Università degli Studi di Padova | Dipartimento Matematica

Master's Degree in Data Science

Academic Year 2023/2024

Natural Language Processing

Exploring SqueezeBERT

Lorenzo Baietti ID: 2130676

Francesco Carlesso ID: 2125806

Noemi Cicala ID: 2105377

Matteo Mazzini ID: 2107797

Lisa Tassinari ID: 2121469

Contents

Abstract

Mobile devices (especially smartphones) stand out as crucial platforms for implementing NLP models. However, smaller devices cannot afford the enormous computational costs associated with existing high-accuracy NLP models, such as BERT and RoBERTa. We propose an analysis of the Squeeze-BERT model, starting from the results presented in some papers that address computer vision techniques applied to NLP tasks. The reviewed research suggests using methods like residual connections and grouped convolutions to build efficient and compact NLP models; one of these models being, of course, SqueezeBERT. The model is said to be especially helpful in situations where quick responses are essential, such as real-time applications, and other interactive artificial intelligence systems. In the first part of the report we outline a theoretical overview of its main architectural components. In the second part we proceed to test the model on some specific tasks, comparing it to the BERT original model.

1 SqueezeBERT's Architecture

Introduction

SqueezeBERT is designed to minimize computational resource consumption while maximizing inference speed. Premises are, that although it may not achieve the same levels of accuracy as some larger models, it still performs competitively and provides a good balance for real-time applications. The model usually employs fewer parameters than other models tailored for the same objectives. Thus, it may be a more efficient use of computational resources than other alternatives.

1.1 Bottleneck Layers Architecture

The idea behind bottleneck layers is to compress the input feature map into fewer channels by temporarily reducing its dimensionality, and then expand it back to allow the network to concentrate only on the most important features. This architecture makes inference on smaller devices easier by preserving the network's depth and efficiency while lowering the number of parameters needed by the model. The main steps of bottleneck layers are:

- **Dimensionality Reduction**: where the number of channels in a network's intermediate layers is reduced, usually by employing a 1x1 convolution, which keeps the spatial dimensions intact while reducing the number of features.
- **Feature Extraction**: where, after the reduction, the network can learn compact representations of the data by applying more intricate transformations using 3x3 or larger convolutions.

SqueezeBERT in particular employs a series of steps to reduce dimensionality. First, it uses 1x1 convolutions in a "squeeze" step, to reduce the number of channels, and next, it uses fully connected layers in an "excitation" step, to model the interdependencies between channels; where Squeeze-and-Excitation (SE) blocks specifically incorporate this practice. Finally, the model expands back to the original dimensions. In the context of transformers, the reduction and expansion steps can be viewed as reducing the embedding dimension, processing the reduced embeddings, and then restoring them to the original size.

1.2 Residual Networks

Residual Networks (ResNet) use residual connections to mitigate the vanishing gradient problem and thus facilitate the training of deep networks, since as networks grow deeper, their performance can saturate and then degrade rapidly. ResNet solve this issue by introducing residual learning through skip connections, allowing gradients to flow through the network more effectively during backpropagation. Implementing ResNet allows to train deep neural networks more effectively, achieving better performance on various tasks, compared to traditional plain deep networks without skip connections.

So far we briefly introduced two of the main architectural components of the SqueezeBERT model, bottleneck layers and residual networks, that are also fundamental blocks for other models which are designed to address the limitations of mobile devices. In the next paragraph we will present Grouped Convolutions, which are the architectural component that particularly distinguish the SqueezeBERT model.

1.3 Grouped Convolutions

Grouped convolutions are a variation of standard convolutions that improve the computational efficiency and flexibility of convolutional neural network (CNN) models. In standard convolutions, each convolutional filter is applied to all input channels, producing an output that combines all the information present. However, in grouped convolutions, the input channels are divided into groups, and each group is convolved separately with its own set of filters. This reduces the total number of parameters and computational costs since the number of convolutional operations is reduced.

Additionally, grouped convolutions can improve the model's ability to capture more specific features, as each group of filters can learn different and more detailed features from different parts of the input. By allowing separate groups of filters to focus on different subsets of the input channels, grouped convolutions enable the network to learn a richer and more diverse set of features.

