UniOviedo(Team1) at LeQua 2022: Sample-based quantification using deep learning

Pablo González[1]

[1]*Artificial Intelligence Center at Gijón, University of Oviedo, Spain*

Abstract

Deep neural networks (DNNs) have become very popular in recent years, being applied to a wide variety of problems. In the field of quantification, DNNs also show a promising future as an alternative to conventional quantification methods with some properties that can be useful in certain problems. In this paper we propose a deep learning architecture for quantification problems based on differentiable histograms in order to obtain invariant sample representations. This approach has obtained competitive results in each of the four subtasks of the *LeQua 2022* competition.

Keywords

quantification, prevalence estimation, deep neural networks

1. Introduction

There are real-world applications where predicting the class of each individual example in a dataset has no real utility, and the goal is to be able to estimate the prevalences of the classes in a data sample (i.e. set of examples). Quantification is a supervised machine learning method that tackles this particular problem and has become a task on its own in recent years [1].

Since the recognition of the quantification problem as a self-standing problem and different from classification, several methods to solve it have been devised [2]. The *LeQua 2022* competition [3] was launched with the aim of taking a step further the quantification field and stimulate the creation of new quantification methods. In this competition, the standard quantification methods are to be considered as baselines and the task of the competitors is to obtain solutions to improve those.

The approach proposed by this team is a DNN architecture designed to learn from samples (i.e. sets of unlabelled examples), that can optimize a specific error measure. The four subtasks of the competition: *T1A*, *T1B*, *T2A* and *T2B* were tackled using basically the same setup that will be described in the following section. This solution has led to pretty consistent results in the four tasks, achieving, two silver medals, one bronze medal and one gold medal in the four subtasks respectively. The solution also obtains state-of-the-art results compared with the standard baselines.

CLEF 2022: Conference and Labs of the Evaluation Forum, September 5–8, 2022, Bologna, Italy

gonzalezgpablo@uniovi.es (P. González)
https://pglez82.github.io/ (P. González)
0000-0002-9250-0920 (P. González)

2. Methods

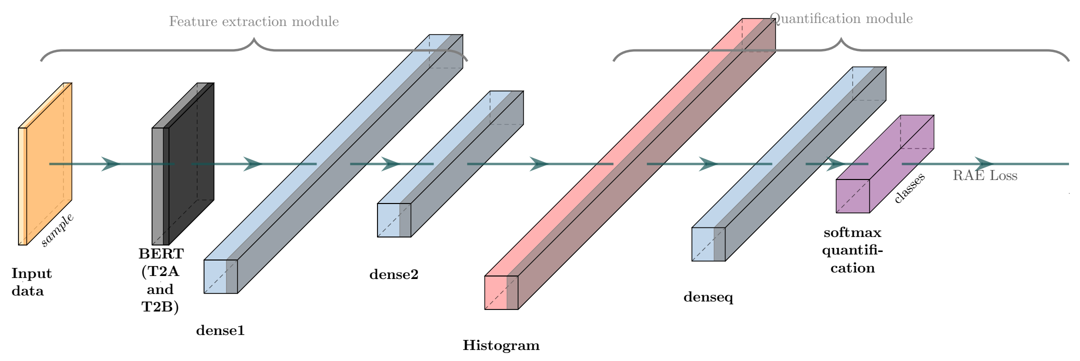

Figure 1: DNN architecture. For tasks *T2A* and *T2B* the feature extraction module uses a *BERT* transformer pretrained. For T1A and T1B this layer is removed. Dense layers number and size depend on the task.

The method used for this competition is a DNN designed for tackling quantification problems. The architecture (see Figure 1) is based on obtaining the representation of the samples using a differentiable histogram layer. This representation is invariant to the order of the examples in the sample. The network consists of a feature extraction layer, that is dependent on the subtask; a histogram layer, that computes a histogram per input feature creating a numeric vector representation of each sample; and a quantification module formed by dense linear layers, able to learn the relationships existent between sample histograms and the sample prevalences. Lastly, one *softmax* activation function translates the outputs of the last fully connected layer (with size equal to the number of classes and specific to the task) into prevalences adding up to one.

The principal advantages of this architecture versus the other approaches are threefold:

- The network can optimize any loss function. In this case, the competition rules established the relative absolute error (RAE) as the official loss function for the competition, making other possible quantification loss functions [4] not important for this particular problem.
- There is no need for labelled examples to train the network. Samples and their associated prevalences are sufficient to train the network.
- The ability to use any existing network architecture in the feature extraction layer (for instance, transformers for natural language processing problems).

