Efficient Management and Optimization of Very Large Machine Learning Dataset for Question Answering

Marek Medveď, Radoslav Sabol, and Aleš Horák

Natural Language Processing Centre
Faculty of Informatics, Masaryk University
Botanická 68a, 602 00, Brno, Czech republic
{xmedved1,xsabol,hales}@fi.muni.cz

Abstract. Question answering strategies lean almost exclusively on deep neural network computations nowadays. Managing a large set of input data (questions, answers, full documents, metadata) in several forms suitable as the first layer of a selected network architecture can be a non-trivial task. In this paper, we present the details and evaluation of preparing a rich dataset of more than 13 thousand question-answer pairs with more than 6,500 full documents. We show, how a Python-optimized database in a network environment was utilized to offer fast responses based on the 26 GiB database of input data. A global hyperparameter optimization process with controlled running of thousands of evaluation experiments to reach a near-optimum setup of the learning process is also explicated.

Keywords: question answering; dataset management; machine learning; optimization

1 Introduction

Current hardware and software architectures for neural network computations are capable of processing tens of thousands of input data units relatively fast, especially in a situation of distributed processing. However, a bottleneck of such processing can lie in copying the input data between the computing machines. Imagine a set of hundreds of possible answers to a question with each answer as a set of 500-dimensional word vectors including a selected broader context of the answer. Then in each training epoch the computations need to engage thousands of such questions from the training set. In case of inefficient storage and data transfer, such dataset can occupy hundreds of gigabytes which can, of course, negatively influence the training process running time.

In this paper, the details of the data processing in the answer selection task with the Simple Question Answering Database (SQAD [1,2]) are presented. The training and testing process is further enhanced by semi-automatized search of optimal hyperparameters setup.

1.1 Data Processing in Related Works

Question answering (QA) systems are particularly developed for the mainstream languages where a number of datasets are published. A well-known example of such dataset is the Stanford Question Answering Database (SQuAD [3]) that was used in more than 80 state-of-the-art works.

In [4], Patel et al enriched the Stanford's JSON formatted data with sentence embeddings (trained on large English corpus) using the InferSent tool [5] by the Facebook research team. The semantic representations of the sentences are then used to evaluate semantic distance between possible answers and questions.

In [6], Park presented indexing all words inside the Stanford database into an internal vocabulary and represented all SQuAD records as a list of word indexes to the internal vocabulary. Finally, Park enriched all words within the vocabulary by GloVe [7] word vectors.

In [8], Tiutiunnyk decided to store a new Ukrainian QA dataset in the PostgreSQL database to allow their system direct access to all the dataset records.

1.2 ZODB Database System

ZODB[1] is a transparent Python-object persistence (database) system that is able to store selected data models (especially classes and objects in the Python source codes). With comparison to standard relational database systems such, the ZODB database system is able to store huge hierarchical structures that are not limited to one data type. Hierarchical databases on the other hand do not support transactions and are bound with all the restrictions resulting from the relational model.

In the current implementation of the Czech SQAD database, the ZODB system is used as the main storage of all the dataset data. This approach differs from the standard approaches used with the Stanford SQuAD database mentioned above. Details of the ZODB engagement are further presented in Section 2.

1.3 Hyperparameter Optimization

The most straightforward approach to neural network hyperparameter optimization is the standard *grid search* – a technique that involves manual choice of possible values for each optimized hyperparameter and generating all possible combinations. This can be computationally expensive as each added hyperparameter increases the number of required evaluations exponentially. Grid search is still available as an option in some major machine learning libraries (e.g. scikit-learn [9] or Optuna [10]).

Random search is designed to replace the exhaustive enumeration by taking random samples of hyperparameter values in the search space. The number of *trials* is typically limited by a pre-defined constant, see e.g. HyperOpt [11] or Optuna [10].

[1] `http://www.zodb.org/en/latest/`

Original text:

Ngoni (někdy též n'goni) je strunný hudební nástroj oblíbený v západní Africe. Někdy bývá označován jako primitivní předchůdce banja. Velikostí se ale podobá spíše ukulele. ...
[Ngoni (also called n'goni) is a string musical instrument popular in west Africa. It may be considered as a primitive predocessor of banjo. But according to its size it is more similar to ukulele. ...]

