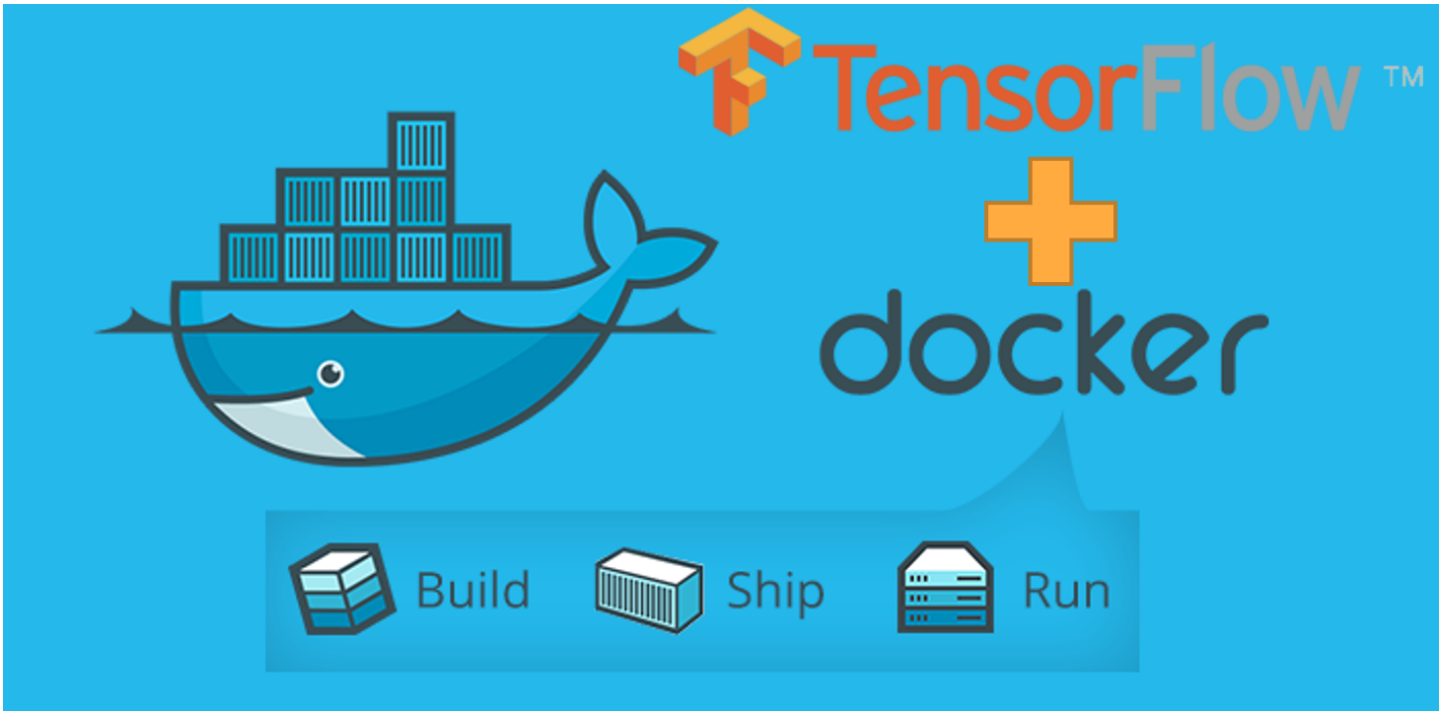
Rahul Kumar Follow
I'm a DeepLearning Enthusiast, an Independent Researcher and Technology Explorer. Chief AI Scientist @ BotSupply.ai | Jatana.ai
Aug 30, 2017 · 4 min read

Tensorflow + Docker = Production ready AI product



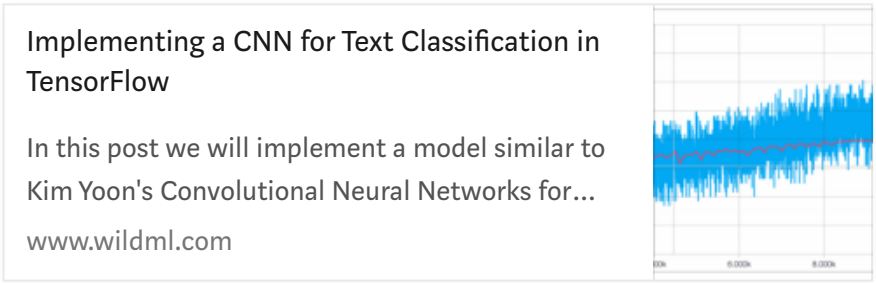
Everyone is talking about training the Deep Learning models and fine tuning them but very few talks about the deployment and the scalability aspects.
In *BotSupply*, we focus not only on building accurate Machine Learning models, but also on delivering them to the clients with the greater efficiency.

As well said by Giovanni Toschi :

> *"Delivery and consistency make the difference, on top of the intrinsic quality of the solution"—Giovanni Toschi, COO BotSupply*

In this article, we will learn to deploy a sentiment analysis model trained on "Character-level Convolutional Networks for Text Classification" (Xiang Zhang, Junbo Zhao, Yann LeCun) which uses character-level ConvNet networks for text classification.

Mostly reused code from https://github.com/dennybritz/cnn-text-classification-tf which was posted by Denny Britz. Check out his great blog post on CNN classification.



As explained in the above blog about the training process, I am pre-assuming that you have already trained your sentiment analysis model.

