deepforge Documentation

Brian Broll

Apr 09, 2020

Getting Started

CHAPTER 1

Getting Started

1.1 What is DeepForge?

Deep learning is a promising, yet complex, area of machine learning. This complexity can both create a barrier to entry for those wanting to get involved in deep learning as well as slow the development of those already comfortable in deep learning.

DeepForge is a development environment for deep learning focused on alleviating these problems. Leveraging principles from Model-Driven Engineering, DeepForge is able to reduce the complexity of using deep learning while providing an opportunity for integrating with other domain specific modeling environments created with WebGME.

1.2 Design Goals

As mentioned above, DeepForge focuses on two main goals:

1. **Improving the efficiency** of experienced data scientists/researchers in deep learning
2. **Lowering the barrier to entry** for newcomers to deep learning

It is important to highlight that although one of the goals is focused on lowering the barrier to entry, DeepForge is intended to be more than simply an educational tool; that is, it is important not to compromise on flexibility and effectiveness as a research/industry tool in order to provide an easier experience for beginners (that's what forks are for!).

1.3 Overview and Features

DeepForge provides a collaborative, distributed development environment for deep learning. The development environment is a hybrid visual and textual programming environment. Higher levels of abstraction, such as creating architectures, use visual environments to capture the overall structure of the task while lower levels of abstraction, such as defining custom training functions, utilize text environments. DeepForge contains both a pipelining language

and editor for defining the high level operations to perform while training or evaluating your models as well as a language for defining neural networks (through installing a DeepForge extension such as DeepForge-Keras).

1.3.1 Concepts and Terminology

- *Operation* - essentially a function written in Python (such as training a model, visualizing results, etc)
- *Pipeline* - directed acyclic graph composed of operations - e.g., a training pipeline may retrieve and normalize data, train an architecture and return the trained model
- *Execution* - when a pipeline is run, an "execution" is created and reports the status of each operation as it is run (distributed over a number of worker machines)
- *Artifact* - an artifact represents some data (either user uploaded or created during an execution)
- *Resource* - a domain specific model (provided by a DeepForge extension) to be used by a pipeline such as a neural network architecture

CHAPTER 2

Quick Start

There are two ways to give DeepForge a try: visit the public deployment at https://editor.deepforge.org, or spin up your own deployment locally.

2.1 Connecting to the Public Deployment

As of this writing, registration is not yet open to the public and is only available upon request.

After getting an account for https://editor.deepforge.org, the only thing required to get up and running with DeepForge is to determine the compute and storage adapters to use. If you already have an account with one of the existing integrations, then you should be able to use those without any further setup!

If not, the easiest way to get started is to connect your own desktop to use for compute and to use the S3 adapter to storage data and trained model weights. Connect your own desktop for computation using the following command (using docker):

```
docker run -it deepforge/worker:latest --host https://dev.deepforge.org -t <access␣
↪token>
```

where <*access token*> is an access token for your user (created from the profile page of https://editor.deepforge.org).

After connecting a machine to use for computation, you can start creating and running pipelines w/o input or output operations! To save artifacts in DeepForge, you will need to connect a storage adapter such as the S3 adapter.

To easily create a custom storage location, minio is recommended. Simply spin up an instance of minio on a machine publicly accessible from the internet. Providing the public IP address of the machine (along with any configured credentials) to DeepForge when executing a pipeline will enable you to save any generated artifacts, such as trained model weights, to the minio instance and register it within DeepForge.

CHAPTER 3

Custom Operations

In this document we will outline the basics of custom operations including the operation editor and operation feedback utilities.

3.1 The Basics

Operations are used in pipelines and have named inputs and outputs. When creating a pipeline, if you don't currently find an operation for the given task, you can easily create your own by selecting the *New Operation...* operation from the add operation dialog. This will create a new operation definition and open it in the operation editor. The operation editor has two main parts, the interface editor and the implementation editor.

