:\Users\Admin\Anaconda3\Scripts\anaconda.exe
```

We are interested to know the name of the folder where Anaconda is installed because we want to create our new environment inside this path. For instance, in

the picture above, Anaconda is installed in the Admin folder. For you, it can be the same, i.e. Admin or the user's name.

In the next, we will set the working directory from c:\ to Anaconda3.

MacOS

for MacOS user, you can use the Terminal and type:

```
which anaconda
```

```
Thomass-MacBook-Pro:~ Thomas$ which anaconda  
/Users/Thomas/anaconda3/bin/anaconda
```

Path of
Anaconda

You will need to create a new folder inside Anaconda which will contain **Ipython**, **Jupyter** and **TensorFlow**. A quick way to install libraries and software is to write a yml file.

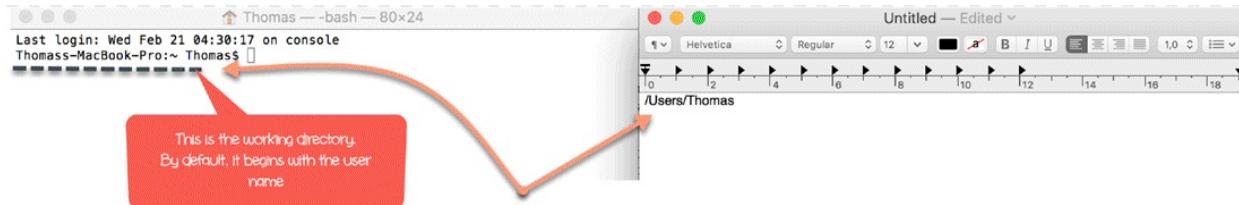
Step 2) Set working directory You need to specify the working directory where you want to create the yml file. As said before, it will be located inside Anaconda.

For MacOS user:

The Terminal sets the default working directory to **Users/USERNAME**. As you can see in the figure below, the path of anaconda3 and the working directory are identical. In MacOS, the latest folder is shown before the \$. The Terminal will install all the libraries in this working directory.

If the path on the text editor does not match the working directory, you can

change it by writing `cd PATH` in the Terminal. PATH is the path you pasted in the text editor. Don't forget to wrap the PATH with '`'PATH'`'. This action will change the working directory to PATH.



Open your Terminal, and type:

```
cd anaconda3
```

For Windows user (make sure of the folder before Anaconda3): `cd C:\Users\Admin\Anaconda3`

or the path "where anaconda" command gives you

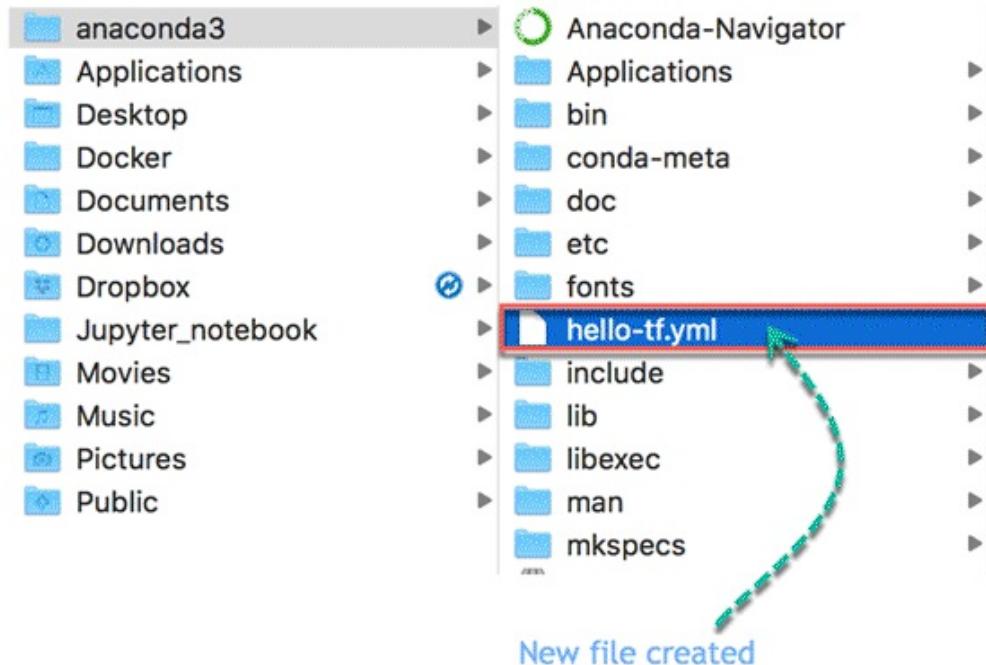
```
base) C:\Users\Admin>cd C:\Users\Admin\Anaconda3  
base) C:\Users\Admin\Anaconda3>
```

Step 3) Create the yml file You can create the yml file inside the new working directory. The file will install the dependencies you need to run TensorFlow. Copy and paste this code into the Terminal.

For MacOS user:

```
touch hello-tf.yml
```

A new file named `hello-tf.yml` should appear inside `anaconda3`



For Windows user:

```
echo.>hello-tf.yml
```

A new file named hello-tf.yml should appear

This PC > Local Disk (C:) > Users > Admin > Anaconda3 >				
	Name	Date modified	Type	Size
	hello-tf.yml	05-04-2018 02:38 ...	YML File	1 KB
	.nonadmin	21-03-2018 12:26 ...	NONADMIN File	0 KB
	Uninstall-Anaconda3.exe	20-02-2018 02:26 ...	Application	296 KB

Step 4) Edit the yml file For MacOS user:

You are ready to edit the yml file. You can paste the following code in the Terminal to edit the file. MacOS user can use **vim** to edit the yml file.

```
vi hello-tf.yml
```

So far, your Terminal looks like this

```

[Thomass-MacBook-Pro:~ Thomas$ cd anaconda3
[Thomass-MacBook-Pro:anaconda3 Thomas$ touch hello-tf.yml
[Thomass-MacBook-Pro:anaconda3 Thomas$ vi hello-tf.yml

```

1 Set path
2 Create yml file
3 Edit yml file

You enter an **edit** mode. Inside this mode, you can, after pressing esc:

- Press i to edit
- Press w to save
- Press q! to quit

Write the following code in the edit mode and press esc followed by :w

```

name: hello-tf
dependencies:
  - python=3.6
  - jupyter
  - ipython
  - numpy
  - pandas
  - pip:
    - https://storage.googleapis.com/tensorflow/mac/cpu/tensorflow-1.5.0-py3-none-any.whl

```

1 Edit the yml file
2 Save the file

Note: The file is case **and** intend sensitive. 2 spaces are required after each intend.

For MacOS

```
name: hello-tf
dependencies:
  - python=3.6
  - jupyter
  - ipython
  - pandas
  - pip:
    - https://storage.googleapis.com/tensorflow/MacOS/cpu/tensorflow-1.5.0-py3-none-any.whl
```

Code Explanation

- name: hello-tf: Name of the yml file
- dependencies:
- python=3.6
- jupyter
- ipython
- pandas: Install Python version 3.6, Jupyter, Ipython, and pandas libraries
- pip: Install a Python library
 - https://storage.googleapis.com/tensorflow/MacOS/cpu/tensorflow-1.5.0-py3-none-any.whl: Install TensorFlow from Google apis.

Press esc followed by :q! to quite the edit mode.

For Windows User:

Windows does not have vim program, so the Notepad is enough to complete this step.

notepad hello-tf.yml

Enter following into the file

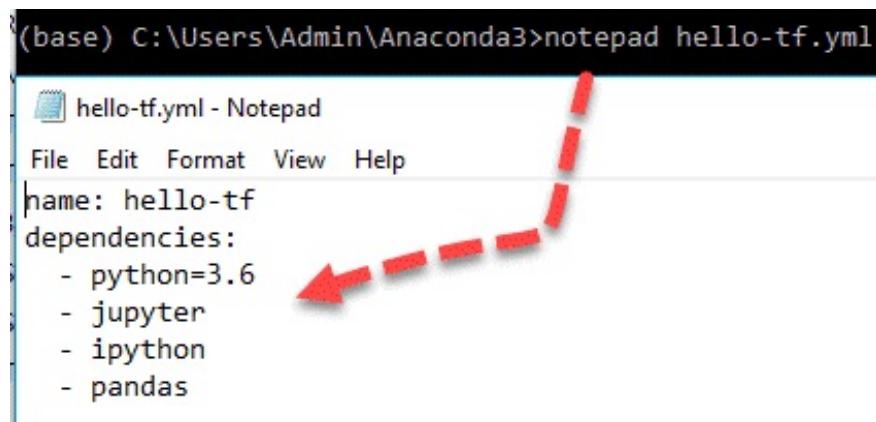
```
name: hello-tfdependencies:  
- python=3.6  
- jupyter  
- ipython  
- pandas
```

Code Explanation

- name: hello-tf: Name of the yml file
 - dependencies:

- python=3.6
- jupyter
- ipython
- pandas: Install Python version 3.6, Jupyter, Ipython, and pandas libraries

It will open the notepad, you can edit the file from here.



```
(base) C:\Users\Admin\Anaconda3>notepad hello-tf.yml
hello-tf.yml - Notepad
File Edit Format View Help
name: hello-tf
dependencies:
- python=3.6
- jupyter
- ipython
- pandas
```

Note: Windows users will install TensorFlow in the next step. In this step, you only prepare the conda environment **Step 5)** Compile the yml file You can compile the .yml file with the following code :

```
conda env create -f hello-tf.yml
```

Note: For Windows users, the new environment is created inside the current user directory.

It takes times. It will take around 1.1gb of space in your hard disk.

Installing libraries

Downloading and Extracting Packages

python-dateutil 2.7.2: #####	100%
openssl 1.0.2o: #####	100%
imagesize 1.0.0: #####	100%
pytz 2018.3: #####	100%
cffi 1.11.5: #####	100%
tornado 5.0.1: #####	100%
setuptools 39.0.1: #####	100%
jupyter_client 5.2.3: #####	100%
python 3.6.5: #####	100%
zeromq 4.2.3: #####	100%
pexpect 4.4.0: #####	100%
bleach 2.1.3: #####	100%
babel 2.5.3: #####	100%
asn1crypto 0.24.0: #####	100%
libsodium 1.0.16: #####	100%
typing 3.6.4: #####	100%
sphinx 1.7.2: #####	100%
jedi 0.11.1: #####	100%
pip 9.0.3: #####	100%
packaging 17.1: #####	100%
ipywidgets 7.1.2: #####	100%
pysocks 1.6.8: #####	100%
ipykernel 4.8.2: #####	100%
send2trash 1.5.0: #####	100%
pyopenssl 17.5.0: #####	100%
widgetsnbextension 3.1.4: #####	100%
cryptography 2.2.1: #####	100%
pyzmq 17.0.0: #####	100%
notebook 5.4.1: #####	100%

In Windows

```
(base) C:\Users\Admin\Anaconda3>conda env create -f hello-tf.yml
Solving environment: done
```

```
==> WARNING: A newer version of conda exists. <==
  current version: 4.4.10
  latest version: 4.5.0
```

```
Please update conda by running
```

```
$ conda update -n base conda
```

```
Downloading and Extracting Packages
```

```
bleach 2.1.3: #####
numpy 1.14.2: #####
sip 4.19.8: #####
jedi 0.11.1: #####
notebook 5.4.1: #####
mkl 2018.0.2: #####1
```

Step 6) Activate conda environment We are almost done. You have now 2 conda environments. You created an isolated conda environment with the libraries you will use during the tutorials. This is a recommended practice because each machine learning project requires different libraries. When the project is over, you can remove or not this environment.

```
conda env list
```

```

Thomas-MACBOOK-Pro:~ Thomas$ conda env list
# conda environment:
#
base           * /Users/Thomas/anaconda3
hello-tf        /Users/Thomas/anaconda3/envs/hello-tf

```

Indicate the environment currently running

Main conda environment

Conda with TensorFlow

The asterix indicates the default one. You need to switch to hello-tf to activate the environment For MacOS user:

```
source activate hello-tf
```

For Windows user:

```
activate hello-tf
```

```

#
# To activate this environment, use:
# > source activate hello-tf
#
# To deactivate an active environment, use:
# > source deactivate
#

```

You can check all dependencies are in the same environment. This is important because it allows Python to use Jupyter and TensorFlow from the same environment. If you don't see the three of them located in the same folder, you need to start all over again.

For MacOS user:

```
which python
which jupyter
which ipython
```

```
Thomass-MacBook-Pro:anaconda3 Thomas$ which python
/Users/Thomas/anaconda3/bin/python
Thomass-MacBook-Pro:anaconda3 Thomas$ which jupyter
/Users/Thomas/anaconda3/bin/jupyter
Thomass-MacBook-Pro:anaconda3 Thomas$ which ipython
/Users/Thomas/anaconda3/bin/ipython
```

Optional: You can check for update.

```
pip install --upgrade tensorflow
```

Step 7) For Windows user only: Install TensorFlow For windows user:

```
where python
where jupyter
where ipython
```

```
(hello-tf) C:\Users\Admin\Anaconda3>where python
C:\Users\Admin\Anaconda3\python.exe
C:\Users\Admin\Anaconda3\envs\hello-tf\python.exe

(hello-tf) C:\Users\Admin\Anaconda3>where jupyter
C:\Users\Admin\Anaconda3\envs\hello-tf\Scripts\jupyter.exe
C:\Users\Admin\Anaconda3\Scripts\jupyter.exe

(hello-tf) C:\Users\Admin\Anaconda3>where ipython
C:\Users\Admin\Anaconda3\envs\hello-tf\Scripts\ipython.exe
C:\Users\Admin\Anaconda3\Scripts\ipython.exe

(hello-tf) C:\Users\Admin\Anaconda3>
```

As you can see, you now have two Python environments. The main one and the newly created one i.e. hello-tf. The main conda environment does not have TensorFlow installed only hello-tf. From the picture, python, jupyter and ipython are installed in the same environment. It means, you can use TensorFlow with a Jupyter Notebook.

You need to install Tensorflow with the following command. Only for Windows

user pip install tensorflow

```
(hello-tf) C:\Users\Admin\Anaconda3>pip install tensorflow
Collecting tensorflow
  Downloading tensorflow-1.7.0-cp36-cp36m-win_amd64.whl (33.1MB)
    100% |██████████| 33.1MB 46kB/s
Collecting grpcio>=1.8.6 (from tensorflow)
  Downloading grpcio-1.10.0-cp36-cp36m-win_amd64.whl (1.3MB)
    100% |██████████| 1.3MB 1.2MB/s
Collecting absl-py>=0.1.6 (from tensorflow)
  Downloading absl-py-0.1.13.tar.gz (80kB)
    100% |██████████| 81kB 7.4MB/s
Requirement already satisfied: wheel>=0.26 in c:\users\admin\anaconda3\envs\hello-tf
Requirement already satisfied: numpy>=1.13.3 in c:\users\admin\anaconda3\envs\hello-tf
Requirement already satisfied: six>=1.10.0 in c:\users\admin\anaconda3\envs\hello-tf
Collecting termcolor>=1.1.0 (from tensorflow)
  Downloading termcolor-1.1.0.tar.gz
Collecting tensorboard<1.8.0,>=1.7.0 (from tensorflow)
  Downloading tensorboard-1.7.0-py3-none-any.whl (3.1MB)
    99% |██████████| 3.1MB 14.6MB/s eta 0:00:01
```

Launch Jupyter Notebook

This part is the same for both OS.

You can open TensorFlow with Jupyter.

Note: Each time you want to open TensorFlow, you need to initialize the environment You will proceed as follow:

- Activate hello-tf conda environment
- Open Jupyter
- Import tensorflow
- Delete Notebook
- Close Jupyter

Step 1) Activate conda

For MacOS user:

```
source activate hello-tf
```

For Windows user:

```
conda activate hello-tf
```

```
[Thomass-MacBook-Pro:anaconda3 Thomas$ source activate hello-tf  
(hello-tf) Thomass-MacBook-Pro:anaconda3 Thomas$]
```

Step 2) Open Jupyter

After that, you can open Jupyter from the Terminal

```
jupyter notebook
```

```

[Thomas-MacBook-Pro:anaconda3 Thomas$ source activate hello-tf
(hello-tf) Thomas-MacBook-Pro:anaconda3 Thomas$ jupyter notebook
[W 16:20:59.106 NotebookApp] Error loading server extension jupyter_tensorboard
  Traceback (most recent call last):
    File "/Users/Thomas/anaconda3/envs/hello-tf/lib/python3.6/site-packages/notebook/notebookapp.py", line 1451, in init_server_extensions
      mod = importlib.import_module(modulename)
    File "/Users/Thomas/anaconda3/envs/hello-tf/lib/python3.6/importlib/__init__.py", line 126, in import_module
      return _bootstrap._gcd_import(name[level:], package, level)
    File "<frozen importlib._bootstrap>", line 994, in _gcd_import
    File "<frozen importlib._bootstrap>", line 971, in _find_and_load
    File "<frozen importlib._bootstrap>", line 953, in _find_and_load_unlocked
ModuleNotFoundError: No module named 'jupyter_tensorboard'
[I 16:20:59.113 NotebookApp] Serving notebooks from local directory: /Users/Thomas/anaconda3
[I 16:20:59.113 NotebookApp] 0 active kernels
[I 16:20:59.113 NotebookApp] The Jupyter Notebook is running at:
[I 16:20:59.113 NotebookApp] http://localhost:8888/?token=9ee3a11fea254e6982421366001678d5a6c01cebc8d5658e
[I 16:20:59.113 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 16:20:59.114 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
  http://localhost:8888/?token=9ee3a11fea254e6982421366001678d5a6c01cebc8d5658e
[I 16:20:59.378 NotebookApp] Accepting one-time-token-authenticated connection from ::1
[W 16:21:00.612 NotebookApp] 404 GET /api/tensorboard?_=1522570860085 (::1) 24.15ms referer=http://localhost:8888/tree

```

If the browser does not open automatically, copy this URL.

Your browser should open automatically, otherwise copy and paste the url provided by the Terminal. It starts by `http://localhost:8888`

Inside the Jupyter Notebook, you can see all the files inside the working directory. To create a new Notebook, you simply click on **new** and **Python 3**

Note: The new notebook is automatically saved inside the working directory.



Step 3) Import Tensorflow Inside the notebook, you can import TensorFlow with the `tf` alias. Click to run. A new cell is created below.

```
import tensorflow as tf
```



Let's write your first code with TensorFlow.

```
hello = tf.constant('Hello, Guru99!')  
hello
```

A new tensor is created. Congratulation. You successfully install TensorFlow with Jupyter on your Machine.



Step 4) Delete file

You can delete the file named Untitled.ipynb inside Jupyter.



Step 5) Close Jupyter

There are two ways of closing Jupyter. The first way is directly from the notebook. The second way is by using the terminal (or Anaconda Prompt) **From Jupyter**

In the main panel of Jupyter Notebook, simply click on **Logout**

Files Running Clusters

Select items to perform actions on them.

Logout

Upload New

You are redirected to the log out page.



Successfully logged out.
Proceed to the [login page](#).

From the terminal

Select the terminal or Anaconda prompt and run twice **ctr+c**.

The first time you do **ctr+c**, you are asked to confirm you want to shut down the notebook. Repeat **ctr+c** to confirm

```
^C[I 12:38:40.238 NotebookApp] interrupted
Serving notebooks from local directory: /Users/Thomas
1 active kernel
The Jupyter Notebook is running at:
http://localhost:8888/?token=79edffd9c156eac054cf668c6576cb074716c2182a391076
Shutdown this notebook server (y/[n])? No answer for 5s: resuming operation...
```

Use twice
Ctrl+C
to close Jupyter

```
[I 12:38:57.737 NotebookApp] Shutting down 1 kernel
[I 12:38:58.043 NotebookApp] Kernel shutdown: 2a34c1da-9b2e-40ea-aa20-2971ac33de
1c
(hello-tf) Thomass-MacBook-Pro:~ Thomas$
```

You have successfully logged out.

Jupyter with the main conda environment

If you want to launch TensorFlow with jupyter for future use, you need to open a new session with source activate hello-tf

If you don't, Jupyter will not find TensorFlow

```
In [1]: import tensorflow as tf
-----
ModuleNotFoundError                         Traceback (most recent call last)
<ipython-input-1-41389fad42b5> in <module>()
----> 1 import tensorflow as tf

ModuleNotFoundError: No module named 'tensorflow'
```

Chapter 6: Jupyter Notebook Tutorial

What is Jupyter Notebook?

A Jupyter notebook is a web application that allows the user to write codes and rich text elements. Inside the Notebooks, you can write paragraph, equations, title, add links, figures and so on. A notebook is useful to share interactive algorithms with your audience by focusing on teaching or demonstrating a technique. Jupyter Notebook is also a convenient way to run data analysis.

Jupyter Notebook App

The Jupyter Notebook App is the interface where you can write your scripts and codes through your web browser. The app can be used locally, meaning you don't need internet access, or a remote server.



The screenshot shows the Jupyter Notebook web interface. At the top, there's a navigation bar with a logo, the word "jupyter", and a "Logout" button. Below the navigation bar, there are three tabs: "Files" (which is selected), "Running", and "Clusters". A message "Select items to perform actions on them." is displayed above a file list. The file list includes a folder structure starting from "/". The files listed are:

File/Folder	Last Modified
anaconda3	19 hours ago
Applications	2 months ago
Desktop	9 minutes ago
Docker	2 months ago
Documents	8 days ago
Downloads	an hour ago
Dropbox	a day ago
Jupyter_notebook	18 minutes ago
Movies	4 months ago
Music	4 months ago
Pictures	2 months ago
Public	8 months ago

Each computation is done via a kernel. A new kernel is created each time you launch a Jupyter Notebook.

How to use Jupyter

In the session below, you will learn how to use Jupyter Notebook. You will write a simple line of code to get familiar with the environment of Jupyter.

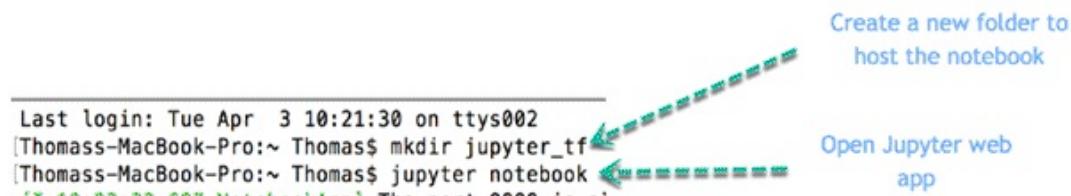
Step 1) You add a folder inside the working directory that will contains all the notebooks you will create during the tutorials about TensorFlow.

Open the Terminal and write

```
mkdir jupyter_tf  
jupyter notebook
```

Code Explanation

- mkdir jupyter_tf: Create a folder names jupyter_tf
- jupyter notebook: Open Jupyter web-app



Step 2) You can see the new folder inside the environment. Click on the folder jupyter_tf.

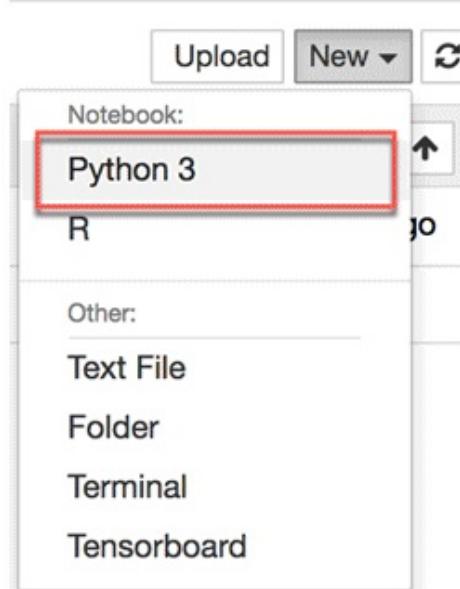
Files Running Clusters

Select items to perform actions on them.

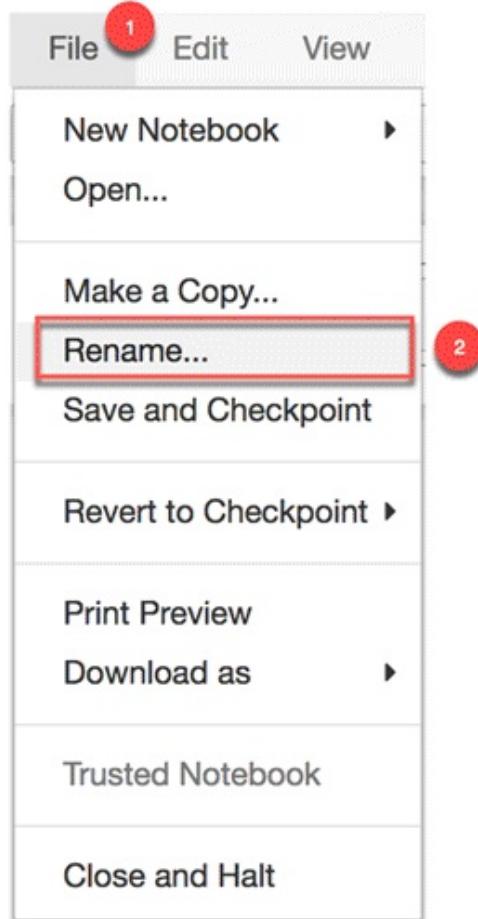
Upload New

	Name	Last Modified
<input type="checkbox"/>	anaconda3	20 hours ago
<input type="checkbox"/>	Applications	2 months ago
<input type="checkbox"/>	Desktop	2 minutes ago
<input type="checkbox"/>	Docker	2 months ago
<input type="checkbox"/>	Documents	8 days ago
<input type="checkbox"/>	Downloads	2 hours ago
<input type="checkbox"/>	Dropbox	a day ago
<input type="checkbox"/>	Jupyter_notebook	4 minutes ago
<input type="checkbox"/>	jupyter_tf	seconds ago
<input type="checkbox"/>	Movies	4 months ago
<input type="checkbox"/>	Music	4 months ago
<input type="checkbox"/>	Pictures	2 months ago
<input type="checkbox"/>	Public	8 months ago

Step 3) Inside this folder, you will create your first notebook. Click on the button **New** and **Python 3**.



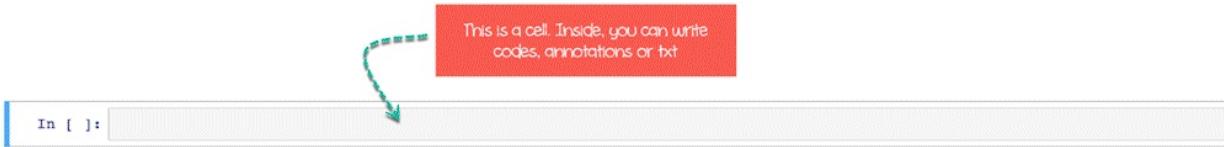
Step 4) You are inside the Jupyter environment. So far, your notebook is called Untitled.ipynb. This is the default name given by Jupyter. Let's rename it by clicking on **File** and **Rename**



You can rename it Introduction_jupyter



In Jupyter Notebook, you write codes, annotation or text inside the cells.



Inside a cell, you can write a single line of code.

```
In [1]: # Single line
import matplotlib.pyplot as plt
```

or multiple lines. Jupyter reads the code one line after another.

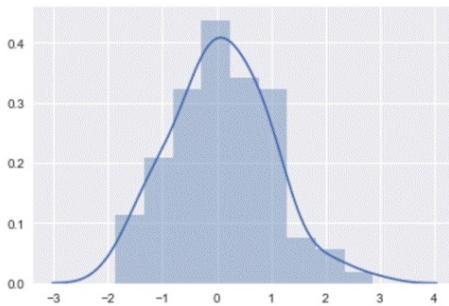
```
In [2]: # Multiple line
import numpy as np
import pandas as pd
from scipy import stats, integrate
```

For instance, if you write following code inside a cell.

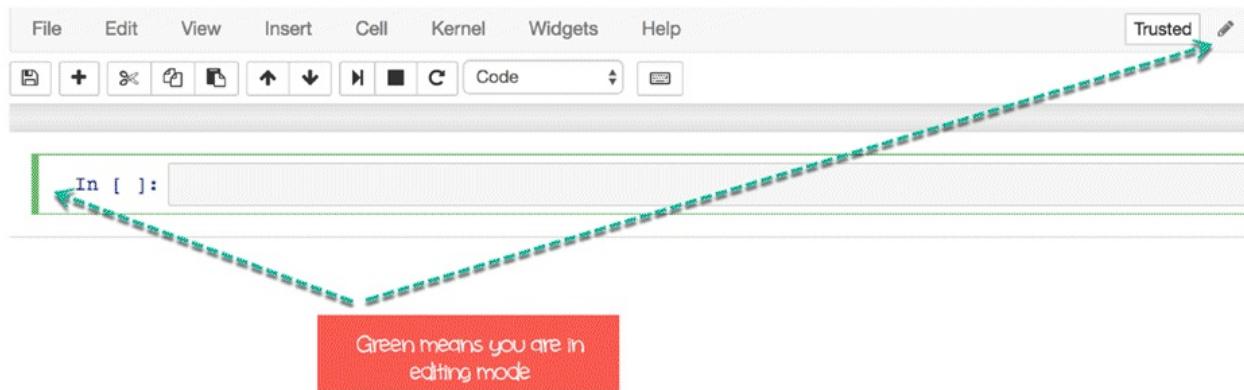
```
In [6]: # Run the code
%matplotlib inline
import seaborn as sns
sns.set(color_codes=True)
np.random.seed(sum(map(ord, "distributions")))
x = np.random.normal(size=100)
sns.distplot(x)
plt
```

It will produce this output.

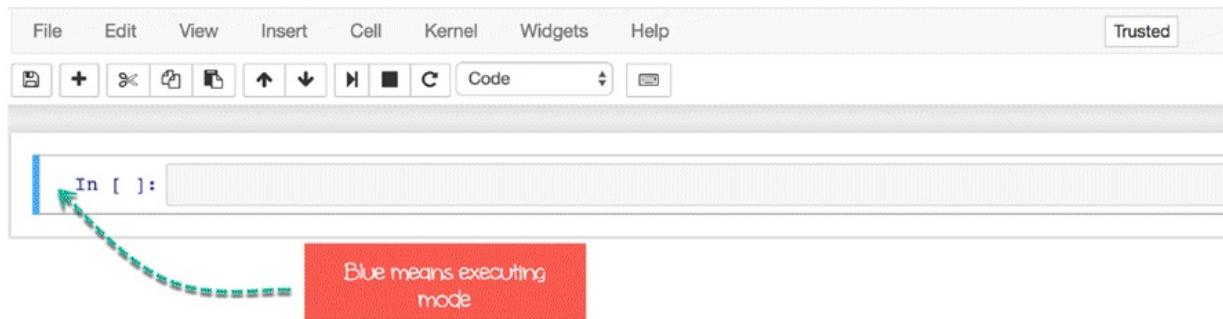
```
Out[6]: <module 'matplotlib.pyplot' from '/Users/Thomas/anaconda3/lib/python3.6/site-packages/matplotlib/pyplot.py'>
```



Step 5) You are ready to write your first line of code. You can notice the cell have two colors. The green color mean you are in the **editing mode**.



The blue color, however, indicates you are in **executing mode**.



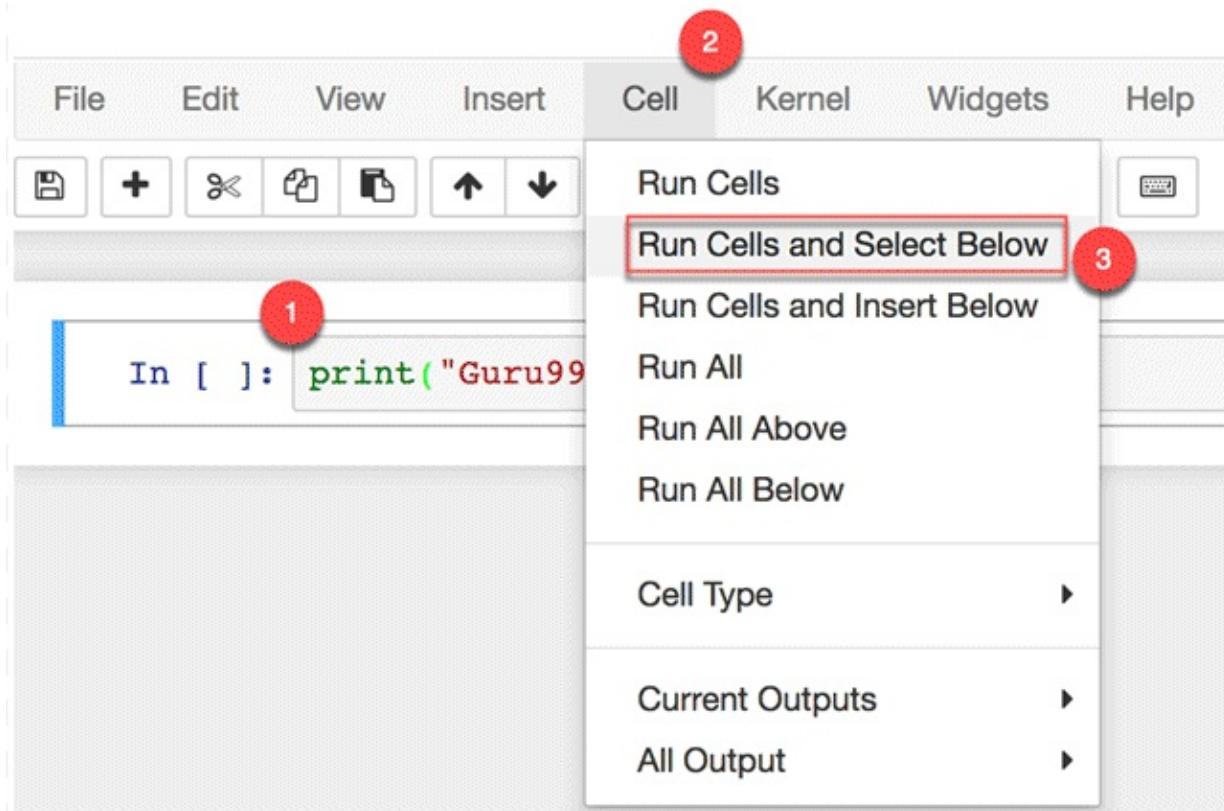
Your first line of code will be to print Guru99!. Inside the cell, you can write `print("Guru99!")`

There are two ways to run a code in Jupyter:

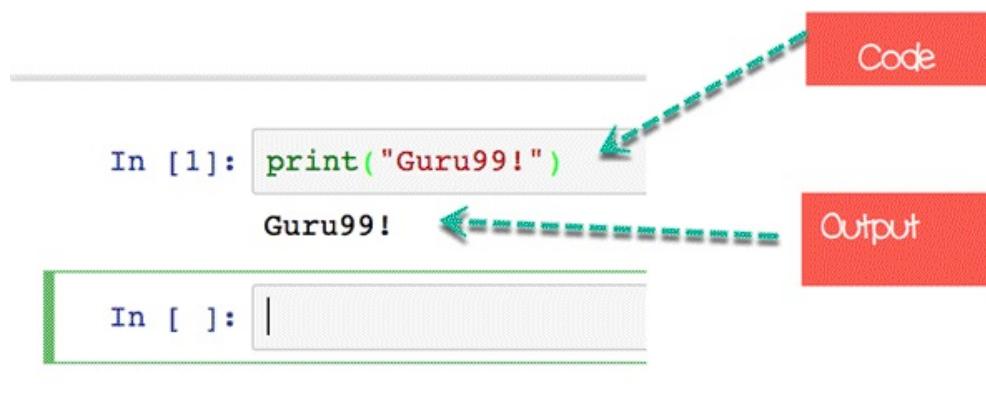
- **Click and Run**

- **Keyboard Shortcuts**

To run the code, you can click on **Cell** and then **Run Cells and Select Below**

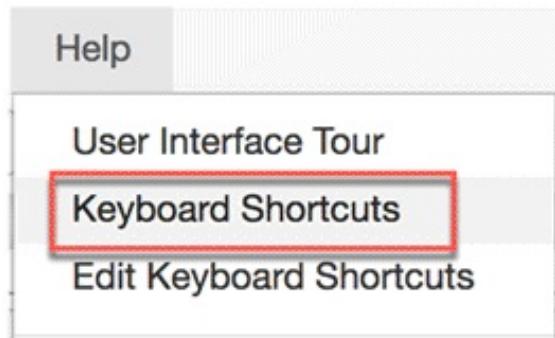


You can see the code is printed below the cell and a new cell has appeared right after the output.



A faster way to run a code is to use the **Keyboard Shortcuts**. To access the

Keyboard Shortcuts, go to **Help** and **Keyboard Shortcuts**



Below the list of shortcuts for a MacOS keyboard. You can edit the shortcuts in the editor.

Keyboard shortcuts

Command Mode (press `Esc` to enable)

<code>F</code> : find and replace	<code>↑↓</code> : extend selected cells below
<code>↶</code> : enter edit mode	<code>↑J</code> : extend selected cells below
<code>⌘↑F</code> : open the command palette	<code>A</code> : insert cell above
<code>⌘↑P</code> : open the command palette	<code>B</code> : insert cell below
<code>P</code> : open the command palette	<code>X</code> : cut selected cells
<code>↑↶</code> : run cell, select below	<code>C</code> : copy selected cells
<code>⌘↶</code> : run selected cells	<code>↑V</code> : paste cells above
<code>↖↶</code> : run cell, insert below	<code>V</code> : paste cells below

Edit Shortcuts

Following are shortcuts for Windows

Command Mode (press `Esc` to enable)

[Edit Shortcuts](#)

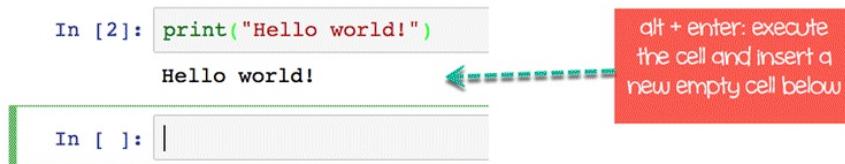
`F`: find and replace
`Ctrl-Shift-F`: open the command palette
`Ctrl-Shift-P`: open the command palette
`Enter`: enter edit mode
`P`: open the command palette
`Shift-Enter`: run cell, select below
`Ctrl-Enter`: run selected cells
`Alt-Enter`: run cell and insert below
`Y`: change cell to code
...
...

`Shift-Down`: extend selected cells below
`Shift-J`: extend selected cells below
`A`: insert cell above
`B`: insert cell below
`X`: cut selected cells
`C`: copy selected cells
`Shift-V`: paste cells above
`V`: paste cells below
`Z`: undo cell deletion
...
...

Write this line

```
print("Hello world!")
```

and try to use the Keyboard Shortcuts to run the code. Use alt+enter. it will execute the cell and insert a new empty cell below, like you did before.



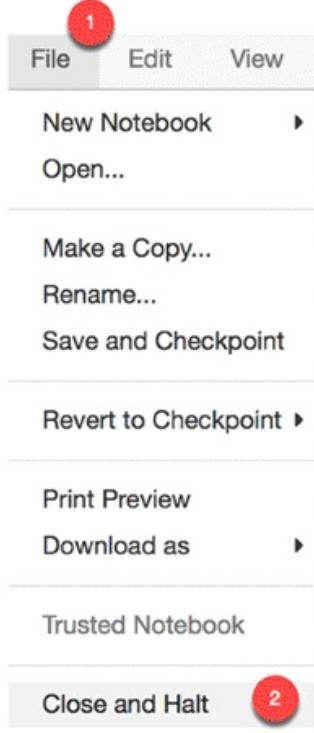
In [2]: `print("Hello world!")`

Hello world!

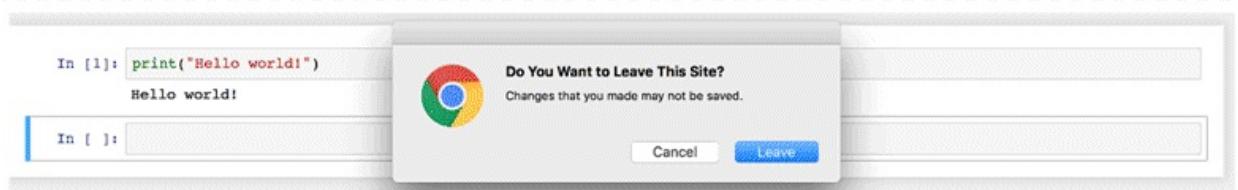
In []: |

A red callout box points from the text "alt + enter: execute the cell and insert a new empty cell below" to the cell separator line between the output and the empty cell input area.

Step 6) It is time to close the Notebook. Go to **File** and click on **Close and Halt**



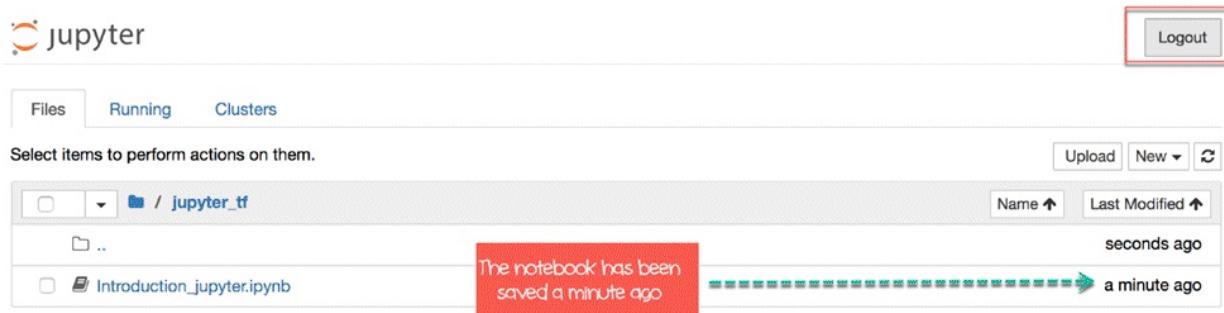
Note: Jupyter automatically saves the notebook with checkpoint. If you have the following message:



It means Jupyter didn't save the file since the last checkpoint. You can manually save the notebook



You will be redirected to the main panel. You can see your notebook has been saved a minute ago. You can safely logout.



Summary

- Jupyter notebook is a web application where you can run your Python and R codes. It is easy to share and deliver rich data analysis with Jupyter.
- To launch jupyter, write in the terminal: jupyter notebook
- You can save your notebook wherever you want
- A cell contains your Python code. The kernel will read the code one by one.
- You can use the shortcut to run a cell. By default: Ctrl+Enter

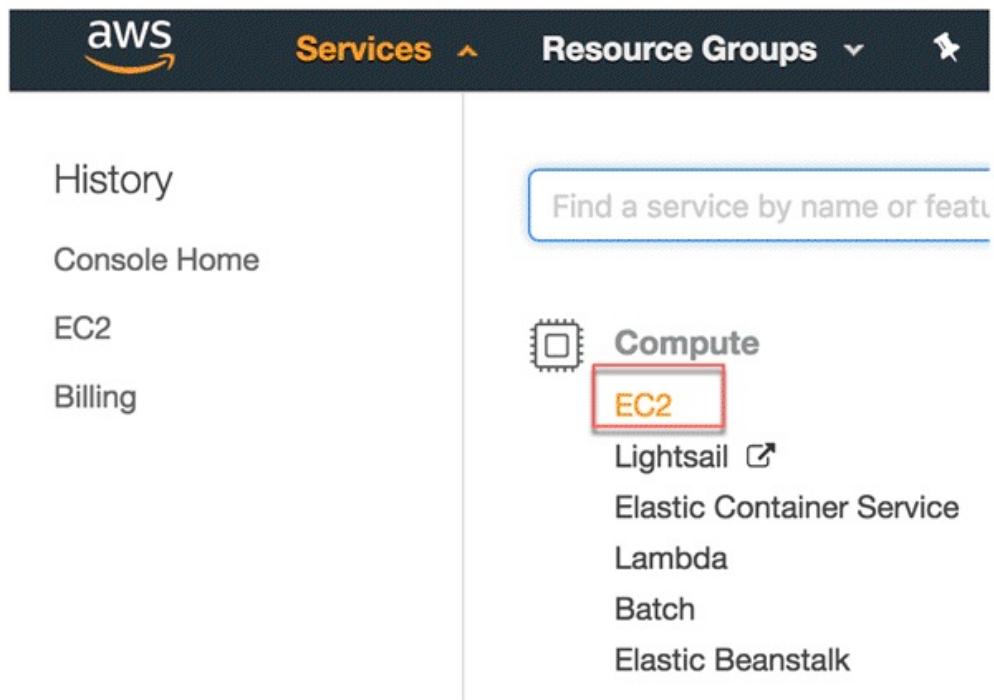
Chapter 7: Tensorflow on AWS

This is a step by step tutorial, to used Jupyter Notebook on AWS

If you do not have an account at AWS, create a free account [here](#).

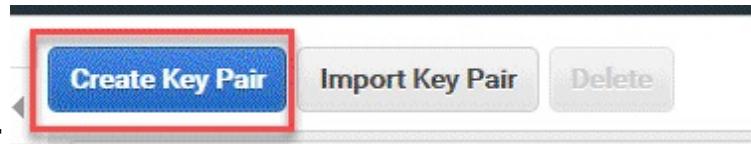
PART 1: Set up a key pair

Step 1) Go to Services and find EC2



Step 2) In the panel and click on **Key Pairs**





Step 3) Click Create Key Pair

1. You can call it Docker key
2. Click Create



A file name Docker_key.pem downloads.

Key pair name: Docker_key

Fingerprint: 71:af:12:d0:1e:79:5e:3a:2c:2e:1f:08:2f:71:dc:05:86:10:5e:85

Key Pair: Docker_key

Feedback English (US)

Docker_key.pem

Key is created and a file Docker_key.pem is downloaded

Step 4) Copy and paste it into the folder key. We will need it soon.

For Mac OS user only

This step concerns only Mac OS user. For Windows or Linux users, please proceed to PART 2

You need to set a working directory that will contain the file key First of all, create a folder named key. For us, it is located inside the main folder Docker.

Then, you set this path as your working directory
mkdir Docker/key
cd Docker/key

The screenshot shows a terminal window with the following text:

```
[Thomass-MacBook-Pro:~ Thomas$ mkdir Docker/key
[Thomass-MacBook-Pro:~ Thomas$ cd Docker/key
Thomass-MacBook-Pro:key Thomas$ ]
```

Two green arrows point from the right side of the image to the terminal text. One arrow points to the word "key" in the first command, with the text "Create new folder" above it. Another arrow points to the word "key" in the second command, with the text "Make users/Docker/key the new working directory" below it.

PART 2: Set up a security group

Step 1) You need to configure a security group. You can access it with the panel

NETWORK & SECURITY

Security Groups

Elastic IPs

Placement Groups

Key Pairs

Network Interfaces

Step 2) Click on Create Security Group



Step 3) In the next Screen

1. Enter Security group name "jupyter_docker" and Description Security Group for Docker
2. You need to add 4 rules on top of
 - ssh: port range 22, source Anywhere
 - http: port range 80, source Anywhere
 - https: port range 443, source Anywhere
 - Custom TCP: port range 8888, source Anywhere
3. Click Create

Create Security Group

1. **Security group name:** jupyter_docker
Description: Security Group for Docker
VPC: vpc-620d0207 (default)

2. **Inbound Rules:**

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	Custom	CIDR, IP or Security Group e.g. SSH for Admin Desktop
HTTP	TCP	80	Custom	0.0.0.0/0, ::/0 e.g. SSH for Admin Desktop
HTTPS	TCP	443	Custom	0.0.0.0/0, ::/0 e.g. SSH for Admin Desktop
Custom TCP F	TCP	8888	Custom	CIDR, IP or Security Group e.g. SSH for Admin Desktop

Add Rule

3. **Create**

Step 4) The newly created Security Group will be listed

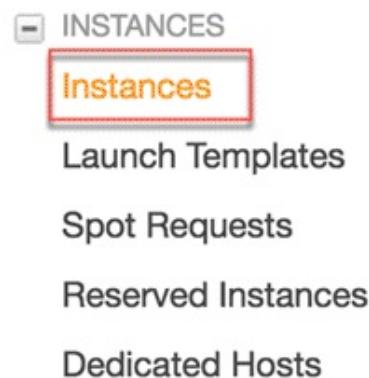
Create Security Group Actions ▾

Filter by tags and attributes or search by keyword

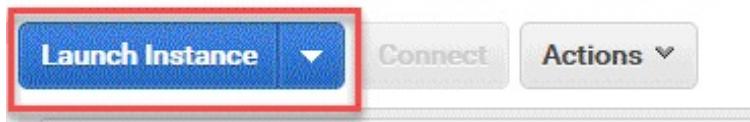
Name	Group ID	Group Name	VPC ID	Description
[checkbox]	[redacted]	[redacted]	[redacted]	[redacted]
[checkbox]	[redacted]	[redacted]	[redacted]	[redacted]
[checkbox]	[redacted]	[redacted]	[redacted]	[redacted]
[checkbox]	sg-c3424a89	jupyter_docker	vpc-620d0207	Security Group for Docker

Part 3: Launch instance

You are finally ready to create the instance



Step 1) Click on Launch Instance



The default server is enough for your need. You can choose Amazon Linux AMI. The current instance is 2018.03.0.

AMI stands for Amazon Machine Image. It contains the information required to successfully starts an instance that run on a virtual server stored in the cloud.



Note that AWS has a server dedicated to deep learning such as:

- **Deep Learning AMI (Ubuntu)**
- **Deep Learning AMI**
- **Deep Learning Base AMI (Ubuntu)**

All of them Comes with latest binaries of deep learning frameworks pre-installed in separate virtual environments:

- TensorFlow,
- Caffe
- PyTorch,
- Keras,
- Theano
- CNTK.

Fully-configured with NVidia CUDA, cuDNN and NCCL as well as Intel MKL-DNN

Step 2) Choose **t2.micro**. It is a free tier server. AWS offers for free this virtual machine equipped with 1 vCPU and 1 GB of memory. This server provides a good tradeoff between computation, memory and network performance. It fits for small and medium database

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. Learn more about instance types and how they can meet your computing needs.

Filter by: All instance types Current generation Show/Hide Columns

Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
General purpose	t2.micro	1	1	EBS only	-	Low to Moderate	Yes
General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes
General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
General purpose	t2.large	2	8	EBS only	-	Low to Moderate	Yes
General purpose	t2.xlarge	4	16	EBS only	-	Moderate	Yes
General purpose	t2.2xlarge	8	32	EBS only	-	Moderate	Yes

Cancel Previous Review and Launch Next: Configure Instance Details

Step 3) Keep settings default in next screen and click Next: Add Storage

Step 3: Configure Instance Details

Network: vpc-620d0207 (default) Create new VPC

Subnet: No preference (default subnet in any Availability Zone) Create new subnet

Auto-assign Public IP: Use subnet setting (Enable)

Placement group: Add instance to placement group.

IAM role: None Create new IAM role

Shutdown behavior: Stop

Enable termination protection: Protect against accidental termination

Monitoring: Enable CloudWatch detailed monitoring Additional charges apply.

Tenancy: Shared - Run a shared hardware instance Additional charges will apply for dedicated tenancy.

T2 Unlimited: Enable Additional charges may apply

Cancel Previous Review and Launch Next: Add Storage

Step 4) Increase storage to 10GB and click Next

Step 4: Add Storage

Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. [Learn more about storage options in Amazon EC2.](#)

The screenshot shows the 'Add Storage' step in the AWS EC2 wizard. A new volume is being created with a size of 10 GiB. The 'Volume Type' is set to 'General Purpose SSD (GP2)'. The 'Delete on Termination' checkbox is checked. The 'Encrypted' checkbox is unchecked. The 'Next: Add Tags' button is highlighted with a red box and a number 2.

Step 5) Keep settings default and click Next: Configure Security Group

Step 5: Add Tags

A tag consists of a case-sensitive key-value pair. For example, you could define a tag with key = Name and value = Webserver. A copy of a tag can be applied to volumes, instances or both. Tags will be applied to all instances and volumes. [Learn more](#) about tagging your Amazon EC2 resources.

The screenshot shows the 'Add Tags' step in the AWS EC2 wizard. A new tag is being added with the key 'Name' and value 'jupyter_docker'. The 'Next: Configure Security Group' button is highlighted with a red box and a number 3.

Step 6) Choose the security group you created before, which is **jupyter_docker**

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more](#) about Amazon EC2 security groups.

The screenshot shows the 'Configure Security Group' step in the AWS EC2 wizard. A new security group is being assigned. The 'Select an existing security group' radio button is selected. A list of existing security groups is shown, with 'sg-c3424a89' (Name: jupyter_docker) selected and highlighted with a red box. The 'Next: Review and Launch' button is highlighted with a red box and a number 3.

Step 7) Review your settings and Click the launch button

Step 7: Review Instance Launch

Please review your instance launch details. You can go back to edit changes for each section. Click **Launch** to assign a key pair to your instance and complete the launch process.

A screenshot of the AWS Step 7: Review Instance Launch page. At the top, there's an orange warning box with the text: "⚠ Improve your instances' security. Your security group, jupyter_docker, is open to the world. Your instances may be accessible from any IP address. We recommend that you update your security group rules to allow access from known IP addresses only. You can also open additional ports in your security group to facilitate access to the application or service you're running, e.g., HTTP (80) for web servers. Edit security groups". Below the alert, there's a section titled "AMI Details" showing "Amazon Linux AMI 2018.03.0 (HVM), SSD Volume Type - ami-cfe4b2b0". A note says: "The Amazon Linux AMI is an EBS-backed, AWS-supported image. The default image includes AWS command line tools, Python, Ruby, Perl, and Java. The repositories include Docker, PHP, MySQL, PostgreSQL, and other packages." Under "Security Group ID", there's a table with columns "Name" and "Description". The "Launch" button at the bottom right is highlighted with a red box.

Step 8) The last step is to link the key pair to the instance.

A screenshot of the "Select an existing key pair or create a new key pair" dialog. It has a title bar "Select an existing key pair or create a new key pair" with a close button. The main content area contains the following text: "A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance." Below this, a note says: "Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#)." There are three input fields: 1. A dropdown menu labeled "Choose an existing key pair". 2. A dropdown menu labeled "Select a key pair" with "Docker_key" selected. 3. A checkbox labeled "I acknowledge that I have access to the selected private key file (Docker_key.pem), and that without this file, I won't be able to log into my instance." with the number "2" next to it. At the bottom are "Cancel" and "Launch Instances" buttons, with the "Launch Instances" button highlighted with a red box and the number "3" next to it.

Step 8) Instance will launch

AWS CloudWatch Metrics dashboard showing metrics for Lambda functions. The top section displays a bar chart for 'Lambda Executed' with values for 'All Regions' and 'All Functions'. Below it is a line chart for 'Lambda Duration' over time. The bottom section displays a table of metrics including 'Lambda Errors', 'Lambda Invocations', and 'Lambda Duration'.

Launch Status

Your instances are now launching
The following instance launches have been initiated: i-090447e9c051efdce [View launch log](#)

Get notified of estimated charges
Create billing alerts to get an email notification when estimated charges on your AWS bill exceed an amount you define (for example, \$100).

How to connect to your instances

Step 9) Below a summary of the instances currently in use. Note the public IP

Instances

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP
	i-090447e9c051efdce	t2.micro	us-east-1b	running	initializing	none	ec2-52-23-241-75.compute-1.amazonaws.com	52.23.241.75

Description

Instance: i-090447e9c051efdce Public DNS: ec2-52-23-241-75.compute-1.amazonaws.com

Instance ID: i-090447e9c051efdce
Instance state: running
Instance type: t2.micro
Elastic IPs:

Public DNS (IPv4): ec2-52-23-241-75.compute-1.amazonaws.com
IPv4 Public IP: 52.23.241.75
IPv6 IPs:
Private DNS: ip-172-31-57-55.ec2.internal

Step 9) Click on Connect

Launch Instance

Connect

Actions

Filter by tags and attributes or search by keyword

Name	Instance ID	Instance Type	Availability Zone
	i-090447e9c051efdce	t2.micro	us-east-1b

You will find the connection details

Connect To Your Instance

X

- I would like to connect with A standalone SSH client
 A Java SSH Client directly from my browser (Java required)

To access your instance:

1. Open an SSH client. (find out how to [connect using PuTTY](#))
2. Locate your private key file (Docker_key.pem). The wizard automatically detects the key you used to launch the instance.
3. Your key must not be publicly viewable for SSH to work. Use this command if needed:

```
chmod 400 Docker_key.pem
```

4. Connect to your instance using its Public DNS:

ec2-18-188-151-171.us-east-2.compute.amazonaws.com

Example:

```
ssh -i "Docker_key.pem" ec2-user@ec2-18-188-151-171.us-east-2.compute.amazonaws.com
```

Please note that in most cases the username above will be correct, however please ensure that you read your AMI usage instructions to ensure that the AMI owner has not changed the default AMI username.

If you need any assistance connecting to your instance, please see our [connection documentation](#).

Launch your instance (Mac OS users)

At first make sure that inside the terminal, your working directory points to the folder with the key pair file docker run the code

```
chmod 400 docker.pem
```

Open the connection with this code.

There are two codes. in some case, the first code avoids Jupyter to open the notebook.

In this case, use the second one to force the connection.

