Journal of Physics: Conference Series **664** (2015) 052042 doi:10.1088/1742-6596/664/5/052042

Applying deep neural networks to HEP job classification

L Wang, J Shi, X Yan

IHEP computing center, P.O. Box 918-7, 19B Yuquan Road, Beijing 100049, China

E-mail: Lu.Wang@ihep.ac.cn

Abstract. The cluster of IHEP computing center is a middle-sized computing system which provides 10 thousands CPU cores, 5 PB disk storage, and 40 GB/s IO throughput. Its 1000+ users come from a variety of HEP experiments. In such a system, job classification is an indispensable task. Although experienced administrator can classify a HEP job by its IO pattern, it is unpractical to classify millions of jobs manually. We present how to solve this problem with deep neural networks in a supervised learning way. Firstly, we built a training data set of 320K samples by an IO pattern collection agent and a semi-automatic process of sample labelling. Then we implemented and trained DNNs models with Torch. During the process of model training, several meta-parameters was tuned with cross-validations. Test results show that a 5-hidden-layer DNNs model achieves 96% precision on the classification task. By comparison, it outperforms a linear model by 8% precision.

1. Introduction

The cluster of IHEP computing center (CC-IHEP) is a middle-sized computing system which provides 10 thousands CPU cores, 3 PB disk storage, and 40 GB/s IO throughput. Its 1000+ users come from a variety of HEP experiments such as ATLAS, BESIII, CMS, Daya Bay etc. On such a cluster, *Job Classification* is an indispensable task in the scenario of job statistics: Different job types may hit different pitfalls of software and may have variant requirements of CPU, memory and bandwidth. Therefore, it is more reasonable to make respective job statistics for different of job types.

Job type is not necessarily specified by cluster users. For example, it is difficult to guess the type information from a job name like “mytest_001.txt”. Although experienced administration can infer the type information by a job’s IO pattern, it is unpractical to classify millions of jobs manually. Therefore, an automatic classification mechanism is need for job statistics at CC-IHEP.

The idea of this paper is to solve this problem by a machine learning algorithm called Deep Neural Networks (DNNs). As shown in figure 1, we trained a DNN model to present the relationship between a job’s IO pattern and its type. Firstly, we provided the learning algorithm with hundreds of thousand labelled samples. These samples work as examples of how the administrator will classify these jobs. By “learning” from these examples, the machine learning program optimized the DNN model to make its estimation of job types as close to the labelled values as possible. After that, when a new job comes, the model can produce a reliable estimation of the job type in an automatic way.

Deep Neural Network [1] is a non-linear machine learning algorithm inspired by the brain. It has properties of scaling to very large datasets and very large feature space, supporting on-line training, capability of feature learning etc. Recently, it has achieved better performance than other state-of-art algorithms in domains of natural language processing [2], computer vision [3], and speech recognition [4]. In high energy physics domain, physicists have started to use DNNs on the problem of exotic particles classification, and encouraging results have been presented [5].

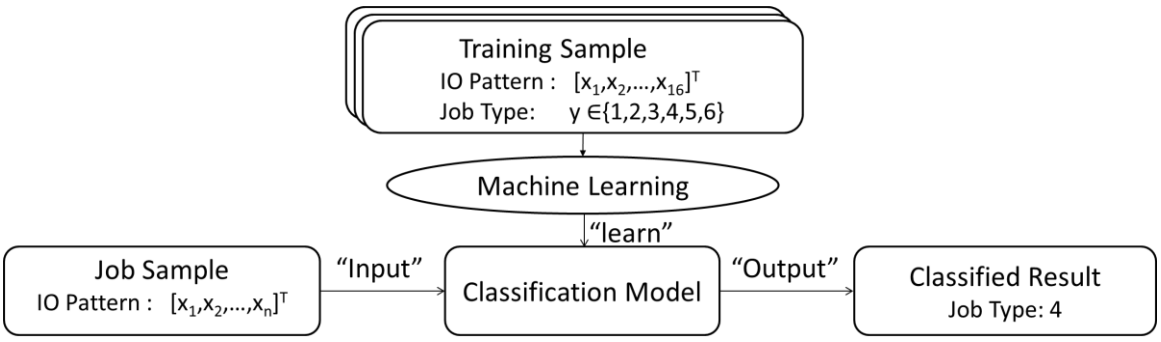


Figure 1. The main idea of this paper.

