Distributed Machine Learning Using Data Parallelism on Mobile Platform

Máté Szabó

Department of Information Technology, Faculty of Informatics
University of Debrecen, Kassai út 26, H-4028 Debrecen, Hungary
E-mail: szabo.mate@inf.unideb.hu

Received 12 May 2020; Accepted 17 July 2020;
Publication 08 September 2020

Abstract

Machine learning has many challenges, and one of them is to deal with large datasets, because the size of them grows continuously year by year. One solution to this problem is data parallelism. This paper investigates the expansion of data parallelism to mobile, which became the most popular platform. Special client-server architecture was created for this purpose. The software implementation of this problem measures the mobile devices training capabilities and the efficiency of the whole system. The results show that doing distributed training on mobile cluster is possible and safe, but its performance depends on the algorithm's implementation.

Keywords: Machine learning, distribution, data parallelism, mobile, client-server architecture, web service.

1 Introduction

Machine learning is a very popular area in the informatics, because most of the applications and services use it directly or indirectly. The application of machine learning techniques can be found in nearly every discipline, for

Journal of Mobile Multimedia, Vol. 16_3, 317–334.
doi: 10.13052/jmm1550-4646.1633
© 2020 River Publishers

example they are used in healthcare, cars, politics, and commerce. In healthcare, it can be used to predict diseases or medical parameters based on the collected data. To be more concrete, researchers use machine learning to predict cancer [1], to improve the diagnosis of ischemic heart disease [2], for hippocampal shape analysis [3], general disease prediction [4]. In other fields, there are many special applications, like facial expression recognition [5] or automated traffic classification [6]. There is a need for machine learning techniques' evolution, because the size of the datasets is growing continuously year by year. Traditional machine learning has its limits, especially in handling big data, because the complexity and size of datasets grow more than the hardware's computing capacity. Of course, there are several methods for handling large datasets with traditional machine learning techniques, like the clouds decision tree [7], Bayesian networks [8], support vector machines [9].

Mobile phones became the most popular devices, and because of this popularity, the hardware of them evolves quite fast. Each year, the flagship mobile's computing capacity grows significantly [10, 11], and most of them are even capable of using machine learning models. There is no doubt that the popularity of these devices will grow more in 10 years [12–14]. The problem with the increasing number of mobile devices is that most of time, their processors and GPU-s are idle, so their users use only a fragment of the computing capacity.

This paper focuses on the utilization of mobile processors for machine learning purposes, especially with large datasets. My aim was to create an architecture, which is capable of processing machine learning models trained on phones. There is a reference implementation of this architecture, so the performance and other properties are measured. This paper deals with challenges from multiple areas such as transferring models between devices, verifying the received models and reconstructing them, and training neural networks on mobiles. One of the most important parts of this system is the web service, which can handle incoming machine learning models, persist them in the database, construct ensemble models, and slice datasets. The other important part is the mobile application, which can train simple neural networks, use trained models, and communicate with the web service. Constructing ensemble models is an advantage of this system because with them, the accuracy can be increased [15]. For handling large datasets, this system uses data parallelism, so each device can train a model on a part of the data. For the challenge of transferring these models to the server, I used byte array serialization, because in this form, the received model can be verified and then deserialized into the same model.

2 Related Works

The present work can be placed in many topics, like distributed machine learning, mobile machine learning, and machine learning model exchange formats.

There are many model exchange formats, and each of them approaches the problem differently. Predictive Model Markup Language (PMML) is an XML format for predictive models [16]. In this language the model is defined by a header, a data dictionary, data transformations, the model itself, the mining schema, the targets and the outputs. It supports multiple model types, like neural networks, Gaussian processes, Bayesian networks, support vector machines. ONNX is another project to share machine learning models between frameworks. It defines a computation graph model, built-in operator and standard data types. The supported frameworks are Caffe2, Chainer, Cognitive Toolkit, MXNet, PyTorch, PaddlePaddle, Matlab, SAS, NNL by Sony. It defines converters for Keras, Tensorflow, scikit-learn, Apple Core ML, Spark ML, LightGBM, libsvm, XGBoost [17, 18].

There are many projects for machine learning on mobile too. A popular usage of neural networks on android is image recognition and manipulation, but these models can be used for detecting malware too [19–21].

Using machine learning models through web services is quite popular nowadays, because there are many devices, which can't train a model by themselves, so they must access pretrained ones [22].

