Tensorflow 2.x Review Session

CS330: Deep Multi-task and Meta Learning
9/17/2019
Rafael Rafailov

Overview

1. Installation
   a. Installing on your machine
   b. Using GPUs
   c. Using Google Colab
2. Tensorflow Basics
   a. Data pipelines
   b. Autograd in TF 2.0
   c. Models
   d. Optimizers
   e. Training loop
3. Other topics
   a. Layers with memory (for HW1)
   b. Tensorflow Probability

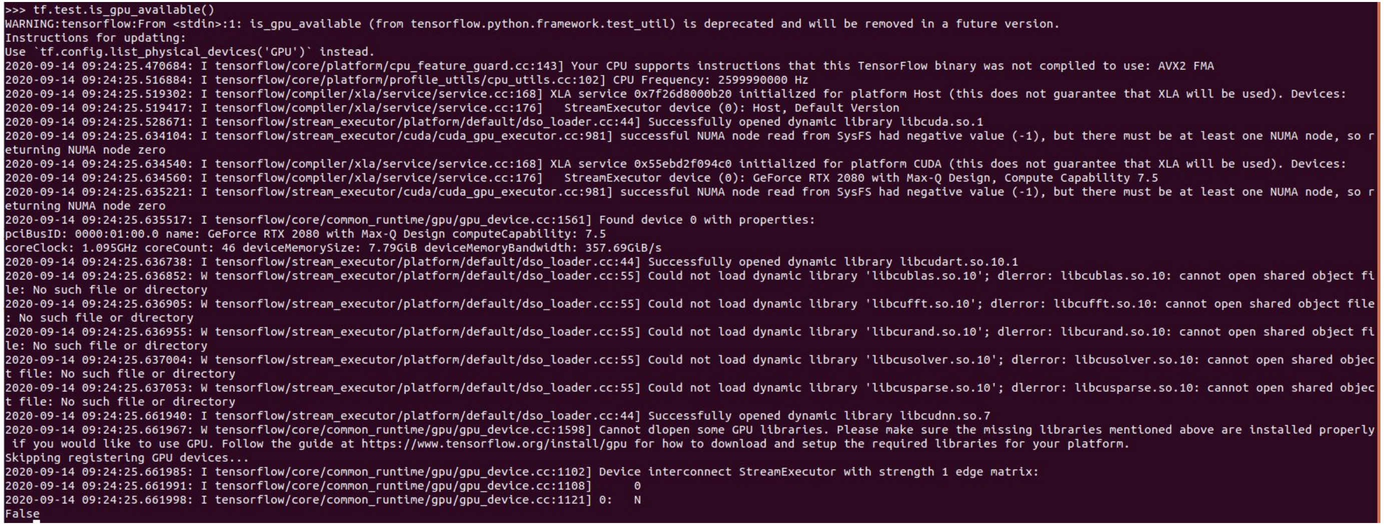
Installing on your machine/cloud instance

Directly:

```
# Requires the latest pip
pip install --upgrade pip

# Current stable release for CPU and GPU
pip install tensorflow(-gpu)
```

