TensorFlow and Keras Journal

Aritz Bercher

July 30, 2018

Contents

Abstract

In this journal, I will try to keep track of the useful references to learn TensorFlow and Keras (mainly webpages) I find online and of my little progresses.

1 Useful Resources

- There is a MOOK online about Tensor Flow: `https://de.udacity.com/course/deep-learning--ud730`
- This code seems to contain a lot of TensorFlow APIs without being too long: `https://github.com/tensorflow/tensorflow/blob/r1.7/tensorflow/examples/tutorials/mnist/mnist_with_summaries.py`
- This tutorial is a good place to get started: `https://www.tensorflow.org/get_started/`
- More advanced tutorials for specific tasks can be found there `https://www.tensorflow.org/tutorials/`
- Here are a lot of different models: `https://github.com/tensorflow/models`

- This tutorial about Pytorch contains a very small basic example of a TensorFlow program which I find very instructive:
  `http://pytorch.org/tutorials/beginner/pytorch_with_examples.html#pytorch-variables-a`
- Roman recommended this coursera mook on NLP using TensorFlow:
  `https://www.coursera.org/learn/language-processing`
- This page contains information about how to **read data without using the `tf.data` API**.
  `https://www.tensorflow.org/api_guides/python/reading_data`
- This page contains a very nice tutorial about the basic APIs from TF and explains the general architecture of TF:
  `http://www.goldsborough.me/tensorflow/ml/ai/python/2017/06/28/20-21-45-a_sweeping_tour_of_tensorflow/`
- This page is the **official documentation for Keras**:
  `https://keras.io/`

1.1 Advanced projects

In this section, I present complicated models that could be interesting to study:

- Here is a project that Nikos recommended to me which uses TF:
  BiDAF
- This script coming from the official repo implements with TF a CNN model for Sentiment Analysis. It uses some recent architecture where the size of some layers vary to make up for the fact that not all input have the same length:
  `https://github.com/tensorflow/models/tree/master/research/sentiment_analysis`

2 Journal

03.04.18 ~ Bi-Directional Attention Flow for Machine Comprehension (BiDAF), Comparison Pytorch and TensorFlow

I looked at this webpage that Nikos recommended me:
`https://allenai.github.io/bi-att-flow/`
It looks like they developed a special architecture for a neural network which goal is to find answers in a text to questions asked in natural language.
I play a bit with their demo version:
`http://allgood.cs.washington.edu:1995/`
but it's easy to fool.

I read this article which gives comparison of pytorch and tensorflow:
`https://towardsdatascience.com/pytorch-vs-tensorflow-spotting-the-difference-25c75777377`

04.04.18 ~ Installing pip3 and tensorflow-gpu

In order to install tensorflow (I read somewhere that it was better to use pip than conda for installing tensorflow), I installed `pip3` with this command:

```
sudo apt install python3-pip
```

I then installed TensorFlow following the recommendations found with the following command:

```
pip3 install --upgrade tensorflow-gpu
```

It went well but at the end I received this warning:

```
You are using pip version 8.1.1, however version 9.0.3 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

I don't understand exactly how it is possible since I just installed `pip3`. I suspect that when one uses sudo apt-get, the computer looks inside some given list of packages put online by the ubuntu community, and that these packages are not always the latest available version. For now I won't upgrade pip as it doesn't seem necessary.
Then I tried to run the example but it didn't work, I received this error:

```
ModuleNotFoundError: No module named 'matplotlib'
```

I installed it using

```
conda install matplotlib
```

and the error disappeared but was replaced by

```
ModuleNotFoundError: No module named 'tensorflow'
```

I don't know the reason. If, after having activated my conda environment mypy36

```
conda list
```

tensorflow doesn't appear, and if I deactivate mypy36, it also doesn't appear. Nikos suggested that it might be related to the fact that I use python 3.6 and not 3.5. There is also an issue with `pip` itself. It might be the case that when using it it doesn't work well with the conda environment. More precisely, it could happen that pip installs the packages in the root environment instead of the activated environment, and it also might

be the case that the packages installed with pip do not appear when using conda list. The solution (suggested by Nikos and hinted by this post) would be to install pip with anaconda, and directly at the creation of the environment.
I think I will create a new environment special for tensorflow with python 3.5 (as suggested in this post).
According to this page:

`https://stackoverflow.com/questions/18640305/how-do-i-keep-track-of-pip-installed-package`

in order to see all the packages installed by both conda and pip, one can use this command (for an environment called mypy36):

```
conda env export -n mypy36
```

I tried again to install tensorflow but this time following this guideline:
`https://github.com/williamFalcon/tensorflow-gpu-install-ubuntu-16.04`
I tried to install cuda with

```
sudo apt-get install cuda-9-0
```

but for some reason my internet connection failed in the middle of the download.
Following the recommendations appearing in the terminal, I entered

```
sudo apt-get install --fix-missing
```

and received the following output

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  gstreamer1.0-fluendo-mp3 gstreamer1.0-libav libenca0 libguess1 librubberband2v5 libsdl2
  linux-headers-4.13.0-32 linux-headers-4.13.0-32-generic linux-headers-4.13.0-36 linux-h
  linux-image-4.13.0-36-generic linux-image-extra-4.10.0-28-generic linux-image-extra-4.1
  ubuntu-restricted-addons
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 7 not upgraded.
```

Then I tried to do

```
sudo apt purge cuda-9-0
```

and received this output:

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
Package 'cuda-9-0' is not installed, so not removed
The following packages were automatically installed and are no longer required:
  gstreamer1.0-fluendo-mp3 gstreamer1.0-libav libenca0 libguess1 librubberband2v5 libsdl2
  linux-headers-4.13.0-32 linux-headers-4.13.0-32-generic linux-headers-4.13.0-36 linux-h
  linux-image-4.13.0-36-generic linux-image-extra-4.10.0-28-generic linux-image-extra-4.1
  ubuntu-restricted-addons
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 7 not upgraded.
```

Then I entered again

