DelBERTo: a Deep Lightweight Transformer for Sentiment Analysis

Luca Molinaro[1,*], Rosalia Tatano[1], Enrico Busto[1], Attilio Fiandrotti[2], Valerio Basile[2] and Viviana Patti[2]

[1]*Addfor Industriale s.r.l., Via Piovola, Empoli, 50053, Italy*
[2]*University of Turin, C.so Svizzera, Turin, 10149, Italy*

Abstract
This article introduces DelBERTo, a resource-efficient Transformer architecture for Natural Language Processing (NLP). Transformers replace convolutions and recurrence with the self-attention mechanism and represent the state-of-the-art in NLP. However, self-attention's complexity grows quadratically with the size of the input, which limits their applications. DelBERTo relies on adaptive input and on a deep yet lightweight Transformer architecture to reduce the number of learnable parameters, and relies on adaptive softmax to improve pre-training speed and memory footprint. We evaluate the proposed architecture in a sentiment analysis task and compare it against AlBERTo, a BERT model representing the state-of-the-art in sentiment analysis over Italian tweets. DelBERTo has only one-seventh of AlBERTo's learnable parameters, is faster, and requires less memory. Despite this, our experiments show that DelBERTo is competitive with AlBERTo over the three SENTIPOLC sub-tasks proposed at EVALITA 2016: subjectivity classification, polarity classification, and irony detection.

Keywords
Efficient Transformer, Sustainable NLP, Sentiment Analysis

1. Introduction

Natural Language Processing refers to the automated analysis of written text for tasks such as machine translation, question answering, and sentiment analysis.

Techniques based on deep learning represent the state-of-the-art in many NLP tasks. Among these, Recurrent Neural Networks (RNNs) [1] enjoyed widespread popularity due to their ability to process sequences of variable length via the recurrence mechanism. RNNs' main downside is that they suffer from the vanishing gradient problem and are not suitable to model long-term dependencies. Newer RNN architectures, such as Long Short Term Memory (LSTM) [2], overcome these limitations, but they still have one major downside: they are not able to take advantage of the parallelism of the hardware since they are sequential in nature. Other architectures, such as CNNs [3], are more parallelizable but cannot model long-term dependencies like LSTMs do.

Transformers [4] were introduced for machine translation tasks, and they simultaneously solved all of the previously mentioned problems. Transformers are based on the self-attention mechanism, which allows discovering relationships between different parts of a sentence. The complexity of the self-attention mechanism, which grows quadratically with the size of the input, represents the main limiting factor in their practical adoption. In recent years, efforts have been made to improve the efficiency of the Transformer architecture. Some techniques directly target the computational complexity of the self-attention mechanism, while others target the model as a whole. Despite these efforts, this complexity problem is still not completely solved.

This paper presents DelBERTo, a transformer-based lightweight architecture for NLP. DelBERTo builds upon the *Deep and Light-weight Transformer* (DeLighT) [5], which reduces the parameters and redistributes them among the different parts of the network. In this work, we leverage *adaptive input* [6] and *adaptive softmax* [7] to further slash the complexity to a point where it becomes affordable for practical applications. We also modified the DeLighT architecture to be encoder-only, which is more suitable for classification and sequence labeling tasks. We evaluate DelBERTo over the SENTIPOLC 2016 challenge [8], whose tasks are subjectivity classification, polarity classification, and irony detection. AlBERTo [9], a BERT [10] model for the Italian language, represents the best performer in this challenge. Our experiments show that DelBERTo achieves an F-score of 73% in subjectivity classification (AlBERTo 79%), 69% in polarity classification (AlBERTo 72%), and 62% in irony detection (AlBERTo 61%), despite having only one-seventh of AlBERTo's learnable parameters, being faster, and requiring much less memory.

Ital-IA 2023: 3rd National Conference on Artificial Intelligence, organized by CINI, May 29–31, 2023, Pisa, Italy
[*]Corresponding author.
l.molinaro@vargroup.it (L. Molinaro); rosalia.tatano@gmail.com (R. Tatano); e.busto@vargroup.it (E. Busto); attilio.fiandrotti@unito.it (A. Fiandrotti); valerio.basile@unito.it (V. Basile); viviana.patti@unito.it (V. Patti)
0000-0002-1944-9292 (L. Molinaro); 0000-0003-1798-1039 (R. Tatano); 0000-0002-9991-6822 (A. Fiandrotti); 0000-0001-8110-6832 (V. Basile); 0000-0001-5991-370X (V. Patti)

2. Background and Related Work

The Transformer was originally proposed as an encoder-decoder architecture for NLP tasks. The key characteristic of the Transformer is the attention mechanism, which aims at uncovering the relationship between words in two input sentences. Input sentences are first tokenized and then passed to an embedding layer that maps every token to a continuous representation $x \in \mathbb{R}^{d_e}$, where d_e is the size of the embedding vectors. Finally, to encode the information about the position of the words in the sentence, a *positional encoding* is used.

