Natural Language Understanding (NLU) on the Edge

Nicolas Crausaz[1], Jacky Casas[1], Karl Daher[1],
Omar Abou Khaled[1] and Elena Mugellini[1]

[1] HES-SO University of Applied Sciences and Arts Western Switzerland,
Bd de Pérolles 80, 1700 Fribourg, Switzerland
nicocrausaz@hotmail.com
{jacky.casas, karl.daher, omar.aboukhaled, elena.mugellini}@hes-so.ch

Abstract. Today, chatbots have evolved to include artificial intelligence and machine learning, such as Natural Language Understanding (NLU). NLU models are trained and run on remote servers because the resource requirements are large and must be scalable. However, people are increasingly concerned about protecting their data. To be efficient, the current NLU models use the latest technologies, which are increasingly large and resource-intensive. These models must therefore run on powerful servers to function. The solution would therefore be to perform the inference part of the NLU model directly on edge, on the client's browser. We used a pre-trained TensorFlow.js model, which allows us to embed this model in the client's browser and run the NLU. The model achieved an accuracy of more than 80%. The primary outcomes of NLU on edge show an effective and possible foundation for further development.

Keywords: Machine learning · Artificial Intelligence · Natural Language Understanding · Chatbot · Edge Computing · Data privacy · Conversational agent

1 Introduction

Natural Language Understanding (NLU) [1] is a technique used in the creation of conversational agents [2]. The goal is to extract the intent of the message from the input text sentence. Chatbots started by being scripted then evolved to include artificial intelligence and machine learning [3]. Nowadays, chatbots can perform as good as humans in specific domains [4].

Today, NLU models are trained and run on remote servers because the resource requirements are large and must be scalable. However, people are increasingly concerned about protecting their data and are not always willing to send it to large multinational companies to use it for their business. This behaviour is specifically very understandable in the health or financial field, where users wish to keep their data private [5]. The solution would therefore be to perform the inference part of the NLU model directly on edge, on the client's browser, so that no data leaves the computer when chatting with a chatbot. In addition to ensuring an advanced secured confidentiality, moving the model inference, should lead to better performance in terms of latency and scalability. In the future, customised and personalised models can be implemented.

In this work, we reviewed the state of the art of existing NLU models and considered porting some of them to the client's browser. To do so, we had to modify the current models into JavaScript so that they could run in the browser. The basic training of the model is still done on a server, but the inference phase is then done on the client-side after he has downloaded the model locally.

2 Related Work

The success of deep learning in recent years is mainly due to the ever-increasing power of computers [6]. Even today's smartphones have ever-increasing computing power. The second cause of this growth is the amount of data that keeps accumulating because it is obvious that deep learning improves with the amount [7].

Today, NLU models are formed and run on remote servers because resource requirements are large and must be scalable. The idea of moving the model inference directly "on the edge" solves several flaws of an inference performed on an external server [8, 9].

The first advantage is confidentiality. By moving the model inference to the browser, we avoid sending data across the network. This ensures optimal protection of user privacy since all information that is sent to the models does not leave the browser. This advantage is very important in certain fields such as the banking and medical sectors.

The second advantage is latency. This is an essential element in areas such as video, audio, areas where data must be processed in real-time and as quickly as possible. When a large amount of data flows to the server, there is a risk of creating a queue. This queue could cause a delay because the network cannot satisfy all requests. If the model inference is done on the browser, delays can be minimised.

The third advantage is scalability. Sending data across the network can lead to flexibility problems. The number of connected devices is constantly increasing [10] and the network is becoming progressively crowded. Once again, in a field like video, a source like bandwidth would become far too demanding if everything had to be sent over the network.

The fourth advantage is the ability to develop a personal model for each user. Having the template on the client's web page allows full customisation of the template.

The last advantage is simply the price. Having the model hosted on the browser eliminates the cost of an external server where the NLU solution would run. Nevertheless, edge computing has its limits both in terms of resources and performance.

2.1 Machine Learning Frameworks

There are several web frameworks (TensorFlow.js, Machine learning tools, Keras.js, ml5.js, nlp.js, ONNX.js, ConvNet.js, WebDNN and Brain.js) that allow you to run machine learning models on the browser. Among them is TensorFlow.js [11], an open-source library that allows to run machine learning models entirely in the browser. The library is part of the TensorFlow [12] ecosystem. This library is of particular interest to us because we can import trained models into the browser. However, not every type of model can be imported. TensorFlow.js allows us to import two python model formats.

The Keras models and the TensorFlow GraphDef models. The framework even makes available pre-trained models ready to use.