```
sudo apt-get install cuda-9-0
```

It seemed to have worked. A lot of messages appeared on the screen including this one:

```
A modprobe blacklist file has been created at /etc/modprobe.d to prevent Nouveau from loa
A new initrd image has also been created. To revert, please replace /boot/initrd-4.13.0-3

*******************************************************************************
*** Reboot your computer and verify that the NVIDIA graphics driver can   ***
*** be loaded.                                                             ***
*******************************************************************************
```

I deviated slightly from the tutorial when I created the conda environment by using this:

```
conda create -n tensorflow python=3.5 pip=9.0.1
```

instead of

```
conda create -n tensorflow
```

After entering
I received the following messages:

```
Requirement already satisfied: tensorflow-gpu in ./.local/lib/python3.5/site-packages
Requirement already satisfied: protobuf>=3.4.0 in ./.local/lib/python3.5/site-packages (i
Requirement already satisfied: astor>=0.6.0 in ./.local/lib/python3.5/site-packages (fron
Requirement already satisfied: grpcio>=1.8.6 in ./.local/lib/python3.5/site-packages (fro
Requirement already satisfied: absl-py>=0.1.6 in ./.local/lib/python3.5/site-packages (fr
Requirement already satisfied: tensorboard<1.8.0,>=1.7.0 in ./.local/lib/python3.5/site-p
Requirement already satisfied: gast>=0.2.0 in ./.local/lib/python3.5/site-packages (from
Requirement already satisfied: numpy>=1.13.3 in ./.local/lib/python3.5/site-packages (fro
Requirement already satisfied: termcolor>=1.1.0 in ./.local/lib/python3.5/site-packages
```

```
Requirement already satisfied: six>=1.10.0 in ./.local/lib/python3.5/site-packages (from
Requirement already satisfied: wheel>=0.26 in ./.local/lib/python3.5/site-packages (from
Requirement already satisfied: setuptools in ./.local/lib/python3.5/site-packages (from p
Requirement already satisfied: werkzeug>=0.11.10 in ./.local/lib/python3.5/site-packages
Requirement already satisfied: html5lib==0.9999999 in ./.local/lib/python3.5/site-package
Requirement already satisfied: bleach==1.5.0 in ./.local/lib/python3.5/site-packages (fro
Requirement already satisfied: markdown>=2.6.8 in ./.local/lib/python3.5/site-packages (i
```

which let me think that I successfully installed tensorflow on my first trial but that for some reason it wasn't accessible when I tried to run the script from the tensorflow tutorial.
When I ran

```
sess = tf.Session()
```

I got this output:

```
2018-04-04 17:32:06.689952: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU
2018-04-04 17:32:06.832301: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:898] s
2018-04-04 17:32:06.832758: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1344] Foun
name: GeForce 940MX major: 5 minor: 0 memoryClockRate(GHz): 1.2415
pciBusID: 0000:02:00.0
totalMemory: 1.96GiB freeMemory: 1.59GiB
2018-04-04 17:32:06.832775: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1423] Addi
2018-04-04 17:34:23.194390: I tensorflow/core/common_runtime/gpu/gpu_device.cc:911] Devic
2018-04-04 17:34:23.194416: I tensorflow/core/common_runtime/gpu/gpu_device.cc:917]
2018-04-04 17:34:23.194422: I tensorflow/core/common_runtime/gpu/gpu_device.cc:930] 0:
2018-04-04 17:34:23.196587: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1041] Crea
```

which I think is good (the guideline says "when you run sess, you should see a bunch of lines with the word gpu in them (if install worked) otherwise, not running on gpu").

05.04.18 ∼ Finishing the installation of tensorflow, some basic TF APIs, Introduction to Keras

Something strange happened: even if I can import tensorflow in the console while my environment "tensorflow" is activated, I still get the error

```
No module named 'tensorflow'
```

when I try to use it in a jupyter notebook. Suspecting that the reason might be that jupyter is not installed inside the environment "tensorflow" but only on the root, I installed juypter with

```
conda install jupyter
```

but the error remained even after restarting jupyter...
Then I tried to install tensorflow from source (following the indication of this page and not of the guideline that I followed yesterday) using this command

```
pip install --ignore-installed --upgrade https://storage.googleapis.com/tensorflow/linux/
```

but still I was getting the same error.
I deactivated and reactivated the tensorflow environment and then tried to execute the cells of my notebook but this time even matplotlib seems not to be installed:

```
ImportError: No module named 'matplotlib'
```

which actually makes sense since I haven't reinstalled it (with conda) after creating my new environment. I installed it (inside my tensorflow environment) and after rebooting my system it finally works.
I read the beginning of this page:
`https://www.tensorflow.org/programmers_guide/eager` which explains what `eager` is and how to use it. "TensorFlow's eager execution is an imperative programming environment that evaluates operations immediately, without an extra graph-building step. Operations return concrete values instead of constructing a computational graph to run later. This makes it easy to get started with TensorFlow, debug models, reduce boilerplate code, and is fun!".
But I don't want to spend to much time on this page.
I read the beginning of this page about **Datasets API**: `https://www.tensorflow.org/programmers_guide/datasets` but not the diverse example. What I understand from it is that it is a classes which help building kind of pipelines to preprocess the input.
I read this basic tutorial about **Keras**:
`https://keras.io/getting-started/sequential-model-guide/`

06.04.18 ~ Getting started with tensorflow
I began this tutorial: `https://www.tensorflow.org/get_started/eager` I went until the training part.

09.04.18 ~ Trying to print the value of the tensors used in the program, etimators, input functions
I tried to print the value of the tensors appearing in the tutorial mentioned in the previous entry of this journal. It is more tricky than I thought due to the architecture of TF:
Stackoverflow: How to print the value of a tensor object in tensorflow?
I will wait to know more about it to try to do it.
In fact, when one uses eager execution mode, one can use