```
# If able to launch Jupyter
ssh -i "docker.pem" ec2-user@ec2-18-219-192-34.us-east-2.compute.amazonaws.com
```

```
# If not able to launch Jupyter
ssh -i "docker.pem" ec2-user@ec2-18-219-192-34.us-east-
2.compute.amazonaws.com -L 8888:127.0.0.1:8888
```

The first time, you are prompted to accept the connection

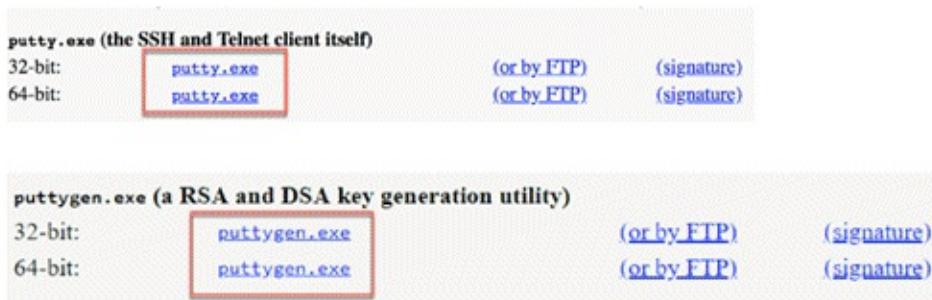
```
thomass-MacBook-Pro:key Thomas$ ssh -i "Docker_key.pem" ec2-user@ec2-18-188-151-
171.us-east-2.compute.amazonaws.com
-L 8888:127.0.0.1:8888
The authenticity of host 'ec2-18-188-151-171.us-east-2.compute.amazonaws.com (18
.188.151.171)' can't be established.
ECDSA key fingerprint is SHA256:UuNljpxxnup20pilz0T1LL60Z1o3TdyE86kB6Pmujf0.
Are you sure you want to continue connecting (yes/no)?
```

Launch your instance (Windows users)

Step 1) Go to this website to download PuTTY and PuTTYgen PuTTY

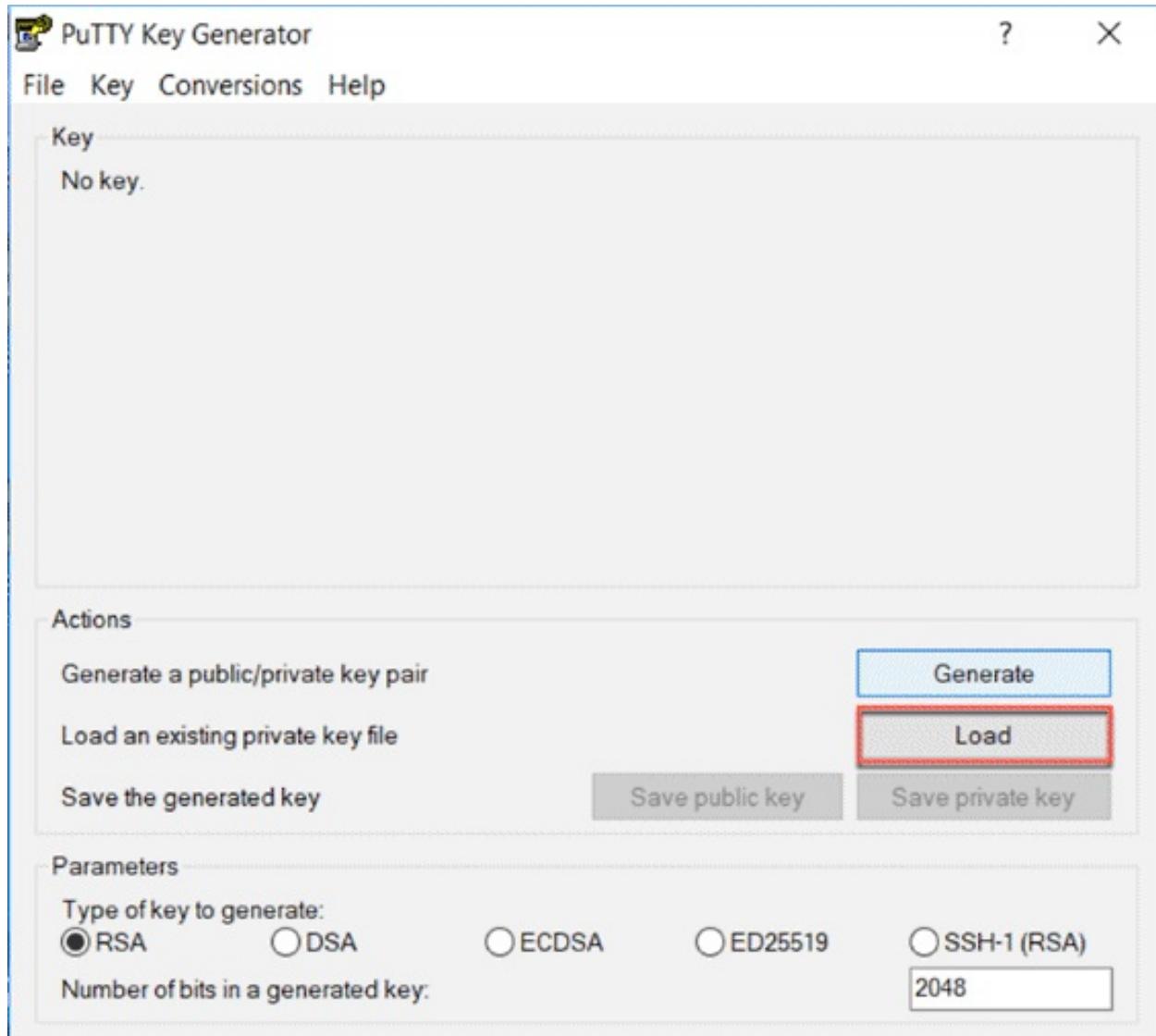
You need to download

- PuTTY: launch the instance
- PuTTYgen: convert the pem file to ppk

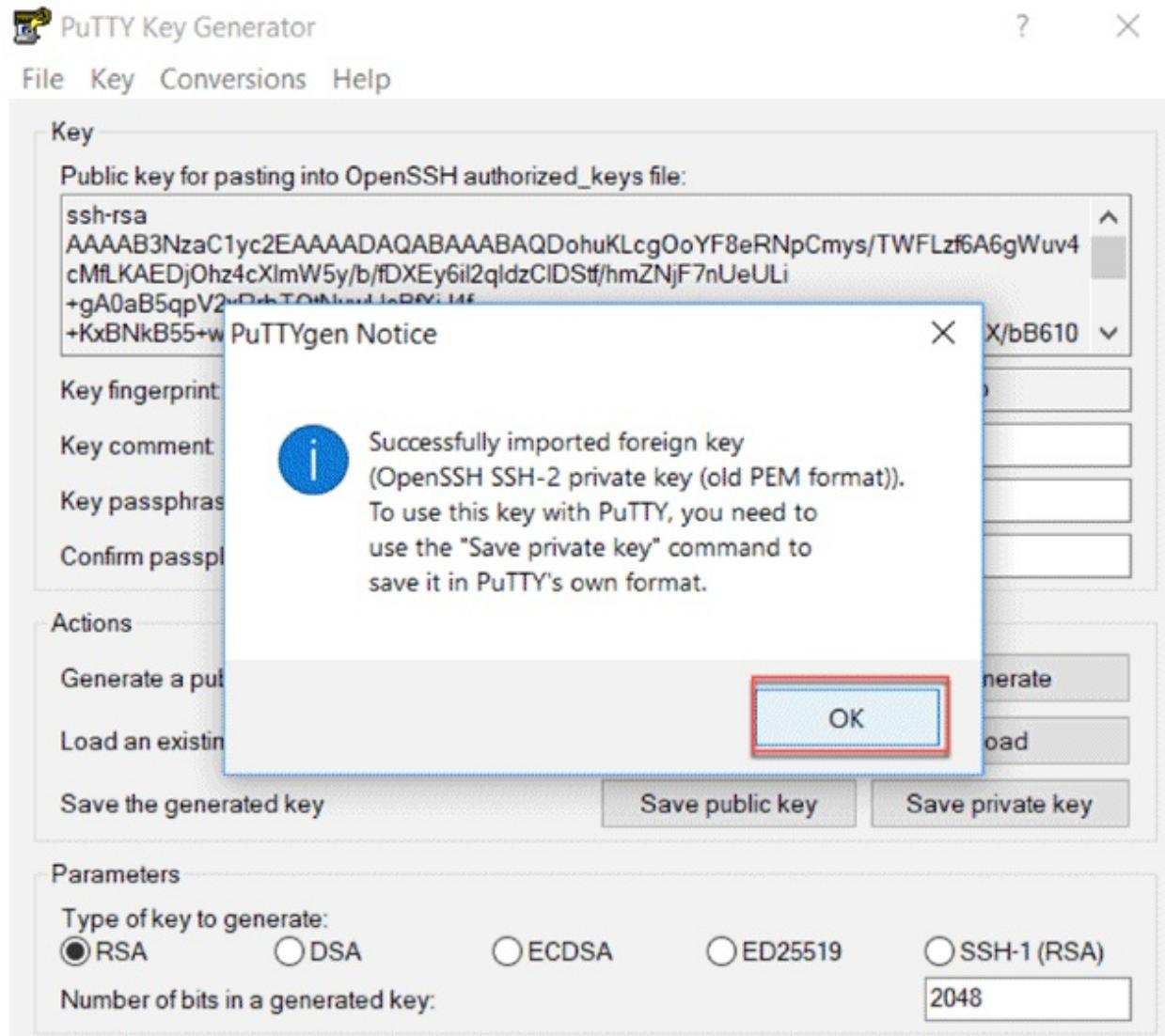


Now that both software are installed, you need to convert the .pem file to .ppk. PuTTY can only read .ppk. The pem file contains the unique key created by AWS.

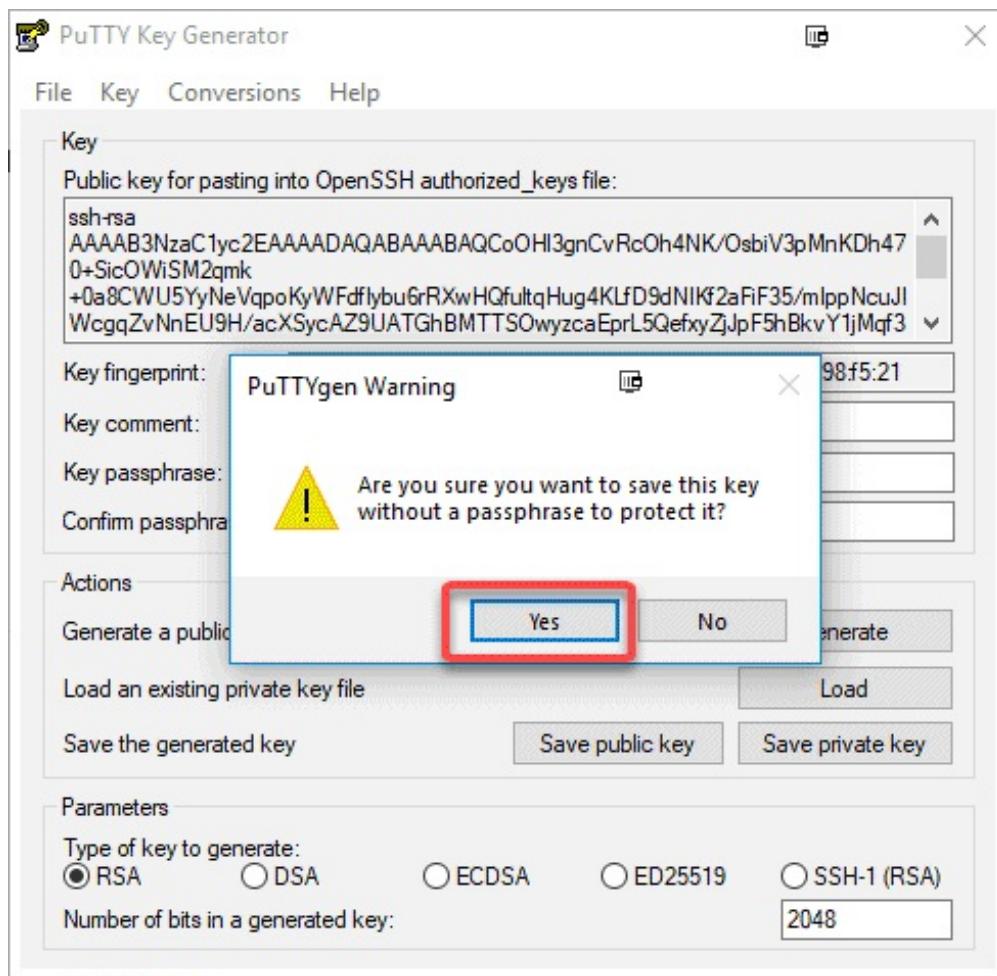
Step 2) Open PuTTYgen and click on Load. Browse the folder where the .pem file is located.



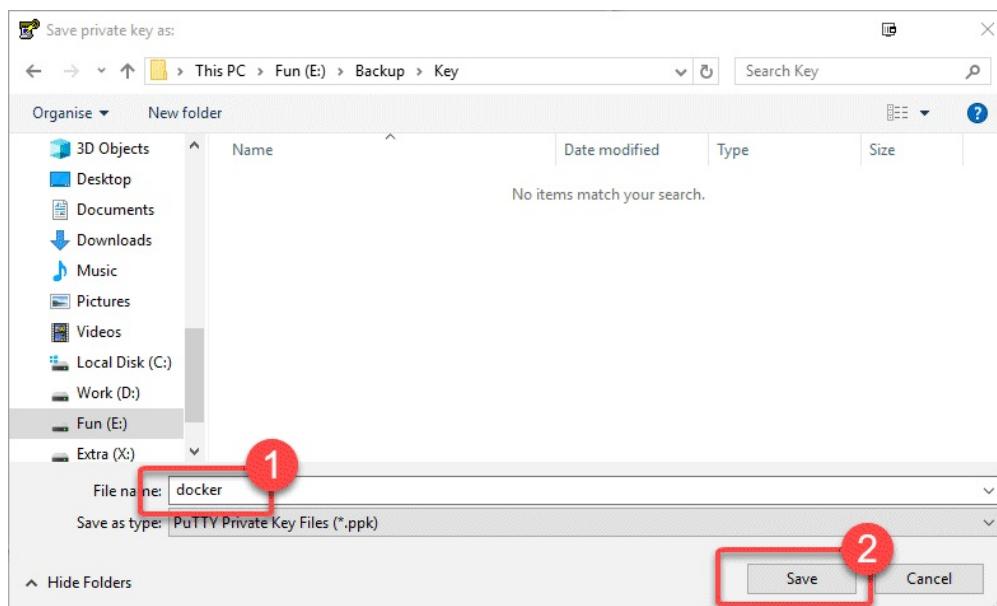
Step 3) After you loaded the file, you should get a notice informing you that the key has been successfully imported. Click on OK



Step 4) Then click on Save private key. You are asked if you want to save this key without a passphrase. Click on yes.



Step 5) Save the Key

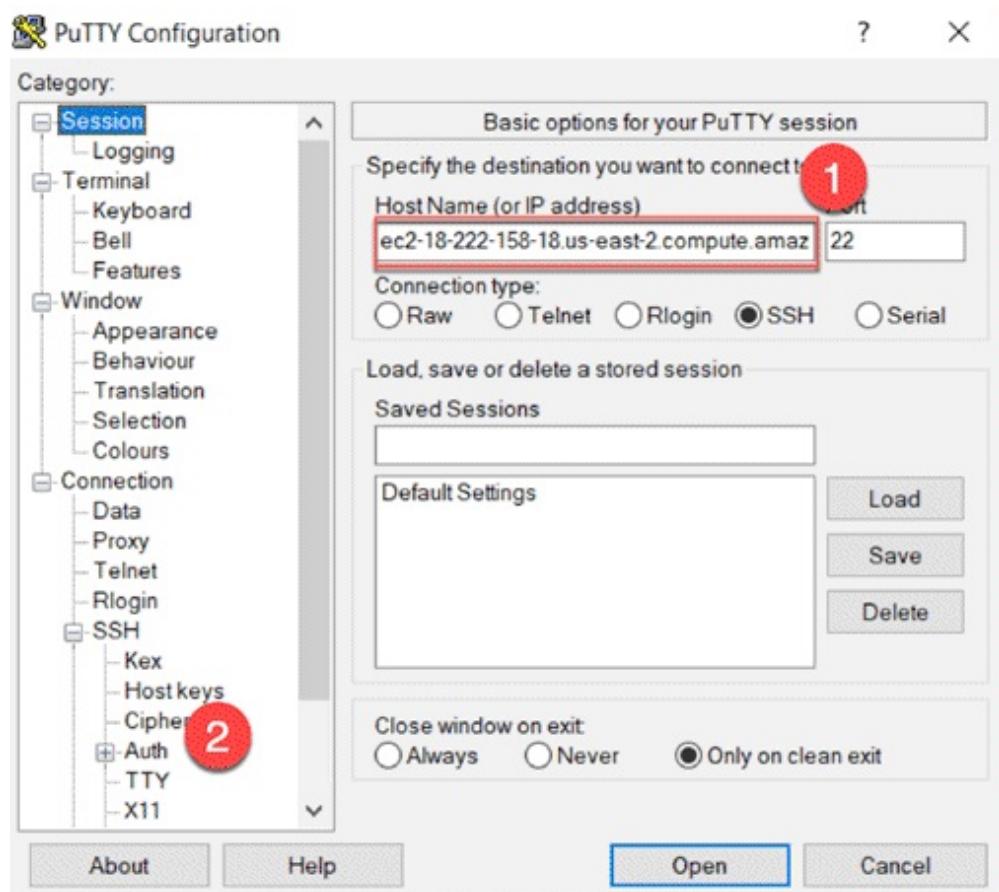


Step 6) Go to AWS and copy the public DNS

4. Connect to your instance using its Public DNS:

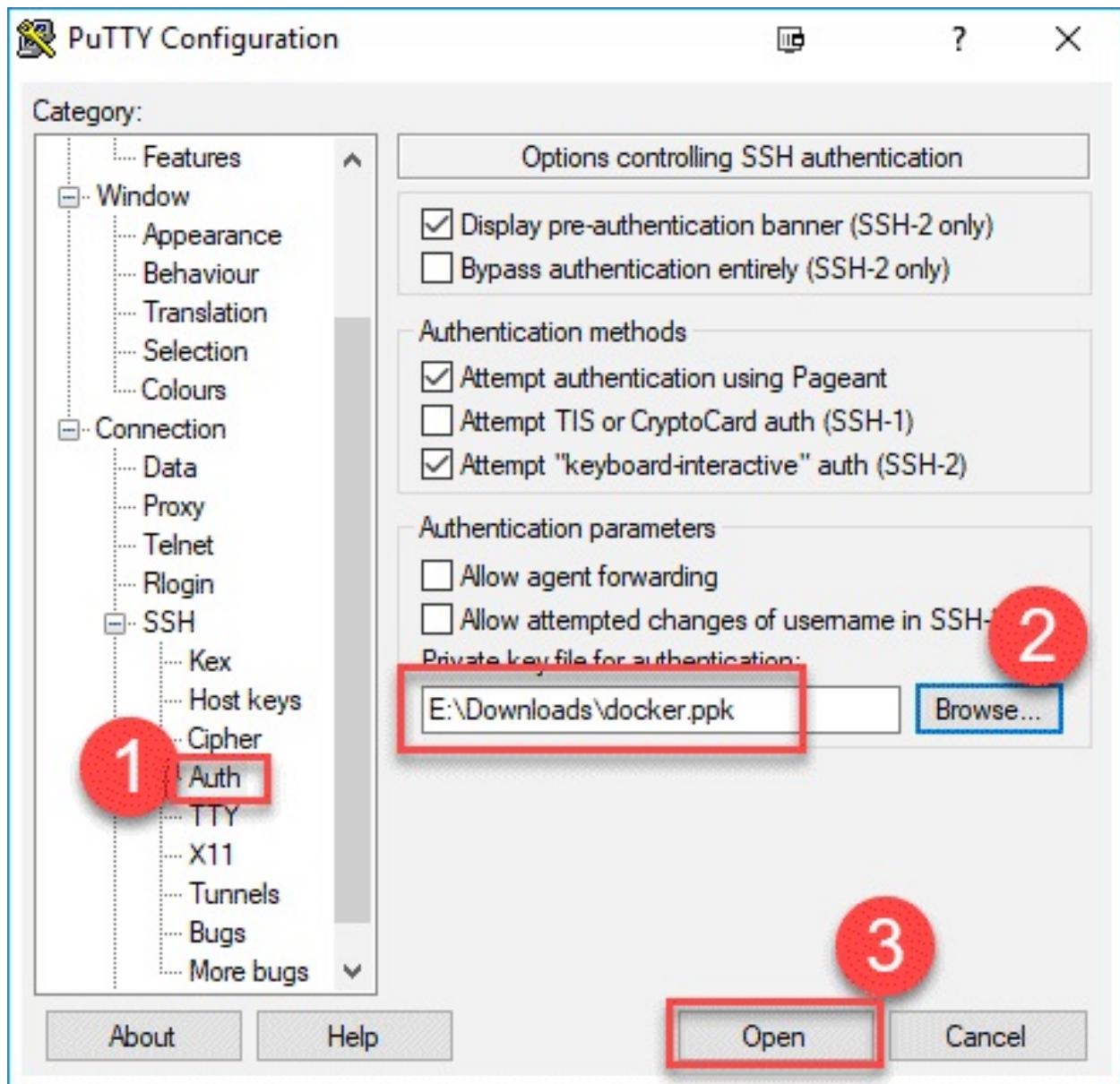
ec2-13-59-162-131.us-east-2.compute.amazonaws.com

Open PuTTY and paste the Public DNS in the Host Name



Step 7)

1. On the left panel, unfold SSH and open Auth
2. Browse the Private Key. You should select the .ppk
3. Click on Open.



Step 8

When this step is done, a new window will be opened. Click Yes if you see this



Step 9)

You need to login as: ec2-user

A screenshot of a Putty terminal window. The title bar says "ec2-18-222-158-18.us-east-2.compute.amazonaws.com - PuTTY". The main pane shows a login prompt: "login as: ec2-user". The cursor is at the end of the "ec2-user" entry.

Step 10)

You are connected to the Amazon Linux AMI.

A screenshot of a Putty terminal window showing a successful SSH connection to an Amazon Linux AMI instance. The title bar says "ec2-user@ip-172-31-57-55:~". The main pane displays the Amazon Linux AMI welcome message, including the logo, and a link to the release notes: "https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/". It also indicates 2 packages needed for security updates.

Part 4: Install Docker

While you are connected with the server via Putty/Terminal, you can install **Docker** container.

Execute the following codes

```
sudo yum update -y  
sudo yum install -y docker  
sudo service docker start  
sudo user-mod -a -G docker ec2-user  
exit
```

Launch again the connection

```
ssh -i "docker.pem" ec2-user@ec2-18-219-192-34.us-east-  
2.compute.amazonaws.com -L 8888:127.0.0.1:8888
```

Windows users use SSH as mentioned above

Part 5: Install Jupyter

Step 1) Create Jupyter with a pre-built image ## Tensorflow
docker run -v
~/work:/home/jovyan/work -d -p 8888:8888 jupyter/tensorflow-notebook ##
Sparkdocker run -v ~/work:/home/jovyan/work -d -p 8888:8888 jupyter/pyspark-
notebook

Code Explanation

- docker run: Run the image
- v: attach a volume
- ~/work:/home/jovyan/work: Volume
- 8888:8888: port
- jupyter/datascience-notebook: Image

For other pre-build images, go here

Allow preserving Jupyter notebook

```
sudo chown 1000 ~/work
```

Step 2) Install tree to see our working directory next sudo yum install -y tree

```
[ec2-user@ip-172-31-16-239 ~]$ tree
.
`-- work
1 directory, 0 files
```

Step 3)

1. Check the container and its name (changes with every installation) Use command docker ps
2. Get the name and use the log to open Jupyter. In the tutorial, the container's

name is `vigilant_easley`. Use command `docker logs vigilant_easley`

3. Get the URL

```
ec2-user@ip-172-31-57-55:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
90a3c09282d6        jupyter/tensorflow-notebook   "tini -g start-notebook.sh"   3 minutes ago    Up 3 minutes     0.0.0.0:8888->8888/tcp   vigilant_easley
/usr/local/bin/start-notebook.sh: ignoring /usr/local/bin/start-notebook.d/*
Container must be run with group "root" to update passwd file
Executing the command: jupyter notebook
[I 09:06:47.206 NotebookApp] Writing notebook server cookie secret to /home/jovyan/.local/share/jupyter/runtime/notebook_cookie_secret
[W 09:06:47.671 NotebookApp] WARNING: The notebook server is listening on all IP addresses and not using encryption. This is not recommended.
[I 09:06:47.724 NotebookApp] JupyterLab extension loaded from /opt/conda/lib/python3.6/site-packages/jupyterlab
[I 09:06:47.725 NotebookApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
[I 09:06:47.735 NotebookApp] Serving notebooks from local directory: /home/jovyan
[I 09:06:47.735 NotebookApp] The Jupyter Notebook is running at:
[I 09:06:47.735 NotebookApp] http://(90a3c09282d6 or 127.0.0.1):8888/?token=f460f1e79ab74c382b19f90fe3fd55f9f99c5222365eceed
[I 09:06:47.735 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 09:06:47.736 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
http://(90a3c09282d6 or 127.0.0.1):8888/?token=f460f1e79ab74c382b19f90fe3fd55f9f99c5222365eceed
[ec2-user@ip-172-31-57-55:~]$
```

1 NAMES
2 http://(90a3c09282d6 or 127.0.0.1):8888/?token=f460f1e79ab74c382b19f90fe3fd55f9f99c5222365eceed
3 http://(90a3c09282d6 or 127.0.0.1):8888/?token=f460f1e79ab74c382b19f90fe3fd55f9f99c5222365eceed

Step 4)

In the URL

`http://(90a3c09282d6 or 127.0.0.1):8888/?`

`token=f460f1e79ab74c382b19f90fe3fd55f9f99c5222365eceed` Replace
`(90a3c09282d6 or 127.0.0.1)` with Public DNS of your instance

Connect To Your Instance

I would like to connect with A standalone SSH client i A Java SSH Client directly from my browser (Java required) i

To access your instance:

1. Open an SSH client. (find out how to [connect using PuTTY](#))
2. Locate your private key file (`Docker_key.pem`). The wizard automatically detects the key you used to launch the instance.
3. Your key must not be publicly viewable for SSH to work. Use this command if needed:

```
chmod 400 Docker_key.pem
```
4. Connect to your instance using its Public DNS:

```
ec2-174-129-135-16.compute-1.amazonaws.com
```

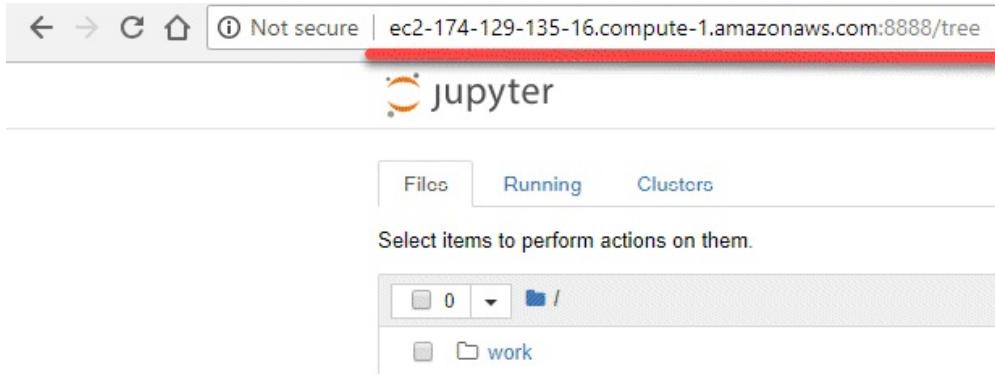
Example:

Step 5)

The new URL becomes

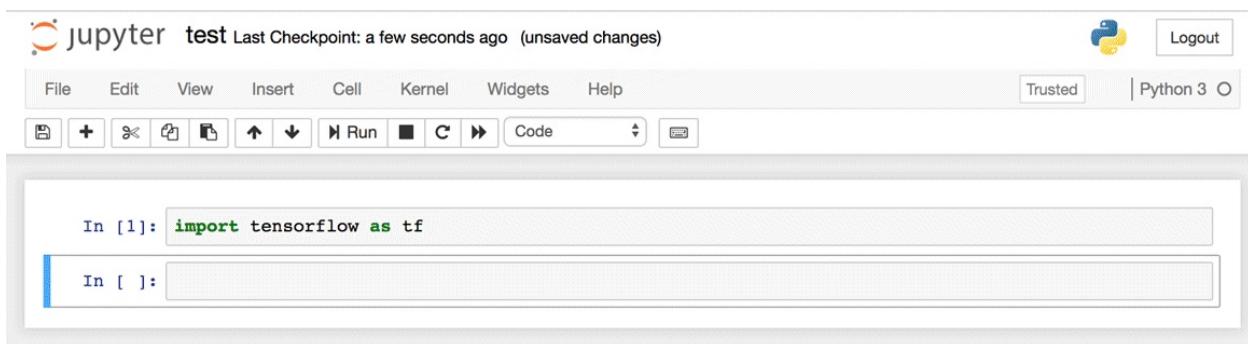
http://ec2-174-129-135-16.compute-1.amazonaws.com:8888/?

token=f460f1e79ab74c382b19f90fe3fd55f9f99c5222365eceed **Step 6)** Copy and paste the URL into your browser. Jupyter Opens



Step 7)

You can write a new Notebook in the work folder

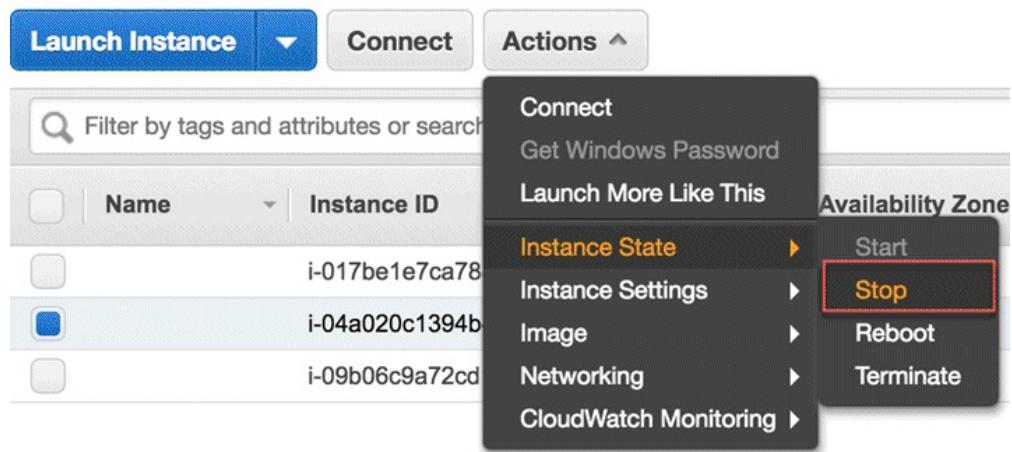


Part 6: Close connection

Close the connection in the terminal

```
exit
```

Go back to AWS and stop the server.



Troubleshooting

If ever docker doesnot work, try to rebuilt image using

```
docker run -v ~/work:/home/jovyan/work -d -p 8888:8888
jupyter/tensorflow-notebook
```

Chapter 8: TensorFlow Basics: Tensor, Shape, Type, Graph, Sessions & Operators

What is a Tensor?

Tensorflow's name is directly derived from its core framework: **Tensor**. In Tensorflow, all the computations involve tensors. A tensor is a **vector** or **matrix** of n-dimensions that represents all types of data. All values in a tensor hold identical data type with a known (or partially known) **shape**. The shape of the data is the dimensionality of the matrix or array.

A tensor can be originated from the input data or the result of a computation. In TensorFlow, all the operations are conducted inside a **graph**. The graph is a set of computation that takes place successively. Each operation is called an **op node** and are connected to each other.

The graph outlines the ops and connections between the nodes. However, it does not display the values. The edge of the nodes is the tensor, i.e., a way to populate the operation with data.

In Machine Learning, models are feed with a list of objects called **feature vectors**. A feature vector can be of any data type. The feature vector will usually be the primary input to populate a tensor. These values will flow into an op node through the tensor and the result of this operation/computation will create a new tensor which in turn will be used in a new operation. All these operations can be viewed in the graph.

Representation of a Tensor

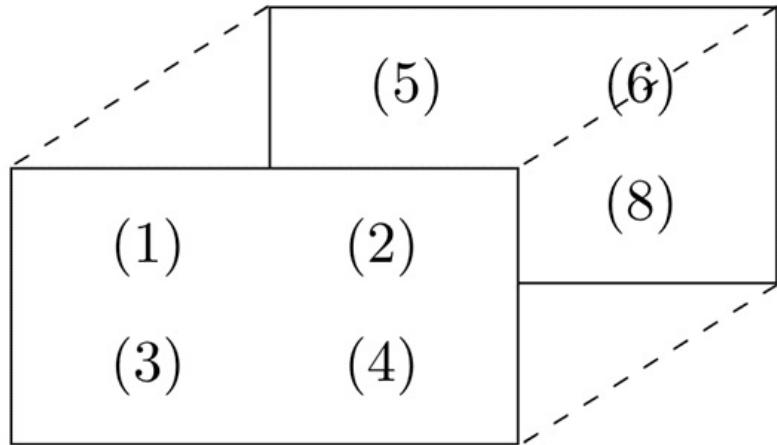
In TensorFlow, a tensor is a collection of feature vectors (i.e., array) of n-dimensions. For instance, if we have a 2x3 matrix with values from 1 to 6, we write:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

TensorFlow represents this matrix as:

```
[[1, 2, 3],  
 [4, 5, 6]]
```

If we create a three-dimensional matrix with values from 1 to 8, we have:



TensorFlow represents this matrix as:

```
[ [[1, 2],  
   [[3, 4],  
    [[5, 6],  
     [[7, 8] ]
```

Note: A tensor can be represented with a scalar or can have a shape of more than three dimensions. It is just more complicated to visualize higher dimension level.

Types of Tensor

In TensorFlow, all the computations pass through one or more tensors. A tensor is an object with three properties:

- A unique label (name)
- A dimension (shape)
- A data type (dtype)

Each operation you will do with TensorFlow involves the manipulation of a tensor. There are four main tensors you can create:

- tf.Variable
- tf.constant
- tf.placeholder
- tf.SparseTensor

In this tutorial, you will learn how to create a tf.constant and a tf.Variable.

Before we go through the tutorial, make sure you activate the conda environment with TensorFlow. We named this environment hello-tf.

For MacOS user:

```
source activate hello-tf
```

For Windows user:

```
activate hello-tf
```

After you have done that, you are ready to import tensorflow

```
# Import tf
import tensorflow as tf
```

Create a tensor of n-dimension

You begin with the creation of a tensor with one dimension, namely a scalar.

To create a tensor, you can use `tf.constant()`

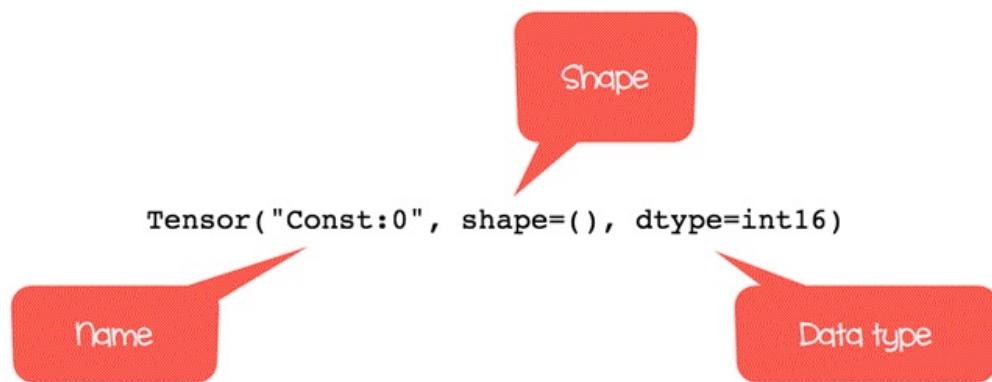
```
tf.constant(value, dtype, name = "")  
arguments  
  
- `value`: Value of n dimension to define the tensor. Optional  
- `dtype`: Define the type of data:  
    - `tf.string`: String variable  
    - `tf.float32`: Flot variable  
    - `tf.int16`: Integer variable  
- "name": Name of the tensor. Optional. By default, `Const_1:0`
```

To create a tensor of dimension 0, run the following code

```
## rank 0  
# Default name  
r1 = tf.constant(1, tf.int16)  
print(r1)
```

Output

```
Tensor("Const:0", shape=(), dtype=int16)
```



```
# Named my_scalar  
r2 = tf.constant(1, tf.int16, name = "my_scalar")
```

```
print(r2)
```

Output

```
Tensor("my_scalar:0", shape=(), dtype=int16)
```

Each tensor is displayed by the tensor name. Each tensor object is defined with a unique label (name), a dimension (shape) and a data type (dtype).

You can define a tensor with decimal values or with a string by changing the type of data.

```
# Decimal
r1_decimal = tf.constant(1.12345, tf.float32)
print(r1_decimal)
# String
r1_string = tf.constant("Guru99", tf.string)
print(r1_string)
```

Output

```
Tensor("Const_1:0", shape=(), dtype=float32)
Tensor("Const_2:0", shape=(), dtype=string)
```

A tensor of dimension 1 can be created as follow:

```
## Rank 1
r1_vector = tf.constant([1,3,5], tf.int16)
print(r1_vector)
r2_boolean = tf.constant([True, True, False], tf.bool)
print(r2_boolean)
```

Output

```
Tensor("Const_3:0", shape=(3,), dtype=int16)
Tensor("Const_4:0", shape=(3,), dtype=bool)
```

You can notice the shape is only composed of 1 column.

To create an array of 2 dimensions, you need to close the brackets after each row. Check the examples below

```
## Rank 2
r2_matrix = tf.constant([ [1, 2],
                         [3, 4] ],tf.int16)
print(r2_matrix)
```

Output

```
Tensor("Const_5:0", shape=(2, 2), dtype=int16)
```

The matrix has 2 rows and 2 columns filled with values 1, 2, 3, 4.

A matrix with 3 dimensions is constructed by adding another level with the brackets.

```
## Rank 3
r3_matrix = tf.constant([ [[1, 2],
                           [3, 4],
                           [5, 6]] ], tf.int16)
print(r3_matrix)
```

Output

```
Tensor("Const_6:0", shape=(1, 3, 2), dtype=int16)
```

The matrix looks like the picture two.

Shape of tensor

When you print the tensor, TensorFlow guesses the shape. However, you can get the shape of the tensor with the shape property.

Below, you construct a matrix filled with a number from 10 to 15 and you check the shape of m_shape

```
# Shape of tensor
m_shape = tf.constant([
    [10, 11],
    [12, 13],
    [14, 15]
])
m_shape.shape
```

Output

```
TensorShape([Dimension(3), Dimension(2)])
```

The matrix has 3 rows and 2 columns.

TensorFlow has useful commands to create a vector or a matrix filled with 0 or 1. For instance, if you want to create a 1-D tensor with a specific shape of 10, filled with 0, you can run the code below:

```
# Create a vector of 0
print(tf.zeros(10))
```

Output

```
Tensor("zeros:0", shape=(10,), dtype=float32)
```

The property works for matrix as well. Here, you create a 10x10 matrix filled with 1

```
# Create a vector of 1
print(tf.ones([10, 10]))
```

Output

```
Tensor("ones:0", shape=(10, 10), dtype=float32)
```

You can use the shape of a given matrix to make a vector of ones. The matrix m_shape is a 3x2 dimensions. You can create a tensor with 3 rows filled by one's with the following code:

```
# Create a vector of ones with the same number of rows as m_shape
print(tf.ones(m_shape.shape[0]))
```

Output

```
Tensor("ones_1:0", shape=(3, ), dtype=float32)
```

If you pass the value 1 into the bracket, you can construct a vector of ones equals to the number of columns in the matrix m_shape.

```
# Create a vector of ones with the same number of column as m_shape
print(tf.ones(m_shape.shape[1]))
```

Output

```
Tensor("ones_2:0", shape=(2, ), dtype=float32)
```

Finally, you can create a matrix 3x2 with only one's

```
print(tf.ones(m_shape.shape))
```

Output

```
Tensor("ones_3:0", shape=(3, 2), dtype=float32)
```

Type of data

The second property of a tensor is the type of data. A tensor can only have one type of data at a time. A tensor can only have one type of data. You can return the type with the property `dtype`.

```
print(m_shape.dtype)
```

Output

```
<dtype: 'int32'>
```

In some occasions, you want to change the type of data. In TensorFlow, it is possible with `tf.cast` method.

Example

Below, a float tensor is converted to integer using you use the method `cast`.

```
# Change type of data
type_float = tf.constant(3.123456789, tf.float32)
type_int = tf.cast(type_float, dtype=tf.int32)
print(type_float.dtype)
print(type_int.dtype)
```

Output

```
<dtype: 'float32'>
<dtype: 'int32'>
```

TensorFlow chooses the type of data automatically when the argument is not specified during the creation of the tensor. TensorFlow will guess what is the most likely types of data. For instance, if you pass a text, it will guess it is a string and convert it to string.

Creating operator

Some Useful TensorFlow operators

You know how to create a tensor with TensorFlow. It is time to learn how to perform mathematical operations.

TensorFlow contains all the basic operations. You can begin with a simple one. You will use TensorFlow method to compute the square of a number. This operation is straightforward because only one argument is required to construct the tensor.

The square of a number is constructed with `tf.sqrt(x)` with `x` as a floating number.

```
x = tf.constant([2.0], dtype = tf.float32)
print(tf.sqrt(x))
```

Output

```
Tensor("Sqrt:0", shape=(1,), dtype=float32)
```

Note: The output returned a tensor object and not the result of the square of 2. In the example, you print the definition of the tensor and not the actual evaluation of the operation. In the next section, you will learn how TensorFlow works to execute the operations.

Following is a list of commonly used operations. The idea is the same. Each operation requires one or more arguments.

- `tf.add(a, b)`
- `tf.subtract(a, b)`
- `tf.multiply(a, b)`
- `tf.div(a, b)`

- `tf.pow(a, b)`
- `tf.exp(a)`
- `tf.sqrt(a)`

Example

```
# Add
tensor_a = tf.constant([[1,2]], dtype = tf.int32)
tensor_b = tf.constant([[3, 4]], dtype = tf.int32)

tensor_add = tf.add(tensor_a, tensor_b)print(tensor_add)
```

Output

```
Tensor("Add:0", shape=(1, 2), dtype=int32)
```

Code Explanation

Create two tensors:

- one tensor with 1 and 2
- one tensor with 3 and 4

You add up both tensors.

Notice: that both tensors need to have the same shape. You can execute a multiplication over the two tensors.

```
# Multiply
tensor_multiply = tf.multiply(tensor_a, tensor_b)
print(tensor_multiply)
```

Output

```
Tensor("Mul:0", shape=(1, 2), dtype=int32)
```

Variables

So far, you have only created constant tensors. It is not of great use. Data always arrive with different values, to capture this, you can use the Variable class. It will represent a node where the values always change.

To create a variable, you can use `tf.get_variable()` method

```
tf.get_variable(name = "", values, dtype, initializer)  
argument  
- `name = ""`: Name of the variable  
- `values`: Dimension of the tensor  
- `dtype`: Type of data. Optional  
- `initializer`: How to initialize the tensor. Optional  
If initializer is specified, there is no need to include the  
`values` as the shape of `initializer` is used.
```

For instance, the code below creates a two-dimensional variable with two random values. By default, TensorFlow returns a random value. You name the variable var

```
# Create a Variable  
## Create 2 Randomized values  
var = tf.get_variable("var", [1, 2])  
print(var.shape)
```

Output

```
(1, 2)
```

In the second example, you create a variable with one row and two columns. You need to use [1,2] to create the dimension of the variable

The initials values of this tensor are zero. For instance, when you train a model, you need to have initial values to compute the weight of the features. Below, you set these initial value to zero.

```
var_init_1 = tf.get_variable("var_init_1", [1, 2], dtype=tf.int32,  
initializer=tf.zeros_initializer)  
print(var_init_1.shape)
```

Output

```
(1, 2)
```

You can pass the values of a constant tensor in a variable. You create a constant tensor with the method `tf.constant()`. You use this tensor to initialize the variable.

The first values of the variable are 10, 20, 30 and 40. The new tensor will have a shape of 2x2.

```
# Create a 2x2 matrix  
tensor_const = tf.constant([[10, 20],  
[30, 40]])  
# Initialize the first value of the tensor equals to tensor_const  
var_init_2 = tf.get_variable("var_init_2", dtype=tf.int32,  
initializer=tensor_const)  
print(var_init_2.shape)
```

Output

```
(2, 2)
```

Placeholder

A placeholder has the purpose of feeding the tensor. Placeholder is used to initialize the data to flow inside the tensors. To supply a placeholder, you need to use the method `feed_dict`. The placeholder will be fed only within a session.

In the next example, you will see how to create a placeholder with the method `tf.placeholder`. In the next session, you will learn to feed a placeholder with actual value.

The syntax is:

```
tf.placeholder(dtype, shape=None, name=None )  
arguments:  
- `dtype`: Type of data  
- `shape`: dimension of the placeholder. Optional. By default,  
shape of the data  
- `name`: Name of the placeholder. Optional  
data_placeholder_a = tf.placeholder(tf.float32, name =  
"data_placeholder_a")  
print(data_placeholder_a)
```

Output

```
Tensor("data_placeholder_a:0", dtype=float32)
```

Session

TensorFlow works around 3 main components:

- Graph
- Tensor
- Session

Components	Description
Graph	The graph is fundamental in TensorFlow. All of the mathematical operations (ops) are performed inside a graph. You can imagine a graph as a project where every operations are done. The nodes represent these ops, they can absorb or create new tensors.
Tensor	A tensor represents the data that progress between operations. You saw previously how to initialize a tensor. The difference between a constant and variable is the initial values of a variable will change over time.
Session	A session will execute the operation from the graph. To feed the graph with the values of a tensor, you need to open a session. Inside a session, you must run an operator to create an output.

Graphs and sessions are independent. You can run a session and get the values to use later for further computations.

In the example below, you will:

- Create two tensors
- Create an operation
- Open a session

- Print the result

Step 1) You create two tensors x and y

```
## Create, run and evaluate a session
x = tf.constant([2])
y = tf.constant([4])
```

Step 2) You create the operator by multiplying x and y

```
## Create operator
multiply = tf.multiply(x, y)
```

Step 3) You open a session. All the computations will happen within the session. When you are done, you need to close the session.

```
## Create a session to run the code
sess = tf.Session()
result_1 = sess.run(multiply)
print(result_1)
sess.close()
```

Output

[8]

Code explanation

- `tf.Session()`: Open a session. All the operations will flow within the sessions
- `run(multiply)`: execute the operation created in step 2.
- `print(result_1)`: Finally, you can print the result
- `close()`: Close the session

The result shows 8, which is the multiplication of x and y.

Another way to create a session is inside a block. The advantage is it automatically closes the session.

```
with tf.Session() as sess:
```

```
result_2 = multiply.eval()  
print(result_2)
```

Output

```
[8]
```

In a context of the session, you can use the eval() method to execute the operation. It is equivalent to run(). It makes the code more readable.

You can create a session and see the values inside the tensors you created so far.

```
## Check the tensors created before  
sess = tf.Session()  
print(sess.run(r1))  
print(sess.run(r2_matrix))  
print(sess.run(r3_matrix))
```

Output

```
1  
[[1 2]  
 [3 4]]  
[[[1 2]  
 [3 4]  
 [5 6]]]
```

Variables are empty by default, even after you create a tensor. You need to initialize the variable if you want to use the variable. The object `tf.global_variables_initializer()` needs to be called to initialize the values of a variable. This object will explicitly initialize all the variables. This is helpful before you train a model.

You can check the values of the variables you created before. Note that you need to use run to evaluate the tensor

```
sess.run(tf.global_variables_initializer())  
print(sess.run(var))  
print(sess.run(var_init_1))
```

```
print(sess.run(var_init_2))
```

Output

```
[[ -0.05356491  0.75867283]]  
[[0 0]]  
[[10 20]  
 [30 40]]
```

You can use the placeholder you created before and feed it with actual value.
You need to pass the data into the method `feed_dict`.

For example, you will take the power of 2 of the placeholder
`data_placeholder_a`.

```
import numpy as np  
power_a = tf.pow(data_placeholder_a, 2)  
with tf.Session() as sess:  
data = np.random.rand(1, 10)  
print(sess.run(power_a, feed_dict={data_placeholder_a: data})) #  
Will succeed.
```

Code Explanation

- `import numpy as np`: Import numpy library to create the data
- `tf.pow(data_placeholder_a, 2)`: Create the ops
- `np.random.rand(1, 10)`: Create a random array of data
- `feed_dict={data_placeholder_a: data}`: Feed the placeholder with data

Output

```
[[ 0.05478134  0.27213147  0.8803037   0.0398424   0.21172127  0.01444725  
 0.02584014  0.3763949   0.66022706  0.7565559  ]]
```

Graph

TensorFlow depends on a genius approach to render the operation. All the computations are represented with a dataflow scheme. The dataflow graph has been developed to see to data dependencies between individual operation. Mathematical formula or algorithm are made of a number of successive operations. A graph is a convenient way to visualize how the computations are coordinated.

The graph shows a **node** and an **edge**. The node is the representation of a operation, i.e. the unit of computation. The edge is the tensor, it can produce a new tensor or consume the input data. It depends on the dependencies between individual operation.

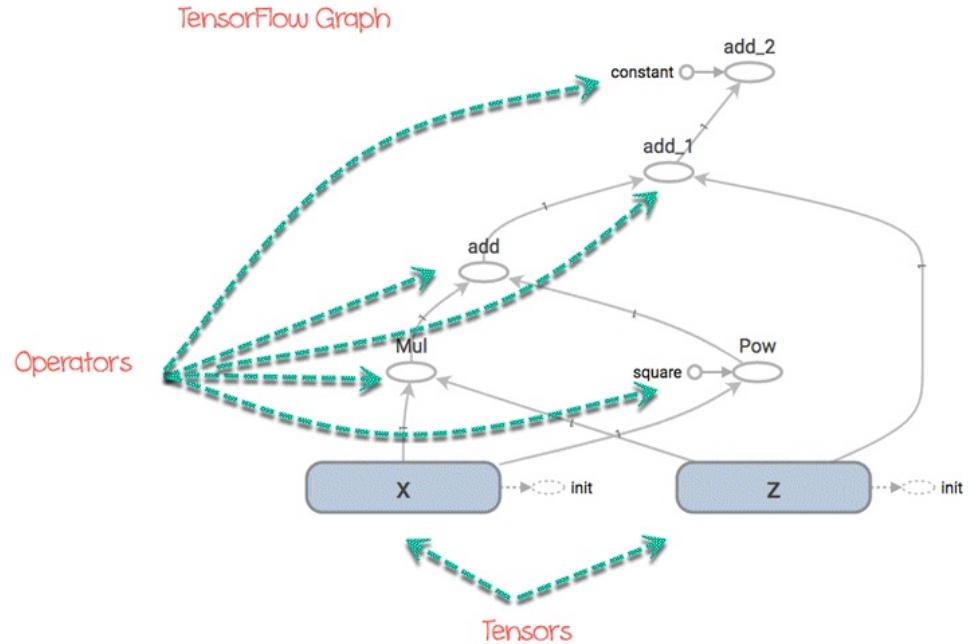
The structure of the graph connects together the operations (i.e. the nodes) and how those are operation are feed. Note that the graph does not display the output of the operations, it only helps to visualize the connection between individual operations.

Let's see an example.

Imagine you want to evaluate the following function:

$$f(x, z) = xz + x^2 + z + 5$$

TensorFlow will create a graph to execute the function. The graph looks like this:



You can easily see the path that the tensors will take to reach the final destination.

For instance, you can see the operation `add` cannot be done before `and`. The graph explains that it will:

1. compute `and`:
2. add 1) together
3. add to 2)
4. add 3) to

```
x = tf.get_variable("x", dtype=tf.int32,
initializer=tf.constant([5]))
z = tf.get_variable("z", dtype=tf.int32,
initializer=tf.constant([6]))
c = tf.constant([5], name =      "constant")square =
tf.constant([2], name =      "square")
f = tf.multiply(x, z) + tf.pow(x, square) + z + c
```

Code Explanation

- **x:** Initialize a variable called `x` with a constant value of 5

- z: Initialize a variable called z with a constant value of 6
- c: Initialize a constant tensor called c with a constant value of 5
- square: Initialize a constant tensor called square with a constant value of 2
- f: Construct the operator

In this example, we choose to keep the values of the variables fixed. We also created a constant tensor called c which is the constant parameter in the function f. It takes a fixed value of 5. In the graph, you can see this parameter in the tensor called constant.

We also constructed a constant tensor for the power in the operator tf.pow(). It is not necessary. We did it so that you can see the name of the tensor in the graph. It is the circle called square.

From the graph, you can understand what will happen of the tensors and how it can return an output of 66.

The code below evaluate the function in a session.

```
init = tf.global_variables_initializer() # prepare to initialize
all variables
with tf.Session() as sess:
    init.run() # Initialize x and y
    function_result = f.eval()
print(function_result)
```

Output

[66]

Summary

TensorFlow works around:

- Graph: Computational environment containing the operations and tensors
- Tensors: Represents the data (or value) that will flow in the graph. It is the edge in the graph
- Sessions: Allow the execution of the operations

Create a constant tensor

constant	object
D0	<code>tf.constant(1, tf.int16)</code>
D1	<code>tf.constant([1,3,5], tf.int16)</code>
D2	<code>tf.constant([[1, 2], [3, 4]], tf.int16)</code>
D3	<code>tf.constant([[[1, 2],[3, 4], [5, 6]]], tf.int16)</code>

Create an operator

Create an operator	Object
a+b	<code>tf.add(a, b)</code>
a*b	<code>tf.multiply(a, b)</code>

Create a variable tensor

Create a variable	object
randomized value	<code>tf.get_variable("var", [1, 2])</code>
initialized first value	<code>tf.get_variable("var_init_2", dtype=tf.int32, initializer=[[1, 2], [3, 4]])</code>

Open a session

Session	object
Create a session	<code>tf.Session()</code>
Run a session	<code>tf.Session.run()</code>
Evaluate a tensor	<code>variable_name.eval()</code>
Close a session	<code>sess.close()</code>
Session by block	with <code>tf.Session()</code> as sess:

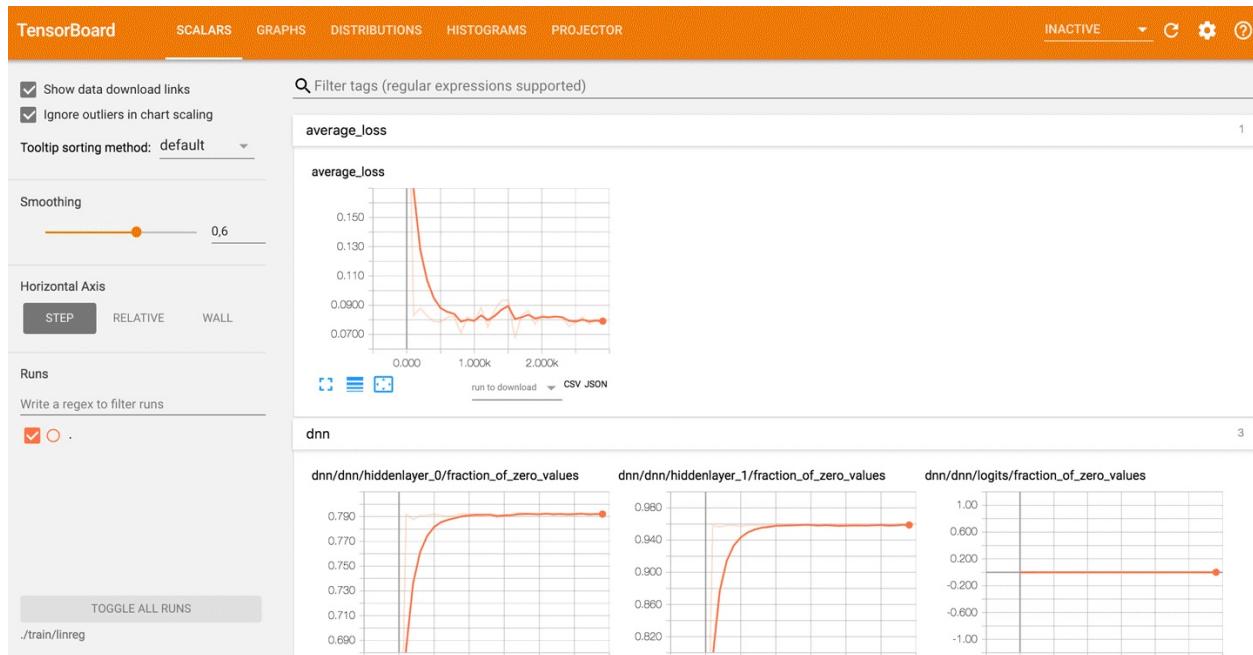
Chapter 9: Tensorboard: Graph Visualization with Example

What is TensorBoard

Tensorboard is the interface used to visualize the graph and other tools to understand, debug, and optimize the model.

Example

The image below comes from the graph you will generate in this tutorial. It is the main panel:



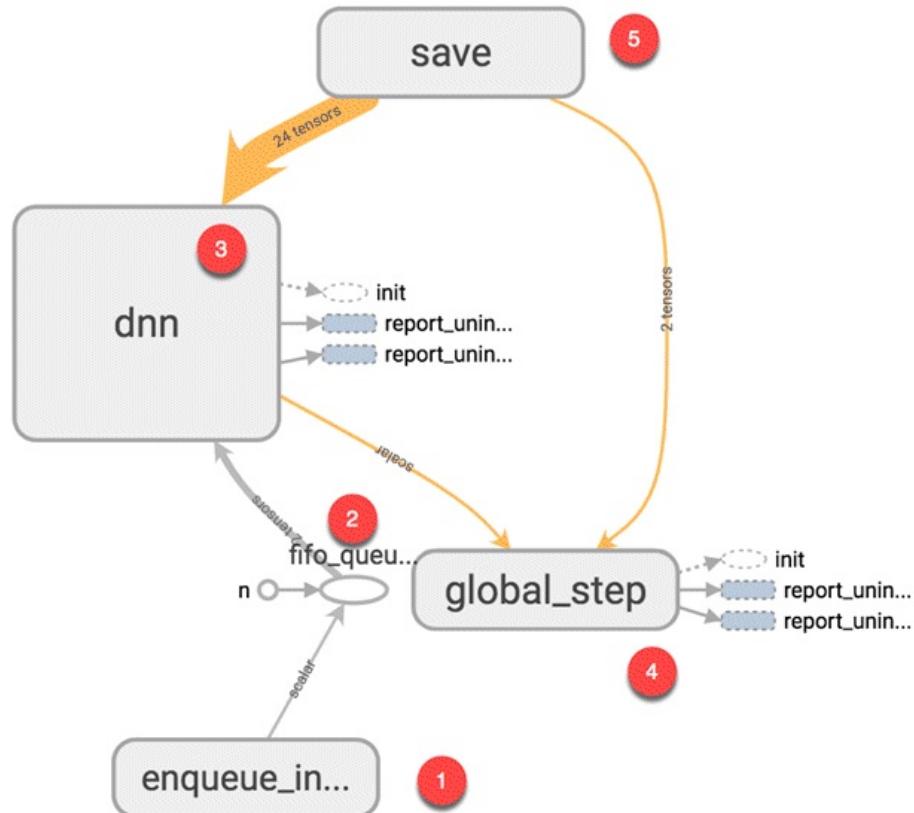
From the picture below, you can see the panel of Tensorboard. The panel contains different tabs, which are linked to the level of information you add when you run the model.

- Scalars: Show different useful information during the model training
- Graphs: Show the model
- Histogram: Display weights with a histogram
- Distribution: Display the distribution of the weight
- Projector: Show Principal component analysis and T-SNE algorithm. The technique uses for dimensionality reduction

During this tutorial, you will train a simple deep learning model. You will learn how it works in a future tutorial.

If you look at the graph, you can understand how the model work.

1. Enqueue the data to the model: Push an amount of data equal to the batch size to the model, i.e., Number of data feed after each iteration
2. Feed the data to the Tensors
3. Train the model
4. Display the number of batches during the training. Save the model on the disk.



The basic idea behind tensorboard is that neural network can be something known as a black box and we need a tool to inspect what's inside this box. You can imagine tensorboard as a flashlight to start dive into the neural network.

It helps to understand the dependencies between operations, how the weights are computed, displays the loss function and much other useful information. When you bring all these pieces of information together, you have a great tool to debug and find how to improve the model.

To give you an idea of how useful the graph can be, look at the picture below:



A neural network decides how to connect the different "neurons" and how many layers before the model can predict an outcome. Once you have defined the architecture, you not only need to train the model but also a metrics to compute the accuracy of the prediction. This metric is referred to as a **loss function**. The objective is to minimize the loss function. In different words, it means the model is making fewer errors. All machine learning algorithms will repeat many times the computations until the loss reach a flatter line. To minimize this loss function, you need to define a **learning rate**. It is the speed you want the model to learn. If you set a learning rate too high, the model does not have time to learn anything. This is the case in the left picture. The line is moving up and down, meaning the model predicts with pure guess the outcome. The picture on the right shows that the loss is decreasing over iteration until the curve got flatten, meaning the model found a solution.

TensorBoard is a great tool to visualize such metrics and highlight potential issues. The neural network can take hours to weeks before they find a solution. TensorBoard updates the metrics very often. In this case, you don't need to wait until the end to see if the model trains correctly. You can open TensorBoard check how the training is going and make the appropriate change if necessary.

In this tutorial, you will learn how to open TensorBoard from the terminal for MacOS and the Command line for Windows.

The code will be explained in a future tutorial, the focus here is on TensorBoard.

First, you need to import the libraries you will use during the training

```
## Import the library
import tensorflow as tf
import numpy as np
```

You create the data. It is an array of 10000 rows and 5 columns

```
x_train = (np.random.sample((10000,5)))
y_train = (np.random.sample((10000,1)))
x_train.shape
```

Output

```
(10000, 5)
```

The codes below transform the data and create the model.

Note that the learning rate is equal to 0.1. If you change this rate to a higher value, the model will not find a solution. This is what happened on the left side of the above picture.

During most of the TensorFlow tutorials, you will use TensorFlow estimator. This is TensorFlow API that contains all the mathematical computations.

To create the log files, you need to specify the path. This is done with the argument `model_dir`.

In the example below, you store the model inside the working directory, i.e., where you store the notebook or python file. Inside this path, TensorFlow will create a folder called `train` with a child folder name `linreg`.

```

feature_columns = [
    tf.feature_column.numeric_column('x',
shape=X_train.shape[1:])]
DNN_reg =
tf.estimator.DNNRegressor(feature_columns=feature_columns,
# Indicate where to store the log file
    model_dir='train/linreg',
    hidden_units=[500, 300],
    optimizer=tf.train.ProximalAdagradOptimizer(
        learning_rate=0.1,
        l1_regularization_strength=0.001
    )
)

```

Output

```

INFO:tensorflow:Using default config.
INFO:tensorflow:Using config: {'_model_dir': 'train/linreg',
'_tf_random_seed': None, '_save_summary_steps': 100,
'_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,
'_session_config': None, '_keep_checkpoint_max': 5,
'_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps':
100, '_train_distribute': None, '_service': None, '_cluster_spec':
<tensorflow.python.training.server_lib.ClusterSpec object at
0x1818e63828>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':
'', '_is_chief': True, '_num_ps_replicas': 0,
'_num_worker_replicas': 1}

```

The last step consists to train the model. During the training, TensorFlow writes information in the model directory.