Recent discoveries in neural networks, such as those seen in AlexNet, have demonstrated that grouped convolutions can be leveraged to distribute computational load between multiple GPUs, resulting in faster computation. Additionally, in ResNeXt, the introduction of the concept of "cardinality" has further highlighted the potential of grouped convolutions. Cardinality refers to the number of groups (G), and it has been shown that increasing the number of groups can enhance model performance while maintaining the total number of parameters at a manageable level. This innovation allows for a more flexible and efficient design of convolutional neural networks, offering improved scalability and performance.

1.4 BERT-based Structures

BERT-based networks, such as BERT itself, are composed of three parts: the embedding, the encoder, and the classifier.

The embedding layer transforms individual words into fixed-length vectors of a specified size. Initially, the input sequence is fed into the input embedding layer, followed by position encoding. Subsequently, these position-aware embeddings are forwarded to the encoder for further processing.

The encoder is composed of a stack of blocks, each of which includes:

- **A self-attention module** with 3 Positionwise Feed-Forward Network (PFC) layers,
- **Three position-wise fully-connected layers** known as feed-forward network layers (FFN1, FFN2, and FFN3)

The classifier, in the end, utilizes the output generated by the encoder to classify or predict the desired final output.

1.5 SqueezeBERT Structure

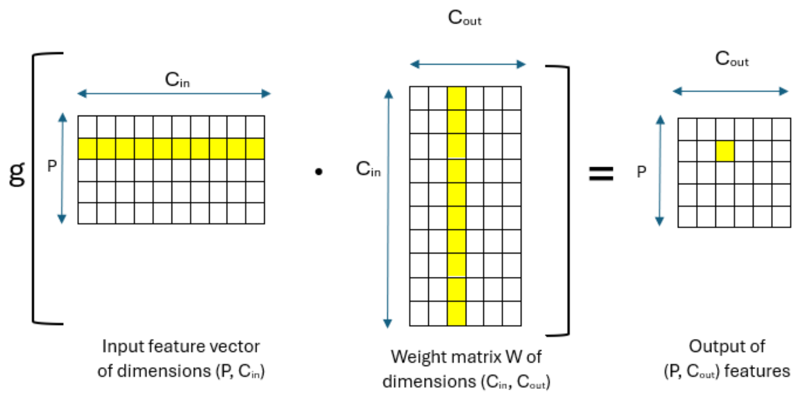
SqueezeBERT is designed with efficiency in mind, optimizing the standard BERT architecture for better performance on mobile and edge devices. The model architecture incorporates various techniques to reduce computational complexity while maintaining accuracy.

1.5.1 Reducing Latency: Replacing Fully Connected Layers with Convolutions

Benchmarking BERT on mobile devices highlights the computationally intensive components: it has been seen that the PFC layers account for the majority (about 85%) of the total latency. How to reduce the latency due to PFC layers? The answer is by replacing the position-wise fully connected layers with grouped convolutions. To convince you of this, it is first necessary to demonstrate the equivalence between the fully connected layers (used throughout the BERT encoder) and a non-grouped 1D convolution.

Position-wise fully-connected operator:

$$\mathrm{PFC}_{(p,c_{\text{out}})}(f,w)=\sum_{i}^{C_{\text{in}}} w_{c_{\text{out}},i} f_{p,i}$$



g
P
C_in
C_out
C_in
=
P
C_out
Input feature vector of dimensions (P, C_in)
Weight matrix W of dimensions (C_in, C_out)
Output of (P, C_out) features

The position-wise fully connected layer applies the same transformation to each position. Each position p in f is transformed using the weights w.

Definitions:

- **Input feature vector**: a vector of length P, where each f_p is, in turn, a vector of size C_{in}

$$f_p=\begin{bmatrix} f_{p,1} & \cdots & f_{p,C_{\text{in}}}\end{bmatrix}$$

- **Weights matrix**:

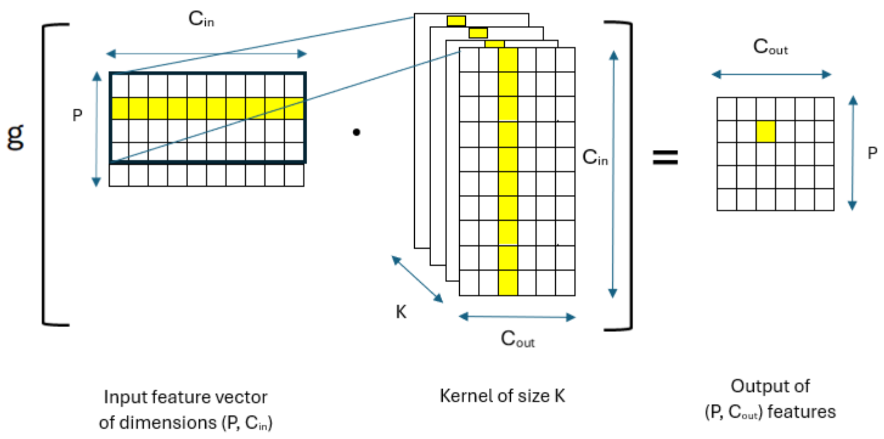
$$W=\begin{bmatrix} w_{1,1} & \cdots & w_{1,C_{\text{out}}} \\ \vdots & \ddots & \vdots \\ w_{C_{\text{in}},1} & \cdots & w_{C_{\text{in}},C_{\text{out}}}\end{bmatrix}$$

- **Output feature vector**: a vector of length P, where each f'_p is, in turn, a vector of size C_{out}

$$f'_p=f_p\cdot w=\begin{bmatrix}\sum_{i=1}^{C_{\text{in}}} f_{p,i}w_{i,1} & \cdots & \sum_{i=1}^{C_{\text{in}}} f_{p,i}w_{i,C_{\text{out}}}\end{bmatrix}$$

1D Convolution with kernel size K:

$$C_{(p,c_{\text{out}})}(f,w)=\sum_{i=1}^{C_{\text{in}}}\sum_{k=0}^{K-1} w_{(c_{\text{out}},i,k)}\cdot f_{(p-((K-1)/2)+k),i}$$


g
P
C_{in}
K
C_{in}
C_{out}
C_{out}
P
Input feature vector of dimensions (P, C_{in})
Kernel of size K
Output of (P, C_{out}) features

Definitions:

- **Kernel (filter)**: a 3D tensor of depth K, where each k_j is, in turn, a matrix of size $(C_{\text{in}}, C_{\text{out}})$

$$k_j = \begin{bmatrix} k_{j,1,1} & \cdots & k_{j,1,C_{\text{out}}} \\ \vdots & \ddots & \vdots \\ k_{j,C_{\text{in}},1} & \cdots & k_{j,C_{\text{in}},C_{\text{out}}} \end{bmatrix}$$

- **Output feature vector**: a vector of length P, where each f'_p is, in turn, a vector of size C_{out}

$$f'_p = \sum_{j=0}^{K-1} \left(f_{(p-((K-1)/2)+j)} \cdot k_j \right) =$$

$$\left[\sum_{i=1}^{C_{\text{in}}} \sum_{j=0}^{K-1} f_{(p-((K-1)/2)+j),i}\, k_{j,i,1} \quad \cdots \quad \sum_{i=1}^{C_{\text{in}}} \sum_{j=0}^{K-1} f_{(p-((K-1)/2)+j),i}\, k_{j,i,C_{\text{out}}} \right]$$

Thus, now it's easy to understand that if k=1 (kernel size) the position-wise fully-connected operation is equivalent to the convolution one.

This approach allows the model to extract relevant spatial features from the input before attending to them. In SqueezeBERT, FFN2 and FFN3 layers typically require four times as many arithmetic operations as the FFN1 layer due to their larger weight dimensions and more complex operations. To balance computational workload across channel groups, FFN1 layers use G = 1, while FFN2, FFN3 and all PFC layers of the self-attention modules employ G = 4. When grouped convolutions with G = 4 are applied in FFN2 and FFN3 layers, the computational workload is evenly distributed among groups, resulting in all FFN layers in SqueezeBERT having similar computational complexity. Aside from adopting a convolution-based implementation and utilizing grouped convolutions, SqueezeBERT maintains the characteristics of BERT-base, including an embedding size of 768, 12 encoder blocks, 12 heads per self-attention module, and the WordPiece tokenizer.

2 Testing

Experimental Methodology

We trained the BERT and SqueezeBERT models on the same datasets for three specific tasks. To compare them, we used two different evaluation metrics.

Average Cosine Similarity for:

- **Masked Language Modeling (MLM) task**: the cosine similarity between each pair of predicted and actual embeddings, then averaging these scores across all masked tokens.

Accuracy for:

- **Text Classification task**: the ratio of correctly predicted labels to the total number of predictions.
- **Token Classification task**: the ratio of correctly identified tokens (both named entities and non-entities) to the total tokens.