The interface editor is provided on the right and presents the interface as a diagram showing the input data and output data as objects flowing into or out of the given operation. Selecting the operation node in the operation interface editor will expand the node and allow the user to add or edit attributes for the given operation. These attributes are exposed when using this operation in a pipeline and can be set at design time - that is, these are set when creating the given pipeline. The interface diagram may also contain light blue nodes flowing into the operation. These nodes represent "references" that the operation accepts as input before running. When using the operation, references will appear alongside the attributes but will allow the user to select from a list of all possible targets when clicked.

The operation editor also provides an interface to specify operation python dependencies. DeepForge uses `conda` to manage python dependencies for an operation. This pairs well with the integration of various compute platforms that available to the user and the only requirement for a user is to have Conda installed in their computing platform. You can specify operation dependencies using a conda environment file as shown in the diagram below:

To the left of the operation editor is the implementation editor. The implementation editor is a code editor specially tailored for programming the implementations of operations in DeepForge. It also is synchronized with the interface editor. A section of the implementation is shown below:

```
import numpy as np
from sklearn.model_selection import train_test_split
import keras
import time
from matplotlib import pyplot as plt
```

(continues on next page)

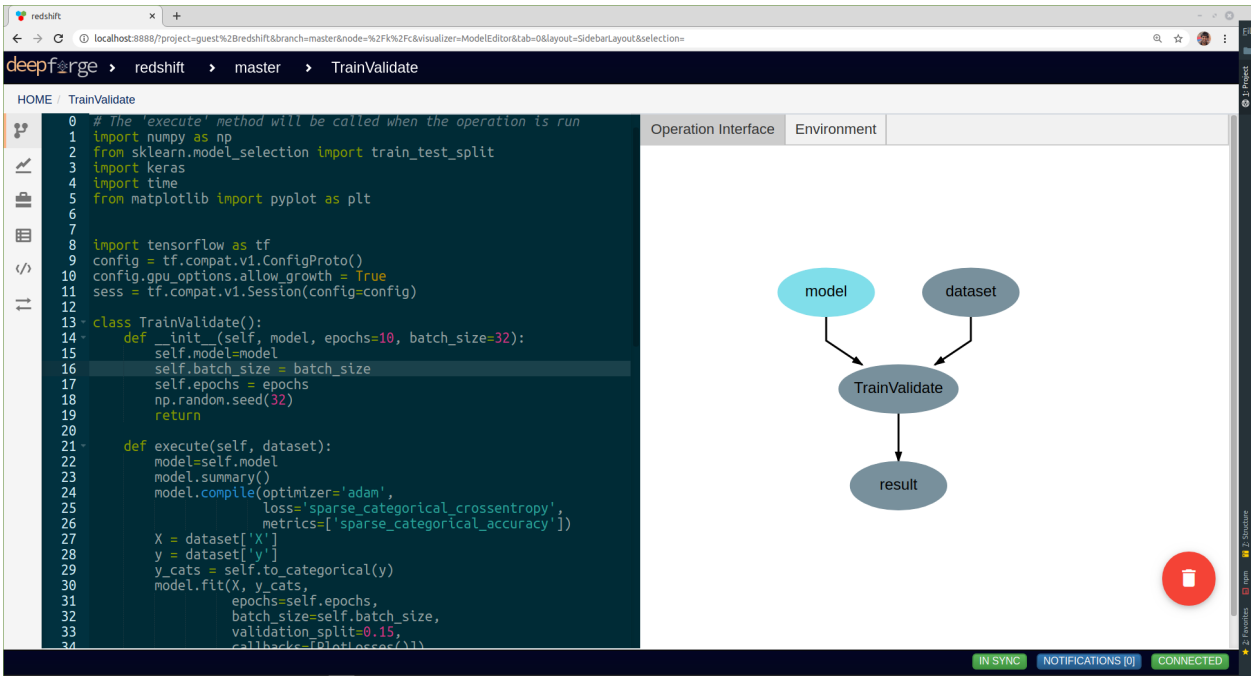

Fig. 1: Editing the "TrainValidate" operation from the "redshift" example

(continued from previous page)

```
import tensorflow as tf

import tensorflow as tf
config = tf.compat.v1.ConfigProto()
config.gpu_options.allow_growth = True
sess = tf.compat.v1.Session(config=config)

class TrainValidate():
    def __init__(self, model, epochs=10, batch_size=32):
        self.model=model
        self.batch_size = batch_size
        self.epochs = epochs
        np.random.seed(32)
        return

    def execute(self, dataset):
        model=self.model
        model.summary()
        model.compile(optimizer='adam',
                      loss='sparse_categorical_crossentropy',
                      metrics=['sparse_categorical_accuracy'])
        X = dataset['X']
        y = dataset['y']
        y_cats = self.to_categorical(y)
        model.fit(X, y_cats,
                  epochs=self.epochs,
                  batch_size=self.batch_size,
                  validation_split=0.15,
                  callbacks=[PlotLosses()])
```

(continues on next page)

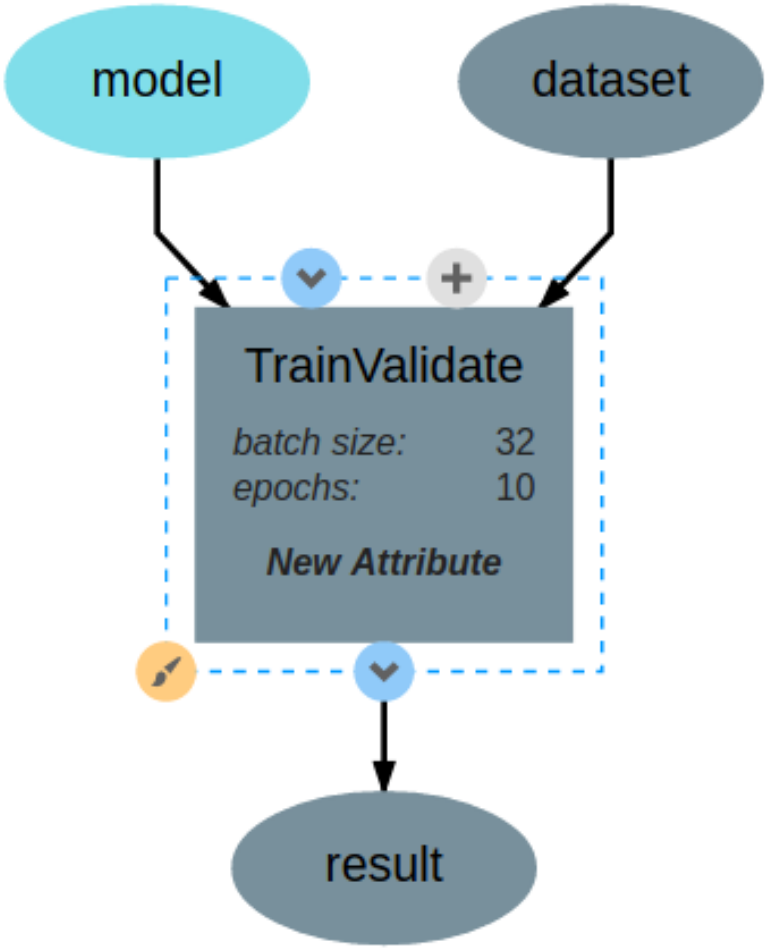

Fig. 2: The TrainValidate operation accepts training data, a model and attributes for setting the batch size, and the number of epochs.

Operation Interface | Environment

```
1  # Conda environment to use when executing the given operation.
2  # For more information, check out https://docs.conda.io/projects/cond
3  dependencies:
4    - python=3.7
5    - pip:
6      - tensorflow==1.14
7      - keras=2.2.5
8      - matplotlib
9      - numpy
10     - scikit-learn
11
```

Fig. 3: The operation environment contains python dependencies for the given operation.