Important - make sure you’re actually using the GPU

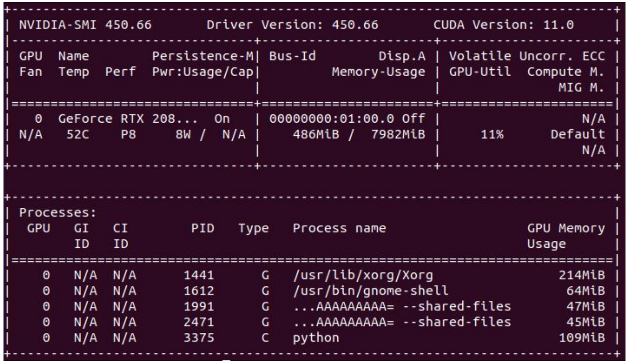
```
tf.test.is_gpu_available(
    cuda_only=False, min_cuda_compute_capability=None
)
```



```
>>> tf.test.is_gpu_available()
WARNING:tensorflow:From <stdin>:1: is_gpu_available (from tensorflow.python.framework.test_util) is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
2020-09-14 09:24:25.470684: I tensorflow/core/platform/cpu_feature_guard.cc:143] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
2020-09-14 09:24:25.516884: I tensorflow/core/platform/profile_utils/cpu_utils.cc:102] CPU Frequency: 2599990000 Hz
2020-09-14 09:24:25.519302: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x7f26d8000b20 initialized for platform Host (this does not guarantee that XLA will be used). Devices:
2020-09-14 09:24:25.519417: I tensorflow/compiler/xla/service/service.cc:176]   StreamExecutor device (0): Host, Default Version
2020-09-14 09:24:25.528671: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcuda.so.1
2020-09-14 09:24:25.634104: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:981] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so r
eturning NUMA node zero
2020-09-14 09:24:25.634540: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x55ebd2f094c0 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:
2020-09-14 09:24:25.634560: I tensorflow/compiler/xla/service/service.cc:176]   StreamExecutor device (0): GeForce RTX 2080 with Max-Q Design, Compute Capability 7.5
2020-09-14 09:24:25.635221: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:981] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so r
eturning NUMA node zero
2020-09-14 09:24:25.635517: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1561] Found device 0 with properties:
pciBusID: 0000:01:00.0 name: GeForce RTX 2080 with Max-Q Design computeCapability: 7.5
coreClock: 1.095GHz coreCount: 46 deviceMemorySize: 7.79GiB deviceMemoryBandwidth: 357.69GiB/s
2020-09-14 09:24:25.636738: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudart.so.10.1
2020-09-14 09:24:25.636852: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load dynamic library 'libcublas.so.10'; dlerror: libcublas.so.10: cannot open shared object fi
le: No such file or directory
2020-09-14 09:24:25.636905: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load dynamic library 'libcufft.so.10'; dlerror: libcufft.so.10: cannot open shared object file
: No such file or directory
2020-09-14 09:24:25.636955: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load dynamic library 'libcurand.so.10'; dlerror: libcurand.so.10: cannot open shared object fi
le: No such file or directory
2020-09-14 09:24:25.637004: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load dynamic library 'libcusolver.so.10'; dlerror: libcusolver.so.10: cannot open shared objec
t file: No such file or directory
2020-09-14 09:24:25.637053: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load dynamic library 'libcusparse.so.10'; dlerror: libcusparse.so.10: cannot open shared objec
t file: No such file or directory
2020-09-14 09:24:25.661940: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudnn.so.7
2020-09-14 09:24:25.661967: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1598] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly
 if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.
Skipping registering GPU devices...
2020-09-14 09:24:25.661985: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1102] Device interconnect StreamExecutor with strength 1 edge matrix:
2020-09-14 09:24:25.661991: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1108]      0
2020-09-14 09:24:25.661998: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1121] 0:   N
False
```

