Neptune – Machine Learning Platform

In January 2016, deepsense.io won the Right Whale Recognition (https://www.kaggle.com/c/noaa-right-whale-recognition) contest on kaggle (https://www.kaggle.com/). The competition's goal was to automate the right whale recognition process using a dataset of aerial photographs of individual whales. The terms and conditions for the competition stated that to collect the prize, the winning team had to provide source code and a description of how to recreate the winning solution. A fair request, but as it turned out, the winning solution's authors spent about three weeks recreating all of the steps that led them to the winning machine learning model.

When data scientists work on a problem, they need to test many different approaches — various algorithms, neural network structures, numerous hyperparameter values that can be optimized etc. The process of validating one approach can be called an experiment. The inputs for every experiment include: source code, data sets, hyperparameter values and configuration files. The outputs of every experiment are: output model weights (definition of the model), metric values (used for comparing different experiments), generated data and execution logs. As we can see, that's a lot of different artifacts for each experiment. It is crucial to save all of these artifacts to keep track of the project — comparing different models, determining which approaches were already tested, expanding research from some experiment from the past etc. Managing the process of experiment executions is a very hard task and it is easy to make a mistake and lose an important artifact.

To make the situation even more complicated, experiments can depend on each other. For example, we can have two different experiments training two different models and a third experiment that takes these two models and creates a hybrid to generate predictions. Recreating the best solution means finding the path from the original data set to the model that gives the best results.
Recreating the path that led to the best model

The deepsense.io research team performed around 1000 experiments to find the competition-winning solution. Knowing all that, it becomes clear why recreating the solution was such a difficult and time consuming task.

The problem of recreating a machine learning solution is present not only in an academic environment. Businesses struggle with the same problem. The common scenario is that the research team works to find the best machine learning model to solve a business problem, but then the software engineering team has to put the model into a production environment. The software engineering team needs a detailed description of how to recreate the model.

Our research team needed a platform that would help them with these common problems. They defined the properties of such a platform as:

- Every experiment and the related artifacts are registered in the system and accessible for browsing and comparing;
- Experiment execution can be monitored via real-time metrics;
- Experiment execution can be aborted at any time;
- Data scientists should not be concerned with the infrastructure for the experiment execution.

deeptsense.io decided to build Neptune – a brand new machine learning platform that organizes data science processes. This platform relieves data scientists of the manual tasks related to
managing their experiments. It helps with monitoring long-running experiments and supports team collaboration. All these features are accessible through the powerful Neptune Web UI and useful for scripting CLI. Neptune is already used in all machine learning projects at deepsense.io. Every week, our data scientists execute around 1000 experiments using this machine learning platform. Thanks to that, the machine learning team can focus on data science and stop worrying about process management.

**Experiment Execution in Neptune**

**Main Concepts of the Machine Learning Platform**

**Job**

A job is an experiment registered in Neptune. It can be registered for immediate execution or added to a queue. The job is the main concept in Neptune and contains a complete set of artifacts related with the experiment:

- source code snapshot: Neptune creates a snapshot of the source code for every job. This allows a user to revert to any job from the past and get the exact version of the code that was executed;
- metadata: name, description, project, owner, tags;
- parameters: customly defined by a user. Neptune supports boolean, numeric and string types of parameters;
- data and logs generated by the job;
- metric values represented as channels.

Neptune is library and framework agnostic. Users can leverage their favorite libraries and frameworks with Neptune. At deepsense.io we currently execute Neptune jobs that use: TensorFlow, Theano, Caffe, Keras, Lasagne or scikit-learn.

**Channel**

A channel is a mechanism for real-time job monitoring. In the source code, a user can create channels, send values through them and then monitor these values live using the Neptune Web UI. During job execution, a user can see how his or her experiment is performing. The Neptune machine learning platform supports three types of channels:

- Numeric: used for monitoring any custom-defined metric. Numeric channels can be displayed as charts. Neptune supports dynamic chart creation from the Neptune Web UI with multiple channels displayed in one chart. This is particularly useful for comparing various metrics;
- Text: used for logs;
Queue

A queue is a very simple mechanism that allows a user to execute his or her job on remote infrastructure. A common setup for many research teams is that data scientists develop their code on local machines (laptops), but due to hardware requirements (powerful GPU, large amount of RAM, etc) code has to be executed on a remote server or in a cloud. For every experiment, data scientists have to move source code between the two machines and then log into the remote server to execute the code and monitor logs. Thanks to our machine learning platform, a user can enqueue a job from a local machine (the job is created in Neptune, all metadata and parameters are saved, source code copied to users’ shared storage). Then, on a remote host that meets the job requirements the user can execute the job with a single command. Neptune takes care of copying the source code, setting parameters etc. The queue mechanism can be used to write a simple script that queries Neptune for enqueued jobs and execute the first job from the queue. If we run this script on a remote server in an infinite loop, we don’t have to log to the server ever again because the script executes all the jobs from the queue and reports the results to the machine learning platform.

Creating a Job

Neptune is language and framework agnostic. A user can communicate with Neptune using REST
API and Web Sockets from his or her source code written in any language. To make the communication easier, we provide a high-level client library for Python (other languages are going to be supported soon).