2.1 Masked Language Modeling

Masked language modeling (MLM) involves predicting a masked token in a sequence, allowing the model to attend to tokens on both the left and right sides. This bidirectional attention enables a deep contextual understanding of the entire sequence.

Importance for Mobile Devices

- **Enhancing Keyboard Prediction**: By training models to predict missing or masked words within sentences, the technology can better understand context and user intent, leading to more accurate and relevant word suggestions as users type.
- **Multilingual Applications and Adaptability**: MLM can be applied to multilingual models, enabling mobile apps to support multiple languages seamlessly. Also, MLM-trained models are more adaptable to language variations, dialects, slang, and new terms, benefiting a diverse user base.

Dataset

The used dataset is an improved version of the well-known DailyDialog dataset created to provide high-quality daily conversations covering various topics and common life situations. It contains a large number of dialogues, covering a wide range of daily topics (work, school, health, travel) and each dialogue consists of multiple exchanges between two speakers, simulating a natural conversation.

Model/Metrics	Average Cosine Similarity	CPU Time
SqueezeBERT	0.6972	115.056 sec
BERT-base	0.7820	174.756 sec

Table 1: Masked Language Modeling Performances

2.2 Text Classification

Text classification involves assigning a sentence or document to an appropriate category. Categories vary depending on the dataset and can cover a range of topics. Examples of text classification problems include emotion classification, news classification, and citation intent classification.

Importance for Mobile Devices

- **Improved User Experience**: It enables the device to automatically categorize and understand user inputs. This capability improves user experience by powering features like email, news and messages categorization.

Dataset

The dataset comprises 3,722 news articles in English, categorized into World, Politics, Tech, Entertainment, Sport, Business, Health, and Science. It includes two columns: Text (the news article) and Category (the corresponding class).

Results

Model/Metrics	Accuracy	CPU Time
SqueezeBERT	0.9463	62.666 sec
BERT-base	0.9705	99.316 sec

Table 2: Text Classification Performances

2.3 Token Classification

Token classification is a natural language understanding task where a label is predicted for each token in a piece of text. Unlike text classification, which assigns a category to the entire text, token classification provides predictions for each individual token. Common token classification tasks include:

- Named Entity Recognition (NER): Identifies specific entities within a text.
- Part-of-Speech (POS) Tagging: Classifies each token according to its part of speech.

We decided to perform NER.

Importance for Mobile Devices

- **Contextual Autocorrect and Predictive Text**: Enhances the understanding of word context within sentences, improving autocorrect and predictive text accuracy, making typing faster and more efficient.
- **Accessibility Features**: Vital for accessibility tools like creating calendar events on smartphones because it helps identify and extract specific entities such as dates, times, locations, and event names from user input.

Dataset

The CoNLL-2003 dataset is used for named entity recognition and includes data covering English and German languages. The dataset focuses on four types of named entities: persons, locations, organizations, and miscellaneous entities. Each file contains four columns: word, part-of-speech (POS) tag, syntactic chunk tag, and named entity tag, with each word on a separate line and an empty line after each sentence.

Results

Model/Metrics	Accuracy	CPU Time
SqueezeBERT	0.9674	172.922 sec
BERT-base	0.9756	300.034 sec

Table 3: Token Classification Performances

Conclusions

From the architectural overview carried out in the first part, it is easy to understand that all the elements involved are aimed at reducing complexity while maintaining efficiency. In the testing phase we could clearly see the advantages of the SqueezeBERT model, and even though the premise was that accuracy would be traded off for inference speed on some level, we can say that the model did not significantly under-perform with respect to the BERT-base, being on average 1.6 times faster than the latter. Furthermore, as the tasks got easier, the evaluation metrics between the two models got closer and closer, while the CPU time discrepancy between them got larger, favouring SqueezeBERT.

An important consideration to make is that training was conducted using a local GPU, but for testing, we used a CPU to better simulate the limited capabilities of mobile devices. This method ensures that our results are more realistic and applicable to our target deployment scenarios.

