ML Design Pattern #4: Keyed Predictions

Export your model so that it passes through client keys

Lak Lakshmanan Oct 27, 2019 · 4 min read

An occasional series of design patterns for ML engineers. Full list here.

Normally, you train your model on the same set of inputs that the model will be supplied in real-time. In many situations, however, it can be advantageous for your model to also pass through a client-supplied key.

Client-supplied keys are essential for high-performance serving infrastructure. Photo by Florian Berger on Unsplash

Hot spots in serving infrastructure

If your model is deployed as a web service and accepts a single input, then it is quite clear which output corresponds to which input. What if the model accepts an array of inputs? You might think that it should be obvious that the first output instance corresponds to the first input instance, the second output instance to the second input instance, etc. However, such a 1:1 relationship will cause significant hot spots in your serving infrastructure since whichever server node is unlucky enough to receive a particularly large request will need to compute the output for all the instances in the array. These hot spots will force that you to make your server machines more powerful than they need to be.

It would, of course, be much more efficient if the receiving node can farm out the instances to a cluster, collect all the resulting outputs, and send them back. But if it does that, then the outputs are going to be jumbled up. How can the client figure out which output corresponds to which input?

Pass-through keys

Solution? Have the client supply a key associated with each input. For example, suppose your model is trained with three inputs (a, b, c), shown in green, to produce the output d, shown in yellow. Make your clients supply (k, a, b, c) to your model where k is a *key*. The key could be as simple as numbering the input instances 1, 2, 3, …, etc. Your model will then return (k, d) and so the client will be able to figure out which output instance corresponds to which input instance.

Clients supply a key with each input instance, so that they can disambiguate the outputs

This is exactly the same situation that you will run into when you do batch predictions, so keys are useful in that context as well.

Why should the keys be supplied by clients?

Wait a second … why can't the server just assign keys to the inputs it receives before it invokes the model, and then remove the keys before sending along the outputs? Why ask clients to specify a key?

This is because there are a couple of other situations where keys are useful — asynchronous serving and evaluation. Many production machine learning models these days are neural networks, and neural networks involve matrix multiplications. Matrix multiplication on hardware like GPUs and TPUs is more efficient if you can ensure that the matrices are within certain size ranges and/or multiples of a certain number. It can, therefore, be helpful to accumulate requests (up to a maximum latency of course) and handle the incoming requests in chunks. Since the chunks will consist of interleaved requests from multiple clients, the key, in this case, needs to have some sort of client identifier as well.

If you are doing continuous evaluation, it can be helpful to log metadata about the prediction requests, so that you can monitor whether performance drops across the board, or only in specific situations. Such slicing is made much easier if the key identifies the situation in question.

Because high-performance servers will support both multiple clients, be backed by a cluster, and batch up requests to gain performance benefits, it's better to plan ahead for this — ask clients supply keys with every prediction and for clients to specify keys that will not cause a collision with other clients.

How to pass through keys in Keras

In order to get your Keras model to pass through keys, supply a serving signature when exporting the model.

For example, this is the code to take a model that would otherwise take 4 inputs (is_male, mother_age, plurality, and gestation_weeks) and have it also take a key that

it will pass through to the output along with the original output of the model (the babyweight):

```
# Serving function that passes through keys
@tf.function(input_signature=[{
      'is_male': tf.TensorSpec([None,], dtype=tf.string,
name='is_male'),
      'mother_age': tf.TensorSpec([None,], dtype=tf.float32,
name='mother_age'),
      'plurality': tf.TensorSpec([None,], dtype=tf.string,
name='plurality'),
      'gestation_weeks': tf.TensorSpec([None,], dtype=tf.float32,
name='gestation_weeks'),
      'key': tf.TensorSpec([None,], dtype=tf.string, name='key')
}])
def my_serve(inputs):
    feats = inputs.copy()
    key = feats.pop('key')
    output = model(feats)
    return {'key': key, 'babyweight': output}

tf.saved_model.save(model, EXPORT_PATH,
                    signatures={'serving_default': my_serve})
```

Note that the code above works even if the original model was not saved with a serving function. Simply load the model using tf.saved_model.load(), attach a serving function and use the code snippet above!

Load a SavedModel, attach a non-default serving function and save it.

Enjoy!

Machine Learning Keras TensorFlow Serving Design Patterns