2. Implementation

The implementation of this paper includes six stages: training data collection, model definition, data pre-processing, model training, cross validation and test. Depending on conditions happened on cross validation stage, this procedure will be iterated many times to get an optimized performance.

2.1. Training data collection

Related work shows [6] that the scale of training dataset is the most important factor to achieve good performance of machine learning. On the premise that reasonable training procedure is followed, the volume of dataset has larger impact on performance than the choice of machine learning algorithms. Before model training, we collected a training set of 320,000 samples. Every training sample includes a 16 dimension IO pattern vector and a labelled job type. The rest of this part will introduce how we collect IO pattern by an instrumenting agent and how we label training samples by keyword matching.

2.1.1. IO pattern collection

Since most HEP jobs are data intensive, IO pattern is chosen as the input features of classification. It is defined as 16 counters of file operations in a fixed time window. By default the time window is one minute. There are mainly four kinds of file operations: *read, write, seek before read, and seek before write*. For each kind of operation, there are four sub-categories divided by operation size. A *small* operation ranges from 0 to 4 kilobytes, a *middle* operation ranges from 4 kilobytes to 1 megabytes, a *large* operation ranges from 1 megabytes to 4 megabytes, and an *extremely large* operation is larger than 4 megabytes. Figure 2 shows how an IO pattern looks like.

SR	MR	LR	XLR	SW	MW	LW	XLW
SRS	MRS	LRS	XLRS	SWS	MWS	LWS	XLWS

- S: small sized operation, [0-4KB]
- M: middle sized operation, [4KB,1MB]
- L: large sized operation, [1MB,4MB]
- XL: extremely large sized operation [>4MB]
- *R: read* *RS: seek before read* *W: Write* *WS: seek before write*

Figure 2. How the 16 dimension IO pattern looks like?

```
my @sim=
(" sim ", " Sim ", " SIM ", " simch ",
 " simcheck ", " simMcPhiEta ",
"simPhiEta ", " simu ", " bgsim ",
"simulation", " DDstarbarsim ",
"testMC ","CocktailMC", "psiMC",
 " zerowidthMC ","DbarDstarsim
","MCggphi", "McJpsi", "psipMC",
"SMALLJPSIPSMC", "ddbarmc", ...);
```

Figure 3. How a key word array of simulation jobs looks like?

IO pattern is collected by an agent running on every computing node. If a job is multi-processed, the numbers of each process will be summed. A job will be sampled at least 15 minutes after it starts. By the assumption that a job's IO pattern is stable during the main part of its life time, each job is sampled only once.

If the job is accessing Lustre file system [7], the agent will parse two /proc files to get numbers of the 16 dimension counter. The two files are: "/proc/fs/lustre/llite/*/extents_stats_per_process" and "/proc/fs/lustre/llite/*/offset_stats".If the job is accessing other file systems, the agent will insert a temporary SystemTap [8] module into Linux kernel. This module filters and counts file operations on the VFS layer. Since IO pattern information is needed for all the cluster jobs, not only the training samples, the agent has been integrated into the IHEP job statistics system.

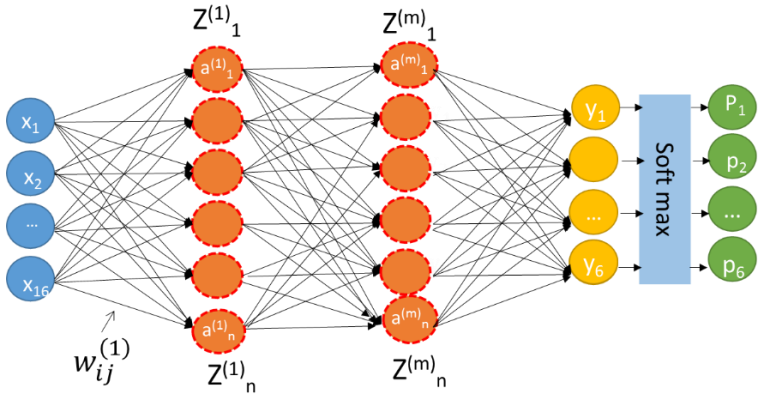
2.1.2. Semi-automatic job labelling

To label millions of training data samples is a laborious work. In this paper, a semi-automatic method is adopted. According to our experience, part of the cluster users has the habit of specifying the job types in job names. We designed a way to leverage this information to build the training dataset.

