SeSE course: GPU programming for Machine Learning and Data Processing

Introduction to TensorFlow

Martin Kronbichler

Uppsala University

TensorFlow: A framework for deep learning

- Library from Google
- Based on creating the computational graph through a syntax rather similar to using `numpy` in Python
  - But not identical
- Working principle of TensorFlow: Given access to the full graph, can optimize memory transfers and evaluation order
- Lots of constructs specifically adapted to neural networks
  - Convolution operations, loss functions, optimizers
- But also a general evaluator of any computational graph
- Caveats:
  - TensorFlow may not always give optimal performance
  - Not every problem is reasonably expressed as computational graph (yet many are)

TensorFlow: What is a tensor?

- In practice, a TensorFlow `tensor` is an object similar to a numpy `ndarray`
- A tensor has a dimensionality and a type
- Even in numpy, not every array-like object is backed by an actual chunk of memory
    - When slicing or transposing an array, that only creates a view inside another array
- In TensorFlow, many operations are just that, if we write `C = A * B`, `C` represents the operation of multiplying `A` and `B`, in their current state
    - That operation might also be executed

TensorFlow: Constants & variables

- To insert a value (scalar or an array) into TensorFlow, use
  - `tf.constant`, e.g. `tf.constant(someData, dtype=tf.float32)`
- To create a large tensor of identical values, most efficient approach is to use `tf.fill`
  - Accepting a scalar and a shape
- To create optimizable variables, use `tf.Variable`
  - These are end targets for gradient computations and optimization algorithms
  - The actual content of a model

- Operations are elementwise per default similarly to `numpy`
- `A * B`, `A + B`, `A / B` all act elementwise
- `A @ B` for matrix multiplication
  - syntactic sugar since Python 3.5, implies `tf.matmul`
- `tf.stack` combines several tensors of rank R into one of rank $R+1$
- `tf.concat` concatenates several tensors of rank R into a new tensor of rank R with larger dimensions
- `tf.where` takes a boolean tensor choosing elements from the arguments, e.g. maximum by `tf.where(A>B, A, B)`
  - `tf.math.maximum(A, B)` is more convenient and probably more efficient
  - `tf.math.reduce_max(A)` instead computes the maximum within a tensor

- Almost all TensorFlow functions accept a name argument
- TensorFlow itself does not “see” user-given variable names, so error messages can sometimes refer to layer names
- Many functions can work on the full tensor, or just along some axis
    - Operation mode can be changed using the axis argument
    - `tf.reduce_sum` has a default axis of `None`, summing over all axes
    - `tf.concat` and `tf.stack` have defaults of `0`, indicating that the common/new axis should be the first one
    - Negative indices are allowed, just like in ordinary Python, to access indices starting from the end

- TensorFlow supports broadcasting similarly to numpy
  - If a tensor is size 1 in some dimension, that can implicitly be interpreted to match any other size for many operations
- Risk for hard-to-understand errors due to this implicit conversion
  - Multiplying a shape $N \times 1$ vector (N rows, 1 column) with a $1 \times N$ vector (1 row, N columns) creates a $N \times N$ matrix
- `tf.broadcast_to`, `tf.reshape` and `tf.expand_dims` can be useful when you need a bit more control
- Broadcasting is far more efficient than actually creating the corresponding tensor with repeated elements
  - In abstract setting, can avoid multiplication with repeated entries and mathematically "transform" results
- The same holds for `tf.tile`, for repeating a tensor multiple times

Eager execution

- Traditional workflow with TensorFlow: first create a graph and then ask TensorFlow to run to get the value of some specific tensor
  - Only viable usage mode in TensorFlow 1
    - Create the graph, then run it
- TensorFlow 2 supports eager execution, where the operations you do are also evaluated
  - Does not build graphs
  - TensorFlow runs in eager mode by default, check by `tf.executing_eagerly()`
- Much easier to debug this way...
  - This approach removes some optimization opportunities

- TensorFlow allows to decorate a function by `@tf.function`
- This will make TensorFlow analyze the full function and try to express it as TensorFlow graphs
- This gives room for optimization
- Sometimes even `for` loops and other “expensive” things in Python can be pushed into a compact graph that is executed all on the GPU
- Enclose well-contained logic in functions (good practice anyway), and tag them as `@tf.function`, unless need to debug them

TensorFlow: Pitfalls

- A tensor can stay on the GPU
  - Depends on the calling context and other dependencies
- TensorFlow can be very helpful in converting to and from numpy arrays
  - This will transfer the information between CPU and GPU
  - All gradient information will be lost when converting from a tensor to numpy, do something, and then transforming back
    - From TensorFlow's point of view, those are two unrelated constant tensors
    - Abstract tensor notation lost, transformed into numbers
- Even just doing an `if` on some value of a tensor outside of a `@tf.function` forces the full evaluation of the tensor and the transfer of that data from GPU to CPU
- It is so easy to do things with TensorFlow that the cost of certain operations gets hidden and thus non-obvious

Auto-differentiation with TensorFlow

- Backpropagation would be terrible to use if implemented manually
- TensorFlow provides automatic computation of gradients
  - Automatic for variables, but can compute the gradient also for a constant
- Consequence: Almost any TensorFlow operation is differentiable
  - But the gradient of e.g. a `max` operation is only related to the maximum value
    - Even if a variable is lower by only a factor $1 - 10^{-5}$, it gets zero gradients
  - This and other discrete operations typically avoided
  - Aim rather for mathematically smooth operations, like computing a norm, or taking the sum of the logged exponentials

TensorFlow: Code example

```
x = tf.constant(3.0)
with tf.GradientTape(persistent=True) as g:
  g.watch(x)
  y = x * x
  z = y * y
dz_dx = g.gradient(z, x) # 108.0 (4*x^3 at x = 3)
dy_dx = g.gradient(y, x) # 6.0
del g # Drop the reference to the tape
```

Keras: A higher-level abstraction for neural networks

- Keras is a higher level abstraction for building neural networks
- Goal: Fast experimentation with deep neural networks
- Has Python interface and interface to TensorFlow
  - In theory, Keras can be used with several backends (not only TensorFlow), but it can also just be used as a convenience layer
- Keras provides numerous implementations of neural-network building blocks:
  - Objectives, layers, activation functions, optimizers
  - Support for convolutional and recurrent neural networks
- Sometimes confusing to find Keras and TensorFlow features to do the same thing