Vitaly Bezgachev Follow
Senior Software Engineer @SAP Health, Artificial Intelligence learner
Jun 24, 2017 · 6 min read

How to deploy Machine Learning models with TensorFlow. Part 1—make your model ready for serving.

. . .

- After finishing the Deep Learning Foundation course at Udacity I had a big question—how did I deploy the trained model and make predictions for new data samples? Fortunately, TensorFlow was developed for production and it provides a solution for model deployment—TensorFlow Serving. Basically there are three steps—export your model for serving, create a Docker container with your model and deploy it with Kubernetes into a cloud platform, i.e. Google Cloud or Amazon AWS. In this article I concentrate on the first part—export the model for serving.

Motivation and tooling

During my course at Udacity I was always asking myself—I have my model, which I can run in Jupyter Notebook and see the result, but what can I do with it? How can other use it?

As a software engineer I’m interested in the pipeline: create the model -> test it locally -> create web service to serve the client request -> create deployment container with my web service inside -> test the container -> put it into production. The answers to my questions are:

> TensorFlow for model creation
>
> Docker for containerization
>
> TensorFlow Serving for a model hosting
>
> Kubernetes for production deployment

TensorFlow

TensorFlow is an open-source library for development of Machine Learning and especially Deep Learning models. It is created and supported by Google and targets not only academia but also product development.

Docker

Docker is a very popular containerization engine and provides a convenient way to pack you stuff with all dependencies together to be deployed locally or in the cloud. The documentation is very comprehensive and I encourage you to check it for the details.

TensorFlow Serving

TensorFlow Serving, as it name points, hosts the model and provides remote access to it. TensorFlow Serving has a good documentation on its architecture and useful tutorials. Unfortunately they are using prepared examples and get a little explanation, what you need to do for your own models to be served.

Kubernetes

Kubernetes is an open-source software created, again, at Google and it provides container orchestration, allowing you automated horizontal scaling, service discovery, load balancing and much more. In simple words — it automates the management of your web services in the cloud.

Intention

As an example I take a GAN model for Semi-Supervised learning, which is taught at Udacity Deep Learning Foundations course. My intention is:

- Train the GAN model for semi-supervised learning on Street View House Number data set
- Use GAN discriminator to make prediction of house numbers. As an output I want to have 10 scores corresponding to numbers from 0 to 9.
- Let TensorFlow serve my model in a Docker container
- Create a client to request the scores for number images
- Deploy the model into a cloud

Model preparation

Based on the Jupyter Notebook, I put functionalities into separated Python files, tested the saved model, implemented the model export and the client for service requests. In general, the base code remains the same. You can find implementation details in my GitHub repository.

Main steps are:

- Train the model saving the checkpoints on the disk
- Load saved model and test that it works properly
- Export model into Protobuf format (details below)
- Create the client to issue requests (details in the next part)

First two steps are pretty easy for anyone who is creating Deep Learning models with TensorFlow and I do not want to go into details here. But the last two steps were pretty new for me and I spent some time to understand how it works and what is required.

TensorFlow Serving. What is it?

TensorFlow Serving implements a server that runs Machine Learning models and provides remote access to them. The common tasks are prediction and classification of provided data (e.g. images).

A couple of technical highlights:

> The server implements GRPC interface, so you cannot issue requests from your browser. Instead we need to create a client, which can communicate over GRPC
>
> TensorFlow Serving already provides operations on models stored as Protobuf
>
> You can create your own implementation to work with models stored in other formats

I didn't create my own implementation, so I needed to exported my model into Protobuf.

Protobuf

Protocol Buffers (or Protobuf) allows efficient data serialization. It is an open-source piece of a software and has been developed ……, right, by Google :-)

Export the model into Protobuf

TensorFlow Serving provides SavedModelBuild class to save the model as Protobuf. It is pretty good described here.

My GAN model accepts image tensor of a shape *[batch_num, width, height, channels]* where number of batches is 1 for serving (you can predict only one image at time), width and height are 32 pixels and number of image channels is 3. Input images must be scaled so that each pixel is in a range of [-1, 1] and not in a range of [0, 255].

From the other side the served model must accept JPEG image as an input, so for serving I needed to inject layers to transform a JPEG into required image tensor.