```

# Train the estimator
train_input = tf.estimator.inputs.numpy_input_fn(
    x={"x": X_train},
    y=y_train, shuffle=False, num_epochs=None)
DNN_reg.train(train_input, steps=3000)

```

Output

```

INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow>Create CheckpointSaverHook.

```

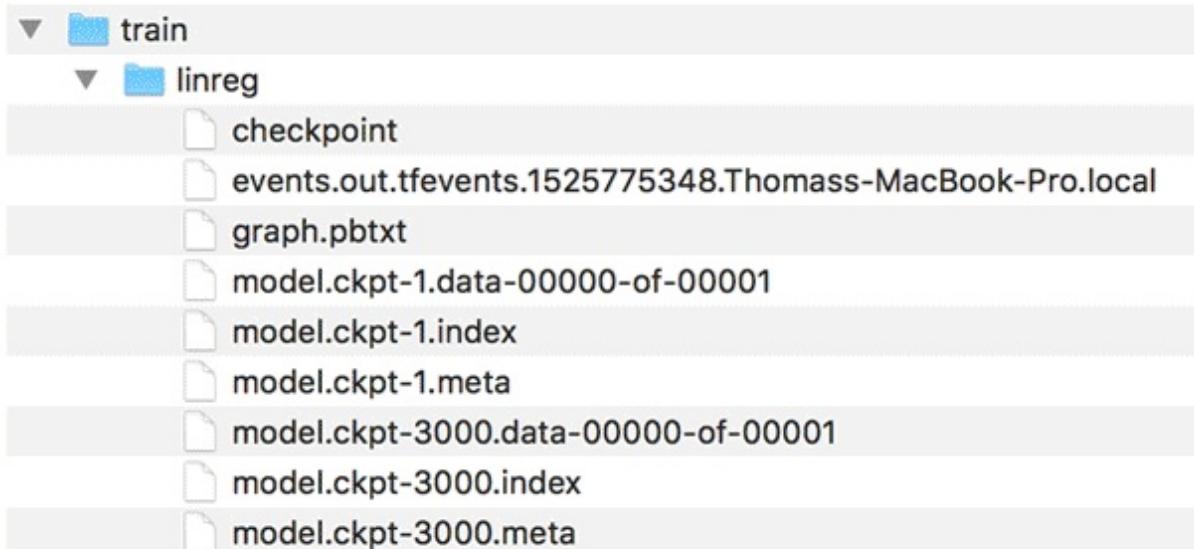
```
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Saving checkpoints for 1 into  
train/linreg/model.ckpt.  
INFO:tensorflow:loss = 40.060104, step = 1  
INFO:tensorflow:global_step/sec: 197.061  
INFO:tensorflow:loss = 10.62989, step = 101 (0.508 sec)  
INFO:tensorflow:global_step/sec: 172.487  
INFO:tensorflow:loss = 11.255318, step = 201 (0.584 sec)  
INFO:tensorflow:global_step/sec: 193.295  
INFO:tensorflow:loss = 10.604872, step = 301 (0.513 sec)  
INFO:tensorflow:global_step/sec: 175.378  
INFO:tensorflow:loss = 10.090343, step = 401 (0.572 sec)  
INFO:tensorflow:global_step/sec: 209.737  
INFO:tensorflow:loss = 10.057928, step = 501 (0.476 sec)  
INFO:tensorflow:global_step/sec: 171.646  
INFO:tensorflow:loss = 10.460144, step = 601 (0.583 sec)  
INFO:tensorflow:global_step/sec: 192.269  
INFO:tensorflow:loss = 10.529617, step = 701 (0.519 sec)  
INFO:tensorflow:global_step/sec: 198.264  
INFO:tensorflow:loss = 9.100082, step = 801 (0.504 sec)  
INFO:tensorflow:global_step/sec: 226.842  
INFO:tensorflow:loss = 10.485607, step = 901 (0.441 sec)  
INFO:tensorflow:global_step/sec: 152.929  
INFO:tensorflow:loss = 10.052481, step = 1001 (0.655 sec)  
INFO:tensorflow:global_step/sec: 166.745  
INFO:tensorflow:loss = 11.320213, step = 1101 (0.600 sec)  
INFO:tensorflow:global_step/sec: 161.854  
INFO:tensorflow:loss = 9.603306, step = 1201 (0.619 sec)  
INFO:tensorflow:global_step/sec: 179.074  
INFO:tensorflow:loss = 11.110269, step = 1301 (0.556 sec)  
INFO:tensorflow:global_step/sec: 202.776  
INFO:tensorflow:loss = 11.929443, step = 1401 (0.494 sec)  
INFO:tensorflow:global_step/sec: 144.161  
INFO:tensorflow:loss = 11.951693, step = 1501 (0.694 sec)  
INFO:tensorflow:global_step/sec: 154.144  
INFO:tensorflow:loss = 8.620987, step = 1601 (0.649 sec)  
INFO:tensorflow:global_step/sec: 151.094  
INFO:tensorflow:loss = 10.666125, step = 1701 (0.663 sec)  
INFO:tensorflow:global_step/sec: 193.644  
INFO:tensorflow:loss = 11.0349865, step = 1801 (0.516 sec)  
INFO:tensorflow:global_step/sec: 189.707  
INFO:tensorflow:loss = 9.860596, step = 1901 (0.526 sec)  
INFO:tensorflow:global_step/sec: 176.423
```

```
INFO:tensorflow:loss = 10.695, step = 2001 (0.567 sec)
INFO:tensorflow:global_step/sec: 213.066
INFO:tensorflow:loss = 10.426752, step = 2101 (0.471 sec)
INFO:tensorflow:global_step/sec: 220.975
INFO:tensorflow:loss = 10.594796, step = 2201 (0.452 sec)
INFO:tensorflow:global_step/sec: 219.289
INFO:tensorflow:loss = 10.4212265, step = 2301 (0.456 sec)
INFO:tensorflow:global_step/sec: 215.123
INFO:tensorflow:loss = 9.668612, step = 2401 (0.465 sec)
INFO:tensorflow:global_step/sec: 175.65
INFO:tensorflow:loss = 10.009649, step = 2501 (0.569 sec)
INFO:tensorflow:global_step/sec: 206.962
INFO:tensorflow:loss = 10.477722, step = 2601 (0.483 sec)
INFO:tensorflow:global_step/sec: 229.627
INFO:tensorflow:loss = 9.877638, step = 2701 (0.435 sec)
INFO:tensorflow:global_step/sec: 195.792
INFO:tensorflow:loss = 10.274586, step = 2801 (0.512 sec)
INFO:tensorflow:global_step/sec: 176.803
INFO:tensorflow:loss = 10.061047, step = 2901 (0.566 sec)
INFO:tensorflow:Saving checkpoints for 3000 into
train/linreg/model.ckpt.
INFO:tensorflow:Loss for final step: 10.73032.

<tensorflow.python.estimator.canned.dnn.DNNRegressor at
0x1818e63630>
```

For MacOS user

New folder inside the working directory



For Windows user

C:\Users\Admin\train\linreg			
	Name	Date modified	Type
	checkpoint	12-05-2018 12:51 ...	File
	events.out.tfevents.1526109703.DESKTOP	12-05-2018 12:51 ...	DESKTOP-5OED84...
	graph.pbtxt	12-05-2018 12:51 ...	PBTXT File
	model.ckpt-1.data-00000-of-00001	12-05-2018 12:51 ...	DATA-00000-OF-0...
	model.ckpt-1.index	12-05-2018 12:51 ...	INDEX File
	model.ckpt-1.meta	12-05-2018 12:51 ...	META File
	model.ckpt-3000.data-00000-of-00001	12-05-2018 12:51 ...	DATA-00000-OF-0...
	model.ckpt-3000.index	12-05-2018 12:51 ...	INDEX File
	model.ckpt-3000.meta	12-05-2018 12:51 ...	META File

You can see this information in the TensorBoard.

Now that you have the log events written, you can open Tensorboard. Tensorboard runs on port 6006 (Jupyter runs on port 8888). You can use the Terminal for MacOs user or Anaconda prompt for Windows user.

For MacOS user

```
# Different for you
cd /Users/Guru99/tuto_TF
source activate hello-tf!
```

The notebook is stored in the path /Users/Guru99/tuto_TF

For Windows users

```
cd C:\Users\Admin\Anaconda3
activate hello-tf
```

The notebook is stored in the path C:\Users\Admin\Anaconda3

To launch Tensorboard, you can use this code

For MacOS user

```
tensorboard --logdir=./train/linreg
```

For Windows users

```
tensorboard --logdir=.\train\linreg
```

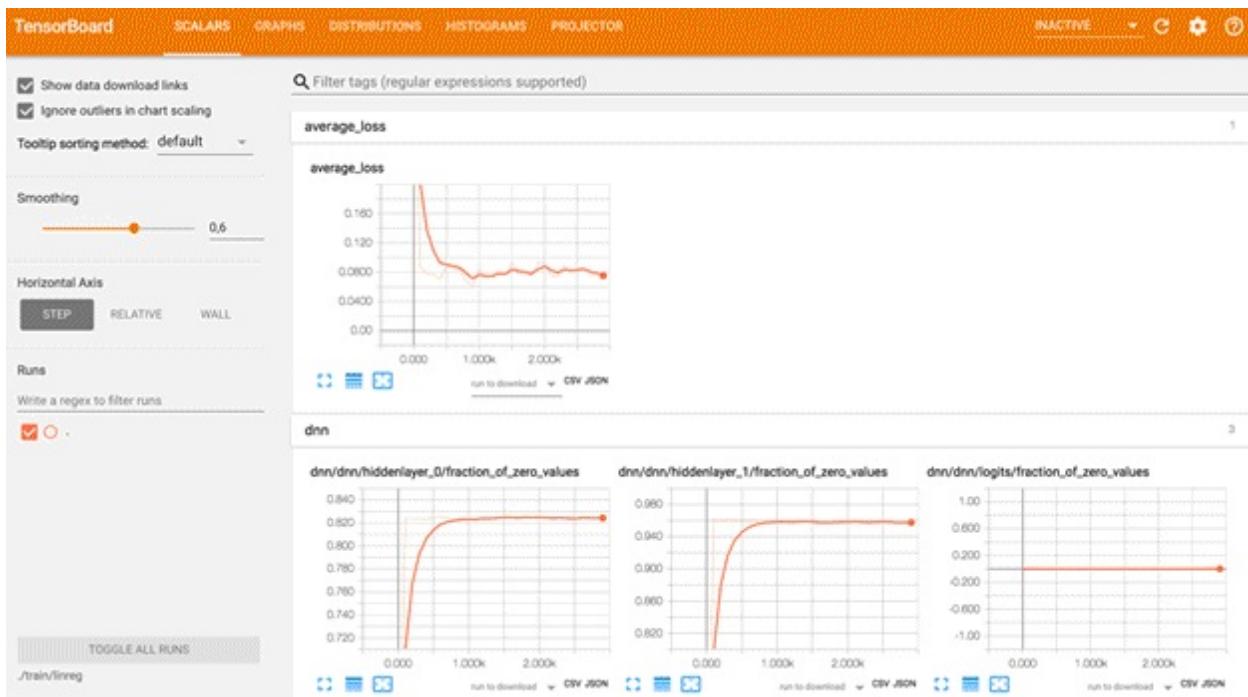
Tensorboard is located in this URL: <http://localhost:6006>

It could also be located at the following location.

```
(hello-tf) C:\Users\Admin>tensorboard --logdir=.\train\linreg
2018-05-13 19:08:34.355370: I T:\src\github\tensorflow\tensorflow\core\platform\cpu_feature_guard.cc:140] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
W0513 19:08:34.371735 Reloader tf_logging.py:121] Found more than one graph event per run, or there was a metagraph containing a graph_def, as well as one or more graph events. Overwriting the graph with the newest event.
W0513 19:08:34.373239 Reloader tf_logging.py:121] Found more than one metagraph event per run. Overwriting the metagraph with the newest event.
TensorBoard 1.8.0 at http://DESKTOP-5OED84V:6006 (Press CTRL+C to quit)
```

Copy and paste the URL into your favorite browser. You should see this:

Note that, we will learn how to read the graph in the tutorial dedicated to the deep learning.



If you see something like this:

No graph definition files were found.

To store a graph, create a `tf.summary.FileWriter` and pass the graph either via the constructor, or by calling its `add_graph()` method. You may want to check out the [graph visualizer tutorial](#).

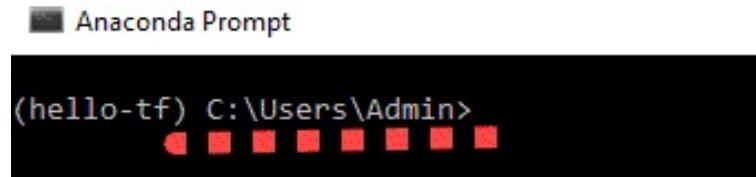
If you're new to using TensorBoard, and want to find out how to add data and set up your event files, check out the [README](#) and perhaps the [TensorBoard tutorial](#).

If you think TensorBoard is configured properly, please see [the section of the README devoted to missing data problems](#) and consider filing an issue on GitHub.

It means Tensorboard cannot find the log file. Make sure you point the cd to the right path or double check if the log event has been creating. If not, re-run the code.

If you want to close TensorBoard Press CTRL+C

Hat Tip: Check your anaconda prompt for the current working directory,



A screenshot of an Anaconda Prompt window. The title bar says "Anaconda Prompt". The command line shows "(hello-tf) C:\Users\Admin>". The background is black and the text is white.

The log file should be created at C:\Users\Admin

Summary:

TensorBoard is a great tool to visualize your model. Besides, many metrics are displayed during the training, such as the loss, accuracy or weights.

To activate Tensorboard, you need to set the path of your file:

```
cd /Users/Guru99/tuto_TF
```

Activate Tensorflow's environment

```
activate hello-tf
```

Launch Tensorboard

```
tensorboard --logdir=.+ PATH
```

Chapter 10: NumPy

What is NumPy?

NumPy is an open source library available in Python that aids in mathematical, scientific, engineering, and data science programming. NumPy is an incredible library to perform mathematical and statistical operations. It works perfectly well for multi-dimensional arrays and matrices multiplication For any scientific project, NumPy is the tool to know. It has been built to work with the N-dimensional array, linear algebra, random number, Fourier transform, etc. It can be integrated to C/C++ and Fortran.

NumPy is a programming language that deals with multi-dimensional arrays and matrices. On top of the arrays and matrices, NumPy supports a large number of mathematical operations. In this part, we will review the essential functions that you need to know for the tutorial on 'TensorFlow.'

Why use NumPy?

NumPy is memory efficient, meaning it can handle the vast amount of data more accessible than any other library. Besides, NumPy is very convenient to work with, especially for matrix multiplication and reshaping. On top of that, NumPy is fast. In fact, TensorFlow and Scikit learn to use NumPy array to compute the matrix multiplication in the back end.

How to install NumPy?

To install Pandas library, please refer our tutorial How to install TensorFlow. NumPy is installed by default. In remote case, NumPy not installed-You can install NumPy using:

- Anaconda: conda conda install -c anaconda numpy
- In Jupyter Notebook :

```
import sys  
!conda install --yes --prefix {sys.prefix} numpy
```

Import NumPy and Check Version

The command to import numpy is

```
import numpy as np
```

Above code renames the Numpy namespace to np. This permits us to prefix Numpy function, methods, and attributes with " np " instead of typing " numpy." It is the standard shortcut you will find in the numpy literature To check your installed version of Numpy use the command print (np.__version__)

Output

```
1.14.0
```

Create a NumPy Array

Simplest way to create an array in Numpy is to use Python List myPythonList = [1,9,8,3]

To convert python list to a numpy array by using the object np.array.

```
numpy_array_from_list = np.array(myPythonList)
```

To display the contents of the list

```
numpy_array_from_list
```

Output

```
array([1, 9, 8, 3])
```

In practice, there is no need to declare a Python List. The operation can be combined.

```
a = np.array([1,9,8,3])
```

NOTE: Numpy documentation states use of np.ndarray to create an array. However, this is the recommended method. You can also create a numpy array from a Tuple

Mathematical Operations on an Array

You could perform mathematical operations like additions, subtraction, division and multiplication on an array. The syntax is the array name followed by the operation (+,-,*,/) followed by the operand Example:

```
numpy_array_from_list + 10
```

Output:

```
array([11, 19, 18, 13])
```

This operation adds 10 to each element of the numpy array.

Shape of Array

You can check the shape of the array with the object shape preceded by the name of the array. In the same way, you can check the type with dtypes.

```
import numpy as np
a = np.array([1,2,3])
print(a.shape)
print(a.dtype)

(3,)
int64
```

An integer is a value without decimal. If you create an array with decimal, then the type will change to float.

```
#### Different type
b = np.array([1.1,2.0,3.2])
print(b.dtype)

float64
```

2 Dimension Array

You can add a dimension with a "," coma

Note that it has to be within the bracket []

```
### 2 dimension
c = np.array([(1,2,3),
              (4,5,6)])
print(c.shape)
(2, 3)
```

3 Dimension Array

Higher dimension can be constructed as follow:

```
### 3 dimension
d = np.array([
    [[1, 2, 3],
     [4, 5, 6]],
    [[7, 8, 9],
     [10, 11, 12]]
])
print(d.shape)
(2, 2, 3)
```

np.zeros and np.ones

You can create matrix full of zeroes or ones. It can be used when you initialized the weights during the first iteration in TensorFlow.

The syntax is

```
numpy.zeros(shape, dtype=float, order='C')
```

```
numpy.ones(shape, dtype=float, order='C')
```

Here,

Shape: is the shape of the array **Dtype:** is the datatype. It is optional. The default value is float64

Order: Default is C which is an essential row style.

Example:

```
np.zeros((2,2))
```

Output:

```
array([[0., 0.],  
       [0., 0.]])
```

```
np.zeros((2,2), dtype=np.int16)
```

Output:

```
array([[0, 0],  
       [0, 0]], dtype=int16)
```

```
## Create 1  
np.ones((1,2,3), dtype=np.int16)  
array([[[1, 1, 1],  
       [1, 1, 1]]], dtype=int16)
```

Reshape and Flatten Data

In some occasion, you need to reshape the data from wide to long.

```
e = np.array([(1,2,3), (4,5,6)])
print(e)
e.reshape(3,2)
```

Output:

```
[[1 2 3]
 [4 5 6]]
```

```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

When you deal with some neural network like convnet, you need to flatten the array. You can use flatten() e.flatten()

```
array([1, 2, 3, 4, 5, 6])
```

hstack and vstack

Numpy library has also two convenient function to horizontally or vertically append the data. Lets study them with an example:

```
## Stack
f = np.array([1,2,3])
g = np.array([4,5,6])
print('Horizontal Append:', np.hstack((f, g)))
print('Vertical Append:', np.vstack((f, g)))
```

Horizontal Append: [1 2 3 4 5 6]
Vertical Append:
[[1 2 3] [4 5 6]]

Generate Random Numbers

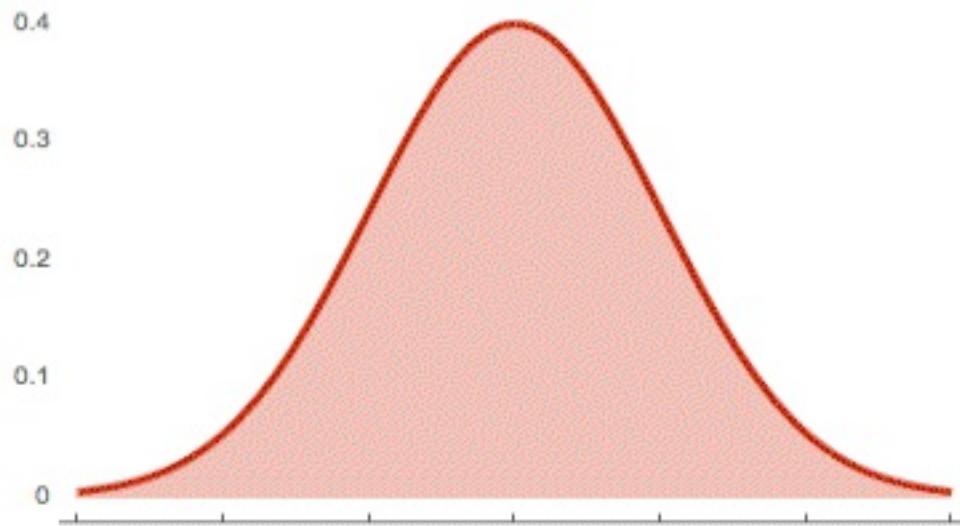
To generate random numbers for Gaussian distribution use numpy
.random.normal(loc, scale, size)

Here

- Loc: the mean. The center of distribution
- scale: standard deviation.
- Size: number of returns

```
## Generate random nmber from normal distribution
normal_array = np.random.normal(5, 0.5, 10)
print(normal_array)
[5.56171852 4.84233558 4.65392767 4.946659   4.85165567 5.61211317
 4.46704244 5.22675736 4.49888936 4.68731125]
```

If plotted the distribution will be similar to following plot



Asarray

Consider the following 2-D matrix with four rows and four columns filled by 1

```
A = np.matrix(np.ones((4,4)))
```

If you want to change the value of the matrix, you cannot. The reason is, it is not possible to change a copy.

```
np.array(A)[2]=2
print(A)
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

Matrix is immutable. You can use asarray if you want to add modification in the original array. let's see if any change occurs when you want to change the value of the third rows with the value 2

```
np.asarray(A)[2]=2
print(A)
```

Code Explanation:

np.asarray(A): converts the matrix A to an array

[2]: select the third rows

```
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [2. 2. 2. 2.] # new value
 [1. 1. 1. 1.]]
```

Arrange

In some occasion, you want to create value evenly spaced within a given interval. For instance, you want to create values from 1 to 10; you can use arrange Syntax:

```
numpy.arange(start, stop, step)
```

- Start: Start of interval
- Stop: End of interval
- Step: Spacing between values. Default step is 1

Example:

```
np.arange(1, 11)
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

If you want to change the step, you can add a third number in the parenthesis. It will change the step.

```
np.arange(1, 14, 4)
```

```
array([ 1,  5,  9, 13])
```

Linspace

Linspace gives evenly spaced samples.

Syntax:

```
numpy.linspace(start, stop, num, endpoint)
```

Here,

- Start: Starting value of the sequence
- Stop: End value of the sequence
- Num: Number of samples to generate. Default is 50
- Endpoint: If True (default), stop is the last value. If False, stop value is not included.

For instance, it can be used to create 10 values from 1 to 5 evenly spaced.

```
np.linspace(1.0, 5.0, num=10)  
array([1.          , 1.44444444, 1.88888889, 2.33333333, 2.77777778,  
3.22222222, 3.66666667, 4.11111111, 4.55555556, 5.          ])
```

If you do not want to include the last digit in the interval, you can set endpoint to false np.linspace(1.0, 5.0, num=5, endpoint=False)

```
array([1. , 1.8, 2.6, 3.4, 4.2])
```

LogSpace

LogSpace returns even spaced numbers on a log scale. Logspace has the same parameters as np.linspace.

```
np.logspace(3.0, 4.0, num=4)
```

```
array([ 1000. , 2154.43469003, 4641.58883361, 10000. ])
```

Finally, if you want to check the size of an array, you can use itemsize x = np.array([1,2,3], dtype=np.complex128) x.itemsize

16

The x element has 16 bytes.

Indexing and slicing

Slicing data is trivial with numpy. We will slice the matrix e. Note that, in Python, you need to use the brackets to return the rows or columns ## Slice e = np.array([(1,2,3), (4,5,6)]) print(e) [[1 2 3] [4 5 6]]

Remember with numpy the first array/column starts at 0.

```
## First column
print('First row:', e[0])

## Second col
print('Second row:', e[1])
First row: [1 2 3]
Second row: [4 5 6]
```

In Python, like many other languages,

- The values before the comma stand for the rows
- The value on the right stands for the columns.
- If you want to select a column, you need to add : before the column index.
- : means you want all the rows from the selected column.

```
print('Second column:', e[:,1])
```

```
Second column: [2 5]
```

To return the first two values of the second row. You use : to select all columns

up to the second ## print(e[1, :2]) [4 5]

Statistical function

Numpy is equipped with the robust statistical function as listed below

Function	Numpy
Min	np.min()
Max	np.max()
Mean	np.mean()
Median	np.median()
Standard deviation	np.std()

```
## Statistical function
### Min
print(np.min(normal_array))

### Max
print(np.max(normal_array))
### Mean
print(np.mean(normal_array))
### Median
print(np.median(normal_array))
### Sd
print(np.std(normal_array))
```

```
4.467042435266913
5.612113171990201
4.934841002270593
4.846995625786663
0.3875019367395316
```

Dot Product

Numpy is powerful library for matrices computation. For instance, you can compute the dot product with np.dot ## Linear algebra ### Dot product: product of two arrays f = np.array([1,2]) g = np.array([4,5]) ### $1*4+2*5$ np.dot(f, g)

14

Matrix Multiplication

In the same way, you can compute matrices multiplication with np.matmul ### Matmul: matruc product of two arrays h = [[1,2],[3,4]] i = [[5,6],[7,8]] ### $1*5+2*7 = 19$ np.matmul(h, i)

```
array([[19, 22],  
       [43, 50]])
```

Determinant

Last but not least, if you need to compute the determinant, you can use np.linalg.det(). Note that numpy takes care of the dimension.

```
## Determinant 2*2 matrix  
### 5*8-7*6np.linalg.det(i)  
  
-2.000000000000005
```

Summary

Below, a summary of the essential functions used with NumPy

Objective	Code
Create array	array([1,2,3])
print the shape	array([.]).shape
reshape	reshape
flat an array	flatten
append vertically	vstack
append horizontally	hstack
create a matrix	matrix
create space	arrange
Create a linear space	linspace
Create a log space	logspace

Below is a summary of basic statistical and arithmetical function

Objective	Code

min	min()
max	max()
mean	mean()
median	median()
standard deviation	std()

Here is the complete code:

```
import numpy as np

##Create array
### list
myPythonList = [1,9,8,3]
numpy_array_from_list = np.array(myPythonList)
### Directly in numpy
np.array([1,9,8,3])

### Shape
a = np.array([1,2,3])
print(a.shape)

### Type
print(a.dtype)

### 2D array
c = np.array([(1,2,3),
(4,5,6)])
print("2d Array",c)

### 3D array
d = np.array([
[[1, 2, 3],
[4, 5, 6]],
[[7, 8, 9],
[10, 11, 12]]
])
```

```
print("3d Array",d)
### Reshape
e = np.array([(1,2,3), (4,5,6)])
print(e)
e.reshape(3,2)
print("After Reshape",e)
### Flatten
e.flatten()

print("After Flatten",e)

### hstack & vstack
f = np.array([1,2,3])
g = np.array([4,5,6])

print('Horizontal Append:', np.hstack((f, g)))
print('Vertical Append:', np.vstack((f, g)))

### random number
normal_array = np.random.normal(5, 0.5, 10)
print("Random Number",normal_array)

### asarray
A = np.matrix(np.ones((4,4)))
np.asarray(A)
print("Asrray",A)
### Arrange

print("Arrange",np.arange(1, 11))

### linspace

lin = np.linspace(1.0, 5.0, num=10)
print("Linspace",lin)

### logspace
log1 = np.logspace(3.0, 4.0, num=4)
print("Logspace",log1)

### Slicing
#### rows
e = np.array([(1,2,3), (4,5,6)])
```

```
print(e[0])  
  
#### columns  
print(e[:,1])  
  
#### rows and columns  
print(e[1, :2])
```

Chapter 11: Pandas

What is Pandas?

Pandas is an opensource library that allows to you perform data manipulation in Python. Pandas library is built on top of Numpy, meaning Pandas needs Numpy to operate. Pandas provide an easy way to create, manipulate and wrangle the data. Pandas is also an elegant solution for time series data.

Why use Pandas?

Data scientists use Pandas for its following advantages:

- Easily handles missing data
- It uses **Series for one-dimensional data structure** and **DataFrame for multi-dimensional data structure**
- It provides an efficient way to slice the data
- It provides a flexible way to merge, concatenate or reshape the data
- It includes a powerful time series tool to work with

In a nutshell, Pandas is a useful library in data analysis. It can be used to perform data manipulation and analysis. Pandas provide powerful and easy-to-use data structures, as well as the means to quickly perform operations on these structures.

How to install Pandas?

To install Pandas library, please refer our tutorial How to install TensorFlow. Pandas is installed by default. In remote case, pandas not installed- You can install Pandas using:

- Anaconda: conda install -c anaconda pandas
- In Jupyter Notebook :

```
import sys  
  
!conda install --yes --prefix {sys.prefix} pandas
```

What is a data frame?

A data frame is a two-dimensional array, with labeled axes (rows and columns).
A data frame is a standard way to store data.

Data frame is well-known by statistician and other data practitioners. A data frame is a tabular data, with rows to store the information and columns to name the information. For instance, the price can be the name of a column and 2,3,4 the price values.

Below a picture of a Pandas data frame:

	Item	Price
0	A	2
1	B	3

What is a Series?

A series is a one-dimensional data structure. It can have any data structure like integer, float, and string. It is useful when you want to perform computation or return a one-dimensional array. A series, by definition, cannot have multiple columns. For the latter case, please use the data frame structure.

Series has one parameters:

- Data: can be a list, dictionary or scalar value

```
pd.Series([1., 2., 3.])
```

```
0    1.0
```

```
1    2.0
```

```
2    3.0
```

```
dtype: float64
```

You can add the index with index. It helps to name the rows. The length should be equal to the size of the column pd.Series([1., 2., 3.], index=['a', 'b', 'c'])

Below, you create a Pandas series with a missing value for the third rows. Note,

missing values in Python are noted "NaN." You can use numpy to create missing value: np.nan artificially pd.Series([1,2,np.nan])

Output

```
0    1.0
1    2.0
2    NaN
dtype: float64
```

Create Data frame

You can convert a numpy array to a pandas data frame with pd.DataFrame(). The opposite is also possible. To convert a pandas Data Frame to an array, you can use np.array() ## Numpy to pandas

```
import numpy as np
h = [[1,2],[3,4]]
df_h = pd.DataFrame(h)
print('Data Frame:', df_h)
## Pandas to numpy
```

```
df_h_n = np.array(df_h)

print('Numpy array:', df_h_n)
```

Data Frame: 0 1

0 1 2

1 3 4

Numpy array: [[1 2]

[3 4]]

You can also use a dictionary to create a Pandas dataframe.

```
dic = {'Name': ["John", "Smith"], 'Age': [30, 40]}

pd.DataFrame(data=dic)
```

	Age	Name
0	30	John
1	40	Smith

Range Data

Pandas have a convenient API to create a range of date

pd.date_range(date, period, frequency):

- The first parameter is the starting date

- The second parameter is the number of periods (optional if the end date is specified)
- The last parameter is the frequency: day: 'D,' month: 'M' and year: 'Y.'

```
## Create date

# Days

dates_d = pd.date_range('20300101', periods=6, freq='D')

print('Day:', dates_d)
```

Output

```
Day: DatetimeIndex(['2030-01-01', '2030-01-02', '2030-01-03',
'2030-01-04', '2030-01-05', '2030-01-06'], dtype='datetime64[ns]',
freq='D')
```

```
# Months

dates_m = pd.date_range('20300101', periods=6, freq='M')

print('Month:', dates_m)
```

Output

```
Month: DatetimeIndex(['2030-01-31', '2030-02-28', '2030-03-31',
'2030-04-30', '2030-05-31', '2030-06-30'], dtype='datetime64[ns]',
```

```
freq='M')
```

Inspecting data

You can check the head or tail of the dataset with head(), or tail() preceded by the name of the panda's data frame **Step 1)** Create a random sequence with numpy. The sequence has 4 columns and 6 rows random = np.random.randn(6,4)

Step 2) Then you create a data frame using pandas.

Use dates_m as an index for the data frame. It means each row will be given a "name" or an index, corresponding to a date.

Finally, you give a name to the 4 columns with the argument columns # Create data with date

```
df = pd.DataFrame(random,
```

```
index=dates_m,
```

```
columns=list('ABCD'))
```

Step 3) Using head function df.head(3)

	A	B	C	D
2030-01-31	1.139433	1.318510	-0.181334	1.615822
2030-02-28	-0.081995	-0.063582	0.857751	-0.527374
2030-03-31	-0.519179	0.080984	-1.454334	1.314947

Step 4) Using tail function

```
df.tail(3)
```

	A	B	C	D

2030-04-30	-0.685448	-0.011736	0.622172	0.104993
2030-05-31	-0.935888	-0.731787	-0.558729	0.768774
2030-06-30	1.096981	0.949180	-0.196901	-0.471556

Step 5) An excellent practice to get a clue about the data is to use `describe()`. It provides the counts, mean, std, min, max and percentile of the dataset.

```
df.describe()
```

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	0.002317	0.256928	-0.151896	0.467601
std	0.908145	0.746939	0.834664	0.908910
min	-0.935888	-0.731787	-1.454334	-0.527374
25%	-0.643880	-0.050621	-0.468272	-0.327419
50%	-0.300587	0.034624	-0.189118	0.436883
75%	0.802237	0.732131	0.421296	1.178404
max	1.139433	1.318510	0.857751	1.615822

Slice data

The last point of this tutorial is about how to slice a pandas data frame.

You can use the column name to extract data in a particular column.

```
## Slice

### Using name

df['A']
```

```
2030-01-31    -0.168655
```

```
2030-02-28    0.689585
```

```
2030-03-31    0.767534
```

```
2030-04-30    0.557299
```

```
2030-05-31    -1.547836
```

```
2030-06-30    0.511551
```

```
Freq: M, Name: A, dtype: float64
```

To select multiple columns, you need to use two times the bracket, [[...]]

The first pair of bracket means you want to select columns, the second pairs of bracket tells what columns you want to return.

```
df[['A', 'B']].
```

	A	B
2030-01-31	-0.168655	0.587590
2030-02-28	0.689585	0.998266

2030-03-31	0.767534	-0.940617
2030-04-30	0.557299	0.507350
2030-05-31	-1.547836	1.276558
2030-06-30	0.511551	1.572085

You can slice the rows with :

The code below returns the first three rows ### using a slice for row

df[0:3]

	A	B	C	D
2030-01-31	-0.168655	0.587590	0.572301	-0.031827
2030-02-28	0.689585	0.998266	1.164690	0.475975
2030-03-31	0.767534	-0.940617	0.227255	-0.341532

The loc function is used to select columns by names. As usual, the values before the coma stand for the rows and after refer to the column. You need to use the brackets to select more than one column.

```
## Multi col

df.loc[:, ['A', 'B']]
```

	A	B
2030-01-31	-0.168655	0.587590
2030-02-28	0.689585	0.998266
2030-03-31	0.767534	-0.940617
2030-04-30	0.557299	0.507350
2030-05-31	-1.547836	1.276558
2030-06-30	0.511551	1.572085

There is another method to select multiple rows and columns in Pandas. You can use iloc[]. This method uses the index instead of the columns name. The code

below returns the same data frame as above df.iloc[:, :2]

	A	B
2030-01-31	-0.168655	0.587590
2030-02-28	0.689585	0.998266
2030-03-31	0.767534	-0.940617
2030-04-30	0.557299	0.507350
2030-05-31	-1.547836	1.276558
2030-06-30	0.511551	1.572085

Drop a column

You can drop columns using pd.drop()

```
df.drop(columns=['A', 'C'])
```

	B	D
2030-01-31	0.587590	-0.031827
2030-02-28	0.998266	0.475975
2030-03-31	-0.940617	-0.341532
2030-04-30	0.507350	-0.296035
2030-05-31	1.276558	0.523017
2030-06-30	1.572085	-0.594772

Concatenation

You can concatenate two DataFrame in Pandas. You can use pd.concat() First of all, you need to create two DataFrames. So far so good, you are already familiar with dataframe creation import numpy as np

```
df1 = pd.DataFrame({'name': ['John', 'Smith', 'Paul'],
'Age': ['25', '30', '50']},
index=[0, 1, 2])
```

```
df2 = pd.DataFrame({'name': ['Adam', 'Smith'],
'Age': ['26', '11']},
index=[3, 4])
```

Finally, you concatenate the two DataFrame

```
df_concat = pd.concat([df1, df2])
```

```
df_concat
```

	Age	name
0	25	John
1	30	Smith
2	50	Paul
3	26	Adam
4	11	Smith

Drop_duplicates

If a dataset can contain duplicates information use, `drop_duplicates` is an easy to exclude duplicate rows. You can see that `df_concat` has a duplicate observation, `Smith` appears twice in the column `name`.

```
df_concat.drop_duplicates('name')
```

	Age	name
0	25	John
1	30	Smith
2	50	Paul
3	26	Adam

Sort values

You can sort value with sort_values

```
df_concat.sort_values('Age')
```

	Age	name
4	11	Smith
0	25	John
3	26	Adam
1	30	Smith
2	50	Paul

Rename: change of index

You can use rename to rename a column in Pandas. The first value is the current

column name and the second value is the new column name.

```
df_concat.rename(columns={"name": "Surname", "Age": "Age_ppl"})
```

	Age_ppl	Surname
0	25	John
1	30	Smith
2	50	Paul
3	26	Adam
4	11	Smith

Import CSV

During the TensorFlow tutorial, you will use the adult dataset. It is often used with classification task. It is available in this URL

<https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data> The data is stored in a CSV format. This dataset includes eight categorical variables: This dataset includes eight categorical variables:

- workclass
- education
- marital
- occupation
- relationship
- race
- sex
- native_country

moreover, six continuous variables:

- age
- fnlwgt
- education_num
- capital_gain
- capital_loss

hours_week

To import a CSV dataset, you can use the object pd.read_csv(). The basic argument inside is: **Syntax**:

```
pandas.read_csv(filepath_or_buffer, sep=',', `names=None`, `index_col=None`, `skipinitialspace=False`)
```

- filepath_or_buffer: Path or URL with the data
- sep=', ': Define the delimiter to use
- `names=None`: Name the columns. If the dataset has ten columns, you need to pass ten names
- `index_col=None`: If yes, the first column is used as a row index
- `skipinitialspace=False`: Skip spaces after delimiter.

For more information about readcsv(), please check the official documentation
https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html.

Consider the following Example

```
## Import csv

import pandas as pd

## Define path data

COLUMNS = ['age', 'workclass', 'fnlwgt', 'education',
'education_num', 'marital', 'occupation', 'relationship', 'race',
'sex', 'capital_gain', 'capital_loss', 'hours_week',
'native_country', 'label']

PATH = "https://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.data"

df_train = pd.read_csv(PATH,
                      skipinitialspace=True,
                      names = COLUMNS,
                      index_col=False)

df_train.shape
```

Output:

```
(32561, 15)
```

Groupby

An easy way to see the data is to use the groupby method. This method can help you to summarize the data by group. Below is a list of methods available with groupby:

- count: count
- min: min
- max: max
- mean: mean
- median: median
- standard deviation: std
- etc

Inside groupby(), you can use the column you want to apply the method.

Let's have a look at a single grouping with the adult dataset. You will get the mean of all the continuous variables by type of revenue, i.e., above 50k or below 50k `df_train.groupby(['label']).mean()`

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_week
label						
<=50K	36.783738	190340.86517	9.595065	148.752468	53.142921	38.840210
>50K	44.249841	188005.00000	11.611657	4006.142456	195.001530	45.473026

You can get the minimum of age by type of household df_train.groupby(['label'])['age'].min()

```
label  
  
<=50K      17  
  
>50K      19  
  
Name: age, dtype: int64
```

You can also group by multiple columns. For instance, you can get the maximum capital gain according to the household type and marital status.

```
df_train.groupby(['label', 'marital'])['capital_gain'].max()  
label    marital  
  
<=50K  Divorced          34095  
  
        Married-AF-spouse    2653  
  
        Married-civ-spouse   41310  
  
        Married-spouse-absent 6849  
  
        Never-married        34095
```

	Separated	7443
	Widowed	6849
>50K	Divorced	99999
	Married-AF-spouse	7298
	Married-civ-spouse	99999
	Married-spouse-absent	99999
	Never-married	99999
	Separated	99999
	Widowed	99999

Name: capital_gain, dtype: int64

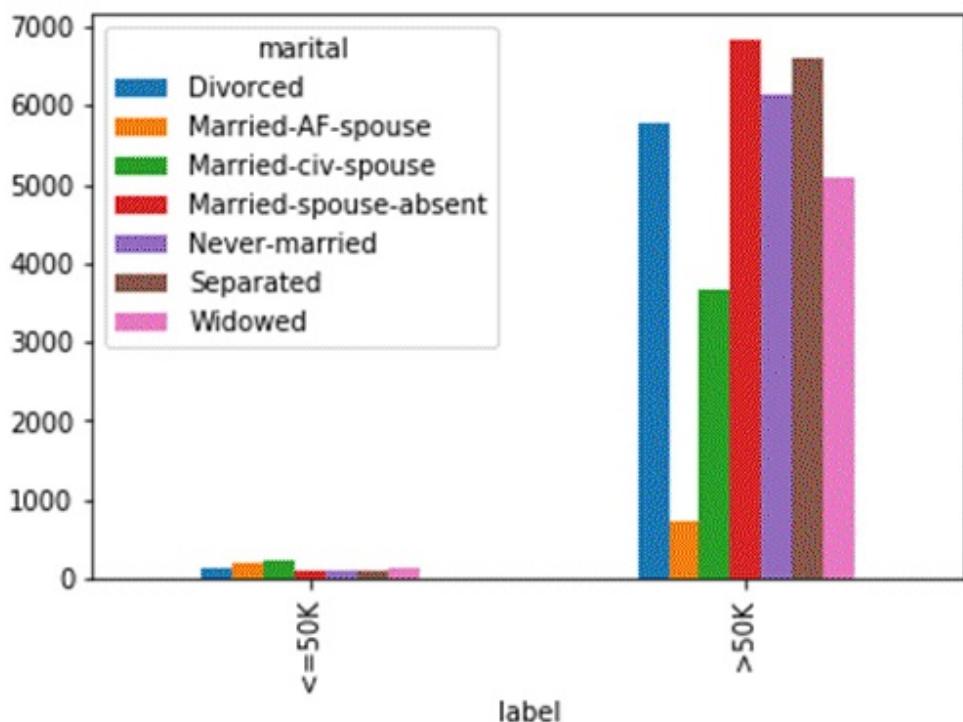
You can create a plot following groupby. One way to do it is to use a plot after the grouping.

To create a more excellent plot, you will use unstack() after mean() so that you have the same multilevel index, or you join the values by revenue lower than

50k and above 50k. In this case, the plot will have two groups instead of 14 (2^7).

If you use Jupyter Notebook, make sure to add % matplotlib inline, otherwise, no plot will be displayed % matplotlib inline

```
df_plot = df_train.groupby(['label', 'marital'])['capital_gain'].mean().unstack()  
df_plot
```



Summary

Below is a summary of the most useful method for data science with Pandas

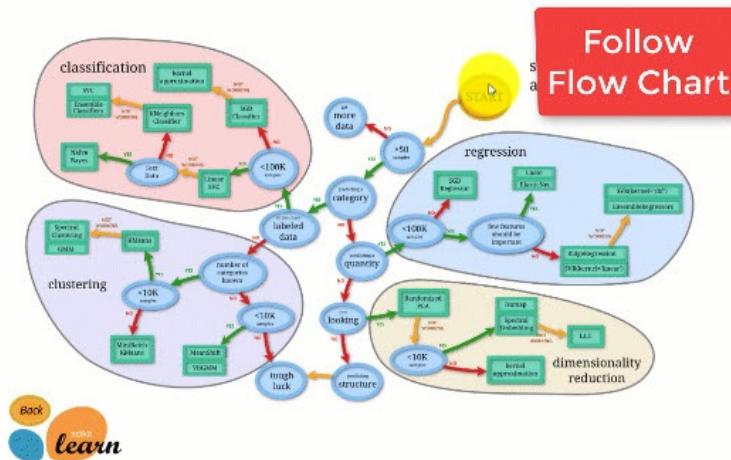
import data	read_csv
create series	Series
Create Dataframe	DataFrame
Create date range	date_range
return head	head
return tail	tail
Describe	describe
slice using name	dataname['columnname']
Slice using rows	data_name[0:5]

Chapter 12: Scikit-Learn

What is Scikit-learn?

Scikit-learn is an open source Python library for machine learning. The library supports state-of-the-art algorithms such as KNN, XGBoost, random forest, SVM among others. It is built on top of Numpy. Scikit-learn is widely used in kaggle competition as well as prominent tech companies. Scikit-learn helps in preprocessing, dimensionality reduction(parameter selection), classification, regression, clustering, and model selection.

Scikit-learn has the best documentation of all opensource libraries. It provides you an interactive chart at http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html.



Scikit-learn is not very difficult to use and provides excellent results. However, scikit learn does not support parallel computations. It is possible to run a deep learning algorithm with it but is not an optimal solution, especially if you know how to use TensorFlow.

Download and Install scikit-learn

Option 1: AWS

scikit-learn can be used over AWS. Please refer The docker image has scikit-learn preinstalled.

To use developer version use the command in Jupyter

```
import sys  
!{sys.executable} -m pip install git+git://github.com/scikit-  
learn/scikit-learn.git
```

Option 2: Mac or Windows using Anaconda

To learn about Anaconda installation refer <https://www.guru99.com/download-install-tensorflow.html>

Recently, the developers of scikit have released a development version that tackles common problem faced with the current version. We found it more convenient to use the developer version instead of the current version.

If you installed scikit-learn with the conda environment, please follow the step to update to version 0.20

Step 1) Activate tensorflow environment

```
source activate hello-tf
```

Step 2) Remove scikit lean using the conda command

```
conda remove scikit-learn
```

Step 3) Install scikit learn developer version along with necessary libraries.

```
conda install -c anaconda git  
pip install Cython  
pip install h5py  
pip install git+git://github.com/scikit-learn/scikit-learn.git
```

NOTE: Windows users will need to install Microsoft Visual C++ 14. You can get it from [here](#)

Machine learning with scikit-learn

This tutorial is divided into two parts:

1. Machine learning with scikit-learn
2. How to trust your model with LIME

The first part details how to build a pipeline, create a model and tune the hyperparameters while the second part provides state-of-the-art in term of model selection.

Step 1) Import the data

During this tutorial, you will be using the adult dataset. For a background in this dataset refer If you are interested to know more about the descriptive statistics, please use Dive and Overview tools. Refer this tutorial learn more about Dive and Overview

You import the dataset with Pandas. Note that you need to convert the type of the continuous variables in float format.

This dataset includes eight categorical variables:

The categorical variables are listed in CATE_FEATURES

- workclass
- education
- marital
- occupation
- relationship
- race
- sex
- native_country

moreover, six continuous variables:

The continuous variables are listed in CONTI_FEATURES

- age
- fnlwgt
- education_num
- capital_gain
- capital_loss

- hours_week

Note that we fill the list by hand so that you have a better idea of what columns we are using. A faster way to construct a list of categorical or continuous is to use:

```
## List Categorical
CATE_FEATURES =
df_train.iloc[:, :-1].select_dtypes('object').columns
print(CATE_FEATURES)

## List continuous
CONTI_FEATURES = df_train._get_numeric_data()
print(CONTI_FEATURES)
```

Here is the code to import the data:

```
# Import dataset
import pandas as pd

## Define path data
COLUMNS = ['age', 'workclass', 'fnlwgt', 'education',
'education_num', 'marital',
    'occupation', 'relationship', 'race', 'sex',
'capital_gain', 'capital_loss',
    'hours_week', 'native_country', 'label']
### Define continuous list
CONTI_FEATURES = ['age', 'fnlwgt', 'capital_gain', 'education_num',
'capital_loss', 'hours_week']
### Define categorical list
CATE_FEATURES = ['workclass', 'education', 'marital', 'occupation',
'relationship', 'race', 'sex', 'native_country']

## Prepare the data
features = ['age', 'workclass', 'fnlwgt', 'education',
'education_num', 'marital',
    'occupation', 'relationship', 'race', 'sex',
'capital_gain', 'capital_loss',
    'hours_week', 'native_country']

PATH = "https://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.data"
```

```

df_train = pd.read_csv(PATH, skipinitialspace=True, names =
COLUMNS, index_col=False)
df_train[CONTI_FEATURES]
=df_train[CONTI_FEATURES].astype('float64')
df_train.describe()

```

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_week
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	40.437456
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	12.347429
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

You can check the count of unique values of the native_country features. You can see that only one household comes from Holand-Netherlands. This household won't bring us any information, but will through an error during the training.

```
df_train.native_country.value_counts()
```

United-States	29170
Mexico	643
?	583
Philippines	198
Germany	137
Canada	121
Puerto-Rico	114
El-Salvador	106
India	100
Cuba	95
England	90
Jamaica	81
South	80
China	75
Italy	73
Dominican-Republic	70
Vietnam	67

Guatemala	64
Japan	62
Poland	60
Columbia	59
Taiwan	51
Haiti	44
Iran	43
Portugal	37
Nicaragua	34
Peru	31
France	29
Greece	29
Ecuador	28
Ireland	24
Hong	20
Cambodia	19
Trinidad&Tobago	19
Thailand	18
Laos	18
Yugoslavia	16
Outlying-US(Guam-USVI-etc)	14
Honduras	13
Hungary	13
Scotland	12
Holland-Netherlands	1

Name: native_country, dtype: int64

You can exclude this uninformative row from the dataset

```
## Drop Netherland, because only one row
df_train = df_train[df_train.native_country != "Holland-
Netherlands"]
```

Next, you store the position of the continuous features in a list. You will need it in the next step to build the pipeline.

The code below will loop over all columns names in CONTI_FEATURES and get its location (i.e., its number) and then append it to a list called conti_features

```
## Get the column index of the categorical features
conti_features = []
for i in CONTI_FEATURES:
```

```
    position = df_train.columns.get_loc(i)
    conti_features.append(position)
print(conti_features)
```

```
[0, 2, 10, 4, 11, 12]
```

The code below does the same job as above but for the categorical variable. The code below repeats what you have done previously, except with the categorical features.

```
## Get the column index of the categorical features
categorical_features = []
for i in CATE_FEATURES:
    position = df_train.columns.get_loc(i)
    categorical_features.append(position)
print(categorical_features)
```

```
[1, 3, 5, 6, 7, 8, 9, 13]
```

You can have a look at the dataset. Note that, each categorical feature is a string. You cannot feed a model with a string value. You need to transform the dataset using a dummy variable.

```
df_train.head(5)
```

In fact, you need to create one column for each group in the feature. First, you can run the code below to compute the total amount of columns needed.

```
print(df_train[CATE_FEATURES].nunique(),
      'There are', sum(df_train[CATE_FEATURES].nunique()), 'groups
in the whole dataset')
```

```
workclass          9
education         16
marital           7
occupation        15
relationship      6
race              5
sex               2
native_country    41
dtype: int64 There are 101 groups in the whole dataset
```

The whole dataset contains 101 groups as shown above. For instance, the features of workclass have nine groups. You can visualize the name of the groups with the following codes

unique() returns the unique values of the categorical features.

```
for i in CATE_FEATURES:  
    print(df_train[i].unique())
```

```
['State-gov' 'Self-emp-not-inc' 'Private' 'Federal-gov' 'Local-gov'  
'?'  
 'Self-emp-inc' 'Without-pay' 'Never-worked']  
['Bachelors' 'HS-grad' '11th' 'Masters' '9th' 'Some-college'  
'Assoc-acdm'  
 'Assoc-voc' '7th-8th' 'Doctorate' 'Prof-school' '5th-6th' '10th'  
 '1st-4th' 'Preschool' '12th']  
['Never-married' 'Married-civ-spouse' 'Divorced' 'Married-spouse-  
absent'  
 'Separated' 'Married-AF-spouse' 'Widowed']  
['Adm-clerical' 'Exec-managerial' 'Handlers-cleaners' 'Prof-  
specialty'  
 'Other-service' 'Sales' 'Craft-repair' 'Transport-moving'  
 'Farming-fishing' 'Machine-op-inspct' 'Tech-support' '?'  
 'Protective-serv' 'Armed-Forces' 'Priv-house-serv']  
['Not-in-family' 'Husband' 'Wife' 'Own-child' 'Unmarried' 'Other-  
relative']  
['White' 'Black' 'Asian-Pac-Islander' 'Amer-Indian-Eskimo' 'Other']  
['Male' 'Female']  
['United-States' 'Cuba' 'Jamaica' 'India' '?' 'Mexico' 'South'  
 'Puerto-Rico' 'Honduras' 'England' 'Canada' 'Germany' 'Iran'  
 'Philippines' 'Italy' 'Poland' 'Columbia' 'Cambodia' 'Thailand'  
 'Ecuador'  
 'Laos' 'Taiwan' 'Haiti' 'Portugal' 'Dominican-Republic' 'El-  
Salvador'  
 'France' 'Guatemala' 'China' 'Japan' 'Yugoslavia' 'Peru'  
 'Outlying-US(Guam-USVI-etc)' 'Scotland' 'Trinadad&Tobago' 'Greece'  
 'Nicaragua' 'Vietnam' 'Hong' 'Ireland' 'Hungary']
```

Therefore, the training dataset will contain $101 + 7$ columns. The last seven columns are the continuous features.

Scikit-learn can take care of the conversion. It is done in two steps:

- First, you need to convert the string to ID. For instance, State-gov will have the ID 1, Self-emp-not-inc ID 2 and so on. The function LabelEncoder does this for you
- Transpose each ID into a new column. As mentioned before, the dataset has 101 group's ID. Therefore there will be 101 columns capturing all categoricals features' groups. Scikit-learn has a function called OneHotEncoder that performs this operation

Step 2) Create the train/test set

Now that the dataset is ready, we can split it 80/20. 80 percent for the training set and 20 percent for the test set.

You can use `train_test_split`. The first argument is the dataframe is the features and the second argument is the label dataframe. You can specify the size of the test set with `test_size`.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(df_train[features],
                  df_train.label,
                  test_size =
0.2,
                  random_state=0)
X_train.head(5)
print(X_train.shape, X_test.shape)
```

(26048, 14) (6512, 14)

Step 3) Build the pipeline

The pipeline makes it easier to feed the model with consistent data. The idea behind is to put the raw data into a 'pipeline' to perform operations. For instance, with the current dataset, you need to standardize the continuous variables and convert the categorical data. Note that you can perform any operation inside the pipeline. For instance, if you have 'NA's' in the dataset, you can replace them by the mean or median. You can also create new variables.

You have the choice; hard code the two processes or create a pipeline. The first choice can lead to data leakage and create inconsistencies over time. A better option is to use the pipeline.

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
```

The pipeline will perform two operations before feeding the logistic classifier:

1. Standardize the variable: `StandardScaler()`
2. Convert the categorical features: OneHotEncoder(sparse=False)

You can perform the two steps using the `make_column_transformer`. This function is not available in the current version of scikit-learn (0.19). It is not possible with the current version to perform the label encoder and one hot encoder in the pipeline. It is one reason we decided to use the developer version.

`make_column_transformer` is easy to use. You need to define which columns to apply the transformation and what transformation to operate. For instance, to standardize the continuous feature, you can do:

- conti_features, StandardScaler() inside make_column_transformer.
 - conti_features: list with the continuous variable
 - StandardScaler: standardize the variable

The object OneHotEncoder inside make_column_transformer automatically encodes the label.

```
preprocess = make_column_transformer(
    (conti_features, StandardScaler()),
    ### Need to be numeric not string to specify columns name
    (categorical_features, OneHotEncoder(sparse=False))
)
```

You can test if the pipeline works with fit_transform. The dataset should have the following shape: 26048, 107

```
preprocess.fit_transform(X_train).shape
(26048, 107)
```

The data transformer is ready to use. You can create the pipeline with make_pipeline. Once the data are transformed, you can feed the logistic regression.

```
model = make_pipeline(
    preprocess,
    LogisticRegression())
```

Training a model with scikit-learn is trivial. You need to use the object fit preceded by the pipeline, i.e., model. You can print the accuracy with the score object from the scikit-learn library

```
model.fit(X_train, y_train)
print("logistic regression score: %f" % model.score(X_test,
y_test))
```

```
logistic regression score: 0.850891
```

Finally, you can predict the classes with predict_proba. It returns the probability

for each class. Note that it sums to one.

```
model.predict_proba(X_test)

array([[0.83576663, 0.16423337],
       [0.94582765, 0.05417235],
       [0.64760587, 0.35239413],
       ...,
       [0.99639252, 0.00360748],
       [0.02072181, 0.97927819],
       [0.56781353, 0.43218647]])
```

Step 4) Using our pipeline in a grid search

Tune the hyperparameter (variables that determine network structure like hidden units) can be tedious and exhausting. One way to evaluate the model could be to change the size of the training set and evaluate the performances. You can repeat this method ten times to see the score metrics. However, it is too much work.

Instead, scikit-learn provides a function to carry out parameter tuning and cross-validation.

Cross-validation

Cross-Validation means during the training, the training set is split n number of times in folds and then evaluates the model n times. For instance, if cv is set to 10, the training set is trained and evaluated ten times. At each round, the classifier chooses randomly nine fold to train the model, and the 10th fold is meant for evaluation.

Grid search Each classifier has hyperparameters to tune. You can try different values, or you can set a parameter grid. If you go to the scikit-learn official website, you can see the logistic classifier has different parameters to tune. To make the training faster, you choose to tune the C parameter. It controls for the regularization parameter. It should be positive. A small value gives more weight to the regularizer.

You can use the object `GridSearchCV`. You need to create a dictionary containing the hyperparameters to tune.

You list the hyperparameters followed by the values you want to try. For instance, to tune the C parameter, you use:

- 'logisticregression__C': [0.1, 1.0, 10.0]: The parameter is preceded by the

name, in lower case, of the classifier and two underscores.

The model will try four different values: 0.001, 0.01, 0.1 and 1.

You train the model using 10 folds: cv=10

```
from sklearn.model_selection import GridSearchCV
# Construct the parameter grid
param_grid = {
    'logisticregression__C': [0.001, 0.01, 0.1, 1.0],
```

You can train the model using GridSearchCV with the parameter gri and cv.

```
# Train the model
grid_clf = GridSearchCV(model,
                        param_grid,
                        cv=10,
                        iid=False)
grid_clf.fit(X_train, y_train)
```

OUTPUT

```
GridSearchCV(cv=10, error_score='raise-deprecating',
            estimator=Pipeline(memory=None,
                               steps=[('columntransformer', ColumnTransformer(n_jobs=1,
                                                                           remainder='drop', transformer_weights=None,
                                                                           transformers=[('standardscaler', StandardScaler(copy=True,
                                                                                           with_mean=True, with_std=True), [0, 2, 10, 4, 11, 12]),
                                                                 ('onehotencoder', OneHotEncoder(categorical_features=None,
                                                                                           categories=None, ...ty='l2', random_state=None, solver='liblinear',
                                                                                           tol=0.0001,
                                                                                           verbose=0, warm_start=False))]),
                               fit_params=None, iid=False, n_jobs=1,
                               param_grid={'logisticregression__C': [0.001, 0.01, 0.1,
                                                                      1.0]}, pre_dispatch='2*n_jobs', refit=True,
                               return_train_score='warn',
                               scoring=None, verbose=0)
```

To access the best parameters, you use best_params_

```
grid_clf.best_params_
```

OUTPUT

```
{'logisticregression__C': 1.0}
```

After trained the model with four different regularization values, the optimal parameter is

```
print("best logistic regression from grid search: %f" %  
grid_clf.best_estimator_.score(X_test, y_test))
```

best logistic regression from grid search: 0.850891

To access the predicted probabilities:

```
grid_clf.best_estimator_.predict_proba(X_test)
```

```
array([[0.83576677, 0.16423323],  
       [0.9458291 , 0.0541709 ],  
       [0.64760416, 0.35239584],  
       ...,  
       [0.99639224, 0.00360776],  
       [0.02072033, 0.97927967],  
       [0.56782222, 0.43217778]])
```

XGBoost Model with scikit-learn

Let's try to train one of the best classifiers on the market. XGBoost is an improvement over the random forest. The theoretical background of the classifier out of the scope of this tutorial. Keep in mind that, XGBoost has won lots of kaggle competitions. With an average dataset size, it can perform as good as a deep learning algorithm or even better.

The classifier is challenging to train because it has a high number of parameters to tune. You can, of course, use GridSearchCV to choose the parameter for you.

Instead, let's see how to use a better way to find the optimal parameters. GridSearchCV can be tedious and very long to train if you pass many values. The search space grows along with the number of parameters. A preferable solution is to use RandomizedSearchCV. This method consists of choosing the values of each hyperparameter after each iteration randomly. For instance, if the classifier is trained over 1000 iterations, then 1000 combinations are evaluated. It works more or less like. GridSearchCV

You need to import xgboost. If the library is not installed, please use pip3 install xgboost or

```
use import sys  
!{sys.executable} -m pip install xgboost
```

In Jupyter environment

Next,

```
import xgboost  
from sklearn.model_selection import RandomizedSearchCV  
from sklearn.model_selection import StratifiedKFold
```

The next step includes specifying the parameters to tune. You can refer to the

official documentation to see all the parameters to tune. For the sake of the tutorial, you only choose two hyperparameters with two values each. XGBoost takes lots of time to train, the more hyperparameters in the grid, the longer time you need to wait.

```
params = {  
    'xgbclassifier__gamma': [0.5, 1],  
    'xgbclassifier__max_depth': [3, 4]  
}
```

You construct a new pipeline with XGBoost classifier. You choose to define 600 estimators. Note that n_estimators are a parameter that you can tune. A high value can lead to overfitting. You can try by yourself different values but be aware it can takes hours. You use the default value for the other parameters

```
model_xgb = make_pipeline(  
    preprocess,  
    xgboost.XGBClassifier(  
        n_estimators=600,  
        objective='binary:logistic',  
        silent=True,  
        nthread=1)  
)
```

You can improve the cross-validation with the Stratified K-Folds cross-validator. You construct only three folds here to faster the computation but lowering the quality. Increase this value to 5 or 10 at home to improve the results.

You choose to train the model over four iterations.

```
skf = StratifiedKFold(n_splits=3,  
                      shuffle = True,  
                      random_state = 1001)  
  
random_search = RandomizedSearchCV(model_xgb,  
                                    param_distributions=params,  
                                    n_iter=4,  
                                    scoring='accuracy',  
                                    n_jobs=4,
```

```
cv=skf.split(X_train, y_train),  
verbose=3,  
random_state=1001)
```

The randomized search is ready to use, you can train the model

```
#grid_xgb = GridSearchCV(model_xgb, params, cv=10, iid=False)  
random_search.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits  
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=0.5  
.....  
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=0.5  
.....  
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=0.5  
.....  
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=0.5  
.....  
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=0.5,  
score=0.8759645283888057, total= 1.0min  
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=0.5  
.....  
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=0.5,  
score=0.8729701715996775, total= 1.0min  
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=0.5,  
score=0.8706519235199263, total= 1.0min  
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=0.5  
.....  
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=1  
.....  
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=0.5,  
score=0.8735460094437406, total= 1.3min  
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=1  
.....  
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=1,  
score=0.8722791661868018, total= 57.7s  
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=1  
.....  
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=1,  
score=0.8753886905447426, total= 1.0min  
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=1  
.....  
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=0.5,
```

```
score=0.8697304768486523, total= 1.3min
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=1
.....
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=0.5,
score=0.8740066797189912, total= 1.4min
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=1
.....
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=1,
score=0.8707671043538355, total= 1.0min
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=1,
score=0.8729701715996775, total= 1.2min
[Parallel(n_jobs=4)]: Done 10 out of 12 | elapsed: 3.6min
remaining: 43.5s
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=1,
score=0.8736611770125533, total= 1.2min
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=1,
score=0.8692697535130154, total= 1.2min
```

```
[Parallel(n_jobs=4)]: Done 12 out of 12 | elapsed: 3.6min
finished
/Users/Thomas/anaconda3/envs/hello-tf/lib/python3.6/site-
packages/sklearn/model_selection/_search.py:737:
DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24.
This will change numeric results when test-set sizes are unequal.
DeprecationWarning)
```

```
RandomizedSearchCV(cv=<generator object _BaseKFold.split at
0x1101eb830>,
    error_score='raise-deprecating',
    estimator=Pipeline(memory=None,
        steps=[('columntransformer', ColumnTransformer(n_jobs=1,
remainder='drop', transformer_weights=None,
            transformers=[('standardscaler', StandardScaler(copy=True,
with_mean=True, with_std=True), [0, 2, 10, 4, 11, 12]),
('onehotencoder', OneHotEncoder(categorical_features=None,
categories=None, ...
            reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
silent=True, subsample=1))]),
        fit_params=None, iid='warn', n_iter=4, n_jobs=4,
        param_distributions={'xgbclassifier__gamma': [0.5, 1],
'xgbclassifier__max_depth': [3, 4]},
        pre_dispatch='2*n_jobs', random_state=1001, refit=True,
        return_train_score='warn', scoring='accuracy', verbose=3)
```

As you can see, XGBoost has a better score than the previous logistic regression.

```
print("Best parameter", random_search.best_params_)
print("best logistic regression from grid search: %f" %
random_search.best_estimator_.score(X_test, y_test))
```

```
Best parameter {'xgbclassifier__max_depth': 3,
'xgbclassifier__gamma': 0.5}
best logistic regression from grid search: 0.873157
```

```
random_search.best_estimator_.predict(X_test)
```

```
array(['<=50K', '<=50K', '<=50K', ..., '<=50K', '>50K', '<=50K'],
dtype=object)
```

Create DNN with MLPClassifier in scikit-learn

Finally, you can train a deep learning algorithm with scikit-learn. The method is the same as the other classifier. The classifier is available at MLPClassifier.

```
from sklearn.neural_network import MLPClassifier
```

You define the following deep learning algorithm:

- Adam solver
- Relu activation function
- Alpha = 0.0001
- batch size of 150
- Two hidden layers with 100 and 50 neurons respectively

```
model_dnn = make_pipeline(  
    preprocess,  
    MLPClassifier(solver='adam',  
                  alpha=0.0001,  
                  activation='relu',  
                  batch_size=150,  
                  hidden_layer_sizes=(200, 100),  
                  random_state=1))
```

You can change the number of layers to improve the model

```
model_dnn.fit(X_train, y_train)  
print("DNN regression score: %f" % model_dnn.score(X_test,  
y_test))
```

DNN regression score: 0.821253

LIME: Trust your Model

Now that you have a good model, you need a tool to trust it. Machine learning algorithm, especially random forest and neural network, are known to be black-box algorithm. Say differently, it works but no one knows why.

