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
)
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
Using Google Colaboratory Notebooks
Using GPU in Colabs - no GPU by default!

```
[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

```python
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](https://www.tensorflow.org/api_docs/python/tf/data/Dataset)
Tensorflow Gradients and Autodiff

```python
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:
    y = x @ w + b
    loss = tf.reduce_mean(y**2)

[dl_dw, dl_db] = tape.gradient(loss, [w, b])

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

class MyModel(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:

```python
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](https://www.tensorflow.org/api_docs/python/tf/keras/losses)

Metrics:

```python
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](https://www.tensorflow.org/api_docs/python/tf/keras/metrics)
Tensorflow Optimizers

```python
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](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers)
def train_step(images, labels):
    with tf.GradientTape() as tape:
        predictions = model(images, training=True)
        loss = loss_object(labels, predictions)
        gradients = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(gradients, model.trainable_variables))

-> sets behaviour for Dropout() layers etc
-> compute model grads
-> apply grads to weights
Applying gradients by hand

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

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

**Important!**

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

Will break the computation graph (will become clear later on)!
Let’s run it!

Colab is here:
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

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

```python
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
import tensorflow as tf
import tensorflow_probability as tfp

mean = tf.Variable([1.0, 2.0, 3.0], 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?