Question:

Jakého typu je hudební nástroj ngoni?
[What kind of musical instrument is ngoni?]

Answer:

strunný hudební nástroj
[string musical instrument]

URL:

`https://cs.wikipedia.org/wiki/Ngoni`

Author:

login

Question type:

ADJ_PHRASE

Question type:

OTHER

Answer selection:

Ngoni (někdy též n'goni) je strunný hudební nástroj oblíbený v západní Africe.
[Ngoni (also called n'goni) is a string musical instrument popular in west Africa.]

Answer extraction:

strunný hudební nástroj
[string musical instrument]

Fig. 1. Example of the `SQAD` record No. 012878

The main downside of the aforementioned techniques is that they do not utilize the information from past trials. The *Sequential model-based optimization* (SMBO) tries to overcome this limitation by iteratively selecting the hyperparameters from the search space using a probabilistic model to minimise/maximise an objective function. The most commonly used probabilistic models are Tree-structured Parzen Estimators (in Optuna [10], HyperOpt [11], RayTune [12]) and Gaussian processes (in GPyOpt [13]). One of the less commonly used models are Gradient Boosting (in scikit-learn [9]) and Random Forests (in TuneRanger [14]).

2 Managing Very Large Machine Learning Dataset

In this section, the implementation of the Czech SQAD dataset storage using the ZODB system is presented. ZODB allows fast access to the SQAD data and efficiently stores all data from the SQAD database raw records in the final form required by the AQA question answering system [15].

The ZODB database system is able to store Python objects without a need of extra format conversion, ZODB loads Python objects directly from the database.

word	lemma	tag
Ngoni	Ngoni	k1gNnSc1
(	(	kIx(
někdy	někdy	k6eAd1
též	též	k9
...		

Fig. 2. An example of the vertical format of POS-tagged text in SQAD.

word	NE tag
...	
přestoupil	O
do	O
Sparty	B
Praha	I
...	

Fig. 3. Example of *Link named entity* training data: `O` – regular word, `B` – beginning of named entity, `I` – continuation of named entity.

Currently, ZODB is used to store the complete SQAD database records.[2] In addition to raw POS-tagged texts, the SQAD database contains all important information derived from texts. In the source form [16], the SQAD database consists of several files per record. The data files within the SQAD database contain morphologically and lexically analyzed text in vertical format [3] that are converted into the SQAD-ZODB (fusion of SQAD data and ZODB database system) database.

Along with the original information stored in the SQAD database, the new SQAD-ZODB database contains several additional information that are important for different modules inside the AQA system. These additional features are automatically computed from the original source data:

– *word vectors* – to boost the training procedure in the AQA answer selection module the new SQAD-ZODB database stores pre-computed word vectors pre-trained from large Czech corpora using the word2vec algorithm. In the current version, the SQAD-ZODB database stores 100-, 300- and 500-dimensional word vectors of each word. This allows a fast access to word vectors and flexibility in the training process of the answer selection module.
– *list of sentences containing exact answer* – during building SQAD-ZODB, the list of sentences that contain the exact answer is computed. This information is then used in the evaluation process of the answer selection module.
– *list of similar answers* – TF-IDF similarity scores between all sentences withing the record full text are computed. These scores are then used to fine tune

[2] See a SQAD record example in Figure 1.
[3] See Table 2 for an example.