The attention layer calculates the scaled dot product attention between input sentences. The inputs to this layer are three matrices: query $\mathbf{Q}$, key $\mathbf{K}$, and value $\mathbf{V}$. The attention mechanism gives the values weighted by the softmax of the dot-product of all the queries and all the keys. If the three input matrices are the same, it is called a *self-attention mechanism*; otherwise, we talk about *cross-attention.* To improve the model performance, the Transformer architecture runs different attention layers in parallel, a mechanism called *multi-head attention.*

After the introduction of the Transformer, several new variants were proposed. One of the most famous is BERT, the current state-of-the-art language understanding model. It is an encoder-only architecture that can be used for more than one NLP task by fine-tuning it. Most importantly, BERT uses a *Masked Language Model* (MLM): during the training, random terms are masked in order to be predicted by the network. Finally, BERT employs a weight sharing technique to improve the efficiency of the Transformer, in particular by reusing the embedding matrix in the output layer.

AlBERTo represents the first BERT model trained on tweets in Italian. In particular, it was pre-trained on TWITA [11], a dataset containing tweets in Italian that is constantly updated. The particular version used contains about 200 million tweets without annotations, which makes it ideal for pre-training models in a self-supervised way by using the MLM technique. AlBERTo was fine-tuned on the 3 tasks of SENTIPOLC 2016, reaching state-of-the-art results on this challenge. In Sec. 4.3, we compare our results to those of AlBERTo.

Even though transformers like BERT score top-notch performance in several NLP tasks, the computational complexity of the attention layer is a major drawback to their practical adoption.

3. Proposed Method

This section introduces our transformer-based architecture for NLP, named DelBERTo, short for *Deep and Lightweight Bidirectional Encoder Representations from Transformers.*

The architecture of DelBERTo is inspired by the DeLighT architecture, and it was designed by us to be encoder-only. Fig. 1 depicts the architecture of DelBERTo. As shown, the embedding layer is composed of adaptive input followed by a DeLighT transformation (both explained later in this section). After the embedding layer, positional encodings are added, followed by a series of N DeLighT encoder blocks. These blocks are similar to the original Transformer's encoder blocks. The main differences are the fact that they start with a DeLighT transformation, the use of a single-head attention, and lastly, the fact that the fully connected network (FFN) first reduces the dimension of the hidden vectors and then expands it, thus saving parameters compared to doing the opposite. After the last encoder block, an adaptive softmax layer is employed. The weights of adaptive input and adaptive softmax are shared.

3.0.1. Adaptive Input and Adaptive Softmax

Transformer architectures that use large vocabularies have embedding layers characterized by a very large weight matrix. The output layer multiplies this weight matrix with the output of the previous layer during pre-training with MLM. This matrix multiplication is slow and problematic due to the large amount of GPU memory required for its calculation. These problems can be mitigated by substituting the embedding layer with an adaptive input layer and by using an adaptive softmax layer instead of the last matrix multiplication followed by the softmax function.

Adaptive input is a drop-in replacement for the embedding layer, i.e., it does not require any modifications to the rest of the model's computational graph. To reduce the number of parameters, each token is assigned to one of n different clusters based on its frequency and has an embedding vector whose size depends on the cluster: the most frequent tokens are assigned to the first cluster and have embedding vectors of size d_e, while the others with a lower frequency are assigned to subsequent clusters and have progressively smaller embedding vectors (of size d_e/k^{i-1}, where i is the index of the cluster, $1 \leq i \leq n$ and k represents the projection factor). The reduction of the embedding vector sizes produces, as an effect, a reduction in the number of parameters needed in the embedding layer.

