AIDL: PROJECTS

Session 5 (2019/05/28): Deep dive into Tensorflow IV

Previously on *AIDL: Projects*...

Automatic differentiation: definition

“Automatic differentiation or computational differentiation, is a set of techniques to numerically evaluate the derivative of a function specified by a computer program. AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, derivatives of arbitrary order can be computed automatically, accurately to working precision, and using at most a small constant factor more arithmetic operations than the original program.”

Automatic differentiation: optimizers

Tensorflow provide a different number of optimizers (doc) that will compute the backprop for you and optimize your models. A few examples:

- SGD: tf.train.GradientDescentOptimizer
- Adam: tf.train.AdamOptimizer
- RMSProp: tf.train.RMSPropOptimizer

```
loss = ...
optimizer = tf.train.AdamOptimizer(lr=10e-4)
train_step = optimizer.minimize(loss)

sess.run(train_step, feed_dict={...})
```

Scopes

Graph scopes

Name scopes (tf.name_scope). Affects:

- tf.Operation
- tf.Variable

Variable scopes (tf.variable_scope). Affects:

- tf.Operation
- tf.Variable
- tf.get_variable

Exercise II

Scope your model

- Add scopes to the model and loss
- Visualize the graph in Tensorboard

How to get the new changes

```
# Get new exercises
> git checkout master
> git pull
> git checkout -b session_05

# Apply changes here
…

# Commit changes
> git commit -am “Whatever message you want”
```

Exercise III

Optimizers

- Add a TF optimizer instead of custom backprop
- Visualize the graph in Tensorboard

Feeding data

Feeding data: so far

- tf.constants: simple scalar o lightweight numeric values that define some part of the graph and do not change between runs
- tf.placeholder & feed_dict: inject python data to the tensorflow graph at each run

Is feed_dict efficient? Not that much when dealing with big datasets!!

- Difficult to parallelize
- Input pipeline outside of the graph

Can we do something better? Of course! → **tf.data**

Feeding data: tf.data

Benefits:

- Off-the-shelf parallelization and scaling
- Embedded in the graph
- Sooooo much better in terms of coding
- All the input manipulation in one place

Main classes:

- tf.data.Dataset: represents the subgraph with the dataset operations
- tf.data.Iterator: it’s able to iterate through the samples in the dataset and yield those values to the network

Feeding data: main ops

Dataset:

- from_generator, from_tensor_slices: create dataset from source data
- shuffle: exactly that
- repeat: how many times should the dataset be repeated (epochs!)
- map: transform the samples
- batch: group samples into batches
- prefetch: create a buffer of items for improved performance
- make_one_shot_iterator: create the iterator of this dataset

Iterator:

- get_next: obtain the reference to the next samples of the dataset

Feeding data: example

```
dataset = tf.data.Dataset.range(100)
    .shuffle(10)
    .repeat(num_epochs)
    .map(parse_fn, num_parallel_calls=10)
    .batch(batch_size)
    .prefetch(prefetch_batches)

iterator = dataset.make_one_shot_iterator()
batch = iterator.get_next()
x, y = batch     # Instead of the placeholders!!!
```