LAMBERT: Layout-Aware (Language) Modeling using BERT for information extraction

Łukasz Garncarek* **Rafał Powalski*** **Tomasz Stanisławek***
Bartosz Topolski* **Piotr Halama**
Applica.ai, ul. Zajęcza 15, 00-351 Warszawa (Poland)
firstname.lastname@applica.ai

Filip Graliński
Applica.ai, ul. Zajęcza 15, 00-351 Warszawa (Poland)
Adam Mickiewicz University,
ul. Wieniawskiego 1, 61-712 Poznań (Poland)
filip.gralinski@applica.ai

April 29, 2020

Abstract

In this paper we introduce a novel approach to the problem of understanding documents where the local semantics is influenced by non-trivial layout. Namely, we modify the Transformer architecture in a way that allows it to use the graphical features defined by the layout, without the need to re-learn the language semantics from scratch, thanks to starting the training process from a model pretrained on classical language modeling tasks.

1 Introduction

The inherent sequential structure of natural language leads to the usual practice of treating text as a sequence of tokens, characters, or—more recently—subword units. In many problems related to Natural Language Processing (NLP), this linear perspective was enough to enable significant breakthroughs, such as the introduction of the Transformer neural architecture [Vaswani et al., 2017]. This linear perspective currently remains as the basis of the state-of-the-art models in NLP problems. Interestingly, contrary to recurrent neural networks, the sequential nature of language is not reflected directly in the Transformer network architecture.

In this setting, the task of computing token embeddings is solved by Transformer encoders, such as BERT [Devlin et al., 2019] and its modifications such as RoBERTa [Liu et al., 2019b], or ALBERT [Lan et al., 2019]. These encoders achieved top scores on the GLUE benchmark [Wang et al., 2019]. Other, non-BERT-derived architectures include Transformer-XL [Dai et al., 2019], XLNet [Yang et al., 2019], GPT [Radford, 2018], and GPT-2 [Radford et al., 2019].

They all deal with problems arising in texts defined as sequences of words. However, in many cases a structure more intricate than just a linear ordering of tokens is available. This is the case for printed or richly formatted documents, where relative vertical and horizontal positions of tokens contained in tables, spacing between paragraphs, or different styles of headers, all carry useful information. After all, the very goal of endowing texts with non-trivial layout and formatting is to improve readability.

*The authors ŁG, RP, TS, and BT have equally contributed to the paper and are listed in alphabetic order.

In this article we present the first attempt to enrich the most successful methods of NLP with layout understanding mechanisms. Our main contribution is the introduction of a *Layout-Aware Language Model*, a general-purpose language model that views texts not simply as sequences of words, but instead as collections of tokens distributed over 2-dimensional pages. Thus, it is able to process documents containing not only plain text, but also tables, headers, forms and various other visual elements.

A key feature of this solution is that it retains the crucial advantage of language models—the ability to learn in an unsupervised setting—thus leveraging the vast abundance of publicly available documents.

Our approach is based on injecting the layout information into an already pretrained instance of BERT, hence the name LAMBERT, and fine-tuning it on a dataset consisting of documents with non-trivial layout. Starting with a pretrained model, helps to decrease the computational resources required to train a new model. Another important advantage is that producing a dataset containing text with positional information and non-trivial layout is more difficult than creating a dataset for standard language modeling. The datasets we are currently using to fine-tune our model are significantly smaller than the ones used to train BERT, and are not large enough to train from scratch. By using a pretrained model, we preserve the representations already learned from the much larger training corpora.

We evaluate our model on an end-to-end information extraction task, where the training set consists of documents and fields to be extracted, without any additional annotations specifying the locations where these fields can be found in the documents. We compare the results with a baseline not using the layout features, relying only on the sequential order of tokens obtained from the OCR.

1.1 Related work

There are two main lines of research on understanding documents with non-trivial layout. The first one is Document Layout Analysis (DLA), the goal of which is to identify contiguous blocks of text and other non-textual objects on the page and determine their logical function and order in the document. The obtained segmentation can be combined with the textual information contained in the detected blocks. Such a method has been recently employed by Liu et al. [2019a].

Many services utilize DLA functionality for OCR (which requires document segmentation), table detection, or form field detection, and their capabilities are still being expanded. The most notable examples are Amazon Textract [Amazon, 2019], Google Cloud Document Understanding AI platform [Google, 2019], and Microsoft Cognitive Services [Microsoft, 2019]. However, they have their limitations, such as the need to create rules for extracting information from the tables recognized by the system, or requiring training datasets with annotated document segments.

Some more recent works on information extraction using DLA include, among others, [Ishitani, 2001, Cesarini et al., 2003, Hamza et al., 2008, Bart and Sarkar, 2010, Medvet et al., 2011, Peanho et al., 2012, Rusinol et al., 2013]. They mostly concentrate on specific types of documents, such as invoices or forms, where the layout plays a relatively greater part than in more general classes of documents, which might contain tables, but also large amounts of unstructured text.

The second idea is to directly combine the methods of Computer Vision and NLP, for instance by representing a text-filled page as a multichannel image, with channels corresponding to the features encoding the semantics of the underlying text, and then using convolutional networks. This method was used, among others, by [Katti et al., 2018, Denk and Reisswig, 2019].

None of the previous approaches go as far as to propose a layout-aware language model, which can be seamlessly integrated into existing solutions based on contextual word embeddings, extending their scope from plain texts to richly formatted documents. Neither the possibility of unsupervised training on the abundant publicly available documents, nor an attempt to

train models using the textual and layout features jointly have been adequately explored in the literature before.