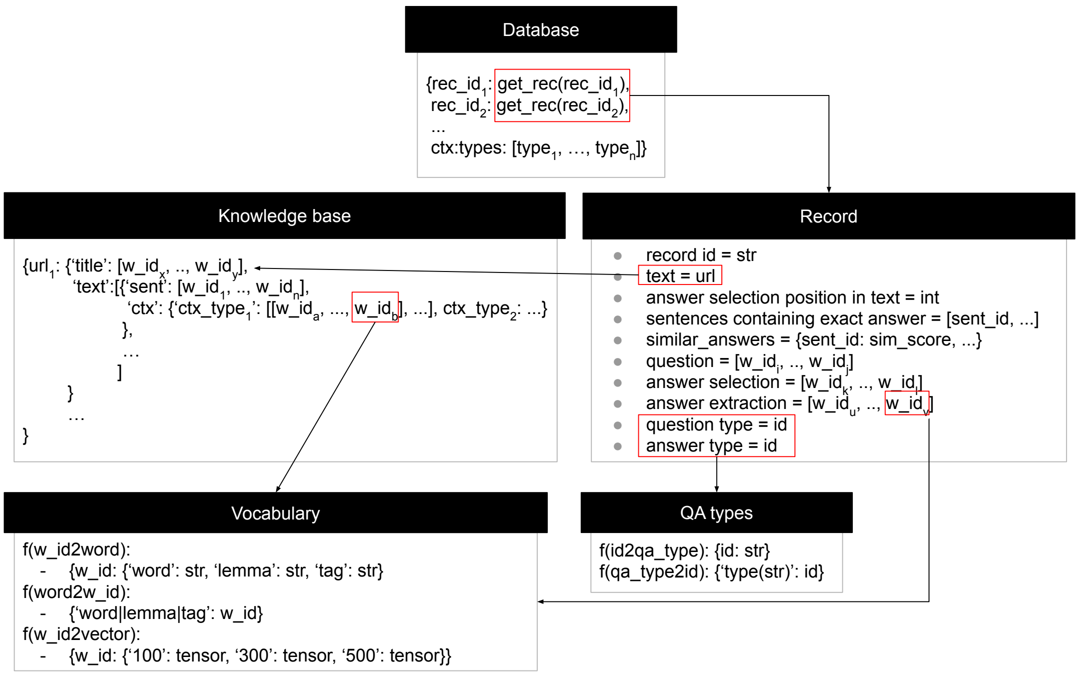

Database
{rec_id1: get_rec(rec_id1),
rec_id2: get_rec(rec_id2),
...
ctx:types: [type1, ..., typen]}
Knowledge base
{url1: {'title': [w_idx, .., w_idy],
'text':[{'sent': [w_id1, .., w_idn],
'ctx': {'ctx_type1': [[w_ida, ..., w_idb], ...], ctx_type2: ...}
},
...
]
}
...
}
Record
record id = str
text = url
answer selection position in text = int
sentences containing exact answer = [sent_id, ...]
similar_answers = {sent_id: sim_score, ...}
question = [w_idi, .., w_idj]
answer selection = [w_idk, .., w_idl]
answer extraction = [w_idu, .., w_idv]
question type = id
answer type = id
Vocabulary
f(w_id2word):
- {w_id: {'word': str, 'lemma': str, 'tag': str}
f(word2w_id):
- {'word|lemma|tag': w_id}
f(w_id2vector):
- {w_id: {'100': tensor, '300': tensor, '500': tensor}}
QA types
f(id2qa_type): {id: str}
f(qa_type2id): {'type(str)': id}

Fig. 4. The database structure of SQAD-ZODB.

the training setup of the answer selection module where the most similar negative sentences are used to boost the module ability to identify the correct answer within a list of very similar sentences.

– *answer context* – the task of identifying the main answer sentence is difficult mainly due to anaphoric expressions which "hide" the relevant entities by pronominal references. To supplement the neural network decision process, an information about the sentence context is provided to the answer selection module. To speed up the whole training process, the SQAD-ZODB database contains several types of context pre-computed from the original data. In the current version, the SQAD-ZODB database contains three types of contexts (more context types are to be added in the future work):
  - previous sentences – the context of N full sentences is added to each input article sentence.
  - phrases from previous sentences – using the rule-based SET parser [17], the system is able to identify all possible noun phrases within each sentence. M noun phrases from each of N preceding sentences are stored as the second context type.
  - "link named entities" from previous sentences form the third type of sentence context. See details in Section 2.1.

2.1 Link named entities

Link named entities (LNEs) are a specific type of standard named entities with regard to the information often expressed in questions and answers. LNEs are defined as entities that are labeled with Wikipedia internal links. Inside each Wikipedia article, links that refer to other Wikipedia articles identify entities which are often significant in denoting an important piece of information. LNEs are identified in general texts by training a named-entity recognition (NER) system[4] with the whole Czech Wikipedia, where for each sentence all link named entities are marked, see Table 3 for an example. The final NER module is applied to the SQAD database and provides information about recognized link named entities which are used as a sentence context.

2.2 The SQAD-ZODB Architecture

The architecture of the current SQAD data in the ZODB database system is displayed in Figure 4. At the first level, the SQAD-ZODB database stores all records IDs and a function that builds the record content form 4 database parts (tables).

