Keras predict not returning inside celery task

Asked 3 years, 11 months ago   Active 1 year, 6 months ago   Viewed 2k times

7

4

Following Keras function (predict) works when called synchronously

```
pred = model.predict(x)
```

But it does not work when called from within an asynchronous task queue (Celery). Keras predict function does not return any output when called asynchronously.

The stack is: Django, Celery, Redis, Keras, TensorFlow

django   tensorflow   redis   celery   keras

Share Follow

asked Aug 2 '17 at 11:09
pX0r
1,117 9 12

It turns out that - swapping out the TensorFlow backend with Theano, strangely but nicely, does not have this issue and returns the output from within an async task queue. Nevertheless, it would still be great to have a solution applicable to TensorFlow backend. – pX0r Aug 2 '17 at 12:15

Does it run when CELERY_ALWAYS_EAGER = True? Have a good look at the celery set up to check if it is configured correctly, then set a task that can indicate a running process to you (send an email, touch a file). Remember Celery runs the django application independently to that served by your web server, and Celery needs to be restarted whenever you update anything. – MagicLAMP Aug 2 '17 at 12:17

It doesn't work with CELERY_ALWAYS_EAGER to True either. – pX0r Aug 2 '17 at 12:35

stackoverflow.com/questions/47295025/… any suggesions – Dexter Nov 15 '17 at 13:39

Like I mentioned, consider using Theano instead, if possible. – pX0r Nov 16 '17 at 14:30

2 Answers

Active Oldest Votes

12

I ran into this exact same issue, and man was it a rabbit hole. Wanted to post my solution here since it might save somebody a day of work:

TensorFlow Thread-Specific Data Structures

In TensorFlow, there are two key data structures that are working behind the scenes when you call `model.predict` (or `keras.models.load_model`, or `keras.backend.clear_session`, or pretty much any other function interacting with the TensorFlow backend):

- A TensorFlow graph, which represents the structure of your Keras model
- A TensorFlow session, which is the connection between your current graph and the TensorFlow runtime

Something that is not explicitly clear in the docs without some digging is that *both the session and the graph are properties of the current thread*. See API docs here and here.

Using TensorFlow Models in Different Threads

It's natural to want to load your model once and then call `.predict()` on it multiple times later:

```
from keras.models import load_model

MY_MODEL = load_model('path/to/model/file')

def some_worker_function(inputs):
    return MY_MODEL.predict(inputs)
```

In a webserver or worker pool context like Celery, what this means is that you will load the model when you import the module containing the `load_model` line, then a different thread will execute `some_worker_function`, running predict on the global variable containing the Keras model. However, trying to run predict on a model loaded in a different thread produces "tensor is not an element of this graph" errors. Thanks to the several SO posts that touched on this topic, such as ValueError: Tensor Tensor(...) is not an element of this graph. When using global variable keras model. In order to get this to work, you need to hang on to the TensorFlow graph that was used-- as we saw earlier, the graph is a property of the current thread. The updated code looks like this:

```
from keras.models import load_model
import tensorflow as tf
```

Join Stack Overflow to learn, share knowledge, and build your career.

Sign up with email | Sign up with Google | Sign up with GitHub | Sign up with Facebook

Why does this only work on `Thread` s?

In Python, `Thread` s share the same global execution context as the parent process. From the [Python _thread docs](https://docs.python.org/3/library/_thread.html):

> This module provides low-level primitives for working with multiple threads (also called light-weight processes or tasks) — multiple threads of control sharing their global data space.

Because threads are not actual separate processes, they use the same python interpreter and thus are subject to the infamous Global Interpeter Lock (GIL). Perhaps more importantly for this investigation, they **share** global data space with the parent.

In contrast to this, `Process` es are *actual* new processes spawned by the program. This means:

- New Python interpreter instance (and no GIL)
- Global address space is **duplicated**

Note the difference here. While `Thread` s have access to a shared single global Session variable (stored internally in the `tensorflow_backend` module of Keras), `Process` es have duplicates of the Session variable.

My best understanding of this issue is that the Session variable is supposed to represent a unique connection between a client (process) and the TensorFlow runtime, but by being duplicated in the forking process, this connection information is not properly adjusted. This causes TensorFlow to hang when trying to use a Session created in a different process. If anybody has more insight into how this is working under the hood in TensorFlow, I would love to hear it!

The Solution / Workaround

I went with adjusting Celery so that it uses `Thread` s instead of `Process` es for pooling. There are some disadvantages to this approach (see GIL comment above), but this allows us to load the model only once. We aren't really CPU bound anyways since the TensorFlow runtime maxes out all the CPU cores (it can sidestep the GIL since it is not written in Python). You have to supply Celery with a separate library to do thread-based pooling; the docs suggest two options: `gevent` or `eventlet` . You then pass the library you choose into the worker via the `--pool` command line argument.

384 2 7

3

I got the reference from this Blog

- Tensorflow is Thread-Specific data structure that are working behind the scenes when you call **model.predict**

```
GRAPH = tf.get_default_graph()
with GRAPH.as_default():
    pred = model.predict
return pred
```

But Celery uses processes to manage all its worker pools. So at this point, things are still not working on Celery for that you need to use gevent or eventlet library

pip install gevent

now run celery as :

celery -A mysite worker **--pool gevent** -l info

Share Follow

answered Dec 19 '19 at 13:10
Deepak
861 10 10

1 Just-released Celery 4.4.0 adds "thread" concurrency to the list. I suppose that would work too. – DejanLekic Dec 19 '19 at 14:25

@DejanLekic using the same 4.4.0, but it didn't worked ! – Deepak Feb 5 '20 at 12:01