The distributed machine learning is a very popular topic, and there are many unique ways achieve it. There are implementations with parameter servers, where the distributed computations update the model parameters on a defined server [23], and there are special systems like MLBase [24], what defines an own architecture for distribution, or MLI [25], what defines an API, or more widely used solutions, like Apache Spark's MLlib [26].

3 Fundamentals of the System

The created software distributed machine learning on mobile is shown in Figure 1. The center of this application is a Spring framework [27] based web service, which manages the datasets and the trained models too. It communicates with a database, which will contain the datasets and the models. The mobile devices can communicate with this service, so they can get data slice, trained models, and ensemble models from it and they can send their trained neural networks back.

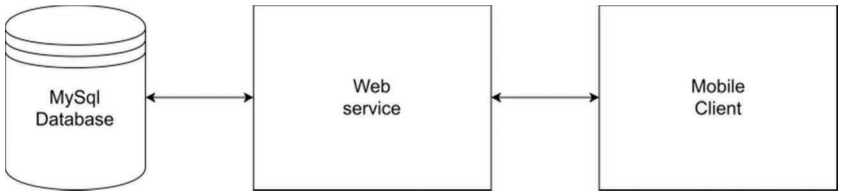

Figure 1 The relationship of the system's components.

3.1 Web Service

The web service is the most complicated part of this system, because it includes methods for managing datasets, processing data, converting data, persisting trained models, retrieving trained models, creating ensemble models and handling Hypertext Transfer Protocol (HTTP) requests. It is important that this part of the system was written in Java using the Spring framework. Of course, this software can be implemented in any programming language, which can handle HTTP requests and responses and has a common data exchange format with the Android system. In our software, the data on both server and client is represented by java byte array.

The data access layer of the web service uses Spring Data to run SQL command in the database. In this layer, there are two sublayers, named entities and repositories. Entities are the Java side representation of the database objects, so the Spring framework will know what kind of data it can access in the database. The repositories define interfaces containing operations like create, read, update and delete, so the framework will use these for generating a complete query with the given entity types.

The service layer is in the middle of the web service software, so it communicates with the data access layer and with the endpoints' layer. It contains the business logic of the software, which includes many functions, like converting database entities to other data types, slicing bigger dataset into smaller ones, creating ensemble models and forwarding database related requests to the data access layer. The conversion of database entities is necessary, because the structure of the objects in the database differs from the object structure needed by the mobile clients. The slicing of the dataset can be managed in many ways, for example the user can request an n large slice of the dataset containing random records from it. In this layer there is a domain sublayer that defines the classes from the android application, so the database entity can be converted to objects from the domain.

The top layer of the web service contains the endpoints and some of the functionalities. Classes defined here have similar tasks like handling HTTP

requests, generating HTTP responses, validating the given JSON web token and beside that they have unique tasks too. The architecture of the web service is shown in Figure 2.

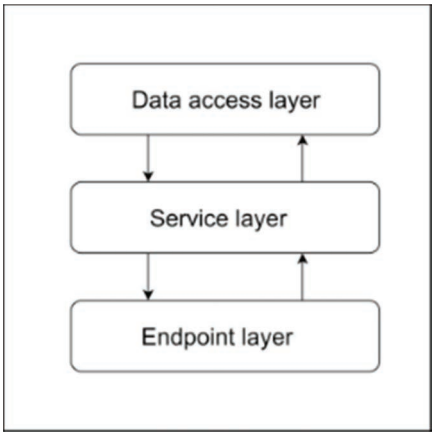

Figure 2 The layer structure of the web service.

For example, with the ensemble endpoint, the user creates an ensemble model with the given type and the given models, or it can query persisted ensemble models from the database, so it can use these models on the client side. This layer has one of the core functionalities that are the conversion of the models to byte array, so it will be easier to send through the network and it can be converted back to a model object on the client side. The list of the defined endpoint is shown in Table 1.

Table 1 Endpoints of the web service

Prefix	Endpoint	Method	Description
/data	/slice	GET	Retrieve an n size dataset slice
/data	/get-dataset	GET	Retrieve the whole dataset
/data	/get-dataset-attributes	GET	Retrieve the attributes of the given dataset
/data	/list-dataset	GET	Retrieve the list of available datasets
/data	/remove-dataset	DELETE	Remove the selected dataset
/model	/list-model	GET	Retrieve model list for given dataset
/model	/get-model	GET	Retrieve a single model by identifier
/model	/save-model	POST	Save the model specified in the body
/model	/remove-model	DELETE	Remove the selected model

The functions of the web service software includes dataset managing, data processing, data conversion, dataset slicing, persisting trained models, retrieving trained models and creating ensemble models. The dataset managing function is about accessing the datasets, which are persisted in the database. Because the variety of parameters the models can have, they must be persisted in a common form, which can be the byte array of them. The data model in Figure 3 shows that the database can work with multiple datasets by having a table for the attributes and the data-types of them and each record of this table references to the related record of the dataset table. With this solution, the web service can query the database for dataset files and the information for reading it can be accessed with a query to the attribute table.

