Paraphrase Detection on PAWS dataset

Sergey Tkachenko, Nikita Pankov

May 2023

Abstract

This project proposes a SOTA solution to the problem of paraphrase identification on PAWS$_{Wiki}$ test set. We used Concatenate Pooler with DeBERTa backbone trained on PAWS$_{Wiki}$ and PAWS$_{QQP}$ train sets to achieve $F1 = 0.95$ (improve from 0.943) [1]. Also, we investigate the effects of PAWS$_{QQP}$ and the unlabeled part of PAWS$_{Wiki}$.

`https://github.com/Sergey-Tkachenko/nlp_project_2023`.

1 Introduction

Paraphrasing is a form of plagiarism that refers to use of other idea, words or work, and presenting it in different ways by switching words, changing sentence construction and changing grammar style. Additionally, paraphrasing may include replacing some words with synonyms [Chowdhury and Bhattacharyya, 2018]. In short, a sentence can be defined as a paraphrase of another sentence if they are not identical but share the same semantic meaning [Liu et al., 2022].

Large language models (LMMs) showed the high efficiency in paraphrasing tasks [Becker et al., 2023]. The use of LLMs may lead to an increase in paraphrasing, which can compromise the integrity of legal writing. Using Transformer-based models for the classification seems to be intuitive to counteract this new form of plagiarism. [Wahle et al., 2021] And therefore, in this study, we provide a solution for paraphrase detection task using transformer-based neural models. We provide SOTA (as far as we know on PAWS-Wiki dataset) architecture, based on DeBERTav3 [He et al., 2021].

1.1 Team

Sergey Tkachenko, **Nikita Pankov**

2 Related Work

In this section we review techniques and approaches for paraphrase detection. A detailed review of methods was presented in the article "Paraphrase Identifica-

[1] Previous SOTA (self-reported metrics) here.

tion with Deep Learning: A Review of Datasets and Methods". [Zhou et al., 2022]

2.1 Traditional approaches

Traditional approaches usually inlcude TF-IDF and N-gram methods. TF-IDF technique works by analyzing the frequency and distribution of individual words and phrases across a set of documents to identify patterns and similarities that can be used to detect paraphrasing. In particular, TF-IDF measures the importance of a term in a document, taking into account how frequently it appears in the document and how often it appears in other documents. This technique is particularly useful in identifying simple paraphrases, such as those that involve the substitution of synonyms or equivalent phrases, and can be used in combination with other machine learning techniques to build more robust models for detecting paraphrasing [Vrbanec and Meštrović, 2020].

N-gram is another machine learning technique that is commonly used in the detection of paraphrasing. By analyzing the frequency and distribution of different N-gram sequences across a set of documents, machine learning algorithms can identify patterns of usage that are indicative of paraphrasing. In particular, N-gram analysis can be used to identify common phrases and sentence structures that are likely to be used in paraphrasing, as well as to identify variations in word choice and grammar. N-gram model is useful in identifying more complex forms of paraphrasing, such as those that involve reordering or restructuring sentences or paragraphs. [Kondrak, 2005]

2.2 Glove

GloVe [Pennington et al., 2014] is another powerful tool used in natural language processing and paraphrase detection. By using word vectors generated using GloVe, it is possible to determine the semantic similarity between words and their meanings, which is useful for identifying phrases that are used similarly or have similar meanings across different texts. This technique can identify subtle differences in word choice and sentence structure that may be indicative of paraphrasing. [Gautam and Jerripothula, 2020]

2.3 FastText

FastText [Bojanowski et al., 2017] is based on the idea of representing words as bags-of-character n-grams. By breaking words into their constituent character n-grams and training classifiers on these representations, FastText can identify similarities and differences between phrases and sentences that might be missed by traditional word-based methods. By training FastText models on large text corpora and combining them with other machine learning techniques, researchers can build highly accurate models for detecting paraphrase across a wide range of languages and domains. One of the baselines for paraphrase detection is using FastText in conjunction with SVM. [Wahle et al., 2021]