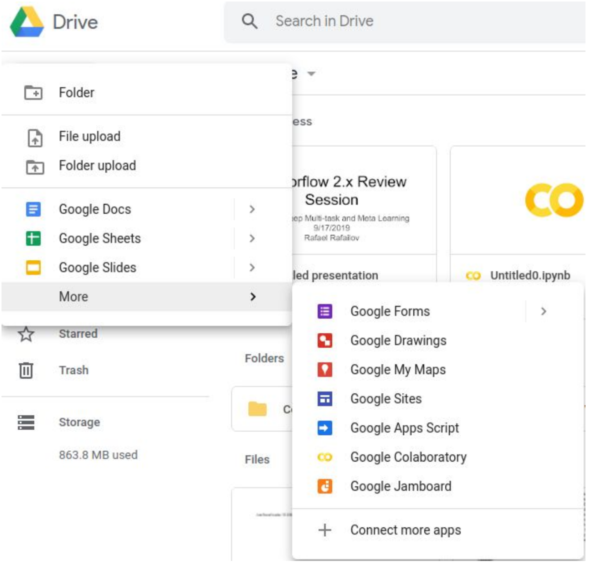
Need to have CUDA packages installed

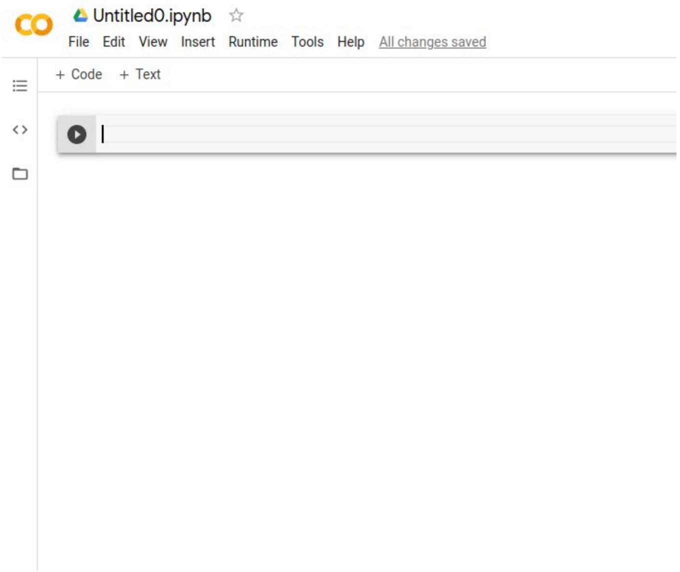
1. Instructions are here - https://www.tensorflow.org/install/gpu
2. Alternatively, to install CUDA dependencies using conda (I find this easier, especially if you do not have sudo access on the machine): https://towardsdatascience.com/managing-cuda-dependencies-with-conda-89c5d817e7e1
3. To check CUDA version - nvidia-smi



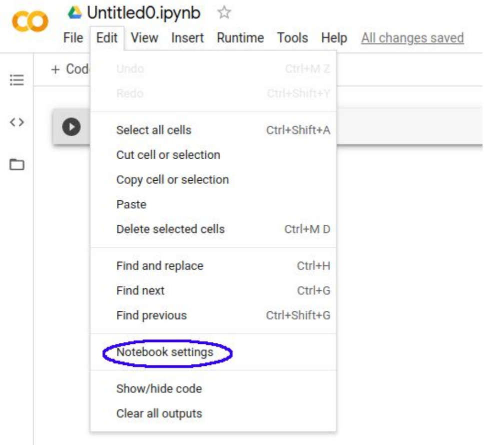
```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.66       Driver Version: 450.66       CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  GeForce RTX 208...  On   | 00000000:01:00.0 Off |                  N/A |
| N/A   52C    P8     8W /  N/A |    486MiB /  7982MiB |     11%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A      1441      G   /usr/lib/xorg/Xorg                214MiB |
|    0   N/A  N/A      1612      G   /usr/bin/gnome-shell               64MiB |
|    0   N/A  N/A      1991      G   ...AAAAAAAAA= --shared-files       47MiB |
|    0   N/A  N/A      2471      G   ...AAAAAAAAA= --shared-files       45MiB |
|    0   N/A  N/A      3375      C   python                            109MiB |
+-----------------------------------------------------------------------------+
```

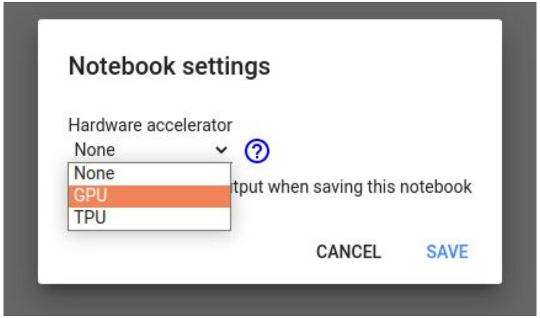
Using Google Colaboratory Notebooks



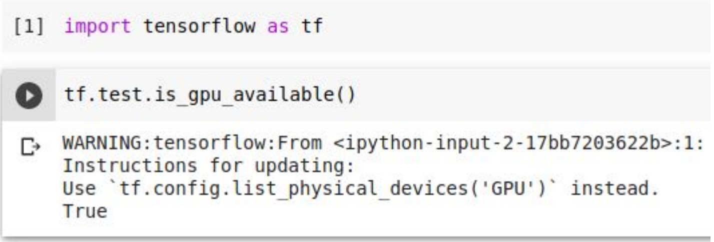


Using GPU in Colabs - no GPU by default!




Notebook settings

Hardware accelerator
None


```
[1] import tensorflow as tf
```

```
tf.test.is_gpu_available()
```

```
WARNING:tensorflow:From <ipython-input-2-17bb7203622b>:1:
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
True
```

Getting started

1. Colab notebooks already have tensorflow (and GPUs) set up.
2. Homeworks should be doable on CPUs too, but might take a bit longer (couple of hours).
3. Only need to set-up TF 2.x is you’re using a separate instance for your project (can be done in PyTorch too).