2.1. Feature extraction layer

The feature extraction layer depends on the subtask. For tasks *T1A* and *T1B*, in which vector features were already computed for each document, fully connected layers were used. The number of layers and their size were considered as hyperparameters dependent on the subtask. For subtasks *T2A* and *T2B*, raw text documents were the input of the network. In this case, the decision taken was to plug-in *BERT* [5], a model pretrained on English language, from the transformers library [6]. The input of this network is tokenized text (default *BERT* tokenizer was used, with a sequence length limit of 200 tokens), and the network outputs a vector of 768 features per each example that is used to compute the histograms that represent the sample. Note that using *BERT* in a network architecture like this is not exempt from problems, as all the examples in a sample must pass through the network before doing a backward pass. As *BERT* has 110M parameters this operation is not feasible from a memory usage point of view. The solution adopted was to use a technique called *gradient checkpointing* [7] that reduces the memory footprint at the cost of having a 30% increase in computation time.

2.2. Differentiable histograms

One of the problems to overcome is that histograms are not differentiable thus, they can not be used directly in a neural network. The solution is to use an approximation that can be computed with basic and differentiable functions as the *sigmoid* function, provided by all deep learning frameworks.

For computing a histogram of a feature f_k for each example i, first, the fixed bin centres μ_b are computed as well as the values $f_{k,i}$ distance to each bin centre. Then, in a second step, two logistic functions are used to approximate which values fall in each bin (see Figure 2).

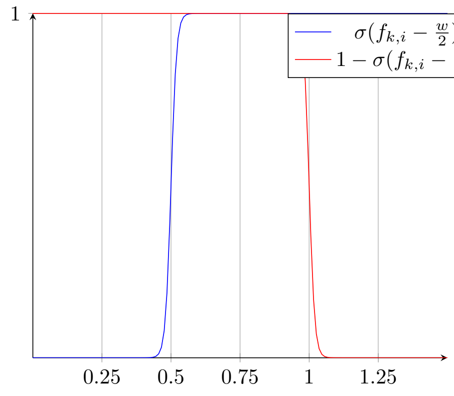

Figure 2: Histogram bin approximation using two sigmoid functions $\sigma(f_{k,i}) = \frac{1}{1+e^{\gamma f_{k,i}}}$. In this example, the bin center is fixed and equal to $\mu = 0.75$. Bin width is also fixed with $w = 0.5$. γ is a constant with a high enough value to make sigmoid functions sharp and closer to a step function.

This method for computing histograms is fast, differentiable (so it can be used with back-propagation), and given a big enough γ value, provides a good-enough approximation to the real histograms. An example of a approximated histogram for a single feature using sigmoid functions can be seen in Figure 3. In this example, the error made by the approximation compared with the true one is close to zero (absolute error: 0.000123).

The number of bins of the histograms is parameter that can be optimized for each problem. For this competition the optimal values were always close to 32, so for simplicity, this is the value that was used for all four tasks (more information in Section 3.1).

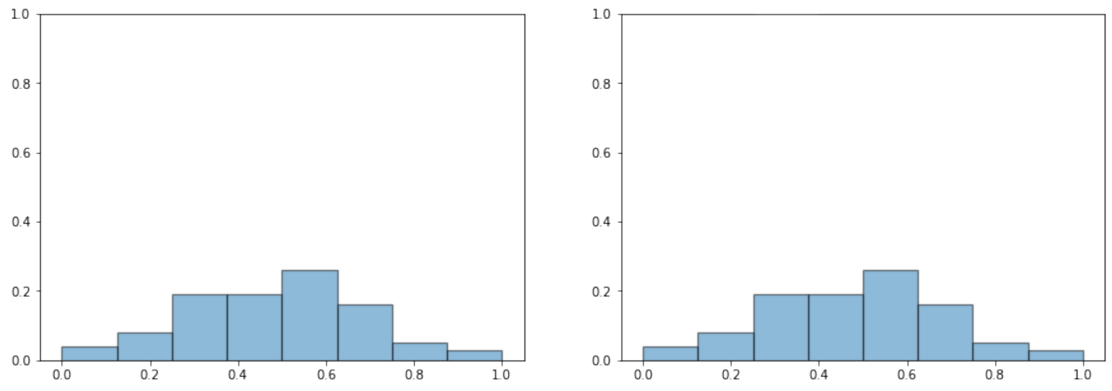

Figure 3: True histogram (left) and sigmoid differentiable histogram (right) with 8 bins for a feature normally distributed with 100 values. Difference between the two is 0.000123 in absolute error.

2.3. Sample generation

For the training process, the network needs training samples, so they can be fed forward through the layers, compute the loss and make a backward pass to learn the network weights. For each of the four subtasks, participants were given a labelled dataset to train the quantification algorithms plus a set of dev samples associated with their prevalences, but without individual example labels, intended to be used as validation samples. As example labels are not needed at all in this DNN architecture, dev samples were used as the base of the training process. From the 1000 thousand dev samples given for each task, 500 were used for training, 200 for validation and early stopping, and the remaining 300 were used for testing, to make sure that the solution was not overfitting. In the subtasks where labelled training data was used, the APP protocol was used to generate random samples.