```
tfe.Iterator(train_dataset).next()
```

I tried to understand how Automatic differentiation works, in particular how to use this `tfe.GradientTape` but it didn't get it. I will come back to it later.
I finished reading this tutorial:
`https://www.tensorflow.org/get_started/eager`
I didn't understand everything but I will try to study other tutorials instead of focusing on this one because some aspect might be specific to this eager tool (I think I asked Nikos about this tape and he told me he didn't know it so I assume it's not central).
I started this tutorial which emphasizes the notion of **estimator** (an object which encapsulates the model as well as methods to train the model, to judge the model's accuracy, and to generate predictions):
`https://www.tensorflow.org/get_started/premade_estimators`
"An Estimator is any class derived from `tf.estimator.Estimator`". I learned about **Input functions**. I think it is the function defining where the data comes from. It is used when the model is trained or used to do predictions. It outputs an object of class `tf.data.Dataset`.
Here is an example taken from custom_estimator.py :

```
classifier.train(
    input_fn=lambda:iris_data.train_input_fn(train_x, train_y, args.batch_size),
    steps=args.train_steps)
```

It also introduces the class `Dataset` (see here) and some of its subclasses like `TextLineDataset` (see here) which have already appeared in this journal (see entry of the 05.04.18).
There is a file called premade_estimator.py which contains most of the code of the tutorial. It is **a good example of a simple script using tensorflow's APIs**

10.04.18 ~ Premade estimators tutorial
I kept reading this tutorial:
`https://www.tensorflow.org/get_started/premade_estimators`
I learned a bit about **feature column** and the `tf.feature_column` API. It seems to be an attribute of models object given to the models when they are initialized. It describes the type of each input. I also learned about some **premade estimator classes** like `tf.estimator.DNNClassifier`.
I learned how to train one of these premade estimators, how evaluate its accuracy, and how to use it for predictions.
I finished reading this tutorial.

11.04.18 ~ Checkpoint format, feature columns
I read this tutorial about **Checkpoint format**:
`https://www.tensorflow.org/get_started/checkpoints`
From what I understand this is just describing how the values of the weights etc... are stored in external files. One should be careful when building a copy of an existing model and changing it as it might use the checkpoint files of the first estimator and create conflict.

Then I started this tutorial:
`https://www.tensorflow.org/get_started/feature_columns`
which explains more about **feature columns**, what they represent and how to build them using the `tf.feature_column` API. My understanding is that it encapsulates inside the model information about the data provided by the input function. This is required when using `tf.estimator.Estimator` objects. It says if an input is supposed to be used as a numerical value, or if it has to be categorized or bucketized, etc... There is a list of the possibilities on this page
`https://www.tensorflow.org/api_docs/python/tf/feature_column/crossed_column`
In this tutorial they present

- Numeric columns
- Bucketized columns
- Categorical identity columns
- Categorical vocabulary columns
- Hashed Columns
- Crossed columns
- Indicator and embedding columnss

It touches the topic of embeddings and gives an interesting rule of thumb to determine the number of embedding dimensions:

```
embedding_dimensions =  number_of_categories**0.25
```

At the end it is precised that not all premade estimators accept all kinds of feature columns. There is a list specifying which ones accept what.

12.04.18 ∼ Two (tree) APIs to create a model

It seems that there are various API's to create a model with TF. In the "Get started with Eager execution" tutorial here:
`https://www.tensorflow.org/get_started/eager`
the model is defined as follow using `keras.Sequential`

```
model = tf.keras.Sequential([
  tf.keras.layers.Dense(10, activation="relu", input_shape=(4,)),  # input shape required
  tf.keras.layers.Dense(10, activation="relu"),
  tf.keras.layers.Dense(3)
])
```

but in the "Pre-made estimator" tutorial found here:
`https://www.tensorflow.org/get_started/premade_estimators`
it is done using a pre-made estimator (`estimator.DNNClassifier`):

```
# Build a DNN with 2 hidden layers and 10 nodes in each hidden layer.
classifier = tf.estimator.DNNClassifier(
    feature_columns=my_feature_columns,
    # Two hidden layers of 10 nodes each.
    hidden_units=[10, 10],
    # The model must choose between 3 classes.
    n_classes=3)
```

I guess that these feature columns are used only by these classes of pre-made estimator. **Edit (04.06.18):** Actually, it seems to be a common practice to build a python class encapsulating a NN model built with TF basic APIs. The assignment of the second week of coursera's course is a good example of this. So there are three ways to define a model.

I finished the tutorial on feature columns.

13.04.18 $\sim$ Loading a dataset, defining number of epochs (with estimator objects), input functions, Dataset objects
I started this tutorial about data sets (`Dataset`), **input functions**:
`https://www.tensorflow.org/get_started/datasets_quickstart`
"The `tf.data` module contains a collection of classes that allows you to easily load data, manipulate it, and pipe it into your model. This document introduces the API by walking through two simple examples:

- Reading in-memory data from numpy arrays.
- Reading lines from a csv file."

Edit (04.07.18):

- Here is a little summary of **how to load data in the model using Estimator, and Dataset APIs**: One has to coordinate and understand **tf.data.Dataset** (an object encapsulating the preprocessed data), **input functions** (functions which are arguments of the `train` and `evaluate` methods of the `tf.estimator.Estimator` object and which are supposed to return a Dataset object), and **feature columns** (an argument of the initializer of an estimator object which specifies the type, shape, and name of the inputs). One tricky detail I didn't see at first is that when creating a Dataset (typically with `tf.data.Dataset.from_tensor_slices`) one has to provide features in the form of a dictionary, and its keys are then used again in the definition of the feature columns (argument of the initializer of an Estimator object). For a good example, see this tutorial) and the premade_estimator.ipynb notebook I saved on my computer.

- To see how to use Dataset together with low-level TF APIs, see this part of the low level tutorial:
  `https://www.tensorflow.org/programmers_guide/low_level_intro#datasets`

Edit (15.07.18): I noticed some inconsistency about the **output type of an input function** According to this page, the input functions is supposed to return a `tf.data.Dataset` object as explained above:
`https://www.tensorflow.org/guide/premade_estimators#create_input_functions`
but in other tutorials, like this one on Custom Estimators or this one on how to turn a Keras model into an Estimator model, it looks like the input functions returns an iterator over a `tf.data.Dataset` object. My guess is that both are possible and that Estimators internally recognize in which case we are.

I think that I understood something concerning the way **epochs** are defined in the example premade_estimator.py. I get the impression that this is dealt with by the argument `steps` of the `train` method of objects of the class (or subclass) `Estimator`. If I understood it well, this argument fixes the number of batches through which the model will be trained (the number of steps of the training). So indirectly it fixes the number of epochs to

$$\#epochs = \#steps \times (batch\ size)/(data\ length).$$

In order to be able to have as many epochs as one wants, we use the method `repeat()` in the definition of the returned data set of the input function (as seen in the definition of `train_input_fn` inside iris_data.py).
In order to read a CSV file and tranform it to an input for the model, the first step is to use `tf.data.TextLineDataset` as here:

```
ds = tf.data.TextLineDataset(train_path).skip(1)
```

This builds a `TextLineDataset` object to read the file one line at a time. The object itself is still not a usual array. It looks more like an object having a pointer to the location of the CSV file being able to bring it to the model, one line at the time (I think that an Iterator has to be built in order to do that, explicitly or using a wrapper function).
The next tool to use is `tf.decode_csv` which allows to turn each line of the CSV file into something that we can really use (like a simple list). To apply a function which does the parsing on one line to the whole data set use the method `map` of the `TextLineDataset` object used before.

So to summarize the two approaches, in the first one we use a tool from the pandas library (`pd.read_csv` inside the function `load_data` inside iris_data.py) to load the data and then we transform the dataframe into a `data.Dataset` object, and in the second we use a tool from tensorflow to load the data (`data.TextLineDataset`).

I wanted to look inside a tensor to see what it was exactly but I didn't know how to do it. I remembered that I had done it in the past (c.f. entry of the 09.04.19) using

eager. So I enabled eager execution and repeated what I had done before and it worked. I also looked at this page explaining what the **Eager** module is:
Stackoverflow: What is tensorflow eager module for?
It seems to be a way to going from a static graph to a dynamic graph like pytorch, and hence making it easier to use.
But it seems that it is not possible to use (premade) estimators in Eager execution mode as I received this error when I tried to train the toy model of the tutorial:

```
Estimators are not supported when eager execution is enabled.
```

I finished the tutorial and here is the summary: "The `tf.data` module provides a collection of classes and functions for easily reading data from a variety of sources. Furthermore, `tf.data` has simple powerful methods for applying a wide variety of standard and custom transformations."
Edit (31.05.18): This should be completed by this page about **how to read data** in TF, giving a broader look on the possibilities to do the task:
`https://www.tensorflow.org/api_guides/python/reading_data`
Edit (09.07.18): This should be completed by this page:
`https://www.tensorflow.org/programmers_guide/datasets`
explaining (among other things)

- **how to use several datasets** in the same model (typically for training, validation, testing)
- how to **create datasets from different data** (data stored in an array, or data stored on the disk using TFRecord format)
- how to create a dataset which applies transformation to each element (using `Dataset.map()`)

18.04.18 ∼ A nice comparison between pytorch and tensorflow
In this pytorch tutorial, they show how to do the same thing with both pytorch and tensorflow. I found it quite instructive concerning how tensorflow works:
pytorch.org: pytorch with examples

19.04.18 ∼ Creating custom estimators
I started this tutorial about **creating custom estimators**:
`https://www.tensorflow.org/get_started/custom_estimators`
It seems to be very important because I guess in real applications, it is better to be able to define the architecture yourself. The tutorial is quite clear and well done. It introduces the API's `tf.feature_column.input_layer` and `tf.layers.dense` which allow to define the input, hidden, and output layers.
Edit (14.07.18): It seems that the tutorial has been moved to
tensorflow.org: Custom estimators
But I saved the directory

`/home/aritz/Documents/CS_Programming_Machine_Learning/Machine_learning_and_AI/Online_cou`

inside a notebook called custom_estimator.ipynb.

20.04.18 ∼ Finishing the custom estimator tutorial and problems with tensorboard

I finished the tutorial about custom estimators. Here is the conclusion of the tutorial: "Although pre-made Estimators can be an effective way to quickly create new models, you will often need the additional flexibility that custom Estimators provide. Fortunately, pre-made and custom Estimators follow the same programming model. The only practical difference is that you must write a model function for custom Estimators; everything else is the same."
The code of the example used in this tutorial is contained in the file custom_estimator.py and should be compared to premade_estimator.py. There are two main things that I got out of the tutorial concerning this `model_fn` function that we have to define (it is an argument of the constructor of the class `tf.estimator.Estimator`):

1. The first one is that the way the architecture is defined. In our example it is this portion of code:

```
net = tf.feature_column.input_layer(features, params['feature_columns'])
for units in params['hidden_units']:
    net = tf.layers.dense(net, units=units, activation=tf.nn.relu)

# Compute logits (1 per class).
logits = tf.layers.dense(net, params['n_classes'], activation=None)

# Compute predictions.
predicted_classes = tf.argmax(logits, 1)
```

2. We have define what the model returns for the three methods `predict()`, `train()`, and `evaluate()` by imposing conditions on the value of the argument `mode`.

If I run

```
python custom_estimator.py
```

I obtain a very long output. This line caught my attention