2.4 BERT

BERT [Devlin et al., 2018] is a powerful language model for natural language processing that has shown great potential in paraphrase detection tasks. Unlike traditional models that simply represent words as vectors, BERT learns contextual representations of words by considering the entire sentence. This allows it to capture subtle nuances in meaning and context that traditional models might miss. BERT is pretrained on large amounts of text data and then fine-tuned on specific tasks, such as paraphrase detection. By fine-tuning BERT on a paraphrase detection task, researchers can build highly accurate models that are capable of detecting paraphrases across a wide range of languages and domains. [Wahle et al., 2021]

2.5 T5

T5 [Raffel et al., 2020] is another powerful language model that has shown much promise in the field of paraphrase detection. T5 is a text-to-text transfer transformer built for translations and similar tasks. By fine-tuning T5 on a paraphrase detection task, researchers have achieved highly accurate models that can detect paraphrases across a variety of domains and languages. [Krishna et al., 2023] Moreover, T5 has shown potential for generating high-quality paraphrases [Nighojkar and Licato, 2021]. T5 shows competitive results, but slightly loses to BERT in paraphrase detection task. [Becker et al., 2023]

2.6 DeBERTa

DeBERTa [He et al., 2020] is yet another language model that has shown promise in the field of paraphrase detection. DeBERTa improves the BERT and RoBERTa models using two novel techniques. Compared to BERT, it has following key differences:

- **Disentagnled attention**. BERT uses the sum of content and positional emdbeddings as a token representation in the first layer, and then process this embeddings with standard transformer layers. Apparently, each token in DeBERTa is represented by two separate vectors: one for the content and one for the relative position. The resulting attention matrix for particular layer is a sum of three different attention matricies: content-to-content, content-to-position and position-to-content.
- **Enhanced mask decoder.** DeBERTa uses relative positional embeddings for attention calculation. In order to capture information about absolute positions, it uses EMD – Enhanced Mask Decoder – a generalization of BERT decoder layer. It allows to capture different information in order to get better answer.

Compared to BERT, DeBERTa was trained only on the half of its training dataset, but still managed to achieve better performance on different NLU tasks of GLUE [Wang et al., 2018] development set. Also, it must be outlined that DeBERTa has 10 times less parameters (1.5 billion) than T5 (11 billion) and still managed to outperform T5 on some tasks of SuperGLUE benchmark [Wang et al., 2019].

This model outperforms BERT on the paraphrase detection task [Liu et al., 2022] [Corbeil and Ghavidel, 2021]. With the disentangled attention mechanism, DeBERTa can capture both local and global dependencies in the input text, making it more effective at detecting paraphrases. Additionally, incorporating decoding strategies allows for more accurate generation of language models. Studies have shown that DeBERTa can identify paraphrases with high accuracy, even for complex sentences and phrases.

3 Model Description

3.1 Backbone

We have chosen DeBERTav3-large as a backbone architecture for two reasons:

1. It has very good performance compared to other large models, but at the same time is easy to train and use;
2. To the best of our knowledge, it is the SOTA architecture on PAWS dataset. [2] [3]

The first key difference of DeBERTa from BERT is so called disentangled attention. In BERT, absolute positional embeddings are added directly to the input token embeddings. DeBERTa's positional information is included in calculation of each attention matrix in the architecture. This is done as follows. Let $H \in \mathcal{R}^{N \times d}$ be the input vectors, where N is a sequence length and d is the size of hidden vectors, $W_{q,c}, W_{k,c}, W_{v,c} \in \mathcal{R}^{d \times d}$ – reprojection matrices for H (c underscore stands for content, as H are assumed to contain information about content), $P \in \mathcal{R}^{2k \times d}$ – relative position embeddings (they are shared across all transformer layers), where k is maximum relative distance, $W_{q,r}, Wk, r \in \mathcal{R}^{d \times d}$ – reprojection matrices for them. Then output embeddings calculated like in Eq. 1.