Three researchers have come up with a great tool to see how the computer makes a prediction. The paper is called Why Should I Trust You?

They developed an algorithm named **Local Interpretable Model-Agnostic Explanations (LIME)**.

Take an example:

sometimes you do not know if you can trust a machine-learning prediction:

A doctor, for example, cannot trust a diagnosis just because a computer said so. You also need to know if you can trust the model before putting it into production.

Imagine we can understand why any classifier is making a prediction even incredibly complicated models such as neural networks, random forests or svms with any kernel

will become more accessible to trust a prediction if we can understand the reasons behind it. From the example with the doctor, if the model told him which symptoms are essential you would trust it, it is also easier to figure out if you should not trust the model.

Lime can tell you what features affect the decisions of the classifier

Data Preparation

They are a couple of things you need to change to run LIME with python. First of all, you need to install lime in the terminal. You can use pip install lime

Lime makes use of LimeTabularExplainer object to approximate the model locally. This object requires:

- a dataset in numpy format
- The name of the features: feature_names
- The name of the classes: class_names
- The index of the column of the categorical features: categorical_features
- The name of the group for each categorical features: categorical_names

Create numpy train set

You can copy and convert df_train from pandas to numpy very easily

```
df_train.head(5)
# Create numpy data
df_lime = df_train
df_lime.head(3)
```

Get the class name The label is accessible with the object unique(). You should see:

- '<=50K'
- '>50K'

```
# Get the class name
class_names = df_lime.label.unique()
class_names
```

```
array(['<=50K', '>50K'], dtype=object)
```

index of the column of the categorical features

You can use the method you lean before to get the name of the group. You

encode the label with LabelEncoder. You repeat the operation on all the categorical features.

```
##  
import sklearn.preprocessing as preprocessing  
categorical_names = {}  
for feature in CATE_FEATURES:  
    le = preprocessing.LabelEncoder()  
    le.fit(df_lime[feature])  
    df_lime[feature] = le.transform(df_lime[feature])  
    categorical_names[feature] = le.classes_  
print(categorical_names)
```



```
{'workclass': array(['?', 'Federal-gov', 'Local-gov', 'Never-worked', 'Private',  
       'Self-emp-inc', 'Self-emp-not-inc', 'State-gov', 'Without-pay'],  
      dtype=object), 'education': array(['10th', '11th', '12th',  
       '1st-4th', '5th-6th', '7th-8th', '9th',  
       'Assoc-acdm', 'Assoc-voc', 'Bachelors', 'Doctorate', 'HS-grad',  
       'Masters', 'Preschool', 'Prof-school', 'Some-college'],  
      dtype=object), 'marital': array(['Divorced', 'Married-AF-spouse',  
       'Married-civ-spouse',  
       'Married-spouse-absent', 'Never-married', 'Separated',  
       'Widowed'],  
      dtype=object), 'occupation': array(['?', 'Adm-clerical',  
       'Armed-Forces', 'Craft-repair',  
       'Exec-managerial', 'Farming-fishing', 'Handlers-cleaners',  
       'Machine-op-inspct', 'Other-service', 'Priv-house-serv',  
       'Prof-specialty', 'Protective-serv', 'Sales', 'Tech-support',  
       'Transport-moving'], dtype=object), 'relationship':  
array(['Husband', 'Not-in-family', 'Other-relative', 'Own-child',  
       'Unmarried', 'Wife'], dtype=object), 'race': array(['Amer-Indian-Eskimo', 'Asian-Pac-Islander', 'Black', 'Other',  
       'White'], dtype=object), 'sex': array(['Female', 'Male'],  
      dtype=object), 'native_country': array(['?', 'Cambodia', 'Canada',  
       'China', 'Columbia', 'Cuba',  
       'Dominican-Republic', 'Ecuador', 'El-Salvador', 'England',  
       'France', 'Germany', 'Greece', 'Guatemala', 'Haiti',  
       'Honduras',  
       'Hong', 'Hungary', 'India', 'Iran', 'Ireland', 'Italy',  
       'Jamaica'],
```

```
'Japan', 'Laos', 'Mexico', 'Nicaragua',
'Outlying-US(Guam-USVI-etc)', 'Peru', 'Philippines',
'Poland',
'Portugal', 'Puerto-Rico', 'Scotland', 'South', 'Taiwan',
'Thailand', 'Trinidad&Tobago', 'United-States', 'Vietnam',
'Yugoslavia'], dtype=object)}
```

```
df_lime.dtypes
```

age	float64
workclass	int64
fnlwgt	float64
education	int64
education_num	float64
marital	int64
occupation	int64
relationship	int64
race	int64
sex	int64
capital_gain	float64
capital_loss	float64
hours_week	float64
native_country	int64
label	object
dtype:	object

Now that the dataset is ready, you can construct the different dataset. You actually transform the data outside of the pipeline in order to avoid errors with LIME. The training set in the LimeTabularExplainer should be a numpy array without string. With the method above, you have a training dataset already converted.

```
from sklearn.model_selection import train_test_split
X_train_lime, X_test_lime, y_train_lime, y_test_lime =
train_test_split(df_lime[features],
                  df_lime.label,
                  test_size =
0.2,
                  random_state=0)
X_train_lime.head(5)
```

You can make the pipeline with the optimal parameters from XGBoost

```
model_xgb = make_pipeline(
    preprocess,
    xgboost.XGBClassifier(max_depth = 3,
                          gamma = 0.5,
                          n_estimators=600,
                          objective='binary:logistic',
                          silent=True,
                          nthread=1))

model_xgb.fit(X_train_lime, y_train_lime)
```

```
/Users/Thomas/anaconda3/envs/hello-tf/lib/python3.6/site-
packages/sklearn/preprocessing/_encoders.py:351: FutureWarning: The
handling of integer data will change in version 0.22. Currently,
the categories are determined based on the range [0, max(values)],
while in the future they will be determined based on the unique
values.
```

If you want the future behavior and silence this warning, you can specify "categories='auto'." In case you used a LabelEncoder before this OneHotEncoder to convert the categories to integers, then you can now use the OneHotEncoder directly.

```
warnings.warn(msg, FutureWarning)
```

```
Pipeline(memory=None,
         steps=[('columntransformer', ColumnTransformer(n_jobs=1,
                                                       remainder='drop',
                                                       transformer_weights=None,
                                                       transformers=[('standardscaler', StandardScaler(copy=True,
                                                               with_mean=True, with_std=True), [0, 2, 10, 4, 11, 12]),
                                                               ('onehotencoder', OneHotEncoder(categorical_features=None,
                                                               categories=None, ...),
                                                               reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                                                               silent=True, subsample=1)])])
```

You get a warning. The warning explains that you do not need to create a label encoder before the pipeline. If you do not want to use LIME, you are fine to use the method from the first part of the tutorial. Otherwise, you can keep with this method, first create an encoded dataset, set get the hot one encoder within the pipeline.

```
print("best logistic regression from grid search: %f" %
model_xgb.score(X_test_lime, y_test_lime))
```

```
best logistic regression from grid search: 0.873157
```

```
model_xgb.predict_proba(X_test_lime)

array([[7.9646105e-01, 2.0353897e-01],
       [9.5173013e-01, 4.8269872e-02],
       [7.9344827e-01, 2.0655173e-01],
       ...,
       [9.9031430e-01, 9.6856682e-03],
       [6.4581633e-04, 9.9935418e-01],
       [9.7104281e-01, 2.8957171e-02]], dtype=float32)
```

Before to use LIME in action, let's create a numpy array with the features of the wrong classification. You can use that list later to get an idea about what mislead the classifier.

```
temp = pd.concat([X_test_lime, y_test_lime], axis= 1)
temp['predicted'] = model_xgb.predict(X_test_lime)
temp['wrong']= temp['label'] != temp['predicted']
temp = temp.query('wrong==True').drop('wrong', axis=1)
temp= temp.sort_values(by=['label'])
temp.shape
```

(826, 16)

You create a lambda function to retrieve the prediction from the model with the new data. You will need it soon.

```
predict_fn = lambda x: model_xgb.predict_proba(x).astype(float)
X_test_lime.dtypes
```

age	float64
workclass	int64
fnlwgt	float64
education	int64
education_num	float64
marital	int64
occupation	int64
relationship	int64
race	int64
sex	int64
capital_gain	float64
capital_loss	float64
hours_week	float64

```
native_country      int64  
dtype: object
```

```
predict_fn(X_test_lime)
```

```
array([[7.96461046e-01,  2.03538969e-01],  
       [9.51730132e-01,  4.82698716e-02],  
       [7.93448269e-01,  2.06551731e-01],  
       ...,  
       [9.90314305e-01,  9.68566816e-03],  
       [6.45816326e-04,  9.99354184e-01],  
       [9.71042812e-01,  2.89571714e-02]])
```

You convert the pandas dataframe to numpy array

```
X_train_lime = X_train_lime.values  
X_test_lime = X_test_lime.values  
X_test_lime
```

```
array([[4.00000e+01,  5.00000e+00,  1.93524e+05,  ...,  0.00000e+00,  
       4.00000e+01,  3.80000e+01],  
       [2.70000e+01,  4.00000e+00,  2.16481e+05,  ...,  0.00000e+00,  
       4.00000e+01,  3.80000e+01],  
       [2.50000e+01,  4.00000e+00,  2.56263e+05,  ...,  0.00000e+00,  
       4.00000e+01,  3.80000e+01],  
       ...,  
       [2.80000e+01,  6.00000e+00,  2.11032e+05,  ...,  0.00000e+00,  
       4.00000e+01,  2.50000e+01],  
       [4.40000e+01,  4.00000e+00,  1.67005e+05,  ...,  0.00000e+00,  
       6.00000e+01,  3.80000e+01],  
       [5.30000e+01,  4.00000e+00,  2.57940e+05,  ...,  0.00000e+00,  
       4.00000e+01,  3.80000e+01]])
```

```
model_xgb.predict_proba(X_test_lime)
```

```
array([[7.9646105e-01,  2.0353897e-01],  
       [9.5173013e-01,  4.8269872e-02],  
       [7.9344827e-01,  2.0655173e-01],  
       ...,  
       [9.9031430e-01,  9.6856682e-03],  
       [6.4581633e-04,  9.9935418e-01],  
       [9.7104281e-01,  2.8957171e-02]], dtype=float32)
```

```
print(features,  
      class_names,
```

```
categorical_features,  
categorical_names)
```

```
['age', 'workclass', 'fnlwgt', 'education', 'education_num',  
'marital', 'occupation', 'relationship', 'race', 'sex',  
'capital_gain', 'capital_loss', 'hours_week', 'native_country']  
['<=50K' '>50K'] [1, 3, 5, 6, 7, 8, 9, 13] {'workclass':  
array(['?', 'Federal-gov', 'Local-gov', 'Never-worked', 'Private',  
'Self-emp-inc', 'Self-emp-not-inc', 'State-gov', 'Without-  
pay'],  
      dtype=object), 'education': array(['10th', '11th', '12th',  
'1st-4th', '5th-6th', '7th-8th', '9th',  
      'Assoc-acdm', 'Assoc-voc', 'Bachelors', 'Doctorate', 'HS-  
grad',  
      'Masters', 'Preschool', 'Prof-school', 'Some-college'],  
      dtype=object), 'marital': array(['Divorced', 'Married-AF-  
spouse', 'Married-civ-spouse',  
      'Married-spouse-absent', 'Never-married', 'Separated',  
'Widowed'],  
      dtype=object), 'occupation': array(['?', 'Adm-clerical',  
'Armed-Forces', 'Craft-repair',  
      'Exec-managerial', 'Farming-fishing', 'Handlers-cleaners',  
      'Machine-op-inspct', 'Other-service', 'Priv-house-serv',  
      'Prof-specialty', 'Protective-serv', 'Sales', 'Tech-  
support',  
      'Transport-moving'], dtype=object), 'relationship':  
array(['Husband', 'Not-in-family', 'Other-relative', 'Own-child',  
      'Unmarried', 'Wife'], dtype=object), 'race': array(['Amer-  
Indian-Eskimo', 'Asian-Pac-Islander', 'Black', 'Other',  
      'White'], dtype=object), 'sex': array(['Female', 'Male'],  
dtype=object), 'native_country': array(['?', 'Cambodia', 'Canada',  
'China', 'Columbia', 'Cuba',  
      'Dominican-Republic', 'Ecuador', 'El-Salvador', 'England',  
      'France', 'Germany', 'Greece', 'Guatemala', 'Haiti',  
'Honduras',  
      'Hong', 'Hungary', 'India', 'Iran', 'Ireland', 'Italy',  
'Jamaica',  
      'Japan', 'Laos', 'Mexico', 'Nicaragua',  
      'Outlying-US(Guam-USVI-etc)', 'Peru', 'Philippines',  
'Poland',  
      'Portugal', 'Puerto-Rico', 'Scotland', 'South', 'Taiwan',  
      'Thailand', 'Trinadad&Tobago', 'United-States', 'Vietnam',  
      'Yugoslavia'], dtype=object)})
```

```
import lime
```

```
import lime.lime_tabular
### Train should be label encoded not one hot encoded
explainer = lime.lime_tabular.LimeTabularExplainer(X_train_lime ,
                                                    feature_names =
features,
class_names=class_names,
categorical_features=categorical_features,
categorical_names=categorical_names,
                                                    kernel_width=3)
```

Lets choose a random household from the test set and see the model prediction and how the computer made his choice.

```
import numpy as np
np.random.seed(1)
i = 100
print(y_test_lime.iloc[i])
>50K
```



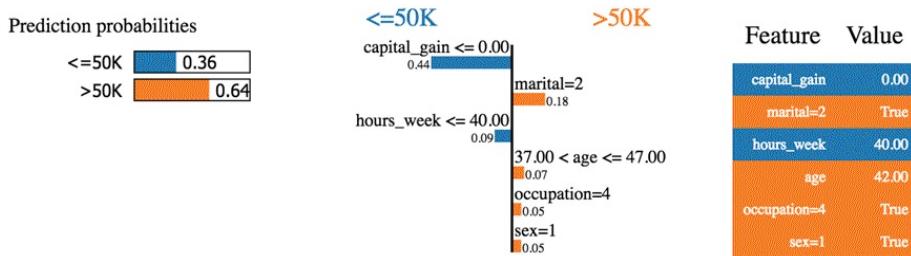
```
X_test_lime[i]
```



```
array([4.20000e+01, 4.00000e+00, 1.76286e+05, 7.00000e+00,
1.20000e+01,
       2.00000e+00, 4.00000e+00, 0.00000e+00, 4.00000e+00,
1.00000e+00,
       0.00000e+00, 0.00000e+00, 4.00000e+01, 3.80000e+01])
```

You can use the explainer with explain_instance to check the explanation behind the model

```
exp = explainer.explain_instance(X_test_lime[i], predict_fn,
num_features=6)
exp.show_in_notebook(show_all=False)
```



We can see that the classifier predicted the household correctly. The income is, indeed, above 50k.

The first thing we can say is the classifier is not that sure about the predicted probabilities. The machine predicts the household has an income over 50k with a probability of 64%. This 64% is made up of Capital gain and marital. The blue color contributes negatively to the positive class and the orange line, positively.

The classifier is confused because the capital gain of this household is null, while the capital gain is usually a good predictor of wealth. Besides, the household works less than 40 hours per week. Age, occupation, and sex contribute positively to the classifier.

If the marital status were single, the classifier would have predicted an income below 50k ($0.64 - 0.18 = 0.46$)

We can try with another household which has been wrongly classified

```
temp.head(3)
temp.iloc[1, :-2]
```

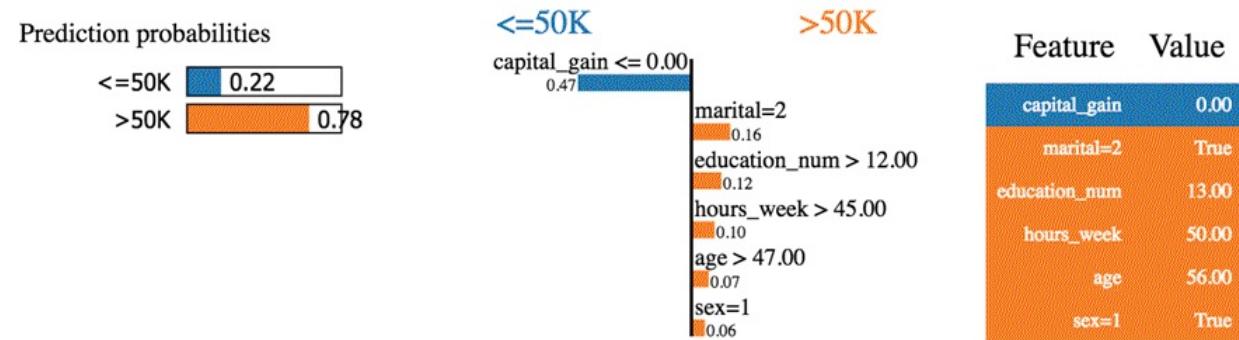
age	58
workclass	4
fnlwgt	68624
education	11
education_num	9
marital	2
occupation	4
relationship	0
race	4
sex	1

```
capital_gain      0
capital_loss      0
hours_week        45
native_country    38
Name: 20931, dtype: object
```

```
i = 1
print('This observation is', temp.iloc[i,-2:])
```

```
This observation is label      <=50K
predicted          >50K
Name: 20931, dtype: object
```

```
exp = explainer.explain_instance(temp.iloc[1,:-2], predict_fn,
num_features=6)
exp.show_in_notebook(show_all=False)
```



The classifier predicted an income below 50k while it is untrue. This household seems odd. It does not have a capital gain, nor capital loss. He is divorced and is 60 years old, and it is an educated people, i.e., `education_num > 12`. According to the overall pattern, this household should, like explain by the classifier, get an income below 50k.

You try to play around with LIME. You will notice gross mistakes from the classifier.

You can check the GitHub of the owner of the library. They provide extra documentation for image and text classification.

Summary

Below is a list of some useful command with scikit learn version >=0.20

create train/test dataset	trainees split
Build a pipeline	
select the column and apply the transformation	makecolumntransformer
type of transformation	
standardize	StandardScaler
min max	MinMaxScaler
Normalize	Normalizer
Impute missing value	Imputer
Convert categorical	OneHotEncoder
Fit and transform the data	fit_transform
Make the pipeline	make_pipeline
Basic model	
logistic regression	LogisticRegression
XGBoost	XGBClassifier

Neural net	MLPClassifier
Grid search	GridSearchCV
Randomized search	RandomizedSearchCV

Chapter 13: Linear Regression

Linear regression

In this tutorial, you will learn basic principles of linear regression and machine learning in general.

TensorFlow provides tools to have full control of the computations. This is done with the low-level API. On top of that, TensorFlow is equipped with a vast array of APIs to perform many machine learning algorithms. This is the high-level API. TensorFlow calls them estimators

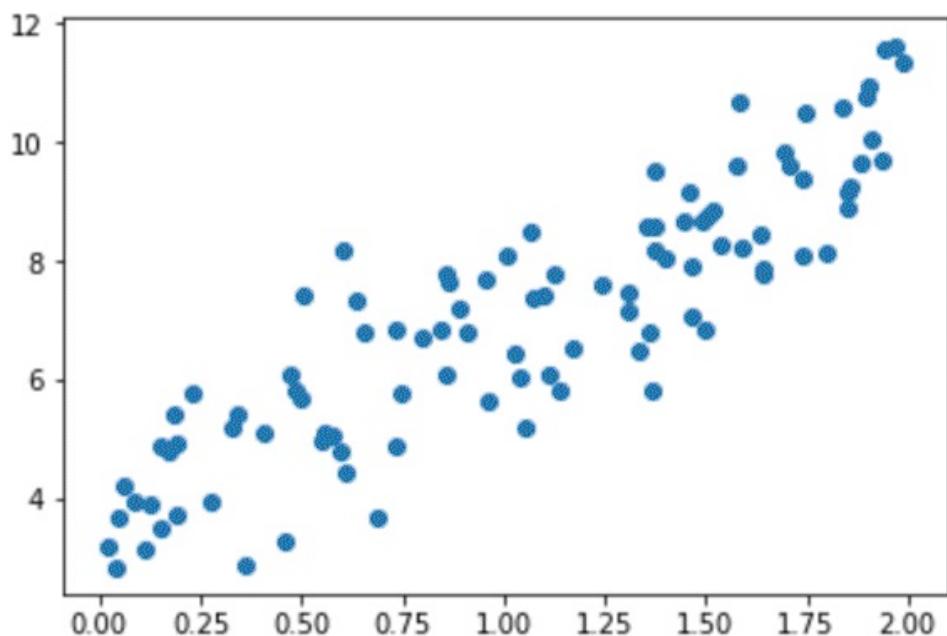
- Low-level API: Build the architecture, optimization of the model from scratch. It is complicated for a beginner
- High-level API: Define the algorithm. It is easier-friendly. TensorFlow provides a toolbox called **estimator** to construct, train, evaluate and make a prediction.

In this tutorial, you will use the **estimators only**. The computations are faster and are easier to implement. The first part of the tutorial explains how to use the gradient descent optimizer to train a linear regression. In a second part, you will use the Boston dataset to predict the price of a house using TensorFlow estimator.

How to train a linear regression model

Before we begin to train the model, let's have look at what is a linear regression.

Imagine you have two variables, x and y and your task is to predict the value of knowing the value of . If you plot the data, you can see a positive relationship between your independent variable, x and your dependent variable y.



You may observe, if $x=1$, y will roughly be equal to 6 and if $x=2$, y will be around 8.5.

This is not a very accurate method and prone to error, especially with a dataset with hundreds of thousands of points.

A linear regression is evaluated with an equation. The variable y is explained by one or many covariates. In your example, there is only one dependent variable. If you have to write this equation, it will be:

$$y = \beta + \alpha X + \epsilon$$

With:

- β is the bias. i.e. if $x=0$, $y=\beta$
- α is the weight associated to x
- ϵ is the residual or the error of the model. It includes what the model cannot learn from the data

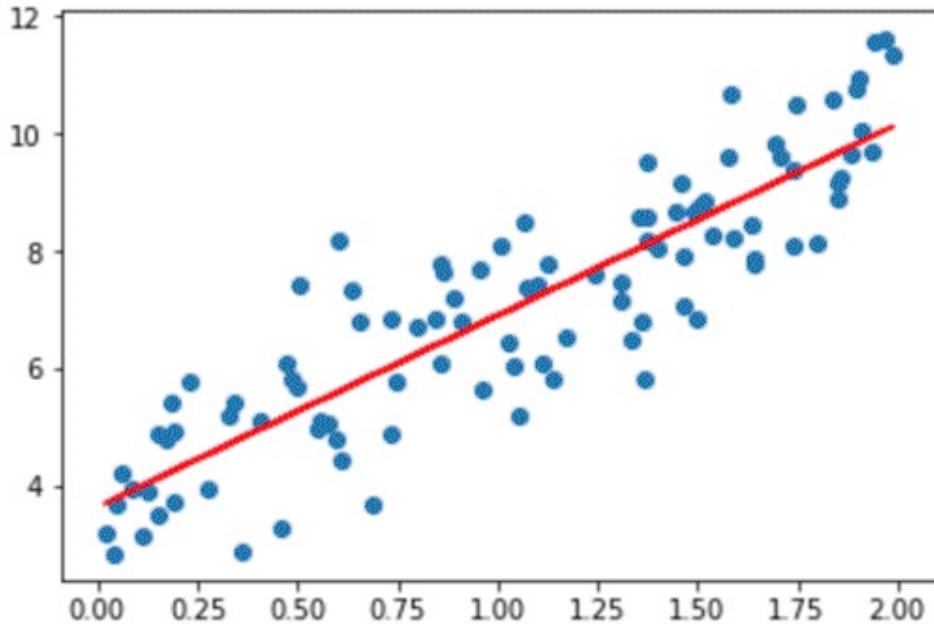
Imagine you fit the model and you find the following solution for:

- $\beta = 3.8$
- $\alpha = 2.78$

You can substitute those numbers in the equation and it becomes:

$$y = 3.8 + 2.78x$$

You have now a better way to find the values for y . That is, you can replace x with any value you want to predict y . In the image below, we have replace x in the equation with all the values in the dataset and plot the result.



The red line represents the fitted value, that is the values of y for each value of x. You don't need to see the value of x to predict y, for each x there is any which belongs to the red line. You can also predict for values of x higher than 2!

If you want to extend the linear regression to more covariates, you can by adding more variables to the model. The difference between traditional analysis and linear regression is the linear regression looks at how y will react for each variable x taken independently.

Let's see an example. Imagine you want to predict the sales of an ice cream shop. The dataset contains different information such as the weather (i.e rainy, sunny, cloudy), customer informations (i.e salary, gender, marital status).

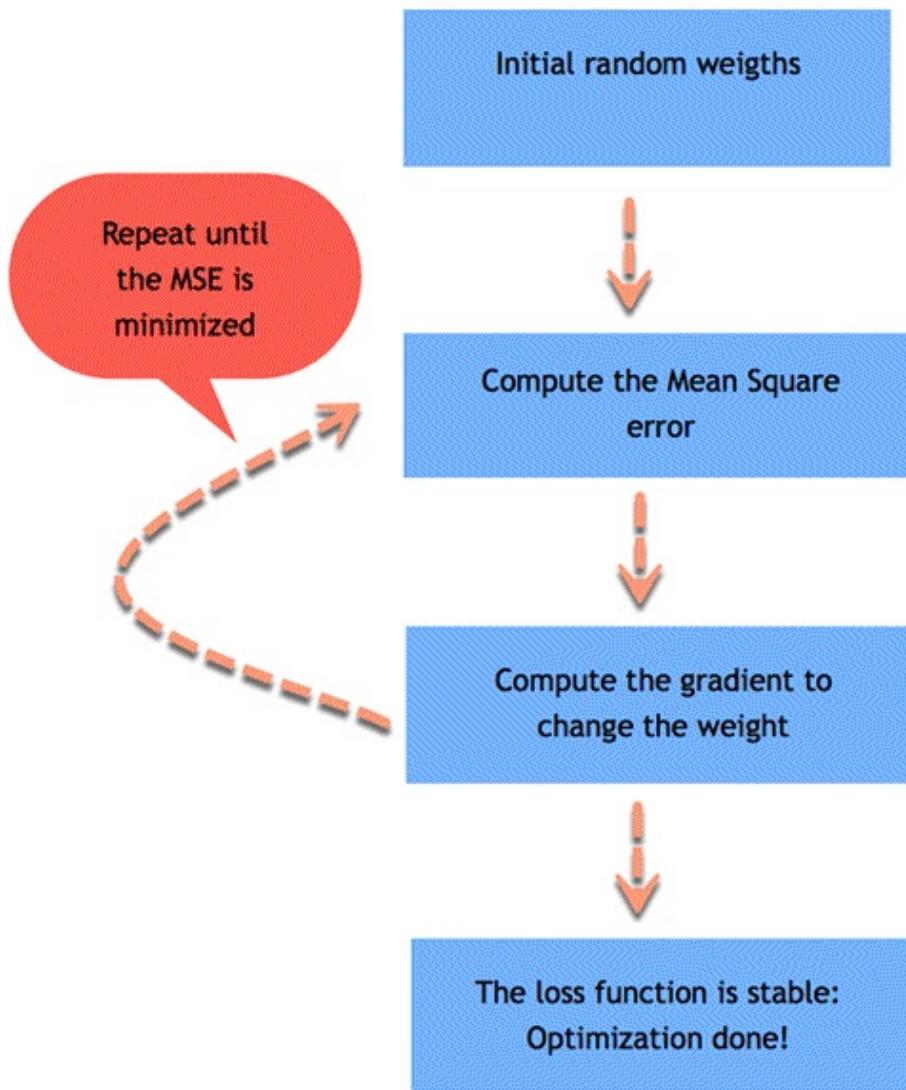
Traditional analysis will try to predict the sale by let's say computing the average for each variable and try to estimate the sale for different scenarios. It will lead to poor predictions and restrict the analysis to the chosen scenario.

If you use linear regression, you can write this equation:

$$Sales = \beta + \alpha_1 \text{weather} + \alpha_2 \text{salary} + \alpha_3 \text{gender} + \alpha_4 \text{marital} + \epsilon$$

The algorithm will find the best solution for the weights; it means it will try to minimize the cost (the difference between the fitted line and the data points).

How the algorithm works



The algorithm will choose a random number for each β and α and replace the value of x to get the predicted value of y . If the dataset has 100 observations, the algorithm computes 100 predicted values.

We can compute the error, noted ϵ of the model, which is the difference between the predicted value and the real value. A positive error means the model underestimates the prediction of y , and a negative error means the model overestimates the prediction of y .

$$\epsilon = y - y_{pred}$$

Your goal is to minimize the square of the error. The algorithm computes the mean of the square error. This step is called minimization of the error. For linear regression is the **Mean Square Error**, also called MSE. Mathematically, it is:

$$MSE(X) = \frac{1}{m} \sum_{i=1}^m (\theta^T x^i - y^i)^2$$

Where:

- θ^T is the weights so $\theta^T x^i$ refers to the predicted value
- y is the real values
- m is the number of observations

Note that θ^T means it uses the transpose of the matrices. The $\frac{1}{m} \sum$ is the mathematical notation of the mean.

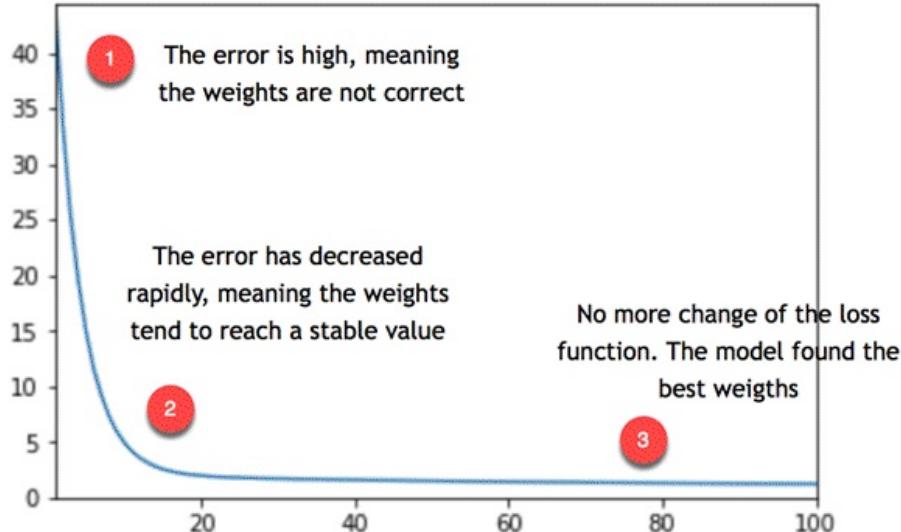
The goal is to find the best θ that minimize the MSE

If the average error is large, it means the model performs poorly and the weights are not chosen properly. To correct the weights, you need to use an optimizer. The traditional optimizer is called **Gradient Descent**.

The gradient descent takes the derivative and decreases or increases the weight. If the derivative is positive, the weight is decreased. If the derivative is negative,

the weight increases. The model will update the weights and recompute the error. This process is repeated until the error does not change anymore. Each process is called an **iteration**. Besides, the gradients are multiplied by a learning rate. It indicates the speed of the learning.

If the learning rate is too small, it will take very long time for the algorithm to converge (i.e requires lots of iterations). If the learning rate is too high, the algorithm might never converge.



You can see from the picture above, the model repeats the process about 20 times before to find a stable value for the weights, therefore reaching the lowest error.

Note that, the error is not equal to zero but stabilizes around 5. It means, the model makes a typical error of 5. If you want to reduce the error, you need to add more information to the model such as more variables or use different estimators.

You remember the first equation

$$y = \beta + \alpha X + \epsilon$$

The final weights are 3.8 and 2.78. The video below shows you how the gradient descent optimize the loss function to find this weights

How to train a Linear Regression with TensorFlow

Now that you have a better understanding of what is happening behind the hood, you are ready to use the estimator API provided by TensorFlow to train your first linear regression.

You will use the Boston Dataset, which includes the following variables

crim	per capita crime rate by town
zn	proportion of residential land zoned for lots over 25,000 sq.ft.
indus	proportion of non-retail business acres per town.
nox	nitric oxides concentration
rm	average number of rooms per dwelling
age	proportion of owner-occupied units built before 1940
dis	weighted distances to five Boston employment centers
tax	full-value property-tax rate per dollars 10,000
ptratio	pupil-teacher ratio by town
medv	Median value of owner-occupied homes in thousand dollars

You will create three different datasets:

dataset	objective	shape
Training	Train the model and obtain the weights	400, 10
Evaluation	Evaluate the performance of the model on unseen data	100, 10
Predict	Use the model to predict house value on new data	6, 10

The objectives is to use the features of the dataset to predict the value of the house.

During the second part of the tutorial, you will learn how to use TensorFlow with three different way to import the data:

- With Pandas

- With Numpy
- Only TF

Note that, all options **provide the same results.**

You will learn how to use the high-level API to build, train and evaluate a linear regression model. If you were using the low-level API, you had to define by hand the:

- Loss function
- Optimize: Gradient descent
- Matrices multiplication
- Graph and tensor

This is tedious and more complicated for beginner.

Pandas

You need to import the necessary libraries to train the model.

```
import pandas as pd
from sklearn import datasets
import tensorflow as tf
import itertools
```

Step 1) Import the data with panda.

You define the column names and store it in COLUMNS. You can use pd.read_csv() to import the data.

```
COLUMNS = ["crim", "zn", "indus", "nox", "rm", "age",
           "dis", "tax", "ptratio", "medv"]
```

```
training_set = pd.read_csv("E:/boston_train.csv",
                           skipinitialspace=True,skiprows=1, names=COLUMNS)
```

```
test_set = pd.read_csv("E:/boston_test.csv", skipinitialspace=True,skiprows=1,
                      names=COLUMNS)
```

```
prediction_set = pd.read_csv("E:/boston_predict.csv",
                             skipinitialspace=True,skiprows=1, names=COLUMNS)
```

You can print the shape of the data.

```
print(training_set.shape, test_set.shape, prediction_set.shape)
```

Output

```
(400, 10) (100, 10) (6, 10)
```

Note that the label, i.e. your y, is included in the dataset. So you need to define two other lists. One containing only the features and one with the name of the

label only. These two lists will tell your estimator what are the features in the dataset and what column name is the label

It is done with the code below.

```
FEATURES = ["crim", "zn", "indus", "nox", "rm",
            "age", "dis", "tax", "ptratio"]
LABEL = "medv"
```

Step 2) Convert the data

You need to convert the numeric variables in the proper format. Tensorflow provides a method to convert continuous variable:

`tf.feature_column.numeric_column()`.

In the previous step, you define a list a feature you want to include in the model. Now you can use this list to convert them into numeric data. If you want to exclude features in your model, feel free to drop one or more variables in the list FEATURES before you construct the feature_cols

Note that you will use Python list comprehension with the list FEATURES to create a new list named feature_cols. It helps you avoid writing nine times `tf.feature_column.numeric_column()`. A list comprehension is a faster and cleaner way to create new lists

```
feature_cols = [tf.feature_column.numeric_column(k) for k in
                FEATURES]
```

Step 3) Define the estimator

In this step, you need to define the estimator. Tensorflow currently provides 6 pre-built estimators, including 3 for classification task and 3 for regression task:

- Regressor
 - DNNRegressor
 - LinearRegressor

- DNNLinearCombinedRegressor
- Classifier
 - DNNClassifier
 - LinearClassifier
 - DNNLinearCombinedClassifier

In this tutorial, you will use the Linear Regressor. To access this function, you need to use `tf.estimator`.

The function needs two arguments:

- `feature_columns`: Contains the variables to include in the model
- `model_dir`: path to store the graph, save the model parameters, etc

Tensorflow will automatically create a file named `train` in your working directory. You need to use this path to access the Tensorboard.

```
estimator = tf.estimator.LinearRegressor(
    feature_columns=feature_cols,
    model_dir="train")
```

Output

```
INFO:tensorflow:Using default config.
INFO:tensorflow:Using config: {'_model_dir': 'train',
'_tf_random_seed': None, '_save_summary_steps': 100,
'_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,
'_session_config': None, '_keep_checkpoint_max': 5,
'_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps':
100, '_train_distribute': None, '_service': None, '_cluster_spec':
<tensorflow.python.training.server_lib.ClusterSpec object at
0x1a215dc550>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':
'', '_is_chief': True, '_num_ps_replicas': 0,
'_num_worker_replicas': 1}
```

The tricky part with TensorFlow is the way to feed the model. Tensorflow is designed to work with parallel computing and very large dataset. Due to the

limitation of the machine resources, it is impossible to feed the model with all the data at once. For that, you need to feed a batch of data each time. Note that, we are talking about huge dataset with millions or more records. If you don't add batch, you will end up with a memory error.

For instance, if your data contains 100 observations and you define a batch size of 10, it means the model will see 10 observations for each iteration (10×10).

When the model has seen all the data, it finishes one **epoch**. An epoch defines how many times you want the model to see the data. It is better to set this step to none and let the model performs iteration number of time.

A second information to add is if you want to shuffle the data before each iteration. During the training, it is important to shuffle the data so that the model does not learn specific pattern of the dataset. If the model learns the details of the underlying pattern of the data, it will have difficulties to generalize the prediction for unseen data. This is called **overfitting**. The model performs well on the training data but cannot predict correctly for unseen data.

TensorFlow makes this two steps easy to do. When the data goes to the pipeline, it knows how many observations it needs (batch) and if it has to shuffle the data.

To instruct Tensorflow how to feed the model, you can use `pandas_input_fn`. This object needs 5 parameters:

- `x`: feature data
- `y`: label data
- `batch_size`: batch. By default 128
- `num_epoch`: Number of epoch, by default 1
- `shuffle`: Shuffle or not the data. By default, None

You need to feed the model many times so you define a function to repeat this process. all this function `get_input_fn`.

```
def get_input_fn(data_set, num_epochs=None, n_batch = 128,
shuffle=True):
    return tf.estimator.inputs.pandas_input_fn(
        x=pd.DataFrame({k: data_set[k].values for k in FEATURES}),
        y = pd.Series(data_set[LABEL].values),
        batch_size=n_batch,
        num_epochs=num_epochs,
        shuffle=shuffle)
```

The usual method to evaluate the performance of a model is to:

- Train the model
 - Evaluate the model in a different dataset
 - Make prediction

Tensorflow estimator provides three different functions to carry out this three steps easily.

Step 4): Train the model

You can use the estimator train to evaluate the model. The train estimator needs an input_fn and a number of steps. You can use the function you created above to feed the model. Then, you instruct the model to iterate 1000 times. Note that, you don't specify the number of epochs, you let the model iterates 1000 times. If you set the number of epoch to 1, then the model will iterate 4 times: There are 400 records in the training set, and the batch size is 128

1. 128 rows
 2. 128 rows
 3. 128 rows
 4. 16 rows

Therefore, it is easier to set the number of epoch to none and define the number of iteration.

```
n_batch = 128,  
shuffle=False),  
steps=1000)
```

Output

```
INFO:tensorflow:Calling model_fn.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow>Create CheckpointSaverHook.  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Saving checkpoints for 1 into train/model.ckpt.  
INFO:tensorflow:loss = 83729.64, step = 1  
INFO:tensorflow:global_step/sec: 238.616  
INFO:tensorflow:loss = 13909.657, step = 101 (0.420 sec)  
INFO:tensorflow:global_step/sec: 314.293  
INFO:tensorflow:loss = 12881.449, step = 201 (0.320 sec)  
INFO:tensorflow:global_step/sec: 303.863  
INFO:tensorflow:loss = 12391.541, step = 301 (0.327 sec)  
INFO:tensorflow:global_step/sec: 308.782  
INFO:tensorflow:loss = 12050.5625, step = 401 (0.326 sec)  
INFO:tensorflow:global_step/sec: 244.969  
INFO:tensorflow:loss = 11766.134, step = 501 (0.407 sec)  
INFO:tensorflow:global_step/sec: 155.966  
INFO:tensorflow:loss = 11509.922, step = 601 (0.641 sec)  
INFO:tensorflow:global_step/sec: 263.256  
INFO:tensorflow:loss = 11272.889, step = 701 (0.379 sec)  
INFO:tensorflow:global_step/sec: 254.112  
INFO:tensorflow:loss = 11051.9795, step = 801 (0.396 sec)  
INFO:tensorflow:global_step/sec: 292.405  
INFO:tensorflow:loss = 10845.855, step = 901 (0.341 sec)  
INFO:tensorflow:Saving checkpoints for 1000 into train/model.ckpt.  
INFO:tensorflow:Loss for final step: 5925.9873.
```

You can check the Tensorboard will the following command:

```
activate hello-tf  
# For MacOS  
tensorboard --logdir=./train  
# For Windows  
tensorboard --logdir=train
```

Step 5) Evaluate your model

You can evaluate the fit of your model on the test set with the code below:

```
ev = estimator.evaluate(  
    input_fn=get_input_fn(test_set,  
    num_epochs=1,  
    n_batch = 128,  
    shuffle=False))
```

Output

```
INFO:tensorflow:Calling model_fn.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow:Starting evaluation at 2018-05-13-01:43:13  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Restoring parameters from train/model.ckpt-1000  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Finished evaluation at 2018-05-13-01:43:13  
INFO:tensorflow:Saving dict for global step 1000: average_loss =  
32.15896, global_step = 1000, loss = 3215.896
```

You can print the loss with the code below:

```
loss_score = ev["loss"]  
print("Loss: {:.f}".format(loss_score))
```

Output

```
Loss: 3215.895996
```

The model has a loss of 3215. You can check the summary statistic to get an idea of how big the error is.

```
training_set['medv'].describe()
```

Output

```
count    400.000000  
mean     22.625500
```

```
std      9.572593
min     5.000000
25%    16.600000
50%    21.400000
75%    25.025000
max    50.000000
Name: medv, dtype: float64
```

From the summary statistic above, you know that the average price for a house is 22 thousand, with a minimum price of 9 thousands and maximum of 50 thousand. The model makes a typical error of 3k dollars.

Step 6) Make the prediction

Finally, you can use the estimator predict to estimate the value of 6 Boston houses.

```
y = estimator.predict(
    input_fn=get_input_fn(prediction_set,
    num_epochs=1,
    n_batch = 128,
    shuffle=False))
```

To print the estimated values of , you can use this code:

```
predictions = list(p["predictions"] for p in itertools.islice(y,
6))print("Predictions: {}".format(str(predictions)))
```

Output

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from train/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
Predictions: [array([32.297546], dtype=float32), array([18.96125],
dtype=float32), array([27.270979], dtype=float32),
array([29.299236], dtype=float32), array([16.436684],
dtype=float32), array([21.460876], dtype=float32)]
```

The model forecast the following values:

House	Prediction
1	32.29
2	18.96
3	27.27
4	29.29
5	16.43
7	21.46

Note that we don't know the true value of . In the tutorial of deep learning, you will try to beat the linear model

Numpy Solution

This section explains how to train the model using a numpy estimator to feed the data. The method is the same except that you will use numpy_input_fn estimator.

```
training_set_n = pd.read_csv("E:/boston_train.csv").values
```

```
test_set_n = pd.read_csv("E:/boston_test.csv").values
```

```
prediction_set_n = pd.read_csv("E:/boston_predict.csv").values
```

Step 1) Import the data

First of all, you need to differentiate the feature variables from the label. You need to do this for the training data and evaluation. It is faster to define a function to split the data.

```
def prepare_data(df):
    X_train = df[:, :-3]
    y_train = df[:, -3]
    return X_train, y_train
```

You can use the function to split the label from the features of the train/evaluate dataset

```
X_train, y_train = prepare_data(training_set_n)
X_test, y_test = prepare_data(test_set_n)
```

You need to exclude the last column of the prediction dataset because it contains only NaN

```
x_predict = prediction_set_n[:, :-2]
```

Confirm the shape of the array. Note that, the label should not have a dimension, it means (400,).

```
print(X_train.shape, y_train.shape, x_predict.shape)
```

Output

```
(400, 9) (400,) (6, 9)
```

You can construct the feature columns as follow:

```
feature_columns = [      tf.feature_column.numeric_column('x',
shape=X_train.shape[1:])]
```

The estimator is defined as before, you instruct the feature columns and where to save the graph.

```
estimator = tf.estimator.LinearRegressor(
    feature_columns=feature_columns,
    model_dir="train1")
```

Output

```
INFO:tensorflow:Using default config.
INFO:tensorflow:Using config: {'_model_dir': 'train1',
'_tf_random_seed': None, '_save_summary_steps': 100,
'_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,
'_session_config': None, '_keep_checkpoint_max': 5,
'_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps':
100, '_train_distribute': None, '_service': None, '_cluster_spec':
<tensorflow.python.training.server_lib.ClusterSpec object at
0x1a218d8f28>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':
'', '_is_chief': True, '_num_ps_replicas': 0,
'_num_worker_replicas': 1}
```

You can use the numpy estimapor to feed the data to the model and then train the model. Note that, we define the input_fn function before to ease the readability.

```
# Train the estimator
train_input =
tf.estimator.inputs.numpy_input_fn(
    x={"x": X_train},
    y=y_train,
    batch_size=128,
```

```
        shuffle=False,  
        num_epochs=None)  
estimator.train(input_fn = train_input, steps=5000)
```

Output

```
INFO:tensorflow:Calling model_fn.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow>Create CheckpointSaverHook.  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Saving checkpoints for 1 into train1/model.ckpt.  
INFO:tensorflow:loss = 83729.64, step = 1  
INFO:tensorflow:global_step/sec: 490.057  
INFO:tensorflow:loss = 13909.656, step = 101 (0.206 sec)  
INFO:tensorflow:global_step/sec: 788.986  
INFO:tensorflow:loss = 12881.45, step = 201 (0.126 sec)  
INFO:tensorflow:global_step/sec: 736.339  
INFO:tensorflow:loss = 12391.541, step = 301 (0.136 sec)  
INFO:tensorflow:global_step/sec: 383.305  
INFO:tensorflow:loss = 12050.561, step = 401 (0.260 sec)  
INFO:tensorflow:global_step/sec: 859.832  
INFO:tensorflow:loss = 11766.133, step = 501 (0.117 sec)  
INFO:tensorflow:global_step/sec: 804.394  
INFO:tensorflow:loss = 11509.918, step = 601 (0.125 sec)  
INFO:tensorflow:global_step/sec: 753.059  
INFO:tensorflow:loss = 11272.891, step = 701 (0.134 sec)  
INFO:tensorflow:global_step/sec: 402.165  
INFO:tensorflow:loss = 11051.979, step = 801 (0.248 sec)  
INFO:tensorflow:global_step/sec: 344.022  
INFO:tensorflow:loss = 10845.854, step = 901 (0.288 sec)  
INFO:tensorflow:Saving checkpoints for 1000 into train1/model.ckpt.  
INFO:tensorflow:Loss for final step: 5925.985.  
Out[23]:  
<tensorflow.python.estimator.canned.linear.LinearRegressor at  
0x1a1b6ea860>
```

You replicate the same step with a different estimator to evaluate your model

```
eval_input = tf.estimator.inputs.numpy_input_fn(  
    x={"x": X_test},  
    y=y_test,  
    shuffle=False,
```

```
    batch_size=128,  
    num_epochs=1)  
estimator.evaluate(eval_input, steps=None)
```

Output

```
INFO:tensorflow:Calling model_fn.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow:Starting evaluation at 2018-05-13-01:44:00  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Restoring parameters from train1/model.ckpt-1000  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Finished evaluation at 2018-05-13-01:44:00  
INFO:tensorflow:Saving dict for global step 1000: average_loss =  
32.158947, global_step = 1000, loss = 3215.8945  
Out[24]:  
{'average_loss': 32.158947, 'global_step': 1000, 'loss': 3215.8945}
```

Finaly, you can compute the prediction. It should be the similar as pandas.

```
test_input = tf.estimator.inputs.numpy_input_fn(  
    x={"x": x_predict},  
    batch_size=128,  
    num_epochs=1,  
    shuffle=False)  
y = estimator.predict(test_input)  
predictions = list(p["predictions"] for p in itertools.islice(y,  
6))  
print("Predictions: {}".format(str(predictions)))
```

Output

```
INFO:tensorflow:Calling model_fn.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Restoring parameters from train1/model.ckpt-1000  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
Predictions: [array([32.297546], dtype=float32), array([18.961248],  
dtype=float32), array([27.270979], dtype=float32),  
array([29.299242], dtype=float32), array([16.43668],  
dtype=float32), array([21.460878], dtype=float32)]
```

Tensorflow solution

The last section is dedicated to a TensorFlow solution. This method is slightly more complicated than the other one.

Note that if you use Jupyter notebook, you need to Restart and clean the kernel to run this session.

TensorFlow has built a great tool to pass the data into the pipeline. In this section, you will build the input_fn function by yourself.

Step 1) Define the path and the format of the data

First of all, you declare two variables with the path of the csv file. Note that, you have two files, one for the training set and one for the testing set.

```
import tensorflow as tf  
  
df_train = "E:/boston_train.csv"  
  
df_eval = "E:/boston_test.csv"
```

Then, you need to define the columns you want to use from the csv file. We will use all. After that, you need to declare the type of variable it is.

Floats variable are defined by [0.]

```
COLUMNS = ["crim", "zn", "indus", "nox", "rm", "age",  
          "dis", "tax", "ptratio", "medv"]RECORDS_ALL =  
[[0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0]]
```

Step 2) Define the input_fn function

The function can be broken into three part:

1. Import the data

2. Create the iterator
3. Consume the data

Below is the overall code to define the function. The code will be explained after

```
def input_fn(data_file, batch_size, num_epoch = None):
    # Step 1
    def parse_csv(value):
        columns = tf.decode_csv(value, record_defaults=
RECORDS_ALL)
        features = dict(zip(COLUMNS, columns))
        #labels = features.pop('median_house_value')
        labels = features.pop('medv')
        return features, labels

    # Extract lines from input files using the
    # Dataset API.
    dataset =
(tf.data.TextLineDataset(data_file) # Read text file
 .skip(1) # Skip header row
 .map(parse_csv))

    dataset = dataset.repeat(num_epoch)
    dataset = dataset.batch(batch_size)
    # Step 3
    iterator = dataset.make_one_shot_iterator()
    features, labels = iterator.get_next()
    return features, labels
```

** Import the data**

For a csv file, the dataset method reads one line at a time. To build the dataset, you need to use the object TextLineDataset. Your dataset has a header so you need to use skip(1) to skip the first line. At this point, you only read the data and exclude the header in the pipeline. To feed the model, you need to separate the features from the label. The method used to apply any transformation to the data is map.

This method calls a function that you will create in order to instruct how to transform the data. In a nutshell, you need to pass the data in the TextLineDataset

object, exclude the header and apply a transformation which is instructed by a function. Code explanation

- `tf.data.TextLineDataset(data_file)`: This line read the csv file
- `.skip(1)` : skip the header
- `.map(parse_csv))`: parse the records into the tensors You need to define a function to instruct the map object. You can call this function `parse_csv`.

This function parses the csv file with the method `tf.decode_csv` and declares the features and the label. The features can be declared as a dictionary or a tuple. You use the dictionary method because it is more convenient. Code explanation

- `tf.decode_csv(value, record_defaults= RECORDS_ALL)`: the method `decode_csv` uses the output of the `TextLineDataset` to read the csv file. `record_defaults` instructs TensorFlow about the columns type.
- `dict(zip(_CSV_COLUMNS, columns))`: Populate the dictionary with all the columns extracted during this data processing
- `features.pop('median_house_value')`: Exclude the target variable from the feature variable and create a label variable

The Dataset needs further elements to iteratively feeds the Tensors. Indeed, you need to add the method `repeat` to allow the dataset to continue indefinitely to feed the model. If you don't add the method, the model will iterate only one time and then throw an error because no more data are fed in the pipeline.

After that, you can control the batch size with the `batch` method. It means you tell the dataset how many data you want to pass in the pipeline for each iteration. If you set a big batch size, the model will be slow.

Step 3) Create the iterator

Now you are ready for the second step: create an iterator to return the elements in the dataset.

The simplest way of creating an operator is with the method make_one_shot_iterator.

After that, you can create the features and labels from the iterator.

Step 4) Consume the data

You can check what happens with input_fn function. You need to call the function in a session to consume the data. You try with a batch size equals to 1.

Note that, it prints the features in a dictionary and the label as an array.

It will show the first line of the csv file. You can try to run this code many times with different batch size.

```
next_batch = input_fn(df_train, batch_size = 1, num_epoch = None)
with tf.Session() as sess:
    first_batch = sess.run(next_batch)
    print(first_batch)
```

Output

```
({'crim': array([2.3004], dtype=float32), 'zn': array([0.], dtype=float32), 'indus': array([19.58], dtype=float32), 'nox': array([0.605], dtype=float32), 'rm': array([6.319], dtype=float32), 'age': array([96.1], dtype=float32), 'dis': array([2.1], dtype=float32), 'tax': array([403.], dtype=float32), 'ptratio': array([14.7], dtype=float32)}, array([23.8], dtype=float32))
```

Step 4) Define the feature column

You need to define the numeric columns as follow:

```
X1= tf.feature_column.numeric_column('crim')
X2= tf.feature_column.numeric_column('zn')
X3= tf.feature_column.numeric_column('indus')
X4= tf.feature_column.numeric_column('nox')
X5= tf.feature_column.numeric_column('rm')
X6= tf.feature_column.numeric_column('age')
```

```
X7= tf.feature_column.numeric_column('dis')
X8= tf.feature_column.numeric_column('tax')
X9= tf.feature_column.numeric_column('ptratio')
```

Note that you need to combined all the variables in a bucket

```
base_columns = [X1, X2, X3,X4, X5, X6,X7, X8, X9]
```

Step 5) Build the model

You can train the model with the estimator LinearRegressor.

```
model = tf.estimator.LinearRegressor(feature_columns=base_columns,
model_dir='train3')
```

Output

```
INFO:tensorflow:Using default config. INFO:tensorflow:Using config:
{'_model_dir': 'train3', '_tf_random_seed': None,
'_save_summary_steps': 100, '_save_checkpoints_steps': None,
'_save_checkpoints_secs': 600, '_session_config': None,
'_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000,
'_log_step_count_steps': 100, '_train_distribute': None,
'_service': None, '_cluster_spec':
<tensorflow.python.training.server_lib.ClusterSpec object at
0x1820a010f0>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':
'', '_is_chief': True, '_num_ps_replicas': 0,
'_num_worker_replicas': 1}
```

You need to use a lambda function to allow to write the argument in the function input_fn. If you don't use a lambda function, you cannot train the model.

```
# Train the estimator
model.train(steps =1000,
            input_fn= lambda : input_fn(df_train,batch_size=128,
num_epoch = None))
```

Output

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
```

```
INFO:tensorflow>Create CheckpointSaverHook.  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Saving checkpoints for 1 into train3/model.ckpt.  
INFO:tensorflow:loss = 83729.64, step = 1  
INFO:tensorflow:global_step/sec: 72.5646  
INFO:tensorflow:loss = 13909.657, step = 101 (1.380 sec)  
INFO:tensorflow:global_step/sec: 101.355  
INFO:tensorflow:loss = 12881.449, step = 201 (0.986 sec)  
INFO:tensorflow:global_step/sec: 109.293  
INFO:tensorflow:loss = 12391.541, step = 301 (0.915 sec)  
INFO:tensorflow:global_step/sec: 102.235  
INFO:tensorflow:loss = 12050.5625, step = 401 (0.978 sec)  
INFO:tensorflow:global_step/sec: 104.656  
INFO:tensorflow:loss = 11766.134, step = 501 (0.956 sec)  
INFO:tensorflow:global_step/sec: 106.697  
INFO:tensorflow:loss = 11509.922, step = 601 (0.938 sec)  
INFO:tensorflow:global_step/sec: 118.454  
INFO:tensorflow:loss = 11272.889, step = 701 (0.844 sec)  
INFO:tensorflow:global_step/sec: 114.947  
INFO:tensorflow:loss = 11051.9795, step = 801 (0.870 sec)  
INFO:tensorflow:global_step/sec: 111.484  
INFO:tensorflow:loss = 10845.855, step = 901 (0.897 sec)  
INFO:tensorflow:Saving checkpoints for 1000 into train3/model.ckpt.  
INFO:tensorflow:Loss for final step: 5925.9873.  
Out[8]:  
<tensorflow.python.estimator.canned.linear.LinearRegressor at  
0x18225eb8d0>
```

You can evaluate the fit of your model on the test set with the code below:

```
results = model.evaluate(steps =None, input_fn=lambda:  
input_fn(df_eval, batch_size =128, num_epoch = 1))  
for key in results:  
print("    {}, was: {}".format(key, results[key]))
```

Output

```
INFO:tensorflow:Calling model_fn.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow:Starting evaluation at 2018-05-13-02:06:02  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Restoring parameters from train3/model.ckpt-1000
```

```
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Finished evaluation at 2018-05-13-02:06:02  
INFO:tensorflow:Saving dict for global step 1000: average_loss =  
32.15896, global_step = 1000, loss = 3215.896  
    average_loss, was: 32.158958435058594  
    loss, was: 3215.89599609375  
    global_step, was: 1000
```

The last step is predicting the value of based on the value of , the matrices of the features. You can write a dictionary with the values you want to predict. Your model has 9 features so you need to provide a value for each. The model will provide a prediction for each of them.

In the code below, you wrote the values of each features that is contained in the df_predict csv file.

