Soft Computing
Job Performance Evaluation Using Back-propagation Network

Bc. Petr Stehlík <xstehl14@stud.fit.vutbr.cz>

1 Introduction

Every user of a supercomputer needs to know whether their submitted job finished successfully and performed well. So far these tedious tasks are usually performed manually using only the output of their program and over-simplified metrics such as job runtime and total utilized resources.

The aim of this project is to create a back-propagation neural network which can classify whether a job ran well or if it is in some way suspicious of unwanted behaviours such as poor performance or execution failure.

The network itself is supplied with fine-granular metric data acquired via Examon framework[2] which was run on Galileo supercomputer located in CINECA, Bologna, Italy. Where all job and metric data were gathered.

The structure of this document is as follows: in section 2 the theoretical background needed for this project is presented together with the description of Examon framework. In the following section 3 the data supplied to the network are described as well as the final format of the data. Afterwards, in section 4 the implementation of the network and its structure are laid out and in the last two sections 5 and 6 the achieved results, summary and further work are discussed.

2 Theoretical Background

2.1 Backpropagation Network

Backpropagation networks are multi-layer feed-forward networks with supervised learning. There is no interconnection between neurons in the same layer but layers are fully connected to the next neighbouring layer in order to be able to do forward and backward propagation of values.

2.1.1 Forward-propagation

Each neuron in all layers but input layer disposes of a weight for each input initialized to a random value in range $< 0, 1 >$, linear base function and sigmoidal activation function.

When forward propagating an input vector the vector is laid out onto the input neurons and the vector is recalculated using following formulas for the next layer until the input vector is propagated to the output layer which outputs the response of the whole network itself.

The base function is shown in equation 1:

$$f(\vec{x}) = \sum_{i=0}^{n} w_i x_i \tag{1}$$

where $\vec{x}$ is the input vector, w are weights for each input of given neuron and n is the length of the neuron input vector and bias (term used as in [3][7]) value resulting in $n = |x| + 1$.

The sigmoidal activation function is presented in equation 2 where λ is a constant set to $\lambda = 1$ and in further equations left out because of this fact.

$$g(u) = \frac{1}{1 + e^{-\lambda u}} \tag{2}$$

The output of a neuron is then given by using the equations 1 and 2:

$$y = g(f(\vec{x})) \tag{3}$$

2.1.2 Back-propagation

Back-propagation is used only when one trains the network and is one of the base methods for training feed-forward networks. The method is based on adjusting weights depending on the error calculated by equation 4 for an output neuron p using produced output (o) and desired output (d) values.

$$E_p = \frac{1}{2}\sum_{j=1}^{m}(d_{pj} - o_{pj})^2 \tag{4}$$

The change of weights is calculated using equation 5 where ∇E_p is the derived error gradient and μ the learning rate.

$$\Delta\vec{w_p} = -\mu\nabla E_p \tag{5}$$

To calculate one particular change of weight we use formula 6 where l is the given layer of the network, j is the j-th neuron in layer L and i is the i-th input of the neuron j.

$$\Delta^l w_{ji} = \mu^l \delta_j{}^l x_i \tag{6}$$

For the output layer l of the network we use formula 7.

$$\delta_j^L = (d_j - {}^l y_j)^l y_j (1 - {}^l y_j) \tag{7}$$

For hidden layers of the network, we use the same principle propagating the error backwards from the output layer as shown in equation 8 where $\lambda = 1$ and therefore left out.

$$\Delta^l w_{ji} = \mu^l \delta_j{}^l x_i = \mu \sum_{k=1}^{n_{l+1}} ({}^{l+1}\delta_k{}^{l+1} w_{kj})^l y_j (1 - {}^l y_j)^l x_i \tag{8}$$

Each repetition of forward and backward propagation of all inputs is called an epoch. Finally, we can choose when to update the weights, in batches after all input vectors were processed or using the stochastic method where weights are updated after processing each input vector.

2.2 Examon

Examon[2] framework is used for exascale monitoring of supercomputing facilities. It is built on top of MQTT protocol[6] which allows measured metrics to be sent to a central broker where received data are processed and stored in KairosDB[1] database utilizing Cassandra[5] cluster.