Firstly, we manually specified a key word array for each of the six job types: *analysis*, *simulation, reconstruction, calibration, skim* and *scan*. Then for each job in the training set, we chuck the job name by slashes and spaces into a string array. Number prefixes and suffixes are skipped during this procedure. After that a key word matching process is done between the string array and the six key word arrays. If a job hits 0 or more than one job type, it will be skipped. Jobs that hit only one type is selected and tagged as a training data sample. By this method, about 320K job samples was tagged. The training data set used in this paper is built with these samples.

2.2. Model definition

The deep neural networks model used in this paper is a feed forward neural network, shown in figure 4. It includes an input layer, several hidden layers and an output layer. The input layer has 16 unites, corresponding to each element of the IO pattern vector. The output layer has 6 unites, representing the six job types respectively. Data flows in the direction of black arrows. The hidden layers can be treated as intermediate mathematic transformations from the input layer to output layer. The circles in hidden layers are hidden unites Z. Each Z, is the result of a linear function of the previous layer, as shown in equation (1). Each dashed circle stands for an activation function, which is a tanh function of Z, as shown in equation (2). There is a soft-max function, shown in equation (3), applied to the output layer. The output of this function is a vector of probabilities. The model chooses the type with largest probability as its decision for an IO pattern specified in input layer.



$$z = b + \sum_{i=1} w_i x_i \qquad (1)$$

$$a = \frac{1-e^z}{1+e^z} \qquad (2)$$

$$P_i = \frac{e^{y_i}}{\sum_{j \epsilon group} e^{y_j}} \qquad (3)$$

$$C = -t_i \log P_i \qquad (4)$$

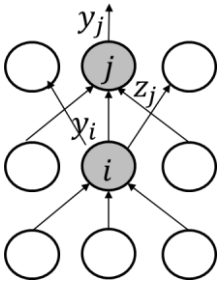
Figure 4. Definition of the deep neural networks**.**

2.2.1. Cost function

The cost function is a measurement of the difference between the result generated by the model and the labelled job type. It is a cross entropy function as shown in equation (4). t_i equals 1 if the sample is tagged as type i , in other cases, t_i equals 0. If the result of the model is very close to the tagged value, the value of cost function should be very small. Therefore, the optimization of the model is actually the process of minimizing the sum of cost functions for all the training samples.

2.2.2. Back propagation

The $W_{ij}^{(k)}$ parameters are the variables of the model. They are initialized randomly. If we group all the parameters of each hidden layer into a single parameter vector, this vector is the only reason that caused the difference between the model results and tagged values. If we tune this vector step by step in its gradient direction carefully, at least a local optimal value of cost function can be reached. To get the gradient direction, the derivatives of C over each $W_{ij}^{(k)}$ are needed. Figure 5 and equation (5) to (7) shows how to compute these derivatives simultaneously. This method was called back propagation [9] in the domain of neural networks.



$$\frac{\partial C}{\partial z_j} = \frac{dy_j}{dz_j}\frac{\partial C}{\partial y_j} = y_j(1-y_j)\frac{\partial C}{\partial y_j} \quad (5)$$

$$\frac{\partial C}{\partial y_i} = \sum_j \frac{dz_j}{dy_i}\frac{\partial C}{\partial z_j} = \sum_j w_{ij}\frac{\partial C}{\partial z_j} \quad (6)$$

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial C}{\partial z_j} = y_i\frac{\partial C}{\partial z_j} \quad (7)$$

Figure 5. Back propagation of error derivatives.

2.3. Data pre-processing

According to initial observation of the training set, the number of different file operation varies in a large range. In that case, it is very difficult for the training process to converge. To solve this problem, firstly, we made a square root of the input matrix. Experiment data shows that this step can improve the final precision by about 2%. Then we made a normalization for each dimension. Finally, we divided the dataset into three parts, 60% samples as training set, 20% as cross validation set and 20% as test set.