Even though the number of samples may seem high, 500 instances for training the network is not enough and would cause overfitting. The solution adopted was to create a bag generator that can generate artificial samples that are a mixture of two real samples, the prevalence of which is aproximated to the average of the prevalences of the two real samples. To also feed the network with real samples, the sample generator was parameterized to control the generated proportion of real samples versus mix samples. This parameter was considered a hyperparameter to optimize for each subtask.

3. Experiments

In order to train the DNN, the bag generator described in the previous section was used to feed the network with samples. After finishing one epoch with a fixed number of samples, validation loss was computed using the 200 samples reserved for validation. For validation no *mix* samples were generated, only real ones.

The loss function for the network was Relative absolute error (RAE) is defined as,

$$RAE(p, \hat{p}) = \frac{1}{|C|} \sum_{c \in C} \frac{|\hat{p} - p|}{p}, \tag{1}$$

where p and $\hat{p}$ are the real and predicted prevalences and C are the classes of the problem. RAE may be undefined because of undefined denominator so we can take the smoothed version p_s of both p and $\hat{p}$ [4]:

$$p_s = \frac{\epsilon + p}{\epsilon |C| + \sum_{c \in C} p}, \tag{2}$$

where $\epsilon = \frac{1}{2 * samplesize}$.

To determine the number of training epochs, an early stopping criterion was established, taking the model with the best validation loss and stopping learning after a number of epochs without any improvement in the validation loss. The learning rate was considered a hyperparameter and was reduced by a factor of 0.5 after a certain number of epochs without improvement in the validation loss. Dropout and weight decay were used to prevent overfitting which was indeed a problem in the training process of the DNN.

After completing the training process, the 300 test samples (not seen during training or validation) were evaluated in order to get a realistic estimation of the performance of the network.

3.1. Hyperparameter search

Exploring the different architectures and hyperparameters in a deep learning problem can be a daunting experience. For tasks *T1A* and *T1B*, the Optuna hyperparameter optimization framework was used [8]. For tasks *T2A* and *T2B* the *BERT* module slowed down the training process to a degree that made impossible an automated hyperparameter search due to the lack of computational resources (check Table 1 for a list of the most important hyperparameters considered and their values for each task). Table 2 illustrates the optimization process conducted for task *T1A* using Optuna. For each hyperparameter, the search space considered and the best value found are presented. Note that an exhaustive search would not be plausible due to the high number of hyperparameters and possible values. As an alternative, Optuna uses a sampler to pick hyperparameter values using TPE (Tree-structured Parzen Estimator) algorithm [9] to explore the search space and find the best values that optimize the objetive function.

3.2. Training process differences by subtask

Even though every subtask was tackled with a similar training process, it is worth noting the subtle differences between them. In subtask *T1A*, only dev samples were used in the training process. Best hyperparameters were found using Optuna and the final model was trained with those. The approach for *T1B* was similar to *T1A* but training the network was more difficult. Due to the high number of classes, the network tended to overfit. In this case, using the samples generated with the labelled train data in a second stage helped reduce overfitting to a certain degree. For task *T2A* the training process was divided into two stages. In the first one, BERT

Hyperparameter	T1A	T1B	T2A	T2B
starting lr	0.00027	0.0005	0.0001	0.001
optimizer	AdamW	AdamW	AdamW	AdamW
batch size	21	500	20	200
weight decay	0.0018	0	0	0
histogram bins	32	32	32	32
dropout	0.52	0.5	0.5	0.1
real bags proportion	0.35	0.5	0.5	0.5
quantification linear layers size	[906, 1554, 38]	[4096]	[2048, 512]	[4096]

Table 1
Summary of the most important hyperparameters used for each task.

Hyperparameter	Values considered	Selected value
starting lr	[0.00001, 0.01]	0.00027
optimizer	AdamW,Adam,RMSprop	AdamW
batch size	[1, 100]	21
weight decay	[0.00001, 0.1]	0.0018
histogram bins	[2, 64]	32
dropout	[0, 0.8]	0.52
real bags proportion	[0, 1]	0.35
feature extraction hidden layer size	[2, 1024]	798
feature extraction output layer size	[2, 256]	118
quantification linear layers	[1, 3]	3
quantification linear layer size	[2, 2048]	[906, 1554, 38]

Table 2
Hyperparameters optimized for task *T1A*. Note that the values tested (except for the optimizer) represent ranges from which Optuna will sample values trying to optimize the RAE over the validation set.

weights were frozen, allowing the network to update the weights only for the quantification layers. In a second stage, all the weights were unfrozen and the network was allowed to converge freely. For subtask *T2B*, BERT was finetuned for a few epochs to the 28 classes of the problem using the training data. Then the process was the same as for *T2A*. This additional step helped the network to converge faster as the feature extraction layer is already optimized to distinguish between the 28 classes of the problem.

