Tijmen Verhulsdonck

Follow 78 Followers About

# An Advanced Example of Tensorflow Estimators Part (3/3)

Note: this story used to be part of a series however all parts are now combined a single post.

Tijmen Verhulsdonck Jul 31, 2018 · 5 min read

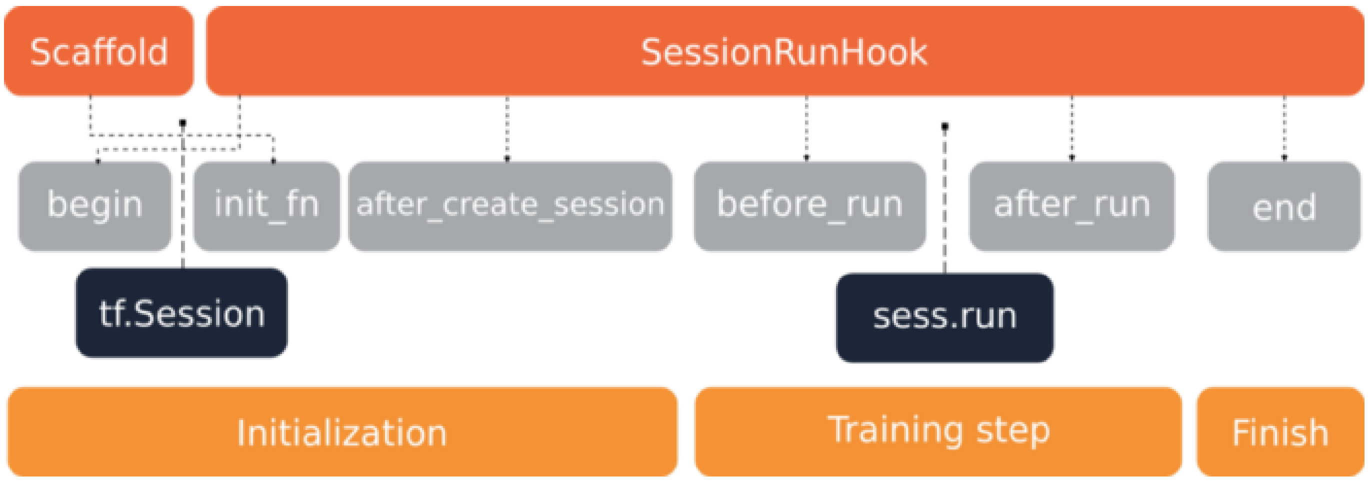
For the final part of the 3 part series (part 1, part 2) presenting an advanced usage example of the Tensorflow Estimator class, the "Scaffold" and "SessionRunHook" classes will be explained. Prediction will also be covered to finish up the complete picture of the Estimator class.

## `Switching to the Estimator class

In part 1 I explained the advantages of using the Estimator class, but switching to an Estimator from an existing code-base is not necessarily straightforward. As the Estimator class abstracts away most of the initialization and execution during training, any custom initialization and execution loops can no longer be an implementation using "*tf.Session*" and "*sess.run()*". This could be a reason for companies with a large code-base not to transition to the Estimator class, as there is no easy and straightforward transition path. While this article is not a transition guide, it will clarify some of the new procedures for initialization and execution. This will hopefully fill in some of the gaps left by the official tutorials, and give someone the confidence needed to switch to Estimators.

## Scaffolds and SessionRunHooks

initialization of a model and can only be used to construct an “EstimatorSpec” in training mode. The “SessionRunHook” on the other hand can be used to construct an “EstimatorSpec” for each mode and is used each time train, evaluate or predict is called. Both the “Scaffold” and “SessionRunHook” provide certain functions that the Estimator class calls during use. Below you can see a timeline showing which function is called and when during the initialization and training process. This also shows that the Estimator under the hood still uses “*tf.Session*” and “*sess.run*”.



Function execution timeline executed by the Estimator of the Scaffold and SessionRunHook classes.

**Scaffolds**

A scaffold can be passed as the scaffold argument when constructing the “EstimatorSpec” for training. In a scaffold you can specify a number of different operations to be called during various stages of initialization. However for now we will focus on the “*init_fn*” argument as the others are more for distributed settings which this series will not cover. The “*init_fn*” is called after graph finalization but before the “*sess.run*” is called for the first time, it is only used once and thus perfect for custom variable initialization. A good example of this is if you want to finetune using a pretrained network and want to be selective about which variables to restore.

```
# look for checkpoint
model_path = tf.train.latest_checkpoint(path)
initializer_fn = None

```

```
        # Create the saver which will be used to restore the variables.
        initializer_fn = slim.assign_from_checkpoint_fn(model_path, variables_to_restore)
    else:
        print("could not find the fine tune ckpt at {}".format(path))
        exit()

    def InitFn(scaffold,sess):
        initializer_fn(sess)
    return InitFn
```

fine_tune.py hosted with ♡ by GitHub view raw

From: https://github.com/Timen/squeezenext-tensorflow/blob/master/tools/fine_tune.py