2.4. Model training

2.4.1. Mini-batched gradient descend

The model training is done by an algorithm called mini-batched gradient descent [6], shown in Figure 6. It computes the derivatives of cost function over the parameter vector with a fraction of training set, and updates the vector in the gradient direction by very little number. The number is the product of each derivative and a number ∂, called learning rate. Suppose the size of training set is m, batch size of gradient descend is n. After one iteration of training, the parameter vector will be updated m/n times. If n equals m then the algorithm is called batched gradient descend. For a large dataset, batched gradient descend will make the training very slow, because every update of parameter vector requires a summation over the whole input matrix. If n equals 1, the algorithm is called stochastic gradient descend. According to our test, n=1 is the best choice for our task. It gets ~2% better precision than n=8, 16, 32. The parameter ∂ should be carefully tuned during the training process. If it is too large, the training process will overshot the optimal point. If it is too small, the training coverages very slow, or it will

struck in the local optimal. According to experience, we prefer a large ∂ at the beginning of training, and a smaller ∂ when we approach the optimal point. Therefore, at the start of training, we use ∂ equals 1e-3, and we set a decay of ∂ after each mini-batch step as 1e-7.

repeat until converge {

b=batch_size

for s=0: data_size, b {

$$w_j := w_j - \alpha \sum_{k=s:s+b} \left(\frac{\partial C}{\partial w_j}\right)^{(k)}$$

// k is the index of training sample

// W_j is the parameter of the model

// α is the learning rate

}}

Figure 6. The mini-batched gradient descend algorithm.

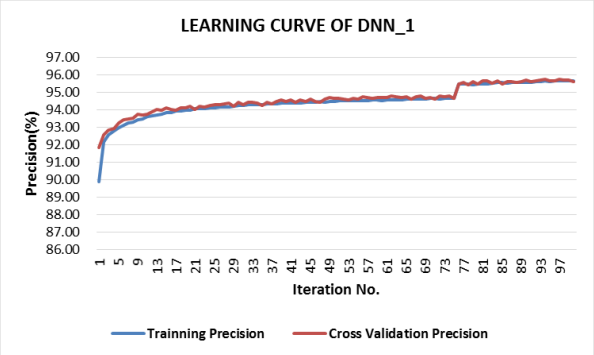


Figure 7. Learning curve of a one hidden layer DNNs model training.

2.4.2. Problems of over-fit

While training a machine learning model, the problem of over-fit should be carefully considered. Over-fit means that the model gets a good performance on the training set while it has a bad performance on a separated data set. The essential reason is that the model is too complex and the training set is too small. Therefore, the model not only represent the true relationship between input and output, but also learns the noise of training set.

The machine learning community provides a bag of tricks to avoid over-fitting on deep neural networks, which includes weight decay, drop-out and auto encoder [9]. For the training of models which have millions of input features and 10s of hidden layers, these tricks have been proved very useful. However, the learning curves shown in figure 7 shows that the problem of over-fit is not so significant. Because the curve of training dataset and cross validation dataset are almost over-lapped. Therefore, tricks to avoid over-fit problem have not been utilized for current task. To save space, we only present the curve of DNN-1 model which has one hidden layer. The situation is similar for other DNN models of during the training process.

3. Test Results

We used a 64000 sample test set to evaluate the performance of our DNNs models. As comparison, we also trained a simple linear model with the same training set, and tested it with same test set. To show the impact of number of hidden layers, results of 1-5 hidden layer neural networks are all presented.

3.1. Test environment

Programs of this work are implemented with Torch 7 [10], a scientific computing framework with wide support for machine learning algorithms. Evaluation results were got on a 12 Core Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz, 16 GB RAM server.

3.2. Precision

Figure 8 shows the precision result with the linear model and all the DNNs models:

1. The DNNs models outperform the linear model by at least 6%.
2. There is not significant performance gain after DNN-2. That means, for current task, a two-hidden-layer neural networks is enough to model the relationship between IO pattern and job type.