Tensorflow Data pipelines

```
dataset = tf.data.Dataset.from_generator(generator, types, shapes)

dataset = dataset.batch(batch_size, drop_remainder=True)

dataset = dataset.map(preprocess)

dataset = dataset.prefetch(10)
```

1. Create a TF dataset object from python generator object
2. Create a batched dataset (i.e. sample batches of batch_size)
3. Apply preprocess() to each batch before returning it (e.g. normalize images to [0,1])
4. Preload batches while computation is running - significant speed up in data I/O

More functionality: https://www.tensorflow.org/api_docs/python/tf/data/Dataset

Tensorflow Gradients and Autodiff

```
w = tf.Variable(tf.random.normal((3, 2)), name='w')
b = tf.Variable(tf.zeros(2, dtype=tf.float32), name='b')
x = [[1., 2., 3.]]

with tf.GradientTape(persistent=True) as tape:               -> Gradient tape tracks differentiable
  y = x @ w + b                                                  operations
  loss = tf.reduce_mean(y**2)                                 -> persistent = True keeps compute
                                                                 graph after tape.gradient

[dl_dw, dl_db] = tape.gradient(loss, [w, b])                 -> Computes tracked variable grads

print(y)
print(dl_db)
tf.Tensor([[ 1.9099498 -8.337775 ]], shape=(1, 2), dtype=float32)
tf.Tensor([ 1.9099499 -8.337775 ], shape=(2,), dtype=float32)
```

Tensorflow Gradients and Autodiff

```
x0 = tf.Variable(3.0, name='x0')
x1 = tf.Variable(3.0, name='x1', trainable=False)
x2 = tf.Variable(2.0, name='x2') + 1.0
x3 = tf.constant(3.0, name='x3')

with tf.GradientTape() as tape:
  y = (x0**2) + (x1**2) + (x2**2)

grad = tape.gradient(y, [x0, x1, x2, x3])

tf.Tensor(6.0, shape=(), dtype=float32)
None
None
None
```

Tensorflow Models

```
import tensorflow as tf
from tensorflow.keras.layers import Dense, Flatten, Conv2D
from tensorflow.keras import Model

class MyModel(Model):                                     -> Define model
  def __init__(self):
    super(MyModel, self).__init__()
    self.conv1 = Conv2D(32, 3, activation='relu')
    self.flatten = Flatten()
    self.d1 = Dense(128, activation='relu')               -> Define model layers
    self.d2 = Dense(10)

  def call(self, x):
    x = self.conv1(x)
    x = self.flatten(x)                                   -> Model processing
    x = self.d1(x)
    return self.d2(x)

# Create an instance of the model
model = MyModel()                                         -> Initialize Model
```

TF Keras layers: https://www.tensorflow.org/api_docs/python/tf/keras/layers

Tensorflow Losses and Metrics

Losses :

```
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
loss_object = tf.keras.losses.MeanSquaredError()
```

Other losses: https://www.tensorflow.org/api_docs/python/tf/keras/losses

Metrics:

```
test_loss = tf.keras.metrics.Mean()
test_accuracy = tf.keras.metrics.SparseCategoricalAccuracy()
test_top_k_accuracy = tf.keras.metrics.SparseTopKCategoricalAccuracy(k=5)
```

Other metrics: https://www.tensorflow.org/api_docs/python/tf/keras/metrics

Tensorflow Optimizers

```
optimizer = tf.keras.optimizers.Adam(
    learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07, amsgrad=False,
    name='Adam')

optimizer = tf.keras.optimizers.RMSprop(
    learning_rate=0.001, rho=0.9, momentum=0.0, epsilon=1e-07, centered=False,
    name='RMSprop')

optimizer = tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.0, nesterov=False, name='SGD')
```

Available optimizers: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers

Putting it all together

```
def train_step(images, labels):
  with tf.GradientTape() as tape:
    predictions = model(images, training=True)                              -> sets behaviour for
    loss = loss_object(labels, predictions)                                    Dropout() layers etc
  gradients = tape.gradient(loss, model.trainable_variables)                -> compute model grads
  optimizer.apply_gradients(zip(gradients, model.trainable_variables))      -> apply grads to weights
```

Applying gradients by hand

Applying gradients by hand (useful for optimization-based meta-learning):

```
gradients = tape.gradient(inner_loss, trainable_weights)
new_weights = ([weight - lr_inner * grad for weight, grad in zip(trainable_weights,
                                                                  gradients)])
```

Important!

```
model.set_weights(weights)
variable.assign(weight)
```

Will break the computation graph (will become clear later on)!