$$\begin{aligned} Q_c = W_{q,c},\ K_c = HW_{k,c},\ V_c = HW_{v,c} \\ Q_r = PW_{q,r}, K_r = PW_{k,r} \\ A_{i,j} = Q_c^i K_c^j + Q_c^i K_r^{\delta(i,j)} + Q_r^{\delta(j,i)} K_c^j \\ H_{output} = \text{softmax}(\frac{\mathbf{A}}{\sqrt{3d}})V_c \end{aligned} \tag{1}$$

, where $\delta(i, j)$ is defined by equation:

[2] Previous SOTA (self-reported metrics) here.
[3] Metrics for the PAWS dataset.

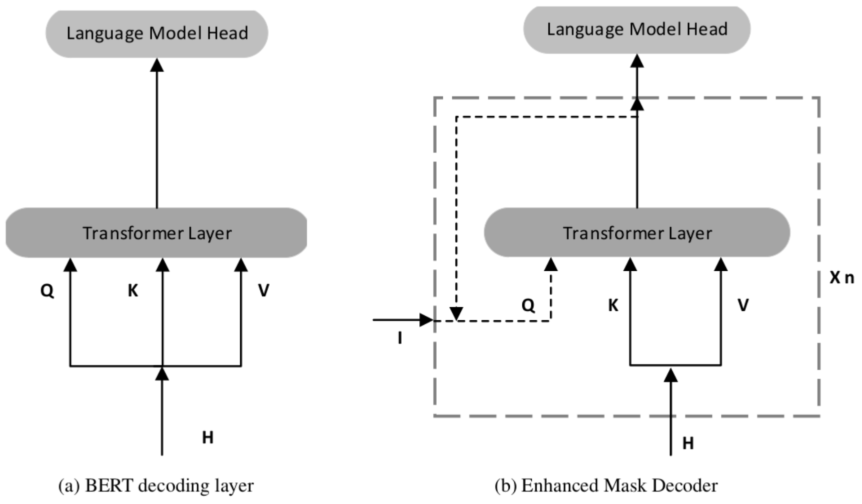

Figure 1: Enhanced Masked Decoder architecture.

$$\delta(i,j) = \begin{cases} 0, i-j \leq k \\ 2k-1, i-j \geq k \\ i-j+k, others \end{cases} \tag{2}$$

P and $\delta(i,j)$ have following interpretation: let m-th row of P represents relative position of tokens i and j so that distance between i and j is m - k. If distance is negative, it means that j is behind i and vise versa.

The second key feature of DeBERTa is Enhanced Masked Decoder. Its architecture is presented on Fig. 1, b. The main difference from BERT decoder (Fig. 1, a) is that it has two inputs: H, which is hidden states from Transformer layers, and I, which can contain arbitrary information. As DeBERTa uses relative position information in the transformer layers, information about the absolute position of token in sequence is incorporated at the Decoder stage as one of the I inputs.

DeBERTa was further improved in the paper [He et al., 2021]. Authors used two novel techniques: Replaced Token Detection (RTD) and Gradient-Disentangled Embedding Sharing (GDES).

RTD was proposed by authors of ELECTRA [Clark et al., 2020] architecture. Let us put $\mathbf{X}$ as an input sequence, $\tilde{\mathbf{X}}$ – the corrupted version of it, $\mathcal{C}$ - the index set of masked tokens. Model is trained in order to optimize following functional:

$$\sum_{i \in \mathcal{C}} \log p_\theta(\tilde{x}_i = x_i | \tilde{\mathbf{X}}) \underset{\theta}{\longrightarrow} max, \tag{3}$$

where θ is model parameters. Apparently, authors of ELECTRA proposed GAN-style learning, in which one network predict masked tokens, and then other network tries to detect them. More formally, let us consider two transformer encoders. First of them will stand for generator network with parameter θ_G, and second one be the discriminator network with the parameters θ_D. Generator tries to solve MLM task like in 3