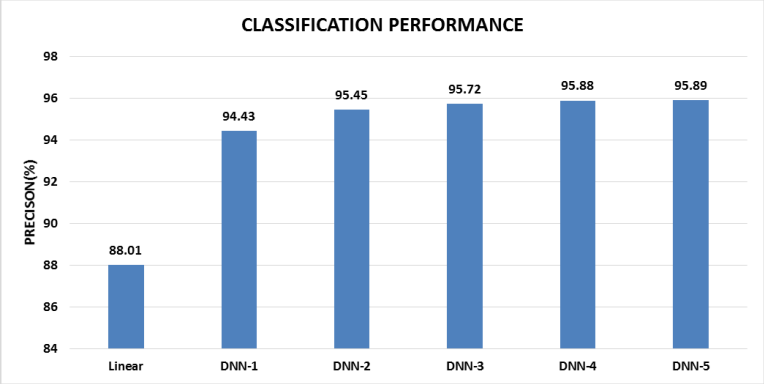


Figure 8. Classification performance of different models

3.3. Confusion matrix

Table 1 is a confusion matrix which shows how the test set are classified with DNN_5 model. The larger the number on diagonal the better.

Table 1. The confusion matrix of DNN_5 model. Average row correct is 92.74 %, average rowUcol correct (VOC measure) is 88.26%, global correct is 95.88 %.

	Analysis	**Recon.**	**Simulation**	**Calibration**	**Scan**	**Skim**	**Row Precision**
Analysis	8843	50	509	0	0	2	94.034
Recon.	50	35540	316	105	0	2	98.687
Simulation	895	567	16428	2	0	0	91.818
Calibration	0	134	2	622	0	0	82.058
Scan	0	4	0	0	124	0	96.875
Skim	6	5	2	0	0	172	92.973

Row precision is the ratio of samples which are right classified for a certain job type. For example, reconstruction jobs get the highest row precision 98%, while calibration jobs get the lowest row precision 82%. That means it is more difficult for the model to classify a calibration job than a reconstruction jobs. The non-diagonal elements stand for the miss- classified cases. For example, the intersection of calibration row with reconstruction column stands for numbers of calibration samples which are miss-classified as reconstruction jobs.

From result of confusion matrix, we can see that to get better performance of calibration, scan, and skim jobs, more training samples of these three types are still needed.

3.4. Time consumption

Figure 9 shows time consumption for one iteration of the training dataset in seconds. Consumption of linear model and DNN-1 are almost the same. Time consumption doubled from DNN-1 to DNN-2, tripled from DNN-1 to DNN-5. Taking into account of the precision gain, a DNN-2 model is good enough for current use case. Torch provides libraries to take advantage of multi-core CPU architectures and GPU accelerated servers. For current case, we find that the multi-thread training mode doesn't help decreasing the time consumption unless we increase the batch size. According to tests in 2.4.1, increasing batch size will decrease the final precision, therefore, those advanced functions have not been leveraged for current task.

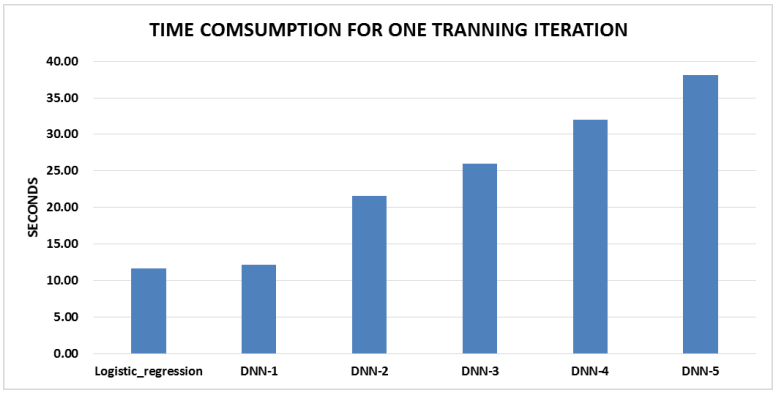


Figure 9. Time consumption of one training iteration, data set size is 192000.

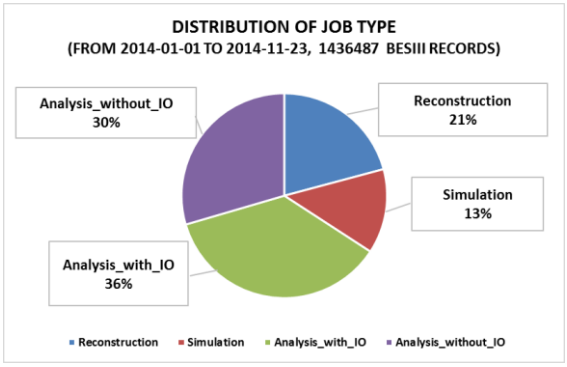


Figure 10. Distribution of BESIII jobs.