Masked Language Modeling (MLM) was the hardest task, we decided to represent performance using Average Cosine Similarity to exploit the feature of word embeddings, since a word can encapsulate the correct meaning in a sentence if it is similar to the masked word; and using accuracy would classify as correct only the exact matching between predicted and masked word. Interestingly, if we look at accuracy for this task (shown only in the outputs of the code, which is available in the references), the gap between the two models is closer, and this could mean that SqueezeBERT is outperformed more on harder examples where contextual information is more difficult to grasp. Another reason to consider the results as a good performance lies again on contextual information and the fact that the examples where composed by relatively short phrases, so the context for prediction was very limited. In terms of inference speed SqueezeBERT was more than 1.5 times faster than BERT-base.

In the Text Classification task, there was about a 3% gap in accuracy, outweighted by an almost 1.6 times faster inference time for SqueezeBERT. This task benefited from a wider context, and for the reasons stated before this could be the explanation for the enhanced performance.

As Token Classification task we performed Named Entity Recognition (NER), this is the task where our model of interest performed best, with just about a 1% negative gap in accuracy with respect to BERT-base, and an inference more than 1.7 times faster. This means that SqueezeBERT was able to precisely categorize the entities to which a word pertains, and that the architecture was able to speed up the process while losing very few information.

In the first two tasks the model had either to predict a word from scratch given little context, or to classify a large sequence of text probably dealing with some inherent ambiguity. In the last task, the model had just to categorize one word using contextual information and the word itself, which already carries a considerable amount of information. This is how we can explain the improvements seen throughout the tasks.

Furthermore, we notice that the results reported on the original paper (2020), which declared a 4 times faster inference time over the BERT-base, are not confirmed by our experimentation. Our results showed a much less remarkable discrepancy between the two models. However, it is necessary to say that during previous experimentation with less recent versions of the `transformers` and `torch` libraries, we obtained results that were more similar to the ones declared on the paper. Hence, we think that during the past years, improvements on the libraries have contributed to make BERT-base a lot more efficient. Additionally, the discrepancy between original results and ours might be attributed to the fact that the original paper's tests were conducted on a smartphone, while in our case, we used a computer. The difference in processor power between the two devices could be contributing to the observed variance in performance.

In conclusion, we are still satisfied with the results obtained from our analysis. The SqueezeBERT model has proven to be efficient, demonstrating fast inference times while maintaining acceptable accuracy levels across various tasks. This proves the effectiveness of the architecture of the model, validating its potential for practical applications.

References

Introduction

Iandola, F. N., Shaw, A. E., Krishna, R., & Keutzer, K. W. (2020). SqueezeBERT: What can computer vision teach NLP about efficient neural networks?. arXiv preprint arXiv:2006.11316.

Turc, I., Chang, M. W., Lee, K., & Toutanova, K. (2019). Well-read students learn better: On the importance of pre-training compact models. arXiv preprint arXiv:1908.08962.

1.1 Bottleneck Layers Architecture

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4510-4520).

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

1.2 Residual Networks

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Identity mappings in deep residual networks. In Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14 (pp. 630-645). Springer International Publishing.

1.3 Grouped Convolutions

Zhang, Z., Li, J., Shao, W., Peng, Z., Zhang, R., Wang, X., & Luo, P. (2019). Differentiable learning-to-group channels via groupable convolutional neural networks. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 3542-3551).

Wang, X., Kan, M., Shan, S., & Chen, X. (2019). Fully learnable group convolution for acceleration of deep neural networks. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 9049-9058).

Gong, J., Li, H., Li, Q., & Meng, L. (2021). A Deep Analysis of Grouped Convolution Schemes for Improving Deep Learning Performance. In ATAIT (pp. 45-52).

1.4 BERT-based Structures

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

1.5 SqueezeBERT Structure

Yang, C., Qiao, S., Kortylewski, A., & Yuille, A. (2021). Locally enhanced self-attention: combining self-attention and convolution as local and context terms. arXiv preprint arXiv:2107.05637.

Pan, X., Ge, C., Lu, R., Song, S., Chen, G., Huang, Z., & Huang, G. (2022). On the integration of self-attention and convolution. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 815-825).

Reducing Latency: Replacing Fully Connected Layers with Convolutions

Ou, J., & Li, Y. (2019). Vector-kernel convolutional neural networks. Neurocomputing, 330, 253-258.

Kiranyaz, S., Avci, O., Abdeljaber, O., Ince, T., Gabbouj, M., & Inman, D. J. (2021). 1D convolutional neural networks and applications: A survey. Mechanical systems and signal processing, 151, 107398.

2. Testing

The codes for the three tasks are available at the following repository:

`https://github.com/BaioSbubens/Exploring-SqueezeBERT`