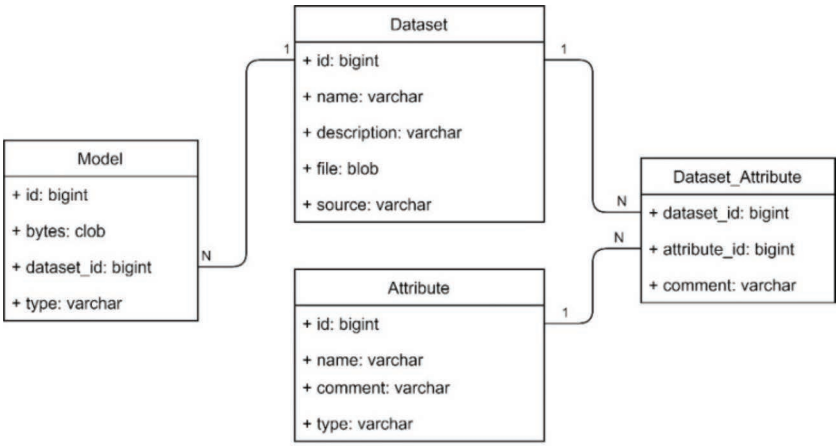

Figure 3 Database structure.

The data processing function is about preparing the queried dataset to be in the proper state for training. The transformation of the dataset can be customized, so the user can access the Deeplearning4J library's functions. The user can apply data reduction or can change the scale of the data; it can use sampling, dimensionality reduction, feature extraction, discretization or binarization. These features are using the Deeplearning4J's corresponding functions, like Principal Component Analysis (PCA). The data slicing function is about getting an n size random sample from the preprocessed dataset, so it relies on the data processing package.

Model handling is the most important function of the web service, because the created software's main feature is the usage of trained models from the network. The first model handling operation was persisting, so the

service can persist the models trained on mobile devices. Here, when a model arrives to the server, it will try to deserialize it from byte array to simple Java object. If this process is successful, then the software knows that the sent byte array is a valid model, so it can be converted into database entity and then it can be persisted too. The flow of the model data is shown in Figure 4.

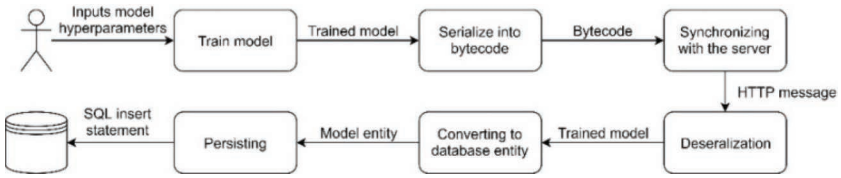

Figure 4 Flow of the model data from mobile to database.

The second model handling operation is retrieving, where the user can query for models from the database. When a specific HTTP request arrives to the server with the number of the model, it will query the database for it; then it will be serialized into byte array and then sent back to the client, where the byte array can be deserialized into simple Java object what will be a trained model. The serialization is a quite simple process to create a compact form of a Java object. The initial state is when there is a trained model on the server, and this model has attributes for hyper-parameters and for the state of the model. These values will form a JSON object, which can be handled as a simple text. This Java text object can be transformed into byte array, so it will be in a more compact and verifiable format, which can be used for model data exchange. The deserialization includes the same steps, just in the opposite way. The flow of the model data is shown in Figure 5.

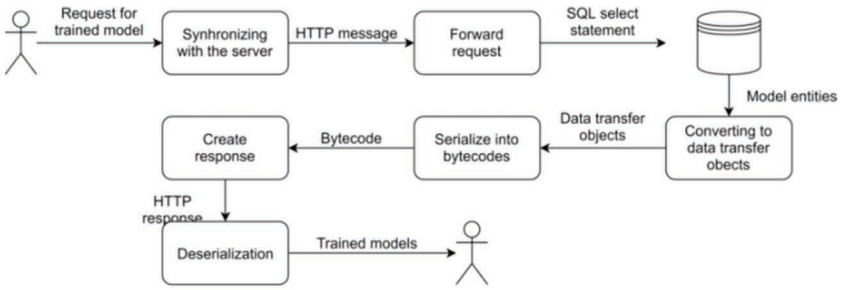

Figure 5 Flow of the model data when retrieving a model.