```
2018-04-20 16:37:05.900885: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU
```

From this page:
`https://stackoverflow.com/questions/47068709/your-cpu-supports-instructions-that-this-te`

this just means that I could optimize the way my CPU is used by tensorflow. But as the page cited explains it is useless since it is anyway better to try to use GPU.
Then looked at this line:

```
2018-04-20 16:37:05.902531: E tensorflow/stream_executor/cuda/cuda_driver.cc:406] failed
```

This is discussed here:
`https://github.com/tensorflow/tensorflow/issues/394`
and some solution mentioned in this tread is repeated here:
`http://kawahara.ca/tensorflow-failed-call-to-cuinit-cuda_error_unknown/`
But I don't know if I should try to apply the solution (i.e. do

```
sudo apt-get install nvidia-modprobe
```

or not because, my program still runs at the end. Thing is, it is not clear to me if TensorFlow manages to use my GPU or not, and what this error is about.

I also couldn't use tensorboard. When I open the webpage, I'm told that there is no information to display. After discussing with Nikos, it seems that no files containing the information used by Tensorboard is created (maybe the code I have is incomplete).

22.04.18 ∼ Verifying that Tensorflow is running on GPU
Since I wasn't sure that I was using the GPU, I looked at this page which gives several approaches to determine if Tensorflow is running on GPU or not:
Stackoverflow: How to tell if Tensorflow is using GPU acceleration from inside python shell
I entered

```
import tensorflow as tf
with tf.device('/gpu:0'):
    a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
    b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
    c = tf.matmul(a, b)

with tf.Session() as sess:
    print (sess.run(c))
```

and obtained the following input (which lets me believe that Tensorflow is running on GPU):

```
2018-04-22 10:54:45.457127: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU
2018-04-22 10:54:45.575997: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:898] s
2018-04-22 10:54:45.576464: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1344] Foun
```

```
name: GeForce 940MX major: 5 minor: 0 memoryClockRate(GHz): 1.2415
pciBusID: 0000:02:00.0
totalMemory: 1.96GiB freeMemory: 1.55GiB
2018-04-22 10:54:45.576479: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1423] Addi
2018-04-22 10:54:46.081373: I tensorflow/core/common_runtime/gpu/gpu_device.cc:911] Devic
2018-04-22 10:54:46.081399: I tensorflow/core/common_runtime/gpu/gpu_device.cc:917]
2018-04-22 10:54:46.081405: I tensorflow/core/common_runtime/gpu/gpu_device.cc:930] 0:
2018-04-22 10:54:46.081609: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1041] Crea
[[22. 28.]
 [49. 64.]]
```

Then I entered

```
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
```

The output was

```
2018-04-22 11:04:49.935218: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1423] Addi
2018-04-22 11:04:49.935309: I tensorflow/core/common_runtime/gpu/gpu_device.cc:911] Devic
2018-04-22 11:04:49.935346: I tensorflow/core/common_runtime/gpu/gpu_device.cc:917]
2018-04-22 11:04:49.935368: I tensorflow/core/common_runtime/gpu/gpu_device.cc:930] 0:
2018-04-22 11:04:49.935656: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1041] Crea
Device mapping:
/job:localhost/replica:0/task:0/device:GPU:0 -> device: 0, name: GeForce 940MX, pci bus i
2018-04-22 11:04:49.936007: I tensorflow/core/common_runtime/direct_session.cc:297] Devic
/job:localhost/replica:0/task:0/device:GPU:0 -> device: 0, name: GeForce 940MX, pci bus i
```

which looks like what is on stack exchange, so I guess that Tensorflow is using GPU.

I read this tutorial:
https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard
which is based on this code:
https://github.com/tensorflow/tensorflow/blob/r1.7/tensorflow/examples/tutorials/mnist/mnist_with_summaries.py
which is a bit more complex than the toy examples I had seen in the past.
I didn't understand everything. I will try to correct the example with the custom estimator as an exercise.

23.04.18 ~ Building a computational graph and running it

There is a section dedicated to tensorflow in this pytorch tutorial and it illustrates pretty well the basic APIs of TensorFlow:
http://pytorch.org/tutorials/beginner/pytorch_with_examples.html#pytorch-variables-and-a
One starts by building a computational graph (my understanding is that it is the architecture of the neural network, together with the weight updating scheme, but without any actual data) with "place holders" for the input and output data of the model, creating "variables" for the weights/parameters of the different layers, using only tensorflow

operations on tensors, and after that one uses these `sess.run` APIs (even though I don't get all the details).
I think it is closer to the core of tensor flow than `tf.estimator.Estimator` objects which seem to be a level of abstraction higher.

26.04.18 ~ Different ways to build a tensorflow model/estimator, Coming back to tensorboard
It looks like there are **several approaches to use tensorflow**, or more precisely **three levels of APIs** to use TF. I will try to list them here and provide examples for each one.

1. One way seems to be to build a computational graph directly from the basic APIs from Tensorflow and then use `sess.run()`. I saved an example under the name toy_basic_approach_pytorch.py that I had found there:
   `http://pytorch.org/tutorials/beginner/pytorch_with_examples.html#tensorflow-static-g`
   Another toy example comes from this TF tutorial:
   `https://www.tensorflow.org/programmers_guide/low_level_intro`
   and I saved the code under basic_APIs_tutorial_2.py.
   Edit (30.05.18): It is interesting to compare the two. In toy_basic_approach_pytorch.py the value of the parameters of the model is updated manually, but in basic_APIs_tutorial_2.py it is done with a subclass of the `tf.train.Optimizer`.
   Edit (04.05.18): Another good example is given by the second assignment of the NLP coursera class from Moscow university. It uses Tensor Flow basic API to perform Named-Entity-Recognition on tweets. The jupyter notebook is called week2-NER_modified.ipynb.

2. One way is to build `tf.estimator.Estimator` objects. From what I could see, it seems to be the best way since it offers all the possibility to create models with completely customizable architecture, while still satisfying the OOP paradigm with separate tasks being encapsulated in different objects with fixed names but flexible implementation. One example is in the file custom_estimator.py. copied from there:
   `https://www.tensorflow.org/get_started/custom_estimators#tensorboard`

3. One way is to use some library like **Keras**, `TFLearn`, or **TensorLayer** offering high-level API built on the top of TF to create a model. Keras imitates the way scikit-learn APIs work. I copied a little toy example under the name toy_keras_approach.py from here:
   `https://keras.io/getting-started/sequential-model-guide/`

27.04.18 ~ Managing to use Tensorboard with Estimator and basic APIs
I managed to use **tensorboard** as I was supposed to do in this tutorial (with **Estimator level APIs**):

https://www.tensorflow.org/get_started/custom_estimators#tensorboard
There are several reasons why I couldn't do it in the first place:

1. The proper tensorboard tutorial: https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard
   uses the basic tensorflow API, with `sess.run()` whereas here we use the `tf.estimator.Estimator` APIs which hide a lot of things. It seems that in this later case, the `tf.summary.merge_all` and `tf.summary.FileWriter` APIs are somehow hidden internally in the body of the Estimator, as explained/hinted there:
   Tensorboard: how to add tensorboard to a tensorflow estimator process?

2. It looks like these "summary log" are sent to the "`model_dir`" directory. This is one of the optional arguments of the constructor of a `tf.estimator.Estimator` object. The default value was not clear to me so I created an explicit temporary directory for it with these lines:

   ```
   LOGDIR = "/tmp/custom_estimator_tutorial/"
   model_dir = "/tmp/custom_estimator_tutorial/"+"model_dir/"
   ```

   and (see last argument)

   ```
   classifier = tf.estimator.Estimator(
       model_fn=my_model,
       params={
           'feature_columns': my_feature_columns,
           # Two hidden layers of 10 nodes each.
           'hidden_units': [10, 10],
           # The model must choose between 3 classes.
           'n_classes': 3,
       },
       model_dir=model_dir)
   ```

It looks like the graph is automatically added to the summary logs when using tensorboard also (since I could see it, without adding a single line of code). But I have difficulty interpreting my plot of the computational graph...
What I take from this is that in order to use Tensorboard with a custom estimator, one has to do the following:

1. Add lines like:

   ```
   tf.summary.scalar('accuracy', accuracy[1])
   ```

   in order to produce some graphs or histograms of some scalar values (see the tutorial page for more options), inside the body of the model function of the `tf.estimator.Estimator` that we want to create.

2. Give an extra `model_dir` argument to the constructor of the `tf.estimator.Estimator` object as follow

