Eager Execution in Tensorflow

Eager Execution

Om Shri Prasath
ee17b113@smail.iitm.ac.in

SysDL Recitation
CS 6886

RECITATION OUTLINE

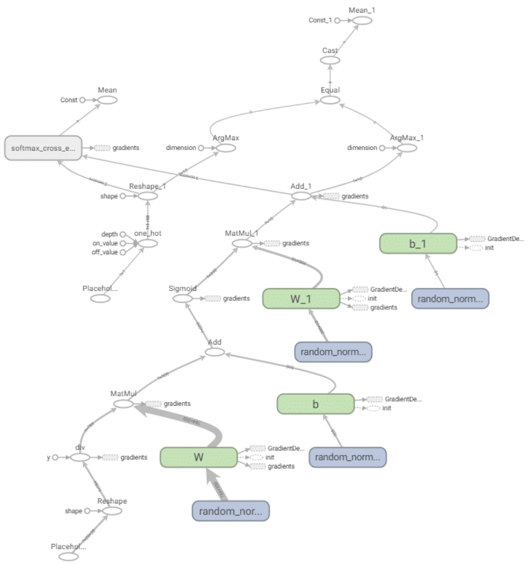
- Tensorflow and its Important Features
- Eager Execution
- Comparing Code Structure
  - **Tensorflow 1.x** vs **Tensorflow 2.0**
- Features of eager execution
- Combining graph and eager using AutoGraph and tf.function
- Eager Training
- Demo

TENSORFLOW

Free and open-source software library for machine learning based on dataflow and differentiable programming.

IMPORTANT FEAUTURES :

- Computation Graph
- Automatic gradient calculation
- Good GPU support (CUDA)
- Parallel computation support
- Varied deployment support
- Tensorboard
- **Eager execution**



EAGER EXECUTION

An imperative programming environment that evaluates operations immediately, without building graphs.

Eager execution provides the following advantages :

- Intuitive coding which is easier for beginners to understand
- Easier debugging due to immediate execution
- Support for dynamic objects with complex control flows

SO, HOW'S THE CODE DIFFERENT?

CODING IN TENSORFLOW 1.X : GRAPH MODE

- Manually stitch together an abstract syntax tree (the graph) by making **tf.*** API calls.
- Manually compile the abstract syntax tree by passing a set of output tensors and input tensors to a **session.run()** call

```
g = tf.Graph()

with g.as_default():
  in_a = tf.placeholder(dtype=v1.float32, shape=(2))
  def forward(x):
    with tf.variable_scope("matmul", reuse=v1.AUTO_REUSE):
      W = tf.get_variable("W", initializer=v1.ones(shape=(2,2)),
                          regularizer=lambda x:tf.reduce_mean(x**2))
      b = tf.get_variable("b", initializer=v1.zeros(shape=(2)))
      return W * x + b
  out_a = forward(in_a)

with tf.Session(graph=g) as sess:
  sess.run(tf.global_variables_initializer())
  outs = sess.run(out_a,feed_dict={in_a: [1, 0]})
```

CODING IN TENSORFLOW 2.0 : EAGER MODE

- Write code the '**pythonic way**' using object and variables, with support for dynamic control.
- Instant execution of the computation without explicit run call.

```
W = tf.Variable(tf.ones(shape=(2,2)), name="W")
b = tf.Variable(tf.zeros(shape=(2)), name="b")

def forward(x):
  return W * x + b

out_a = forward([1,0])
```

LESS BOILERPLATE, SMALLER CODE!

FEATURES IN EAGER EXECUTION

Eager execution contains features which are either used to replicate the graph mode in eager execution mode, or offer advantages which are not in graph mode.