In models that use the decoder-only or encoder-decoder architectures, the output of the network is the probability distribution for the next token, calculated using the softmax activation function. Adaptive softmax is a speedup technique used to replace the last dense layer with the softmax activation function in a neural network used for language modeling. Similarly to adaptive input, adaptive softmax performs a partition of the vocabulary into clusters. The first cluster contains the words oc-

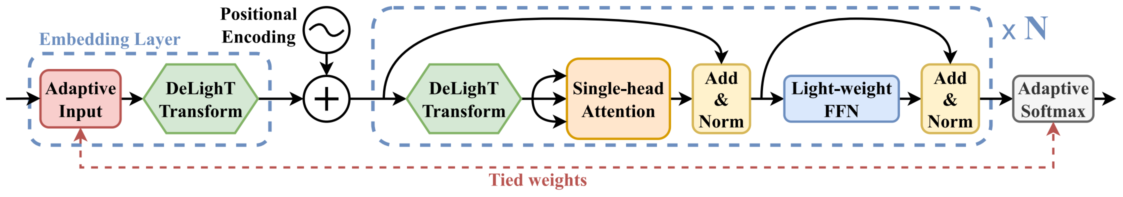

Figure 1: Architecture of DelBERTo

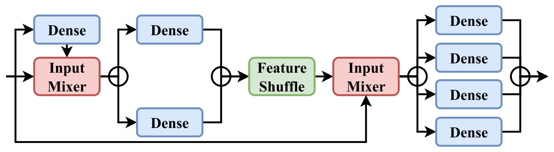

Figure 2: Example of a DeLighT Transformation during the expansion phase

curring with the highest frequency, and it represents the distribution's head, while the other clusters contain words occurring with a lower frequency and represent the tail of the distribution. Adaptive softmax takes inspiration from the class-based hierarchical softmax. The head cluster contains a special token for each tail cluster, which is used to model the probability that an output token belongs to the considered tail cluster. Then, since in the MLM task the labels are one-hot vectors indicating which token is the right one, the softmax output in the loss function is not explicitly calculated for all clusters but only for the head cluster (containing the most frequent tokens) and for the cluster containing the token indicated by the one-hot vector (which might likely coincide with the head cluster). This optimized loss function makes it possible to save GPU memory and speed up the computation.

3.0.2. The DeLighT Transformation

Let us now describe what the DeLighT transformation actually is. As shown in Fig. 2, the DeLighT transformation is composed of a series of three different kinds of layers: the Group Linear Transformation (GLT) [12] layer, which corresponds to the parallel dense layers in the figure; the *input mixer* [13] layer; and the *feature shuffle* layer [14]. The DeLighT transformation is composed of two distinct phases: the expansion phase, characterized by an increase in the number of parallel dense layers (groups) in the GLTs and in the size of the GLTs' output vectors, and the reduction phase, in which the opposite situation occurs.

Let us now describe the three layers of the DeLighT transformation. The GLT layer splits an input vector of size d_i into g vectors of size $\frac{d_i}{g}$. Each of these vectors is then passed as input to a different dense layer, which produces vectors of size $\frac{d_o}{g}$. Finally, the output vectors are concatenated into a single vector of size d_o. The splitting and concatenation operations are shown in Fig. 2 as circles. The feature shuffle layer performs a reshaping of the input vector $x \in \mathbb{R}^d$ into a matrix of size $g \times \frac{d}{g}$, then transposes this matrix and reshapes it back into a vector.

The input mixer layer takes two vectors, $x \in \mathbb{R}^{d_x}$ and $y \in \mathbb{R}^{d_y}$, as input. It then splits each input vector into g vectors such that $x = \text{Concat}(x_1, \cdots, x_g)$, $y = \text{Concat}(y_1, \cdots, y_g)$ and produces an output vector $z = \text{Concat}(x_1, y_1, \cdots, x_g, y_g)$. This layer acts as a replacement for residual connections since they cannot be used in the DeLighT transformation because they require vectors of the same size. The encoder layers in DelBERTo gradually become deeper and wider when moving from the first one to the last one.

3.0.3. Configuration

To instantiate DelBERTo, the vocabulary size V, the size of the embedding vectors d_e, and the maximum length of the input sequence take values of 128k, 768, and 128 respectively, as in AlBERTo, to ensure a fair comparison.

3.1. Training Procedure

The training procedure consists in a preliminary, self-supervised pre-training stage over a larger dataset, followed by a supervised fine-tuning stage over a smaller dataset. In both stages, the text of each tweet is preprocessed as in [9]. Following AlBERTo's training procedure, the network is first trained on the TWITA dataset over the MLM task. In this task, a single sentence is used as input, and 15% of its tokens are randomly selected to be either masked, replaced with a random token, or left unchanged. For each selected token, one of these 3 transformations is chosen randomly. In particular, masking is chosen 80% of the time, while the other two transformations are chosen 10% of the time each. The network is then trained to predict the selected tokens, therefore learning a language model. As for AlBERTo, we did not use the Next Sentence Prediction task since the dataset is not structured in a suitable way.