```
classifier = tf.estimator.Estimator(
    model_fn=my_model,
    params={
        'feature_columns': my_feature_columns,
        # Two hidden layers of 10 nodes each.
        'hidden_units': [10, 10],
        # The model must choose between 3 classes.
        'n_classes': 3,
        },
        model_dir=model_dir)
```

   where `model_dir` is the path where we put the "summary logs". See for instance

```
model_dir = "/tmp/custom_estimator_tutorial/"+"model_dir/"
```

3. Use the following command in the terminal with the model_dir path as argument:

```
aritz@aritz-ThinkPad-T460p:/tmp/custom_estimator_tutorial/model_dir$ tensorboard --l
```

4. Go to this address in the browser: http://localhost:6006 (or click on the link displayed in this tutorial:`https://www.tensorflow.org/get_started/custom_estimators#tensorboard`).

It looks like a new summary log is created by the train method of an estimator every 100 steps (by default). I guess this could be changed if one used something like

```
my_estimator = tf.estimator.Estimator(
model_fn = my_model_fn,
model_dir = my_model_dir,
config=tf.contrib.learn.RunConfig(
save_checkpoints_steps=20,
save_checkpoints_secs=None,
save_summary_steps=40,
)
)
```

Edit (13.07.18): I also managed to use **Tensorboard with the basic APIs** (in my IMDB sentiment analysis project) even if I still have problems with the `global_step` argument of the `add_summary` method. One of the main difference, is that one has to explicitely define one (or serveral) "writer(s)" (instance(s) of the class `tf.train.SummaryWriter`) and tell it (them) when to write the event files which are used by Tensorboard, and what to put in it. There is a relatively simple example there:

`https://stackoverflow.com/questions/37902705/how-to-manually-create-a-tf-summary#37915182`
even if most of the time, the actual summary is not created this way but by evaluating a variable `merged` defined by something like

```
tf.summary.scalar('accuracy', accuracy[1])
tf.summary.scalar('loss', loss)
merged = tf.summary.merge_all()
```

inside the computational graph, with some command like

```
summary, acc = sess_debug.run([merged, accuracy])
```

and then calling the writer like this

```
writer_train.add_summary(summary, step)
```

where the writer has been previous defined by