$$L_{MLM} = \mathbf{E}\left(-\sum_{i \in C} \log p_{\theta_G}(\tilde{x}_i = x_i | \tilde{X}_G)\right) \xrightarrow[\theta_G]{} min, \tag{4}$$

where $\tilde{X}_G$ is the same to $\tilde{\mathbf{X}}$. In order to build input sequence for discriminator, we replace masked tokens in $\tilde{X}_G$ by sampling tokens according to the output probability distribution of generator. After that, we calculate RTD loss as follows:

$$L_{RTD} = \mathbf{E}\left(-\sum_{i} p_{\theta_D}\left(\mathbf{1}(\tilde{x}_{i,D} = x_i) | \tilde{X}_D, i\right)\right) \tag{5}$$

Two encoders are optimized jointly with the total loss $L = L_{MLM} + \lambda L_{RTD}$. In the original ELECTRA network, token embedding are shared across generator and discriminator.

However, authors of [He et al., 2021] argue that sharing embeddings between generator and discriminator leads to multitask learning problem. Specifically, generator and discriminator wants to learn different token embeddings, which leads to slow convergence. At the same time, using separate embeddings for generator and discriminator does not allow discriminator to benefit from the generator embeddings. Authors proposed the following solution. Let E_G be the generator embeddings, E_D be the discriminator embeddings. Then they are linked by following equation:

$$E_D = sg(E_G) + \Delta E, \tag{6}$$

where ΔE is some residuals and $sg(\cdot)$ is so called stop gradient function. This function prevents generator embedding from learning from discriminator loss function L_{RTD}. Experiments showed that GDES technique provides both good convergence and performance.

3.2 Pooler layers

As DeBERTa is a strong backbone, we decided to concentrate our efforts on effective aggregation of the information it provides. More concretely, in baseline network last hidden state of CLS token – first token in the sequence – was used to predict classes. It was achieved by applying two sequential linear layers to it.

But there is a lot of other information DeBERTa can provide: first of all, sentence tokens – they can also provide some information. Another option is utilizing hidden states from the other layers of DeBERTa – they can contain lower level features that can be useful for our task. Here and further we will address layer that incorporates all outputs of DeBERTa as Pooler layer. We describe different types of pooler layers below.

3.2.1 Perceptron pooler

Perceptron pooler - is a set of simple sequential linear layers that are applied to the last hidden state of CLS token. Almost the same as baseline, but it can have arbitrary number of layers and their sizes.

In our experiments, we used the following setup:

- number of layers – 3;
- layer sizes: (1024, 200), (200, 40), (40, 2);
- Activation – ReLU;
- Batch Normalization layer between linear layers for faster convergence.

Note that we used the same architecture of perceptron network to process outputs from other poolers as well.

3.2.2 Mean Max Pooler

The architecture of Mean Max Pooler is shown on Fig. 2. The idea is to take the last hidden state of CLS token as is and add information from the last hidden state of all other tokens. It is done by simply applying mean and max pooler to the all except CLS token last hidden state.

In our experiments we computed mean and max poolings only for relevant tokens, i.e. padding tokens were filtered. This was achieved by applying input attention mask from the tokenizer. Resulting hidden states were additionally downsized by linear layer.

After pooling, CLS embedding and pooling results were concatenated into one tensor and then processed by the perceptron network.

In our experiments, we used the following setup:

- a downsize linear layer with the size of (1024, 256);
- a perceptron with 3 layers of following sizes: (1536, 300), (300, 60), (60, 2).

3.2.3 Convolutional Pooler

It closely resembles the architecture of Mean Max Pooler, but two sequential 1D convolutional layers were applied before pooling. Also we used only max pooling to aggregate temporal information in our implementation.

Architecture of ConvNet is shown on Fig. 3.

In our experiments, we used the following setup:

- ConvNet, both convoltion layers had kernel size 3, padding 1. Their input and ouput channel count were as follows: (1024, 256), (256, 128);
- a perceptron with 3 layers of following sizes: (1024 + 128, 512, 128, 2).