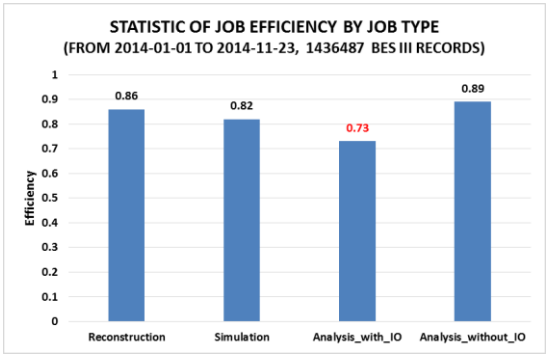


Figure 11. Statistic of BESIII job efficiency.

3.5. Integration with Job statistics

We have integrated the DNNs classification model with our job statistic system. As presented before, every running job will be instrumented by an agent. Then valid IO patterns will be sent to a classification server which has the Torch program running at the backend. By applying the IO patterns to the DNN-2 model, the program will generate a classification result, and the server will forward it back to the client agent. The client agent will then insert both the IO pattern and job type into the statistic database.

With data collected for more than 6 months, we generated a distribution of about 1.4 million BESIII jobs, shown in figure 10. Among these 1.4 million jobs, there are 60 percent analysis jobs, 21 percent reconstruction jobs and about 18 percent simulation jobs. The rest three types took too small fractions, therefore have not been displayed on this figure. Then we made a statistic of job efficiency for each job type shown in figure 11. Analysis jobs with IO operations have the lowest efficiency. It means that there is still optimization space for backend file system and the analysis framework to optimize the IO operation of analysis jobs. We have also embedded a problematic IO behavior detector in the instrumenting agent. Misbehaved jobs with unrecognized IO pattern and extremely low efficiency will be detected and an alarming E-mail will be sent to system administers by the agent.

4. Conclusion and future work

This paper implements the idea of applying DNNs to HEP job classification. DNNs models are leveraged to discover the relationship between IO patterns and job type. An instrumenting agent is developed and deployed on every client node to get IO pattern of running cluster jobs. Training dataset is labelled automatically by key word matching. Meta-parameters were carefully tuned with cross

validation. A Precision of 96% was achieved by a 5-hidden-layer DNN model. DNNs models outperform the linear model by at least 6%. Currently, we have integrated the DNN-2 model with the job statistic system, interesting fine-grained statistics have been generated.

To adapt to the variation of IO pattern, continuous training set collection and model update is needed. Fortunately, the gradient descend algorithm provides naturally support for this on-line learning requirement. As the training dataset grows, GPU acceleration should be taken into consideration for the next step. With experience of DNNs training, we are planning to apply this technology to a wide range of cluster administration cases, such as log analysis, fault prediction, scheduling optimization and so on.

References

[1] Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786):504-507.
[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *Proceedings of Workshop at ICLR*, 2013.
[3] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton.ImageNet Classification with Deep Convolutional Neural Networks. *NIPS 2012*: 1106-1114.
[4] Dong Yu, Li Deng, Frank Seide. Large Vocabulary Speech Recognition Using Deep Tensor Neural Networks. *INTERSPEECH 2012*: 6-9.
[5] Baldi, P., Sadowski, P., & Whiteson, D.. Searching for exotic particles in high-energy physics with deep learning. *Nature communications,* 5 (2014).
[6] Machine learning. Andrew Ng. Coursera course: https://www.coursera.org/course/ml
[7] Lustre File Systrem Manual. https://wiki.hpdd.intel.com/display/PUB/Documentation
[8] Jacob B, Larson P, Leitao B, et al. SystemTap: instrumenting the Linux kernel for analyzing performance and functional problems [J]. IBM Redbook, 2008.
[9] Neural Networks for Machine Learning. Geoffery Hinton. Coursera course: https://class.coursera.org/neuralnets-2012-001
[10] Collobert, Ronan, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning.*BigLearn, NIPS Workshop.* No. EPFL-CONF-192376.2011.