The *Record* object stores ten most critical information. Each *Record ID* is a unique identifier of a SQAD record.

The *Text* variable contains a unique *URL* that points to specific article inside the *Knowledge base* table. Thus for multiple records concerning one Wikipedia article, the database do not need to store that same article twice.

The *Answer selection position* stores an index of the sentence that contains the expected answer (used in the training part of the answer selection module).

Sentences containing exact answer is a list of sentence IDs that contain the exact answer (used in the training phase of the answer selection module where the system excludes these sentences as negative examples).

Similar answers is a list of similar sentences with their similarity scores (used to train the module to distinguish the correct answer within a list of very similar sentences).

Question, *answer selection*, *answer extraction* are lists of words IDs. Each word ID can be transformed into word, lemma, POS tag, 100-, 300- or 500-dimensional vector using the *Vocabulary table*.

The last two record features *question type* and *answer type* are also IDs pointing to specific question and answer type using the *QA types table*.

The *Knowledge base* table stores all articles used within the SQAD database. Avoiding duplicates and storing only list of the words IDs makes the knowledge base compact while maintaining all important information.

2.3 Updating the SQAD-ZODB Database

Due to the database transaction support in ZODB, updating the database is a straightforward task. After establishing a connection to the database, a user can add new records or add new features to existing records. In the SQAD development process, each new feature is a standalone script that can supplement the database with a single new feature of each record. The current transformation system consist of:

[4] The BERT-NER from `https://github.com/kamalkraj/BERT-NER` is currently used.

Table 1. Running times (in seconds) for a random sample of 100 queries. w – word, l – lemma, t – morphological tag, v1 – 100-dimensional vector, v3 – 300-dimensional vector, v5 – 500-dimensional vector.

Preloaded vocabulary						Not preloaded vocabulary					
init	12.44	w;l;t	1.63	w;l;t;v5	3.13	init	5.74	w;l;t	2.86	w;l;t;v5	5.05
w	13.21	w;l;v1	3.00	w;l;v1;v3	3.40	w	7.17	w;l;v1	4.58	w;l;v1;v3	5.43
l	2.02	w;l;v3	2.99	w;l;v1;v5	3.40	l	2.25	w;l;v3	4.58	w;l;v1;v5	5.45
t	1.42	w;l;v5	3.00	w;l;v3;v5	3.41	t	2.25	w;l;v5	4.59	w;l;v3;v5	5.47
v1	4.78	w;t;v1	3.00	w;t;v1;v3	3.40	v1	7.32	w;t;v1	4.75	w;t;v1;v3	5.46
v3	2.61	w;t;v3	3.03	w;t;v1;v5	3.42	v3	3.34	w;t;v3	4.63	w;t;v1;v5	5.47
v5	2.61	w;t;v5	2.58	w;t;v3;v5	3.01	v5	3.33	w;t;v5	4.59	w;t;v3;v5	5.45
w;l	1.53	w;v1;v3	3.30	w;v1;v3;v5	3.68	w;l	2.76	w;v1;v3	4.95	w;v1;v3;v5	5.77
w;t	1.95	w;v1;v5	3.28	l;t;v1;v3	3.40	w;t	2.41	w;v1;v5	4.95	l;t;v1;v3	5.47
w;v1	2.91	w;v3;v5	3.29	l;t;v1;v5	3.44	w;v1	4.11	w;v3;v5	4.95	l;t;v1;v5	5.49
w;v3	2.86	l;t;v1	3.01	l;t;v3;v5	3.41	w;v3	4.15	l;t;v1	4.60	l;t;v3;v5	5.45
w;v5	2.48	l;t;v3	3.01	l;v1;v3;v5	3.70	w;v5	4.10	l;t;v3	4.65	l;v1;v3;v5	5.76
l;t	1.94	l;t;v5	2.98	t;v1;v3;v5	3.71	l;t	2.75	l;t;v5	4.61	t;v1;v3;v5	5.76
l;v1	2.88	l;v1;v3	2.89	w;l;t;v1;v3	3.50	l;v1	4.11	l;v1;v3	4.94	w;l;t;v1;v3	5.87
l;v3	2.87	l;v1;v5	3.30	w;l;t;v1;v5	3.09	l;v3	4.11	l;v1;v5	4.96	w;l;t;v1;v5	5.92
l;v5	2.92	l;v3;v5	3.28	w;l;t;v3;v5	3.54	l;v5	4.13	l;v3;v5	4.98	w;l;t;v3;v5	5.89
t;v1	2.60	t;v1;v3	3.30	w;l;v1;v3;v5	3.80	t;v1	4.11	t;v1;v3	4.95	w;l;v1;v3;v5	5.88
t;v3	2.96	t;v1;v5	3.33	w;t;v1;v3;v5	3.81	t;v3	4.11	t;v1;v5	4.98	w;t;v1;v3;v5	6.25
t;v5	2.88	t;v3;v5	3.30	l;t;v1;v3;v5	3.81	t;v5	4.11	t;v3;v5	4.60	l;t;v1;v3;v5	6.24
v1;v3	3.08	v1;v3;v5	3.55	w;l;t;v1;v3;v5	4.03	v1;v3	4.24	v1;v3;v5	5.09	w;l;t;v1;v3;v5	6.86
v1;v5	3.06	w;l;t;v1	2.72			v1;v5	4.27	w;l;t;v1	5.08		
v3;v5	3.06	w;l;t;v3	3.08			v3;v5	4.22	w;l;t;v3	5.05		
total time					3m 38s	total time					5m 9s