The last model handling operation is creating the ensemble models based on the earlier persisted models. The mobile client can send HTTP request

to get earlier persisted models for a dataset. From this list the user can select models and then send another HTTP request with the models' id as parameters. With this data, the web service queries the models from the database, then creates a selected type of ensemble model with it. This model can be persisted in the database and will be sent back to the mobile client. The sending works the same as sending regular models, so the ensemble model will be serialized into byte array and the mobile client can use it after a deserialization.

3.2 Database

This part of the software consists of a simple relational MySQL database, which is created for managing trained models and datasets along with information about their data structure. The database model is shown in Figure 3. The most important table is the model, which contains the machine learning model's generated id, the byte array of it, the dataset's id and the type of the model. In this representation, the user can select models by types or datasets. The related table is the dataset, which includes the name, source, file and the description of it. The dataset consists of attributes, which are represented in the attribute table. It has an identifier, a name, a type and a comment, so the user can search datasets by attributes. The relation between the attribute and the dataset table is represented in the dataset_attribute joining table.

3.3 Mobile Client

This part of the software consists of two layers and it has the most important functionalities like training and evaluating models. The main used technologies here are Retrofit for HTTP communication, Room database for a local database on Android, and Deeplearning4j for machine learning.

The first layer of the mobile client is created for the services. It is connected with the web service's endpoint layer and the client's presentation layer. The classes in the service layer define the core functionalities of the application, so the user can send http requests and process responses from here and it can process the given dataset and train models on them. When the user of the application wants to use a trained model, it will call a function from the service layer to send a request to the web service, what will send back the byte array of the model. This byte array will be deserialized and then can be used as a trained model.

The presentation layer communicates with the user itself and the service layer, so it will react to the user's interactions and forward them to the service

layer. Here there are only files that define the user interface and the logic behind it.

The mobile client's main task is to train models and then send them back to the web service. For this task, each layer in the web service and the client will be used, so the data goes through the whole system. The first operation needed for training is retrieving the dataset, for what the client will use the web service's slice endpoint, which is created for sending dataset slices. When the client has the subdataset, the user can start training the chosen model from Deeplearning4j. The model can be customized with a given seed number, optimization algorithm, learning rate, regularization, and of course custom layers can be added too. After the model training process, the trained model will be converted into byte array and will be sent to the server where it will be persisted.

Of course, there are a lot of limits for this mobile client because by default Deeplearning4J was not created for mobiles, so it does not support many Android features. A fine example for this is that Deeplearning4J's model training and other operations are not thread safe so it cannot utilize all of the mobile's resources, mostly the processor. The other problem with the mobile platform is the lack of memory, because machine learning often needs a lot of memory, especially with bigger datasets. That is why the slice operation was introduced, so the mobile devices can do training on smaller datasets, and the result can be synchronized to the server.

The trained models are not sent to the server automatically, it is a user choice. There is an evaluation process on the mobile client, so the user can check the created model. The scores of accuracy, precision, recall and F1 can be written to the screen. With the evaluation, the user can see the confusion matrix, false positive/negative rate, true positive/negative, class counts, F-beta, G-measure, Matthews correlation coefficient.

The last function of the mobile client is to reuse the trained models from the database to create ensemble models. This can be achieved from the user interface, where the user can retrieve the list of the models for the selected dataset. For communication with the web service, the mobile application's service layer uses the list-model endpoint. Here the user can select some models and then an ensemble method. When the ensemble model is created, it can be evaluated with the evaluation service. If the user finds the model good enough, it can be sent to the web service to persist it in the database. For saving, the mobile will use the web service's save-model endpoint. The persisted model can be reused later by other mobile devices.

4 Learning on Mobile with Data Parallelism

Most machine learning systems use some kind of parallelism when they have to deal with bigger datasets. They can be separated into model and data parallelism. In this work, model parallelism was not used, so this software is about utilizing mobile processors with a fully data parallel machine learning system.