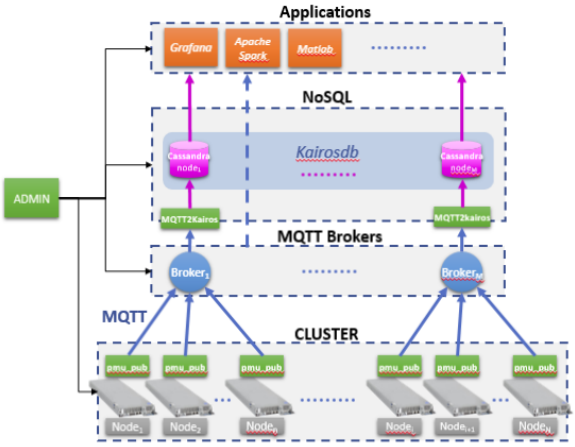

Figure 1: Examon framework architecture

KairosDB is used for storing metric data in time-series format whereas Cassandra, serving as a backend for KairosDB is also used for storing job-related data. More on data semantics is described in 3.

3 Data

As previously stated in 2.2 data are gathered via Examon framework and stored in Cassandra database. We can split the data into two categories. Job data and metric data.

The job data come from PBS Pro[8] hooks which report various info about the job, mainly the allocated nodes, cores and other computational resources. The timestamps and events which are triggered by PBS Pro during the lifecycle of the job. We use this data for determining the runtime of a job, its resource allocations and location of the job in a cluster (node names and core numbers). The job data are sent via MQTT and stored directly to a Cassandra cluster.

Using the job data one can query metric data. These are measured independently of job data and are monitored on per-core, per-CPU or per-node basis categorized into metrics. Each measured value is then sent via MQTT to KairosDB where it is stored according to its cluster location and metric. There are over 30 metrics monitored including but not limited to core load; C6 and C3 CPU state shares; system, CPU, IO and memory utilization or various temperatures gathered from various places inside a node.

For the project, twelve major metrics were chosen for the best reflection of job performance. A short summary of the chosen metrics is shown in table 1.

Metric name	Metric tag	unit	sampling rate	base
core's load	load_core	%	2s	per-core
C6 states	C6res	%	2s	per-core
C3 states	C3res	%	2s	per-core
instructions per second	ips	IPS	2s	per-core
system utilization	Sys_Utilization	%	20s	per-node
CPU utilization	CPU_Utilization	%	20s	per-node
IO utilization	IO_Utilization	%	20s	per-node
memory utilization	Memory_Utilization	%	20s	per-node
L1 and L2 bounds	L1L2_Bound	%	2s	per-core
L3 bounds	L3_Bound	%	2s	per-core
front-end bounds	front_end_bound	%	2s	per-core
back-end bounds	back_end_bound	%	2s	per-core

Table 1: Overview of measured metrics

KairosDB provides us with a REST API for querying metric data in various ways. The queries are formed using JSON objects and results are also returned as JSON objects. The KairosDB limits all stored data to 21 days

For complete data acquisition, we combined the job data and metric data together and queried only jobs which fit several conditions described in 4.1.

4 Implementation

In this section, we describe the implementation phase of the project. The project was implemented using Python 2.7.13 with the usage of library SciPy[4] for linear data interpolation. All code was tested and evaluated on the Merlin server at FIT BUT.

4.1 Data Acquisition

First, job data were queried and filtered according to several rules:

- job runtime must be between 10 and 60 minutes
- job must occupy the whole node (multiplies of 16 cores)
- job must be run in 21-day period (30. 10. 2017 – 21. 11. 2017)

These rules were satisfied by 442 jobs in total with their runtime closer to 10 minutes than 60 minutes. The Examon framework was run only on a small portion of Galileo cluster and therefore not all filtered jobs had metric data present in the metric database.

This resulted in 32 jobs which gave enough diversity for correct classification. All data points were aggregated by 30 seconds on cluster level[1] using averaging aggregator available in KairosDB. There are 19 jobs classified as suspicious and 120 out of 384 metric data vectors classified as suspicious.

The rule of minimum 10 minutes is because of any shorter job is usually a development, not the production version of a program and in current HPC facilities, such program is very cheap to run and therefore no deep performance analysis is needed. The same goes for jobs smaller than one node (16 cores in case of Galileo supercomputer).

Jobs longer than one hour are not suited for training the network because of too large input vectors exceeding the scope of this project and the loss of information in further data processing.

4.2 Data Labelling

In order to correctly label all chosen jobs and their metrics, a simple graphical user interface was made. The GUI is based on Examon Web which is an extension of Examon framework visualizing specific views of all collected data.

Visualization is done in time series fashion using Dygraphs library. This helps to better understand the in time correlations between all metrics combined.

The GUI is shown in figure 2 where you can see a checkbox for each metric and at the bottom a checkbox for the whole job. The metric checkbox labels the accompanying metric of suspicious behaviour and job one of suspicious behaviour of the job as a whole. A suspicious metric or job is labelled with value 1.0 and unsuspicious was labelled by value 0.0.