2.2 NLP Model

TensorFlow.js provides two models for text processing: Universal Sentence Encoder (USE) [13] and Text Toxicity [14]. The Universal Sentence Encoder model will be of interest in this work. This model allows us to encode text using the "Embedding" technique. The encoded text can then be used as input for natural language processing tasks. The USE lite model [15], a lighter version than the original [16] (often used when computing resources are limited) has been converted into the Tensorflow.js GraphModel format [17] so that it can be used from the TensorFlow.js library. Unlike word embedding, which encodes a single word, this model is optimised to encode sentences, expressions and small paragraphs. This model, therefore, allows us to vectorise sentences. In order to compare the similarity between two sentences, i.e. vectors, we can use the cosine similarity formula.

Universal Sentence Encoder is not the only model that has been analysed. ULMFiT [18] from fast.ai is also a model that allows us to perform NLP tasks. The model does not have a solution prepared to be deployed on the browser. Fast.ai does not plan to develop a Tensorflow Hub module that will allow easy use of the model from TensorFlow.js, but they support the idea if someone wants to start such a project.

Hugging Face DistilBERT [19] is a distilled version of BERT. It is therefore smaller, faster, cheaper and lighter. It is interesting to analyse the possibilities offered by this model since it obtains very good results. DistilBERT offers several models compatible with Tensorflow that can easily be exported to SavedModel format once trained. The SavedModel format can be converted by the TensorFlow.js converter to a format that can be loaded directly into TensorFlow.js for inference.

2.3 Synthesis and Discussion

The analysis of the related work of this project revealed a huge potential to run the model on the client's browser. Moving the inference of the model directly "on the edge" allows for better confidentiality and faster inference results. It avoids overloading the network. Once the page is loaded, the model's inference continues to work even if the network is no longer available. There is also tremendous potential for model customisation and reduction of deployment costs.

After analysis, the prototype NLU solution will use the Universal Sentence Encoder model from TensorFlow.js. ULMFiT and DistilBert did not produce a working prototype. The problems are largely due to operators used by models that are not currently supported by JavaScript frameworks.

3 Methods

The prototype we developed for this research is a chatbot that uses an NLU model on the edge that the user can use directly on the web browser. The chatbot is implemented in Vue.js. A dataset of 28 different intents and 3286 training sentences is used to train

the model and serves as a basis for our study. Each one of the 3286 recordings contain the sentence in English and the corresponding intent. The intents range from "/bot.hobbies" to "/event". The machine learning model has, therefore, to solve a multi-class classification problem with these 28 classes. The chatbot uses the NLU solution to determine the user's intent, and so he can continue to discuss with it.

The concept of the prototype is to encode the 3286 training phrases with the USE model. We then obtain 3286 vectors with 512 dimensions. Then, when the user sends a message to the bot, we encode his sentence and compare it with the other vectors. The goal is to find out which vector is most similar to the client's sentence and determine the intent of the user's sentence. The Cosine similarity [20] formula is used to determine the similarity between the vectors. The formula used to calculate the similarity between two n-dimensional vectors is described in equation 1. The cosine similarity formula returns a matrix. We calculate the average of the elements present in this matrix and, the closer the value is to 1, the greater the similarity between the two sentences.

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \, \|B\|} \quad (1)$$

3.1 Encoding

In order not to overload the processors of the client's computer, but also to speed up the process, we will not encode the sentences at each passage. Instead, we choose to encode these 3286 sentences in advance, once only. A script was written to encode the sentences and save the tensors in a JSON file, as seen in Fig. 1. Loading the 3286 tensors in the browser would be too greedy. For each intent of the text classification problem, a "medium" tensor will be computed.

The first step to achieve this is to group all tensors that have the same intent. Then, we will use the *torch.stack* method from PyTorch [21] which will allow us to concatenate all the tensors of the same dimension into a single tensor (n, 512) with n being the number of tensors for this category. To calculate the "medium" tensor, we now use the *torch.mean* method which takes as input our tensor (n, 512). We specify the dimension at 0 for a correct calculation of the mean and to obtain a tensor dimension (1, 512) as a result. By performing this work for each intent, we finally end up with 28 tensors, each of them representing an intent.