(continued from previous page)

```
        return model.get_weights()

    def to_categorical(self, y, max_y=0.4, num_possible_classes=32):
        one_step = max_y / num_possible_classes
        y_cats = []
        for values in y:
            y_cats.append(int(values[0] / one_step))
        return y_cats

    def datagen(self, X, y):
        # Generates a batch of data
        X1, y1 = list(), list()
        n = 0
        while 1:stash@{1}
            for sample, label in zip(X, y):
                n += 1
                X1.append(sample)
                y1.append(label)
                if n == self.batch_size:
                    yield [[np.array(X1)], y1]
                    n = 0
                    X1, y1 = list(), list()


class PlotLosses(keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        self.i = 0
        self.x = []
        self.losses = []

    def on_epoch_end(self, epoch, logs={}):
        self.x.append(self.i)
        self.losses.append(logs.get('loss'))
        self.i += 1

        self.update()

    def update(self):
        plt.clf()
        plt.title("Training Loss")
        plt.ylabel("CrossEntropy Loss")
        plt.xlabel("Epochs")
        plt.plot(self.x, self.losses, label="loss")
        plt.legend()
        plt.show()
```

The "TrainValidate" operation uses capabilities from the `keras` package to train the neural network. This operation sets all the parameters using values provided to the operation as either attributes or references. In the implementation, attributes are provided as arguments to the constructor making the user defined attributes accessible from within the implementation. References are treated similarly to operation inputs and are also arguments to the constructor. This can be seen with the `model` constructor argument. Finally, operations return their outputs in the `execute` method; in this example, it returns a single output named `model`, that is, the trained neural network.

After defining the interface and implementation, we can now use the "TrainValidate" operation in our pipelines! An example is shown below.

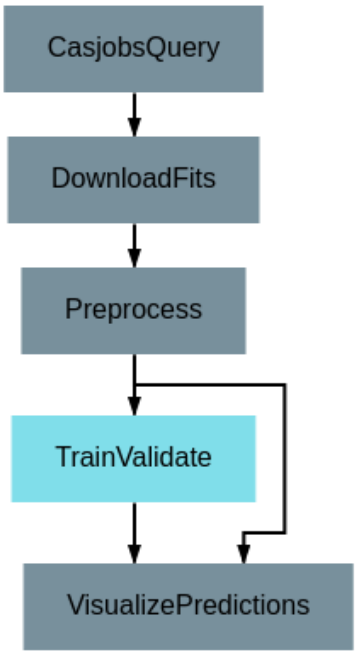

Fig. 4: Using the "TrainValidate" operation in a pipeline

3.2 Operation Feedback

Operations in DeepForge can generate metadata about its execution. This metadata is generated during the execution and provided back to the user in real-time. An example of this includes providing real-time plotting feedback. When implementing an operation in DeepForge, this metadata can be created using the `matplotlib` plotting capabilities.