Next, the fine-tuning stage consists in training DelBERTo on all three tasks of the dataset provided for the SENTIPOLC 2016 challenge described in Sec. 4.1. In particular, we trained a DelBERTo binary classifier for each of the four labels in the dataset, as was done for AlBERTo. More details about how the training was performed in the different experiments can be found in Sec. 4.2.

4. Experiments

To evaluate the proposed architecture, we tested DelBERTo on the *SENTIment POLarity Classification Task 2016* (SENTIPOLC 2016). In this section, we first describe the SENTIPOLC 2016 dataset, and then we report the results of our experiments on this challenge, comparing them against the state-of-the-art.

4.1. Datasets

The SENTIPOLC 2016 challenge proposes three tasks on sentiment classification at the message level for Italian tweets: subjectivity classification, polarity classification, and irony detection. The training set of SENTIPOLC 2016 contains 7410 tweets in Italian. Each of them is annotated with a binary label for irony, one for subjectivity, and two binary labels for polarity. The combination of these two labels can express positive, negative, neutral, or mixed sentiment.

4.2. Setup

All experiments except one were performed using two Nvidia GTX 1080 Ti GPUs with 10 GB of memory each. Ideally, we wished to use the same training hyperparameters as AlBERTo to guarantee comparable results. However, the hardware constraints forced us to adapt the hyperparameters.

4.2.1. Pre-training

Like AlBERTo, we defined one epoch as equal to 2500 training steps. For the initial pre-training of DelBERTo, we did use the same batch size as AlBERTo (128) since the GPU memory was enough. We also used the same number of epochs (400). The pre-training took about 9 days.

4.2.2. Fine-tuning

We experimented with two different fine-tuning strategies, starting from the best checkpoint produced during pre-training. During fine-tuning, about 5% of the train set was left out for validation purposes, and the epoch was defined as one iteration of the dataset. In the following, the first fine-tuning strategy is referred to as *DelBERTo-1S*. In this case, we simply fine-tuned the entire network for 219 epochs with a learning rate (LR) of 2e−5, as for AlBERTo. Yet, due to GPU memory limitations, we could not afford AlBERTo's batches of 512 and had to use smaller batches of 128. The second fine-tuning strategy, *DelBERTo-2S*, relies on a more sophisticated two-stage approach. During the first stage, we froze the weights of all layers except the output layer and trained the network for 50 epochs with a LR of 1e−4. By training the output layer only, we could afford a larger batch size of 512 in this first stage. Furthermore, this prevents very large gradient updates, which would degrade the pre-trained weights since we are mixing pre-trained layers with a new, randomly initialized, output layer. Once convergence is reached, it is safe to proceed with the second stage, in which we trained all the layers with a lower LR of 1e−6 for 169 epochs. Due to the complexity of training all the layers and the fact that one GPU was not available to us for this fine-tuning, a smaller batch size of 64 was used.

4.2.3. Ablation Study

Finally, for the purpose of evaluating the impact of adaptive input and adaptive softmax in isolation, we consider another model that we call *AdalBERTo*. This model is based on BERT. The only difference is that AdalBERTo uses adaptive input and adaptive softmax instead of BERT's embedding layer and output layer. AdalBERTo was pre-trained and fine-tuned almost exactly like *DelBERTo-1S*. The only difference is that during pre-training and fine-tuning, we had to use a batch size of 64 because of hardware limitations. Nonetheless, we compensate for the smaller batch size by using 800 epochs during pre-training and by doubling the number of train steps per epoch during fine-tuning. The pre-training took about 17 days, nearly twice as much as DelBERTo. All the models were fine-tuned once for each label, creating 4 binary classifiers like it was done for AlBERTo.