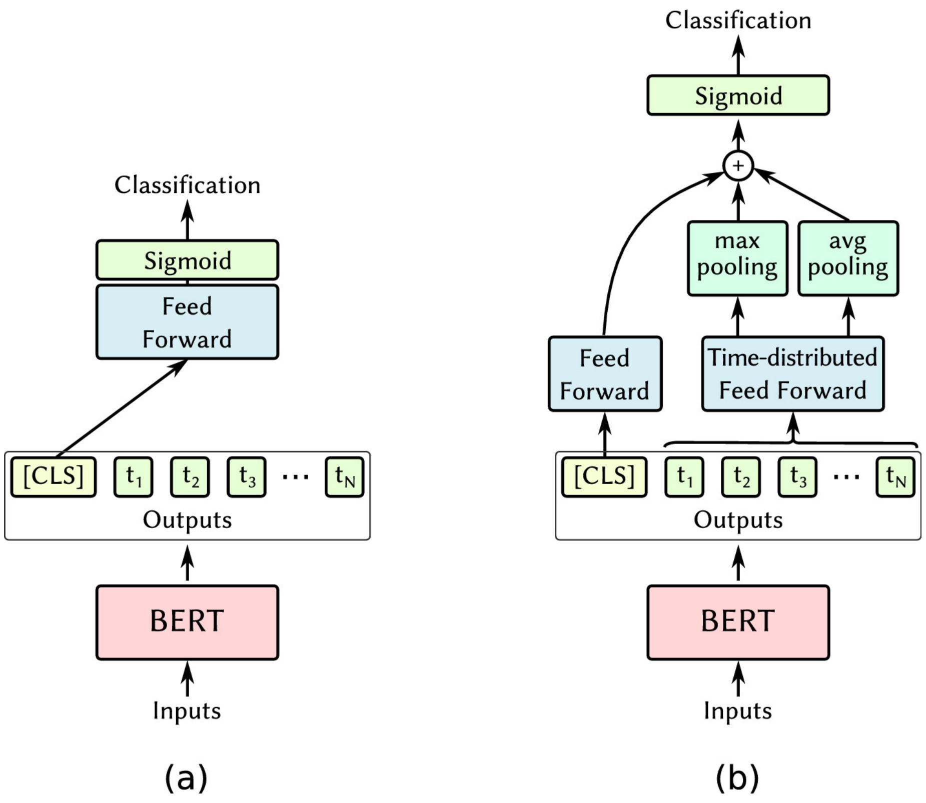

Figure 2: Mean Max Pooler architercture. Source.

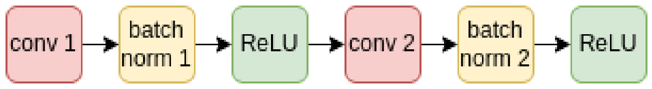

Figure 3: Architecture of convolutional part.

3.2.4 Concatenate Pooler

We took hidden states of CLS token from last 4 layers, concatenated them into one tensor and applied perceptron on it.

In our experiments, we used the following setup:

- A perceptron with 3 layers of following sizes: (4096, 512), (512, 64), (64, 2).

3.2.5 LSTM Pooler

Concatenate pooler has an important drawback – we must decide what layers to use. We can not use all layers as the resulting tensor would be very large. Hence, we loose low-level context while it can be useful for our task.

In order to solve this problem, we can treat hidden states from different layers as a sequence. To process this sequence, we can use Recurrent Neural Network as it can deal with the sequences of arbitrary length. We apply RNN to the hidden states of CLS tokens from different layers in order to obtain embedding of this sequence. This embedding was further processed by perceptron network in order to get final labels.

In our experiments, we used the following setup:

- RNN – bidirectional LSTM architecture [Sak et al., 2014], hidden size = 256, number of layers = 2;
- A perceptron with 3 layers of following sizes: (1024, 256), (256, 32), (32, 2).