Overview

- Creating inference of sentiment analysis model
- Creating RESTful API of the model using Web.py
- Packaging your model into Docker containers

Prereqs:

- Basic machine learning understanding
- Basic RESTful understanding
- Basic understanding of Docker (https://www.docker.com/get-docker)

Tensorflow

https://www.tensorflow.org

Prediction Inference

Now we will create an inference to load the sentiment analysis model with Tensorflow which holds the models pre-trained weights.

> *The important thing to remember while creating an inference of the model is to load the exact graph which we created during the training process.*

To accomplish this we load the model components such as placeholders by calling

```
tf.get_default_graph().get_operation_by_name()
```

Once we have all the graph nodes ready in memory we'll create a prediction function *engine()* with accepts few parameters as defined below.

```
def engine(query = '' , senderid = '', senderName = '')
```

Few pre-processing is performed on the given *query* to transform it into vectors which will be passed into the graph nodes as input to make sentiment predictions.

Here is the complete code gist for the inference.

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
"""
@author: Rahul.kumar
"""

import data_helpers2 as data_helpers
import numpy as np
import os
import tensorflow as tf
import pandas as pd
from datetime import datetime

# Parameters
# ==================================================

# Eval Parameters
batch_size= 64
path ='/root/w2v_cnn_sentiment/'
checkpoint_dir = "./runs/1487512553/checkpoints"
vocab_file =  "./vocab1487512553.json"

# Misc Parameters
allow_soft_placement = True
log_device_placement =  False



# Load data. Load your own data here
print("Loading data...")
x_test, y_test, vocabulary, vocabulary_inv = data_hel
eval=True, vocab_file=vocab_file,cat1= "./data/sentim

# Evaluation
# ==================================================
checkpoint_file = "./runs/1487512553/checkpoints/mode
print("checkpoint file: {}".format(checkpoint_file))

session_conf = tf.ConfigProto(
    allow_soft_placement=allow_soft_placement,
    log_device_placement=log_device_placement)
sess = tf.Session(config=session_conf)
#    with sess.as_default():

# Load the saved meta graph and restore variables
```

```
45  # Load the saved meta graph and restore variables
46  saver = tf.train.import_meta_graph("{}.meta".format(c
47  saver.restore(sess, checkpoint_file)
48
49  # Get the placeholders from the graph by name
50  input_x = tf.get_default_graph().get_operation_by_nam
51  dropout_keep_prob = tf.get_default_graph().get_operat
52
53  # Tensors we want to evaluate
54  predictions = tf.get_default_graph().get_operation_by
55
56
57  def engine(query = " " , senderid = '', senderName =
58
59      new_question = query.strip()
60      new_question = data_helpers.clean_str(new_questio
61      new_question = new_question.split(" ")
62
63      num_padd = x_test.shape[1] - len(new_question)
64      new_question = new_question + ["<PAD/>"] * num_pa
```

Serving Predictions

With the model loaded, create a web server that can accept strings using webpy.

```
# -*- coding: utf-8 -*-
"""
@author: Rahul.kumar
"""

import web
#import sys,os
import json
import sentiment_model as model

urls = ('/sentiment', 'Sentiment')
app = web.application(urls, globals())


class Sentiment:
    def GET(self):
        web.header('Content-Type', 'application/json')
        user_data = web.input()
        comment= user_data.message
#         senderid= user data.senderid
```

We now have sentiment analysis as a service! Let's test it out locally before deploying. First, start the server. Then make a GET request to the prediction service.

```
$ python sentiment_api.py

(in another terminal window)
```

```
$ curl -X GET \
  'http://localhost:8080/sentiment?message=i really like
this product' \
  -H 'cache-control: no-cache' \
  -H 'postman-token: f9870fd3-90c6-2a5b-fad6-55cbcb8a7398'
```

If everthing is working correctly, you'll see a response back

```
{
 "Name": "Rahul",
 "Sentiment": "Positive",
 "Response": "Thank you for your positive response."
}
```

Deployment

We've got the model created and generating predictions. Time to deploy the model and package it into the docker image. Below, is a dockerfile file to set up and serve the prediction api.

```
FROM continuumio/anaconda:4.4.0
MAINTAINER Rahul Kumar, www.hellorahulk.com
COPY sentiment/ /usr/local/python/
EXPOSE 8180
WORKDIR /usr/local/python/
RUN pip install -r requirements.txt
CMD python sentiment_api.py 8180
```

Save this file named as `Dockerfile` as shown below.

```
|____Dockerfile
|____sentiment
| |____sentiment_api.py
| |____sentiment_model.py
| |____requirements.txt
```

Final Steps

Once we are all setup with the code, its time to test the end-to-end package. To do so follow the below steps:

1. Install Docker
2. Open docker terminal and navigate to `/path/to/tf-sentiment-docker`
3. Run `docker build -t sentiment-api .`
4. Run `docker run -p 8180:8180 sentiment-api`
5. Access `http://0.0.0.0:8180/sentiment?message=i love it` from your browser.

Conclusion

Voila !!! We just created a highly efficient, low latency and easy to distribute AI product which can run on any operating system.

I have created a GitHub repository with all the above mentioned steps.

Future Work

In my future post, I will explain how to create a flexible, high-performance serving system for machine learning models using **TensorFlow Serving API's** (https://www.tensorflow.org/serving).

. . .

Please feel free to add any suggestions or any problems that you faced while working on the about steps.