```
writer_train = tf.summary.FileWriter(model_dir+'Basic_log/Plot_train/', sess_debug.graph)
```

One of the tricks to differentiate the curve of the training and the curve of the testing on Tensorboard is to use two different writers for the training and for the testing set as explained there:
Quora: How to plot training nad validation loss on the same graph using TensorFlow

For an **tutorial showing more capabilities of Tensorboard** see this page:
`https://www.tensorflow.org/guide/summaries_and_tensorboard`

This **video** displays other **fancy visualization tools of Tensorboard**:
Youtube: Hands-on TensorBoard (TensorFlow Dev Summit 2017)
There is a way to **embed the multidimensional space of labels in a three dimensional space to visualize the distance between the different samples**. It can be useful in NLP.

I started this tutorial about **low-level TensorFlow APIs**:
tensorflow.org: low level intro
But the basic example with tensorboard doesn't work. It looks like no event file is produced.

29.04.18 $\sim$ **TF basic API's:** `tf.Graph`, `tf.Session`
I came back to this tutorial: tensorflow.org: low level intro
and it seems to be working (I mean Tensorboard is displaying the graph of the addition of the two constants now, if I enter the constants described whereas it wasn't displaying anything the other day).
Here is a little list of these basic APIs that are covered in this tutorial:

- `tf.constant`
- `tf.summary.FileWriter`
- `writer.add_graph(tf.get_default_graph())`
- `tf.Session()`
- `sess.run`
- `sess.run('ab':(a, b), 'total':total)`
- `tf.placeholder`
- `tf.data.Dataset.from_tensor_slices`
- `slices.make_one_shot_iterator()`
- `get_next()`
- `sess.run(iterator.initializer)`
- `tf.layers.Dense` and `tf.layers.dense`
- `tf.global_variables_initializer()`
- `tf.feature_column.categorical_column_with_vocabulary_list`
- `tf.feature_column.indicator_column`
- `tf.feature_column.numeric_column`
- `tf.feature_column.input_layer` (accepts only dense columns as inputs (c.f. the tutorial section "Feature columns").
- `tf.tables_initializer()`

13.05.18 ~ Break and links toward tutorials and NLP coursera courses
I didn't touch Tensorflow for a while due to my job searches. I will put here the **links toward the tutorials**:
`https://www.tensorflow.org/get_started/`
`https://www.tensorflow.org/get_started/custom_estimators`
`https://www.tensorflow.org/programmers_guide/low_level_intro`
and the **NLP coursera course**:
`https://www.coursera.org/learn/language-processing/home/welcome`

15.05.18 ~ Starting a new Journal for Coursera NLP course
I will start a separate journal for the Coursera course on NLP. Since the course is based on TensorFlow, there will be some overlap with this journal.

30.05.18 ~ An advanced implementation of LSTM with dropout, how to set parameters of a model via command line arguments
I copied the files ptb_word_lm.py and reader.py from
`https://github.com/tensorflow/models/tree/master/tutorials/rnn/ptb`
to be able to read this tutorial about **RNN with tensorflow to build a language model**:
`https://www.tensorflow.org/tutorials/recurrent`
(I had actually already downloaded it together with all the TF tutorials in the folder "(...)/Online_courses/Tensor_flow_tutorials/Official_tutorials/models/tutorials/rnn/ptb/")
I started reading the code.
I discovered (in this code the) `flags.DEFINE_string` (and similar) API which allows to **specify parameters of the model with command line arguments** before running the training.

31.05.18 ~ More about this RNN model, a good tutorial for TF basic APIs
One thing I don't fully understand in the code above is why do we put the model in a class.
I think that the way the data is read in this example is outdated (it uses Queue Runner but according to this page this is better to use the `tf.data` API now).
I learned about the usefulness of

```
with tf.Graph().as_default():
```

there:
Stack Overflow: Why do we need TensorFlow tf.Graph?
I read this tutorial about the **basic APIs of tensorflow** and **how tensorflow is organized**:
`http://www.goldsborough.me/tensorflow/ml/ai/python/2017/06/28/20-21-45-a_sweeping_tour_of_tensorflow/`
and it is very good. In particular it explains well APIs like

- ```
  graph = tf.Graph()
  with graph.as_default():
  ```
- `with tf.Session(graph=graph) as session:`
- `with tf.Session():`
- `with tf.name_scope('conv1'):`

Otherwise I gave up the idea of going in all the details of this code:
`https://github.com/tensorflow/models/blob/master/tutorials/rnn/ptb/ptb_word_lm.py`
  ```

because it is to tedious and maybe useless. But I keep it in mind as an example of a complex network.

03.06.18 ~ A useful example for basic API and LSTM
The second assignment of the NLP coursera course is a very good example of how to build and train a model with the basic APIs of TF. The aim is to build a language model based on LSTM.
I would like to try various modification on it. I will list here ideas I have and should try later:

- Replace the variable declaration using `tf.Variable` by `tf.get_variable`
- Try to use Estimators instead of the basic APIs
- Try to use Keras instead of the basic APIs
- Use spacy and other tools to feed the data into the model, build the dictionary, etc...
- Try to use the `tf.data` to replace `batches_generator`.
- Can we use pretrained embeddings for instance? Or pretrained models?

04.06.18 ~ Summary of lessons learned with second assignment from Coursera NLP course
In the second assignment of the Coursera NLP course that I follow currently, I had to use an LSTM RNN implemented with TF to perform Named Entity recognition. I will list here the useful things I learned. Not everything is TF API's but I will still write it here.

- How to **read a text file** containing all the tweets separated by line jumps, and build two lists of lists. In the first one each sublist contains strings for the tokens present in the corresponding tweet. In the second each sublist contains strings for the tags of the tokens of the corresponding tweet. Implemented in `read_data`
- How to **build the dictionaries** mapping tokens to indexes and indexes to token, using the outputs of `read_data`. Implemented in `build_dict`.
- How to **create a batch-generator** from scratch. Implemented in `batches_generator`.
- How to **build a model using TF-basic APIs** and embed it in a class with all necessary methods. More precisely, the different steps/methods were:
  - Declare placeholders which will recieve the inputs.
  - Build the different layers:
    * Embeddings of indices representing tokens, into $\mathbb{R}^n$ vectors.

    - BiLSTM layer
    - Dense layer
  - Compute the predictions (using softmax, and then taking the argmax).
  - Compute the loss
  - Perform optimization, AKA adjust the weights.
  - How to train the model on one batch. This part is quite subtle. Upon initialization of the model the whole graph is built. And then we use a `tf.Session` to run the operation which adjusts the weights. To compute this one, all the previous ones have to be executed as well.
  - Predict the output for one batch of data.