Our idea is related to the one used by Rahman et al. [2019], although in a different setting. Namely, they considered texts accompanied by audio-visual signal, which they injected into a pretrained BERT instance, by combining it with the input embeddings. Also, recently a similar model to ours has been independently proposed by [Xu et al., 2019]. It is based on the same central idea of injecting layout information into BERT by suitably adjusting its input embeddings.

2 Problem description

2.1 Data model

We will not be working with images of pages directly, but with data obtained by transforming them with an OCR system[1], having the form of a sequence of tokens, endowed with their bounding boxes. Thus, in our data model, a document will be represented by two sequences of equal length, (t_i) and (b_i), composed of tokens and their bounding boxes given as quadruples $(x_1, y_1, x_2, y_2) \in \mathbb{R}^4$, respectively. We will consider only single-page documents, which means that longer documents will be divided into separately processed pages. For each document we assume that the page bounding box is provided, so that the normalization of coordinates is possible.

The requirement that the tokens are linearly ordered is satisfied by the output of OCR systems which we used to process the documents. Although this order does not always perfectly correspond to the reading order, it is close enough. In particular, it is reasonable to assume that it is locally correct within contiguous blocks of text, and any inconsistencies stem from incorrect ordering of the blocks. This is important, as even if the model could completely discard the token ordering, it is still necessary for the pre- and postprocessing (dividing texts into shorter fragments, and composing multiple tokens into longer entities).

A natural question arises, whether it is possible to deliberately discard the sequential information, and force the model to learn it from the joint semantic and layout representations. At first, it might seem that in our approach this strategy is infeasible because we rely on modifying already pretrained models, which require the sequential information. Nonetheless, we manage to gradually reduce the importance of the sequential order during training (see Section 3.4) and, as a result, we obtain a model which does not rely on the sequential order at all.

Finally, note that our data model preserves not only the relative positions of tokens on the page, significant in case of tables, but also some style information, implicitly encoded in the relative heights of bounding boxes, which may be helpful for determining important contextual information such as section titles.

2.2 Problem statement

Suppose we are given a document in the form of a sequence of tokens and their bounding boxes as described in Section 2.1, together with the bounding box of the page. Our aim is to construct a sequence of corresponding contextualized embeddings $v_i \in \mathbb{R}^d$ for some fixed embedding dimension d, taking into consideration not only the semantics of the text, but also the positional information encoded in the sequence of bounding boxes of tokens.

[1] in our case, Tesseract 4.1.1-rc1-7-gb36c

3 Proposed method

3.1 Background

The basic Transformer encoder, used in, for instance, BERT [Devlin et al., 2019], is a sequence-to-sequence model transforming a sequence of input embeddings $x_i \in \mathbb{R}^n$ into a sequence of output embeddings $y_i \in \mathbb{R}^m$ of the same length, for the input/output dimensions n and m. One of the main distinctive features of this architecture is that it discards the order of its input vectors, enabling parallelization level unattainable for traditional recurrent neural networks.

In such a setting, the information about the order of tokens is preserved not by the structure of the input, but instead passed explicitly to the model, by defining the input embeddings as

$$x_i = s_i + p_i, \tag{1}$$

where $s_i \in \mathbb{R}^n$ is the semantic embedding of the token at position i, taken from a trainable embedding layer, while $p_i \in \mathbb{R}^n$ is a positional embedding, depending only on i.

Since in BERT, on which we base our approach, the embeddings p_i are trainable, the number of pretrained embeddings (in this case 512) defines a limit on the length of input sequence. In general, there are many ways to circumvent this limit, such as using predefined [Vaswani et al., 2017] or relative [Dai et al., 2019] positional embeddings.

3.2 Proposed solution

In order to inject layout information into a pretrained BERT model, we apply two modifications to the input embeddings defined in (1). Let us begin by writing down the new formula

$$x_i = s_i + D_q(p_i) + L(c_i). \tag{2}$$

The first adjustment is the application of *unnormalized* dropout D_q with probability q to the positional embeddings p_i. By 'unnormalized' we mean that after replacing some elements with 0, we do not perform any rescaling of the resulting vector in order to preserve its norm; we simply let it decrease. This modification is discussed in more detail in Section 3.4.

The second change is adding a new term $L(\ell_i)$ to the input embeddings. Here, $\ell_i \in \mathbb{R}^k$ stands for *layout embeddings*, described in more detail in Section 3.3. They carry the information about the position of the i-th token on the page. The dimension k of the layout embeddings is allowed to differ from the input embedding dimension n, and thus we apply a trainable linear layer $L\colon \mathbb{R}^k \to \mathbb{R}^n$ to ℓ_i in order to adjust it to the input embedding dimension n.

Apart from establishing the proper dimension, the adapter layer L serves a similar purpose as the dropout D_q applied to the positional embeddings. Namely, it enables the pretrained model to gradually learn how to use additional information coming from the layout embeddings. We initialize the weight matrix of L according to a normal distribution $\mathcal{N}(0, \sigma^2)$, with the standard deviation σ being a hyperparameter. We have to choose σ carefully, so that in the initial phase of training, the ℓ_i term does not interfere too much with the already learned representations. We experimentally determined the value $\sigma = 0.02$ to be near-optimal.

3.3 Layout embeddings

Recall that in our setting, a document is represented by a sequence of tokens t_i and their bounding boxes b_i. To each element of this sequence, we assign its layout embedding ℓ_i, carrying the information about the position of the token with respect to the whole document. This can be done in various ways. What they all have in common is that the embeddings ℓ_i should depend only on the bounding boxes b_i and not on the tokens t_i.