You need to write a new input_fn function because there is no label in the dataset. You can use the API from_tensor from the Dataset.

```
prediction_input = {  
    'crim':  
[0.03359, 5.09017, 0.12650, 0.05515, 8.15174, 0.24522],  
    'zn': [75.0, 0.0, 25.0, 33.0, 0.0, 0.0],  
    'indus': [2.95, 18.10, 5.13, 2.18, 18.10, 9.90],  
    'nox': [0.428, 0.713, 0.453, 0.472, 0.700, 0.544],  
    'rm': [7.024, 6.297, 6.762, 7.236, 5.390, 5.782],  
    'age': [15.8, 91.8, 43.4, 41.1, 98.9, 71.7],  
    'dis': [5.4011, 2.3682, 7.9809, 4.0220, 1.7281, 4.0317],  
    'tax': [252, 666, 284, 222, 666, 304],  
    'ptratio': [18.3, 20.2, 19.7, 18.4, 20.2, 18.4]  
}  
def test_input_fn():  
    dataset = tf.data.Dataset.from_tensors(prediction_input)  
    return dataset  
  
# Predict all our prediction_input  
pred_results =  
model.predict(input_fn=test_input_fn)
```

Finaly, you print the predictions.

```
for pred in enumerate(pred_results):
    print(pred)
```

Output

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from train3/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
(0, {'predictions': array([32.297546], dtype=float32)})
(1, {'predictions': array([18.96125], dtype=float32)})
(2, {'predictions': array([27.270979], dtype=float32)})
(3, {'predictions': array([29.299236], dtype=float32)})
(4, {'predictions': array([16.436684], dtype=float32)})
(5, {'predictions': array([21.460876], dtype=float32)})

INFO:tensorflow:Calling model_fn. INFO:tensorflow:Done calling
model_fn. INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from train3/model.ckpt-5000
INFO:tensorflow:Running local_init_op. INFO:tensorflow:Done running
local_init_op. (0, {'predictions': array([35.60663],
dtype=float32)}) (1, {'predictions': array([22.298521],
dtype=float32)}) (2, {'predictions': array([25.74533],
dtype=float32)}) (3, {'predictions': array([35.126694],
dtype=float32)}) (4, {'predictions': array([17.94416],
dtype=float32)}) (5, {'predictions': array([22.606628],
dtype=float32)})
```

Summary

To train a model, you need to:

- Define the features: Independent variables: X
- Define the label: Dependent variable: y
- Construct a train/test set
- Define the initial weight
- Define the loss function: MSE
- Optimize the model: Gradient descent

- Define:
 - Learning rate
 - Number of epoch
 - Batch size

In this tutorial, you learned how to use the high level API for a linear regression estimator. You need to define:

1. Feature columns. If continuous: `tf.feature_column.numeric_column()`. You can populate a list with python list comprehension
2. The estimator: `tf.estimator.LinearRegressor(feature_columns, model_dir)`
3. A function to import the data, the batch size and epoch: `input_fn()`

After that, you are ready to train, evaluate and make prediction with `train()`, `evaluate()` and `predict()`

Chapter 14: Linear Regression Case Study

In this tutorial, you will learn how to check the data and prepare it to create a linear regression task.

This tutorial is divided into two parts:

- Look for interaction
- Test the model

In the previous tutorial, you used the Boston dataset to estimate the median price of a house. Boston dataset has a small size, with only 506 observations. This dataset is considered as a benchmark to try new linear regression algorithms.

The dataset is composed of:

Variable	Description
zn	The proportion of residential land zoned for lots over 25,000 sq.ft.
indus	The proportion of non-retail business acres per town.
nox	nitric oxides concentration
rm	average number of rooms per dwelling
age	the proportion of owner-occupied units built before 1940
dis	weighted distances to five Boston employment centers
tax	full-value property-tax rate per dollars 10,000
ptratio	the pupil-teacher ratio by a town
medv	The median value of owner-occupied homes in thousand dollars
crim	per capita crime rate by town
chas	Charles River dummy variable (1 if bounds river; 0 otherwise)
B	the proportion of blacks by the town

In this tutorial, we will estimate the median price using a linear regressor, but the focus is on one particular process of machine learning: "data preparation."

A model generalizes the pattern in the data. To capture such a pattern, you need to find it first. A good practice is to perform a data analysis before running any machine learning algorithm.

Choosing the right features makes all the difference in the success of your model. Imagine you try to estimate the wage of a people, if you do not include the gender as a covariate, you end up with a poor estimate.

Another way to improve the model is to look at the correlation between the independent variable. Back to the example, you can think of education as an excellent candidate to predict the wage but also the occupation. It is fair to say, the occupation depends on the level of education, namely higher education often leads to a better occupation. If we generalize this idea, we can say the correlation between the dependent variable and an explanatory variable can be magnified of yet another explanatory variable.

To capture the limited effect of education on occupation, we can use an interaction term.

INTERACTION TERM

Interaction terms allow us model relationships when the effects of a feature on the target is influenced by another feature.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + e$$

↑
The interaction of features
 x_1 and x_2 .

ChrisAlbon

If you look at the wage equation, it becomes:

$$wage = \alpha + \beta_1 occupation + \beta_2 education + \beta_3 occupation * education + \epsilon$$

If β_3 is positive, then it implies that an additional level of education yields a higher increase in the median value of a house for a high occupation level. In other words, there is an interaction effect between education and occupation.

In this tutorial, we will try to see which variables can be a good candidate for interaction terms. We will test if adding this kind of information leads to better price prediction.

Summary statistics

There are a few steps you can follow before proceeding to the model. As mentioned earlier, the model is a generalization of the data. The best practice is to understand the data and make a prediction. If you do not know your data, you have slim chances to improve your model.

As a first step, load the data as a pandas dataframe and create a training set and testing set.

Tips: For this tutorial, you need to have matplotlib and seaborn installed in Python. You can install Python package on the fly with Jupyter. You **Should not** do this

```
!conda install -- yes matplotlib
```

but

```
import sys
!{sys.executable} -m pip install matplotlib # Already installed
!{sys.executable} -m pip install seaborn
```

Note that this step is not necessary if you have matplotlib and seaborn installed.

Matplotlib is the library to create a graph in Python. Seaborn is a statistical visualization library built on top of matplotlib. It provides attractive and beautiful plots.

The code below imports the necessary libraries.

```
import pandas as pd
from sklearn import datasets
import tensorflow as tf
from sklearn.datasets import load_boston
import numpy as np
```

The library sklearn includes the Boston dataset. You can call its API to import the data.

```
boston = load_boston()
df = pd.DataFrame(boston.data)
```

The feature's name are stored in the object feature_names in an array.

```
boston.feature_names
```

Output

```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='|<U7')
```

You can rename the columns.

```
df.columns = boston.feature_names
df['PRICE'] = boston.target
df.head(2)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.9	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.9	9.14	21.6

You convert the variable CHAS as a string variable and label it with yes if CHAS = 1 and no if CHAS = 0

```
df['CHAS'] = df['CHAS'].map({1:'yes', 0:'no'})
df['CHAS'].head(5)
0    no
1    no
2    no
3    no
4    no
Name: CHAS, dtype: object
```

With pandas, it is straightforward to split the dataset. You randomly divide the dataset with 80 percent training set and 20 percent testing set. Pandas have a

built-in function to split a data frame sample.

The first parameter `frac` is a value from 0 to 1. You set it to 0.8 to select randomly 80 percent of the data frame.

`Random_state` allows to have the same dataframe returned for everyone.

```
### Create train/test set  
df_train=df.sample(frac=0.8,random_state=200)  
df_test=df.drop(df_train.index)
```

You can get the shape of the data. It should be:

- Train set: $506 \times 0.8 = 405$
- Test set: $506 \times 0.2 = 101$

```
print(df_train.shape, df_test.shape)
```

Output

```
(405, 14) (101, 14)
```

```
df_test.head(5)
```

Output

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PR
0	0.00632	18.0	2.31	no	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	no	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.0
3	0.03237	0.0	2.18	no	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
6	0.08829	12.5	7.87	no	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	12.43	22.9
7	0.14455	12.5	7.87	no	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.90	19.15	27.0

Data is messy; it's often misbalanced and sprinkled with outlier values that throw off the analysis and machine learning training.

The first step to getting the dataset cleaned up is understanding where it needs cleaning. Cleaning up a dataset can be tricky to do, especially in any

generalizable manner

Google Research team has developed a tool for this job called **Facets** that help to visualize the data and slice it in all sorts of manners. This is a good starting point to comprehend how the dataset is laid out.

Facets allow you to find where the data does not quite look the way you are thinking.

Except for their web app, Google makes it easy to embed the toolkit into a Jupyter notebook.

There are two parts to Facets:

- Facets Overview
- Facets Deep Dive

Facets Overview

Facets Overview gives an overview of the dataset. Facets Overview splits the columns of the data into rows of salient information showing

1. the percentage of missing observation
2. min and max values
3. statistics like the mean, median, and standard deviation.
4. It also adds a column that shows the percentage of values that are zeroes, which is helpful when most of the values are zeroes.
5. It is possible to see these distributions on the test dataset as well as the training set for each feature. It means you can double-check that the test has a similar distribution to the training dataset.

This is at least the minimum to do before any machine learning task. With this tool, you do not miss this crucial step, and it highlights some abnormalities.

Facets Deep Dive

Facets Deep Dive is a cool tool. It allows to have some clarity on your dataset and zoom all the way in to see an individual piece of data. It means you can facet the data by row and column across any of the features of the dataset.

We will use these two tools with the Boston dataset.

Note: You cannot use Facets Overview and Facets Deep Dive at the same time. You need to clear the notebook first to change the tool.

Install Facet

You can use the Facet web app for most of the analysis. In this tutorial, you will see how to use it within a Jupyter Notebook.

First of all, you need to install nbextensions. It is done with this code. You copy and paste the following code in the terminal of your machine.

```
pip install jupyter_contrib_nbextensions
```

Right after that, you need to clone the repositories in your computer. You have two choices:

Option 1) Copy and paste this code in the terminal (**Recommended**)

If you do not have Git installed on your machine, please go to this URL <https://git-scm.com/download/win> and follow the instruction. Once you are done, you can use the git command in the terminal for Mac User or Anaconda prompt for Windows user

```
git clone https://github.com/PAIR-code/facets
```

Option 2) Go to <https://github.com/PAIR-code/facets> and download the repositories.

105 commits 16 branches 2 releases 17 contributors Apache-2.0

Branch: master New pull request Create new file Upload files Find file Clone or download

File	Description	Time Ago
jameswex Merge pull request #131 from PAIR-code/searchstring	...	
facets-dist	added rank histogram to custom stats	11 months ago
facets	fix facets jupyter build and distribute new facets dist compi	11 months ago
facets_dive	fix bug with selected indices	11 months ago
facets_overview	expose search string publically	11 months ago
img	adding images for readme	11 months ago
.gitignore	Initial commit	11 months ago
AUTHORS	Initial commit	11 months ago
CONTRIBUTING.md	Initial commit	11 months ago
CONTRIBUTORS	Initial commit	11 months ago
LICENSE	Initial commit	11 months ago
README.md	Add blank lines before lists	9 months ago
WORKSPACE	Add compile=True to HTML binary rules	2 months ago

Clone with HTTPS Use SSH
Use Git or checkout with SVN using the web URL.
<https://github.com/PAIR-code/facets.git>

Open in Desktop Download ZIP

Click here to download Facets

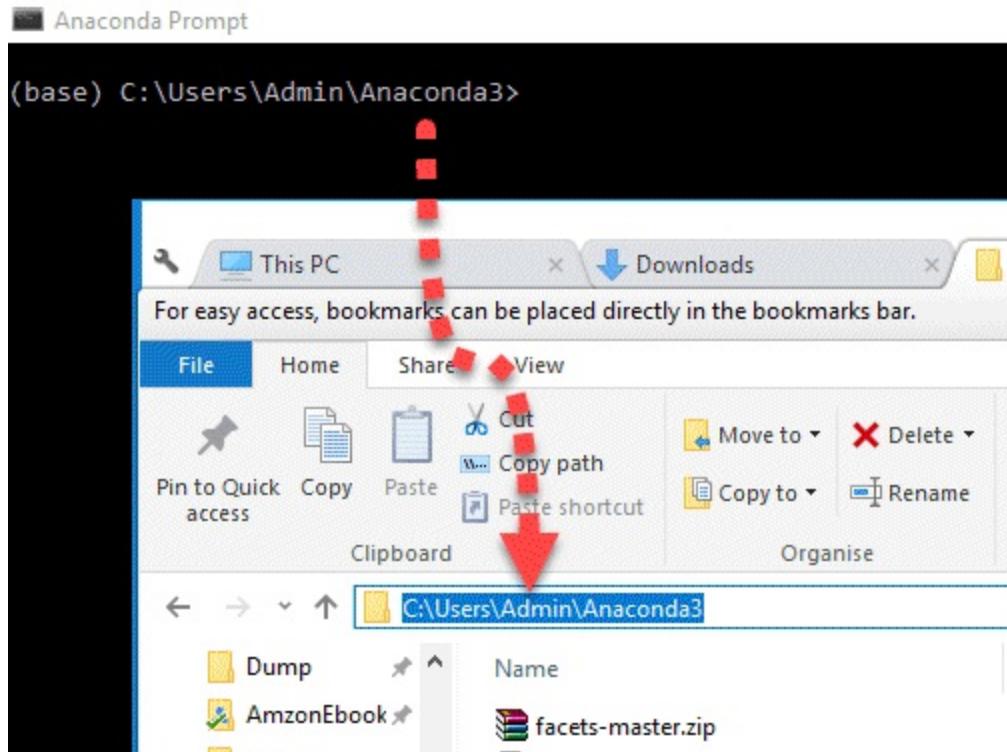
If you choose the first option, the file ends up in your download file. You can either let the file in download or drag it to another path.

You can check where Facets is stored with this command line:

```
echo `pwd`/`ls facets`
```

Now that you have located Facets, you need to install it in Jupyter Notebook. You need to set the working directory to the path where facets is located.

Your present working directory and location of Facets zip should be same.



You need to point the working directory to Facet:

```
cd facets
```

To install Facets in Jupyter, you have two options. If you installed Jupyter with Conda for all the users, copy this code:

can use jupyter nbextension install facets-dist/

```
jupyter nbextension install facets-dist/
```

Otherwise, use:

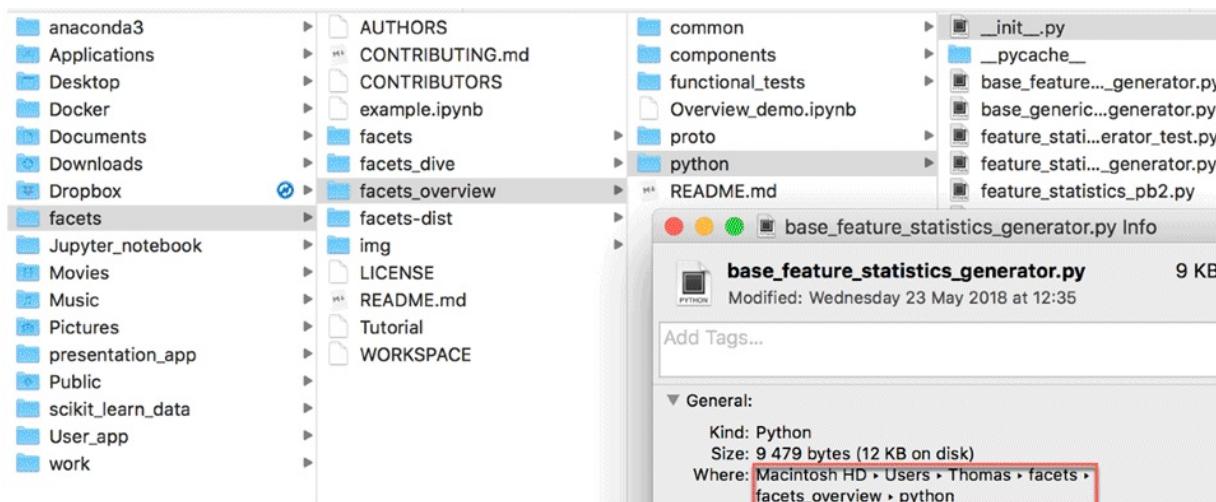
```
jupyter nbextension install facets-dist/ --user
```

All right, you are all set. Let's open Facet Overview.

Overview

Overview uses a Python script to compute the statistics. You need to import the script called generic_feature_statistics_generator to Jupyter. Don't worry; the script is located in the facets files.

You need to locate its path. It is easily done. You open facets, open the file facets_overview and then python. Copy the path



After that, go back to Jupyter, and write the following code. Change the path '/Users/Thomas/facets/facets_overview/python' to your path.

```
# Add the facets overview python code to the python path# Add t
import sys
sys.path.append('/Users/Thomas/facets/facets_overview/python')
```

You can import the script with the code below.

```
from generic_feature_statistics_generator import
GenericFeatureStatisticsGenerator
```

In windows, the same code becomes

```
import sys
sys.path.append(r"C:\Users\Admin\Anaconda3\facets-
master\facets_overview\python")

from generic_feature_statistics_generator import
GenericFeatureStatisticsGenerator
```

To calculate the feature statistics, you need to use the function `GenericFeatureStatisticsGenerator()`, and you use the object `ProtoFromDataFrames`. You can pass the data frame in a dictionary. For instance, if we want to create a summary statistic for the train set, we can store the information in a dictionary and use it in the object `'ProtoFromDataFrames'`

- `'name': 'train', 'table': df_train`

Name is the name of the table displays, and you use the name of the table you want to compute the summary. In your example, the table containing the data is `df_train`

```
# Calculate the feature statistics proto from the datasets and
# stringify it for use in facets overview
import base64

gfsg = GenericFeatureStatisticsGenerator()

proto = gfsg.ProtoFromDataFrames([{'name': 'train', 'table':
df_train},
                                  {'name': 'test', 'table':
df_test}])

#proto = gfsg.ProtoFromDataFrames([{'name': 'train', 'table':
df_train}])
protostr = base64.b64encode(proto.SerializeToString()).decode("utf-
8")
```

Lastly, you just copy and paste the code below. The code comes directly from GitHub. You should be able to see this:



```
# Display the facets overview visualization for this data# Disp
from IPython.core.display import display, HTML

HTML_TEMPLATE = """<link rel="import" href="/nbextensions/facets-
dist/facets-jupyter.html" >
    <facets-overview id="elem"></facets-overview>
    <script>
        document.querySelector("#elem").protoInput =
{protostr}";</script>"""
html = HTML_TEMPLATE.format(protostr=protostr)
display(HTML(html))
```

Graph

After you check the data and their distribution, you can plot a correlation matrix. The correlation matrix computes the Pearson coefficient. This coefficient is bonded between -1 and 1, with a positive value indicates a positive correlation and negative value a negative correlation.

You are interested to see which variables can be a good candidate for interaction terms.

```
## Choose important feature and further check with Dive
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="ticks")
# Compute the correlation matrix
corr = df.corr('pearson')
# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

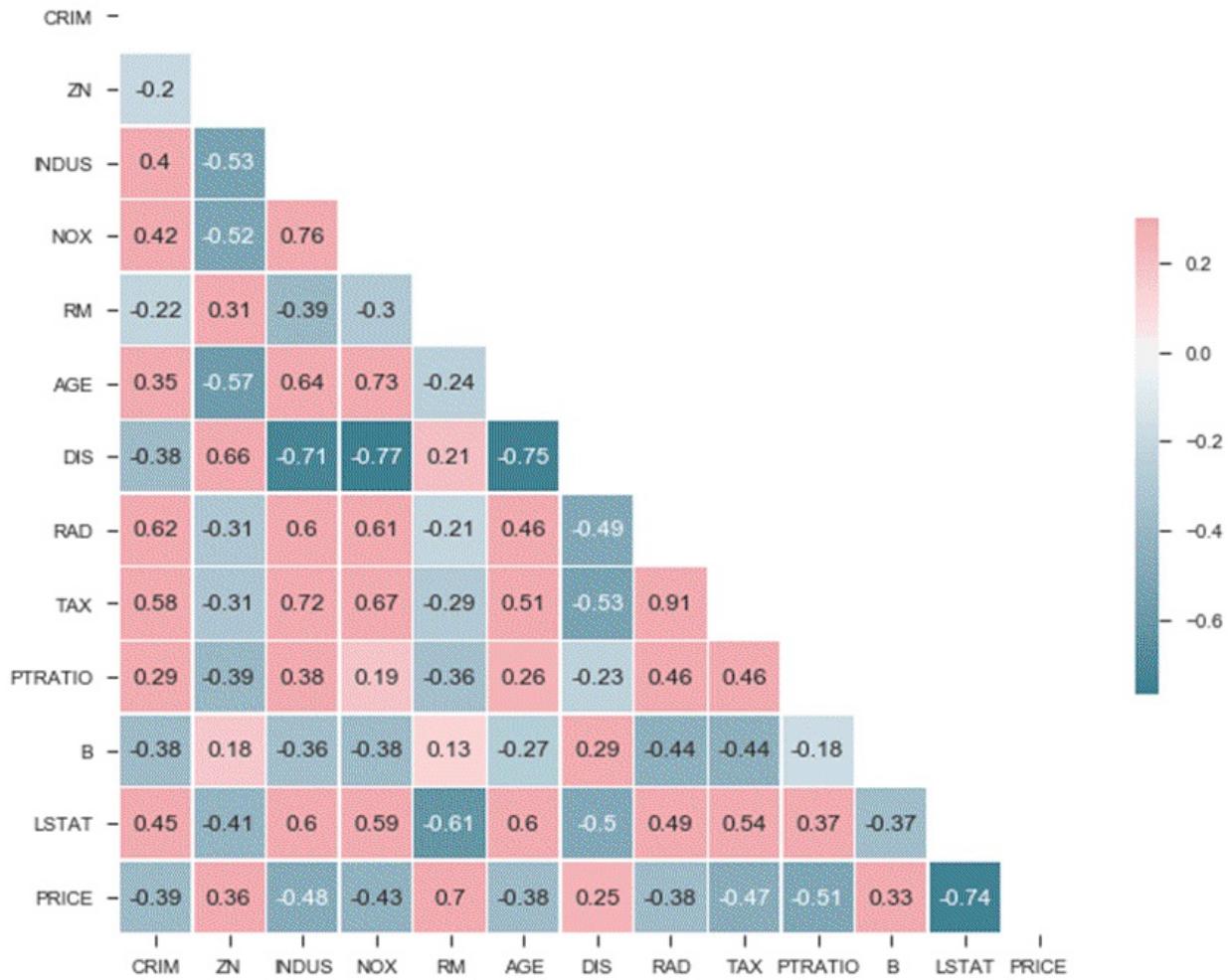
# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3,
            center=0, annot=True,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

Output

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a184d6518>
```

png



From the matrix, you can see:

- LSTAT
- RM

Are strongly correlated with PRICE. Another exciting feature is the strong positive correlation between NOX and INDUS, which means those two variables move in the same direction. Besides, there are also correlated with the PRICE. DIS is also highly correlated with IND and NOX.

You have some first hint that IND and NOX can be good candidates for the interaction term and DIS might also be interesting to focus on.

You can go a little bit deeper by plotting a pair grid. It will illustrate more in detail the correlation map you plotted before.

The pair grid we are composed as follow:

- Upper part: Scatter plot with fitted line
- Diagonal: Kernel density plot
- Lower part: Multivariate kernel density plot

You choose the focus on four independent variables. The choice corresponds to the variables with strong correlation with PRICE

- INDUS
- NOX
- RM
- LSTAT

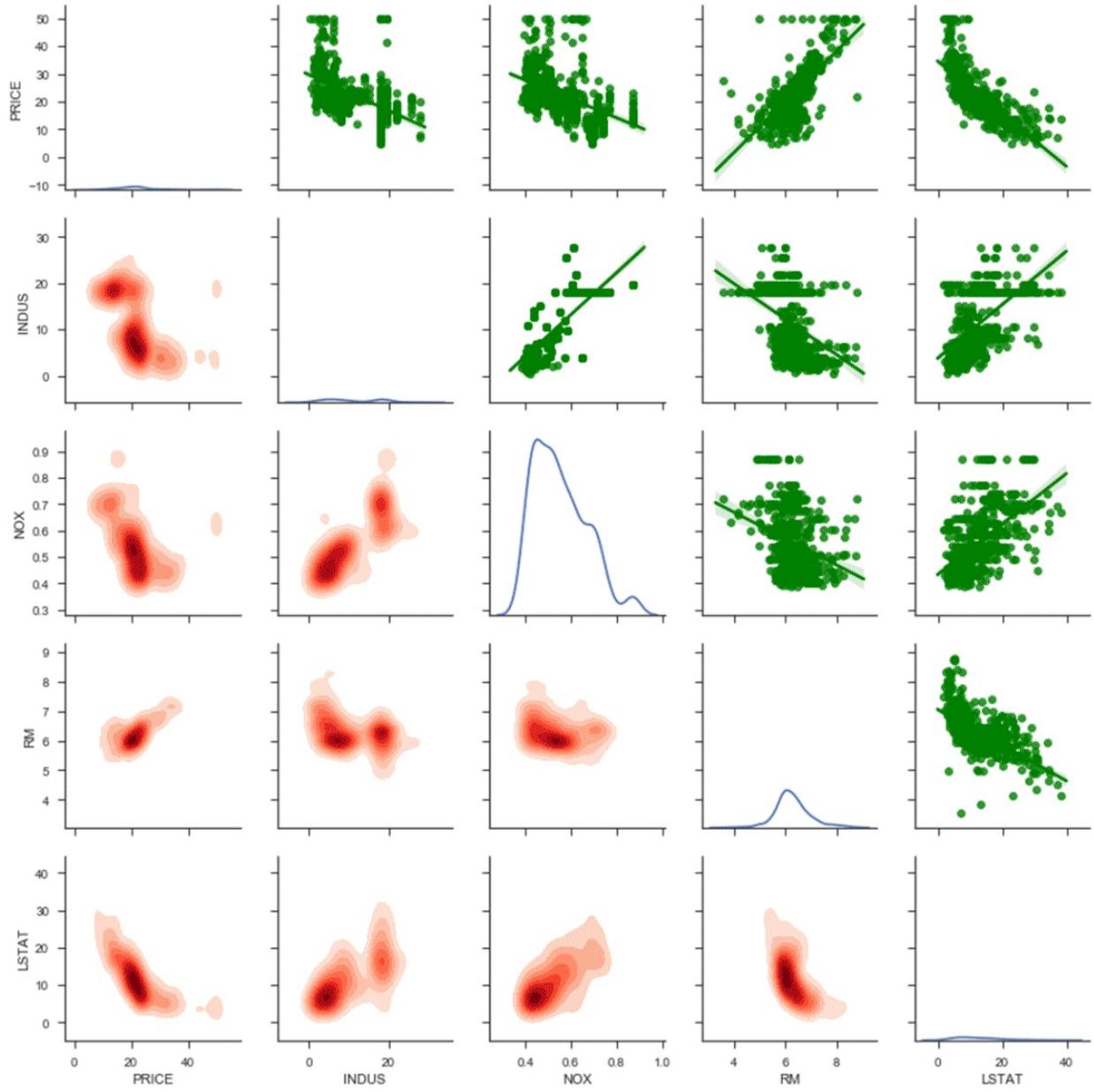
moreover, the PRICE.

Note that the standard error is added by default to the scatter plot.

```
attributes = ["PRICE", "INDUS", "NOX", "RM", "LSTAT"]

g = sns.PairGrid(df[attributes])
g = g.map_upper(sns.regplot, color="g")
g = g.map_lower(sns.kdeplot, cmap="Reds", shade=True,
shade_lowest=False)
g = g.map_diag(sns.kdeplot)
```

Output



Let's begin with the upper part:

- Price is negatively correlated with INDUS, NOX, and LSTAT; positively correlated with RM.
- There is a slightly non-linearity with LSTAT and PRICE
- There is like a straight line when the price is equal to 50. From the description of the dataset, PRICE has been truncated at the value of 50

Diagonal

- NOX seems to have two clusters, one around 0.5 and one around 0.85.

To check more about it, you can look at the lower part. The Multivariate Kernel Density is interesting in a sense it colors where most of the points are. The difference with the scatter plot draws a probability density, even though there is no point in the dataset for a given coordinate. When the color is stronger, it indicates a high concentration of point around this area.

If you check the multivariate density for INDUS and NOX, you can see the positive correlation and the two clusters. When the share of the industry is above 18, the nitric oxides concentration is above 0.6.

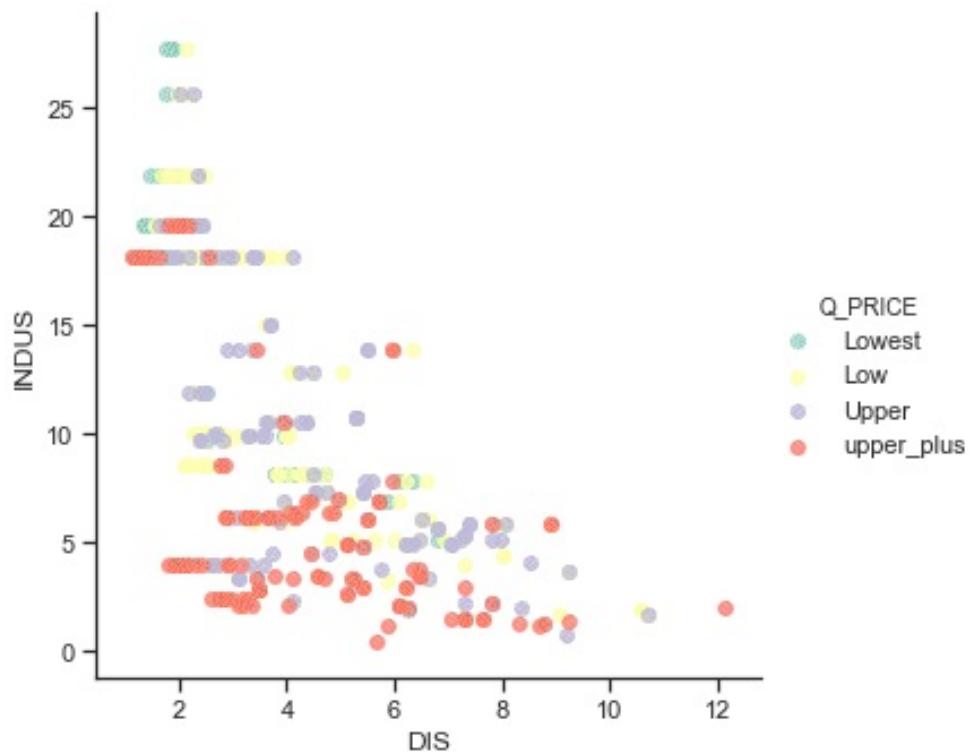
You can think about adding an interaction between INDUS and NOX in the linear regression.

Finally, you can use the second tools created by Google, Facets Deep Dive. The interface is divided up into four main sections. The central area in the center is a zoomable display of the data. On the top of the panel, there is the drop-down menu where you can change the arrangement of the data to controls faceting, positioning, and color. On the right, there is a detailed view of a specific row of data. It means you can click on any dot of data in the center visualization to see the detail about that particular data point.

During the data visualization step, you are interested in looking for the pairwise correlation between the independent variable on the price of the house. However, it involves at least three variables, and 3D plots are complicated to work with.

One way to tackle this problem is to create a categorical variable. That is, we can create a 2D plot a color the dot. You can split the variable PRICE into four categories, with each category is a quartile (i.e., 0.25, 0.5, 0.75). You call this new variable Q_PRICE.

```
## Check non linearity with important features
df['Q_PRICE'] = pd.qcut(df['PRICE'], 4, labels=["Lowest", "Low",
"Upper", "upper_plus"])
## Show non linearity between RM and LSTAT
ax = sns.lmplot(x="DIS", y="INDUS", hue="Q_PRICE", data=df, fit_reg=
= False, palette="Set3")
```



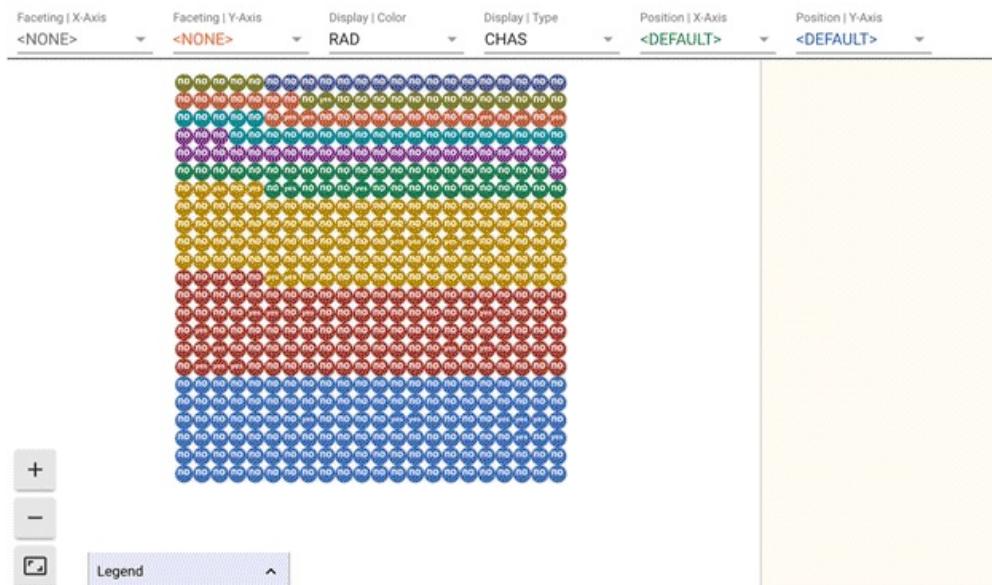
Facets Deep Dive

To open Deep Dive, you need to transform the data into a json format. Pandas as an object for that. You can use to_json after the Pandas dataset.

The first line of code handle the size of the dataset.

```
df['Q_PRICE'] = pd.qcut(df['PRICE'], 4, labels=["Lowest", "Low", "Upper", "upper_plus"])
sprite_size = 32 if len(df.index)>50000 else 64
jsonstr = df.to_json(orient='records')
```

The code below comes from Google GitHub. After you run the code, you should be able to see this:



```
# Display thde Dive visualization for this data
from IPython.core.display import display, HTML

# Create Facets template
HTML_TEMPLATE = """<link rel="import" href="/nbextensions/facets-dist/facets-jupyter.html">
    <facets-dive sprite-image-width="{sprite_size}" sprite-image-height="{sprite_size}" id="elem" height="600"></facets-dive>
```

```

<script>
    document.querySelector("#elem").data = {jsonstr};
</script>"""

# Load the json dataset and the sprite_size into the template
html = HTML_TEMPLATE.format(jsonstr=jsonstr,
sprite_size=sprite_size)

# Display the template
display(HTML(html))

```

You are interested to see if there is a connection between the industry rate, oxide concentration, distance to the job center and the price of the house.

For that, you first split the data by industry range and color with the price quartile:

- Select facetting X and choose INDUS.
- Select Display and choose DIS. It will color the dots with the quartile of the house price

here, darker colors mean the distance to the first job center is far.

So far, it shows again what you know, lower industry rate, higher price. Now you can look at the breakdown by INDUX, by NOX.

- Select facetting Y and choose NOX.

Now you can see the house far from the first job center have the lowest industry share and therefore the lowest oxide concentration. If you choose to display the type with Q_PRICE and zoom the lower-left corner, you can see what type of price it is.

You have another hint that the interaction between IND, NOX, and DIS can be good candidates to improve the model.

TensorFlow

In this section, you will estimate the linear classifier with TensorFlow estimators API. You will proceed as follow:

- Prepare the data
- Estimate a benchmark model: No interaction
- Estimate a model with interaction

Remember, the goal of machine learning is to minimize the error. In this case, the model with the lowest mean square error will win. The TensorFlow estimator automatically computes this metric.

Preparation data

In most of the case, you need to transform your data. That is why Facets Overview is fascinating. From the summary statistic, you saw there are outliers. Those values affect the estimates because they do not look like the population you are analyzing. Outliers usually biased the results. For instance, a positive outlier tends to overestimate the coefficient.

A good solution to tackle this problem is to standardize the variable. Standardization means a standard deviation of one and means of zero. The process of standardization involves two steps. First of all, it subtracts the mean value of the variable. Secondly, it divides by the variance so that the distribution has a unit variance

The library `sklearn` is helpful to standardize variables. You can use the module `preprocessing` with the object `scale` for this purpose.

You can use the function below to scale a dataset. Note that you don't scale the label column and categorical variables.

```

from sklearn import preprocessing
def standardize_data(df):
    X_scaled = preprocessing.scale(df[['CRIM', 'ZN', 'INDUS',
'NOX', 'RM', 'AGE', 'DIS', 'RAD',
'TAX', 'PTRATIO', 'B', 'LSTAT']])
    X_scaled_df = pd.DataFrame(X_scaled, columns = ['CRIM', 'ZN',
'INDUS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
'TAX', 'PTRATIO', 'B', 'LSTAT'])
    df_scale = pd.concat([X_scaled_df,
                          df['CHAS'],
                          df['PRICE']],axis=1, join='inner')
    return df_scale

```

You can use the function to construct the scaled train/test set.

```

df_train_scale = standardize_data(df_train)
df_test_scale = standardize_data(df_test)

```

Basic regression:Benchmark

First of all, you train and test a model without interaction. The purpose is to see the performance metric of the model.

The way to train the model is exactly as the tutorial on **High-level API**. You will use the TensorFlow estimator LinearRegressor.

As a reminder, you need to choose:

- the features to put in the model
- transform the features
- construct the linear regressor
- construct the input_fn function
- train the model
- test the model

You use all the variables in the dataset to train the model. In total, there are eleven continuous variables and one categorical variable

```

## Add features to the bucket:
### Define continuous list
CONTI_FEATURES = ['CRIM', 'ZN', 'INDUS', 'NOX', 'RM', 'AGE',
'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
CATE_FEATURES = ['CHAS']

```

You convert the features into a numeric column or categorical column

```

continuous_features = [tf.feature_column.numeric_column(k) for k in
CONTI_FEATURES]
#categorical_features =
tf.feature_column.categorical_column_with_hash_bucket(CATE_FEATURES,
, hash_bucket_size=1000)
categorical_features =
[tf.feature_column.categorical_column_with_vocabulary_list('CHAS',
['yes', 'no'])]

```

You create the model with the linearRegressor. You store the model in the folder train_Boston

```

model = tf.estimator.LinearRegressor(
    model_dir="train_Boston",
    feature_columns=categorical_features + continuous_features)

```

Output

```

INFO:tensorflow:Using default config.
INFO:tensorflow:Using config: {'_model_dir': 'train_Boston',
'_tf_random_seed': None, '_save_summary_steps': 100,
'_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,
'_session_config': None, '_keep_checkpoint_max': 5,
'_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps':
100, '_train_distribute': None, '_service': None, '_cluster_spec':
<tensorflow.python.training.server_lib.ClusterSpec object at
0x1a19e76ac8>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':
'', '_is_chief': True, '_num_ps_replicas': 0,
'_num_worker_replicas': 1}

```

Each column in the train or test data is converted into a Tensor with the the function get_input_fn

```

FEATURES = ['CRIM', 'ZN', 'INDUS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'CHAS']
LABEL= 'PRICE'
def get_input_fn(data_set, num_epochs=None, n_batch = 128,
shuffle=True):
    return tf.estimator.inputs.pandas_input_fn(
        x=pd.DataFrame({k: data_set[k].values for k in FEATURES}),
        y = pd.Series(data_set[LABEL].values),
        batch_size=n_batch,
        num_epochs=num_epochs,
        shuffle=shuffle)

```

You estimate the model on the train data.

```

model.train(input_fn=get_input_fn(df_train_scale,
                                   num_epochs=None,
                                   n_batch = 128,
                                   shuffle=False),
            steps=1000)

```

Output

```

INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow>Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into
train_Boston/model.ckpt.
INFO:tensorflow:loss = 56417.703, step = 1
INFO:tensorflow:global_step/sec: 144.457
INFO:tensorflow:loss = 76982.734, step = 101 (0.697 sec)
INFO:tensorflow:global_step/sec: 258.392
INFO:tensorflow:loss = 21246.334, step = 201 (0.383 sec)
INFO:tensorflow:global_step/sec: 227.998
INFO:tensorflow:loss = 30534.78, step = 301 (0.439 sec)
INFO:tensorflow:global_step/sec: 210.739
INFO:tensorflow:loss = 36794.5, step = 401 (0.477 sec)
INFO:tensorflow:global_step/sec: 234.237
INFO:tensorflow:loss = 8562.981, step = 501 (0.425 sec)
INFO:tensorflow:global_step/sec: 238.1
INFO:tensorflow:loss = 34465.08, step = 601 (0.420 sec)

```

```
INFO:tensorflow:global_step/sec: 237.934
INFO:tensorflow:loss = 12241.709, step = 701 (0.420 sec)
INFO:tensorflow:global_step/sec: 220.687
INFO:tensorflow:loss = 11019.228, step = 801 (0.453 sec)
INFO:tensorflow:global_step/sec: 232.702
INFO:tensorflow:loss = 24049.678, step = 901 (0.432 sec)
INFO:tensorflow:Saving checkpoints for 1000 into
train_Boston/model.ckpt.
INFO:tensorflow:Loss for final step: 23228.568.
```

```
<tensorflow.python.estimator.canned.linear.LinearRegressor at
0x1a19e76320>
```

At last, you estimate the performances of the model on the test set

```
model.evaluate(input_fn=get_input_fn(df_test_scale,
                                      num_epochs=1,
                                      n_batch = 128,
                                      shuffle=False),
                steps=1000)
```

Output

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2018-05-29-02:40:43
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from train_Boston/model.ckpt-
1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2018-05-29-02:40:43
INFO:tensorflow:Saving dict for global step 1000: average_loss =
86.89361, global_step = 1000, loss = 1650.9785

{'average_loss': 86.89361, 'global_step': 1000, 'loss': 1650.9785}
```

The loss of the model is 1650. This is the metric to beat in the next section

Improve the model: Interaction term

During the first part of the tutorial, you saw an interesting relationship between the variables. The different visualization techniques revealed that INDUS and NOS are linked together and turns to magnify the effect on the price. Not only the interaction between INDUS and NOS affects the price but also this effect is stronger when it interacts with DIS.

It is time to generalize this idea and see if you can improve the model prediction.

You need to add two new columns to each dataset set: train + test. For that, you create one function to compute the interaction term and another one to compute the triple interaction term. Each function produces a single column. After the new variables are created, you can concatenate them to the training dataset and test dataset.

First of all, you need to create a new variable for the interaction between INDUS and NOX.

The function below returns two dataframes, train and test, with the interaction between var_1 and var_2, in your case INDUS and NOX.

```
def interaction_term(var_1, var_2, name):
    t_train = df_train_scale[var_1]*df_train_scale[var_2]
    train = t_train.rename(name)
    t_test = df_test_scale[var_1]*df_test_scale[var_2]
    test = t_test.rename(name)
    return train, test
```

You store the two new columns

```
interation_ind_ns_train, interation_ind_ns_test=
interaction_term('INDUS', 'NOX', 'INDUS_NOS')
interation_ind_ns_train.shape
(325,)
```

Secondly, you create a second function to compute the triple interaction term.

```
def triple_interaction_term(var_1, var_2, var_3, name):
    t_train =
df_train_scale[var_1]*df_train_scale[var_2]*df_train_scale[var_3]
    train = t_train.rename(name)
    t_test =
df_test_scale[var_1]*df_test_scale[var_2]*df_test_scale[var_3]
    test = t_test.rename(name)
    return train, test
interation_ind_ns_dis_train, interation_ind_ns_dis_test=
triple_interaction_term('INDUS', 'NOX', 'DIS', 'INDUS_NOS_DIS')
```

Now that you have all columns needed, you can add them to train and test dataset. You name these two new dataframe:

- df_train_new
- df_test_new

```
df_train_new = pd.concat([df_train_scale,
                           interation_ind_ns_train,
                           interation_ind_ns_dis_train],
                           axis=1, join='inner')
df_test_new = pd.concat([df_test_scale,
                        interation_ind_ns_test,
                        interation_ind_ns_dis_test],
                        axis=1, join='inner')
df_train_new.head(5)
```

Output

	CRIM	ZN	INDUS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	CHAS	PRICE	INDUS_NOS
2	-0.275582	-0.47701	-0.464046	-0.162933	-0.188265	0.812916	0.105941	-0.661477	-0.616881	1.147718	0.444455	0.803221	no	34.7	0.075608
4	1.017983	-0.47701	0.992729	1.594192	-0.595967	0.987593	-0.907724	1.636106	1.502932	0.776192	-1.278797	1.504488	no	36.2	1.582601
5	-0.407050	-0.47701	-1.155868	-0.589166	1.036257	0.620414	-0.164461	-0.891235	-0.835358	-0.338387	0.444455	-1.025327	no	28.7	0.680999
8	-0.367794	-0.47701	-0.747795	-0.432591	-0.157121	0.798656	-0.346465	-0.201960	-0.616881	-0.524150	0.426162	1.182209	no	16.5	0.323489
9	1.832214	-0.47701	0.992729	1.246247	-2.699598	0.795092	-1.129861	1.636106	1.502932	0.776192	-0.779497	2.450575	no	18.9	1.237185

That is it; you can estimate the new model with the interaction terms and see how is the performance metric.

```
CONTI_FEATURES_NEW = ['CRIM', 'ZN', 'INDUS', 'NOX', 'RM', 'AGE',
```

```

'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT',
           'INDUS_NOS', 'INDUS_NOS_DIS']

### Define categorical list
continuous_features_new = [tf.feature_column.numeric_column(k) for
k in CONTI_FEATURES_NEW]
model = tf.estimator.LinearRegressor(
    model_dir="train_Boston_1",
    feature_columns= categorical_features +
continuous_features_new)

```

Output

```

INFO:tensorflow:Using default config.
INFO:tensorflow:Using config: {'_model_dir': 'train_Boston_1',
'_tf_random_seed': None, '_save_summary_steps': 100,
'_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,
'_session_config': None, '_keep_checkpoint_max': 5,
'_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps': 100,
'_train_distribute': None, '_service': None, '_cluster_spec':
<tensorflow.python.training.server_lib.ClusterSpec object at
0x1a1a5d5860>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':
'', '_is_chief': True, '_num_ps_replicas': 0,
'_num_worker_replicas': 1}

```

CODE

```

FEATURES = ['CRIM', 'ZN', 'INDUS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'INDUS_NOS',
'INDUS_NOS_DIS', 'CHAS']
LABEL= 'PRICE'
def get_input_fn(data_set, num_epochs=None, n_batch = 128,
shuffle=True):
    return tf.estimator.inputs.pandas_input_fn(
        x=pd.DataFrame({k: data_set[k].values for k in FEATURES}),
        y = pd.Series(data_set[LABEL].values),
        batch_size=n_batch,
        num_epochs=num_epochs,
        shuffle=shuffle)

model.train(input_fn=get_input_fn(df_train_new,
                                   num_epochs=None,
                                   n_batch = 128,

```



```
steps=1000)
```

Output

```
INFO:tensorflow:Calling model_fn.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow:Starting evaluation at 2018-05-29-02:41:14  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Restoring parameters from  
train_Boston_1/model.ckpt-1000  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Finished evaluation at 2018-05-29-02:41:14  
INFO:tensorflow:Saving dict for global step 1000: average_loss =  
79.78876, global_step = 1000, loss = 1515.9863  
  
{'average_loss': 79.78876, 'global_step': 1000, 'loss': 1515.9863}
```

The new loss is 1515. Just by adding two new variables, you were able to decrease the loss. It means you can make a better prediction than with the benchmark model.

Chapter 15: Linear Classifier in TensorFlow

What is Linear Classifier?

The two most common supervised learning tasks are linear regression and linear classifier. Linear regression predicts a value while the linear classifier predicts a class. This tutorial is focused on Linear Classifier.

Classification problems represent roughly 80 percent of the machine learning task. Classification aims at predicting the probability of each class given a set of inputs. The label (i.e., the dependent variable) is a discrete value, called a class.

1. If the label has only two classes, the learning algorithm is a binary classifier.
2. Multiclass classifier tackles labels with more than two classes.

For instance, a typical binary classification problem is to predict the likelihood a customer makes a second purchase. Predict the type of animal displayed on a picture is multiclass classification problem since there are more than two varieties of animal existing.

The theoretical part of this tutorial puts primary focus on the binary class. You will learn more about the multiclass output function in a future tutorial.

How Binary classifier works?

You learned in the previous tutorial that a function is composed of two kind of variables, a dependent variable and a set of features (independent variables). In the linear regression, a dependent variable is a real number without range. The primary objective is to predict its value by minimizing the mean squared error.

For a binary task, the label can have had two possible integer values. In most case, it is either [0,1] or [1,2]. For instance, the objective is to predict whether a customer will buy a product or not. The label is defined as follow:

- $Y = 1$ (customer purchased the product)
- $Y = 0$ (customer does not purchase the product)

The model uses the features X to classify each customer in the most likely class he belongs to, namely, potential buyer or not.

The probability of success is computed with **logistic regression**. The algorithm will compute a probability based on the feature X and predicts a success when this probability is above 50 percent. More formally, the probability is calculated as follow:

$$P(Y = 1|x) = \frac{1}{1 + \exp(-(\theta^T x + b))}$$

where θ is the set of weights, the features and b the bias.

The function can be decomposed into two parts:

- The linear model
- The logistic function

Linear model

You are already familiar with the way the weights are computed. Weights are computed using a dot product: $\theta^T x + b$ that is $\sum_{i=0}^n x_i w_i + b$. Y is a linear function of all the features x_i . If the model does not have features, the prediction is equal to the bias, b.

The weights indicate the direction of the correlation between the features x_i and the label y. A positive correlation increases the probability of the positive class while a negative correlation leads the probability closer to 0, (i.e., negative class).

The linear model returns only real number, which is inconsistent with the probability measure of range [0,1]. The logistic function is required to convert the linear model output to a probability,

Logistic function

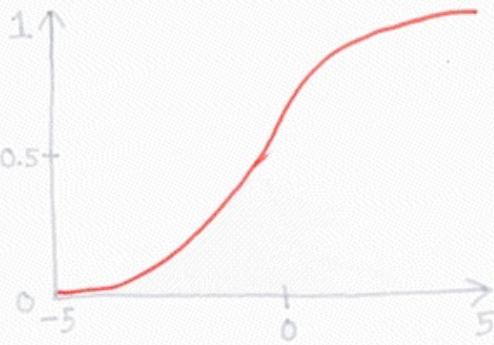
The logistic function, or sigmoid function, has an S-shape and the output of this function is always between 0 and 1.

$$\frac{1}{1 + \exp(-t)}$$

LOGISTIC SIGMOID FUNCTION

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

Chris Albon



It is easy to substitute the output of the linear regression into the sigmoid function. It results in a new number with a probability between 0 and 1.

The classifier can transform the probability into a class

- Values between 0 to 0.49 become class 0
- Values between 0.5 to 1 become class 1

How to Measure the performance of Linear Classifier?

Accuracy

The overall performance of a classifier is measured with the accuracy metric. Accuracy collects all the correct values divided by the total number of observations. For instance, an accuracy value of 80 percent means the model is correct in 80 percent of the cases.

ACCURACY

$$A_{cc} = \frac{1}{n} \sum 1(\hat{y}_i = y_i)$$

number of observations

Predicted y

True y

Indicator function

You can note a shortcoming with this metric, especially for imbalance class. An imbalance dataset occurs when the number of observations per group is not equal. Let's say; you try to classify a rare event with a logistic function. Imagine the classifier tries to estimate the death of a patient following a disease. In the data, 5 percent of the patients pass away. You can train a classifier to predict the number of death and use the accuracy metric to evaluate the performances. If the classifier predicts 0 death for the entire dataset, it will be correct in 95 percent of

the case.

Confusion matrix

A better way to assess the performance of a classifier is to look at the confusion matrix.

		Predicted	
		TRUE	FALSE
Actual	TRUE	TP	FN
	FALSE	FP	TN

The confusion matrix visualizes the accuracy of a classifier by comparing the actual and predicted classes. The binary confusion matrix is composed of squares:

- TP: True Positive: Predicted values correctly predicted as actual positive
- FP: Predicted values incorrectly predicted an actual positive. i.e., Negative values predicted as positive
- FN: False Negative: Positive values predicted as negative
- TN: True Negative: Predicted values correctly predicted as actual negative

From the confusion matrix, it is easy to compare the actual class and predicted class.

Precision and Sensitivity

The confusion matrix provides a good insight into the true positive and false positive. In some case, it is preferable to have a more concise metric.

Precision

The precision metric shows the accuracy of the positive class. It measures how likely the prediction of the positive class is correct.

$$Precision = \frac{TP}{TP + FP}$$

The maximum score is 1 when the classifier perfectly classifies all the positive values. Precision alone is not very helpful because it ignores the negative class. The metric is usually paired with Recall metric. Recall is also called sensitivity or true positive rate.

Sensitivity

Sensitivity computes the ratio of positive classes correctly detected. This metric gives how good the model is to recognize a positive class.

$$Recall = \frac{TP}{TP + FN}$$

Linear Classifier with TensorFlow

For this tutorial, we will use the census dataset. The purpose is to use the variables in the census dataset to predict the income level. Note that the income is a binary variable

- with a value of 1 if the income > 50k
- 0 if income < 50k.

This variable is your label

This dataset includes eight categorical variables:

- workplace
- education
- marital
- occupation
- relationship
- race
- sex
- native_country

moreover, six continuous variables:

- age
- fnlwgt
- education_num
- capital_gain
- capital_loss
- hours_week

Through this example, you will understand how to train a linear classifier with

TensorFlow estimator and how to improve the accuracy metric.

We will proceed as follow:

- Step 1) Import the data
- Step 2) Data Conversion
- Step 3) Train the classifier
- Step 4) Improve the model
- Step 5) Hyperparameter:Lasso & Ridge

Step 1) Import the data

You first import the libraries used during the tutorial.

```
import tensorflow as tf  
import pandas as pd
```

Next, you import the data from the archive of UCI and defines the columns names. You will use the COLUMNS to name the columns in a pandas data frame.

Note that you will train the classifier using a Pandas dataframe.

```
## Define path data  
COLUMNS = ['age', 'workclass', 'fnlwgt', 'education',  
'education_num', 'marital',  
          'occupation', 'relationship', 'race', 'sex',  
'capital_gain', 'capital_loss',  
          'hours_week', 'native_country', 'label']  
PATH = "https://archive.ics.uci.edu/ml/machine-learning-  
databases/adult/adult.data"  
PATH_test = "https://archive.ics.uci.edu/ml/machine-learning-  
databases/adult/adult.test"
```

The data stored online are already divided between a train set and test set.

```
df_train = pd.read_csv(PATH, skipinitialspace=True, names =  
COLUMNS, index_col=False)
```

```
df_test = pd.read_csv(PATH_test, skiprows = 1,  
skipinitialspace=True, names = COLUMNS, index_col=False)
```

The train set contains 32,561 observations and the test set 16,281

```
print(df_train.shape, df_test.shape)  
print(df_train.dtypes)  
(32561, 15) (16281, 15)  
age           int64  
workclass     object  
fnlwgt        int64  
education     object  
education_num int64  
marital       object  
occupation    object  
relationship   object  
race          object  
sex           object  
capital_gain  int64  
capital_loss  int64  
hours_week    int64  
native_country object  
label         object  
dtype: object
```

Tensorflow requires a Boolean value to train the classifier. You need to cast the values from string to integer. The label is stored as an object, however, you need to convert it into a numeric value. The code below creates a dictionary with the values to convert and loop over the column item. Note that you perform this operation twice, one for the train test, one for the test set

```
label = {'<=50K': 0, '>50K': 1}  
df_train.label = [label[item] for item in df_train.label]  
label_t = {'<=50K.': 0, '>50K.': 1}  
df_test.label = [label_t[item] for item in df_test.label]
```

In the train data, there are 24,720 incomes lower than 50k and 7841 above. The ratio is almost the same for the test set. Please refer this tutorial on Facets for more.

```
print(df_train["label"].value_counts())
```

```
### The model will be correct in atleast 70% of the case
print(df_test["label"].value_counts())
## Unbalanced label
print(df_train.dtypes)
0    24720
1    7841
Name: label, dtype: int64
0    12435
1    3846
Name: label, dtype: int64
age            int64
workclass      object
fnlwgt          int64
education      object
education_num   int64
marital         object
occupation     object
relationship   object
race            object
sex             object
capital_gain   int64
capital_loss   int64
hours_week     int64
native_country object
label           int64
dtype: object
```

Step 2) Data Conversion

A few steps are required before you train a linear classifier with Tensorflow. You need to prepare the features to include in the model. In the benchmark regression, you will use the original data without applying any transformation.

The estimator needs to have a list of features to train the model. Hence, the column's data requires to be converted into a tensor.

A good practice is to define two lists of features based on their type and then pass them in the feature_columns of the estimator.

You will begin by converting continuous features, then define a bucket with the

categorical data.

The features of the dataset have two formats:

- Integer
- Object

Each feature is listed in the next two variables as per their types.

```
## Add features to the bucket:  
### Define continuous list  
CONTI_FEATURES = ['age', 'fnlwgt', 'capital_gain', 'education_num',  
'capital_loss', 'hours_week']  
### Define the categorical list  
CATE_FEATURES = ['workclass', 'education', 'marital', 'occupation',  
'relationship', 'race', 'sex', 'native_country']
```

The feature_column is equipped with an object numeric_column to help in the transformation of the continuous variables into tensor. In the code below, you convert all the variables from CONTI_FEATURES into a tensor with a numeric value. This is compulsory to construct the model. All the independent variables need to be converted into the proper type of tensor.

Below we write a code to let you see what is happening behind feature_column.numeric_column. We will print the converted value for age It is for explanatory purpose, hence there is no need to understand the python code. You can refer to the official documentation to understand the codes.

```
def print_transformation(feature = "age", continuous = True, size = 2):  
    #X = fc.numeric_column(feature)  
    ## Create feature name  
    feature_names = [  
        feature]  
  
    ## Create dict with the data  
    d = dict(zip(feature_names, [df_train[feature]]))  
  
    ## Convert age
```

```

if continuous == True:
    c = tf.feature_column.numeric_column(feature)
    feature_columns = [c]
else:
    c =
tf.feature_column.categorical_column_with_hash_bucket(feature,
hash_bucket_size=size)
    c_indicator = tf.feature_column.indicator_column(c)
    feature_columns = [c_indicator]

## Use input_layer to print the value
input_layer = tf.feature_column.input_layer(
    features=d,
    feature_columns=feature_columns
)
## Create lookup table
zero = tf.constant(0, dtype=tf.float32)
where = tf.not_equal(input_layer, zero)
## Return lookup tble
indices = tf.where(where)
values = tf.gather_nd(input_layer, indices)
## Initiate graph
sess = tf.Session()
## Print value
print(sess.run(input_layer))
print_transformation(feature = "age", continuous = True)
[[39.]
[50.]
[38.]
...
[58.]
[22.]
[52.]]

```

The values are exactly the same as in df_train

```
continuous_features = [tf.feature_column.numeric_column(k) for k in
CONTI_FEATURES]
```

According to TensorFlow documentation, there are different ways to convert categorical data. If the vocabulary list of a feature is known and does not have plenty of values, it is possible to create the categorical column with

categorical_column_with_vocabulary_list. It will assign to all unique vocabulary list an ID.

For instance, if a variable status has three distinct values:

- Husband
- Wife
- Single

Then three ID will be attributed. For instance, Husband will have the ID 1, Wife the ID 2 and so on.

For illustration purpose, you can use this code to convert an object variable to a categorical column in TensorFlow.

The feature sex can only have two value: male or female. When we will convert the feature sex, Tensorflow will create 2 new columns, one for male and one for female. If the sex is equal to male, then the new column male will be equal to 1 and female to 0. This example is displayed in the table below:

rows	sex	after transformation	male	female
1	male	=>	1	0
2	male	=>	1	0
3	female	=>	0	1

In tensorflow:

```
print_transformation(feature = "sex", continuous = False, size = 2)
[[1. 0.]
 [1. 0.]
 [1. 0.]]
```

```

...
[0. 1.]
[1. 0.]
[0. 1.]]
```

```

relationship =
tf.feature_column.categorical_column_with_vocabulary_list(
    'relationship', [
        'Husband', 'Not-in-family', 'Wife', 'Own-child',
    'Unmarried',
    'Other-relative'])
```

Below, we added Python code to print the encoding. Again, you don't need to understand the code, the purpose is to see the transformation

However, a faster way to transform the data is to use the method `categorical_column_with_hash_bucket`. Altering string variables in a sparse matrix will be useful. A sparse matrix is a matrix with mostly zero. The method takes care of everything. You only need to specify the number of buckets and the key column. The number of buckets is the maximum amount of groups that Tensorflow can create. The key column is simply the name of the column to convert.

In the code below, you create a loop over all the categorical features.

```

categorical_features =
[tf.feature_column.categorical_column_with_hash_bucket(k,
hash_bucket_size=1000) for k in CATE_FEATURES]
```

Step 3) Train the Classifier

TensorFlow currently provides an estimator for the linear regression and linear classification.

- Linear regression: `LinearRegressor`
- Linear classification: `LinearClassifier`

The syntax of the linear classifier is the same as in the tutorial on linear regression except for one argument, n_class. You need to define the feature column, the model directory and, compare with the linear regressor; you have to define the number of class. For a logit regression, it the number of class is equal to 2.

The model will compute the weights of the columns contained in continuous_features and categorical_features.

```
model = tf.estimator.LinearClassifier(  
    n_classes = 2,  
    model_dir="ongoing/train",  
    feature_columns=categorical_features+ continuous_features)
```

OUTPUT:

```
INFO:tensorflow:Using default config.  
INFO:tensorflow:Using config: {'_model_dir': 'ongoing/train',  
'_tf_random_seed': None, '_save_summary_steps': 100,  
'_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,  
'_session_config': None, '_keep_checkpoint_max': 5,  
'_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps':  
100, '_train_distribute': None, '_service': None, '_cluster_spec':  
<tensorflow.python.training.server_lib.ClusterSpec object at  
0x181f24c898>, '_task_type': 'worker', '_task_id': 0,  
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':  
'', '_is_chief': True, '_num_ps_replicas': 0,  
'_num_worker_replicas': 1}
```

Now that the classifier is defined, you can create the input function. The method is the same as in the linear regressor tutorial. Here, you use a batch size of 128 and you shuffle the data.

```
FEATURES = ['age', 'workclass', 'fnlwgt', 'education',  
'education_num', 'marital', 'occupation', 'relationship', 'race',  
'sex', 'capital_gain', 'capital_loss', 'hours_week',  
'native_country']  
LABEL= 'label'  
def get_input_fn(data_set, num_epochs=None, n_batch = 128,  
shuffle=True):
```

```
return tf.estimator.inputs.pandas_input_fn(  
    x=pd.DataFrame({k: data_set[k].values for k in FEATURES}),  
    y = pd.Series(data_set[LABEL].values),  
    batch_size=n_batch,  
    num_epochs=num_epochs,  
    shuffle=shuffle)
```

You create a function with the arguments required by the linear estimator, i.e., number of epochs, number of batches and shuffle the dataset or note. Since you use the Pandas method to pass the data into the model, you need to define the X variables as a pandas data frame. Note that you loop over all the data stored in FEATURES.

Let's train the model with the object model.train. You use the function previously defined to feed the model with the appropriate values. Note that you set the batch size to 128 and the number of epochs to None. The model will be trained over a thousand steps.

```
model.train(input_fn=get_input_fn(df_train,  
                                  num_epochs=None,  
                                  n_batch = 128,  
                                  shuffle=False),  
            steps=1000)
```

```
INFO:tensorflow:Calling model_fn.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow>Create CheckpointSaverHook.  
INFO:tensorflow: Graph was finalized.  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Saving checkpoints for 1 into  
ongoing/train/model.ckpt.  
INFO:tensorflow:loss = 88.722855, step = 1  
INFO:tensorflow:global_step/sec: 65.8282  
INFO:tensorflow:loss = 52583.64, step = 101 (1.528 sec)  
INFO:tensorflow:global_step/sec: 118.386  
INFO:tensorflow:loss = 25203.816, step = 201 (0.837 sec)  
INFO:tensorflow:global_step/sec: 110.542  
INFO:tensorflow:loss = 54924.312, step = 301 (0.905 sec)  
INFO:tensorflow:global_step/sec: 199.03  
INFO:tensorflow:loss = 68509.31, step = 401 (0.502 sec)
```

```
INFO:tensorflow:global_step/sec: 167.488
INFO:tensorflow:loss = 9151.754, step = 501 (0.599 sec)
INFO:tensorflow:global_step/sec: 220.155
INFO:tensorflow:loss = 34576.06, step = 601 (0.453 sec)
INFO:tensorflow:global_step/sec: 199.016
INFO:tensorflow:loss = 36047.117, step = 701 (0.503 sec)
INFO:tensorflow:global_step/sec: 197.531
INFO:tensorflow:loss = 22608.148, step = 801 (0.505 sec)
INFO:tensorflow:global_step/sec: 208.479
INFO:tensorflow:loss = 22201.918, step = 901 (0.479 sec)
INFO:tensorflow:Saving checkpoints for 1000 into
ongoing/train/model.ckpt.
INFO:tensorflow:Loss for final step: 5444.363.