- *sqad2zodb* transforms all records from the original SQAD source to the SQAD-ZODB database and adds pre-computed vectors to each word.
- *add_similar_sentences* enhances each record with the list of similar sentences.
- *add_sentences_containing_exact_answer* adds a list of sentences that contain the expected answer to the record.
- *context_previous_sentences* for each sentence in the article adds *N* preceding sentences as a context (where *N* is a user-defined parameter).
- *context_noun_phrases* for each sentence in the article adds *M* phrases for each of *N* preceding sentences.
- *context_ner* for each sentence in the article adds all link named entities recognized in *N* preceding sentences.

That is how each new record feature can be developed and tested separately.

2.4 SQAD-ZODB Performance

In the transformation process from SQAD to SQAD-ZODB, the database interface allows the training workers to transfer only those pieces of information that are required for the training process. The choice of the right information can greatly influence the data transfer running times. Table 1 summarizes the times needed to transfer different parts of the record.

The running times are proportional to the amount of data that need to be transferred. The efficiency of the ZODB storage is depicted by comparing the space requirements of the SQAD database in three formats are presented in Table 2.

2.5 SQAD-ZODB over Network

The SQAD-ZODB database is particularly used in the answer selection module. The training and hyperparameter optimization of this module requires a large amount of training setups to be tested. To speed up the training by distributing the process to multiple GPU-based servers, the SQAD-ZODB database was implemented within the ZEO[5] (Zope Enterprise Objects) library that allows to run the database in the client-server mode over network.

3 Large-scale Optimization of Machine Learning Hyperparameters

When training machine learning models, hyperparameter optimization is one of the key steps required to achieve acceptable performance. However, this process requires considerable efforts, especially in large search spaces of hyperparameter values. A variety of tools and libraries were developed to automate this process, employing sophisticated algorithms to achieve the task.

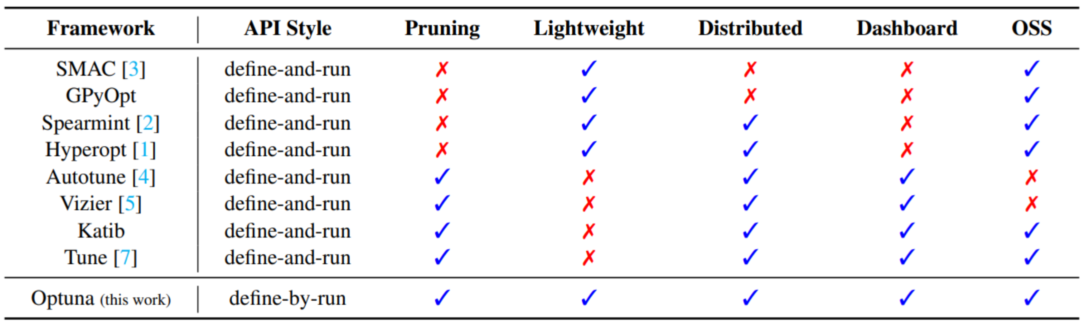
3.1 About Optuna

Optuna [10] is a relatively new hyperparameter optimization framework that aims to provide a simple setup for defining hyperparameter search spaces

[5] `http://www.zodb.org/en/latest/articles/old-guide/zeo.html`

Table 2. Disk usage for various storage methods. *Plain text* refers to the original plain text form of SQADv3 with all the pre-computed vectors. *Pickle* is a serialized dataset (using the Python `pickle` library) with only the necessary data to train models using the 500-dimensional embeddings.