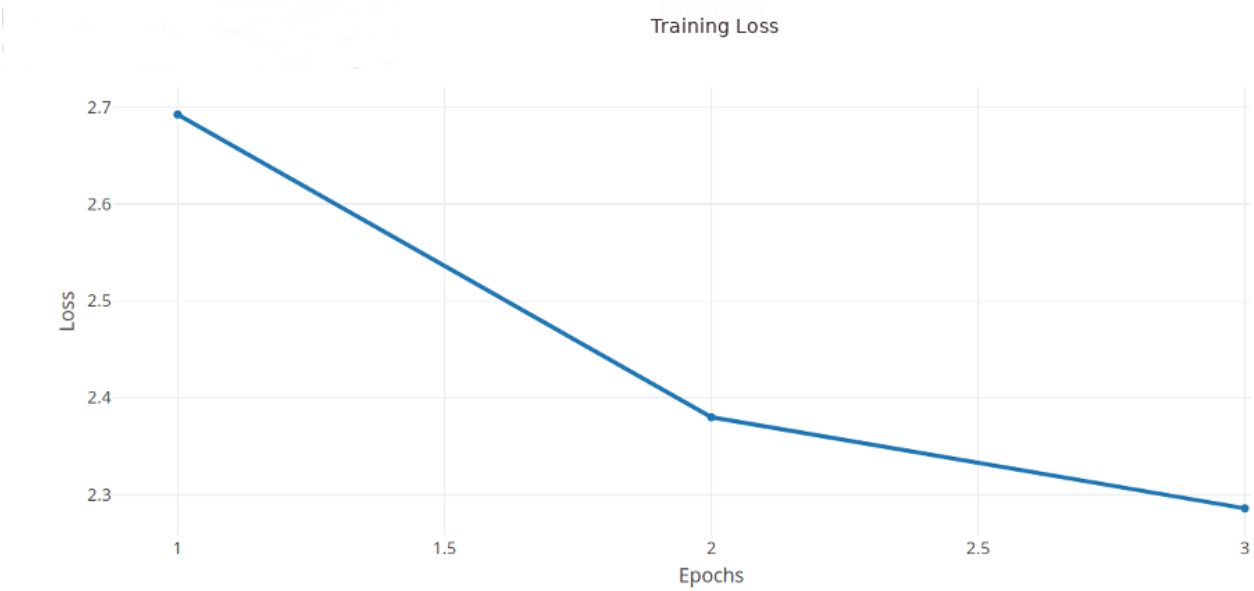

Fig. 5: An example graph of the loss function while training a neural network.

CHAPTER 4

Storage and Compute Adapters

DeepForge is designed to integrate with existing computational and storage resources and is not intended to be a competitor to existing HPC or object storage frameworks. This integration is made possible through the use of compute and storage adapters. This section provides a brief description of these adapters as well as currently supported integrations.

4.1 Storage Adapters

Projects in DeepForge may contain artifacts which reference datasets, trained model weights, or other associated binary data. Although the project code, pipelines, and models are stored in MongoDB, this associated data is stored using a storage adapter. Storage adapters enable DeepForge to store this associated data using an appropriate storage resource, such as a object store w/ an S3-compatible API. This also enables users to "bring their own storage" as they can connect their existing cyberinfrastructure to a public deployment of DeepForge. Currently, DeepForge supports 3 different storage adapters:

1. S3 Storage: Object storage with an S3-compatible API such as minio or AWS S3
2. SciServer Files Service : Files service from SciServer
3. WebGME Blob Server : Blob storage provided by WebGME

4.2 Compute Adapters

Similar to storage adapters, compute adapters enable DeepForge to integrate with existing cyberinfrastructure used for executing some computation or workflow. This is designed to allow users to leverage their existing HPC or other computational resources with DeepForge. Compute adapters provide an interface through which DeepForge is able to execute workflows (e.g., training a neural network) on external machines.

Currently, the following compute adapters are available:

1. WebGME Worker: A worker machine which polls for jobs via the WebGME Executor Framework. Registered users can connect their own compute machines enabling them to use their personal desktops with DeepForge.
2. SciServer-Compute: Compute service offered by SciServer

3. Server Compute: Execute the job on the server machine. This is similar to the execution model used by Jupyter notebook servers.

CHAPTER 5

Quick Start

The recommended (and easiest) way to get started with DeepForge is using docker-compose. First, install docker and docker-compose.

Next, download the docker-compose file for DeepForge:

```
wget https://raw.githubusercontent.com/deepforge-dev/deepforge/master/docker/docker-
↪compose.yml
```

Then start DeepForge using docker-compose:

```
docker-compose up
```

and now DeepForge can be used by opening a browser to http://localhost:8888!

For detailed instructions about deployment installations, check out our deployment installation instructions An example of customizing a deployment using docker-compose can be found here.

CHAPTER 6

Overview

6.1 DeepForge Component Overview

DeepForge is composed of four main elements:

- *Client*: The connected browsers working on DeepForge projects.
- *Server*: Main component hosting all the project information and is connected to by the clients.
- *Compute*: Connected computational resources used for executing pipelines.
- *Storage*: Connected storage resources used for storing project data artifacts such as datasets or trained model weights.

6.2 Component Dependencies

The following dependencies are required for each component:

- *Server* (NodeJS LTS)
- *Database* (MongoDB v3.0.7)
- *Client*: We recommend using Google Chrome and are not supporting other browsers (for now). In other words, other browsers can be used at your own risk.

6.3 Configuration

After installing DeepForge, it can be helpful to check out configuring DeepForge

CHAPTER 7

Native Installation

7.1 Dependencies

First, install NodeJS (LTS) and MongoDB. You may also need to install git if you haven't already.