4.1 Data Parallelism

Data parallelism is a popular way to introduce parallelism to software because it is fairly easy to implement, and it is independent from the chosen machine learning model. Data parallelism is not only for machine learning, although it is obvious to use it for handling big data or reducing computation need [28, 29]. Basically, this method is about having one bigger dataset, what will be used for machine learning and from it multiple subdatasets can be created. It is important to note that if the size of the subdataset is too small, it possibly contains too few information for a machine learning system, but the current system does not warn the user about the dataset size. These smaller datasets are easier to process, so they can be processed on separate processors or devices. The result of the training on the subdataset will be a smaller model; it can be named as a submodel, which contains information about the subdataset. These submodels can be synchronized to a server for aggregation. The output of the synchronization process will be a bigger model combined from the submodels, so it will contain information about each smaller subdatasets. The basic data parallelism can be seen in Figure 6.

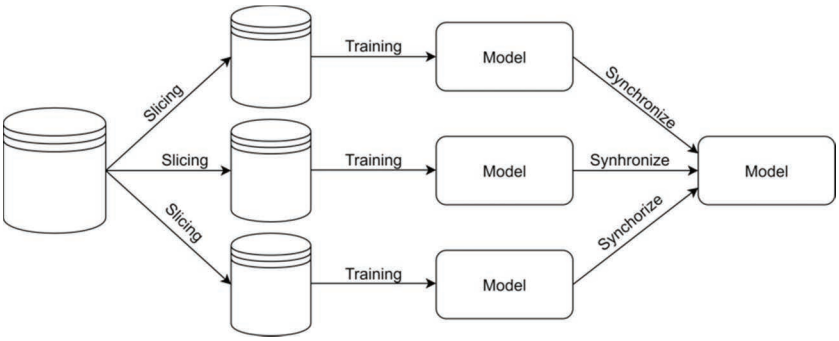

Figure 6 The simplified data parallelism.

4.2 Data Parallelism with Mobile

Using data parallelism with mobile is only an extension of the basic data parallelism, so the main steps are the same. The problem here is handling bigger datasets too, but the computing capacity here is in the mobile devices, not simple processors. The first step is the slicing of the data, what will be resulted in multiple smaller dataset. While creating smaller datasets, it is more important to care about the size of the resulted datasets, because beside the lower size limit, the mobile devices have upper size limit too. If the subdataset is too large, the mobile device cannot process it because of the lack of memory in these devices. With the introduction of the mobile devices, the subdatasets will be sent to the mobiles through the network. The training process will start here on these devices and when the training is over, the model can be sent back to the server with the synchronization process, where they can be aggregated. The output here will be a bigger model combined from the submodels, just as in simple data parallelism. So basically, this method involves the new synchronization process to the mobile, but with it, the summed computing capacity will be bigger, because mobile processors can run training processes on each cores. The mobile extension of data parallelism is shown in Figure 7.

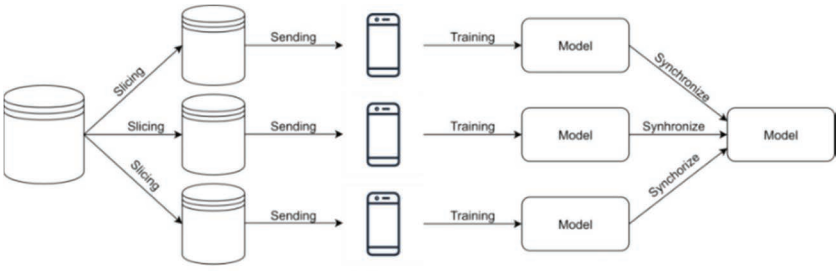

Figure 7 Extension of data parallelism to mobile.

5 Synchronization

The synchronization had many problems, which had to be solved. The biggest one was that to use the trained model on the server, the client and the server should use the same machine learning library. This compatibility problem was solved with Deeplearning4J, because it is written in Java, so Android devices and web services can use it too. The other problem was that sending complex model through the network can be problematic. The solution for this was the serialization of models to byte array, so they act like compressed

messages which can be deserialized on the other size. The serialization process starts with the trained model, what is a Java object that can be converted into JSON object. This object includes the hyperparameters and the state of the model in a special text format. This text object will be converted into byte array, because it is more compact and easier to verify. When software uses network to send and retrieve data, it has to prepare for data loss. The message can arrive malformed, partially missing or it's possible that it does not even arrive. The byte array serialization is a solution for this problem too, because if the byte array is malformed or partially missing, it cannot be deserialized on the other side, so the software will know about the error. The last discovered problem of the serialization is that its performance depends on the network speed, so sending models and data will be slower on slower networks.