We experimented with a number of different layout embeddings, but in the end, the more sophisticated methods did not perform better than the simplest one. The discussion of these

other approaches is postponed to Appendix A, while here we deal with the simplest, yet the most effective case.

We base our layout embeddings on the method originally used in [Vaswani et al., 2017] to define the positional embeddings. First, to make things formal, let d be an even positive integer. For $j = 1, \ldots, d$ and a sequence of scaling factors $\theta \in \mathbb{R}^{d/2}$, let us introduce an auxiliary function $F^\theta \colon \mathbb{R} \to \mathbb{R}^d$ given by the formula

$$F^\theta_j(t) = \begin{cases} \cos(\theta_r t) & \text{if } j = 2r \\ \sin(\theta_r t) & \text{if } j = 2r + 1. \end{cases} \tag{3}$$

It maps the real line into a product of $d/2$ unit circles, by winding it around each of the circles with stretch factor defined by the corresponding θ_r.

Just to give a better intuition, let us mention a real-life example where this function occurs. Consider a wall clock with three hands of unit length, representing hours, minutes, and seconds. If we assume that the center of the clock is at the origin, and time is measured in seconds, for $\theta = (2\pi/3600, 2\pi/60, 2\pi)$ the corresponding function F^θ describes the concatenated coordinates of the endpoints of the clock's hands at a given time.

In [Vaswani et al., 2017, Section 3.5], the positional embeddings were defined as

$$p_{ij} = F^\theta_j(i) \tag{4}$$

with θ_r being a geometric progression interpolating between 1 and some big number M. This approach was subsequently used in other Transformer-based models with precomputed positional embeddings.

To compute our layout embeddings, we first normalize the bounding boxes by scaling (preserving the aspect ratio) and translation. This causes the page bounding box to take the form $(0, 0, w, h)$, where w and h are the normalized width and height of the page, the larger of them equal to 1. Then, we use the function F^θ to transform the coordinates of the normalized bounding boxes and finally concatenate the four resulting vectors. We use scaling factors (θ_r) forming a geometric progression interpolating between 0.25 and 500 of length $n/8$, where n is the dimension of the input embeddings.

In other words, if (x_1, y_1, x_2, y_2) are the coordinates of the normalized token bounding box,

$$\ell_i = \left[F^\theta(x_1) \mid F^\theta(y_1) \mid F^\theta(x_2) \mid F^\theta(y_2)\right] \in \mathbb{R}^n, \tag{5}$$

where the symbol $\mid$ denotes vector concatenation.

3.4 Suppressing the sequential order

In eq. (2) we adjusted the positional embedding term p_i by applying dropout D_q with probability q. As it turned out in the experiments, we might either discard whole positional embeddings for randomly selected tokens, hide certain random elements of all the vectors p_i, or even mask random elements in each vector independently—the specific kind of dropout employed did not have a statistically significant impact on the results.

What matters is that by adjusting the probability q during training, we may gradually suppress the positional term; since our dropout is unnormalized, it weakens the signal coming from the sequential position, and forces the model to rely on other input features to replace the missing information. In the end, when $q = 1$, we obtain a model which does not use the sequential position at all, and we may just discard the positional term altogether. In particular, we no longer rely on the imperfect reading order predicted by the OCR system being used.

In our experiments using dropout on the positional embeddings, we set $q = 0$ in the beginning, and linearly increased it during the first half of the training, until reaching $q = 1$, which remained fixed for the remaining half.

4 Experiments

We based our evaluation on an end-to-end information extraction task. As our pretrained BERT instance we used RoBERTa in its smaller, base variant. The implementation from the *pytorch-transformers* library by Hugging Face [2019] was used. We have used the same entity-extraction pipeline as Graliński et al. [2020].

The models were trained on a language modeling task extended with layout information; and subsequently, on downstream information extraction tasks. In the remainder of the paper, these two stages will be referred to as *training* and *fine-tuning*, respectively.

4.1 Training

To train the models, we used a masked language modeling task, similar to the one used in the original BERT training, with input tokens endowed with their bounding boxes.

4.1.1 Datasets

The models were trained on a combination of 8 datasets (some of them publicly available and some private) containing a variety of documents with non-trivial layout:

EDGAR This dataset was retrieved from the Electronic Data Gathering, Analysis and Retrieval system (EDGAR) [SEC, 2019], made publicly available by the U.S. Securities and Exchange Commission (SEC). It consists of forms (10-K, 20-F, and 40-F) submitted by companies supervised by SEC. Its size[2] is 119,088 pages.

RVL-CDIP The Ryerson Vision Lab Complex Document Information Processing (RVL-CDIP) dataset [Harley et al., 2019] consists of 400k scanned pages of documents of various kinds, including letters, forms, invoices, advertisements, scientific reports, and many others. It contains 90,054 pages.

Common Crawl PDFs This is a dataset produced by downloading PDF files pointed to by links in Common Crawl [2019]. From each domain, only one file was randomly chosen and downloaded, yielding a total amount of 389,469 pages.

cTDaR The dataset from ICDAR 2019 Competition on Table Detection and Recognition (cTDaR) [ICDAR, 2019a], containing various kinds of tables, both modern and historical. We used only modern ones, amounting to 782 pages.

Private datasets We also included four datasets containing private documents, mostly financial reports and filled forms, consisting of 151,074 pages.