In Table 3, training and inference times are shown for each subtask. Computation for tasks *T1A* and *T1B* was done in a NVidia Titan Xp card, with 12Gb of RAM. For tasks *T2A* and *T2B* a NVidia GeForce RTX 3080 with 24Gb of RAM was used. Note that *T2A* and *T2B* training and inference times are hugely conditioned by the feature extracture layer (BERT). Difference in training times between *T1A* and *T1B* is due to the higher number of epochs needed in *T1B* for the network to converge and the bigger sample size (250 for *T1A* vs 1000 examples for *T1B*).

Task	Training time	Inference time
T1A	30 min	2 min
T1B	23 hours	5 min
T1A	52 hours	1 hour
T2A	90 hours	4 hours

Table 3
Training/inference times for each of the subtasks. Inference time represent the time required to evaluate the 5000 test samples.

3.3. Results

After selecting the best model for each subtask, based on the MRAE (mean RAE across samples), the solution was submitted to the competition website. The following table shows the results for each subtask in MRAE and MAE (mean absolute error, defined as the mean by sample of $abs(p - \hat{p})$), compared with the best baseline method, which was consistently the EM method [10]. Results for the rest of the competitors and the other baselines methods can be found on the website of the competition [1].

Task	MRAE			MAE		
	Validation	Test	Baseline (EM)	Validation	Test	Baseline (EM)
T1A	0.1194	**0.1090**	0.113823	0.0229	**0.0233**	0.02518
T1B	0.9093	**0.8842**	1.182070	0.0280	0.0280	**0.01976**
T2A	0.0834	0.1070	**0.087029**	0.0184	**0.0192**	0.01952
T2B	1.1911	**1.2309**	1.309778	0.0319	0.0321	**0.01552**

Table 4
Validation: results over the last 300 dev samples that were not seen by the network in the training process; Test: final results with the competition test data; Baseline (EM): results of the best baseline method over the competition test data

3.4. Discussion

The results obtained in the final test data, released after the end of the competition, improve MRAE for the existing baselines in three out of four subtasks. For subtasks *T2A* and *T2B* the validation result is better than the final result. This may have been caused by overfitting to validation data. Note that these subtasks were solved using a *BERT* transformer which is a huge model with millions of parameters. Maybe the results using a slightly simpler method as a feature extraction layer could have led to better results.

[1]https://codalab.lisn.upsaclay.fr/competitions/4134#results

Acknowledgments

This research was funded by MINECO (Ministerio de Economía y Competitividad) and FEDER (Fondo Europeo de Desarrollo Regional), grant PID2019-110742RB-I00 (MINECO/FEDER).

Also, I would like to thank the organization of the *LeQua 2022* competition, for making possible this research and bringing us the opportunity to learn more about new and existing methods, pushing a little further the quantification field. Lastly, thanks to Juan José del Coz for his insightful comments and the enriching discussions about the methods and the competition.

References

[1] P. González, J. Díez, N. Chawla, J. J. del Coz, Why is quantification an interesting learning problem?, Progress in Artificial Intelligence 6 (2017) 53–58. URL: https://doi.org/10.1007/s13748-016-0103-3. doi:10.1007/s13748-016-0103-3.

[2] P. González, A. Castaño, N. V. Chawla, J. J. D. Coz, A review on quantification learning, ACM Computing Surveys (CSUR) 50 (2017) 1–40.

[3] A. Esuli, A. Moreo, F. Sebastiani, G. Sperduti, Overview of lequa 2022: Learning to quantify, 2022.

[4] F. Sebastiani, Evaluation measures for quantification: an axiomatic approach, Information Retrieval Journal 23 (2020) 255–288. URL: https://doi.org/10.1007/s10791-019-09363-y. doi:10.1007/s10791-019-09363-y.

[5] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv preprint arXiv:1810.04805 (2018).

[6] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, A. M. Rush, Transformers: State-of-the-art natural language processing, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Association for Computational Linguistics, Online, 2020, pp. 38–45. URL: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[7] T. Chen, B. Xu, C. Zhang, C. Guestrin, Training deep nets with sublinear memory cost, arXiv preprint arXiv:1604.06174 (2016).

[8] T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: A next-generation hyperparameter optimization framework, in: Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019.

[9] J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, Advances in neural information processing systems 24 (2011).

[10] M. Saerens, P. Latinne, C. Decaestecker, Adjusting the Outputs of a Classifier to New a Priori Probabilities: A Simple Procedure, Neural Computation 14 (2002) 21–41. URL: https://doi.org/10.1162/089976602753284446. doi:10.1162/089976602753284446.