6 Measurements

An implementation was created for the mentioned software architecture, so measurements can be made with it. The web service used Spring Boot, Hibernate and Deeplearning4J to implement the endpoints and the mobile client used Room database, Retrofit and Deeplearning4J. For comparing the training time on a simple computer and on this software, the dataset was 'Record Linkage Comparison Patterns Data Set' from the Epidemiological Cancer Registry of North Rhine-Westphalia and the UCI Machine learning repository [30, 31]. This is a simple dataset for classification with 12 attributes and 5749132 records. The reason for choosing this is because the number of records is large, and it was sure that it cannot be processed on mobile devices without the slicing. The dataset is about comparing two records based on the agreement of first name, family name, sex, and the date of birth's day, month, and year components separately. The same Deeplearning4J code ran on the desktop and on mobile too, so they trained the network with the same parameters. The evaluation of the desktop and mobile models shows that both of them had nearly the same results as seen on Table 2. This accuracy was computed for a 10000 large subdataset.

Table 2 Accuracy and best score on mobile and computer

Environment	Accuracy	Score at Best Epoch
Mobile	0.7907	4.81924921875
Computer	0.7907	4.81931338500

The accuracy of the submodels is lower than the bigger model, which was trained on the entire dataset; however ensemble models constructed from the

smaller ones have as good accuracy as the bigger model. In numbers, the accuracies were about 0.79 for submodels, 0.99 for ensemble and the model trained on the full dataset.

The creators of Deeplearning4J noted that this library can show weak performance on Android and that was confirmed by my measurements too. The concrete runtime of 1 epoch based on the size of the dataset can be seen in Figure 8. The X-axis shows the number of records in the dataset slice and

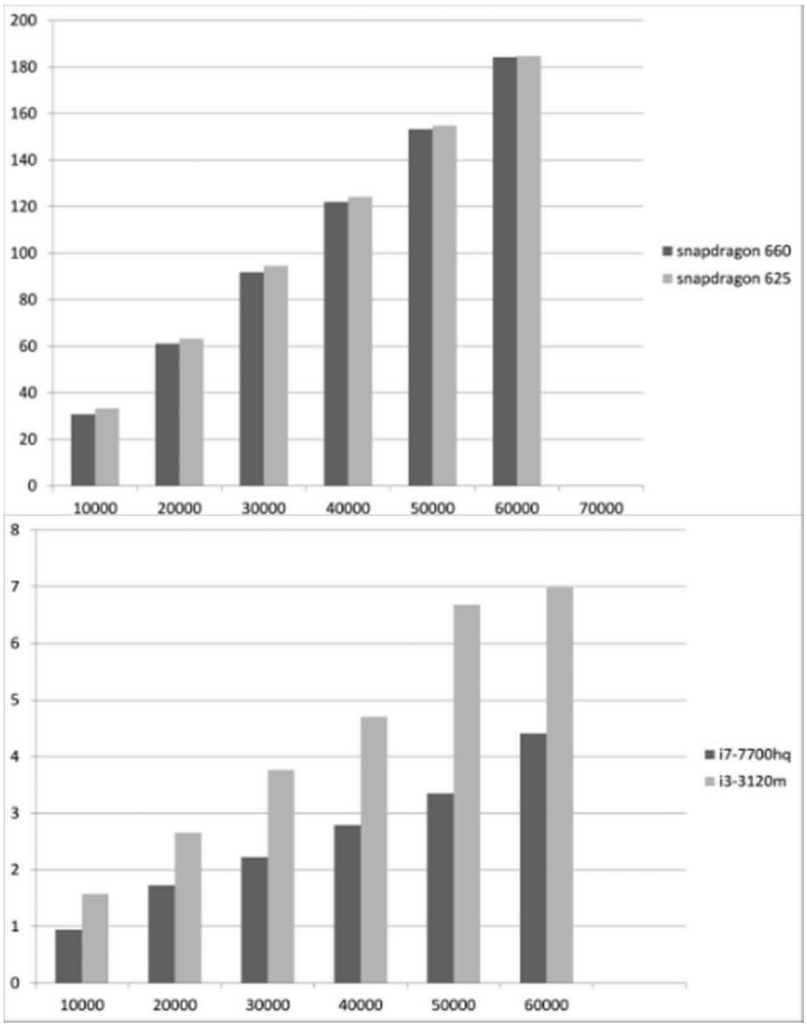

Figure 8 Runtime of 1 epoch based on the size of the dataset on (a) mobile and (b) desktop.

the Y-axis shows the runtime of 1 epoch in seconds. The figure shows that these mobile processors can handle datasets containing smaller than 70000 records. In Figure 8(a) it can be seen that the training speed is quite slow on the mobile platform with the given Deeplearning4J version, so it can process only smaller datasets effectively. Figure 8(b) shows that the training speed is quite faster on a desktop environment.