4.3. Results

Table 1 summarizes the results of our experiments. The top half contains the results of AlBERTo and other references from [9]. The bottom part contains the results for DelBERTo and AdalBERTo produced by our experiments. F1 Scores have been calculated by running the official evaluation script and using the official SENTIPOLC 2016 test set, which contains 2000 examples. For each of the three tasks, three columns are presented. The first two columns show the F1 Score of each class, while the `F` column represents the mean of the two columns. For the polarity task, since the number of classes is 4, the `Pos` column shows the mean of the F1 Scores for the positive class and the non-positive class, and the `Neg` column shows the mean of the F1 Scores for the negative

Table 1
Results over the three SENTIPOLC 2016 classification tasks

	Subjectivity			Polarity			Irony			Params
	Obj	Subj	F	Pos	Neg	F	Non-I	Iro	F	[M]
AlBERTo	**73.98**	**84.15**	**79.06**	71.55	**72.91**	**72.23**	**94.08**	27.72	60.90	184
Unitor.1.u	67.84	81.05	74.44	63.54	68.85	66.20	n/a	n/a	n/a	n/a
UniPI.2.c	n/a	n/a	n/a	68.50	64.26	66.38	n/a	n/a	n/a	n/a
tweet2check16.c	n/a	n/a	n/a	n/a	n/a	n/a	91.15	17.10	54.12	n/a
AdalBERTo	71.74	78.38	75.06	**73.84**	64.76	69.30	93.79	17.20	55.50	97
DelBERTo-2S	68.11	78.34	73.22	70.43	68.93	69.68	93.00	**31.35**	**62.18**	**24**
DelBERTo-1S	68.29	76.03	72.16	59.30	63.31	61.31	93.05	26.59	59.82	

Table 2
Benchmark results for batch size = 32

Metric	Task	AlBERTo	AdalBERTo	DelBERTo
Speed (ex/sec)	PT	70	179 (2.57x)	282 (4.03x)
	FT	169	185 (1.09x)	291 (1.72x)
	INF	204	200 (0.98x)	229 (1.12x)
Memory (MB)	PT	25308	7435 (0.294x)	4250 (0.168x)
	FT	8115	7090 (0.874x)	3915 (0.482x)
	INF	1041	1043 (1.012x)	204 (0.196x)
Parameters		184.0 M	97.5 M (0.529x)	24.1 M (0.131x)

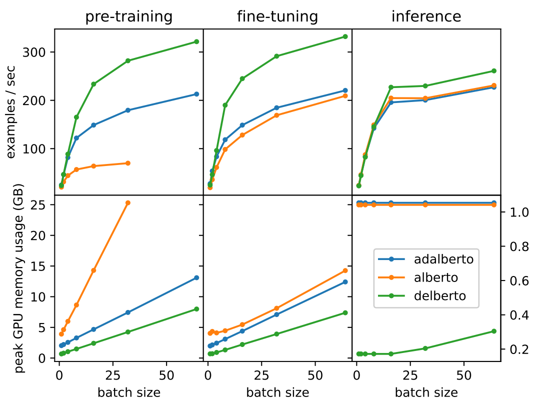

Figure 3: Benchmark results. The first row shows the speed, while the second one shows the peak GPU memory usage. Each column shows a different task.

class and the non-negative class. As shown in Table 1, *DelBERTo-2S* achieves results close to AlBERTo's at a fraction of the parameters. At the irony detection task, *DelBERTo-2S* gets better results than AlBERTo, whereas *DelBERTo-1S* is close but does not surpass it. The reason why *DelBERTo-2S* is better than *DelBERTo-1S* is the fact that it uses the two-stage fine-tuning process, which, as previously described, prevents the degradation of the pre-trained weights.

4.3.1. Complexity

For the purpose of comparing the computational complexity of DelBERTo with AlBERTo, we implemented the same BERT architecture used by AlBERTo. AdalBERTo uses the same code but with the modifications previously described, so as to minimize the differences in the implementation. For these benchmarks, we were given temporary access to one Nvidia A40 GPU with 48 GB of memory. In particular, we measured the computational performance differences in terms of the peak GPU memory consumption and the number of examples per second that the models were able to process. We measured these two metrics during pre-training, fine-tuning and inference with different batch sizes (all the powers of two between 1 and 64 inclusive).

Figure 3 shows the results of all the benchmarks, while Table 2 focuses on the results for a batch size of 32 examples. The first result to notice is the vast difference in speed and memory footprint between DelBERTo and the references during pre-training (PT), which constitutes the bulk of the training time. Due to the lightweight DeLighT architecture and the adaptive input and adaptive softmax layers, DelBERTo is 4.03 times faster than AlBERTo for a batch size of 32. AdalBERTo shares the same BERT architecture as AlBERTo, yet it is still 2.57 times faster than AlBERTo thanks to adaptive input and adaptive softmax. So, we conclude that the DeLighT architecture, adaptive input, and adaptive softmax are about equally responsible for the faster training of DelBERTo. Regarding the memory footprint, DelBERTo and AdalBERTo use only 16.8% and 29.4% of the memory used by AlBERTo, respectively, with adaptive softmax representing the major source of memory savings. Regarding fine-tuning (FT) and inference (INF), the performance of AlBERTo and AdalBERTo is, as expected, similar since i) they share the same BERT architecture and ii) fine-tuning and inference do not use the optimized loss function.