Above you can see an example of this implemented, the function checks for the existence of the finetune checkpoint and creates an initializer function using slim that filters on the "*scope_name*" variable. This function is then encapsulated in the function format that the Estimator expects which consumes the "Scaffold" object itself and a "*tf.Session*" object. Inside the "*init_fn*" the session is used to run the "*initializer_fn*".

```
# setup fine tune scaffold
scaffold = tf.train.Scaffold(init_op=None,
     init_fn=tools.fine_tune.init_weights(params["fine_tune_ckpt"]))

# create estimator training spec
return tf.estimator.EstimatorSpec(tf.estimator.ModeKeys.TRAIN,
                                 loss=loss,
                                 train_op=train_op,scaffold=scaffold)
```

This "*init_fn*" can then be passed as an argument to construct a "Scaffold" object, as show above. This "Scaffold" object is used to construct the "EstimatorSpec" for the train mode. While this is a relatively simple example (and can also be achieved with "WarmStartSettings"), it can be easily expanded for more advanced onetime initialization functionality.

## SessionRunHooks

as hooking, I will not go into to much detail on what hooking is exactly as in this case it is pretty self explanatory. But using a “SessionRunHook” is slightly different from using a scaffold. Instead of initializing the “SessionRunHook” object with one or more functions as arguments it is used as a super class. It is then possible to overwrite the methods “*begin*”, “*after_create_session*”, “*before_run*”, “*after_run*” and “*end*”, to extend functionality. Below an excerpt showing how to overwrite the “*begin*” function is shown.

```

class ModelStats(tf.train.SessionRunHook):
    """Logs model stats to a csv."""

    def __init__(self, scope_name, path,batch_size):
        """
        Set class variables
        :param scope_name:
            Used to filter for tensors which name contain that specific variable scope
        :param path:
            path to model dir
        :param batch_size:
            batch size during training
        """
        self.scope_name = scope_name
        self.batch_size = batch_size
        self.path = path

    def begin(self):
        """
            Method to output statistics of the model to an easy to read csv, listing the multiply
            number of parameters, in the model dir.
        :param session:
            Tensorflow session
        :param coord:
            unused
        """
        # get graph and operations
        graph = tf.get_default_graph()
        operations = graph.get_operations()
        # setup dictionaries
        biases = defaultdict(lambda: None)
        stat_dict = defaultdict(lambda: {"params":0,"maccs":0,"adds":0, "comps":0})
```

From: https://github.com/Timen/squeezenext-tensorflow/blob/master/tools/stats.py

Keep in mind that the "*begin*" function does not get any arguments but each hook is slightly different, please refer here to check what arguments are passed to which function. Now one can create a "SessionRunHook" with the extended begin method by calling:

```
stats_hook = tools.stats.ModelStats("squeezenext",
                                    params["model_dir"],
                                    self.batch_size)
```

This object can then be used when constructing an "EstimatorSpec" for training, evaluating and predicting by passing it to the respective arguments "training_hooks", "evaluation_hooks" and "prediction_hooks". Note that each arguments ends with hooks, and they expect an iterable of one or multiple hooks so always encapsulate your hooks in an iterable.

## Prediction

Part 1 and 2 of this series have extensively covered training and evaluation using Estimators, but prediction has not yet been fully explained. This is mostly because prediction is the easiest of the the 3 (train,eval,predict) Estimator modes, but there are still some pitfalls one can run into. In part 2 it was explained how to setup an "EstimatorSpec" for prediction, but not how to use it. Below an example is shown how to actually use an Estimator for prediction.

```
my_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"image": image},
    shuffle=False,
    batch_size=1)

# setup synset lookup table for human readable labels
lookup_table = _build_synset_lookup(args.imagenet_metadata_file)
challenge_synsets = [l.strip() for l in
                     tf.gfile.FastGFile(args.labels_file, 'r').readlines()]

```

```python
  # Print top 5 results
  for result in predictions:
    print(result)
```

predict.py hosted with ♡ by GitHub view raw

From: https://github.com/Timen/squeezenext-tensorflow/blob/master/predict.py

For prediction it does not make sense to make tfrecords, so instead the "*numpy_input_fn*" is used. To the "*x*" argument a dictionary of numpy arrays is passed containing the features, in this case an image.

> *Note: Depending on whether or not you preprocess images in the* "input_fn" *during training, you would need to perform the same preprocessing here,either in numpy before passing to the "numpy_input_fn" or in Tensorflow.*

With the numpy input function setup, a simple call to predict is enough to create a prediction. The "*predictions*" variable in the example above will not actually contain the result yet, instead it is an object of the generator class, to get the actual result you need to iterate over the it. This iteration will start the Tensorflow execution and produce the actual result.

## Final Remarks

In this final part the use of the "Scaffold" and "SessionRunHook" to influence the initialization and execution loop was explained. A short example of a prediction was also shown, which concludes this 3 part series. I hope it gave some more information and insight in the use of the Estimator class for a custom and slightly advanced application. All the code comes from this github repo, and please leave a comment if you have any further questions I'd be happy to help. I plan to write another series of using video data for training, but that will take a little longer. Stay Tuned!

TensorFlow  Machine Learning  Scaffolding  Predictions  Software Engineering