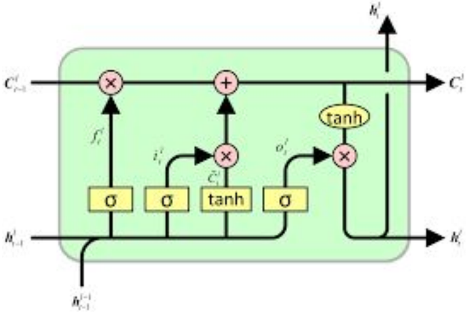
Let's run it!

Colab is here:

https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/quickstart/advanced.ipynb

Recurrent Cells

```
tf.keras.layers.LSTMCell(units)

import tensorflow as tf

inputs = tf.random.normal([32, 10, 8])                                 -> batch x length x size of data

cell = tf.keras.layers.LSTMCell(4)
state = cell.get_initial_state(batch_size = 32, dtype = tf.float32)    -> initialize cell state
output, state = cell(inputs[:,0], state)                               -> process data one at a time

print(output.shape)
print(state[0].shape)
print(state[1].shape)
(32, 4)
(32, 4)
(32, 4)
```



Recurrent Networks

```
inputs = tf.random.normal([32, 10, 8])
rnn = tf.keras.layers.RNN(tf.keras.layers.LSTMCell(4))    -> wrap cell to process sequence
output = rnn(inputs)
print(output.shape)
(32, 4)
rnn = tf.keras.layers.RNN(
   tf.keras.layers.LSTMCell(4),
   return_sequences=True,
   return_state=True)
whole_seq_output, final_memory_state, final_carry_state = rnn(inputs)
print(whole_seq_output.shape)
(32, 10, 4)
print(final_memory_state.shape)
(32, 4)
print(final_carry_state.shape)
(32, 4)
```

Recurrent Networks

```
inputs = tf.random.normal([32, 10, 8])
lstm = tf.keras.layers.LSTM(4)
output = lstm(inputs)
print(output.shape)
(32, 4)
lstm = tf.keras.layers.LSTM(4, return_sequences=True, return_state=True)
whole_seq_output, final_memory_state, final_carry_state = lstm(inputs)
print(whole_seq_output.shape)
(32, 10, 4)
print(final_memory_state.shape)
(32, 4)
print(final_carry_state.shape)
(32, 4)
```

Black-box model for HW1

Tensorflow Probability

Installation:

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

Uses:

1. Generative models (i.e. VAEs, Autoregressive Models, Normalizing Flows)
2. Statistical Models (i.e. Bayesian Models, Hamiltonian MCMC)
3. Reinforcement Learning (i.e. stochastic policies)

Some advanced examples:

https://github.com/tensorflow/probability/tree/master/tensorflow_probability/examples

Tensorflow Probability

```
import tensorflow as tf
import tensorflow_probability as tfp

mean = tf.Variable([1.0, 2.0, 3.], name='mean')
std = tf.Variable([0.1, 0.1, 0.1], name='std')
var = tf.constant([3.0, 0.1, 2.0], name='var')

with tf.GradientTape(persistent=True) as tape:
    dist = tfp.distributions.Normal(loc = mean, scale = std)
    s = dist.sample()
    loss1 = tf.reduce_mean(s**2)
    loss2 = tf.reduce_mean(dist.log_prob(var))
    loss3 = tf.reduce_mean(dist.log_prob(s))

grad1 = tape.gradient(loss1, [mean])
grad2 = tape.gradient(loss2, [mean])
grad3 = tape.gradient(loss3, [mean])

print(grad1)
print(grad2)
print(grad3)
```

```
[<tf.Tensor: shape=(3,), dtype=float32, numpy=array([0.63096106, 1.3687671 , 1.9575679 ], dtype=float32)>]
[<tf.Tensor: shape=(3,), dtype=float32, numpy=array([ 66.66667 , -63.333336, -33.333336], dtype=float32)>]
[<tf.Tensor: shape=(3,), dtype=float32, numpy=array([0., 0., 0.], dtype=float32)>]
```

TF Agents

```
pip install tf-agents

import tensorflow as tf
from tf_agents.networks import q_network
from tf_agents.agents.dqn import dqn_agent

q_net = q_network.QNetwork(
  train_env.observation_spec(),
  train_env.action_spec(),
  fc_layer_params=(100,))

agent = dqn_agent.DqnAgent(
  train_env.time_step_spec(),
  train_env.action_spec(),
  q_network=q_net,
  optimizer=optimizer,
  td_errors_loss_fn=common.element_wise_squared_loss,
  train_step_counter=tf.Variable(0))

agent.initialize()
```

https://www.tensorflow.org/agents

Questions?