7 Possible Improvements and Conclusion

The measurements showed that the training speed with Deeplearning4J on mobile as of 1.0.0-beta5 is quite slow and is limited by the not thread safe machine learning library, however the created models' accuracy, precision and recall values were similar to the simple models. The android machine learning support mainly focuses on running pretrained models rather than training on mobile processors, so the support of it is very limited. Overall, it can be seen that the data parallelism on mobile can work well, so these devices can be part of a cluster environment for training models. The architecture is easy to implement and it was created to be extendible.

It is important to note that because of the web service that handles all of the models and the data, additional methods should be added to ensure the security of the system. It could be extended with some kind of role based access control.

The implementation could be improved in many ways, for example the slow training process can be solved with another machine learning library, which is available on mobile too, so it should be changed, maybe for Tensorflow.

The architecture itself can improve to support the multiplatform distribution, so training could be done either on mobile or a simple processor, the results can be aggregated. This may increase the computing capacity, but it can generate a massive load on the server. This load could be handled with redesigning the architecture to support microservices with multiple servers.

For future work, I would like to involve more platforms and models from different environments, so the system won't depend that much from the Java language.

Acknowledgement

The work is supported by the EFOP-3.6.1-16-2016-00022 project. The project is co-financed by the European Union and the European Social Fund.

References

[1] Kourou K., Exarchos T. P., Exarchos K. P., Karamouzis M. V., Fotiadis D. I. Machine learning applications in cancer prognosis and prediction, Computational and Structural Biotechnology Journal, Vol. 13, 2015, pp. 8–17.

[2] Kukar M., Kononenko I., Groselj C., Kralj K., Fettich J. Analyzing and improving diagnosis of ischaemic heart disease with machine learning, Artificial Intelligence in Medicine, Vol. 16, No. 1, 1999, pp. 25–50.

[3] Li S., Shi F., Pu F., Li X., Jiang T., Xie S., Wang Y. Hippocampal shape analysis of Alzheimer disease based on machine learning methods, American Journal of Neuroradiology, Vol. 28, No. 7, 2007, pp. 1339–1345.

[4] Chen M., Hao Y, Hwang K., Wang L., Wang L. Disease prediction by machine learning over big data from healthcare communities, IEEE Access, Vol. 5, 2017, pp. 8869–8879.

[5] Bartlett M. S., Littleworth G., Frank M., Lainscsek C., Fasel I., Movellan J., Recognizing facial expression: machine learning and application to spontaneous behavior, Conference on Computer Vision and Pattern Recognition, San Diego, CA, USA, 20–25 June 2005, pp. 568–573.

[6] Zander S., Nguyen T., Armitage G. Automated traffic classification and application identification using machine learning, The IEEE Conference on Local Computer Networks 30th Anniversary, Sydney, NSW, Australia, 17 November 2005, pp. 250–257.

[7] Alsabti K., Ranka S., Singh V. Clouds: A decision tree classifier for large datasets, Proceedings of the 4th Knowledge Discovery and Data Mining Conference, New York, USA, 27–31 Aug. 1998, pp. 2–8.

[8] Lee S. M., Abbott P. A. Bayesian networks for knowledge discovery in large datasets: basics for nurse researchers, Journal of Biomedical Informatics, Vol. 36 No. 4–5, 2003, pp. 389–399.

[9] Yu H., Yang J., Han J. Classifying large data sets using SVMs with hierarchical clusters, Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, D.C., USA, 24–27 August 2003, pp. 306–315.

[10] PassMark Android Benchmark Charts, CPU Rating, https://www.androidbenchmark.net/cpumark_chart.html, (last visited 22 December 2019).

[11] Smartphone Processors, Benchmark List, NotebookCheck.net Tech, https://www.notebookcheck.net/Smartphone-Processors-Benchmark-List.149513.0.html, (last visited 22 December 2019).