Next, you can install DeepForge using npm:

```
npm install -g deepforge
```

Now, you can check that it installed correctly:

```
deepforge --version
```

After installing DeepForge, it is recommended to install the deepforge-keras extension which provides capabilities for modeling neural network architectures:

```
deepforge extensions add deepforge-dev/deepforge-keras
```

DeepForge can now be started with:

```
deepforge start
```

7.1.1 Database

Download and install MongoDB from the website. If you are planning on running MongoDB locally on the same machine as DeepForge, simply start *mongod* and continue to setting up DeepForge.

If you are planning on running MongoDB remotely, set the environment variable "MONGO_URI" to the URI of the Mongo instance that DeepForge will be using:

```
MONGO_URI="mongodb://pathToMyMongo.com:27017/myCollection" deepforge start
```

7.1.2 Server

The DeepForge server is included with the deepforge cli and can be started simply with

```
deepforge start --server
```

By default, DeepForge will start on *http://localhost:8888*. However, the port can be specified with the *–port* option. For example:

```
deepforge start --server --port 3000
```

7.1.3 Worker

The DeepForge worker (used with WebGME compute) can be used to enable users to connect their own machines to use for any required computation. This can be installed from *https://github.com/deepforge-dev/worker*. It is recommended to install Conda on the worker machine so any dependencies can be automatically installed.

7.1.4 Updating

DeepForge can be updated with the command line interface rather simply:

```
deepforge update
```

```
deepforge update --server
```

For more update options, check out *deepforge update –help*!

7.2 Manual Installation (Development)

Installing DeepForge for development is essentially cloning the repository and then using *npm* (node package manager) to run the various start, test, etc, commands (including starting the individual components). The deepforge cli can still be used but must be referenced from *./bin/deepforge*. That is, *deepforge start* becomes *./bin/deepforge start* (from the project root).

7.2.1 DeepForge Server

First, clone the repository:

```
git clone https://github.com/dfst/deepforge.git
```

Then install the project dependencies:

```
npm install
```

To run all components locally start with

```
./bin/deepforge start
```

and navigate to *http://localhost:8888* to start using DeepForge!

Alternatively, if jobs are going to be executed on an external worker, run *./bin/deepforge start -s* locally and navigate to *http://localhost:8888*.

7.2.2 Updating

Updating can be done the same as any other git project; that is, by running *git pull* from the project root. Sometimes, the dependencies need to be updated so it is recommended to run *npm install* following *git pull*.

7.3 Manual Installation (Production)

To deploy a deepforge server in a production environment, follow the following steps. These steps are for using a Linux server and if you are using a platform other than Linux, we recommend using a dockerized deployment.

1. Make sure you have a working installation of Conda in your server.
2. Clone this repository to your production server.

```
git clone https://github.com/deepforge-dev/deepforge.git
```

3. Install dependencies and add extensions:

```
cd deepforge && npm install
./bin/deepforge extensions add deepforge-dev/deepforge-keras
```

2. Generate token keys for user-management (required for user management).

```
chmod +x utils/generate_token_keys.sh
./utils/generate_token_keys.sh
```

Warning: The token keys are generated in the root of the project by default. If the token keys are stored in the project root, they are accessible via */extlib*, which is a security risk. So, please make sure you move the created token keys out of the project root.

3. Configure your environment variables:

```
export MONGO_URI=mongodb://mongo:port/deepforge_database_name
export DEEPFORGE_HOST=https://url.of.server
export DEEPFORGE_PUBLIC_KEY=/path/to/public_key
export DEEPFORGE_PRIVATE_KEY=/path/to/private_key
```

4. Add a site-admin account by using `deepforge-users` command:

```
./bin/deepforge-users useradd -c -s admin_username admin_email admin_password
```

5. Now you should be ready to deploy a production server which can be done using `deepforge` command.

```
NODE_ENV=production ./bin/deepforge start --server
```

Note: The default port for a deepforge server is 8888. It can be changed using the option *-p* in the command above.