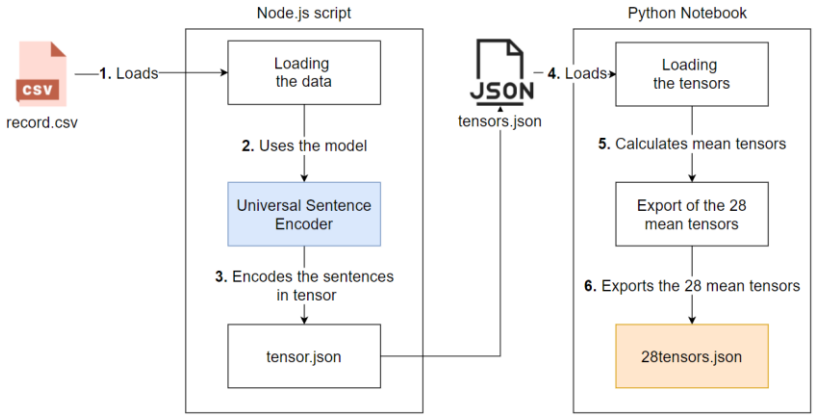

Fig. 1. Encoding

Now we have 28 tensors representing our 28 categories. We want to save them so that we can reuse them in the browser. The tensors and their intention are saved in a JSON file.

3.2 Inference on the Edge

The first task "on the edge" is to load our 28 tensors from the JSON file. When the user sends a message to the bot, the user's sentence is encoded using the USE model. We then compute the similarity only between the user's tensor (sentence previously encoded) and our 28 tensors. Fig. 2 shows this process. The similarity score is then retrieved for each intention. To normalise the results to 1, we have implemented a softmax function [22] in JavaScript. We get the percentage of similarity for each intention and the chatbot can use these results to continue the discussion with the user.

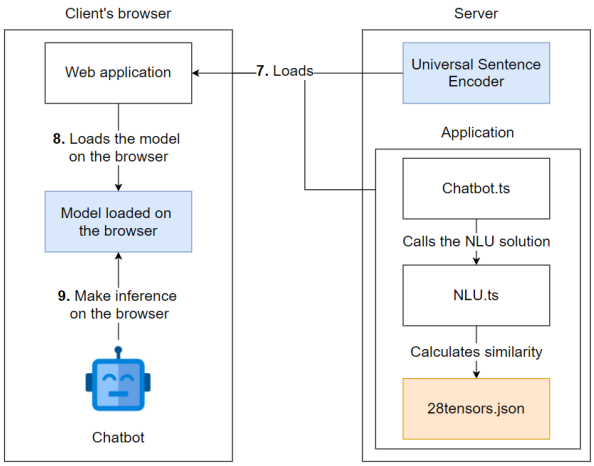

Fig. 2. Inference on the browser

In order to improve the performance of the first prediction, which can take some time, it is smart to "warm up" the model by passing an input tensor of the same shape before using the actual data [23].

4 Results

Currently, the prototype is compatible with most browsers (Chrome, Firefox, Safari, Edge) except Internet Explorer which does not support the use of the TensorFlow.js library, including the Universal Sentence Encoder model.

4.1 Model Evaluation

To calculate the performance of the learning machine model on the dataset, the data is split 90% for training and 10% for validation. The results metrics we used are precision, recall and F-score. We also used classification reports and confusion matrix from Sklearn [24] during the testing phase. The version of the Universal Sentence Encoder we used is the basic model called “Lite”. The different results we get with the model are shown in Table 1.

Table 1. Model performance

Precision	Recall	F1-score
0.817	0.796	0.794

Other NLP models have also been evaluated on the same dataset. In terms of comparison, DistilBert [19] obtains a precision of 0.925, DIET [25] of Rasa a precision of 0.897 and ULMFiT [18] of fast.ai obtains a precision of 0.906 (weighted average). The developed prototype uses the USE model, which obtained the worst performances, but the figures are nevertheless acceptable for such a task.

4.2 Inference Time and Weight on the Edge

Several tests were performed to calculate the inference time. The tests were performed with sentences of different lengths. Out of a total of 20 tests, an average inference time of 3.24 seconds was obtained. The inference time varies and sometimes we can even exceed 6 seconds with a response from our NLU solution. Note also that when the model is questioned with the same sentence, the inference time decreases with each reiteration.

Concerning the data that is loaded on the browser, there are two elements necessary for the operation of the learning on the edge machine: the JSON file that contains the 28 tensors and which is 533 KB which is really reasonable and the Universal Sentence Encoder model. The size of the model is 247 KB and the size of the vocabulary is 281 KB. With the weights of the model we reach a total of 29.3 MB for the use of the model with the tensors. Table 2 shows the different weights of the models trained on the same data. The USE model is a huge size saving compared to another model.

Table 2. Weight of the models

Model	Weight
USE	29.3 MB
ULMFiT	78.5 MB
DistilBERT	758.6 MB

5 Discussion and Conclusion

Concerning the prototype, the use of the NLU on the edge solution for the chatbot is somewhat tricky. As demonstrated in chapter 4.2, the inference time varies, but it is more significant than an inference on an external server. In a chat interaction, the user needs a quick answer, otherwise he does not feel like he is chatting live, and he quickly loses interest. The performance of the model could be improved. By analysing the tensors in detail, we could find a better solution to compute the average tensors. We could also find a solution to reduce the inference time, but this requires further analysis.