SOME OF THE KEY FEATURES ARE :

- Dynamic Control Flow
- Auto Differentiation (Eager Training)
- Dynamic Models
- Custom Gradients

DYNAMIC CONTROL FLOW

All the functionality of the host language is available while your model is executing. This allows to write code which looks native to the language while executing via Tensorflow.

```python
def fizzbuzz(max_num):
  counter = tf.constant(0)
  max_num = tf.convert_to_tensor(max_num)
  for num in range(1, max_num.numpy()+1):
    num = tf.constant(num)
    if int(num % 3) == 0 and int(num % 5) == 0:
      print('FizzBuzz')
    elif int(num % 3) == 0:
      print('Fizz')
    elif int(num % 5) == 0:
      print('Buzz')
    else:
      print(num.numpy())
    counter += 1
```

AUTOMATIC DIFFERENTIATION

Automatic differentiation, which is used in back propagation for training neural networks, is implemented by recording the forward pass in a 'tape' and playing the 'tape' backwards to compute the gradients. **tf.GradientTape** implements this functionality.

```
w = tf.Variable([[1.0]])
with tf.GradientTape() as tape:
  loss = w * w

grad = tape.gradient(loss, w)
print(grad)  # => tf.Tensor([[ 2.]], shape=(1, 1), dtype=float32)
```

NOTE : A particular **tf.GradientTape** can compute only one gradient, subsequent calls give runtime error.

DYNAMIC MODELS

Tensorflow supports automatic differentiation even for dynamic models. This allows flexibility in usage of the eagerly-coded function to use multiple models for same type of operation.

```
def line_search_step(fn, init_x, rate=1.0):

  with tf.GradientTape() as tape:
    tape.watch(init_x)
    value = fn(init_x)

  grad = tape.gradient(value, init_x)
  grad_norm = tf.reduce_sum(grad * grad)
  init_value = value - 1

  while value > init_value:
    x = init_x - rate * grad
    value = fn(x)
    rate /= 2.0

  return x, value
```

CUSTOM GRADIENTS

We can define custom gradients to override normal gradients for whatever function we define eagerly. It is commonly used to provide numerically stable gradient for a sequence of operations.

Normal Gradient

```
def log1pexp(x):
  return tf.math.log(1 + tf.exp(x))

def grad_log1pexp(x):
  with tf.GradientTape() as tape:
    tape.watch(x)
    value = log1pexp(x)
  return tape.gradient(value, x)





grad_log1pexp(tf.constant(100.)).numpy()

# Output : nan
```

Custom Gradient

```
@tf.custom_gradient
def log1pexp(x):
  e = tf.exp(x)
  def grad(dy):
    return dy * (1 - 1 / (1 + e))
  return tf.math.log(1 + e), grad

def grad_log1pexp(x):
  with tf.GradientTape() as tape:
    tape.watch(x)
    value = log1pexp(x)
  return tape.gradient(value, x)

grad_log1pexp(tf.constant(100.)).numpy()

# Output : 1.0
```

WHY GRAPHS THEN?

Even though eager execution makes Tensorflow easier , using the graph mode offers its own advantages which was what made Tensorflow extremely popular for production purposes.

Graph Mode provides the following advantages :

- Optimized deployment across devices and processors
- Easier distributed computing across multiple machines
- Specific graph-based compute optimizations
- Generally better perfomance

SO HOW DO WE COMBINE ADVANTAGES OF BOTH THE MODES ?

TF.FUNCTION AND AUTOGRAPH

Tensorflow provides an module called **tf.function** which can be used to convert eager functions into graphs. **AutoGraph** is a sub-library which converts Python flow control and loops (if,while) into Tensorflow ops (tf.cond, tf.while_loop).

To use the above features efficiently :

- Refactor code into smaller functions that are called as needed
- Decorate the higher level computations with **@tf.function**

CODE IN EAGER & EXECUTE IN GRAPH

EAGER TRAINING

Eager execution features which are useful for training model

- High-level **tf.keras** APIs to create layers and optimizer.
- Developing new layers and models using **tf.Layers** and **tf.Models** via subclassing.
- **tf.Variable** provides mutable **tf.Tensors** which provides easier automatic differentiation.
- Object-based model saving instead of saving using graphs.
- Object-oriented metrics for inferring the results of training steps.

DEMO

Notebook Link :
https://colab.research.google.com/drive/1vqY732X3mjT-CcbTYiW5FbE1PGahK1Fh?usp=sharing

Inside Tensorflow : Eager Execution Runtime by Alex Passos : https://youtu.be/qjx65mD6nrc

What I cannot create, I do not understand.

- Richard Feynman

Om Shri Prasath
ee17b113@smail.iitm.ac.in

SysDL Recitation
CS 6886