4.1.2 Data preprocessing

The datasets underwent a procedure of removing undesirable documents. First, we discarded handwritten material as well as scanned documents with low OCR quality. Among the documents that could be reliably transformed into our data model, we filtered out all that could not be identified as written in English.

Based on the observation that pages containing plain text have more tokens than those with more interesting layouts, we removed all pages with either more than 1000 or less than 50 tokens. Some more complex heuristics based on the distribution of the number of tokens in separate lines were also used to further avoid pages suspected of containing just a large block of text.

[2] Dataset sizes are given after preprocessing described in Section 4.1.2

All the documents were tokenized using the Byte Pair Encoding (BPE) tokenizer distributed together with the corresponding pretrained model, thus subdividing the original tokens. The bounding boxes were interpolated, under the simplifying assumption that all characters in a token have equal width.

Since BERT imposes a limit on the number of tokens in the input sequence, we simply divided longer pages into chunks, according to the sequential order of tokens.

4.1.3 Setup

All experiments were performed on a single Tesla V100 GPU with 16 GB memory. We used the Adam optimizer with the weight decay fix from Hugging Face [2019]. We employed a learning rate scheduling method similar to the one used by Devlin et al. [2019], increasing the learning rate linearly for the warm-up period of 10% of the training time and then decreasing it linearly to 0. The dropout on the positional embeddings was initially set to 0, and linearly increased to 1 during the first half of training, after which it remained fixed.

The models were trained with batch size of 128 sequences (i.e. 64K tokens) for approximately 150k steps, corresponding to 20 epochs over a 0.5 billion word corpus. For comparison, this is around 1/13 of training needed to pretrain the original 12-layer BERT model. Each training took approximately 5 days to complete.

4.2 Fine-tuning

After training our models, we fine-tuned them on downstream information extraction tasks, in order to evaluate their performance in an end-to-end setting.

4.2.1 Datasets

For fine-tuning and evaluation on downstream tasks, we used the *Kleister* datasets [Graliński et al., 2020]. They are a new collection of datasets created in order to evaluate end-to-end solutions for extracting information from documents with complex layout, containing noisy multimodal input, forms, tables, etc. The authors have partitioned their datasets into train, development, and test subsets, and we use their partition.

Note that information extraction is more difficult than entity recognition. In the latter case, the desired information occurs directly in the text, while in the former, retrieving the information may require normalization, and in more complex scenarios, synthesis of the clues dispersed in the document. The Kleister datasets merely provide the correct values of extractable attributes (or the information that the attribute does not appear in the document), without specifying their source in the document text.

Kleister-NDA The dataset is composed of non-disclosure agreements found in attachments to other contracts or forms in the EDGAR database (see Section 4.1.1). Their layout is quite regular, and being legal agreements, they have hierarchical structure with many sections and subsections.

Each document contains four attributes to be extracted, namely the duration of the contract; the date after which it becomes legally binding; parties of the contract; and jurisdiction whose law governs the contract. The training subset contains 1,486 pages and 982 extractable attributes; the development subset has 431 pages and 334 attributes; and the test subset consists of 747 pages and 844 attributes.

Kleister-Charity This dataset is composed of financial reports of charities registered in England and Wales, made publicly available by the Charity Commission for England and Wales. There is no fixed format of such a report, and the documents in the dataset are diverse. Some

have only a few pages filled with the key information, while others are long, richly illustrated, and full of tables and charts.

The eight extractable attributes in the documents include separate address components (street line, post code, post town); charity names; identification numbers; report dates; and annual income and spending amounts. There are 36,911 pages and 13,449 extractable attributes in the training set; 10,639 pages and 3,423 attributes in the development set; and 14,672 pages and 4,740 attributes in the test set.

SROIE The dataset from the ICDAR 2019 Competition on Scanned Receipts OCR and Information Extraction [ICDAR, 2019b]. It consists of scanned receipt images. Since the dataset is used also for OCR evaluation, its annotations include the gold truth bounding boxes of tokens. For training, we have manually validated these annotations, and incorporated some corrections, so that we could use them instead of processing the images with Tesseract. The testing was performed against the original ground truth annotations, to allow comparability with other methods evaluated on this dataset.

Each receipt contains four attributes for extraction: company name, date, address, and total cost. The dataset is split into a training set, containing 626 receipts, and a test set with the remaining 347 receipts.

4.2.2 Data preprocessing

We chose to approach the problem of information extraction through entity recognition, by tagging entity types of tokens. This method requires a suitably tagged training dataset, hence we had to perform automatic tagging of documents in the Kleister datasets. We followed exactly the same procedure as described by Graliński et al. [2020]. Note, that as a consequence, the quality of the *auto-tagging* impacts the overall end-to-end results.

4.2.3 Setup

The model was extended with a simple classification head on top, consisting of a single linear layer, and fine-tuned on the task of classifying entity types of tokens. Fine-tuning on each of the Kleister datasets was performed separately, to more closely replicate the real-world situation where additional datasets are unavailable.

We used the same hardware configuration as for model training (Section 4.1.3). The models were trained with early stopping, based on the F_1-score achieved on the development set. We used the Adam optimizer, but this time without the learning rate warm-up, as it turned out to have no impact on the results.

4.3 Evaluation

The extended model operates as a tagger on the token level, allowing to classify separate tokens, while the datasets contain only the values of extractable attributes. Therefore, further processing of output is required. Every contiguous sequence of tokens tagged as a given entity type is treated as a recognized entity and assigned a score equal to the geometric mean of the scores of its constituent tokens.