The purpose of this paper is to show that we are at the dawn of a new era, an era where even the most complex models can be made lighter and run on a browser. JavaScript frameworks that allow machine learning are booming. We even have frameworks that allow deep learning on smartphones. Moreover, the advantages of running a model "on the edge" are very real. The preliminary results show the potential held behind the on edge technology. Deeper research and further development are needed to improve the qualitative outcome.

6 Future Work

It would be really interesting to get another working NLP model that can solve the text classification problem on the browser. This would provide a way to really compare different models. We have already looked at other models and other ways to import models into the browser. The most common problem is that sometimes the operators used by the model are simply not supported by JavaScript frameworks. Of course, it is only a matter of time before these problems are resolved.

References

1. Jung, S.: Semantic vector learning for natural language understanding. Comput. Speech Lang. 56, 130–145 (2019)
2. Ramesh, K., Ravishankaran, S., Joshi, A., Chandrasekaran, K.: A Survey of Design Techniques for Conversational Agents. In: Information, Communication and Computing Technology. pp. 336–350. Springer Singapore (2017)
3. Rahman, A.M., Mamun, A.A., Islam, A.: Programming challenges of chatbot: Current and future prospective. In: 2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC). pp. 75–78 (2017)
4. Adamopoulou, E., Moussiades, L.: An Overview of Chatbot Technology. In: Artificial Intelligence Applications and Innovations. pp. 373–383. Springer International Publishing (2020)

5. Spring, T., Casas, J., Daher, K., Mugellini, E., Abou Khaled, O.: Empathic Response Generation in Chatbots. In: Proceedings of the 4th Swiss Text Analytics Conference (SwissText 2019), CEUR-WS, Winterthur (2019)
6. Baji, T.: Evolution of the GPU Device widely used in AI and Massive Parallel Processing. In: 2018 IEEE 2nd Electron Devices Technology and Manufacturing Conference (EDTM). pp. 7–9 (2018)
7. Halevy, A., Norvig, P., Pereira, F.: The Unreasonable Effectiveness of Data. IEEE Intell. Syst. 24, 8–12 (2009)
8. Ma, Y., Xiang, D., Zheng, S., Tian, D., Liu, X.: Moving Deep Learning into Web Browser: How Far Can We Go? In: The World Wide Web Conference. pp. 1234–1244. Association for Computing Machinery, New York, NY, USA (2019)
9. Chen, J., Ran, X.: Deep Learning With Edge Computing: A Review. Proc. IEEE. 107, 1655–1674 (2019)
10. Internet of Things - active connections worldwide 2015-2025, https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/
11. Smilkov, D., Thorat, N., Assogba, Y., Yuan, A., Kreeger, N., Yu, P., Zhang, K., Cai, S., Nielsen, E., Soergel, D., Bileschi, S., Terry, M., Nicholson, C., Gupta, S.N., Sirajuddin, S., Sculley, D., Monga, R., Corrado, G., Viégas, F.B., Wattenberg, M.: TensorFlow.js: Machine Learning for the Web and Beyond, (2019)
12. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, (2016)
13. Universal Sentence Encoder lite converted for Tensorflow.js, https://github.com/tensorflow/tfjs-models/tree/master/universal-sentence-encoder
14. Toxicity Classifier, https://github.com/tensorflow/tfjs-models/tree/master/toxicity
15. Universal Sentence Encoder lite, https://tfhub.dev/google/universal-sentence-encoder-lite/2
16. Cer, D., Yang, Y., Kong, S.-Y., Hua, N., Limtiaco, N., St. John, R., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., Sung, Y.-H., Strope, B., Kurzweil, R.: Universal Sentence Encoder, (2018)
17. Load Graph Model, https://js.tensorflow.org/api/1.0.0/#loadGraphModel
18. Howard, J., Ruder, S.: Universal Language Model Fine-tuning for Text Classification, (2018)
19. Sanh, V., Debut, L., Chaumond, J., Wolf, T.: DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter, (2019)
20. Wikipedia contributors.: Cosine similarity –- Wikipedia, The Free Encyclopedia. (2020)
21. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.) Advances in Neural Information Processing Systems. pp. 8026–8037. Curran Associates, Inc. (2019)
22. Wikipedia contributors.: Softmax function –- Wikipedia, The Free Encyclopedia. (2020)
23. Tensorflow for JS, Platform and environment, https://www.tensorflow.org/js/guide/platform_environment
24. Sklearn metrics, https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics
25. Bunk, T., Varshneya, D., Vlasov, V., Nichol, A.: DIET: Lightweight Language Understanding for Dialogue Systems, (2020)