3.2.6 Concatenate + LSTM Pooler

In all previous poolers we did not considered the fact that we actually have two concatenated sentences instead of one. We can try to use this information in order to get better embeddings of our input sequence.

The architecture of this pooler is shown on Fig. 4. In this pooler, we concatenate last 4 hidden states of each token to get embeddings for each token in sequence. Then, we mask (multiplied by binary tensor) sequence of concatenated tokens in order to get sequence of hidden state of tokens only from first and second sentences separately. We apply LSTM to each of sequences in order to get embeddings for each sentence. We concatenate them and CLS embedding into one tensor and process it by perceptron.

In our experiments, we used the following setup:

- LSTM: hidden size 256, number of layers 1;
- A perceptron with 3 layers of following sizes: (4608, 512), (512, 64), (64, 2).

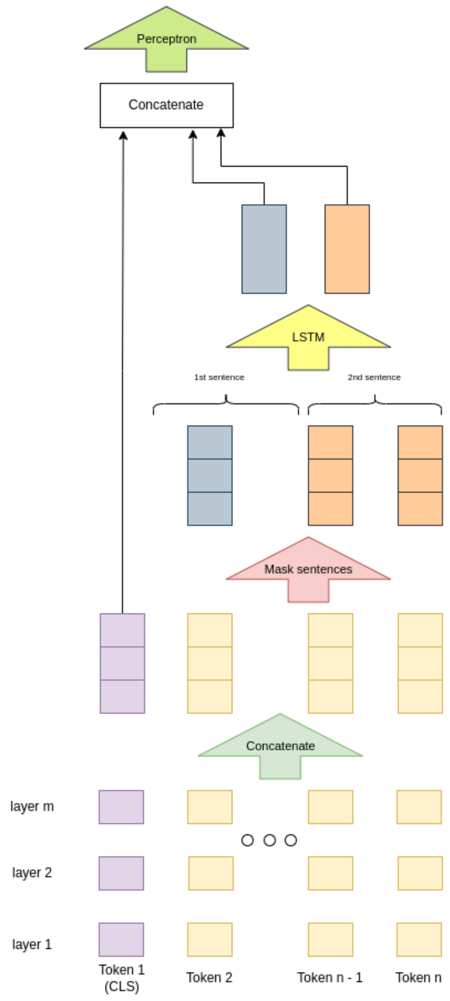

Figure 4: The architecture of LSTM + Concatenate Pooler.

4 Dataset

In out work, we use the PAWS[4] dataset. PAWS training data dramatically improves performance on challenging examples and makes models more robust to real world examples. [Zhang et al., 2019] The dataset consists of 2 parts: WiKi and QQP (Quora Question Pairs). Count of split for each PAWS subset can be found on Tab. 1.

	Train	Dev	Test	Paraphrases %
$PAWS_{QQP}$	11,998	677	-	31.3
$PAWS_{Wiki}$	49,401	8,000	8,000	44.2
$PAWS_{Wiki-swap}$	30,397	-	-	9.6

Table 1: Counts of split for each PAWS dataset. The final column gives the proportion of paraphrase (positive) pairs. There are 108,463 PAWS pairs in total.

Examples are generated from controlled language models and back translation, and given five human ratings each in both phases. The main idea of the PAWS is generating adversarial examples to break NLP systems.

On the Tab. 2 you can see the examples from PAWS dataset.

non-paraphrase
asking him for a passport to return to England through Scotland
and asked him for a passport to return to Scotland through England
paraphrase
The NBA season of 1975 – 76 was the 30th season of the National Basketball Association
The 1975 – 76 season of the National Basketball Association was the 30th season of the NBA

Table 2: Some examples of the PAWS dataset.