Then, every recognized entity undergoes a normalization procedure specific to its general data type (e.g. date, monetary amount, address, etc.), and duplicates are aggregated by summing their scores, giving bias to entities detected multiple times. Finally, the highest-scoring normalized entity is selected as the output of the information extraction system.

The predictions obtained this way are compared with target values provided in the dataset using F_1-score as the evaluation metric.

4.4 Results

Model		F_1-score: mean(standard deviation)				
Name	Remarks	NDA	Charity			SROIE
			income	spending	all	
RoBERTa (baseline)	fine-tune only	0.742(19)	0.573(26)	0.555(11)	0.757(12)	0.940(3)
	train & fine-tune	0.767(9)	0.608(12)	0.618(2)	0.778(1)	0.935(7)
LAMBERT	no suppression	0.750(13)	0.627(8)	0.626(8)	0.783(2)	**0.946(3)**
	dropout	0.757(6)	**0.704(15)**	**0.680(13)**	**0.799(4)**	**0.950(2)**

Table 1: End-to-end evaluation results on Kleister and SROIE datasets. The results for each model are averaged over 3 runs, and given together with the standard deviations (expressed as multiples of the least significant digit). For NDA the uncertainty intervals for all models overlap, and none of the models is marked as best. The first baseline RoBERTa was only fine-tuned on the Kleister datasets (see Section 4.2). The second one was trained on the training sets (see Section 4.1; the documents were OCRed using Tesseract) before being fine-tuned (according to the terminology defined in Section 4). The first LAMBERT variant was trained without the application of dropout to the positional embeddings, while in the second variant the sequential information was suppressed (see Section 3.4).

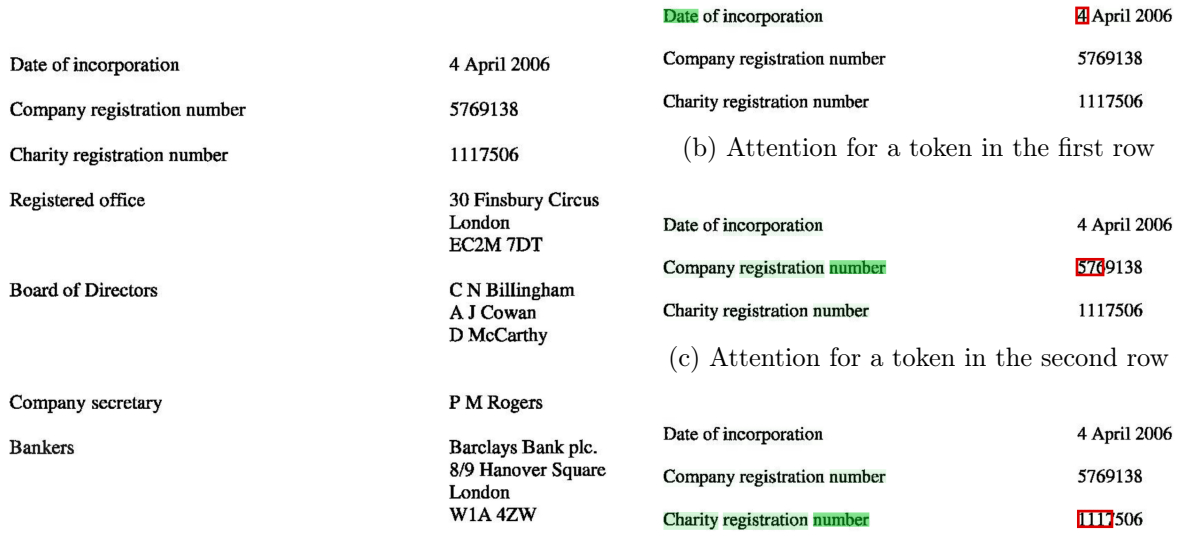
In Table 1, we present the evaluation results achieved on downstream tasks by the LAMBERT model. Each experiment was repeated 3 times, and the results were averaged. The standard deviations are also provided. As a baseline, we also included the evaluation of two variants of original pretrained models, utilizing only the sequential positional information (i.e. without the layout embeddings), the first one both trained and fine-tuned, and the second only fine-tuned.

In the SROIE dataset, LAMBERT performs significantly better that the baseline. Regarding suppression of sequential embeddings, both variants score similarly—the difference is not significant. We think that this can be partially attributed to the better sequential ordering of tokens in the ground truth annotations.

In the Charity dataset, we singled out the results for extraction of income and spending amounts. These attributes usually appear in large tables containing many other numerical values. Moreover, for such tables, the reading order produced by the Tesseract OCR is often column by column, so the proper alignment of tokens into rows must be deduced by the model from the layout information. This is the kind of problem where adding the layout embeddings should significantly improve the results. This is indeed the case, especially with disabled positional embeddings.

To further investigate this claim, we prepared some visualizations of the attention weights. In Figure 1a, we present a document with tabular data laid out in two columns. The OCR recognized the document as a two-column text, i.e. it produced a reading order in which all the tokens in the left column precede the tokens in the right column. Relying only on this order is not enough to tie the keys in the left column with corresponding values on the right.

We used our layout-only model to compute the embeddings of tokens in this document, and visualized its attention weights. We have found that some heads rely on relative positions of tokens. An example is pictured in Figure 1, where we can see that the head correctly ties the values in the right column with the corresponding row labels. This seems impossible, or at least very difficult for a model knowing only the sequential order of tokens. More visualizations, including both positive and negative examples, are included in Appendix B.