- How to evaluate a given model on a batch of data (independently of the model) to get predictions and transforms indices to tokens and tags. Implemented in `predict_tags`
- How to calculate precision, recall and F1 for the results. Implemented in `eval_conll`.

15.06.18 ~ TF wrappers: Keras, TFLearn, TensorLayer, an interesting post about sentiment analysis with TF

I just realized that there is not only Keras which is built on the top of TF, but also **TFLearn** (see for instance this code for an example), and **TensorLearn**. Here are a few webpages I read comparing the respective performances of these libraries:
`https://datascience.stackexchange.com/questions/25317/what-are-the-pros-and-cons-of-kera`
`https://www.reddit.com/r/MachineLearning/comments/50eokb/which_one_should_i_choose_keras_tensorlayer/`
`https://progur.com/2018/04/tflearn-vs-keras-which-better.html`

I read this post about using TF for sentiment analysis:
`http://domkaukinen.com/sentiment-analysis-with-tensorflow/`
it gives plenty of useful tips.

10.07.18 ~ Computing accuracy over different batches with TF basic APIs

I learned about `tf.metrics.accuracy` in the documentation:
`https://www.tensorflow.org/api_docs/python/tf/metrics/accuracy`
and also in this Stack Overflow post which provides a good insight:
`https://stackoverflow.com/questions/46409626/how-to-properly-use-tf-metrics-accuracy`
It looks like it computes the accuracy **over all the batches**. It is not the accuracy on each specific batch of data. This page gives some further details:
`https://stackoverflow.com/questions/46409626/how-to-properly-use-tf-metrics-accuracy`

14.07.18 ∼ **Saving and restoring models with Estimators APIs**
For a while, I didn't know how to save and restore a tf.estimator.Estimator that I had created and trained. I found this tutorial which explains everything:
tensorflow.org: checkpoints
In brief there are two ways to do it:

- with checkpoints, which is a format dependent on the code that created the model.
- SavedModel, which is a format independent of the code that created the model.

The first method is explained in this tutorial. Basically, everything is done automatically. If one reuse a script where the same estimator is defined, and its `model_dir` argument corresponds to the `model_dir` argument of the estimator previously defined and trained, the new estimator will automatically reuse the "checkpoint" and "model..." files created by the previous one (without having to write any command for this).
For the other method (which might be better in an industry setting), this link seems to be the right place to look for a way to **save estimators models**:
tensorflow.org: using SavedModel with Estimators

15.07.18 ∼ **Turn a Keras model into a `tf.estimator.Estimator`**
I read this tutorial explaining how to **transform a Keras model into a TF custom estimator**:
`https://www.dlology.com/blog/an-easy-guide-to-build-new-tensorflow-datasets-and-estimato`
It seems that the opposite is not as easy.

I found this list of **Keras models** for various tasks:
`https://github.com/keras-team/keras/tree/master/examples`

I tried to **install Keras**. When I tried to install it inside my 'tensorflow' environment, a message warned me that it would have to first install tensorflow 1.1.0. This seems strange to me because it seems I have a more advanced version of tensorflow already. But it may be because I installed 'tensorflow-gpu' (version 1.6.0 according to the output of `conda list`) and not simply 'tensorflow'. What is even stranger is that when I enter the command I found on stack exchange to know if I have tensorflow or not (while having my 'tensorflow' environment activated):

```
ython3 -c 'import tensorflow as tf; print(tf.__version__)'
```

I'm told that I have the version 1.7.0...
So I followed the advice of Nikos and I created a new conda environment named 'keras' where I will download keras:

```
conda create -n keras python=3.5 pip=9.0.1
```

to avoid messing with my tensorflow.

23.07.18 ~ TensorBoard with Keras
It seems that it is possible to use TensorBoard with Keras:
use this link together with this one.

3 Questions to ask to Nikos

1. How does he start implementing a model? What are the first things he defines? Does he use Jupyter notebook for testing? Does he use some smaller dataset to make tests? These TF models are quite complex, the chance of doing everything right from the start are slim.
2. Could he give me his code? It sounds like a good example to train.
3. When a model (like a Neural network) can have plenty of different parameters or variation (number of layers, simple LSTM vs Bi-LSTM, etc...) how does he find the one which performs the best? Does he do an exhaustive search? Is there some easier approach?

4 Questions

1. What are these different "scopes" and "graph"? Is it only to be able to have a clearer representation in Tensorboard? For instance, at the end of this script:
`https://github.com/tensorflow/models/blob/master/tutorials/rnn/ptb/ptb_word_lm.py`
there is

```
with tf.Graph().as_default():
   initializer = tf.random_uniform_initializer(-config.init_scale,
                                               config.init_scale)

   with tf.name_scope("Train"):
     train_input = PTBInput(config=config, data=train_data, name="TrainInput")
     with tf.variable_scope("Model", reuse=None, initializer=initializer):
       m = PTBModel(is_training=True, config=config, input_=train_input)
     tf.summary.scalar("Training Loss", m.cost)
     tf.summary.scalar("Learning Rate", m.lr)

   with tf.name_scope("Valid"):
```

```
      valid_input = PTBInput(config=config, data=valid_data, name="ValidInput")
      with tf.variable_scope("Model", reuse=True, initializer=initializer):
        mvalid = PTBModel(is_training=False, config=config, input_=valid_input)
tf.summary.scalar("Validation Loss", mvalid.cost)
```

2. How to use this `tf.Print`? Very often I would like to test that what I do makes sense and it would be convenient to be able to print the content of my `tf.string` or `tf.data`.

3. On this page:
   `https://www.tensorflow.org/get_started/checkpoints`
   I don't really understand what is meant by "To run experiments in which you train and compare slightly different versions of a model, save a copy of the code that created each `model_dir`, possibly by creating a separate git branch for each version".

5 TO DO list

1. I should learn more how to make custom layers with Keras to be able to have the power of TF inside Keras.