CHAPTER 8

Command Line Interface

This document outlines the functionality of the deepforge command line interface (provided after installing deepforge with `npm install -g deepforge`).

- Installation Configuration
- Starting DeepForge or Components
- Update or Uninstall DeepForge
- Managing Extensions

8.1 Installation Configuration

Installation configuration can be edited using the `deepforge config` command as shown in the following examples:

Printing all the configuration settings:

```
deepforge config
```

Printing the value of a configuration setting:

```
deepforge config blob.dir
```

Setting a configuration option, such as `blob.dir` can be done with:

```
deepforge config blob.dir /some/new/directory
```

For more information about the configuration settings, check out the configuration page.

8.2 Starting DeepForge Components

The DeepForge server can be started with the `deepforge start` command. By default, this command will start both the server and a mongo database (if applicable).

The server can be started by itself using

```
deepforge start --server
```

8.3 Update/Uninstall DeepForge

DeepForge can be updated or uninstalled using

```
deepforge update
```

DeepForge can be uninstalled using `deepforge uninstall`

8.4 Managing Extensions

DeepForge extensions can be installed and removed using the `deepforge extensions` subcommand. Extensions can be added, removed and listed as shown below

```
deepforge extensions add https://github.com/example/some-extension
deepforge extensions remove some-extension
deepforge extensions list
```

CHAPTER 9

Configuration

Configuration of deepforge is done through the *deepforge config* command from the command line interface. To see all config options, simply run *deepforge config* with no additional arguments. This will print a JSON representation of the configuration settings similar to:

```
Current config:
{
  "blob": {
    "dir": "/home/irishninja/.deepforge/blob"
  },
  "mongo": {
    "dir": "~/.deepforge/data"
  }
}
```

Setting an attribute, say *blob.dir*, is done as follows

```
deepforge config blob.dir /tmp
```

9.1 Environment Variables

Most settings have a corresponding environment variable which can be used to override the value set in the cli's configuration. This allows the values to be temporarily set for a single run. For example, starting the server with a different blob location can be accomplished by setting *blob.dir* can be done with:

```
DEEPFORGE_BLOB_DIR=/tmp deepforge start -s
```

The complete list of the environment variable overrides for the configuration options can be found here.

9.2 Settings

9.2.1 blob.dir

The path to the blob (large file storage containing models, datasets, etc) to be used by the deepforge server.

This can be overridden with the *DEEPFORGE_BLOB_DIR* environment variable.

9.2.2 mongo.dir

The path to use for the *–dbpath* option of mongo if starting mongo using the command line interface. That is, if the MONGO_URI is set to a local uri and the cli is starting the deepforge server, the cli will check to verify that an instance of mongo is running locally. If not, it will start it on the given port and use this setting for the *–dbpath* setting of mongod.

CHAPTER 10

Operation Feedback

Operations can provide a variety of forms of real-time feedback including subplots, 2D and 3D plots and images using `matplotlib`.

10.1 Graphs

The following example shows a sample 3D scatter plot and its rendering in DeepForge.

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D

class Scatter3DPlots():

    def execute(self):
        # Set random seed for reproducibility
        np.random.seed(19680801)


        def randrange(n, vmin, vmax):
            '''
            Helper function to make an array of random numbers having shape (n, )
            with each number distributed Uniform(vmin, vmax).
            '''
            return (vmax - vmin)*np.random.rand(n) + vmin

        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')

        n = 100

        # For each set of style and range settings, plot n random points in the box
        # defined by x in [23, 32], y in [0, 100], z in [zlow, zhigh].
```

(continues on next page)

(continued from previous page)