Figure 2: Part of Examon Web based data labelling tool

4.3 Data Processing

After labelling the job all metric data with labels are generated and can be worked on further. All metric vectors were interpolated to 80 values which provide good performance/detail compromise. Larger input vectors resulted in extremely long runs and smaller input vectors resulted in poor outputs of the network.

All values were also normalized to values between $< 0.0, 1.0 >$.

[1] Averaging every used core and node in the job to one time serie per metric.

4.4 Metric Networks

To achieve best results/speed combination a backpropagation neural network was created for each metric. This gives us the total of twelve networks completely independent of each other meaning the training process can be fully parallelized.

All metric networks were configured the same way to achieve uniform results. The input layer disposes of 80 neurons with hidden layers of 4 and then 3 neurons and one output neuron.

This configuration was chosen based on trial and error experiments.

4.5 Job Network

Once all metric networks compute their output we have a vector of 12 values serving as an input to the job classification network.

The network is created with 12 input neurons, one hidden layer of 4 neurons and one output neuron.

All networks were set to train a maximum of 50 000 epochs or until the sum error reached 0.1. They were presented with 27 jobs keeping the other 5 jobs as the evaluation dataset. All outputs are a single value–metric networks because of further processing in the job network and the job network because of clearer interpretation of results.

5 Achieved Results

The sum error was calculated using equation 4. Each metric had its sum error as presented in table 2. As one can see the sum error varies through the dataset but sum errors below 1.0 can be considered as acceptable. The largest error comes from core's load and CPU utilization since these two metrics are very similar.

Name	Epochs	Sum Error
core's load	49999	5.81947
C6 states	1506	0.07591
C3 states	4	0.09447
instructions per second	49999	3.18977
system utilization	17988	0.09695
CPU utilization	49999	7.65817
IO utilization	49999	0.94946
memory utilization	49999	0.82774
L1 and L2 bounds	49999	0.52690
L3 bounds	49999	0.77589
front-end bounds	49999	2.72293
back-end bounds	49999	4.63103
job classifier	49999	0.73037

Table 2: Overview of sum errors for all networks

Job #	Expected	Result
1	0	0.04
2	0	0.05
3	0	0.58
4	1	1.0
5	1	0.96

Table 3: Sum error for job evaluation dataset

Even with these sum errors the job classifier labelled the evaluation dataset correctly with minor errors as seen in table 3

6 Summary

As a proof of concept for classifying jobs on HPC cluster, it was verified that such classification can be made using a set of backpropagation networks. With rather a small dataset to train and evaluate the networks the error rate wasn't minimal and can be further reduced using a larger labelled dataset and users' feedback.

The dataset itself cannot be randomly generated because of too complex dependencies between all metrics. Having randomly generated data we even couldn't determine whether such artificial job would be really suspicious or not.

Using the intermediate input from metric networks it can be used for intelligent dashboards in Examon Web where only suspicious metrics can be shown in combination with several fixed ones.

This work is the base for my master thesis and will be further expanded to provide complex insights on users' jobs suggesting the possible cause of the suspicious behaviour.

7 Program Manual

The Python code can be run using command `python jobclassifier` from the root folder of the project. The program makes available following arguments:

- `-h, --help` show help
- `--train` train the networks using `data.json` dataset
- `--dir CONFIG_DIR` where to store configurations for trained networks
- `--max-epochs EPOCHS` train the network using `data.json` dataset (default: 10 000)
- `--eval CONFIG_EVAL` maximum number of epochs

An example command for training the networks: `python jobclassifier --train --dir example` and example command for evaluating the dataset using networks trained from previous example: `python jobclassifier --eval example`.

Acknowledgements

This work was done using the Examon framework created by F. Beneventi, A. Bartolini, A. Borghesi and collective at UNIBO, Italy. Examon Web was created by P. Stehlík during the PRACE Summer of HPC stay at CINECA, Bologna, Italy. Data used in this project were collected using the CINECA infrastructure and Galileo supercomputer and anonymized before any use in this project.

References

[1] KairosDB. `https://kairosdb.github.io`. Accessed: 2017-11-25.

[2] F. Beneventi, A. Bartolini, C. Cavazzoni, and L. Benini. Continuous learning of hpc infrastructure models using big data analytics and in-memory processing tools. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 1038–1043, March 2017.

[3] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[4] E. Jones, T. Oliphant, and P. Peterson. {SciPy}: open source scientific tools for {Python}. 2014.

[5] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[6] D. Locke. Mq telemetry transport (mqtt) v3. 1 protocol specification. *IBM developerWorks Technical Library*, 2010.

[7] K. Mehrotra, C. K. Mohan, and S. Ranka. *Elements of artificial neural networks*. MIT press, 1997.

[8] A. P. Works. Pbs professional®14.1 administrator's guide, 2016.