Representation	Disk usage
Plain text	1,312.89 GB
Pickle	240.20 GB
SQAD-ZODB	25.08 GB



Framework	API Style	Pruning	Lightweight	Distributed	Dashboard	OSS
SMAC [3]	define-and-run	✗	✓	✗	✗	✓
GPyOpt	define-and-run	✗	✓	✗	✗	✓
Spearmint [2]	define-and-run	✗	✓	✓	✗	✓
Hyperopt [1]	define-and-run	✗	✓	✓	✗	✓
Autotune [4]	define-and-run	✓	✗	✓	✓	✗
Vizier [5]	define-and-run	✓	✗	✓	✓	✗
Katib	define-and-run	✓	✗	✓	✓	✓
Tune [7]	define-and-run	✓	✗	✓	✓	✓
Optuna (this work)	define-by-run	✓	✓	✓	✓	✓

Fig. 5. Comparison of hyperparameter optimization frameworks in terms of available features [10].

while being highly customizable. The parameter search spaces are defined using the *define-by-run* approach which allows the search spaces to be created and adjusted dynamically at runtime.

The Optuna framework is highly scalable from simple experimental computations to large-scale distributed optimizations. In order to accomplish this flexibility, Optuna supports many result storage forms like in-memory database, SQLite databases, or PostresSQL databases.

Optuna also includes a support for pruning algorithms that monitor intermediate values for the objective, and terminate a trial if a user-defined condition is not met. This premature termination is useful for saving time from unpromising trials.

3.2 Optuna and Answer Selection

In this section, we present improved results for the AQA answer selection task achieved using the Optuna library. Overall, 1,507 setups were trained in a fully automated fashion. In order to define the search space, a set of hyperparameters and their values was identified as affecting the model performance as displayed in Table 3.

The objective to maximize was the Mean Average Precision (MAP) of each trial. Out of the 1,506 trials, 455 were succesful (reached the evaluation on

Table 3. Hyperparameter values search space for the answer selection model

Hyperparameter name	Optuna distribution used	Range of values
BiGRU hidden size	discrete uniform	100-600 with the step of 20
Dropout	discrete uniform	0.0-0.6 with the step of 0.1
Batch size	categorical	1, 2, 4, 8, 16, 32, 64, 128, 256
Optimizer	categorical	Adam, Adagrad, Adadelta, SGD
Learning rate	logarithmic uniform	from 10^{-4} to 10^{-1}
Embedding dimension	categorical	300, 500

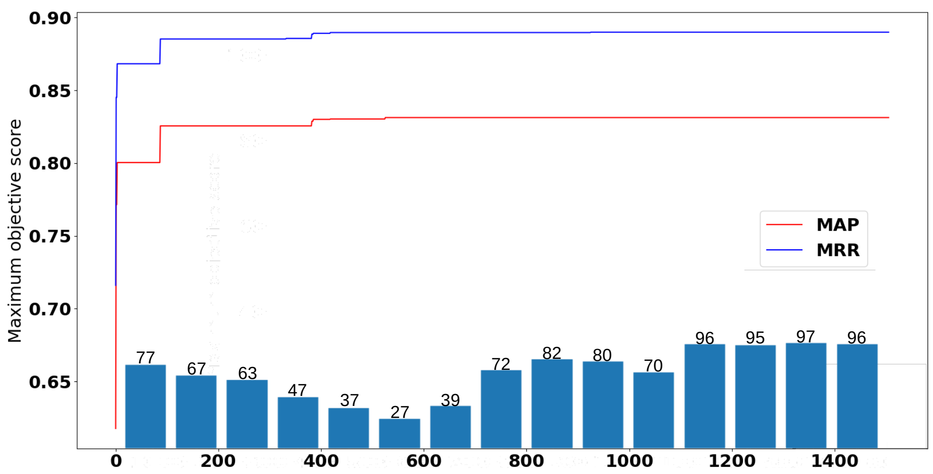


Fig. 6. Increase in MAP and MRR measured over 1,506 recorded runs. The histogram displays the amount of errors/prunes in the groups of 100.

test set), 365 were pruned, and 686 were errors due to GPU out-of-memory exception.