(a) Original document

(b) Attention for a token in the first row

(c) Attention for a token in the second row

(d) Attention for a token in the third row

Figure 1: Visualizations of one of the attention heads. The word for which the embedding is computed is marked with a contour and attended words are highlighted. In this particular example the model was able to recognize row captions and cell values in a tabular layout.

5 Conclusions

The first conclusion is that providing the layout information improves the model performance on tasks where graphical features are useful, while it does not lead to deterioration when dealing with flat layout. Not only this, but after completely replacing the sequential position signal with the layout embeddings in the rich-layout Charity dataset, the results improve even more.

It means that whatever can be inferred from the sequential order of tokens, the model is able to deduce from their positions on the page, and the original positional embeddings seem to be only a distraction when they are accompanied by the layout embeddings.

We propose the following intuition to explain this phenomenon. In eq. (2) we are summing three terms containing different pieces of information, which has to be subsequently extracted by the model. The more summands are used, the more difficult it is to learn how to retrieve the information they carry. If one of the terms is redundant, it imposes the cost of an additional summand without giving any new benefits in return.

Finally, note that while using the positional embeddings in the Charity dataset improves the results of the plain RoBERTa model, in the NDA dataset we don't see a significant difference. The RoBERTa behavior again confirms that the model can effectively utilize the layout features, without forsaking what it learned during pretraining. In the case of BERT, this could be caused by the small size of the NDA dataset.

References

Amazon. Amazon textract. https://aws.amazon.com/textract/ (accessed November 25, 2019), 2019.

E. Bart and P. Sarkar. Information extraction by finding repeated structure. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*, DAS '10, page 175–182, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605587738. doi: 10.1145/1815330.1815353. URL https://doi.org/10.1145/1815330.1815353.

F. Cesarini, E. Francesconi, M. Gori, and G. Soda. Analysis and understanding of multi-class invoices. *IJDAR*, 6:102–114, 10 2003. doi: 10.1007/s10032-002-0084-6.

Common Crawl. Common Crawl. https://commoncrawl.org (accessed December 6, 2019), 2019.

Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. Le, and R. Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1285. URL https://www.aclweb.org/anthology/P19-1285.

T. I. Denk and C. Reisswig. BERTgrid: Contextualized embedding for 2D document representation and understanding, 2019. arXiv:1909.04948.

J. Devlin, M.-W. Chang, K. Lee, and 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)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://www.aclweb.org/anthology/N19-1423.

Google. Cloud Document Understanding AI. https://cloud.google.com/document-understanding/docs/ (accessed November 25, 2019), 2019.

F. Graliński, T. Stanisławek, A. Wróblewska, D. Lipiński, A. Kaliska, P. Rosalska, B. Topolski, and P. Biecek. Kleister: A novel task for information extraction involving long documents with complex layout, 2020.

H. Hamza, Y. Belaïd, A. Belaïd, and B. B. Chaudhuri. An end-to-end administrative document analysis system. In *2008 The Eighth IAPR International Workshop on Document Analysis Systems*, pages 175–182, Sep. 2008. doi: 10.1109/DAS.2008.43.

A. W. Harley, A. Ufkes, and K. G. Derpanis. Evaluation of deep convolutional nets for document image classification and retrieval. In *International Conference on Document Analysis and Recognition (ICDAR)*, 2019. http://www.cs.cmu.edu/~aharley/rvl-cdip/ (accessed November 26, 2019).

Hugging Face. Transformers. https://github.com/huggingface/transformers (accessed November 27, 2019), 2019.

ICDAR. Competition on Table Detection and Recognition. http://sac.founderit.com/index.html (accessed November 26, 2019), 2019a.

ICDAR. Competition on Scanned Receipts OCR and Information Extraction. https://rrc.cvc.uab.es/?ch=13 (accessed April 6, 2020), 2019b.

Y. Ishitani. Model-based information extraction method tolerant of ocr errors for document images. In *2013 12th International Conference on Document Analysis and Recognition*, pages 908–915, Los Alamitos, CA, USA, sep 2001. IEEE Computer Society. doi: 10.1109/ICDAR.2001.953918. URL https://doi.ieeecomputersociety.org/10.1109/ICDAR.2001.953918.

A. R. Katti, C. Reisswig, C. Guder, S. Brarda, S. Bickel, J. Höhne, and J. B. Faddoul. Chargrid: Towards understanding 2D documents. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018. doi: 10.18653/v1/d18-1476. URL http://dx.doi.org/10.18653/v1/d18-1476.

Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. ALBERT: A lite BERT for self-supervised learning of language representations, 2019.

X. Liu, F. Gao, Q. Zhang, and H. Zhao. Graph convolution for multimodal information extraction from visually rich documents. *Proceedings of the 2019 Conference of the North*, 2019a. doi: 10.18653/v1/n19-2005. URL http://dx.doi.org/10.18653/v1/n19-2005.

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach, 2019b.

E. Medvet, A. Bartoli, and G. Davanzo. A probabilistic approach to printed document understanding. *IJDAR*, 14:335–347, 12 2011. doi: 10.1007/s10032-010-0137-1.

Microsoft. Cognitive Services. https://azure.microsoft.com/en-us/services/cognitive-services/ (accessed November 25, 2019), 2019.

C. Peanho, H. Stagni, and F. Silva. Semantic information extraction from images of complex documents. *Applied Intelligence*, 37:543–557, 12 2012. doi: 10.1007/s10489-012-0348-x.

A. Radford. Improving Language Understanding by Generative Pre-Training. 2018.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.