[12] Mobile percentage of website traffic 2019, Statista, https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/, (last visited 22 December 2019).
[13] U.S. daily mobile media usage time 2018, Statista, https://www.statista.com/statistics/469983/time-spent-mobile-media-type-usa/, (last visited 22 December 2019).
[14] Mobile vs desktop traffic in 2019, Perficient DigiDigital, Perficient Digital Agency, https://www.perficientdigital.com/insights/our-research/mobile-vs-desktop-usage-study, (last visited 22 December 2019).
[15] Seni G., Elder J. F. Ensemble methods in data mining: improving accuracy through combining predictions. Synthesis lectures on data mining and knowledge discovery, 2010, 2.1 pp. 1–126.
[16] Guazzelli A., Lin W. C., Jena T. Taylor J. PMML in action: Unleashing the power of open standards for data mining and predictive analytics, CreateSpace, Paramount, CA, 2010.
[17] Lin W. F., Tsai D. Y., Tang L., Hsieh C. T., Chou C. Y., Chang P. H., Hsu L. ONNC: A compilation framework connecting ONNX to proprietary deep learning accelerators, IEEE International Conference on Artificial Intelligence Circuits and Systems, Hsinchu, Taiwan, Taiwan, 18–20 March 2019, pp. 214–218.
[18] GitHub – onnx/onnx: Open neural network exchange, https://github.com/onnx/onnx, (last visited 22 December 2019).
[19] Amos B., Turner H., White J. Applying machine learning classifiers to dynamic android malware detection at scale, 9th International Wireless Communications and Mobile Computing Conference, Sardinia, Italy, 1–5 July 2013, pp. 1666–1671.
[20] Peiravian N., Zhu X. Machine learning for Android malware detection using permission and API calls, IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, 4–6 November 2013, pp. 300–305.
[21] Sahs J., Khan L. A machine learning approach to android malware detection, European Intelligence and Security Informatics Conference, Odense, Denmark, 22–24 August 2012, pp. 141–147.
[22] Jovanović, Željko, et al. Java Spring Boot Rest WEB Service Integration with Java Artificial Intelligence Weka Framework. International Scientific Conference UNITECH 2017. 2017. pp. 323–327.

[23] Li M., Andersen D. G., Park J. W., Smola A. J., Ahmed A., Josifovski V., Long J., Shekita E. J., Su B. Y. Scaling distributed machine learning with the parameter server, 11th Symposium on Operating Systems Design and implementation, Broomfield, CO, 6–8 October 2014, pp. 583–598.

[24] Kraska T., Talwalkar A., Duchi J., Griffith R., Franklin M. J., Jordan M. MLbase: A distributed machine-learning system, 6th Biennial Conference on Innovative Data Systems Research, Asilomar, California, USA, 6–9 January 2013.

[25] Sparks E. R., Talwalkar A., Smith V., Kottalam J., Pan X., Gonzalez J., Franklin M. J., Jordan M. I., Kraska T. MLI: An API for distributed machine learning, IEEE 13th International Conference on Data Mining, Dallas, TX, USA, 7–10 December 2013, pp. 1187–1192.

[26] Meng X., Bradley J., Yavuz B., Sparks E., Venkataraman S., Liu D., Freeman J., Tsai D. B., Amde M., Owen S., Xin D., Xin R., Franklin M. J., Zadeh R., Zaharia M., Talwalka A. Mllib: Machine learning in apache spark, The Journal of Machine Learning Research, Vol. 17, No. 1, 2016, pp. 1235–1241.

[27] Johnson, Rod, et al. The spring framework-reference documentation. interface, 2004, 21: 27.

[28] LI, Hao, et al. Malt: distributed data-parallelism for existing ml applications. In: Proceedings of the Tenth European Conference on Computer Systems. 2015. pp. 1–16.

[29] Xing, Eric P., et al. Petuum: A new platform for distributed machine learning on big data. IEEE Transactions on Big Data, 2015, 1.2, pp. 49–67.

[30] UCI machine learning repository: Record linkage comparison patterns data set, https://archive.ics.uci.edu/ml/datasets/record+linkage+comparison+patterns, (Last visited 22 December 2019).

[31] Sariyar M., Borg A., Pommerening K. Controlling false match rates in record linkage using extreme value theory, Journal of Biomedical Informatics, Vol. 44, No. 4, 2011, pp. 648–654.

Biography

Máté Szabó received his MSc degree in computer science from the University of Debrecen in 2018. His research interests are machine learning, web services and distributed systems. He worked as a freelancer web application developer, but currently he is PhD student and lecturer at the University of Debrecen.