Challenging pairs are generated by controlled word swapping and back translation, followed by fluency and paraphrase judgments by human raters. Models that do not capture non-local contextual information fail with PAWS training examples. As such, PAWS provides an effective instrument for driving further progress on models that better exploit structure, context, and pairwise comparisons. [Zhang et al., 2019]

In addition to the labeled PAWS dataset, Google researches presented an unlabeled part of PAWS_{Wiki}. The difference is that they did not use human rating to this part of dataset, and simply treat all pairs from word-swapping as non-paraphrases and all pairs from back translation as paraphrases. The total

[4] PAWS dataset can be downloaded here.

number of such pairs is 656k. Due to lack of computational power, we included only first 50k pairs to our dataset.

5 Experiments

In this work, we conducted two major experiments.

The first experiment was designed to find the best effect of adding PAWS$_{QQP}$ and Unlabeled Wiki.

The second major experiment was designed to determine the optimal architecture.

5.1 Metrics

We used standard binary classification metrics to evaluate our models:

- Accuracy
- Recall
- Precision
- F1 measure
- ROC AUC score

They were computed both for validation and test set. In each experiment we choose the best epoch on dev (aka validation) set of PAWS$_{Wiki}$. The final metrics are then calculated on the test set of PAWS$_{Wiki}$.

5.2 Experiment Setup

We used standard split provided by authors of PAWS Wiki dataset[Zhang et al., 2019]. We also added the SWAP part of the data to our training set. We evaluated model on validation dataset after every epoch. The best model among all epochs was selected based on F1 metric value, and then final metrics on the test set were computed.

We used the following setup in all our experiments:

- Optimizer – Adam [Kingma and Ba, 2014], with learning rate = $6 \cdot 10^{-6}$, $\beta_1 = 0.9, \beta_2 = 0.999$;
- Number of epochs = 3
- Batch size = 8
- Loss – Cross Entropy

Architecture	**F1**	**Recall**	**Precision**	**Accuracy**	**ROC AUC**
Baseline (current SOTA)	0.943	0.956	0.93	-	-
Perceptron Pooler	0.943	0.940	0.946	0.950	0.984
Mean Max Pooler	0.946	0.959	0.934	0.952	0.986
Convolutional Pooler	0.948	0.955	**0.942**	0.954	**0.988**
Concatenate Pooler	**0.950**	**0.965**	0.935	**0.955**	0.985
LSTM Pooler	0.946	0.962	0.930	0.951	0.985
Concatenate + LSTM Pooler	0.948	0.959	0.937	0.954	0.985

Table 3: Performance of different architectures on the PAWS-Wiki test set.

Train sets	**F1**	**Recall**	**Precision**	**Accuracy**	**ROC AUC**
Baseline					
PAWS$_{Wiki}$	0.943	0.956	0.93	-	-
PAWS$_{Wiki}$ + unlabeled	0.937	0.940	0.933	0.944	0.986
PAWS$_{Wiki}$ + PAWS$_{QQP}$	**0.949**	**0.963**	**0.937**	**0.955**	**0.988**
Concatenate Pooler					
PAWS$_{Wiki}$	**0.950**	0.965	**0.935**	**0.955**	0.985
PAWS$_{Wiki}$ + unlabeled	0.943	0.955	0.930	0.949	0.985
PAWS$_{Wiki}$ + PAWS$_{QQP}$	**0.950**	**0.970**	0.930	0.954	**0.986**

Table 4: Metrics on the PAWS-Wiki test set depending on architecture and train sets.

5.2.1 Baselines

We have chosen **model from huggingface** as baseline as it is currently a SOTA model on PAWS dataset.

6 Results

The results corresponding to the choice of architecture and train sets are shown on Tab.3 and Tab.4.

The results shows that adding Pooler Layers improves the performance of the model, and the best architecture is Concatenate Pooler. This could happened due to the fact that resulting hidden vector was large enough to capture all necessary information about the input sequence. Also, it seems that capturing information from all sequence does not improves performance as Concatenate + LSTM Pooler showed slightly lower performance than plain Concatenate Pooler. This may be caused by overfitting as well.