Let's examine a simple job that provided with `amplitude` and `sampling_rate` generates sine and cosine as functions of time (in seconds).

```python
import math
import time
from deepsense import neptune

ctx = neptune.Context()

amplitude = ctx.params.amplitude
sampling_rate = ctx.params.sampling_rate

sin_channel = ctx.job.create_channel(name='sin', channel_type=neptune.ChannelType.NUMERIC)

logins_channel = ctx.job.create_channel(name='logging', channel_type=neptune.ChannelType.TEXT)

cxo.channel.create_chart(name='sin & cos chart', series={'sin': sin_channel, 'cos': cos_channel})

cxo.job.finalize_preparation()

# The time interval between samples.
period = 1.0 / sampling_rate

# The initial timestamp, corresponding to x = 0 in the coordinate axis.
zero_x = time.time()

iteration = 0

while True:
    iteration += 1

    # Computes the values of sine and cosine.
    now = time.time()
x = now - zero_x

    sin_y = amplitude * math.sin(x)
    cos_y = amplitude * math.cos(x)

    # Sends the computed values to the defined numeric channels.
    sin_channel.send(x=x, y=sin_y)
    cos_channel.send(x=x, y=cos_y)

    # Formats a logging entry.
    logging_entry = "sin({x})={sin_y}; cos({x})={cos_y}".format(x=x, sin_y=sin_y, cos_y=cos_y)

    # Sends a logging entry.
    logging_channel.send(x=iteration, y=logging_entry)

    time.sleep(period)
```

The first thing that we can see is that we need to import Neptune library and create a `neptune.Context` object. The Context object is an entrypoint for Neptune integration. Afterwards, using the context we obtain values for job parameters: `amplitude` and `sampling_rate`.

Then, using `neptune.Context.job` we create numeric channels for sending sine and cosine values and a text channel for sending logs. We want to display `sin_channel` and `cos_channel` on a chart, so
we use `neptune.Context.job.create_chart` to define a chart with two series named `sin` and `cos`. After that, we need to tell Neptune that the preparation phase is over and we are starting the proper computation. That is what: `ctx.job.finalize_preparation()` does.

In an infinite loop we calculate sine and cosine functions values and send these values to Neptune using the `channel.send` method. We also create a human-readable log and send it through `logging_channel`.

To run main.py as a Neptune job we need to create a configuration file – a descriptor file with basic metadata for the job.

```yaml
1 name: Sine-Cosine Generator
2 project: Trigonometry
3 owner: Your Name
4 parameters:
5   - name: amplitude
6     type: double
7     default: 1.0
8     required: false
9   - name: sampling_rate
10    type: double
11    default: 2
12    required: false
```

`config.yaml` contains basic information about the job: name, project, owner and parameter definitions. For our simple Sine-Cosine Generator we need two parameters of double type: `amplitude` and `sampling_rate` (we already saw in the `main.py` how to obtain parameter values in the code).

To run the job we need to use the Neptune CLI command:

```
neptune run main.py --config config.yaml --dump-dir-url my_dump_dir -- --amplitude 5 --sampling_rate 2.5
```

For `neptune run` we specify: the script that we want to execute, the configuration for the job and a path to a directory where snapshot of the code will be copied to. We also pass values of the custom-defined parameters.

**Job Monitoring**

Every job executed in the machine learning platform can be monitored in the Neptune Web UI. A user can see all useful information related to the job:

- metadata (name, description, project, owner);
- job status (queued, running, failed, aborted, succeeded);
- location of the job source code snapshot;
- location of the job execution logs;
A data scientist can monitor custom metrics sent to Neptune through the channel mechanism. Values of the incoming channels are displayed in the Neptune Web UI in real time. If the metrics are not satisfactory, the user can decide to abort the job. Aborting the job can also be done from the Neptune Web UI.
Comparing values of multiple metrics using Neptune channels

Numeric channels can be displayed graphically as charts. A chart representation is very useful to compare various metrics and to track changes of metrics during job execution.
Chart for Sine-Cosine Generator

Charts displaying custom metrics
For every job a user can define a set of tags. Tags are useful for marking significant differences between jobs and milestones in the project (i.e. if we are doing a MINST project, we can start our research by running the job with a well known and publicly available algorithm and tag it ‘benchmark’).

## Comparing Results and Collaboration

Every job executed in the Neptune machine learning platform is registered and available for browsing. Neptune’s main screen shows a list of all executed jobs. User can filter jobs using job metadata, execution time and tags.

![Neptune jobs list](image)

A user can select custom-defined metrics to show as columns on the list. The job list can be sorted using values from every column. That way, a user can select which metric he or she wants to use for comparison, sort all jobs using this metric and then find the job with the best score.

Thanks to a complete history of job executions, data scientists can compare their jobs with jobs executed by their teammates. They can compare results, metrics values, charts and even get access to the snapshot of code of a job they’re interested in.

Thanks to Neptune, the machine learning team at deepsense.io was able to:
- get rid off spreadsheets for keeping history of executed experiments and their metrics values;
- eliminate sharing source code across the team as an email attachment or other innovative tricks;
- limit communication required to keep track of project progress and achieved milestones;
- unify visualisation for metrics and generated data.

**Join the Early Adopters Program**

Apply for our Early Adopters Program (/neptune-early-adopter-program/) and get early access to Neptune – Machine Learning Platform. Benefits of joining the program include:

- You will be one of the first to get full access to this innovative product designed especially for data scientists for FREE;
- You will have direct impact on future product features;
- You will get support from our team of engineers;
- You can share your ideas with our experts and the community of the world’s leading data scientists.