<tensorflow.python.estimator.canned.linear.LinearClassifier at
0x181f223630>
```

Note that the loss decreased subsequently during the last 100 steps, i.e., from 901 to 1000.

The final loss after one thousand iterations is 5444. You can estimate your model on the test set and see the performance. To evaluate the performance of your model, you need to use the object `evaluate`. You feed the model with the test set and set the number of epochs to 1, i.e., the data will go to the model only one time.

```
model.evaluate(input_fn=get_input_fn(df_test,
                                      num_epochs=1,
                                      n_batch = 128,
                                      shuffle=False),
                      steps=1000)
```

```
INFO:tensorflow:Calling model_fn.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect
PR-AUCs; please switch to "careful_interpolation" instead.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect
PR-AUCs; please switch to "careful_interpolation" instead.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2018-06-02-08:28:22
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from ongoing/train/model.ckpt-
1000
```

```
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Evaluation [100/1000]  
INFO:tensorflow:Finished evaluation at 2018-06-02-08:28:23  
INFO:tensorflow:Saving dict for global step 1000: accuracy =  
0.7615626, accuracy_baseline = 0.76377374, auc = 0.63300294,  
auc_precision_recall = 0.50891197, average_loss = 47.12155,  
global_step = 1000, label/mean = 0.23622628, loss = 5993.6406,  
precision = 0.49401596, prediction/mean = 0.18454961, recall =  
0.38637546  
  
{'accuracy': 0.7615626,  
 'accuracy_baseline': 0.76377374,  
 'auc': 0.63300294,  
 'auc_precision_recall': 0.50891197,  
 'average_loss': 47.12155,  
 'global_step': 1000,  
 'label/mean': 0.23622628,  
 'loss': 5993.6406,  
 'precision': 0.49401596,  
 'prediction/mean': 0.18454961,  
 'recall': 0.38637546}
```

TensorFlow returns all the metrics you learnt in the theoretical part. Without surprise, the accuracy is large due to the unbalanced label. Actually, the model performs slightly better than a random guess. Imagine the model predict all household with income lower than 50K, then the model has an accuracy of 70 percent. On a closer analysis, you can see the prediction and recall are quite low.

Step 4) Improve the model

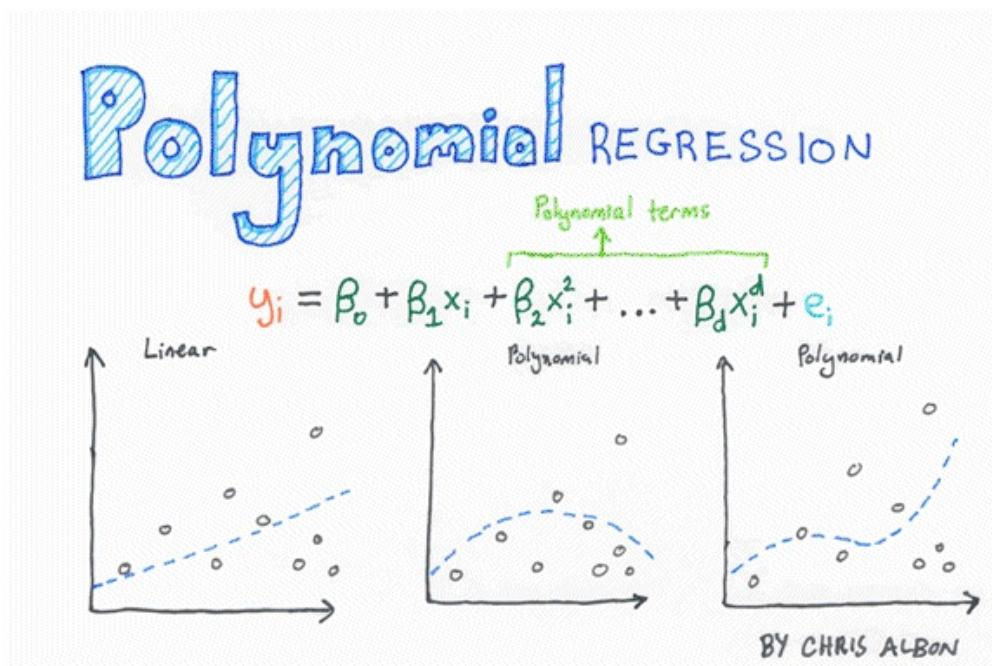
Now that you have a benchmark model, you can try to improve it, that is, increase the accuracy. In the previous tutorial, you learned how to improve the prediction power with an interaction term. In this tutorial, you will revisit this idea by adding a polynomial term to the regression.

Polynomial regression is instrumental when there is non-linearity in the data. There are two ways to capture non-linearity in the data.

- Add polynomial term
- Bucketize the continuous variable into a categorical variable

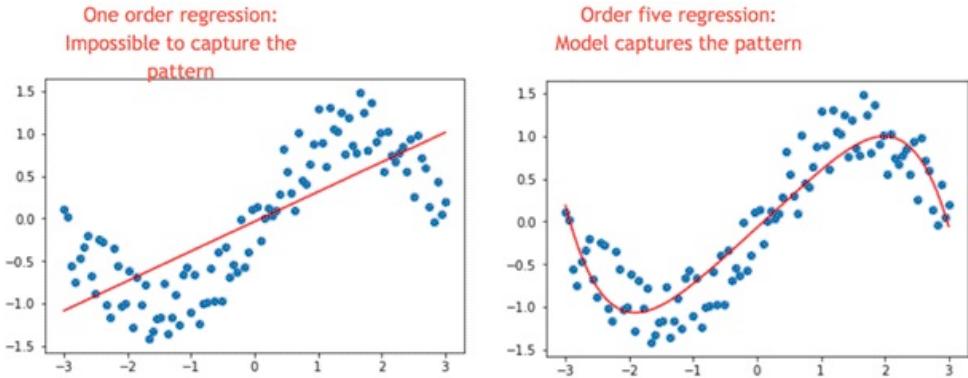
Polynomial term

From the picture below, you can see what a polynomial regression is. It is an equation with X variables with different power. A second-degree polynomial regression has two variables, X and X squared. Third degree has three variables, X, X², and X³



Below, we constructed a graph with two variables, X and Y. It is obvious the relationship is not linear. If we add a linear regression, we can see the model is unable to capture the pattern (left picture).

Now, look at the left picture from the picture below, we added five-term to the regression (that is $y=x+x^2+x^3+x^4+x^5$). The model now captures way better the pattern. This is the power of polynomial regression.



Let's go back to our example. Age is not in a linear relationship with income. Early age might have a flat income close to zero because children or young people do not work. Then it increases in working age and decreases during retirement. It is typically an Inversed-U shape. One way to capture this pattern is by adding a power two to the regression.

Let's see if it increases the accuracy.

You need to add this new feature to the dataset and in the list of continuous feature.

You add the new variable in the train and test dataset, so it is more convenient to write a function.

```
def square_var(df_t, df_te, var_name = 'age'):
    df_t['new'] = df_t[var_name].pow(2)
    df_te['new'] = df_te[var_name].pow(2)
    return df_t, df_te
```

The function has 3 arguments:

- df_t: define the training set
- df_te: define the test set
- var_name = 'age': Define the variable to transform

You can use the object pow(2) to square the variable age. Note that the new

variable is named 'new'

Now that the function square_var is written, you can create the new datasets.

```
df_train_new, df_test_new = square_var(df_train, df_test, var_name = 'age')
```

As you can see, the new dataset has one more feature.

```
print(df_train_new.shape, df_test_new.shape)
(32561, 16) (16281, 16)
```

The square variable is called new in the dataset. You need to add it to the list of continuous features.

```
CONTI_FEATURES_NEW = ['age', 'fnlwgt', 'capital_gain',
'education_num', 'capital_loss', 'hours_week', 'new']
continuous_features_new = [tf.feature_column.numeric_column(k) for
k in CONTI_FEATURES_NEW]
```

Note that you changed the directory of the Graph. You can't train different models in the same directory. It means, you need to change the path of the argument model_dir. If you don't TensorFlow will throw an error.

```
model_1 = tf.estimator.LinearClassifier(
    model_dir="ongoing/train1",
    feature_columns=categorical_features+ continuous_features_new)
```

```
INFO:tensorflow:Using default config.
INFO:tensorflow:Using config: {'_model_dir': 'ongoing/train1',
'_tf_random_seed': None, '_save_summary_steps': 100,
'_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,
'_session_config': None, '_keep_checkpoint_max': 5,
'_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps': 100,
'_train_distribute': None, '_service': None, '_cluster_spec': <tensorflow.python.training.server_lib.ClusterSpec object at 0x1820f04b70>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master': '',
'_is_chief': True, '_num_ps_replicas': 0,
'_num_worker_replicas': 1}
FEATURES_NEW = ['age', 'workclass', 'fnlwgt', 'education',
```

```
'education_num', 'marital', 'occupation', 'relationship', 'race',
'sex', 'capital_gain', 'capital_loss', 'hours_week',
'native_country', 'new']
def get_input_fn(data_set, num_epochs=None, n_batch = 128,
shuffle=True):
    return tf.estimator.inputs.pandas_input_fn(
        x=pd.DataFrame({k: data_set[k].values for k in
FEATURES_NEW}),
        y = pd.Series(data_set[LABEL].values),
        batch_size=n_batch,
        num_epochs=num_epochs,
        shuffle=shuffle)
```

Now that the classifier is designed with the new dataset, you can train and evaluate the model.

```
model_1.train(input_fn=get_input_fn(df_train,
                                     num_epochs=None,
                                     n_batch = 128,
                                     shuffle=False),
               steps=1000)
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow>Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into
ongoing/train1/model.ckpt.
INFO:tensorflow:loss = 88.722855, step = 1
INFO:tensorflow:global_step/sec: 81.487
INFO:tensorflow:loss = 70077.66, step = 101 (1.228 sec)
INFO:tensorflow:global_step/sec: 111.169
INFO:tensorflow:loss = 49522.082, step = 201 (0.899 sec)
INFO:tensorflow:global_step/sec: 128.91
INFO:tensorflow:loss = 107120.57, step = 301 (0.776 sec)
INFO:tensorflow:global_step/sec: 132.546
INFO:tensorflow:loss = 12814.152, step = 401 (0.755 sec)
INFO:tensorflow:global_step/sec: 162.194
INFO:tensorflow:loss = 19573.898, step = 501 (0.617 sec)
INFO:tensorflow:global_step/sec: 204.852
INFO:tensorflow:loss = 26381.986, step = 601 (0.488 sec)
INFO:tensorflow:global_step/sec: 188.923
```

```
INFO:tensorflow:loss = 23417.719, step = 701 (0.529 sec)
INFO:tensorflow:global_step/sec: 192.041
INFO:tensorflow:loss = 23946.049, step = 801 (0.521 sec)
INFO:tensorflow:global_step/sec: 197.025
INFO:tensorflow:loss = 3309.5786, step = 901 (0.507 sec)
INFO:tensorflow:Saving checkpoints for 1000 into
ongoing/train1/model.ckpt.
INFO:tensorflow:Loss for final step: 28861.898.
```

```
<tensorflow.python.estimator.canned.linear.LinearClassifier at
0x1820f04c88>
```

```
model_1.evaluate(input_fn=get_input_fn(df_test_new,
                                         num_epochs=1,
                                         n_batch = 128,
                                         shuffle=False),
                  steps=1000)
```

```
INFO:tensorflow:Calling model_fn.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect
PR-AUCs; please switch to "careful_interpolation" instead.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect
PR-AUCs; please switch to "careful_interpolation" instead.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2018-06-02-08:28:37
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from
ongoing/train1/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Evaluation [100/1000]
INFO:tensorflow:Finished evaluation at 2018-06-02-08:28:39
INFO:tensorflow:Saving dict for global step 1000: accuracy =
0.7944229, accuracy_baseline = 0.76377374, auc = 0.6093755,
auc_precision_recall = 0.54885805, average_loss = 111.0046,
global_step = 1000, label/mean = 0.23622628, loss = 14119.265,
precision = 0.6682401, prediction/mean = 0.09116262, recall =
0.2576703
```

```
{'accuracy': 0.7944229,
 'accuracy_baseline': 0.76377374,
 'auc': 0.6093755,
 'auc_precision_recall': 0.54885805,
 'average_loss': 111.0046,
 'global_step': 1000,
```

```
'label/mean': 0.23622628,  
'loss': 14119.265,  
'precision': 0.6682401,  
'prediction/mean': 0.09116262,  
'recall': 0.2576703}
```

The squared variable improved the accuracy from 0.76 to 0.79. Let's see if you can do better by combining bucketization and interaction term together.

Bucketization and interaction

As you saw before, a linear classifier is unable to capture the age-income pattern correctly. That is because it learns a single weight for each feature. To make it easier for the classifier, one thing you can do is bucket the feature. Bucketing transforms a numeric feature into several certain ones based on the range it falls into, and each of these new features indicates whether a person's age falls within that range.

With these new features, the linear model can capture the relationship by learning different weights for each bucket.

In TensorFlow, it is done with `bucketized_column`. You need to add the range of values in the boundaries.

```
age = tf.feature_column.numeric_column('age')  
age_buckets = tf.feature_column.bucketized_column(  
    age, boundaries=[18, 25, 30, 35, 40, 45, 50, 55, 60, 65])
```

You already know age is non-linear with income. Another way to improve the model is through interaction. In the word of TensorFlow, it is feature crossing. Feature crossing is a way to create new features that are combinations of existing ones, which can be helpful for a linear classifier that can't model interactions between features.

You can break down age with another feature like education. That is is, some

groups are likely to have a high income and others low (Think about the Ph.D. student).

```
education_x_occupation = [tf.feature_column.crossed_column(  
    ['education', 'occupation'], hash_bucket_size=1000)]  
age_buckets_x_education_x_occupation =  
[tf.feature_column.crossed_column(  
    [age_buckets, 'education', 'occupation'],  
    hash_bucket_size=1000)]
```

To create a cross feature column, you use crossed_column with the variables to cross in a bracket. The hash_bucket_size indicates the maximum crossing possibilities. To create interaction between variables (at least one variable needs to be categorical), you can use tf.feature_column.crossed_column. To use this object, you need to add in square bracket the variable to interact and a second argument, the bucket size. The bucket size is the maximum number of group possible within a variable. Here you set it at 1000 as you do not know the exact number of groups

age_buckets needs to be squared before to add it to the feature columns. You also add the new features to the features columns and prepare the estimator

```
base_columns = [  
    age_buckets,  
]  
  
model_imp = tf.estimator.LinearClassifier(  
    model_dir="ongoing/train3",  
  
feature_columns=categorical_features+base_columns+education_x_occupation+age_buckets_x_education_x_occupation)
```

OUTPUT

```
INFO:tensorflow:Using default config.  
INFO:tensorflow:Using config: {'_model_dir': 'ongoing/train3',  
'_tf_random_seed': None, '_save_summary_steps': 100,  
'_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,  
'_session_config': None, '_keep_checkpoint_max': 5,
```

```
'_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps': 100, '_train_distribute': None, '_service': None, '_cluster_spec': <tensorflow.python.training.server_lib.ClusterSpec object at 0x1823021be0>, '_task_type': 'worker', '_task_id': 0, '_global_id_in_cluster': 0, '_master': '', '_evaluation_master': '', '_is_chief': True, '_num_ps_replicas': 0, '_num_worker_replicas': 1}
```

```
FEATURES_imp = ['age', 'workclass', 'education', 'education_num', 'marital', 'occupation', 'relationship', 'race', 'sex', 'native_country', 'new']

def get_input_fn(data_set, num_epochs=None, n_batch = 128, shuffle=True):
    return tf.estimator.inputs.pandas_input_fn(
        x=pd.DataFrame({k: data_set[k].values for k in FEATURES_imp}),
        y = pd.Series(data_set[LABEL].values),
        batch_size=n_batch,
        num_epochs=num_epochs,
        shuffle=shuffle)
```

You are ready to estimate the new model and see if it improves the accuracy.

```
model_imp.train(input_fn=get_input_fn(df_train_new,
                                         num_epochs=None,
                                         n_batch = 128,
                                         shuffle=False),
                                         steps=1000)
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow>Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into
ongoing/train3/model.ckpt.
INFO:tensorflow:loss = 88.722855, step = 1
INFO:tensorflow:global_step/sec: 94.969
INFO:tensorflow:loss = 50.334488, step = 101 (1.054 sec)
INFO:tensorflow:global_step/sec: 242.342
INFO:tensorflow:loss = 56.153225, step = 201 (0.414 sec)
```

```
INFO:tensorflow:global_step/sec: 213.686
INFO:tensorflow:loss = 45.792007, step = 301 (0.470 sec)
INFO:tensorflow:global_step/sec: 174.084
INFO:tensorflow:loss = 37.485672, step = 401 (0.572 sec)
INFO:tensorflow:global_step/sec: 191.78
INFO:tensorflow:loss = 56.48449, step = 501 (0.524 sec)
INFO:tensorflow:global_step/sec: 163.436
INFO:tensorflow:loss = 32.528934, step = 601 (0.612 sec)
INFO:tensorflow:global_step/sec: 164.347
INFO:tensorflow:loss = 37.438057, step = 701 (0.607 sec)
INFO:tensorflow:global_step/sec: 154.274
INFO:tensorflow:loss = 61.1075, step = 801 (0.647 sec)
INFO:tensorflow:global_step/sec: 189.14
INFO:tensorflow:loss = 44.69645, step = 901 (0.531 sec)
INFO:tensorflow:Saving checkpoints for 1000 into
ongoing/train3/model.ckpt.
INFO:tensorflow:Loss for final step: 44.18133.
```

```
<tensorflow.python.estimator.canned.linear.LinearClassifier at
0x1823021cf8>
```

```
model_imp.evaluate(input_fn=get_input_fn(df_test_new,
                                         num_epochs=1,
                                         n_batch = 128,
                                         shuffle=False),
                     steps=1000)
```

```
INFO:tensorflow:Calling model_fn.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect
PR-AUCs; please switch to "careful_interpolation" instead.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect
PR-AUCs; please switch to "careful_interpolation" instead.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2018-06-02-08:28:52
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from
ongoing/train3/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Evaluation [100/1000]
INFO:tensorflow:Finished evaluation at 2018-06-02-08:28:54
INFO:tensorflow:Saving dict for global step 1000: accuracy =
0.8358209, accuracy_baseline = 0.76377374, auc = 0.88401634,
auc_precision_recall = 0.69599575, average_loss = 0.35122654,
global_step = 1000, label/mean = 0.23622628, loss = 44.67437,
```

```
precision = 0.68986726, prediction/mean = 0.23320661, recall =  
0.55408216
```

```
{'accuracy': 0.8358209,  
'accuracy_baseline': 0.76377374,  
'auc': 0.88401634,  
'auc_precision_recall': 0.69599575,  
'average_loss': 0.35122654,  
'global_step': 1000,  
'label/mean': 0.23622628,  
'loss': 44.67437,  
'precision': 0.68986726,  
'prediction/mean': 0.23320661,  
'recall': 0.55408216}
```

The new accuracy level is 83.58 percent. It is four percent higher than the previous model.

Finally, you can add a regularization term to prevent overfitting.

Step 5) Hyperparameter:Lasso & Ridge

Your model can suffer from **overfitting** or **underfitting**.

- Overfitting: The model is unable to generalize the prediction to new data
- Underfitting: The model is unable to capture the pattern of the data. i.e., linear regression when the data is non-linear

When a model has lots of parameters and a relatively low amount of data, it leads to poor predictions. Imagine, one group only have three observations; the model will compute a weight for this group. The weight is used to make a prediction; if the observations of the test set for this particular group is entirely different from the training set, then the model will make a wrong prediction. During the evaluation with the training set, the accuracy is good, but not good with the test set because the weights computed is not the true one to generalize the pattern. In this case, it does not make a reasonable prediction on unseen data.

To prevent overfitting, regularization gives you the possibilities to control for such complexity and make it more generalizable. There are two regularization techniques:

- L1: Lasso
- L2: Ridge

In TensorFlow, you can add these two hyperparameters in the optimizer. For instance, the higher the hyperparameter L2, the weight tends to be very low and close to zero. The fitted line will be very flat, while an L2 close to zero implies the weights are close to the regular linear regression.

You can try by yourself the different value of the hyperparameters and see if you can increase the accuracy level.

Note that if you change the hyperparameter, you need to delete the folder ongoing/train4 otherwise the model will start with the previously trained model.

Let's see how is the accuracy with the hype

```
model_regu = tf.estimator.LinearClassifier(  
    model_dir="ongoing/train4",  
    feature_columns=categorical_features+base_columns+education_x_occupation+age_buckets_x_education_x_occupation,  
    optimizer=tf.train.FtrlOptimizer(  
        learning_rate=0.1,  
        l1_regularization_strength=0.9,  
        l2_regularization_strength=5))
```

OUPUT

```
INFO:tensorflow:Using default config.  
INFO:tensorflow:Using config: {'_model_dir': 'ongoing/train4',  
'_tf_random_seed': None, '_save_summary_steps': 100,  
'_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,  
'_session_config': None, '_keep_checkpoint_max': 5,  
'_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps':  
100, '_train_distribute': None, '_service': None, '_cluster_spec':
```

```
<tensorflow.python.training.server_lib.ClusterSpec object at  
0x1820d9c128>, '_task_type': 'worker', '_task_id': 0,  
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':  
'', '_is_chief': True, '_num_ps_replicas': 0,  
'_num_worker_replicas': 1}
```

```
model_regu.train(input_fn=get_input_fn(df_train_new,  
                                         num_epochs=None,  
                                         n_batch = 128,  
                                         shuffle=False),  
                                         steps=1000)
```

OUPUT

```
INFO:tensorflow:Calling model_fn.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow>Create CheckpointSaverHook.  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Saving checkpoints for 1 into  
ongoing/train4/model.ckpt.  
INFO:tensorflow:loss = 88.722855, step = 1  
INFO:tensorflow:global_step/sec: 77.4165  
INFO:tensorflow:loss = 50.38778, step = 101 (1.294 sec)  
INFO:tensorflow:global_step/sec: 187.889  
INFO:tensorflow:loss = 55.38014, step = 201 (0.535 sec)  
INFO:tensorflow:global_step/sec: 201.895  
INFO:tensorflow:loss = 46.806694, step = 301 (0.491 sec)  
INFO:tensorflow:global_step/sec: 217.992  
INFO:tensorflow:loss = 38.68271, step = 401 (0.460 sec)  
INFO:tensorflow:global_step/sec: 193.676  
INFO:tensorflow:loss = 56.99398, step = 501 (0.516 sec)  
INFO:tensorflow:global_step/sec: 202.195  
INFO:tensorflow:loss = 33.263622, step = 601 (0.497 sec)  
INFO:tensorflow:global_step/sec: 216.756  
INFO:tensorflow:loss = 37.7902, step = 701 (0.459 sec)  
INFO:tensorflow:global_step/sec: 240.215  
INFO:tensorflow:loss = 61.732605, step = 801 (0.416 sec)  
INFO:tensorflow:global_step/sec: 220.336  
INFO:tensorflow:loss = 46.938225, step = 901 (0.456 sec)  
INFO:tensorflow:Saving checkpoints for 1000 into  
ongoing/train4/model.ckpt.  
INFO:tensorflow:Loss for final step: 43.4942.
```

```
<tensorflow.python.estimator.canned.linear.LinearClassifier at  
0x181ff39e48>
```

```
model_regu.evaluate(input_fn=get_input_fn(df_test_new,  
                                         num_epochs=1,  
                                         n_batch = 128,  
                                         shuffle=False),  
                     steps=1000)
```

OUTPUT

```
INFO:tensorflow:Calling model_fn.  
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect  
PR-AUCs; please switch to "careful_interpolation" instead.  
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect  
PR-AUCs; please switch to "careful_interpolation" instead.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow:Starting evaluation at 2018-06-02-08:29:07  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Restoring parameters from  
ongoing/train4/model.ckpt-1000  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Evaluation [100/1000]  
INFO:tensorflow:Finished evaluation at 2018-06-02-08:29:09  
INFO:tensorflow:Saving dict for global step 1000: accuracy =  
0.83833915, accuracy_baseline = 0.76377374, auc = 0.8869794,  
auc_precision_recall = 0.7014905, average_loss = 0.34691378,  
global_step = 1000, label/mean = 0.23622628, loss = 44.12581,  
precision = 0.69720596, prediction/mean = 0.23662092, recall =  
0.5579823
```

```
{'accuracy': 0.83833915,  
'accuracy_baseline': 0.76377374,  
'auc': 0.8869794,  
'auc_precision_recall': 0.7014905,  
'average_loss': 0.34691378,  
'global_step': 1000,
```

```
'label/mean': 0.23622628,  
'loss': 44.12581,  
'precision': 0.69720596,  
'prediction/mean': 0.23662092,  
'recall': 0.5579823}
```

With this hyperparameter, you slightly increase the accuracy metrics. In the next tutorial, you will learn how to improve a linear classifier using a kernel method.

Summary

To train a model, you need to:

- Define the features: Independent variables: X
- Define the label: Dependent variable: y
- Construct a train/test set
- Define the initial weight
- Define the loss function: MSE
- Optimize the model: Gradient descent
- Define:
 - Learning rate
 - Number of epoch
 - Batch size
 - Number of class

In this tutorial, you learned how to use the high-level API for a linear regression classifier. You need to define:

1. Feature columns. If continuous: `tf.feature_column.numeric_column()`. You can populate a list with python list comprehension
2. The estimator: `tf.estimator.LinearClassifier(feature_columns, model_dir, n_classes = 2)`
3. A function to import the data, the batch size and epoch: `input_fn()`

After that, you are ready to train, evaluate and make a prediction with `train()`, `evaluate()` and `predict()`

To improve the performance of the model, you can:

- Use polynomial regression
- Interaction term: `tf.feature_column.crossed_column`
- Add regularization parameter

Chapter 16: Kernel Methods

The purpose of this tutorial is to make a dataset linearly separable. The tutorial is divided into two parts:

1. Feature transformation
2. Train a Kernel classifier with Tensorflow

In the first part, you will understand the idea behind a kernel classifier while in the second part, you will see how to train a kernel classifier with Tensorflow. You will use the adult dataset. The objective of this dataset is to classify the revenue below and above 50k, knowing the behavior of each household.

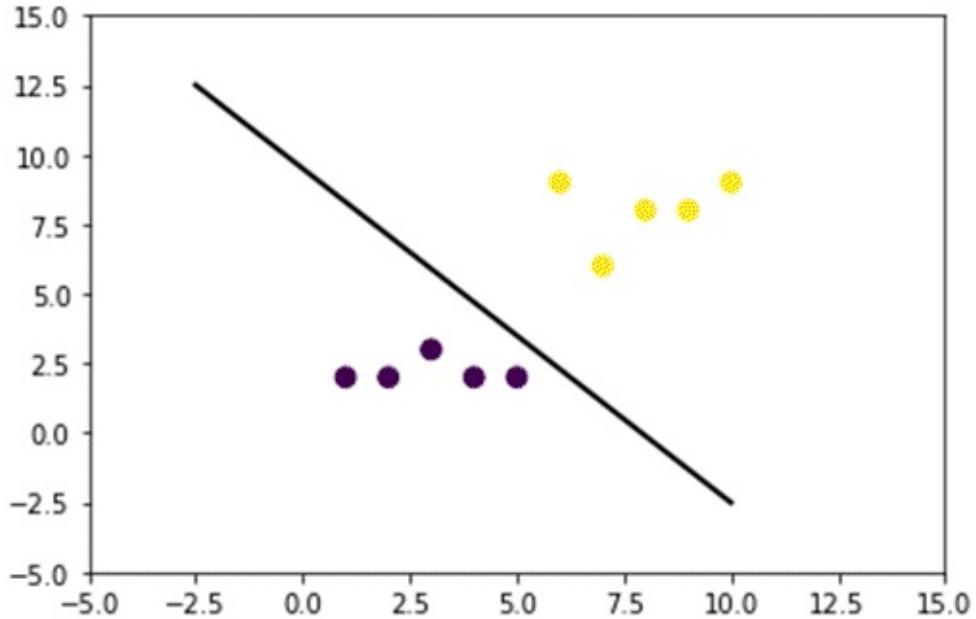
Why do you need Kernel Methods?

The aim of every classifier is to predict the classes correctly. For that, the dataset should be separable. Look at the plot below; it is fairly simple to see that all points above the black line belong to the first class and the other points to the second class. However, it is extremely rare to have a dataset that simple. In most case, the data are not separable. It gives naive classifiers like a logistic regression a hard time.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

x_lin = np.array([1,2,3,4,5,6,7,8,9,10])
y_lin = np.array([2,2,3,2,2,9,6,8,8,9])
label_lin = np.array([0,0,0,0,0,1,1,1,1,1])

fig = plt.figure()
ax=fig.add_subplot(111)
plt.scatter(x_lin, y_lin, c=label_lin, s=60)
plt.plot([-2.5, 10], [12.5, -2.5], 'k-', lw=2)
ax.set_xlim([-5,15])
ax.set_ylim([-5,15])plt.show()
```

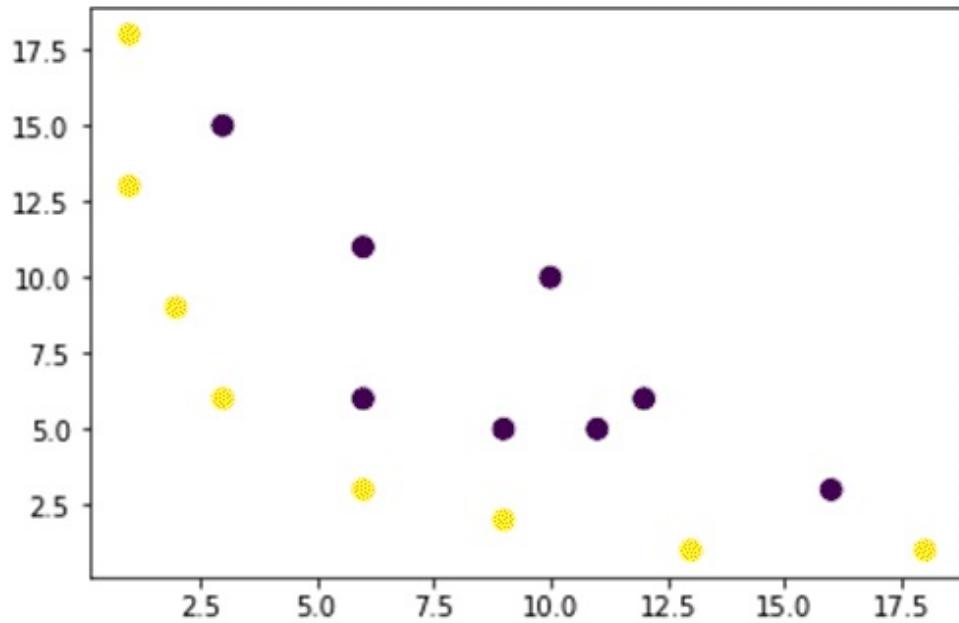


In the figure below, we plot a dataset which is not linearly separable. If we draw a straight line, most of the points will be not be classified in the correct class.

One way to tackle this problem is to take the dataset and transform the data in another feature map. It means, you will use a function to transform the data in another plan, which should be linearizable.

```
x = np.array([1,1,2,3,3,6,6,6,9,9,10,11,12,13,16,18])
y = np.array([18,13,9,6,15,11,6,3,5,2,10,5,6,1,3,1])
label = np.array([1,1,1,1,0,0,0,1,0,1,0,0,0,1,0,1])
```

```
fig = plt.figure()
plt.scatter(x, y, c=label, s=60)
plt.show()
```



The data from the figure above is in a two-dimension plan which is not separable. You can try to transform these data in a three-dimension, it means, you create a figure with 3 axes.

In our example, we will apply a polynomial mapping to bring our data to a 3D dimension. The formula to transform the data is as follow.

$$\phi(x, y) = (x^2, \sqrt{2}xy, y^2)$$

You define a function in Python to create the new feature maps

You can use numpy to code the above formula:

Formula	Equivalent Numpy Code
x	$x[:,0]^{**}$
y	$x[:,1]$
x^2	$x[:,0]^{**2}$
$\sqrt{2}$	$np.sqrt(2)*$
xy	$x[:,0]*x[:,1]$
y^2	$x[:,1]^{**2}$

```
### illustration purpose
def mapping(x, y):
    x = np.c_[(x, y)]
    if len(x) > 2:
        x_1 = x[:, 0]**2
        x_2 = np.sqrt(2)*x[:, 0]*x[:, 1]
        x_3 = x[:, 1]**2
    else:
        x_1 = x[0]**2
        x_2 = np.sqrt(2)*x[0]*x[1]
        x_3 = x[1]**2
    trans_x = np.array([x_1, x_2, x_3])
    return trans_x
```

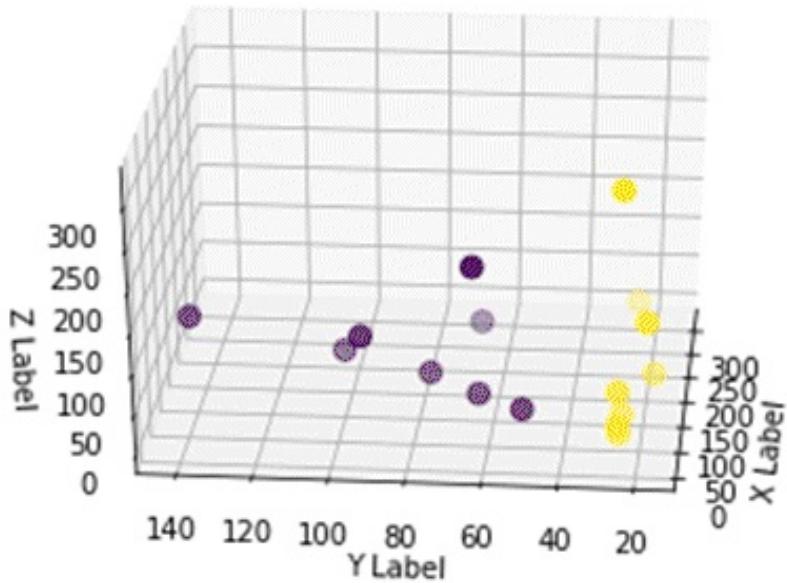
The new mapping should be with 3 dimensions with 16 points

```
x_1 = mapping(x, y)
x_1.shape
```

```
(3, 16)
```

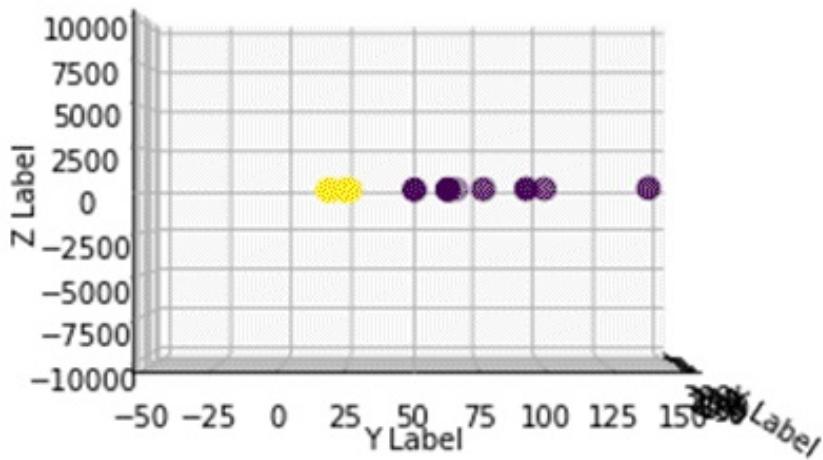
Let's make a new plot with 3 axis, x, y and z respectively.

```
# plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x_1[0], x_1[1], x_1[2], c=label, s=60)
ax.view_init(30, 185)
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')
plt.show()
```



We see an improvement but if we change the orientation of the plot, it is clear that the dataset is now separable

```
# plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x_1[0], x_1[1], x_1[1], c=label, s=60)
ax.view_init(0, -180)ax.set_xlim([150, -50])
ax.set_zlim([-10000, 10000])
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')plt.show()
```



To manipulate a large dataset and you may have to create more than 2 dimensions, you will face a big problem using the above method. In fact, you need to transform all data points, which is clearly not sustainable. It will take you ages, and your computer may run out of memory.

The most common way to overcome this issue is to use a **kernel**.

What is a Kernel in machine learning?

The idea is to use a higher-dimension feature space to make the data almost linearly separable as shown in the figure above.

There are plenty of higher dimensional spaces to make the data points separable. For instance, we have shown that the polynomial mapping is a great start.

We have also demonstrated that with lots of data, these transformation is not efficient. Instead, you can use a kernel function to modify the data without changing to a new feature plan.

The magic of the kernel is to find a function that avoids all the trouble implied by the high-dimensional computation. The result of a kernel is a scalar, or said differently we are back to one-dimensional space

After you found this function, you can plug it to the standard linear classifier.

Let's see an example to understand the concept of Kernel. You have two vectors, x_1 and x_2 . The objective is to create a higher dimension by using a polynomial mapping. The output is equal to the dot product of the new feature map. From the method above, you need to:

1. Transform x_1 and x_2 into a new dimension
2. Compute the dot product: common to all kernels
3. Transform x_1 and x_2 into a new dimension

You can use the function created above to compute the higher dimension.

```
## Kernel
x1 = np.array([3, 6])
x2 = np.array([10, 10])

x_1 = mapping(x1, x2)
```

```
print(x_1)
```

Output

```
[[ 9.          100.         ]
 [ 25.45584412 141.42135624]
 [ 36.          100.        ]]
```

Compute the dot product

You can use the object `dot` from numpy to compute the dot product between the first and second vector stored in `x_1`.

```
print(np.dot(x_1[:,0], x_1[:,1]))
8100.0
```

The output is 8100. You see the problem, you need to store in memory a new feature map to compute the dot product. If you have a dataset with millions of records, it is computationally ineffective.

Instead, you can use the **polynomial kernel** to compute the dot product without transforming the vector. This function computes the dot product of `x1` and `x2` as if these two vectors have been transformed into the higher dimension. Said differently, a kernel function computes the results of the dot product from another feature space.

You can write the polynomial kernel function in Python as follow.

```
def polynomial_kernel(x, y, p=2):
    return (np.dot(x, y)) ** p
```

It is the power of the dot product of two vectors. Below, you return the second degree of the polynomial kernel. The output is equal to the other method. This is the magic of the kernel.

```
polynomial_kernel(x1, x2, p=2)
8100
```

Type of Kernel Methods

There are lots of different kernels available. The simplest is the linear kernel. This function works pretty well for text classification. The other kernel is:

- Polynomial kernel
- Gaussian Kernel

In the example with TensorFlow, we will use the Random Fourier. TensorFlow has a build in estimator to compute the new feature space. This function is an approximation of the Gaussian kernel function.

$$e^{-\frac{\|x - y\|^2}{2\sigma^2}}$$

This function computes the similarity between the data points in a much higher dimensional space.

Train Gaussian Kernel classifier with TensorFlow

The objective of the algorithm is to classify the household earning more or less than 50k.

You will evaluate a logistic regression to have a benchmark model. After that, you will train a Kernel classifier to see if you can get better results.

You use the following variables from the adult dataset:

- age
- workclass
- fnlwgt
- education
- education_num
- marital
- occupation
- relationship
- race
- sex
- capital_gain
- capital_loss
- hours_week
- native_country
- label

You will proceed as follow before you train and evaluate the model:

- Step 1) Import the libraries
- Step 2) Import the data

- Step 3) Prepare the data
- Step 4) Construct the input_fn
- Step 5) Construct the logistic model: Baseline model
- Step 6) Evaluate the model
- Step 7) Construct the Kernel classifier
- Step 8) Evaluate the Kernel classifier

Step 1) Import the libraries

To import and train the model, you need to import tensorflow, pandas and numpy

```
#import numpy as np
from sklearn.model_selection
import train_test_split
import tensorflow as tf
import pandas as pd
import numpy as np
```

Step 2) Import the data

You download the data from the following website and you import it as a panda dataframe.

```
## Define path data
COLUMNS = ['age', 'workclass', 'fnlwgt', 'education',
'education_num', 'marital', 'occupation', 'relationship', 'race',
'sex', 'capital_gain', 'capital_loss', 'hours_week',
'native_country', 'label']
PATH = "https://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.data"
PATH_test ="https://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.test"
"## Import
df_train = pd.read_csv(PATH, skipinitialspace=True, names =
COLUMNS, index_col=False)
df_test = pd.read_csv(PATH_test,skiprows = 1,
skipinitialspace=True, names = COLUMNS, index_col=False)
```

Now that the train and test set are defined, you can change the column label from

string to integer. tensorflow does not accept string value for the label.

```
label = {'<=50K': 0, '>50K': 1}
df_train.label = [label[item] for item in df_train.label]
label_t = {'<=50K.': 0, '>50K.': 1}
df_test.label = [label_t[item] for item in df_test.label]
df_train.shape

(32561, 15)
```

Step 3) Prepare the data

The dataset contains both continuous and categorical features. A good practice is to standardize the values of the continuous variables. You can use the function StandardScaler from sci-kit learn. You create a user-defined function as well to make it easier to convert the train and test set. Note that, you concatenate the continuous and categorical variables to a common dataset and the array should be of the type: float32

```
COLUMNS_INT = ['age', 'fnlwgt', 'education_num', 'capital_gain',
'capital_loss', 'hours_week']
CATE_FEATURES = ['workclass', 'education', 'marital', 'occupation',
'relationship', 'race', 'sex', 'native_country']
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing

def prep_data_str(df):
    scaler = StandardScaler()
    le = preprocessing.LabelEncoder()
    df_toscale = df[COLUMNS_INT]
    df_scaled = scaler.fit_transform(df_toscale.astype(np.float64))
    X_1 = df[CATE_FEATURES].apply(le.fit_transform)
    y = df['label'].astype(np.int32)
    X_conc = np.c_[df_scaled, X_1].astype(np.float32)
    return X_conc, y
```

The transformer function is ready, you can convert the dataset and create the input_fn function.

```
X_train, y_train = prep_data_str(df_train)
```

```
X_test, y_test = prep_data_str(df_test)
print(X_train.shape)
(32561, 14)
```

In the next step, you will train a logistic regression. It will give you a baseline accuracy. The objective is to beat the baseline with a different algorithm, namely a Kernel classifier.

Step 4) Construct the logistic model: Baseline model

You construct the feature column with the object `real_valued_column`. It will make sure all variables are dense numeric data.

```
feat_column = tf.contrib.layers.real_valued_column('features',
dimension=14)
```

The estimator is defined using TensorFlow Estimator, you instruct the feature columns and where to save the graph.

```
estimator = tf.estimator.LinearClassifier(feature_columns=
[feat_column],
                                         n_classes=2,
                                         model_dir = "kernel_log"
                                         )
```

```
INFO:tensorflow:Using default config.INFO:tensorflow:Using config:
{'_model_dir': 'kernel_log', '_tf_random_seed': None,
'_save_summary_steps': 100, '_save_checkpoints_steps': None,
'_save_checkpoints_secs': 600, '_session_config': None,
'_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000,
'_log_step_count_steps': 100, '_train_distribute': None,
'_service': None, '_cluster_spec':
<tensorflow.python.training.server_lib.ClusterSpec object at
0x1a2003f780>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':
'', '_is_chief': True, '_num_ps_replicas': 0,
'_num_worker_replicas': 1}
```

You will train the logistic regression using mini-batches of size 200.

```
# Train the model
```

```
train_input_fn = tf.estimator.inputs.numpy_input_fn(  
    x={"features": X_train},  
    y=y_train,  
    batch_size=200,  
    num_epochs=None,  
    shuffle=True)
```

You can train the model with 1.000 iteration

```
estimator.train(input_fn=train_input_fn, steps=1000)
```

```
INFO:tensorflow:Calling model_fn.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow>Create CheckpointSaverHook.  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Saving checkpoints for 1 into  
kernel_log/model.ckpt.  
INFO:tensorflow:loss = 138.62949, step = 1  
INFO:tensorflow:global_step/sec: 324.16  
INFO:tensorflow:loss = 87.16762, step = 101 (0.310 sec)  
INFO:tensorflow:global_step/sec: 267.092  
INFO:tensorflow:loss = 71.53657, step = 201 (0.376 sec)  
INFO:tensorflow:global_step/sec: 292.679  
INFO:tensorflow:loss = 69.56703, step = 301 (0.340 sec)  
INFO:tensorflow:global_step/sec: 225.582  
INFO:tensorflow:loss = 74.615875, step = 401 (0.445 sec)  
INFO:tensorflow:global_step/sec: 209.975  
INFO:tensorflow:loss = 76.49044, step = 501 (0.475 sec)  
INFO:tensorflow:global_step/sec: 241.648  
INFO:tensorflow:loss = 66.38373, step = 601 (0.419 sec)  
INFO:tensorflow:global_step/sec: 305.193  
INFO:tensorflow:loss = 87.93341, step = 701 (0.327 sec)  
INFO:tensorflow:global_step/sec: 396.295  
INFO:tensorflow:loss = 76.61518, step = 801 (0.249 sec)  
INFO:tensorflow:global_step/sec: 359.857  
INFO:tensorflow:loss = 78.54885, step = 901 (0.277 sec)  
INFO:tensorflow:Saving checkpoints for 1000 into  
kernel_log/model.ckpt.  
INFO:tensorflow:Loss for final step: 67.79706.
```

```
<tensorflow.python.estimator.canned.linear.LinearClassifier at
```

```
0x1a1fa3cbe0>
```

Step 6) Evaluate the model

You define the numpy estimator to evaluate the model. You use the entire dataset for evaluation

```
# Evaluation
test_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"features": X_test},
    y=y_test,
    batch_size=16281,
    num_epochs=1,
    shuffle=False)
estimator.evaluate(input_fn=test_input_fn, steps=1)
```

```
INFO:tensorflow:Calling model_fn.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect
PR-AUCs; please switch to "careful_interpolation" instead.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect
PR-AUCs; please switch to "careful_interpolation" instead.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2018-07-12-15:58:22
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from kernel_log/model.ckpt-
1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Evaluation [1/1]
INFO:tensorflow:Finished evaluation at 2018-07-12-15:58:23
INFO:tensorflow:Saving dict for global step 1000: accuracy =
0.82353663, accuracy_baseline = 0.76377374, auc = 0.84898686,
auc_precision_recall = 0.67214864, average_loss = 0.3877216,
global_step = 1000, label/mean = 0.23622628, loss = 6312.495,
precision = 0.7362797, prediction/mean = 0.21208474, recall =
0.39417577
```

```
{'accuracy': 0.82353663,
'accuracy_baseline': 0.76377374,
'auc': 0.84898686,
'auc_precision_recall': 0.67214864,
'average_loss': 0.3877216,
'global_step': 1000,
```

```
'label/mean': 0.23622628,  
'loss': 6312.495,  
'precision': 0.7362797,  
'prediction/mean': 0.21208474,  
'recall': 0.39417577}
```

You have an accuracy of 82 percents. In the next section, you will try to beat the logistic classifier with a Kernel classifier

Step 7) Construct the Kernel classifier

The kernel estimator is not so different from the traditional linear classifier, at least in term of construction. The idea behind is to use the power of explicit kernel with the linear classifier.

You need two pre-defined estimators available in TensorFlow to train the Kernel Classifier:

- RandomFourierFeatureMapper
- KernelLinearClassifier

You learned in the first section that you need to transform the low dimension into a high dimension using a kernel function. More precisely, you will use the Random Fourier, which is an approximation of the Gaussian function. Luckily, Tensorflow has the function in its library: RandomFourierFeatureMapper. The model can be trained using the estimator KernelLinearClassifier.

To build the model, you will follow these steps:

1. Set the high dimension Kernel function
2. Set the L2 hyperparameter
3. Build the model
4. Train the model
5. Evaluate the model

Step A) Set the high dimension Kernel function

The current dataset contains 14 features that you will transform to a new high dimension of the 5.000-dimensional vector. You use the random Fourier features to achieve the transformation. If you recall the Gaussian Kernel formula, you note that there is the standard deviation parameter to define. This parameter controls for the similarity measure employs during the classification.

You can tune all the parameters in RandomFourierFeatureMapper with:

- input_dim = 14
- output_dim= 5000
- stddev=4

```
### Prep Kernel
kernel_mapper =
tf.contrib.kernel_methods.RandomFourierFeatureMapper(input_dim=14,
output_dim=5000, stddev=4, name='rffm')
```

You need to construct the kernel mapper by using the feature columns created before: feat_column

```
### Map Kernel
kernel_mappers = {feat_column: [kernel_mapper]}
```

Step B) Set the L2 hyperparameter

To prevent overfitting, you penalize the loss function with the L2 regularizer. You set the L2 hyperparameter to 0.1 and the learning rate to 5

```
optimizer = tf.train.FtrlOptimizer(learning_rate=5,
l2_regularization_strength=0.1)
```

Step C) Build the model

The next step is similar to the linear classification. You use the build-in estimator KernelLinearClassifier. Note that you add the kernel mapper defined previously and change the model directory.

```
### Prep estimator
estimator_kernel =
tf.contrib.kernel_methods.KernelLinearClassifier(
    n_classes=2,
    optimizer=optimizer,
    kernel_mappers=kernel_mappers,
    model_dir="kernel_train")
```

```
WARNING:tensorflow:From /Users/Thomas/anaconda3/envs/hello-
tf/lib/python3.6/site-
packages/tensorflow/contrib/kernel_methods/python/kernel_estimators
.py:305: multi_class_head (from
tensorflow.contrib.learn.python.learn.estimators.head) is
deprecated and will be removed in a future version.
Instructions for updating:
Please switch to tf.contrib.estimator.*_head.
WARNING:tensorflow:From /Users/Thomas/anaconda3/envs/hello-
tf/lib/python3.6/site-
packages/tensorflow/contrib/learn/python/learn/estimators/estimator
.py:1179: BaseEstimator.__init__ (from
tensorflow.contrib.learn.python.learn.estimators.estimator) is
deprecated and will be removed in a future version.
Instructions for updating:
Please replace uses of any Estimator from tf.contrib.learn with an
Estimator from tf.estimator.*
WARNING:tensorflow:From /Users/Thomas/anaconda3/envs/hello-
tf/lib/python3.6/site-
packages/tensorflow/contrib/learn/python/learn/estimators/estimator
.py:427: RunConfig.__init__ (from
tensorflow.contrib.learn.python.learn.estimators.run_config) is
deprecated and will be removed in a future version.
Instructions for updating:
When switching to tf.estimator.Estimator, use
tf.estimator.RunConfig instead.
INFO:tensorflow:Using default config.
INFO:tensorflow:Using config: {'_task_type': None, '_task_id': 0,
'_cluster_spec': <tensorflow.python.training.server_lib.ClusterSpec
object at 0x1a200ae550>, '_master': '', '_num_ps_replicas': 0,
'_num_worker_replicas': 0, '_environment': 'local', '_is_chief':
True, '_evaluation_master': '', '_train_distribute': None,
'_tf_config': gpu_options {
    per_process_gpu_memory_fraction: 1.0
},
'_tf_random_seed': None, '_save_summary_steps': 100,
'_save_checkpoints_secs': 600, '_log_step_count_steps': 100,
```

```
'_session_config': None, '_save_checkpoints_steps': None,
'_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000,
'_model_dir': 'kernel_train'}
```

Step D) Train the model

Now that the Kernel classifier is built, you are ready to train it. You choose to iterate 2000 times the model

```
### estimate
estimator_kernel.fit(input_fn=train_input_fn, steps=2000)

WARNING:tensorflow:Casting <dtype: 'int32'> labels to bool.
WARNING:tensorflow:Casting <dtype: 'int32'> labels to bool.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect
PR-AUCs; please switch to "careful_interpolation" instead.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect
PR-AUCs; please switch to "careful_interpolation" instead.
WARNING:tensorflow:From /Users/Thomas/anaconda3/envs/hello-
tf/lib/python3.6/site-
packages/tensorflow/contrib/learn/python/learn/estimators/head.py:6
78: ModelFnOps.__new__ (from
tensorflow.contrib.learn.python.learn.estimators.model_fn) is
deprecated and will be removed in a future version.
Instructions for updating:
When switching to tf.estimator.Estimator, use
tf.estimator.EstimatorSpec. You can use the `estimator_spec` method
to create an equivalent one.
INFO:tensorflow>Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into
kernel_train/model.ckpt.
INFO:tensorflow:loss = 0.6931474, step = 1
INFO:tensorflow:global_step/sec: 86.6365
INFO:tensorflow:loss = 0.39374447, step = 101 (1.155 sec)
INFO:tensorflow:global_step/sec: 80.1986
INFO:tensorflow:loss = 0.3797774, step = 201 (1.247 sec)
INFO:tensorflow:global_step/sec: 79.6376
INFO:tensorflow:loss = 0.3908726, step = 301 (1.256 sec)
INFO:tensorflow:global_step/sec: 95.8442
```

```
INFO:tensorflow:loss = 0.41890752, step = 401 (1.043 sec)
INFO:tensorflow:global_step/sec: 93.7799
INFO:tensorflow:loss = 0.35700393, step = 501 (1.066 sec)
INFO:tensorflow:global_step/sec: 94.7071
INFO:tensorflow:loss = 0.35535482, step = 601 (1.056 sec)
INFO:tensorflow:global_step/sec: 90.7402
INFO:tensorflow:loss = 0.3692882, step = 701 (1.102 sec)
INFO:tensorflow:global_step/sec: 94.4924
INFO:tensorflow:loss = 0.34746957, step = 801 (1.058 sec)
INFO:tensorflow:global_step/sec: 95.3472
INFO:tensorflow:loss = 0.33655524, step = 901 (1.049 sec)
INFO:tensorflow:global_step/sec: 97.2928
INFO:tensorflow:loss = 0.35966292, step = 1001 (1.028 sec)
INFO:tensorflow:global_step/sec: 85.6761
INFO:tensorflow:loss = 0.31254214, step = 1101 (1.167 sec)
INFO:tensorflow:global_step/sec: 91.4194
INFO:tensorflow:loss = 0.33247527, step = 1201 (1.094 sec)
INFO:tensorflow:global_step/sec: 82.5954
INFO:tensorflow:loss = 0.29305756, step = 1301 (1.211 sec)
INFO:tensorflow:global_step/sec: 89.8748
INFO:tensorflow:loss = 0.37943482, step = 1401 (1.113 sec)
INFO:tensorflow:global_step/sec: 76.9761
INFO:tensorflow:loss = 0.34204718, step = 1501 (1.300 sec)
INFO:tensorflow:global_step/sec: 73.7192
INFO:tensorflow:loss = 0.34614792, step = 1601 (1.356 sec)
INFO:tensorflow:global_step/sec: 83.0573
INFO:tensorflow:loss = 0.38911164, step = 1701 (1.204 sec)
INFO:tensorflow:global_step/sec: 71.7029
INFO:tensorflow:loss = 0.35255936, step = 1801 (1.394 sec)
INFO:tensorflow:global_step/sec: 73.2663
INFO:tensorflow:loss = 0.31130585, step = 1901 (1.365 sec)
INFO:tensorflow:Saving checkpoints for 2000 into
kernel_train/model.ckpt.
INFO:tensorflow:Loss for final step: 0.37795097.
```

```
KernelLinearClassifier(params={'head':
<tensorflow.contrib.learn.python.learn.estimators.head._BinaryLogisticHead object at 0x1a2054cd30>, 'feature_columns':
{_RealValuedColumn(column_name='features_MAPPED', dimension=5000,
default_value=None, dtype=tf.float32, normalizer=None)},
'optimizer': <tensorflow.python.training.ftrl.FtrlOptimizer object
at 0x1a200aec18>, 'kernel_mappers':
{_RealValuedColumn(column_name='features', dimension=14,
default_value=None, dtype=tf.float32, normalizer=None)}:
[<tensorflow.contrib.kernel_methods.python.mappers.random_fourier_f
```

```
eatures.RandomFourierFeatureMapper object at 0x1a200ae400>]})
```

Step E) Evaluate the model

Last but not least, you evaluate the performance of your model. You should be able to beat the logistic regression.

```
# Evaluate and report metrics.  
eval_metrics = estimator_kernel.evaluate(input_fn=test_input_fn,  
steps=1)
```

```
WARNING:tensorflow:Casting <dtype: 'int32'> labels to bool.  
WARNING:tensorflow:Casting <dtype: 'int32'> labels to bool.  
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect  
PR-AUCs; please switch to "careful_interpolation" instead.  
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect  
PR-AUCs; please switch to "careful_interpolation" instead.  
INFO:tensorflow:Starting evaluation at 2018-07-12-15:58:50  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Restoring parameters from kernel_train/model.ckpt-  
2000  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Evaluation [1/1]  
INFO:tensorflow:Finished evaluation at 2018-07-12-15:58:51  
INFO:tensorflow:Saving dict for global step 2000: accuracy =  
0.83975184, accuracy/baseline_label_mean = 0.23622628,  
accuracy/threshold_0.500000_mean = 0.83975184, auc = 0.8904007,  
auc_precision_recall = 0.72722375, global_step = 2000,  
labels/actual_label_mean = 0.23622628, labels/prediction_mean =  
0.23786618, loss = 0.34277728,  
precision/positive_threshold_0.500000_mean = 0.73001117,  
recall/positive_threshold_0.500000_mean = 0.5104004
```

The final accuracy is 84%, it is a 2% improvement compared to the logistic regression. There is a tradeoff between accuracy improvement and computational cost. You need to think if 2% improvement worth the time consumed by the different classifier and if it has a compelling impact on your business.

Summary

A kernel is a great tool to transform non-linear data to (almost) linear. The shortcoming of this method is it computationally time-consuming and costly.

Below, you can find the most important code to train a kernel classifier

Set the high dimension Kernel function

- input_dim = 14
- output_dim= 5000
- stddev=4

```
### Prep Kernel
kernel_mapper =
tf.contrib.kernel_methods.RandomFourierFeatureMapper(input_dim=14,
output_dim=5000, stddev=4, name='rffm')
```

Set the L2 hyperparameter

```
optimizer = tf.train.FtrlOptimizer(learning_rate=5,
l2_regularization_strength=0.1)
```

Build the model

```
estimator_kernel =
tf.contrib.kernel_methods.KernelLinearClassifier(      n_classes=2,
optimizer=optimizer,
kernel_mappers=kernel_mappers,
model_dir="kernel_train")
```

Train the model

```
estimator_kernel.fit(input_fn=train_input_fn, steps=2000)
```

Evaluate the model

```
eval_metrics = estimator_kernel.evaluate(input_fn=test_input_fn,
steps=1)
```

Chapter 17: TensorFlow ANN (Artificial Neural Network)

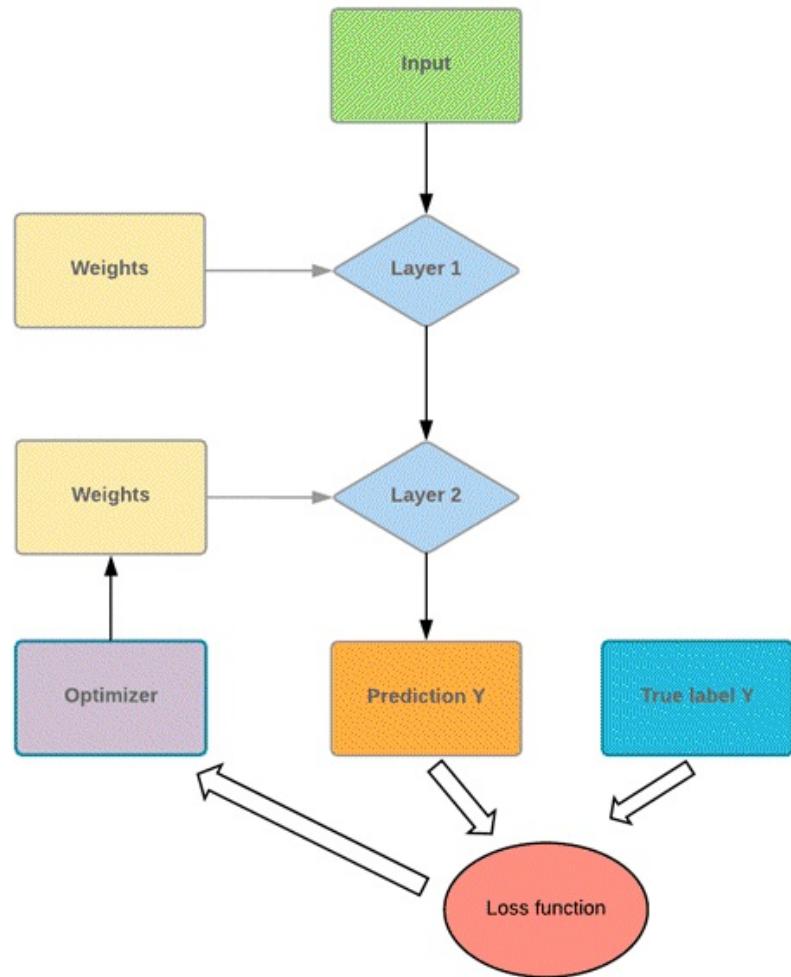
What is Artificial Neural Network?