W. Rahman, M. K. Hasan, A. Zadeh, L.-P. Morency, and M. E. Hoque. M-BERT: Injecting multimodal information in the BERT structure, 2019.

M. Rusinol, T. Benkhelfallah, and V. Poulain d'Andecy. Field extraction from administrative documents by incremental structural templates. In *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, pages 1100–1104, 2013. doi: 10.1109/ICDAR.2013.223.

SEC. Electronic Data Gathering, Analysis and Re-trieval system. https://www.sec.gov/edgar.shtml (accessed November 26, 2019), 2019.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of ICLR*, 2019. https://gluebenchmark.com/ (accessed November 26, 2019).

K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks?, 2018.

Y. Xu, M. Li, L. Cui, S. Huang, F. Wei, and M. Zhou. Layoutlm: Pre-training of text and layout for document image understanding, 2019.

Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le. XLNet: Generalized autoregressive pretraining for language understanding, 2019.

A Other layout embeddings

We experimented also with other, more complex layout embeddings, which however did not improve our results in any significant way. In this appendix, the original positional embeddings described in the main text of the paper will be referred to as *winding embeddings*.

A.1 Autoencoder embeddings

Autoencoder embeddings[3] are defined using methods from image processing applied to a simplified image representation of the document, created from the bounding boxes of tokens. First, the average line height h in the document is determined. Then, for each token, we consider its square neighborhood with side length Nh, where the optimal value for N was experimentally determined to be $N = 22$ (corresponding to 350px in the initial experiments). The neighborhood is transformed into a 64×64 black and white bitmap by placing black rectangles corresponding to token bounding boxes onto a white background. This image representation of the neighborhood is then encoded by an autoencoder, trained on a subset of the training set, yielding the layout embedding of the token.

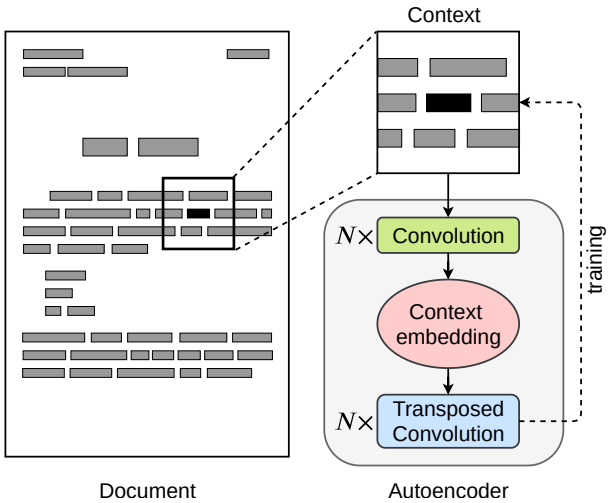


Figure 2: Autoencoder embeddings

We use a convolutional autoencoder, whose encoder is a stack of convolutional layers with stride 2, each of them halving the dimensions and doubling the number of channels of its input. The decoder is built analogously from transposed convolutional layers. Since we fix the dimensions of the neighborhood representation to $64 \times 64 \times 1$ (one binary channel), both the encoder and decoder contain $\log_2 64 = 6$ layers. See Figure 2 for the diagram of this architecture.

A.2 Graph embeddings

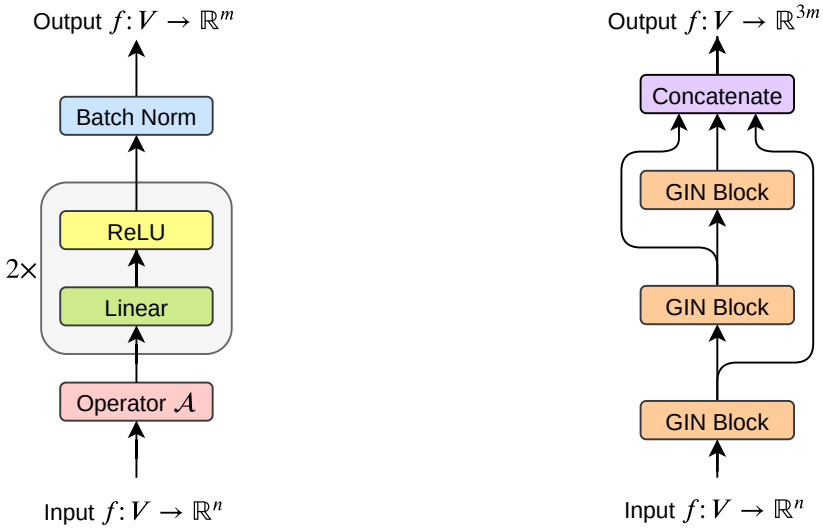
Graph neural networks (GNNs) provide a framework to deal with data whose structure is described by a graph. More precisely, let $G = (V, E)$ be a graph, and let $f\colon V \to \mathbb{R}^d$ be a function that assigns to each vertex $v \in V$ its feature vector $f(v) \in \mathbb{R}^d$. Generally speaking, a GNN computes vertex embeddings for the graph G endowed with vertex feature vectors described by f. In other words, it defines an operator from $\mathbb{R}^d$-valued functions on V to $\mathbb{R}^t$-valued functions on V for some output dimension t.

Graph structure Given a document with segmentation, we turn it into a graph whose vertices correspond to segments. To define the edges, we first independently normalize the vertical and

[3] Autoencoder embeddings were introduced in the (unpublished) MSc thesis of Bartosz Topolski.

horizontal coordinates, so that on average the bounding boxes of segments are square. Then, we choose a distance function (not necessarily a metric) and join each vertex to its K nearest neighbors (the upper left corners of bounding boxes were used for calculating distances).