First, I implemented image transformation. It was a bit tricky for me.

```
serialized_tf_example = tf.placeholder(
                            tf.string, name='input_image')

feature_configs = { 'image/encoded': tf.FixedLenFeature(
                                         shape=[],
                                         dtype=tf.string), }

tf_example = tf.parse_example(serialized_tf_example,
feature_configs)

jpegs = tf_example['image/encoded']

images = tf.map_fn(preprocess_image, jpegs,
dtype=tf.float32)

images = tf.squeeze(images, [0])
# now the image shape is (1, ?, ?, 3)
```

Basically, you need a placeholder for a serialized incoming image, a feature configuration (dictionary name-to-feature) where you list expected inputs (*image/encoded* for JPEG in my case) and feature type. Then you parse serialized example and extract JPEG from it. And the last step is to transform a JPEG into desired image tensor. See my GitHub for a implementation details (*preprocess_image* method).

Then I can use that image tensor as an input for my GAN model, create a session object and load saved checkpoints.

```
......
net = GAN(images, z_size, learning_rate, drop_rate=0.)
......

saver = tf.train.Saver()

with tf.Session() as sess:
    # Restore the model from last checkpoints
    ckpt =
tf.train.get_checkpoint_state(FLAGS.checkpoint_dir)
    saver.restore(sess, ckpt.model_checkpoint_path)
```

```
......
```

Next challenge for me was to understand, how to transform restored model into Protobuf with provided SavedModelBuilder.

```
builder =
tf.saved_model.builder.SavedModelBuilder(export_path)
```

You have to create so-called signature with input, output and method name (e.g. classification or prediction). TensorFlow provides a method *tf.saved_model.utils.build_tensor_info* to create the tensor information. I use it to define input and output (scores in my case).

```
predict_tensor_inputs_info = \
    tf.saved_model.utils.build_tensor_info(jpegs)

predict_tensor_scores_info = \

tf.saved_model.utils.build_tensor_info(net.discriminator_out
)
```

And now I'm ready to create the signature.

```
prediction_signature = (
    tf.saved_model.signature_def_utils.build_signature_def(
        inputs={'images': predict_tensor_inputs_info},
        outputs={'scores': predict_tensor_scores_info},
        method_name=\

tf.saved_model.signature_constants.PREDICT_METHOD_NAME)
```

'images' and *'scores'* are predefined names and you must use them in input and output dictionaries.

In the tutorial TensorFlow team create two signatures—one for classification and one for prediction. I do not want any classification results, so prediction signature is enough for me.

The last step—save the model.

```
legacy_init_op = tf.group(tf.tables_initializer(),
                          name='legacy_init_op')

builder.add_meta_graph_and_variables(
    sess,
    [tf.saved_model.tag_constants.SERVING],
    signature_def_map={ 'predict_images':
prediction_signature },
    legacy_init_op=legacy_init_op)

builder.save()
```

It is pretty straightforward and now you have your model stored as Protobuf. The structure of the export folder should be something like:

- *variables* folder with *variables.data-xxx-of-yyy* and *variables.index*
- *saved_model.pb* file

The first part of the work is done—the model is exported as Protobuf successfully.

Put it all together

Environment

I developed and tested in the following environment:

- GPU-powered PC (NVidia GeForce GTX 1060 6 GB)
- Ubuntu 16.04
- Anaconda 4.3.14

Python 3.5

TensorFlow 1.1, GPU build. **Caution**: I had problems with TensorFlow 1.2, so I went back to a previous version

Try it yourself

Here are the steps that you need to execute to try it yourself.

Clone the sources

```
cd ~

git clone https://github.com/Vetal1977
/tf_serving_example.git

cd tf_serving_example
```

Train the model

```
python svnh_semi_supervised_model_train.py
```

It takes about 5–10 minutes to download train and test Street View House Numbers data set and another ca. 20 minutes to train the model (in my environment).

Check that you save the model

```
ls ./checkpoints
```

You should see data, index and metadata files.

Export model into Protobuf to be served by TensorFlow

```
python svnh_semi_supervised_model_saved.py --checkpoint-
dir=./checkpoints --output_dir=./gan-export --model-
version=1
```

The following should be printed out

```
Successfully exported GAN model version '1' into './gan-
export'
```

If you type

```
ls ./gan-export/1
```

You should get *variables* folder and *saved_model.pb* file.

Instead of conclusion

Probably it sounds simple, but I required a couple of hours to understand, how TensorFlow serves models. What inputs and outputs do I have. How do I register them to let TensorFlow know, what and how to serve.

In the next part I'll create a Docker container, put my model into it and create a client to make requests.