Unlabeled part of dataset did not boost performance of our models. We suppose, the main reason is that the unlabeled part is too noisy. This part of dataset may be used for pre-training, but not for finetuning. PAWS$_{QQP}$ did increased the overall performance for both baseline and our solution. Therefore, the best combination of training sets are PAWS$_{Wiki}$ and PAWS$_{QQP}$.

7 Conclusion

In this work, we presented a SOTA model on PAWS$_{Wiki}$ test set. We used DeBERTa as backbone with different Pooler Layers to achieve the best result. PAWS$_{QQP}$ did increase the overall performance when included in train set.

References

[Becker et al., 2023] Becker, J., Wahle, J. P., Ruas, T., and Gipp, B. (2023). Paraphrase detection: Human vs. machine content. *arXiv preprint arXiv:2303.13989.*

[Bojanowski et al., 2017] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146.

[Chowdhury and Bhattacharyya, 2018] Chowdhury, H. A. and Bhattacharyya, D. K. (2018). Plagiarism: Taxonomy, tools and detection techniques. *arXiv preprint arXiv:1801.06323.*

[Clark et al., 2020] Clark, K., Luong, M.-T., Le, Q. V., and Manning, C. D. (2020). Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555.*

[Corbeil and Ghavidel, 2021] Corbeil, J.-P. and Ghavidel, H. A. (2021). Assessing the eligibility of backtranslated samples based on semantic similarity for the paraphrase identification task. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2021)*, pages 301–308.

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

[Gautam and Jerripothula, 2020] Gautam, A. and Jerripothula, K. R. (2020). Sgg: Spinbot, grammarly and glove based fake news detection. In *2020 IEEE Sixth international conference on multimedia big data (bigMM)*, pages 174–182. IEEE.

[He et al., 2021] He, P., Gao, J., and Chen, W. (2021). Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543.*

[He et al., 2020] He, P., Liu, X., Gao, J., and Chen, W. (2020). Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654.*

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

[Kondrak, 2005] Kondrak, G. (2005). N-gram similarity and distance. In *String Processing and Information Retrieval: 12th International Conference, SPIRE 2005, Buenos Aires, Argentina, November 2-4, 2005. Proceedings 12*, pages 115–126. Springer.

[Krishna et al., 2023] Krishna, K., Song, Y., Karpinska, M., Wieting, J., and Iyyer, M. (2023). Paraphrasing evades detectors of ai-generated text, but retrieval is an effective defense. *arXiv preprint arXiv:2303.13408*.

[Liu et al., 2022] Liu, T. et al. (2022). Towards better characterization of paraphrases. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8592–8601.

[Nighojkar and Licato, 2021] Nighojkar, A. and Licato, J. (2021). Improving paraphrase detection with the adversarial paraphrasing task. *arXiv preprint arXiv:2106.07691*.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

[Raffel et al., 2020] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

[Sak et al., 2014] Sak, H., Senior, A., and Beaufays, F. (2014). Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition.

[Vrbanec and Meštrović, 2020] Vrbanec, T. and Meštrović, A. (2020). Corpus-based paraphrase detection experiments and review. *Information*, 11(5):241.

[Wahle et al., 2021] Wahle, J. P., Ruas, T., Meuschke, N., and Gipp, B. (2021). Are neural language models good plagiarists? a benchmark for neural paraphrase detection. In *2021 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pages 226–229. IEEE.

[Wang et al., 2019] Wang, A., Pruksachatkun, Y., Nangia, N., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. (2019). Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32.

[Wang et al., 2018] Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2018). Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

[Zhang et al., 2019] Zhang, Y., Baldridge, J., and He, L. (2019). Paws: Paraphrase adversaries from word scrambling. *arXiv preprint arXiv:1904.01130*.

[Zhou et al., 2022] Zhou, C., Qiu, C., and Acuna, D. E. (2022). Paraphrase identification with deep learning: A review of datasets and methods. *arXiv preprint arXiv:2212.06933*.