```
        for m, zlow, zhigh in [('o', -50, -25), ('^', -30, -5)]:
            xs = randrange(n, 23, 32)
            ys = randrange(n, 0, 100)
            zs = randrange(n, zlow, zhigh)
            ax.scatter(xs, ys, zs, marker=m)

        ax.set_xlabel('X Label')
        ax.set_ylabel('Y Label')
        ax.set_zlabel('Z Label')
        plt.show()
```

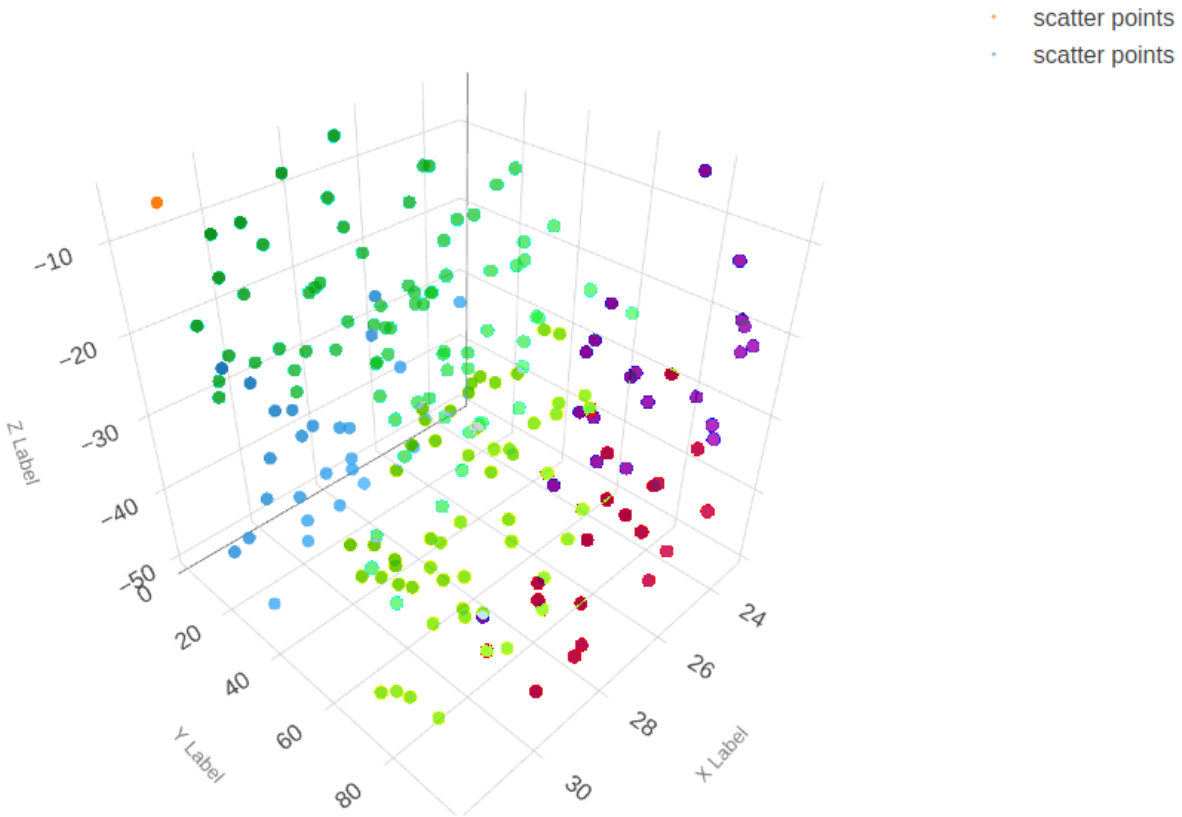

Fig. 1: Example of a 3D scatter plot using matplotlib in DeepForge

10.2 Images

Visualizing images using *matplotlib* is also supported. The following example shows images from the *MNIST fashion dataset*.

```
from matplotlib import pyplot
from keras.datasets import fashion_mnist

class MnistFashion():

    def execute(self):

        (trainX, trainy), (testX, testy) = fashion_mnist.load_data()
        # summarize loaded dataset
        print('Train: X=%s, y=%s' % (trainX.shape, trainy.shape))
        print('Test: X=%s, y=%s' % (testX.shape, testy.shape))
        for i in range(9):
            pyplot.subplot(330 + 1 + i)  # define subplot
            pyplot.imshow(trainX[i], cmap=pyplot.get_cmap('gray'))  # plot raw pixel␣
↪data
        pyplot.show()
```