The most successful setup reached MAP of **83.13** and MRR of **88.99**, which is an increase of **0.8** percent when compared to last published result (MAP of **82.33**). The best trial's setup uses the embedding dimension of 300, BiGRU hidden size of 520, the dropout probability of 0.3, the batch size of 4 (with each sentence in the document used), the Adagrad optimizer with the learning rate set to 0.0042.

If we compare the result with the best model setup that uses the 300-dimensional embeddings, the current best setup has achieved an increase of **1.15**% (versus MAP of **81.92**). Unfortunately due to many erroneous trials with 500-dimensional word embeddings, Optuna's search was more focused in optimization with 300-dimensional embeddings instead.

4 Conclusions and Future Work

In the paper, we have presented the details of managing the efficient storage and data transfer of very large question answering dataset. The implementation using the ZODB database framework allows fast data distribution for network-based training and hyperparameter optimization computations.

Using the Optuna parameter optimization framework we have achieved a 0.8 percent MAP increase over the previous published results where the parameters were optimized manually. The future efforts will aim towards decreasing the amounts of erroneous trials. One of the solutions is to carefully select the paramaters in the search space to fit the available GPU memory. In order to eliminate the bias towards lower embedding dimensions, separate studies can be constructed for each embedding dimension available.

Acknowledgements. This work has been partly supported by the Czech Science Foundation under the project GA18-23891S. Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

References

1. Sabol, R., Medveď, M., Horák, A.: Czech question answering with extended SQAD v3.0 benchmark dataset. (2019) 99–108
2. Medveď, M., Horák, A., Sabol, R.: Improving RNN-based answer selection for morphologically rich languages. In: Proceedings of the 12th International Conference on Agents and Artificial Intelligence (ICAART 2020), Valetta, Malta, SCITEPRESS (2020) 644–651
3. Rajpurkar, P., Zhang, J., Lopyrev, K., Liang, P.: Squad: 100,000+ questions for machine comprehension of text. arXiv preprint arXiv:1606.05250 (2016)
4. Patel, D., Raval, P., Parikh, R., Shastri, Y.: Comparative study of machine learning models and BERT on SQuAD (2020)
5. Conneau, A., Kiela, D., Schwenk, H., Barrault, L., Bordes, A.: Supervised learning of universal sentence representations from natural language inference data (2018)
6. Park, D.H.: Question answering on the squad dataset. (2017)
7. Pennington, J., Socher, R., Manning, C.D.: GloVe: Global vectors for word representation. In: In EMNLP. (2014)
8. Tiutiunnyk, S.: Context-based question-answering system for the Ukrainian language. (2020)
9. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12** (2011) 2825–2830
10. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. (2019) 2623–2631
11. Bergstra, J., Yamins, D., Cox, D.D.: Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms, Citeseer (2013)
12. Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J.E., Stoica, I.: Tune: A research platform for distributed model selection and training. arXiv preprint arXiv:1807.05118 (2018)
13. authors, T.G.: Gpyopt: A bayesian optimization framework in Python. `http://github.com/SheffieldML/GPyOpt` (2016)
14. Probst, P., Wright, M., Boulesteix, A.L.: Hyperparameters and tuning strategies for random forest. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery (2018)
15. Medveď, M., Horák, A.: Sentence and word embedding employed in open question-answering. In: Proceedings of the 10th International Conference on Agents and Artificial Intelligence (ICAART 2018), Setúbal, Portugal, SCITEPRESS - Science and Technology Publications (2018) 486–492
16. Horák, A., Medveď, M.: SQAD: Simple question answering database. In: Eighth Workshop on Recent Advances in Slavonic Natural Language Processing, Brno, Tribun EU (2014) 121–128

17. Kovář, V., Horák, A., Jakubíček, M.: Syntactic analysis using finite patterns: A new parsing system for Czech. In: Language and Technology Conference, Springer (2009) 161–171