An Artificial Neural Network(ANN) is composed of four principal objects:

- Layers: all the learning occurs in the layers. There are 3 layers 1) Input 2) Hidden and 3) Output
- feature and label: Input data to the network(features) and output from the network (labels)
- loss function: Metric used to estimate the performance of the learning phase
- optimizer: Improve the learning by updating the knowledge in the network

A neural network will take the input data and push them into an ensemble of layers. The network needs to evaluate its performance with a loss function. The loss function gives to the network an idea of the path it needs to take before it masters the knowledge. The network needs to improve its knowledge with the help of an optimizer.

If you take a look at the figure below, you will understand the underlying mechanism.



The program takes some input values and pushes them into two fully connected layers. Imagine you have a math problem, the first thing you do is to read the corresponding chapter to solve the problem. You apply your new knowledge to solve the problem. There is a high chance you will not score very well. It is the same for a network. The first time it sees the data and makes a prediction, it will not match perfectly with the actual data.

To improve its knowledge, the network uses an optimizer. In our analogy, an optimizer can be thought of as rereading the chapter. You gain new insights/lesson by reading again. Similarly, the network uses the optimizer, updates its knowledge, and tests its new knowledge to check how much it still needs to learn. The program will repeat this step until it makes the lowest error

possible.

In our math problem analogy, it means you read the textbook chapter many times until you thoroughly understand the course content. Even after reading multiple times, if you keep making an error, it means you reached the knowledge capacity with the current material. You need to use different textbook or test different method to improve your score. For a neural network, it is the same process. If the error is far from 100%, but the curve is flat, it means with the current architecture; it cannot learn anything else. The network has to be better optimized to improve the knowledge.

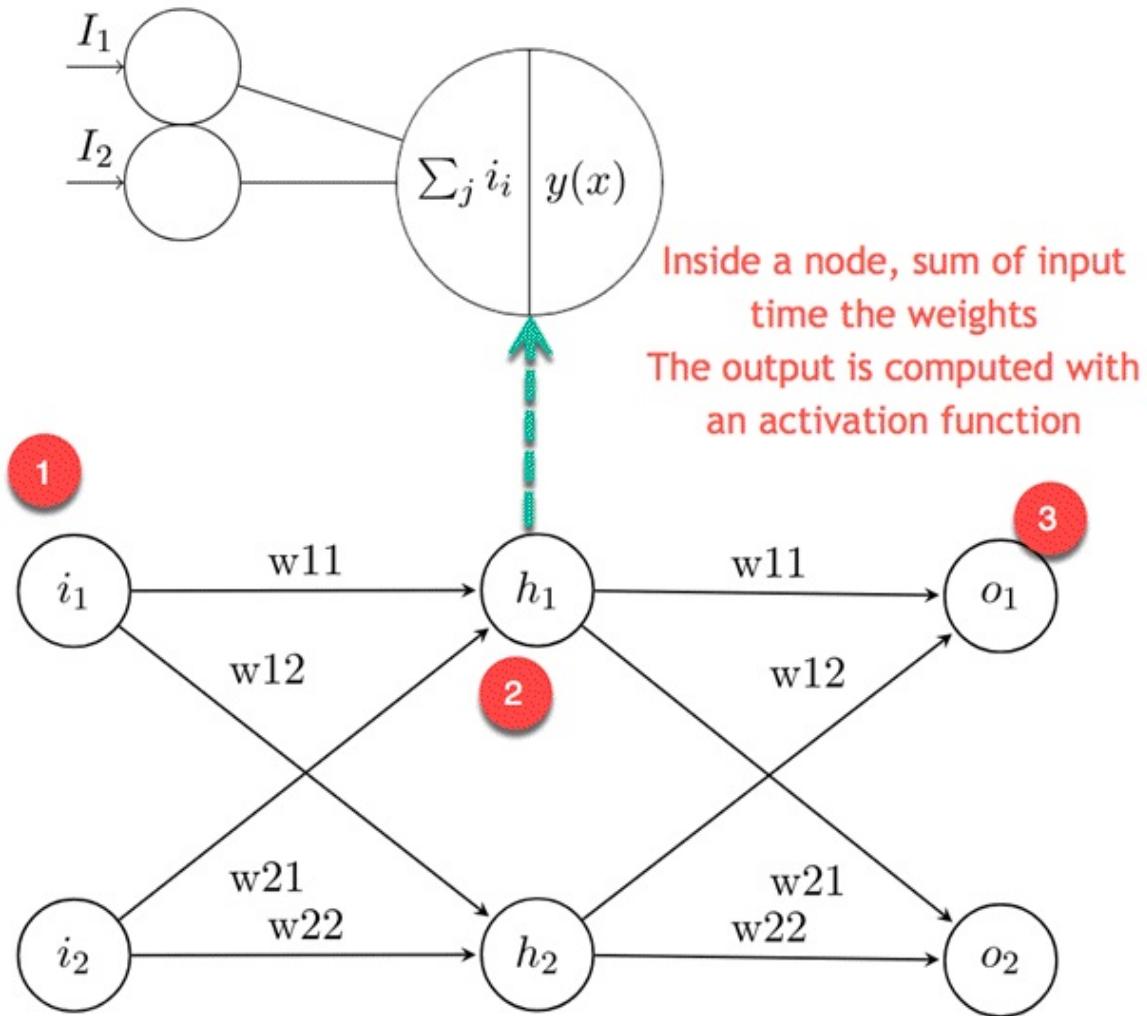
Neural Network Architecture

Layers

A layer is where all the learning takes place. Inside a layer, there are an infinite amount of weights (neurons). A typical neural network is often processed by densely connected layers (also called fully connected layers). It means all the inputs are connected to the output.

A typical neural network takes a vector of input and a scalar that contains the labels. The most comfortable set up is a binary classification with only two classes: 0 and 1.

The network takes an input, sends it to all connected nodes and computes the signal with an **activation** function.

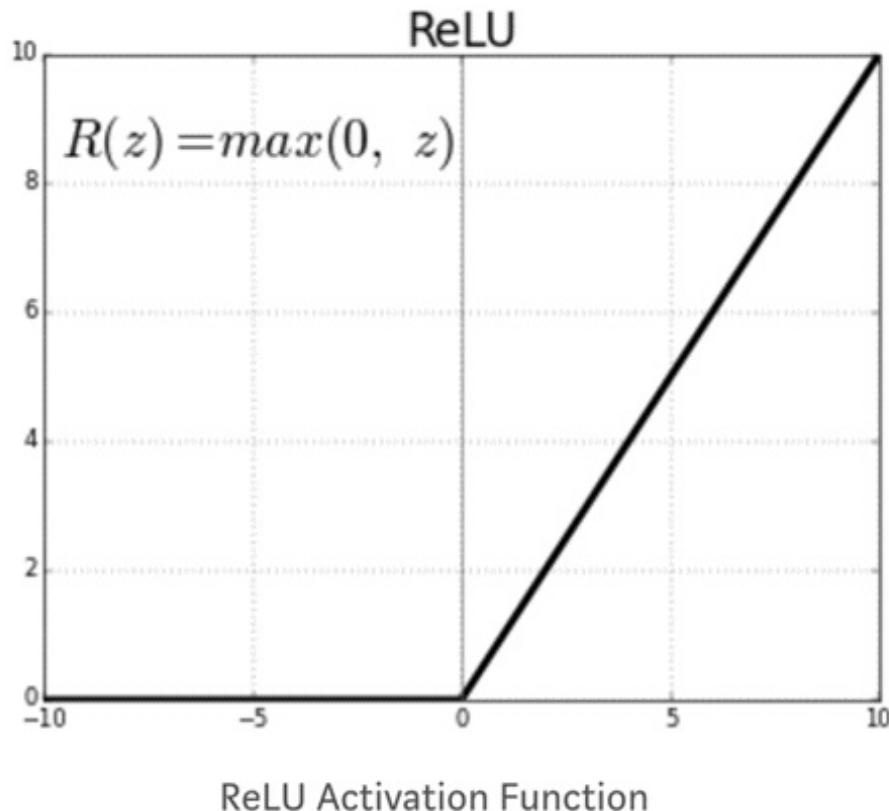


The figure above plots this idea. The first layer is the input values for the second layer, called the hidden layer, receives the weighted input from the previous layer

1. The first node is the input values
2. The neuron is decomposed into the input part and the activation function.
The left part receives all the input from the previous layer. The right part is the sum of the input passes into an activation function.
3. Output value computed from the hidden layers and used to make a prediction. For classification, it is equal to the number of class. For regression, only one value is predicted.

Activation function

The activation function of a node defines the output given a set of inputs. You need an activation function to allow the network to learn non-linear pattern. A common activation function is a **Relu, Rectified linear unit**. The function gives a zero for all negative values.



The other activation functions are:

- Piecewise Linear
- Sigmoid
- Tanh
- Leaky Relu

The critical decision to make when building a neural network is:

- How many layers in the neural network
- How many hidden units for each layer

Neural network with lots of layers and hidden units can learn a complex representation of the data, but it makes the network's computation very expensive.

Loss function

After you have defined the hidden layers and the activation function, you need to specify the loss function and the optimizer.

For binary classification, it is common practice to use a binary cross entropy loss function. In the linear regression, you use the mean square error.

The loss function is an important metric to estimate the performance of the optimizer. During the training, this metric will be minimized. You need to select this quantity carefully depending on the type of problem you are dealing with.

Optimizer

The loss function is a measure of the model's performance. The optimizer will help improve the weights of the network in order to decrease the loss. There are different optimizers available, but the most common one is the Stochastic Gradient Descent.

The conventional optimizers are:

- Momentum optimization,
- Nesterov Accelerated Gradient,
- AdaGrad,
- Adam optimization

Limitations of Neural Network

Overfitting

A common problem with the complex neural net is the difficulties in generalizing unseen data. A neural network with lots of weights can identify specific details in the train set very well but often leads to overfitting. If the data are unbalanced within groups (i.e., not enough data available in some groups), the network will learn very well during the training but will not have the ability to generalize such pattern to never-seen-before data.

There is a trade-off in machine learning between optimization and generalization.

Optimize a model requires to find the best parameters that minimize the loss of the training set.

Generalization, however, tells how the model behaves for unseen data.

To prevent the model from capturing specific details or unwanted patterns of the training data, you can use different techniques. The best method is to have a balanced dataset with sufficient amount of data. The art of reducing overfitting is called **regularization**. Let's review some conventional techniques.

Network size

A neural network with too many layers and hidden units are known to be highly sophisticated. A straightforward way to reduce the complexity of the model is to reduce its size. There is no best practice to define the number of layers. You need to start with a small amount of layer and increases its size until you find the model overfit.

Weight Regularization

A standard technique to prevent overfitting is to add constraints to the weights of the network. The constraint forces the size of the network to take only small values. The constraint is added to the loss function of the error. There are two kinds of regularization:

L1: Lasso: Cost is proportional to the absolute value of the weight coefficients

L2: Ridge: Cost is proportional to the square of the value of the weight coefficients

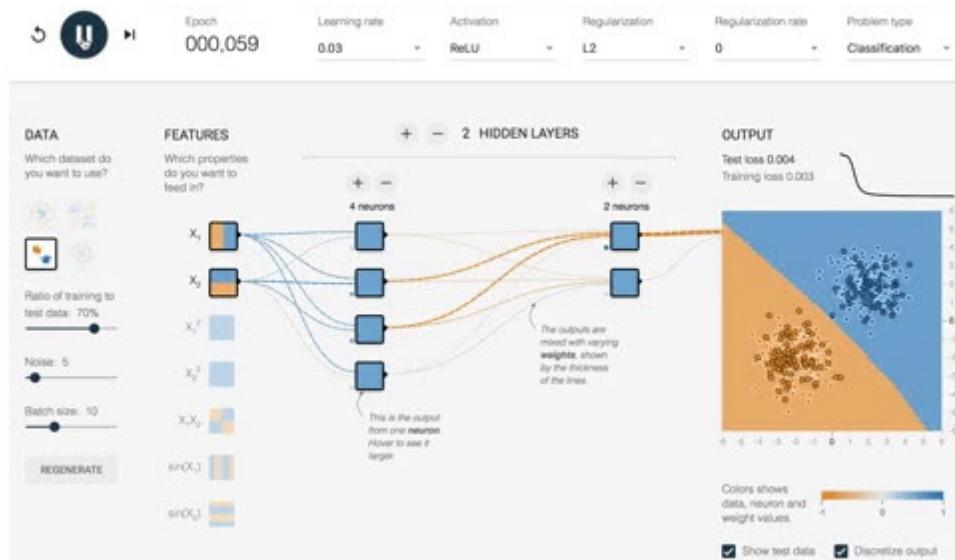
Dropout

Dropout is an odd but useful technique. A network with dropout means that some weights will be randomly set to zero. Imagine you have an array of weights [0.1, 1.7, 0.7, -0.9]. If the neural network has a dropout, it will become [0.1, 0, 0, -0.9] with randomly distributed 0. The parameter that controls the dropout is the dropout rate. The rate defines how many weights to be set to zeroes. Having a rate between 0.2 and 0.5 is common.

Example Neural Network in TensorFlow

Let's see in action how a neural network works for a typical classification problem. There are two inputs, x_1 and x_2 with a random value. The output is a binary class. The objective is to classify the label based on the two features. To carry out this task, the neural network architecture is defined as following:

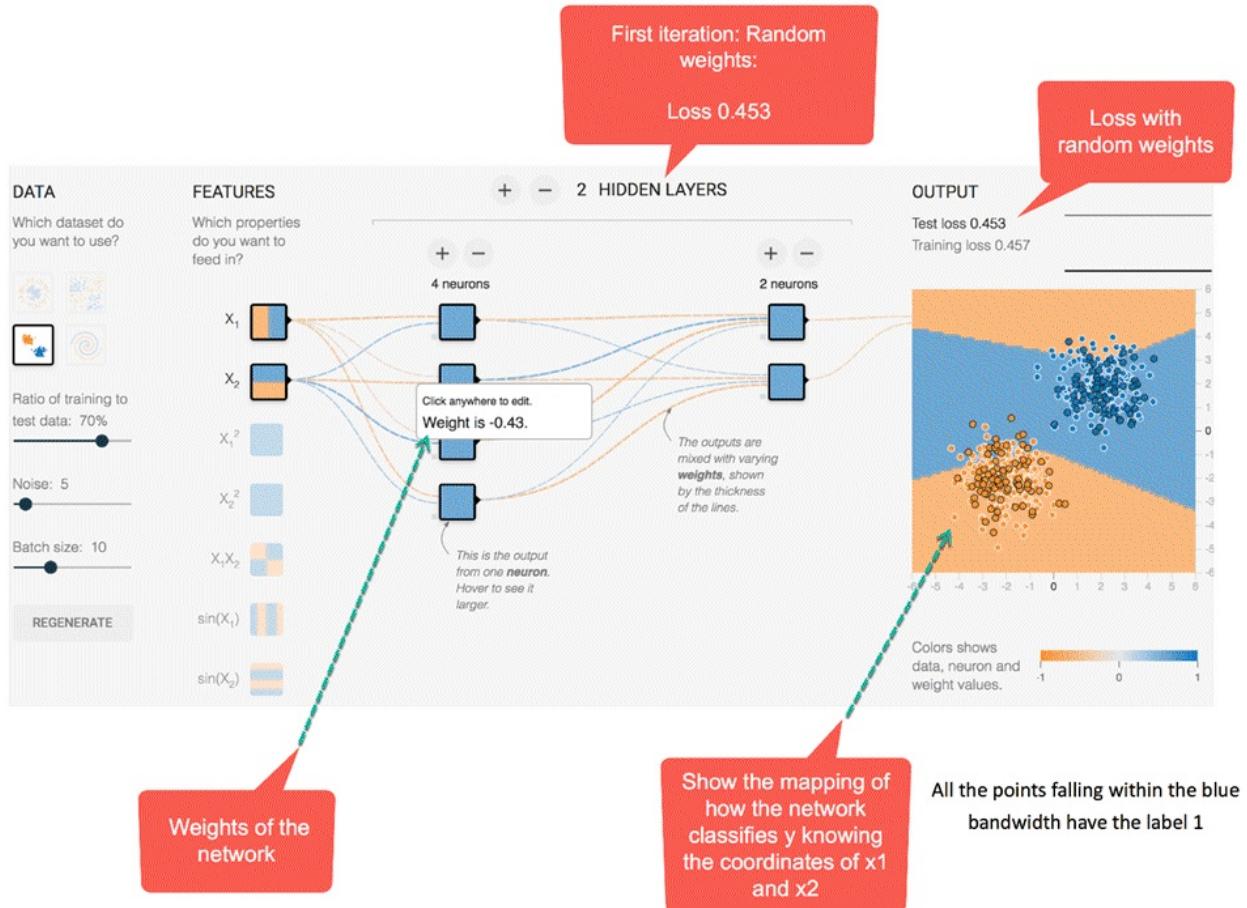
- Two hidden layers
 - First layer has four fully connected neurons
 - Second layer has two fully connected neurons
- The activation function is a Relu
- Add an L2 Regularization with a learning rate of 0.003



The network will optimize the weight during 180 epochs with a batch size of 10. In the video below you can see how the weights evolve over and how the network improves the classification mapping.

First of all, the network assigns random values to all the weights.

- With the random weights, i.e., without optimization, the output loss is 0.453. The picture below represents the network with different colors.
- In general, the orange color represents negative values while the blue colors show the positive values.
- The data points have the same representation; the blue ones are the positive labels and the orange one the negative labels.

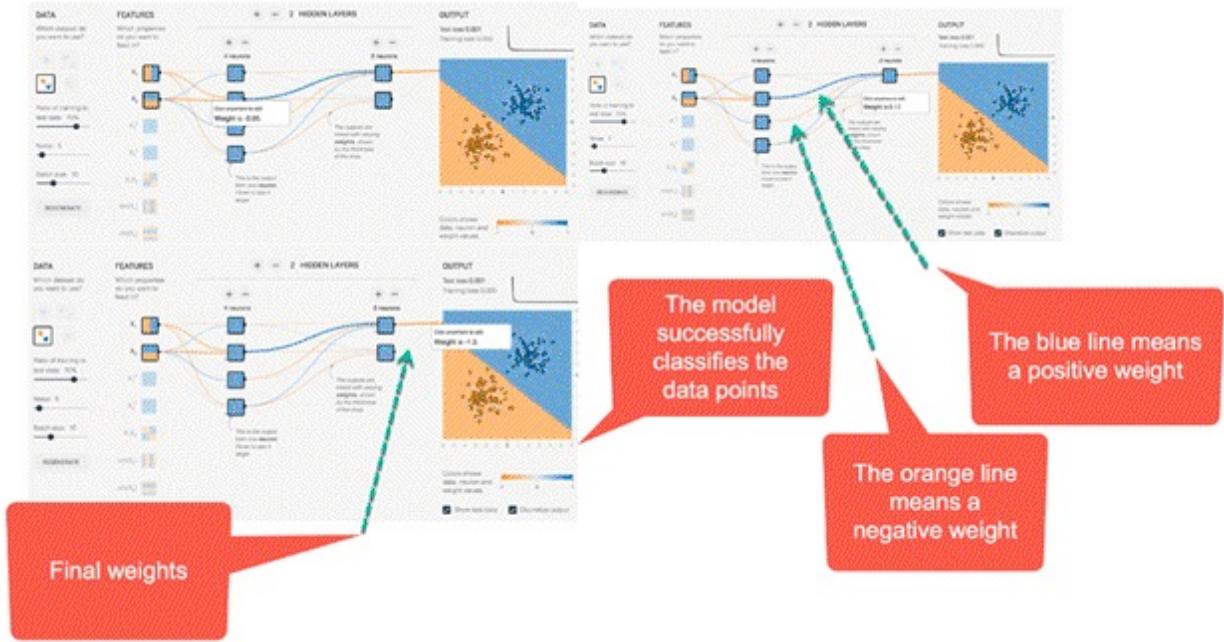


Inside the second hidden layer, the lines are colored following the sign of the weights. The orange lines assign negative weights and the blue one a positive weights

As you can see, in the output mapping, the network is making quite a lot of mistake. Let's see how the network behaves after optimization.

The picture below depicts the results of the optimized network. First of all, you

notice the network has successfully learned how to classify the data point. You can see from the picture before; the initial weight was -0.43 while after optimization it results in a weight of -0.95.



The idea can be generalized for networks with more hidden layers and neurons. You can play around in the link.

Train a neural network with TensorFlow

In this part of the tutorial, you will learn how to train a neural network with TensorFlow using the API's estimator DNNClassifier.

We will use the MNIST dataset to train your first neural network. Training a neural network with Tensorflow is not very complicated. The preprocessing step looks precisely the same as in the previous tutorials. You will proceed as follow:

- Step 1: Import the data
- Step 2: Transform the data
- Step 3: Construct the tensor
- Step 4: Build the model
- Step 5: Train and evaluate the model
- Step 6: Improve the model

Step 1) Import the data

First of all, you need to import the necessary library. You can import the MNIST dataset using scikit learn.

The MNIST dataset is the commonly used dataset to test new techniques or algorithms. This dataset is a collection of 28x28 pixel image with a handwritten digit from 0 to 9. Currently, the lowest error on the test is 0.27 percent with a committee of 7 convolutional neural networks.

```
import numpy as np
import tensorflow as tf
np.random.seed(1337)
```

You can download scikit learn temporarily at this address. Copy and paste the dataset in a convenient folder. To import the data to python, you can use

fetch_mldata from scikit learn. Paste the file path inside fetch_mldata to fetch the data.

```
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata(
    '/Users/Thomas/Dropbox/Learning/Upwork/tuto_TF/data/mldata/MNIST
original')
print(mnist.data.shape)
print(mnist.target.shape)
```

After that, you import the data and get the shape of both datasets.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(mnist.data,
mnist.target, test_size=0.2, random_state=42)
y_train = y_train.astype(int)
y_test = y_test.astype(int)
batch_size = len(X_train)

print(X_train.shape, y_train.shape, y_test.shape )
```

Step 2) Transform the data

In the previous tutorial, you learnt that you need to transform the data to limit the effect of outliers. In this tutorial, you will transform the data using the min-max scaler. The formula is:

$$(X - \min_X) / (\max_X - \min_X)$$

Scikit learns has already a function for that: MinMaxScaler()

```
## resclae
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
# Train
X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
# test
X_test_scaled = scaler.fit_transform(X_test.astype(np.float64))
```

Step 3) Construct the tensor

You are now familiar with the way to create tensor in Tensorflow. You can convert the train set to a numeric column.

```
feature_columns = [tf.feature_column.numeric_column('x',
shape=X_train_scaled.shape[1:])]
```

Step 4) Build the model

The architecture of the neural network contains 2 hidden layers with 300 units for the first layer and 100 units for the second one. We use these value based on our own experience. You can tune theses values and see how it affects the accuracy of the network.

To build the model, you use the estimator DNNClassifier. You can add the number of layers to the feature_columns arguments. You need to set the number of classes to 10 as there are ten classes in the training set. You are already familiar with the syntax of the estimator object. The arguments features columns, number of classes and model_dir are precisely the same as in the previous tutorial. The new argument hidden_unit controls for the number of layers and how many nodes to connect to the neural network. In the code below, there are two hidden layers with a first one connecting 300 nodes and the second one with 100 nodes.

To build the estimator, use `tf.estimator.DNNClassifier` with the following parameters:

- `feature_columns`: Define the columns to use in the network
- `hidden_units`: Define the number of hidden neurons
- `n_classes`: Define the number of classes to predict
- `model_dir`: Define the path of TensorBoard

```
estimator = tf.estimator.DNNClassifier(  
    feature_columns=feature_columns,  
    hidden_units=[300, 100],  
    n_classes=10,  
    model_dir = '/train/DNN')
```

Step 5) Train and evaluate the model

You can use the numpy method to train the model and evaluate it

```
# Train the estimator  
train_input = tf.estimator.inputs.numpy_input_fn(  
    x={"x": X_train_scaled},  
    y=y_train,  
    batch_size=50,  
    shuffle=False,  
    num_epochs=None)  
estimator.train(input_fn = train_input, steps=1000)  
eval_input = tf.estimator.inputs.numpy_input_fn(  
    x={"x": X_test_scaled},  
    y=y_test,  
    shuffle=False,  
    batch_size=X_test_scaled.shape[0],  
    num_epochs=1)  
estimator.evaluate(eval_input, steps=None)
```

Output:

```
{'accuracy': 0.9637143,  
 'average_loss': 0.12014342,  
 'loss': 1682.0079,  
 'global_step': 1000}
```

The current architecture leads to an accuracy on the evaluation set of 96 percent.

Step 6) Improve the model

You can try to improve the model by adding regularization parameters.

We will use an Adam optimizer with a dropout rate of 0.3, L1 of X and L2 of y. In TensorFlow, you can control the optimizer using the object train following by the name of the optimizer. TensorFlow is a built-in API for Proximal AdaGrad optimizer.

To add regularization to the deep neural network, you can use `tf.train.ProximalAdagradOptimizer` with the following parameter

- Learning rate: `learning_rate`
- L1 regularization: `l1_regularization_strength`
- L2 regularization: `l2_regularization_strength`

```
estimator_imp = tf.estimator.DNNClassifier(  
    feature_columns=feature_columns,  
    hidden_units=[300, 100],  
    dropout=0.3,  
    n_classes = 10,  
    optimizer=tf.train.ProximalAdagradOptimizer(  
        learning_rate=0.01,  
        l1_regularization_strength=0.01,  
        l2_regularization_strength=0.01  
    ),  
    model_dir = '/train/DNN1')  
estimator_imp.train(input_fn = train_input, steps=1000)  
estimator_imp.evaluate(eval_input, steps=None)
```

Output:

```
{'accuracy': 0.95057142,  
'average_loss': 0.17318928,  
'loss': 2424.6499,  
'global_step': 2000}
```

The values chosen to reduce the over fitting did not improve the model accuracy. Your first model had an accuracy of 96% while the model with L2 regularizer has an accuracy of 95%. You can try with different values and see how it impacts the accuracy.

Summary

In this tutorial, you learn how to build a neural network. A neural network requires:

- Number of hidden layers
- Number of fully connected node
- Activation function
- Optimizer
- Number of classes

In TensorFlow, you can train a neural network for classification problem with:

- `tf.estimator.DNNClassifier`

The estimator requires to specify:

- `feature_columns=feature_columns,`
- `hidden_units=[300, 100]`
- `n_classes=10`
- `model_dir`

You can improve the model by using different optimizers. In this tutorial, you learned how to use Adam Grad optimizer with a learning rate and add a control to prevent overfitting.

Chapter 18: ConvNet(Convolutional Neural Network): TensorFlow Image Classification

What is Convolutional Neural Network?

Convolutional neural network, also known as convnets or CNN, is a well-known method in computer vision applications. This type of architecture is dominant to recognize objects from a picture or video.

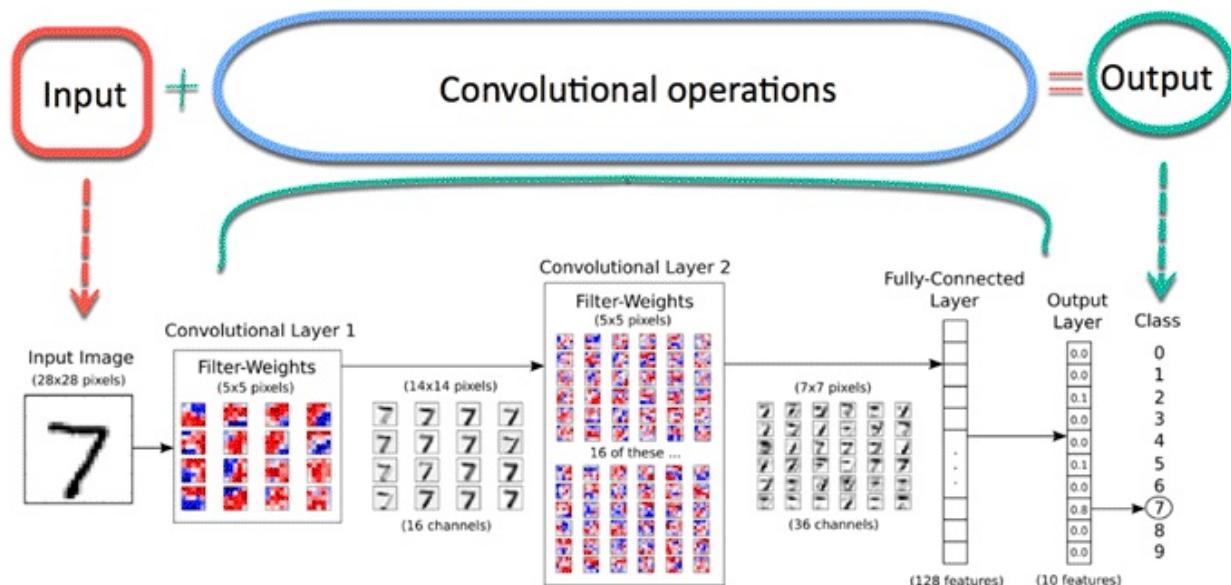
In this tutorial, you will learn how to construct a convnet and how to use TensorFlow to solve the handwritten dataset.

Architecture of a Convolutional Neural Network

Think about Facebook a few years ago, after you uploaded a picture to your profile, you were asked to add a name to the face on the picture manually. Nowadays, Facebook uses convnet to tag your friend in the picture automatically.

A convolutional neural network is not very difficult to understand. An input image is processed during the convolution phase and later attributed a label.

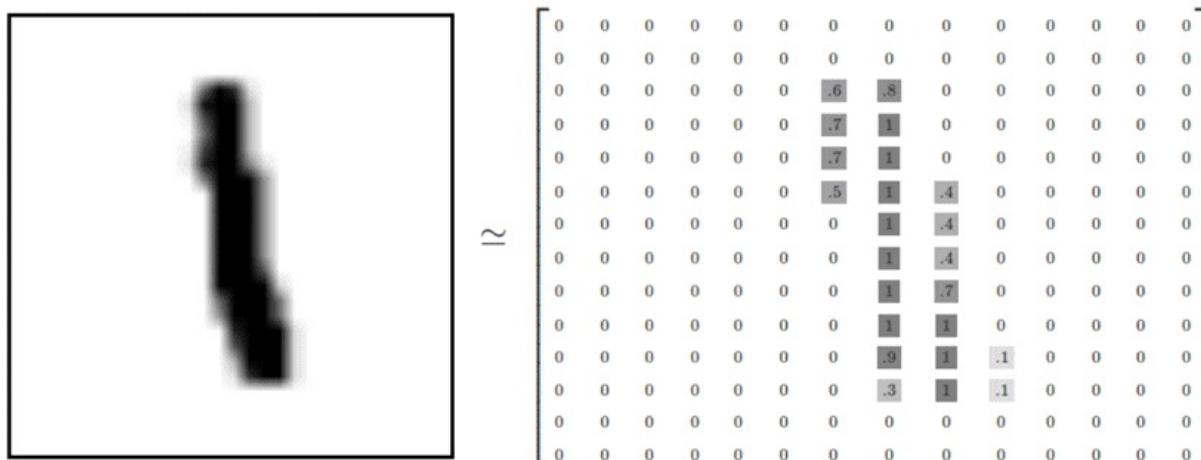
A typical convnet architecture can be summarized in the picture below. First of all, an image is pushed to the network; this is called the input image. Then, the input image goes through an infinite number of steps; this is the convolutional part of the network. Finally, the neural network can predict the digit on the image.



An image is composed of an array of pixels with height and width. A grayscale image has only one channel while the color image has three channels (each one

for Red, Green, and Blue). A channel is stacked over each other. In this tutorial, you will use a grayscale image with only one channel. Each pixel has a value from 0 to 255 to reflect the intensity of the color. For instance, a pixel equals to 0 will show a white color while pixel with a value close to 255 will be darker.

Let's have a look of an image stored in the MNIST dataset. The picture below shows how to represent the picture of the left in a matrix format. Note that, the original matrix has been standardized to be between 0 and 1. For darker color, the value in the matrix is about 0.9 while white pixels have a value of 0.



Convolutional operation

The most critical component in the model is the convolutional layer. This part aims at reducing the size of the image for faster computations of the weights and improve its generalization.

During the convolutional part, the network keeps the essential features of the image and excludes irrelevant noise. For instance, the model is learning how to recognize an elephant from a picture with a mountain in the background. If you use a traditional neural network, the model will assign a weight to all the pixels, including those from the mountain which is not essential and can mislead the network.

Instead, a convolutional neural network will use a mathematical technique to extract only the most relevant pixels. This mathematical operation is called convolution. This technique allows the network to learn increasingly complex features at each layer. The convolution divides the matrix into small pieces to learn to most essential elements within each piece.

Components of Convnets

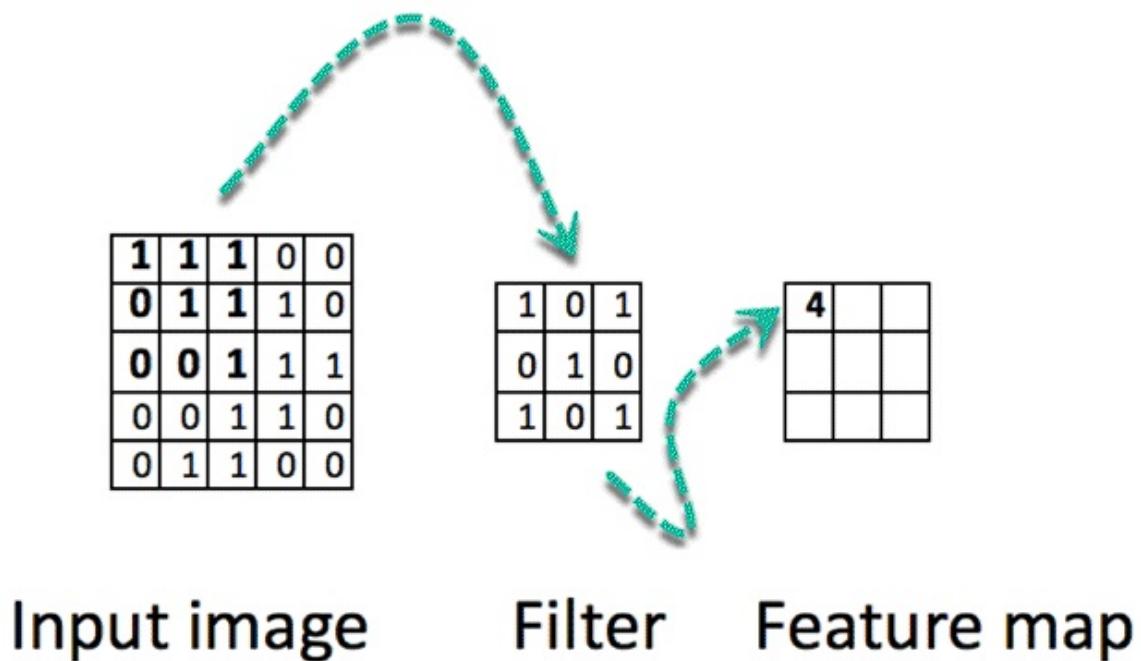
There are four components of a Convnets

1. Convolution
 2. Non Linearity (ReLU)
 3. Pooling or Sub Sampling
 4. Classification (Fully Connected Layer)
- Convolution

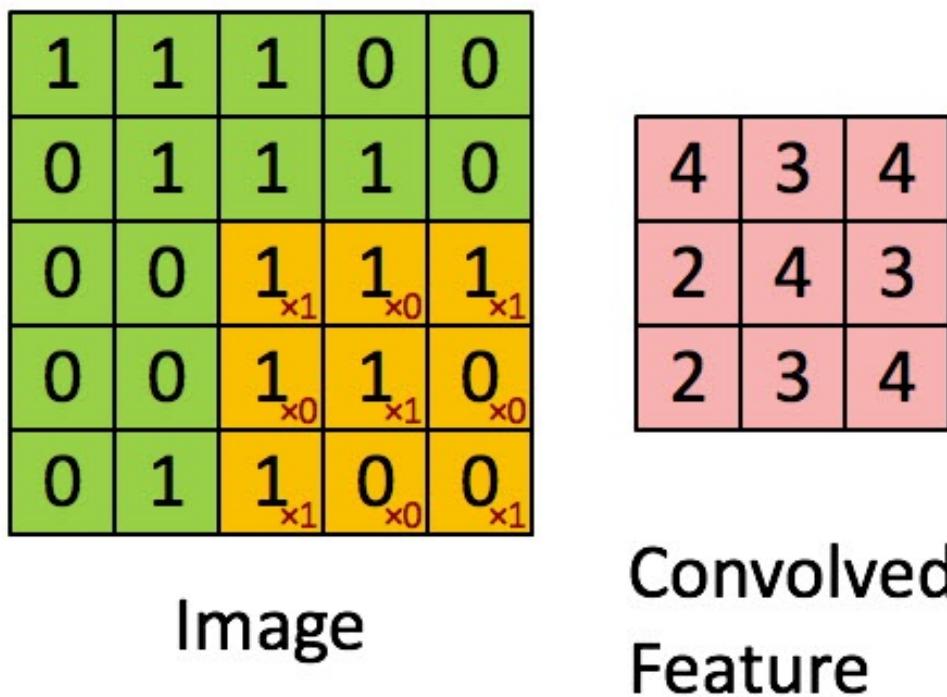
The purpose of the convolution is to extract the features of the object on the image locally. It means the network will learn specific patterns within the picture and will be able to recognize it everywhere in the picture.

Convolution is an element-wise multiplication. The concept is easy to understand. The computer will scan a part of the image, usually with a dimension of 3x3 and multiplies it to a filter. The output of the element-wise multiplication is called a feature map. This step is repeated until all the image is scanned. Note that, after the convolution, the size of the image is reduced.

Convolution



Below, there is a URL to see in action how convolution works.

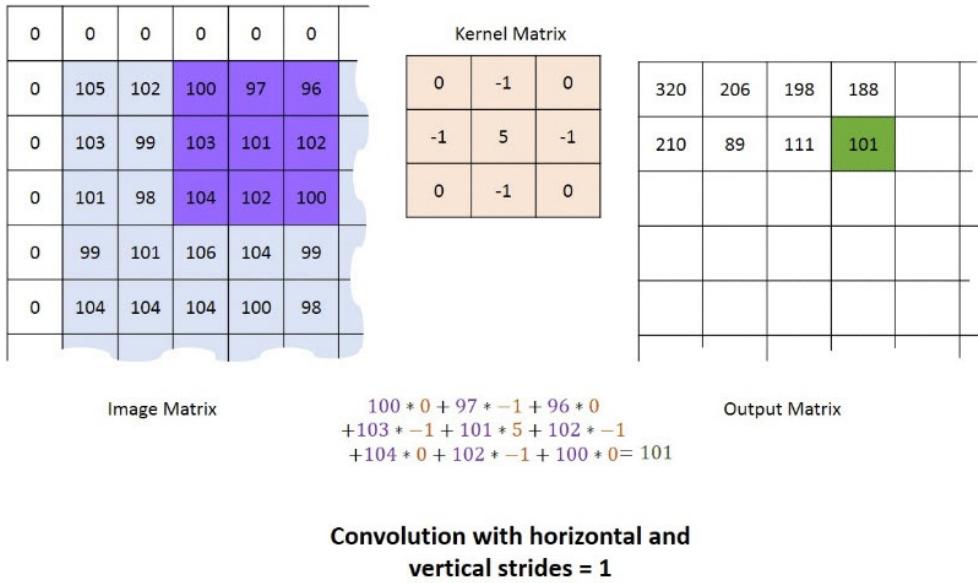


There are numerous channels available. Below, we listed some of the channels. You can see that each filter has a specific purpose. Note, in the picture below; the Kernel is a synonym of the filter.

Operation	Kernel	Image result
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Arithmetic behind the convolution

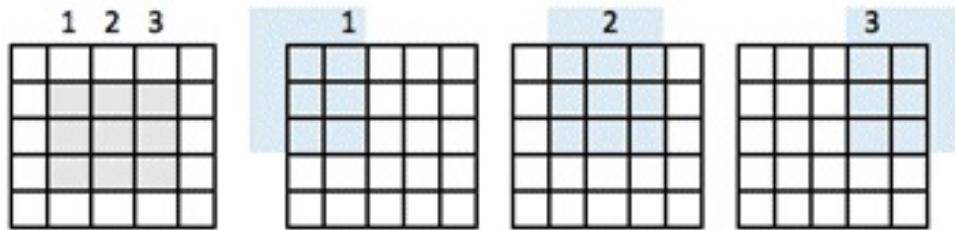
The convolutional phase will apply the filter on a small array of pixels within the picture. The filter will move along the input image with a general shape of 3x3 or 5x5. It means the network will slide these windows across all the input image and compute the convolution. The image below shows how the convolution operates. The size of the patch is 3x3, and the output matrix is the result of the element-wise operation between the image matrix and the filter.



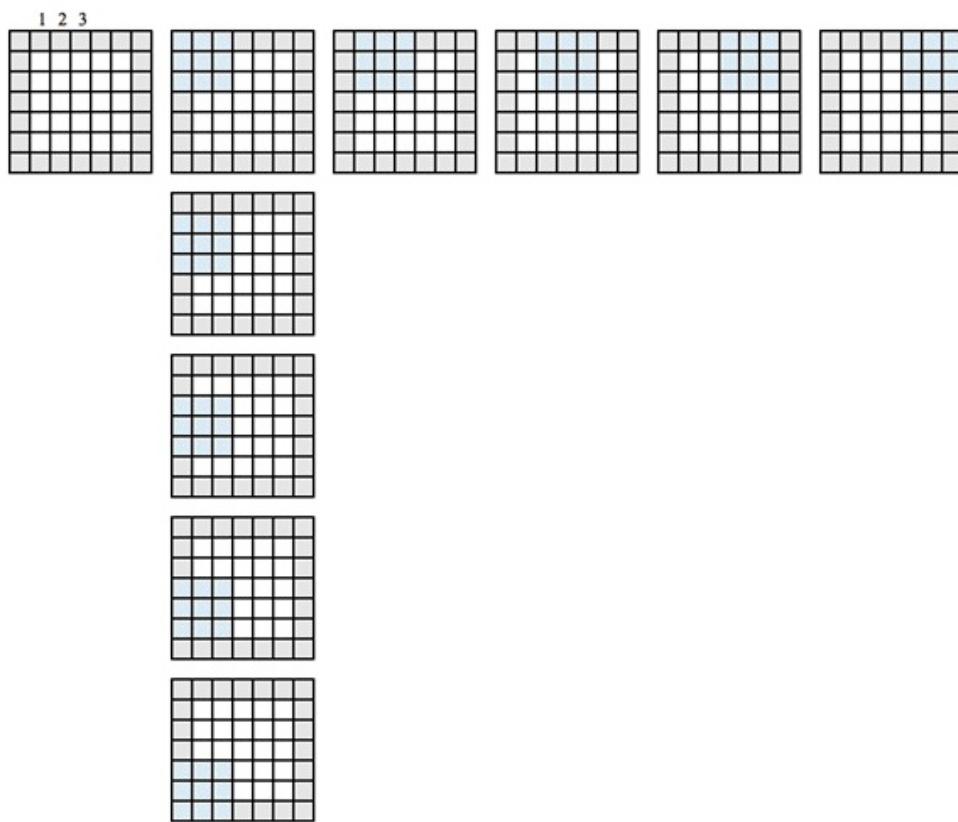
You notice that the width and height of the output can be different from the width and height of the input. It happens because of the border effect.

Border effect

Image has a 5x5 features map and a 3x3 filter. There is only one window in the center where the filter can screen an 3x3 grid. The output feature map will shrink by two tiles alongside with a 3x3 dimension.



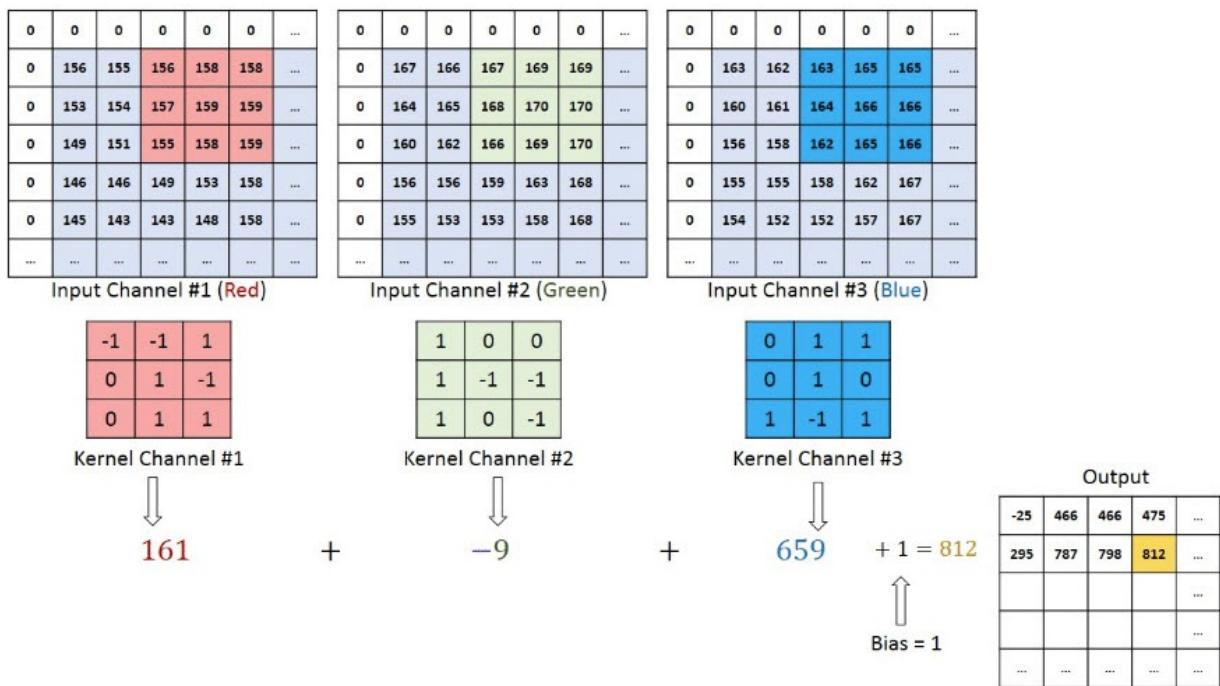
To get the same output dimension as the input dimension, you need to add padding. Padding consists of adding the right number of rows and columns on each side of the matrix. It will allow the convolution to center fit every input tile. In the image below, the input/output matrix have the same dimension 5x5



When you define the network, the convolved features are controlled by three parameters:

1. Depth: It defines the number of filters to apply during the convolution. In the previous example, you saw a depth of 1, meaning only one filter is used.

In most of the case, there is more than one filter. The picture below shows the operations done in a situation with three filters



2. Stride: It defines the number of "pixel's jump" between two slices. If the stride is equal to 1, the windows will move with a pixel's spread of one. If the stride is equal to two, the windows will jump by 2 pixels. If you increase the stride, you will have smaller feature maps.

Example stride 1

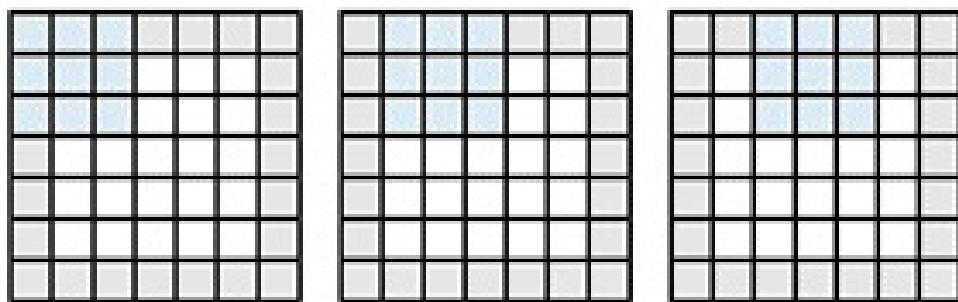
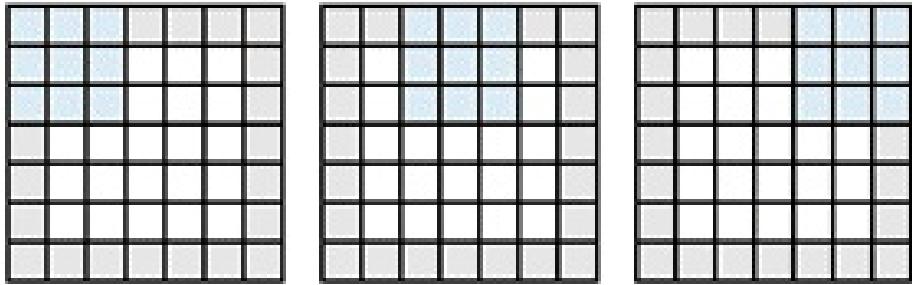


Image stride 2



3. Zero-padding: A padding is an operation of adding a corresponding number of rows and column on each side of the input features maps. In this case, the output has the same dimension as the input.

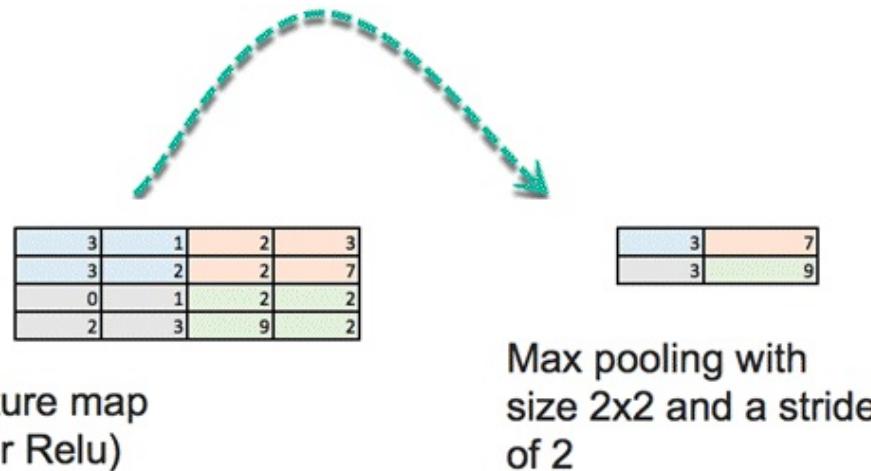
4. Non Linearity (ReLU)

At the end of the convolution operation, the output is subject to an activation function to allow non-linearity. The usual activation function for convnet is the Relu. All the pixel with a negative value will be replaced by zero.

- Max-pooling operation

This step is easy to understand. The purpose of the pooling is to reduce the dimensionality of the input image. The steps are done to reduce the computational complexity of the operation. By diminishing the dimensionality, the network has lower weights to compute, so it prevents overfitting.

In this stage, you need to define the size and the stride. A standard way to pool the input image is to use the maximum value of the feature map. Look at the picture below. The "pooling" will screen a four submatrix of the 4x4 feature map and return the maximum value. The pooling takes the maximum value of a 2x2 array and then move this windows by two pixels. For instance, the first submatrix is [3,1,3,2], the pooling will return the maximum, which is 3.



There is another pooling operation such as the mean.

This operation aggressively reduces the size of the feature map

- Fully connected layers

The last step consists of building a traditional artificial neural network as you did in the previous tutorial. You connect all neurons from the previous layer to the next layer. You use a softmax activation function to classify the number on the input image.

Recap:

Convolutional Neural network compiles different layers before making a prediction. A neural network has:

- A convolutional layer
- Relu Activation function
- Pooling layer
- Densely connected layer

The convolutional layers apply different filters on a subregion of the picture. The Relu activation function adds non-linearity, and the pooling layers reduce the

dimensionality of the features maps.

All these layers extract essential information from the images. At last, the features map are feed to a primary fully connected layer with a softmax function to make a prediction.

Train CNN with TensorFlow

Now that you are familiar with the building block of a convnets, you are ready to build one with TensorFlow. We will use the MNIST dataset for image classification.

The data preparation is the same as the previous tutorial. You can run the codes and jump directly to the architecture of the CNN.

You will follow the steps below:

Step 1: Upload Dataset

Step 2: Input layer

Step 3: Convolutional layer

Step 4: Pooling layer

Step 5: Second Convolutional Layer and Pooling Layer

Step 6: Dense layer

Step 7: Logit Layer

Step 1: Upload Dataset

The MNIST dataset is available with scikit to learn at this URL. Please download it and store it in Downloads. You can upload it with `fetch_mldata('MNIST original')`.

Create a train/test set

You need to split the dataset with train_test_split

Scale the features

Finally, you can scale the feature with MinMaxScaler

```
import numpy as np
import tensorflow as tf
from sklearn.datasets import fetch_mldata

#Change USERNAME by the username of your machine
## Windows USER
mnist = fetch_mldata('C:\\\\Users\\\\USERNAME\\\\Downloads\\\\MNIST
original')
## Mac User
mnist = fetch_mldata('/Users/USERNAME/Downloads/MNIST original')

print(mnist.data.shape)
print(mnist.target.shape)
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(mnist.data,
mnist.target, test_size=0.2, random_state=42)
y_train = y_train.astype(int)
y_test = y_test.astype(int)
batch_size =len(X_train)

print(X_train.shape, y_train.shape,y_test.shape )
## resclae
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
# Train
X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
# test
X_test_scaled = scaler.fit_transform(X_test.astype(np.float64))
feature_columns = [tf.feature_column.numeric_column('x',
shape=X_train_scaled.shape[1:])]

X_train_scaled.shape[1:]
```

Define the CNN

A CNN uses filters on the raw pixel of an image to learn details pattern compare

to global pattern with a traditional neural net. To construct a CNN, you need to define:

1. A convolutional layer: Apply n number of filters to the feature map. After the convolution, you need to use a Relu activation function to add non-linearity to the network.
2. Pooling layer: The next step after the convolution is to downsample the feature map. The purpose is to reduce the dimensionality of the feature map to prevent overfitting and improve the computation speed. Max pooling is the conventional technique, which divides the feature maps into subregions (usually with a 2x2 size) and keeps only the maximum values.
3. Fully connected layers: All neurons from the previous layers are connected to the next layers. The CNN will classify the label according to the features from the convolutional layers and reduced with the pooling layer.

CNN architecture

- Convolutional Layer: Applies 14 5x5 filters (extracting 5x5-pixel subregions), with ReLU activation function
- Pooling Layer: Performs max pooling with a 2x2 filter and stride of 2 (which specifies that pooled regions do not overlap)
- Convolutional Layer: Applies 36 5x5 filters, with ReLU activation function
- Pooling Layer #2: Again, performs max pooling with a 2x2 filter and stride of 2
- 1,764 neurons, with dropout regularization rate of 0.4 (probability of 0.4 that any given element will be dropped during training)
- Dense Layer (Logits Layer): 10 neurons, one for each digit target class (0–9).

There are three important modules to use to create a CNN:

- `conv2d()`. Constructs a two-dimensional convolutional layer with the number of filters, filter kernel size, padding, and activation function as

arguments.

- `max_pooling2d()`. Constructs a two-dimensional pooling layer using the max-pooling algorithm.
- `dense()`. Constructs a dense layer with the hidden layers and units

You will define a function to build the CNN. Let's see in detail how to construct each building block before to wrap everything together in the function.

Step 2: Input layer

```
def cnn_model_fn(features, labels, mode):
    input_layer = tf.reshape(tensor = features["x"], shape =[-1, 28,
28, 1])
```

You need to define a tensor with the shape of the data. For that, you can use the module `tf.reshape`. In this module, you need to declare the tensor to reshape and the shape of the tensor. The first argument is the features of the data, which is defined in the argument of the function.

A picture has a height, a width, and a channel. The MNIST dataset is a monochromatic picture with a 28x28 size. We set the batch size to -1 in the shape argument so that it takes the shape of the `features["x"]`. The advantage is to make the batch size hyperparameters to tune. If the batch size is set to 7, then the tensor will feed 5,488 values ($28*28*7$).

Step 3: Convolutional layer

```
# first Convolutional Layer
conv1 = tf.layers.conv2d(
    inputs=input_layer,
    filters=14,
    kernel_size=[5, 5],
    padding="same",
    activation=tf.nn.relu)
```

The first convolutional layer has 14 filters with a kernel size of 5x5 with the

same padding. The same padding means both the output tensor and input tensor should have the same height and width. Tensorflow will add zeros to the rows and columns to ensure the same size.

You use the Relu activation function. The output size will be [28, 28, 14].

Step 4: Pooling layer

The next step after the convolution is the pooling computation. The pooling computation will reduce the dimensionality of the data. You can use the module max_pooling2d with a size of 2x2 and stride of 2. You use the previous layer as input. The output size will be [batch_size, 14, 14, 14]

```
# first Pooling Layer
pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2],
strides=2)
```

Step 5: Second Convolutional Layer and Pooling Layer

The second convolutional layer has 32 filters, with an output size of [batch_size, 14, 14, 32]. The pooling layer has the same size as before and the output shape is [batch_size, 14, 14, 18].

```
conv2 = tf.layers.conv2d(
    inputs=pool1,
    filters=36,
    kernel_size=[5, 5],
    padding="same",
    activation=tf.nn.relu)
pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2],
strides=2)
```

Step 6: Dense layer

Then, you need to define the fully-connected layer. The feature map has to be flatten before to be connected with the dense layer. You can use the module reshape with a size of 7*7*36.

The dense layer will connect 1764 neurons. You add a Relu activation function. Besides, you add a dropout regularization term with a rate of 0.3, meaning 30 percents of the weights will be set to 0. Note that, the dropout takes place only during the training phase. The function cnn_model_fn has an argument mode to declare if the model needs to be trained or to evaluate.

```
pool2_flat = tf.reshape(pool2, [-1, 7 * 7 * 36])

dense = tf.layers.dense(inputs=pool2_flat, units=7 * 7 * 36,
activation=tf.nn.relu)
dropout = tf.layers.dropout(
    inputs=dense, rate=0.3, training=mode ==
tf.estimator.ModeKeys.TRAIN)
```

Step 7: Logit Layer

Finally, you can define the last layer with the prediction of the model. The output shape is equal to the batch size and 10, the total number of images.

```
# Logits Layer
logits = tf.layers.dense(inputs=dropout, units=10)
```

You can create a dictionary containing the classes and the probability of each class. The module tf.argmax() with returns the highest value if the logit layers. The softmax function returns the probability of each class.

```
predictions = {
    # Generate predictions
    "classes": tf.argmax(input=logits, axis=1),
    "probabilities": tf.nn.softmax(logits, name="softmax_tensor")
}
```

You only want to return the dictionnary prediction when mode is set to

prediction. You add this codes to display the predictions

```
if mode == tf.estimator.ModeKeys.PREDICT:  
    return tf.estimator.EstimatorSpec(mode=mode,  
predictions=predictions)
```

The next step consists to compute the loss of the model. In the last tutorial, you learnt that the loss function for a multiclass model is cross entropy. The loss is easily computed with the following code:

```
# Calculate Loss (for both TRAIN and EVAL modes)  
loss = tf.losses.sparse_softmax_cross_entropy(labels=labels,  
logits=logits)
```

The final step is to optimize the model, that is to find the best values of the weights. For that, you use a Gradient descent optimizer with a learning rate of 0.001. The objective is to minimize the loss

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)  
train_op = optimizer.minimize(  
    loss=loss,  
    global_step=tf.train.get_global_step())
```

You are done with the CNN. However, you want to display the performance metrics during the evaluation mode. The performance metrics for a multiclass model is the accuracy metrics. Tensorflow is equipped with a module accuracy with two arguments, the labels, and the predicted values.

```
eval_metric_ops = {  
    "accuracy": tf.metrics.accuracy(labels=labels,  
predictions=predictions["classes"])}  
return tf.estimator.EstimatorSpec(mode=mode, loss=loss,  
eval_metric_ops=eval_metric_ops)
```

That's it. You created your first CNN and you are ready to wrap everything into a function in order to use it to train and evaluate the model.

```
def cnn_model_fn(features, labels, mode):  
    """Model function for CNN."""
```

```

# Input Layer
input_layer = tf.reshape(features["x"], [-1, 28, 28, 1])

# Convolutional Layer
conv1 = tf.layers.conv2d(
    inputs=input_layer,
    filters=32,
    kernel_size=[5, 5],
    padding="same",
    activation=tf.nn.relu)

# Pooling Layer
pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2],
strides=2)

# Convolutional Layer #2 and Pooling Layer
conv2 = tf.layers.conv2d(
    inputs=pool1,
    filters=36,
    kernel_size=[5, 5],
    padding="same",
    activation=tf.nn.relu)
pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2],
strides=2)

# Dense Layer
pool2_flat = tf.reshape(pool2, [-1, 7 * 7 * 36])
dense = tf.layers.dense(inputs=pool2_flat, units=7 * 7 * 36,
activation=tf.nn.relu)
dropout = tf.layers.dropout(
    inputs=dense, rate=0.4, training=mode ==
tf.estimator.ModeKeys.TRAIN)

# Logits Layer
logits = tf.layers.dense(inputs=dropout, units=10)

predictions = {
    # Generate predictions (for PREDICT and EVAL mode)
    "classes": tf.argmax(input=logits, axis=1),
    "probabilities": tf.nn.softmax(logits, name="softmax_tensor")
}

if mode == tf.estimator.ModeKeys.PREDICT:
    return tf.estimator.EstimatorSpec(mode=mode,
predictions=predictions)

```

```

# Calculate Loss
loss = tf.losses.sparse_softmax_cross_entropy(labels=labels,
logits=logits)

# Configure the Training Op (for TRAIN mode)
if mode == tf.estimator.ModeKeys.TRAIN:
    optimizer =
tf.train.GradientDescentOptimizer(learning_rate=0.001)
    train_op = optimizer.minimize(
        loss=loss,
        global_step=tf.train.get_global_step())
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss,
train_op=train_op)

# Add evaluation metrics Evaluation mode
eval_metric_ops = {
    "accuracy": tf.metrics.accuracy(
        labels=labels, predictions=predictions["classes"])}
return tf.estimator.EstimatorSpec(
    mode=mode, loss=loss, eval_metric_ops=eval_metric_ops)

```

The steps below are the same as the previous tutorials.