In conclusion, DelBERTo blends the benefits of the DeLighT architecture with those of adaptive input and adaptive softmax to reduce the complexity at training and inference time. Furthermore, its encoder-only architecture makes it more suitable for classification tasks and sequence labeling tasks.

5. Conclusions and Discussion

This paper presents DelBERTo, a transformer architecture with performance comparable to BERT but significantly lighter. Our comparison with AlBERTo, a state-of-the-art model for the Italian language, showed that the performance of the two models on a well-known sentiment analysis task on Italian tweets is similar, while the training time and memory footprint are significantly reduced.

While the results are promising, we consider these experiments preliminary. In particular, the hardware resources available at the time constrained our hyperparameter choices; we expect that removing such constraints will boost performance. Finally, while the present work focused on the Italian language, DelBERTo is language-agnostic and can be deployed for tasks other than sentiment analysis.

References

[1] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning Internal Representations by Error Propagation, MIT Press, Cambridge, MA, USA, 1986, p. 318–362.

[2] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (1997) 1735–80. doi:10.1162/neco.1997.9.8.1735.

[3] X. Zhang, J. Zhao, Y. LeCun, Character-level convolutional networks for text classification, in: C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, R. Garnett (Eds.), Advances in Neural Information Processing Systems, volume 28, Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper/2015/file/250cf8b51c773f3f8dc8b4be867a9a02-Paper.pdf.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, I. Polosukhin, Attention is all you need, in: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, volume 30, Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[5] S. Mehta, M. Ghazvininejad, S. Iyer, L. Zettlemoyer, H. Hajishirzi, Delight: Deep and light-weight transformer, arXiv preprint arXiv:2008.00623 (2020).

[6] A. Baevski, M. Auli, Adaptive input representations for neural language modeling, in: International Conference on Learning Representations, 2019. URL: https://openreview.net/forum?id=ByxZX20qFQ.

[7] É. Grave, A. Joulin, M. Cissé, D. Grangier, H. Jégou, Efficient softmax approximation for GPUs, in: D. Precup, Y. W. Teh (Eds.), Proceedings of the 34th International Conference on Machine Learning, volume 70 of *Proceedings of Machine Learning Research*, PMLR, 2017, pp. 1302–1310. URL: https://proceedings.mlr.press/v70/grave17a.html.

[8] F. Barbieri, V. Basile, D. Croce, M. Nissim, N. Novielli, V. Patti, Overview of the evalita 2016 sentiment polarity classification task, in: Proceedings of CLiC-it 2016 and EVALITA 2016, Napoli, Italy, December 5-7, 2016., 2016. URL: http://ceur-ws.org/Vol-1749/paper_026.pdf.

[9] M. Polignano, P. Basile, M. Degemmis, G. Semeraro, V. Basile, Alberto: Italian bert language understanding model for nlp challenging tasks based on tweets, in: CLiC-it, 2019.

[10] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Association for Computational Linguistics, Minneapolis, Minnesota, 2019, pp. 4171–4186. URL: https://aclanthology.org/N19-1423. doi:10.18653/v1/N19-1423.

[11] V. Basile, M. Lai, M. Sanguinetti, Long-term social media data collection at the university of turin, in: Proceedings of the Fifth Italian Conference on Computational Linguistics (CLiC-it 2018), Torino, Italy, December 10-12, 2018., 2018. URL: http://ceur-ws.org/Vol-2253/paper48.pdf.

[12] S. Mehta, R. Koncel-Kedziorski, M. Rastegari, H. Hajishirzi, Pyramidal recurrent unit for language modeling, in: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Brussels, Belgium, 2018, pp. 4620–4630. URL: https://aclanthology.org/D18-1491. doi:10.18653/v1/D18-1491.

[13] S. Mehta, R. Koncel-Kedziorski, M. Rastegari, H. Hajishirzi, Define: Deep factorized input token embeddings for neural sequence modeling, in: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020, OpenReview.net, 2020. URL: https://openreview.net/forum?id=rJeXS04FPH.

[14] X. Zhang, X. Zhou, M. Lin, J. Sun, Shufflenet: An extremely efficient convolutional neural network for mobile devices, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 6848–6856. doi:10.1109/CVPR.2018.00716.