First of all, you define an estimator with the CNN model.

```

# Create the Estimator
mnist_classifier = tf.estimator.Estimator(
    model_fn=cnn_model_fn, model_dir="train/mnist_convnet_model")

```

A CNN takes many times to train, therefore, you create a Logging hook to store the values of the softmax layers every 50 iterations.

```

# Set up logging for predictions
tensors_to_log = {"probabilities": "softmax_tensor"}
logging_hook = tf.train.LoggingTensorHook(tensors=tensors_to_log,
every_n_iter=50)

```

You are ready to estimate the model. You set a batch size of 100 and shuffle the data. Note that we set training steps of 16.000, it can take lots of time to train. Be patient.

```

# Train the model
train_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"x": X_train_scaled},
    y=y_train,
    batch_size=100,
    num_epochs=None,
    shuffle=True)
mnist_classifier.train(
    input_fn=train_input_fn,
    steps=16000,
    hooks=[logging_hook])

```

Now that the model is train, you can evaluate it and print the results

```

# Evaluate the model and print results
eval_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"x": X_test_scaled},
    y=y_test,
    num_epochs=1,
    shuffle=False)
eval_results = mnist_classifier.evaluate(input_fn=eval_input_fn)
print(eval_results)

```

```

INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2018-08-05-12:52:41
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from
train/mnist_convnet_model/model.ckpt-15652
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2018-08-05-12:52:56
INFO:tensorflow:Saving dict for global step 15652: accuracy =
0.9589286, global_step = 15652, loss = 0.13894269
{'accuracy': 0.9689286, 'loss': 0.13894269, 'global_step': 15652}

```

With the current architecture, you get an accuracy of 97%. You can change the architecture, the batch size and the number of iteration to improve the accuracy. The CNN neural network has performed far better than ANN or logistic regression. In the tutorial on artificial neural network, you had an accuracy of 96%, which is lower the CNN. The performances of the CNN are impressive with a larger image **set**, both in term of speed computation and accuracy.

Summary

A convolutional neural network works very well to evaluate picture. This type of architecture is dominant to recognize objects from a picture or video.

To build a CNN, you need to follow six steps:

Step 1: Input layer:

This step reshapes the data. The shape is equal to the square root of the number of pixels. For instance, if a picture has 156 pixels, then the shape is 26x26. You need to specify if the picture has colour or not. If yes, then you had 3 to the shape- 3 for RGB-, otherwise 1.

```
input_layer = tf.reshape(tensor = features["x"], shape =[-1, 28, 28, 1])
```

Step 2: Convolutional layer

Next, you need to create the convolutional layers. You apply different filters to allow the network to learn important feature. You specify the size of the kernel and the amount of filters.

```
conv1 = tf.layers.conv2d(  
    inputs=input_layer,  
    filters=14,  
    kernel_size=[5, 5],  
    padding="same",  
    activation=tf.nn.relu)
```

Step 3: Pooling layer

In the third step, you add a pooling layer. This layer decreases the size of the input. It does so by taking the maximum value of the a sub-matrix. For instance, if the sub-matrix is [3,1,3,2], the pooling will return the maximum, which is 3.

```
pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2],
```

```
strides=2)
```

Step 4: Add Convolutional Layer and Pooling Layer

In this step, you can add as much as you want conv layers and pooling layers. Google uses architecture with more than 20 conv layers.

Step 5: Dense layer

The step 5 flatten the previous to create a fully connected layers. In this step, you can use different activation function and add a dropout effect.

```
pool2_flat = tf.reshape(pool2, [-1, 7 * 7 * 36])

dense = tf.layers.dense(inputs=pool2_flat, units=7 * 7 * 36,
activation=tf.nn.relu)
dropout = tf.layers.dropout(
    inputs=dense, rate=0.3, training=mode ==
tf.estimator.ModeKeys.TRAIN)
```

Step 6: Logit Layer

The final step is the prediction.

```
logits = tf.layers.dense(inputs=dropout, units=10)
```

Chapter 19: Autoencoder with TensorFlow

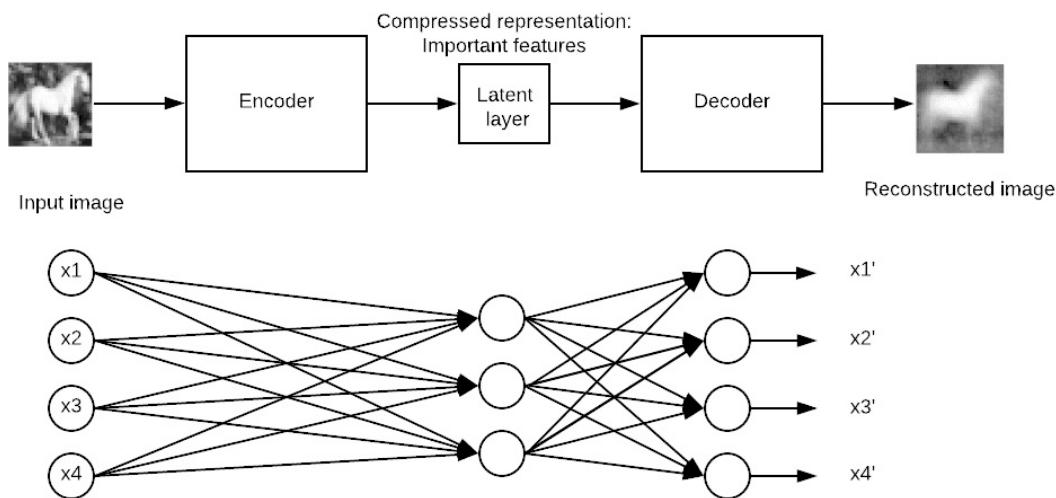
What is an Autoencoder?

An autoencoder is a great tool to recreate an input. In a simple word, the machine takes, let's say an image, and can produce a closely related picture. The input in this kind of neural network is unlabelled, meaning the network is capable of learning without supervision. More precisely, the input is encoded by the network to focus only on the most critical feature. This is one of the reasons why autoencoder is popular for dimensionality reduction. Besides, autoencoders can be used to produce **generative learning models**. For example, the neural network can be trained with a set of faces and then can produce new faces.

How does Autoencoder work?

The purpose of an autoencoder is to produce an approximation of the input by focusing only on the essential features. You may think why not merely learn how to copy and paste the input to produce the output. In fact, an autoencoder is a set of constraints that force the network to learn new ways to represent the data, different from merely copying the output.

A typical autoencoder is defined with an input, an internal representation and an output (an approximation of the input). The learning occurs in the layers attached to the internal representation. In fact, there are two main blocks of layers which looks like a traditional neural network. The slight difference is the layer containing the output must be equal to the input. In the picture below, the original input goes into the first block called the **encoder**. This internal representation compresses (reduces) the size of the input. In the second block occurs the reconstruction of the input. This is the decoding phase.



The model will update the weights by minimizing the loss function. The model is penalized if the reconstruction output is different from the input.

Concretely, imagine a picture with a size of 50x50 (i.e., 250 pixels) and a neural network with just one hidden layer composed of one hundred neurons. The learning is done on a feature map which is two times smaller than the input. It means the network needs to find a way to reconstruct 250 pixels with only a vector of neurons equal to 100.

Stacked Autoencoder Example

In this tutorial, you will learn how to use a stacked autoencoder. The architecture is similar to a traditional neural network. The input goes to a hidden layer in order to be compressed, or reduce its size, and then reaches the reconstruction layers. The objective is to produce an output image as close as the original. The model has to learn a way to achieve its task under a set of constraints, that is, with a lower dimension.

Nowadays, autoencoders are mainly used to denoise an image. Imagine an image with scratches; a human is still able to recognize the content. The idea of denoising autoencoder is to add noise to the picture to force the network to learn the pattern behind the data.

The other useful family of autoencoder is variational autoencoder. This type of network can generate new images. Imagine you train a network with the image of a man; such a network can produce new faces.

Build an Autoencoder with TensorFlow

In this tutorial, you will learn how to build a stacked autoencoder to reconstruct an image.

You will use the CIFAR-10 dataset which contains 60000 32x32 color images. The dataset is already split between 50000 images for training and 10000 for testing. There are up to ten classes:

- Airplane
- Automobile
- Bird
- Cat
- Deer
- Dog
- Frog
- Horse
- Ship
- Truck

You need download the images in this URL

<https://www.cs.toronto.edu/~kriz/cifar.html> and unzip it. The folder for-10-batches-py contains five batches of data with 10000 images each in a random order.

Before you build and train your model, you need to apply some data processing. You will proceed as follow:

1. Import the data
2. Convert the data to black and white format
3. Append all the batches
4. Construct the training dataset

5. Construct an image visualizer

Image preprocessing

Step 1) Import the data.

According to the official website, you can upload the data with the following code. The code will load the data in a dictionary with the **data** and the **label**. Note that the code is a function.

```
import numpy as np
import tensorflow as tf
import pickle
def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='latin1')
    return dict
```

Step 2) Convert the data to black and white format

For simplicity, you will convert the data to a grayscale. That is, with only one dimension against three for colors image. Most of the neural network works only with one dimension input.

```
def grayscale(im):
    return im.reshape(im.shape[0], 3, 32,
32).mean(1).reshape(im.shape[0], -1)
```

Step 3) Append all the batches

Now that both functions are created and the dataset loaded, you can write a loop to append the data in memory. If you check carefully, the unzip file with the data is named `data_batch_` with a number from 1 to 5. You can loop over the files and append it to `data`.

When this step is done, you convert the colours data to a gray scale format. As

you can see, the shape of the data is 50000 and 1024. The 32*32 pixels are now flatten to 2014.

```
# Load the data into memory
data, labels = [], []
## Loop over the b
for i in range(1, 6):
    filename = './cifar-10-batches-py/data_batch_' + str(i)
    open_data = unpickle(filename)
    if len(data) > 0:
        data = np.vstack((data, open_data['data']))
        labels = np.hstack((labels, open_data['labels']))
    else:
        data = open_data['data']
        labels = open_data['labels']

data = grayscale(data)
x = np.matrix(data)
y = np.array(labels)
print(x.shape)
(50000, 1024)
```

Note: Change './cifar-10-batches-py/data_batch_' to the actual location of your file. For instance for Windows machine, the path could be filename = 'E:\cifar-10-batches-py\data_batch_' + str(i)

Step 4) Construct the training dataset

To make the training faster and easier, you will train a model on the horse images only. The horses are the seventh class in the label data. As mentioned in the documentation of the CIFAR-10 dataset, each class contains 5000 images. You can print the shape of the data to confirm there are 5.000 images with 1024 columns.

```
horse_i = np.where(y == 7)[0]
horse_x = x[horse_i]
print(np.shape(horse_x))
(5000, 1024)
```

Step 5) Construct an image visualizer

Finally, you construct a function to plot the images. You will need this function to print the reconstructed image from the autoencoder.

An easy way to print images is to use the object imshow from the matplotlib library. Note that, you need to convert the shape of the data from 1024 to 32*32 (i.e. format of an image).

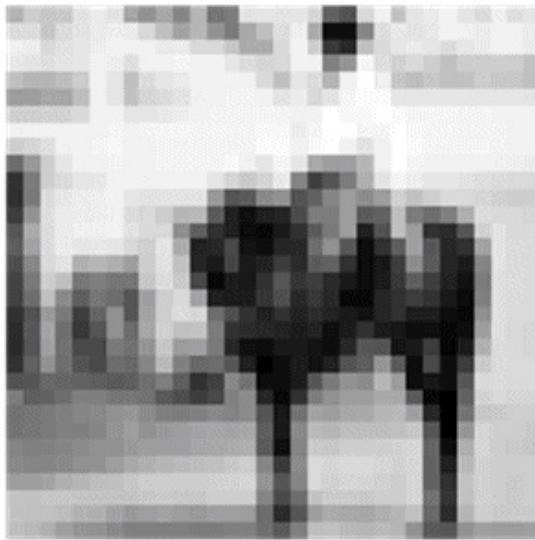
```
# To plot pretty figures
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
def plot_image(image, shape=[32, 32], cmap = "Greys_r"):
    plt.imshow(image.reshape(shape),
cmap=cmap, interpolation="nearest")
    plt.axis("off")
```

The function takes 3 arguments:

- Image: the input
- Shape: list, the dimension of the image
- Cmap:choose the color map. By default, grey

You can try to plot the first image in the dataset. You should see a man on a horse.

```
plot_image(horse_x[1], shape=[32, 32], cmap = "Greys_r")
```



Set Dataset Estimator

All right, now that the dataset is ready to use, you can start to use Tensorflow. Before to build the model, let's use the Dataset estimator of Tensorflow to feed the network.

You will build a Dataset with TensorFlow estimator. To refresh your mind, you need to use:

- from_tensor_slices
- repeat
- batch

The full code to build the dataset is:

```
dataset =  
tf.data.Dataset.from_tensor_slices(x).repeat().batch(batch_size)
```

Note that, x is a placeholder with the following shape:

- [None,n_inputs]: Set to None because the number of image feed to the

network is equal to the batch size.

for details, please refer to the tutorial on linear regression.

After that, you need to create the iterator. Without this line of code, no data will go through the pipeline.

```
iter = dataset.make_initializable_iterator() # create the iterator
features = iter.get_next()
```

Now that the pipeline is ready, you can check if the first image is the same as before (i.e., a man on a horse).

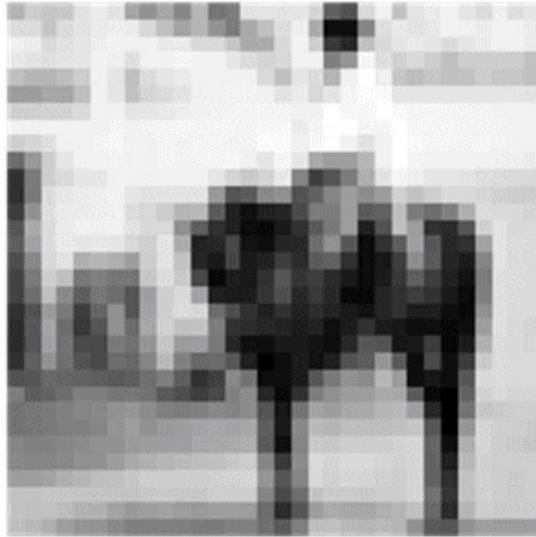
You set the batch size to 1 because you only want to feed the dataset with one image. You can see the dimension of the data with print(sess.run(features).shape). It is equal to (1, 1024). 1 means only one image with 1024 is feed each. If the batch size is set to two, then two images will go through the pipeline. (Don't change the batch size. Otherwise, it will throw an error. Only one image at a time can go to the function plot_image()).

```
## Parameters
n_inputs = 32 * 32
BATCH_SIZE = 1
batch_size = tf.placeholder(tf.int64)

# using a placeholder
x = tf.placeholder(tf.float32, shape=[None, n_inputs])
## Dataset
dataset =
tf.data.Dataset.from_tensor_slices(x).repeat().batch(batch_size)
iter = dataset.make_initializable_iterator() # create the iterator
features = iter.get_next()

## Print the image
with tf.Session() as sess:
    # feed the placeholder with data
    sess.run(iter.initializer, feed_dict={x: horse_x,
                                          batch_size: BATCH_SIZE})
    print(sess.run(features).shape)
    plot_image(sess.run(features), shape=[32, 32], cmap =
```

```
"Greys_r")  
(1, 1024)
```

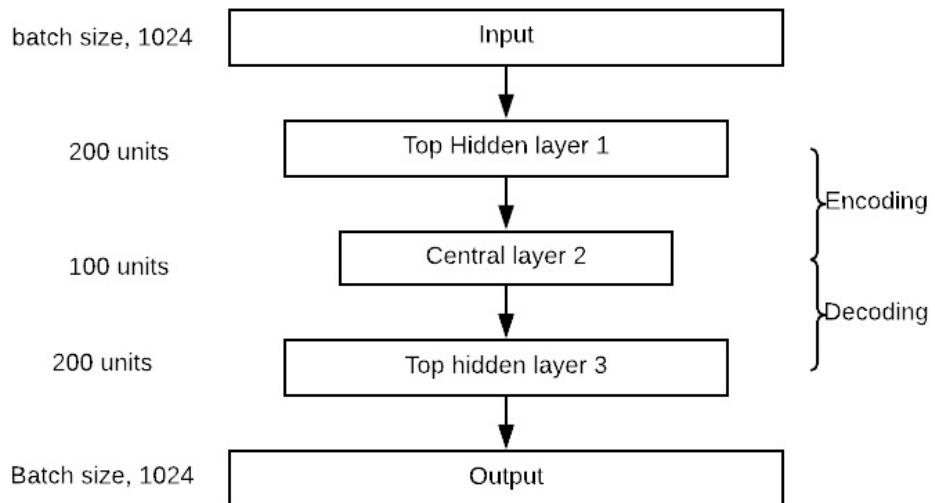


Build the network

It is time to construct the network. You will train a stacked autoencoder, that is, a network with multiple hidden layers.

Your network will have one input layer with 1024 points, i.e., 32x32, the shape of the image.

The encoder block will have one top hidden layer with 300 neurons, a central layer with 150 neurons. The decoder block is symmetric to the encoder. You can visualize the network in the picture below. Note that you can change the values of hidden and central layers.



Building an autoencoder is very similar to any other deep learning model.

You will construct the model following these steps:

1. Define the parameters
2. Define the layers
3. Define the architecture
4. Define the optimization
5. Run the model
6. Evaluate the model

In the previous section, you learned how to create a pipeline to feed the model, so there is no need to create once more the dataset. You will construct an autoencoder with four layers. You use the Xavier initialization. This is a technique to set the initial weights equal to the variance of both the input and output. Finally, you use the elu activation function. You regularize the loss function with L2 regularizer.

Step 1) Define the parameters

The first step implies to define the number of neurons in each layer, the learning rate and the hyperparameter of the regularizer.

Before that, you import the function partially. It is a better method to define the parameters of the dense layers. The code below defines the values of the autoencoder architecture. As listed before, the autoencoder has two layers, with 300 neurons in the first layers and 150 in the second layers. Their values are stored in n_hidden_1 and n_hidden_2.

You need to define the learning rate and the L2 hyperparameter. The values are stored in learning_rate and l2_reg

```
from functools import partial

## Encoder
n_hidden_1 = 300
n_hidden_2 = 150 # codings

## Decoder
n_hidden_3 = n_hidden_1
n_outputs = n_inputs

learning_rate = 0.01
l2_reg = 0.0001
```

The Xavier initialization technique is called with the object xavier_initializer from the estimator contrib. In the same estimator, you can add the regularizer with l2_regularizer

```
## Define the Xavier initialization
xav_init = tf.contrib.layers.xavier_initializer()
## Define the L2 regularizer
l2_regularizer = tf.contrib.layers.l2_regularizer(l2_reg)
```

Step 2) Define the layers

All the parameters of the dense layers have been set; you can pack everything in the variable dense_layer by using the object partial. dense_layer which uses the ELU activation, Xavier initialization, and L2 regularization.

```
## Create the dense layer
dense_layer = partial(tf.layers.dense,
```

```
activation=tf.nn.elu,  
kernel_initializer=xav_init,  
kernel_regularizer=l2_regularizer)
```

Step 3) Define the architecture

If you look at the picture of the architecture, you note that the network stacks three layers with an output layer. In the code below, you connect the appropriate layers. For instance, the first layer computes the dot product between the inputs matrice features and the matrices containing the 300 weights. After the dot product is computed, the output goes to the Elu activation function. The output becomes the input of the next layer, that is why you use it to compute hidden_2 and so on. The matrices multiplication are the same for each layer because you use the same activation function. Note that the last layer, outputs, does not apply an activation function. It makes sense because this is the reconstructed input

```
## Make the mat mul  
hidden_1 = dense_layer(features, n_hidden_1)  
hidden_2 = dense_layer(hidden_1, n_hidden_2)  
hidden_3 = dense_layer(hidden_2, n_hidden_3)  
outputs = dense_layer(hidden_3, n_outputs, activation=None)
```

Step 4) Define the optimization

The last step is to construct the optimizer. You use the Mean Square Error as a loss function. If you recall the tutorial on linear regression, you know that the MSE is computed with the difference between the predicted output and the real label. Here, the label is the feature because the model tries to reconstruct the input. Therefore, you want the mean of the sum of difference of the square between predicted output and input. With TensorFlow, you can code the loss function as follow:

```
loss = tf.reduce_mean(tf.square(outputs - features))
```

Then, you need to optimize the loss function. You use Adam optimizer to compute the gradients. The objective function is to minimize the loss.

```
## Optimize
loss = tf.reduce_mean(tf.square(outputs - features))
optimizer = tf.train.AdamOptimizer(learning_rate)
train = optimizer.minimize(loss)
```

One more setting before training the model. You want to use a batch size of 150, that is, feed the pipeline with 150 images each iteration. You need to compute the number of iterations manually. This is trivial to do:

If you want to pass 150 images each time and you know there are 5000 images in the dataset, the number of iterations is equal to . In python you can run the following codes and make sure the output is 33:

```
BATCH_SIZE = 150
### Number of batches : length dataset / batch size
n_batches = horse_x.shape[0] // BATCH_SIZE
print(n_batches)
33
```

Step 5) Run the model

Last but not least, train the model. You are training the model with 100 epochs. That is, the model will see 100 times the images to optimized weights.

You are already familiar with the codes to train a model in Tensorflow. The slight difference is to pipe the data before running the training. In this way, the model trains faster.

You are interested in printing the loss after ten epochs to see if the model is learning something (i.e., the loss is decreasing). The training takes 2 to 5 minutes, depending on your machine hardware.

```
## Set params
n_epochs = 100

## Call Saver to save the model and re-use it later during
evaluation
saver = tf.train.Saver()
```

```

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    # initialise iterator with train data
    sess.run(iterator.initializer, feed_dict={x: horse_x,
                                              batch_size: BATCH_SIZE})
    print('Training...')
    print(sess.run(features).shape)
    for epoch in range(n_epochs):
        for iteration in range(n_batches):
            sess.run(train)
        if epoch % 10 == 0:
            loss_train = loss.eval() # not shown
            print("\r{}{}".format(epoch), "Train MSE:", loss_train)
            saver.save(sess, "./my_model_all_layers.ckpt")
        save_path = saver.save(sess, "./model.ckpt")
        print("Model saved in path: %s" % save_path)
Training...
(150, 1024)
0 Train MSE: 2934.455
10 Train MSE: 1672.676
20 Train MSE: 1514.709
30 Train MSE: 1404.3118
40 Train MSE: 1425.058
50 Train MSE: 1479.0631
60 Train MSE: 1609.5259
70 Train MSE: 1482.3223
80 Train MSE: 1445.7035
90 Train MSE: 1453.8597
Model saved in path: ./model.ckpt

```

Step 6) Evaluate the model

Now that you have your model trained, it is time to evaluate it. You need to import the test set from the file /cifar-10-batches-py/.

```

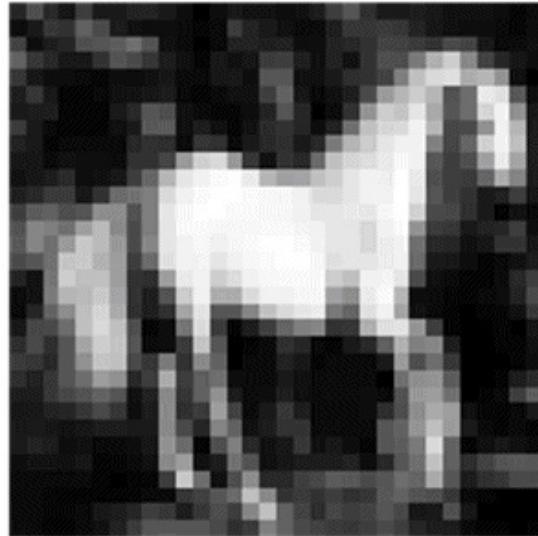
test_data = unpickle('./cifar-10-batches-py/test_batch')
test_x = grayscale(test_data['data'])
#test_labels = np.array(test_data['labels'])

```

NOTE: For a Windows machine, the code becomes test_data = unpickle(r"E:\cifar-10-batches-py\test_batch")

You can try to print the images 13, which is an horse

```
plot_image(test_x[13], shape=[32, 32], cmap = "Greys_r")
```



To evaluate the model, you will use the pixel value of this image and see if the encoder can reconstruct the same image after shrinking 1024 pixels. Note that, you define a function to evaluate the model on different pictures. The model should work better only on horses.

The function takes two arguments:

- df: Import the test data
- image_number: indicate what image to import

The function is divided into three parts:

1. Reshape the image to the correct dimension i.e 1, 1024
2. Feed the model with the unseen image, encode/decode the image
3. Print the real and reconstructed image

```
def reconstruct_image(df, image_number = 1):
    ## Part 1: Reshape the image to the correct dimension i.e 1,
    1024
```

```

x_test = df[image_number]
x_test_1 = x_test.reshape((1, 32*32))

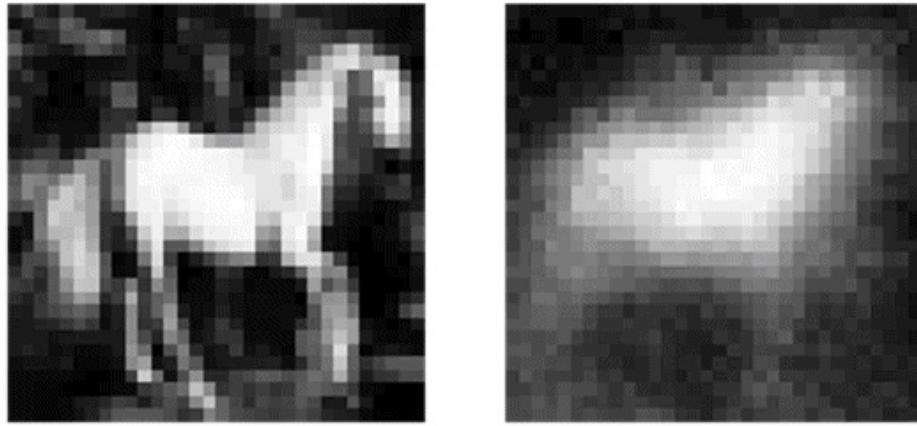
## Part 2: Feed the model with the unseen image, encode/decode
the image
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    sess.run(iterator.initializer, feed_dict={x: x_test_1,
                                              batch_size: 1})
## Part 3: Print the real and reconstructed image
# Restore variables from disk.
saver.restore(sess, "./model.ckpt")
print("Model restored.")
# Reconstruct image
outputs_val = outputs.eval()
print(outputs_val.shape)
fig = plt.figure()
# Plot real
ax1 = fig.add_subplot(121)
plot_image(x_test_1, shape=[32, 32], cmap = "Greys_r")
# Plot estimated
ax2 = fig.add_subplot(122)
plot_image(outputs_val, shape=[32, 32], cmap = "Greys_r")
plt.tight_layout()
fig = plt.gcf()

```

Now that the evaluation function is defined, you can have a look of the reconstructed image number thirteen

```
reconstruct_image(df =test_x, image_number = 13)
```

```
INFO:tensorflow:Restoring parameters from ./model.ckpt
Model restored.
(1, 1024)
```



Summary

The primary purpose of an autoencoder is to compress the input data, and then uncompress it into an output that looks closely like the original data.

The architecture of an autoencoder symmetrical with a pivot layer named the central layer.

You can create the autoencoder using:

- Partial: to create the dense layers with the typical setting:

```
• tf.layers.dense,  
    activation=tf.nn.elu,  
    kernel_initializer=xav_init,  
    kernel_regularizer=l2_regularizer
```

- dense_layer(): to make the matrix multiplication

you can define the loss function and optimization with:

```
loss = tf.reduce_mean(tf.square(outputs - features))  
optimizer = tf.train.AdamOptimizer(learning_rate)  
train = optimizer.minimize(loss)
```

Last run a session to train the model.

Chapter 20: RNN(Recurrent Neural Network) TensorFlow

What do we need an RNN?

The structure of an Artificial Neural Network is relatively simple and is mainly about matrix multiplication. During the first step, inputs are multiplied by initially random weights, and bias, transformed with an activation function and the output values are used to make a prediction. This step gives an idea of how far the network is from the reality.

The metric applied is the loss. The higher the loss function, the dumber the model is. To improve the knowledge of the network, some optimization is required by adjusting the weights of the net. The stochastic gradient descent is the method employed to change the values of the weights in the right direction. Once the adjustment is made, the network can use another batch of data to test its new knowledge.

The error, fortunately, is lower than before, yet not small enough. The optimization step is done iteratively until the error is minimized, i.e., no more information can be extracted.

The problem with this type of model is, it does not have any memory. It means the input and output are independent. In other words, the model does not care about what came before. It raises some question when you need to predict time series or sentences because the network needs to have information about the historical data or past words.

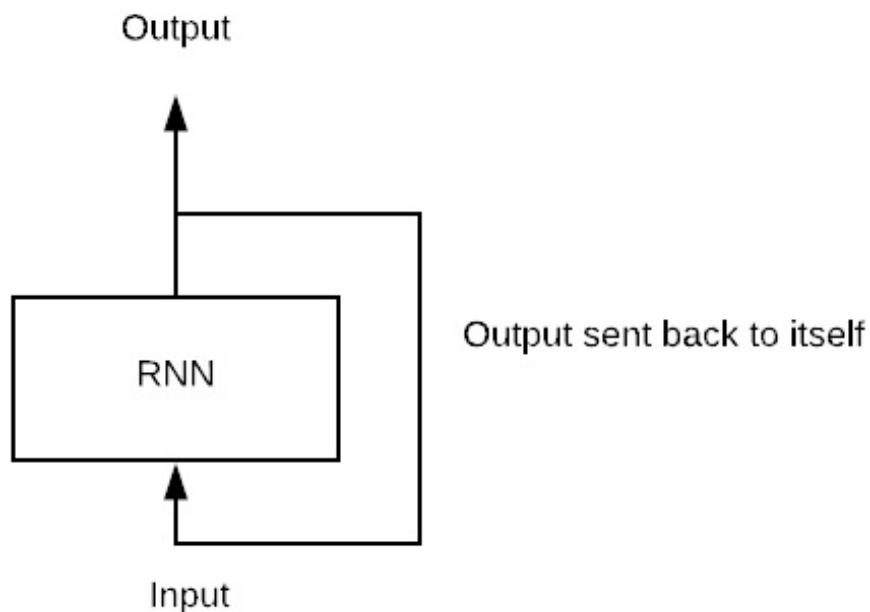
To overcome this issue, a new type of architecture has been developed: Recurrent Neural network (RNN hereafter)

What is RNN?

A recurrent neural network looks quite similar to a traditional neural network except that a memory-state is added to the neurons. The computation to include a memory is simple.

Imagine a simple model with only one neuron feeds by a batch of data. In a traditional neural net, the model produces the output by multiplying the input with the weight and the activation function. With an RNN, this output is sent back to itself number of time. We call **timestep** the amount of time the output becomes the input of the next matrice multiplication.

For instance, in the picture below, you can see the network is composed of one neuron. The network computes the matrices multiplication between the input and the weight and adds non-linearity with the activation function. It becomes the output at t-1. This output is the input of the second matrices multiplication.

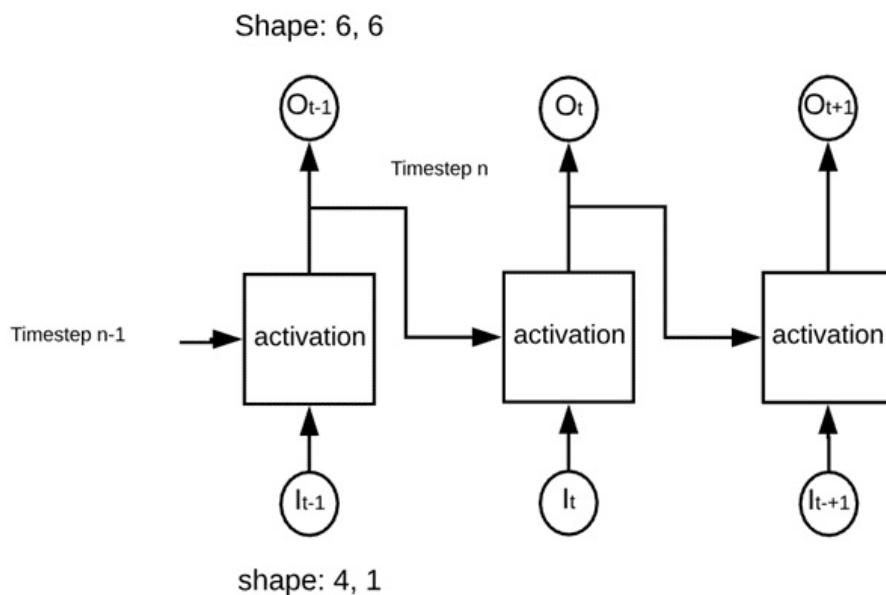


Below, we code a simple RNN in tensorflow to understand the step and also the shape of the output.

The network is composed of:

- Four inputs
- Six neurons
- 2-time steps

The network will proceed as depicted by the picture below.



The network is called 'recurrent' because it performs the same operation in each activate square. The network computed the weights of the inputs and the previous output before to use an activation function.

```
import numpy as np
import tensorflow as tf
n_inputs = 4
n_neurons = 6
n_timesteps = 2
The data is a sequence of a number from 0 to 9 and divided into
three batches of data.
## Data
X_batch = np.array([
    [[0, 1, 2, 5], [9, 8, 7, 4]], # Batch 1
    [[3, 4, 5, 2], [0, 0, 0, 0]], # Batch 2
])
```

```
    [[6, 7, 8, 5], [6, 5, 4, 2]], # Batch 3  
])
```

We can build the network with a placeholder for the data, the recurrent stage and the output.

1. Define the placeholder for the data

```
x = tf.placeholder(tf.float32, [None, n_timesteps, n_inputs])
```

Here:

- None: Unknown and will take the size of the batch
- n_timesteps: Number of time the network will send the output back to the neuron
- n_inputs: Number of input per batch

2. Define the recurrent network

As mentioned in the picture above, the network is composed of 6 neurons. The network will compute two dot product:

- Input data with the first set of weights (i.e., 6: equal to the number of neurons)
- Previous output with a second set of weights (i.e., 6: corresponding to the number of output)

Note that, during the first feedforward, the values of the previous output are equal to zeroes because we don't have any value available.

The object to build an RNN is `tf.contrib.rnn.BasicRNNCell` with the argument `num_units` to define the number of input

```
basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=n_neurons)
```

Now that the network is defined, you can compute the outputs and states

```
outputs, states = tf.nn.dynamic_rnn(basic_cell, X,  
dtype=tf.float32)
```

This object uses an internal loop to multiply the matrices the appropriate number of times.

Note that the recurrent neuron is a function of all the inputs of the previous time steps. This is how the network build its own memory. The information from the previous time can propagate in future time. This is the magic of Recurrent neural network

```
## Define the shape of the tensor  
X = tf.placeholder(tf.float32, [None, n_timesteps, n_inputs])  
## Define the network  
basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=n_neurons)  
outputs, states = tf.nn.dynamic_rnn(basic_cell, X,  
dtype=tf.float32)  
init = tf.global_variables_initializer()  
init = tf.global_variables_initializer()  
with tf.Session() as sess:  
    init.run()  
    outputs_val = outputs.eval(feed_dict={X: X_batch})  
print(states.eval(feed_dict={X: X_batch}))  
[[ 0.38941205 -0.9980438   0.99750966   0.7892596   0.9978241  
0.9999997 ]  
 [ 0.61096436   0.7255889   0.82977575  -0.88226104   0.29261455  
-0.15597084]  
 [ 0.62091285  -0.87023467   0.99729395  -0.58261937   0.9811445  
0.99969864]]
```

For explanatory purposes, you print the values of the previous state. The output printed above shows the output from the last state. Now print all the output, you can notice the states are the previous output of each batch. That is, the previous output contains the information about the entire sequence.e

```
print(outputs_val)  
print(outputs_val.shape)  
[[[-0.75934666 -0.99537754   0.9735819   -0.9722234   -0.14234993  
-0.9984044 ]  
 [ 0.99975264 -0.9983206   0.9999993   -1.           -0.9997506
```

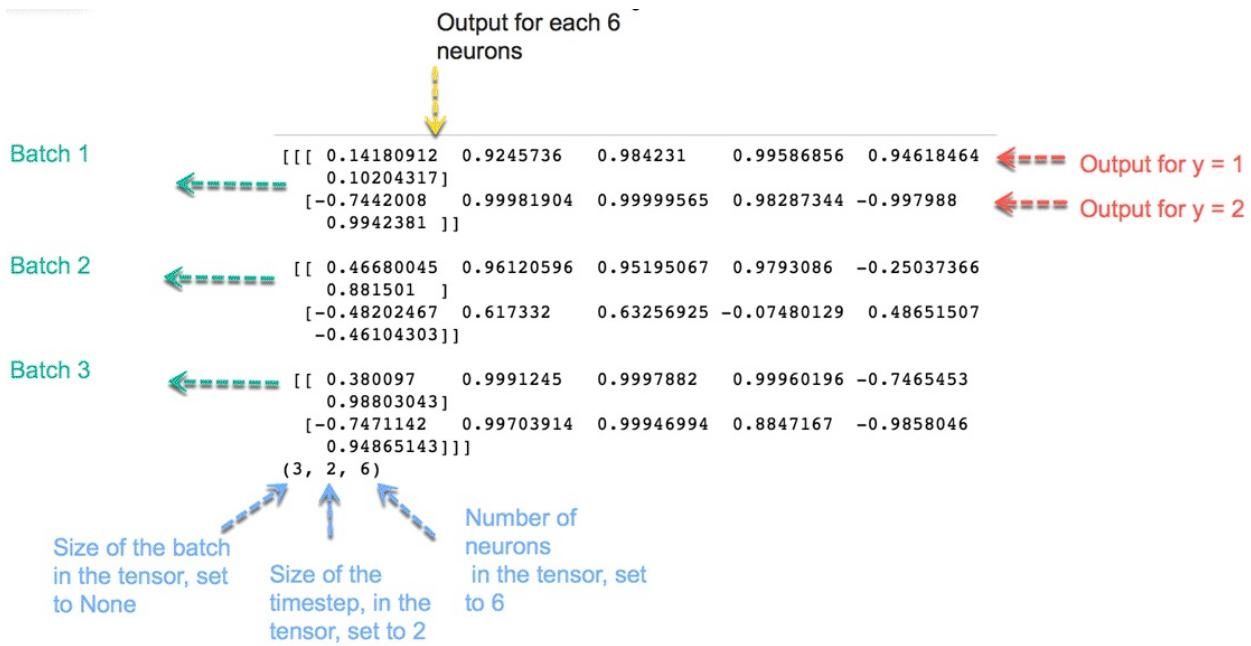
```

-1.      ]]

[[ 0.97486496 -0.98773265  0.9969686  -0.99950117 -0.7092863
-0.99998885]
[ 0.9326837   0.2673438   0.2808514  -0.7535883  -0.43337247
 0.5700631 ]]

[[ 0.99628735 -0.9998728   0.99999213 -0.99999976 -0.9884324
-1.
[ 0.99962527 -0.9467421   0.9997403   -0.99999714 -0.99929446
-0.9999795 ]]]
(3, 2, 6)

```



The output has the shape of (3, 2, 6):

- 3: Number of batches
- 2: Number of the timestep
- 6: Number of neurons

The optimization of a recurrent neural network is identical to a traditional neural network. You will see in more detail how to code optimization in the next part of this tutorial.

Applications of RNN

RNN has multiple uses, especially when it comes to predicting the future. In the financial industry, RNN can be helpful in predicting stock prices or the sign of the stock market direction (i.e., positive or negative).

RNN is useful for an autonomous car as it can avoid a car accident by anticipating the trajectory of the vehicle.

RNN is widely used in text analysis, image captioning, sentiment analysis and machine translation. For example, one can use a movie review to understand the feeling the spectator perceived after watching the movie. Automating this task is very useful when the movie company does not have enough time to review, label, consolidate and analyze the reviews. The machine can do the job with a higher level of accuracy.

Limitations of RNN

In theory, RNN is supposed to carry the information up to time . However, it is quite challenging to propagate all this information when the time step is too long. When a network has too many deep layers, it becomes untrainable. This problem is called: **vanishing gradient problem**. If you remember, the neural network updates the weight using the gradient descent algorithm. The gradients grow smaller when the network progress down to lower layers.

In conclusion, the gradients stay constant meaning there is no space for improvement. The model learns from a change in the gradient; this change affects the network's output. However, if the difference in the gradient is too small (i.e., the weights change a little), the network can't learn anything and so the output. Therefore, a network facing a vanishing gradient problem cannot converge toward a good solution.

Improvement LSMT

To overcome the potential issue of vanishing gradient faced by RNN, three researchers, Hochreiter, Schmidhuber and Bengio improved the RNN with an architecture called Long Short-Term Memory (LSTM). In brief, LSTM provides to the network relevant past information to more recent time. The machine uses a better architecture to select and carry information back to later time.

LSTM architecture is available in TensorFlow, `tf.contrib.rnn.LSTMCell`. LSTM is out of the scope of the tutorial. You can refer to the official documentation for further information

RNN in time series

In this tutorial, you will use an RNN with time series data. Time series are dependent to previous time which means past values includes relevant information that the network can learn from. The idea behind time series prediction is to estimate the future value of a series, let's say, stock price, temperature, GDP and so on.

The data preparation for RNN and time series can be a little bit tricky. First of all, the objective is to predict the next value of the series, meaning, you will use the past information to estimate the value at $t + 1$. The label is equal to the input sequence and shifted one period ahead. Secondly, the number of input is set to 1, i.e., one observation per time. Lastly, the time step is equal to the sequence of the numerical value. For instance, if you set the time step to 10, the input sequence will return ten consecutive times.

Look at the graph below, we have represented the time series data on the left and a fictive input sequence on the right. You create a function to return a dataset with random value for each day from January 2001 to December 2016

```
# To plot pretty figures
```

```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
def create_ts(start = '2001', n = 201, freq = 'M'):
    rng = pd.date_range(start=start, periods=n, freq=freq)
    ts = pd.Series(np.random.uniform(-18, 18, size=len(rng)),
rng).cumsum()
    return ts
ts= create_ts(start = '2001', n = 192, freq = 'M')
ts.tail(5)
```

Output

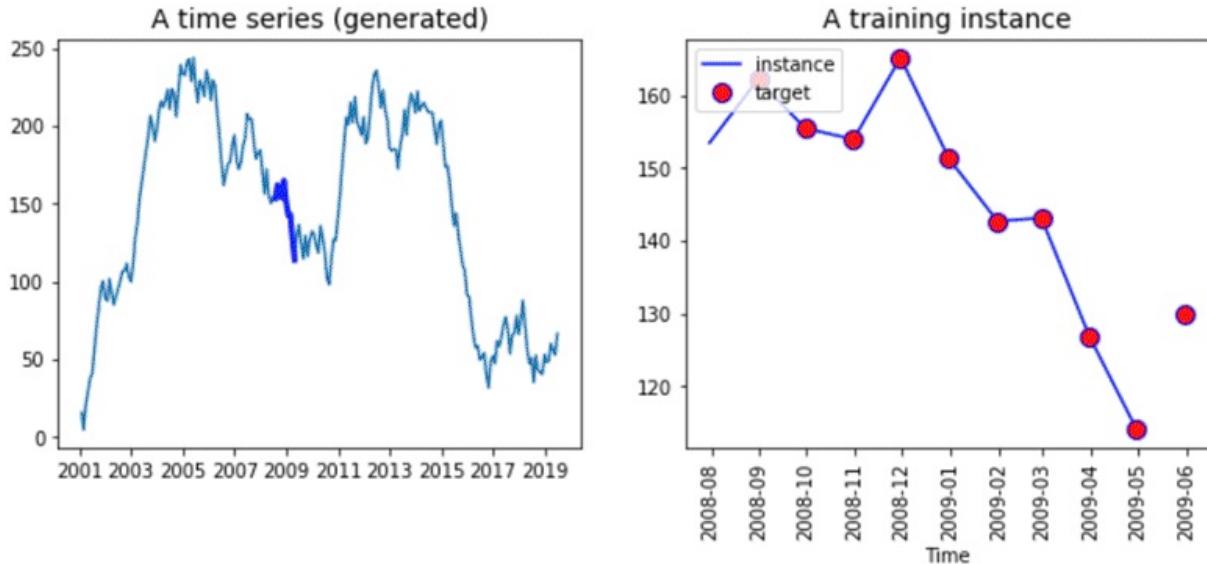
```
2016-08-31      -93.459631
2016-09-30      -95.264791
2016-10-31      -95.551935
2016-11-30      -105.879611
2016-12-31      -123.729319
Freq: M, dtype: float64
```

```
ts = create_ts(start = '2001', n = 222)

# Left
plt.figure(figsize=(11,4))
plt.subplot(121)
plt.plot(ts.index, ts)
plt.plot(ts.index[90:100], ts[90:100], "b-", linewidth=3, label="A
training instance")
plt.title("A time series (generated)", fontsize=14)

# Right
plt.subplot(122)
plt.title("A training instance", fontsize=14)
plt.plot(ts.index[90:100], ts[90:100], "b-", markersize=8,
label="instance")
plt.plot(ts.index[91:101], ts[91:101], "bo", markersize=10,
label="target", markerfacecolor='red')
plt.legend(loc="upper left")
plt.xlabel("Time")

plt.show()
```



The right part of the graph shows all series. It starts from 2001 and finishes in 2019. It makes no sense to feed all the data in the network, instead, you need to create a batch of data with a length equal to the time step. This batch will be the X variable. The Y variable is the same as X but shifted by one period (i.e., you want to forecast $t+1$).

Both vectors have the same length. You can see it in the right part of the above graph. The line represents the ten values of the X input, while the red dots are the ten values of the label, Y. Note that, the label starts one period ahead of X and finishes one period after.

Build an RNN to predict Time Series in TensorFlow

Now, it is time to build your first RNN to predict the series above. You need to specify some hyperparameters (the parameters of the model, i.e., number of neurons, etc.) for the model:

- Number of input: 1
- Time step (windows in time series): 10
- Number of neurons: 120
- Number of output: 1

Your network will learn from a sequence of 10 days and contain 120 recurrent neurons. You feed the model with one input, i.e., one day. Feel free to change the values to see if the model improved.

Before to construct the model, you need to split the dataset into a train set and test set. The full dataset has 222 data points; you will use the first 201 point to train the model and the last 21 points to test your model.

After you define a train and test set, you need to create an object containing the batches. In this batches, you have X values and Y values. Remember that the X values are one period lagged. Therefore, you use the first 200 observations and the time step is equal to 10. The X_batches object should contain 20 batches of size 10*1. The y_batches has the same shape as the X_batches object but with one period ahead.

Step 1) Create the train and test

First of all, you convert the series into a numpy array; then you define the windows (i.e., the number of time the network will learn from), the number of input, output and the size of the train set.

```
series = np.array(ts)
n_windows = 20
n_input = 1
n_output = 1
size_train = 201
```

After that, you simply split the array into two datasets.

```
## Split data
train = series[:size_train]
test = series[size_train:]
print(train.shape, test.shape)
(201,) (21,)
```

Step 2) Create the function to return X_batches and y_batches

To make it easier, you can create a function that returns two different arrays, one for X_batches and one for y_batches.

Let's write a function to construct the batches.

Note that, the X batches are lagged by one period (we take value t-1). The output of the function should have three dimensions. The first dimensions equal the number of batches, the second the size of the windows and last one the number of input.

The tricky part is to select the data points correctly. For the X data points, you choose the observations from $t = 1$ to $t = 200$, while for the Y data point, you return the observations from $t = 2$ to 201. Once you have the correct data points, it is straightforward to reshape the series.

To construct the object with the batches, you need to split the dataset into ten batches of equal length (i.e., 20). You can use the reshape method and pass -1 so that the series is similar to the batch size. The value 20 is the number of observations per batch and 1 is the number of input.

You need to do the same step but for the label.

Note that, you need to shift the data to the number of time you want to forecast. For instance, if you want to predict one timeahead, then you shift the series by 1. If you want to forecast two days, then shift the data by 2.

```
x_data = train[:size_train-1]: Select all the training instance  
minus one day  
X_batches = x_data.reshape(-1, windows, input): create the right  
shape for the batch e.g (10, 20, 1)  
def create_batches(df, windows, input, output):  
    ## Create X  
    x_data = train[:size_train-1] # Select the data  
    X_batches = x_data.reshape(-1, windows, input) # Reshape  
the data  
    ## Create y  
    y_data = train[n_output:size_train]  
    y_batches = y_data.reshape(-1, windows, output)  
    return X_batches, y_batches
```

Now that the function is defined, you can call it to create the batches.

```
X_batches, y_batches = create_batches(df = train,  
                                      windows = n_windows,  
                                      input = n_input,  
                                      output = n_output)
```

You can print the shape to make sure the dimensions are correct.

```
print(X_batches.shape, y_batches.shape)  
(10, 20, 1) (10, 20, 1)
```

You need to create the test set with only one batch of data and 20 observations.

Note that, you forecast days after days, it means the second predicted value will be based on the true value of the first day ($t+1$) of the test dataset. In fact, the true value will be known.

If you want to forecast $t+2$ (i.e., two days ahead), you need to use the predicted value $t+1$; if you're going to predict $t+3$ (three days ahead), you need to use the predicted value $t+1$ and $t+2$. It makes sense that, it is difficult to predict

accurately t+n days ahead.

```
X_test, y_test = create_batches(df = test, windows = 20, input = 1,  
output = 1)  
print(X_test.shape, y_test.shape)  
(10, 20, 1) (10, 20, 1)
```

Alright, your batch size is ready, you can build the RNN architecture. Remember, you have 120 recurrent neurons.

Step 3) Build the model

To create the model, you need to define three parts:

1. The variable with the tensors
2. The RNN
3. The loss and optimization

Step 3.1) Variables

You need to specify the X and y variables with the appropriate shape. This step is trivial. The tensor has the same dimension as the objects X_batches and y_batches.

For instance, the tensor X is a placeholder (Check the tutorial on Introduction to Tensorflow to refresh your mind about variable declaration) has three dimensions:

- Note: size of the batch
- n_windows: Length of the windows. i.e., the number of time the model looks backward
- n_input: Number of input

The result is:

```
tf.placeholder(tf.float32, [None, n_windows, n_input])
```

```
## 1. Construct the tensors
X = tf.placeholder(tf.float32, [None, n_windows, n_input])
y = tf.placeholder(tf.float32, [None, n_windows, n_output])
```

Step 3.2) Create the RNN

In the second part, you need to define the architecture of the network. As before, you use the object BasicRNNCell and dynamic_rnn from TensorFlow estimator.

```
## 2. create the model
basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=r_neuron,
activation=tf.nn.relu)
rnn_output, states = tf.nn.dynamic_rnn(basic_cell, X,
dtype=tf.float32)
```

The next part is a bit trickier but allows faster computation. You need to transform the run output to a dense layer and then convert it again to have the same dimension as the input.

```
stacked_rnn_output = tf.reshape(rnn_output, [-1, r_neuron])
stacked_outputs = tf.layers.dense(stacked_rnn_output, n_output)
outputs = tf.reshape(stacked_outputs, [-1, n_windows, n_output])
```

Step 3.3) Create the loss and optimization

The model optimization depends of the task you are performing. In the previous tutorial on CNN, your objective was to classify images, in this tutorial, the objective is slightly different. You are asked to make a prediction on a continuous variable compare to a class.

This difference is important because it will change the optimization problem. The optimization problem for a continuous variable is to minimize the mean square error. To construct these metrics in TF, you can use:

- `tf.reduce_sum(tf.square(outputs - y))`

The remaining of the code is the same as before; you use an Adam optimizer to

reduce the loss (i.e., MSE):

- `tf.train.AdamOptimizer(learning_rate=learning_rate)`
- `optimizer.minimize(loss)`

That's it, you can pack everything together, and your model is ready to train.

```
tf.reset_default_graph()
r_neuron = 120

## 1. Construct the tensors
x = tf.placeholder(tf.float32, [None, n_windows, n_input])
y = tf.placeholder(tf.float32, [None, n_windows, n_output])

## 2. create the model
basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=r_neuron,
activation=tf.nn.relu)
rnn_output, states = tf.nn.dynamic_rnn(basic_cell, x,
dtype=tf.float32)

stacked_rnn_output = tf.reshape(rnn_output, [-1, r_neuron])
stacked_outputs = tf.layers.dense(stacked_rnn_output, n_output)
outputs = tf.reshape(stacked_outputs, [-1, n_windows, n_output])

## 3. Loss + optimization
learning_rate = 0.001

loss = tf.reduce_sum(tf.square(outputs - y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

init = tf.global_variables_initializer()
```

You will train the model using 1500 epochs and print the loss every 150 iterations. Once the model is trained, you evaluate the model on the test set and create an object containing the predictions.

```
iteration = 1500

with tf.Session() as sess:
    init.run()
    for iters in range(iteration):
```

```

        sess.run(training_op, feed_dict={X: X_batches, y:
y_batches})
        if iters % 150 == 0:
            mse = loss.eval(feed_dict={X: X_batches, y: y_batches})
            print(iters, "\tMSE:", mse)

    y_pred = sess.run(outputs, feed_dict={X: X_test})
0      MSE: 502893.34
150     MSE: 13839.129
300     MSE: 3964.835
450     MSE: 2619.885
600     MSE: 2418.772
750     MSE: 2110.5923
900     MSE: 1887.9644
1050    MSE: 1747.1377
1200    MSE: 1556.3398
1350    MSE: 1384.6113

```

At last, you can plot the actual value of the series with the predicted value. If your model is corrected, the predicted values should be put on top of the actual values.

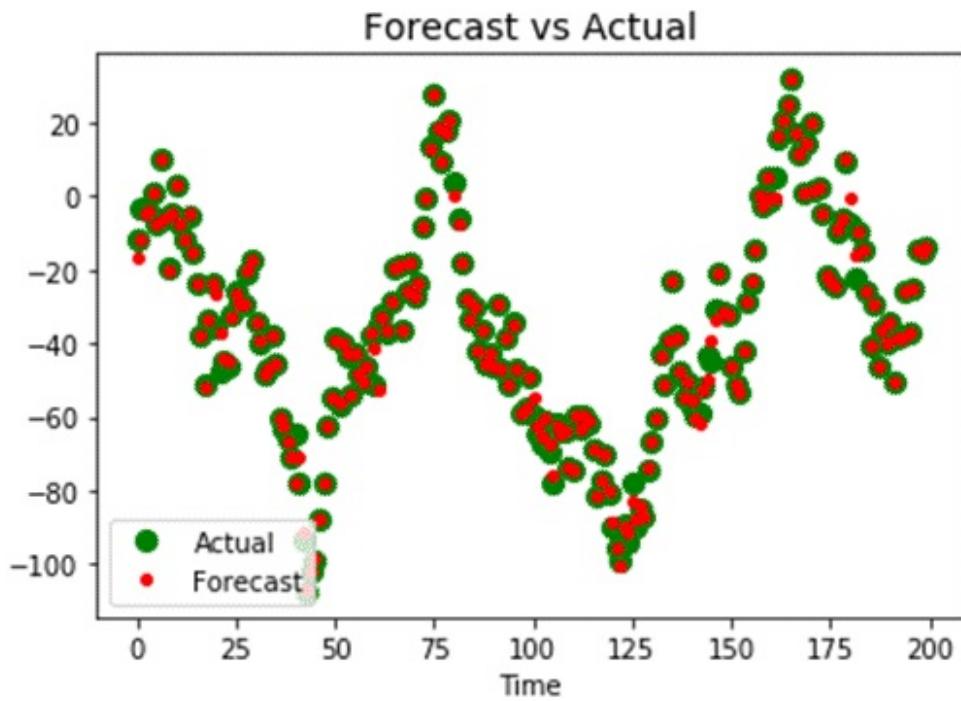
As you can see, the model has room of improvement. It is up to you to change the hyperparameters like the windows, the batch size of the number of recurrent neurons.

```

plt.title("Forecast vs Actual", fontsize=14)
plt.plot(pd.Series(np.ravel(y_test)), "bo", markersize=8,
label="Actual", color='green')
plt.plot(pd.Series(np.ravel(y_pred)), "r.", markersize=8,
label="Forecast", color='red')
plt.legend(loc="lower left")
plt.xlabel("Time")

plt.show()

```



Summary

A recurrent neural network is a robust architecture to deal with time series or text analysis. The output of the previous state is feedback to preserve the memory of the network over time or sequence of words.

In TensorFlow, you can use the following codes to train a recurrent neural network for time series:

Parameters of the model

```
n_windows = 20
n_input = 1
n_output = 1
size_train = 201
```

Define the model

```
x = tf.placeholder(tf.float32, [None, n_windows, n_input])
y = tf.placeholder(tf.float32, [None, n_windows, n_output])
```

```
basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=r_neuron,
activation=tf.nn.relu)
rnn_output, states = tf.nn.dynamic_rnn(basic_cell, x,
dtype=tf.float32)

stacked_rnn_output = tf.reshape(rnn_output, [-1, r_neuron])
stacked_outputs = tf.layers.dense(stacked_rnn_output, n_output)
outputs = tf.reshape(stacked_outputs, [-1, n_windows, n_output])
```

Construct the optimization

```
learning_rate = 0.001

loss = tf.reduce_sum(tf.square(outputs - y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)
```

Train the model

```
init = tf.global_variables_initializer()
iteration = 1500

with tf.Session() as sess:
    init.run()
    for iters in range(iteration):
        sess.run(training_op, feed_dict={X: X_batches, y:
y_batches})
        if iters % 150 == 0:
            mse = loss.eval(feed_dict={X: X_batches, y: y_batches})
            print(iters, "\tMSE:", mse)

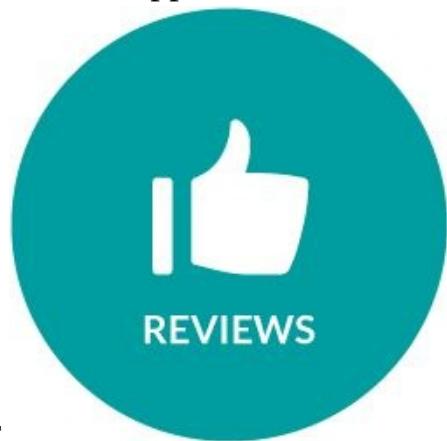
    y_pred = sess.run(outputs, feed_dict={X: X_test})
```

One Last Thing....

DID YOU ENJOY THE BOOK?

IF SO, THEN LET ME KNOW LEAVING A REVIEW ON AMAZON!

Reviews are lifeblood of independent authors. I would appreciate even a few



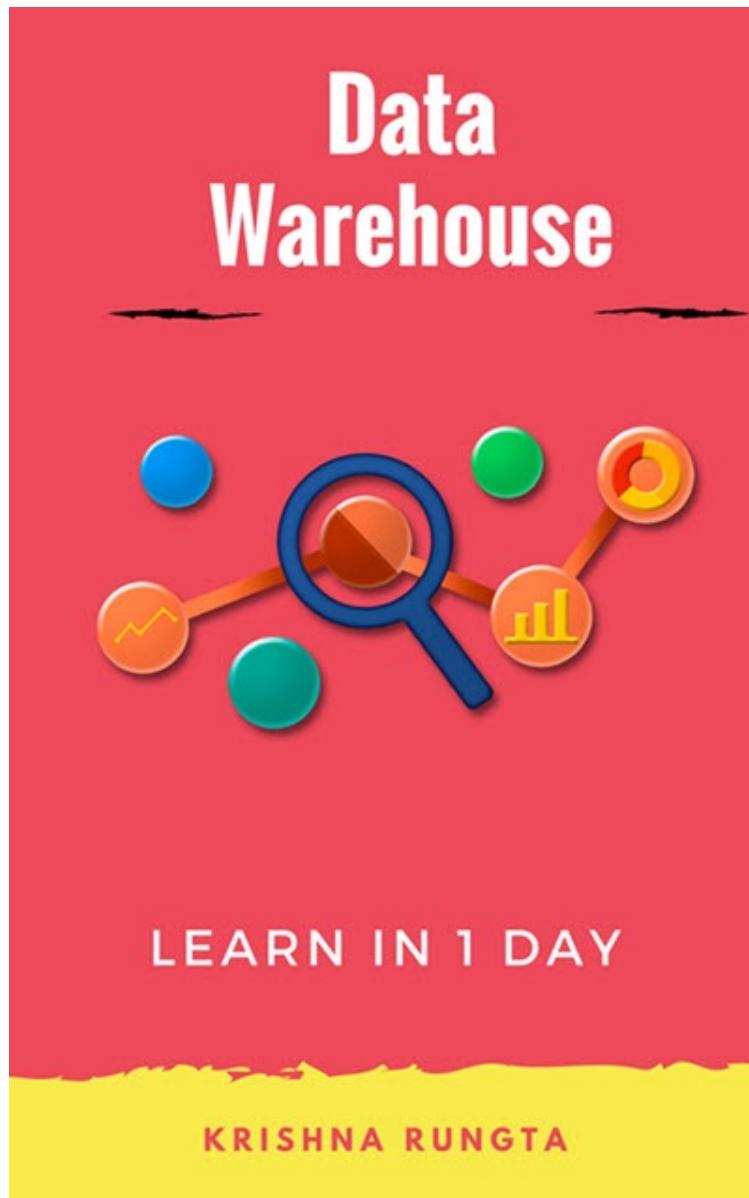
words and rating if that's all you have time for

IF YOU DID NOT LIKE THIS BOOK, THEN PLEASE TELL ME! [EMAIL me](#)

and Let me know what you didn't like! Perphase I can change it. In today's world a book doesn't have to be stagnant, it can improve with time and feedback from readers like you. You can impact this book, and I welcome your feedback. Help make this book better for everyone!

More Books -

Learn Data Warehousing in 1 Day: Complete ETL guide for beginners



[BUY NOW](#)

Learn R Programming in 1 Day: Complete Guide for Beginners

R

Programming



LEARN IN 1 DAY

KRISHNA RUNGTA

[BUY NOW](#)