As a distance we chose ℓ^p with $p = 1/2$. It promotes neighbors in the same row or column as the original segment, over the diagonally positioned ones. We experimentally determined the optimal value $K = 5$.



(a) GIN block (b) GIN module

Figure 3: The architecture of a single GIN block and a GIN module.

Architecture Our layout embeddings are based on the Graph Isomorphism Network (GIN) [Xu et al., 2018], whose main building block is the operator $\mathcal{A}$ defined by

$$\mathcal{A}f(v) = (1 + \epsilon)f(v) + \sum_{w \sim v} f(w), \tag{6}$$

which for each vertex v of the graph aggregates the features of its neighbors by summing them, and combines them with the feature vector of v. In our setting, we assign $\epsilon = 0$.

In the original GIN architecture, a single block consists of the operator $\mathcal{A}$ followed by a multi-layer perceptron, and this is also the case in our network. We use two linear layers with ReLU activations, followed by batch normalization. In Figure 3, the architectures of a *GIN block* and a *GIN module* consisting of three GIN blocks are presented. The network computing the layout embeddings is a stack of 2 GIN modules followed by a linear layer. The input dimension of the network is 4 (the feature vector of a segment is simply its bounding box), the output dimension of the final linear layer is 128, and all the hidden dimensions (except the output dimensions of GIN modules) are set to 64.

The evaluation results of the LAMBERT model using other layout embeddings are presented in Table 2. It seems that suppressing the sequential information is not as helpful as in case of the winding embeddings, and especially in the case of the autoencoder embeddings it turns out to be harmful. We believe that this can be explained by the fact, that both graph and autoencoder embeddings capture only the local layout information, depending on their configuration. The autoencoder embeddings see only a fixed neighborhood of the token, while the graph neural network used to compute the graph embeddings aggregates information from the vertex neighborhood of radius equal to the total number of GIN blocks used.

Model		F_1-score: mean(standard deviation)			
Embedding	Dropout	NDA	Charity		
			income	spending	all
winding	○	0.750(13)	0.627(8)	0.626(8)	0.783(2)
	●	0.757(6)	0.704(15)	0.680(13)	0.799(4)
autoencoder	○	0.747(9)	0.630(18)	0.624(4)	0.785(11)
	●	0.744(11)	0.572(33)	0.519(16)	0.753(6)
graph	○	0.753(17)	0.619(2)	0.608(11)	0.780(4)
	●	0.711(39)	0.677(13)	0.644(14)	0.776(9)

Table 2: Evaluation results for three kinds of layout embeddings: winding embeddings (Section 3.3), autoencoder embeddings (Section A.1), and graph embeddings (Section A.2). The *Dropout* column refers to applying dropout to suppress the sequential information, as described in Section 3.4. Each experiment was repeated 3 times.

B Attention visualizations

In this appendix we present three examples where our model was able to effectively make use of the positional features. By no means do we claim that such a behavior can be found in all documents—but only that the model does actually learn to use the context embeddings to improve its scores.

CHELMSFORD EDUCATIONAL FOUNDATION

RECEIPTS AND PAYMENTS ACCOUNT FOR THE YEAR ENDED 31ST DECEMBER 2018

	Note	2018		2017
Receipts		£		£
Investment income	1	27,036		26,504
Interest received		50		30
Donations		580		240
Total receipts			**27,666**	**26,774**
Payments				
Grants	2	20,700		21,400
Secretary's fee		2,330		2,213
Accountancy fee		300		300
Secretary's expenses and stationery		53		58
Secretary office allowance		240		240
Use of premises		440		420
Subscriptions		-		-
Data protection fee		35		35
Investments		-		-
Total payments		-	**24,098**	- **24,666**
Net payments			3,568	2,108
Cash funds at 1st January			21,866	19,758
Cash funds at 31st December			**25,434**	**21,866**

(a) Original document. Sequential token order follows table columns.

	Note	2018		2017
Receipts		£		£
Investment income	1	27,036		26,504
Interest received		50		30
Donations		580		240
Total receipts			**27,666**	**26,774**

(b) Attention for the income value.

Payments				
Grants	2	20,700		21,400
Secretary's fee		2,330		2,213
Accountancy fee		300		300
Secretary's expenses and stationery		53		58
Secretary office allowance		240		240
Use of premises		440		420
Subscriptions		-		-
Data protection fee		35		35
Investments		-		-
Total payments		-	**24,098**	- **24,666**

(c) Attention for the spending value.

Figure 4: Visualizations of one of the attention heads using model with suppressed positional embeddings and finetuned on Charity dataset. Selected tokens are parts of properly extracted income and spending values.

Reserves

The charity's operational reserve decreased from £13.9m to £11.6m due to the annual contribution to our final salary pension scheme of £3.1m and the cost of withdrawing from the Growth Plan being higher than the net income. As part of its pension de-risking strategy, Christian Aid withdrew from the Growth Plan, a multi-employer defined benefit pension scheme managed by the Pensions Trust and settled all debts (present and future) related to its participation in the scheme for a value of £755k. During the year we further reduced unrestricted expenditure to mitigate the impact of reduced unrestricted income on the operational reserve. Based on our reserves policy, our target operational reserve for 2015/16 was £19.5m. The trustees are in the process of reviewing the operational reserve policy and target level, meanwhile the trustees anticipate that the level of this reserve will remain below target for at least three years as we implement changes to our business model to strengthen our financial resilience.

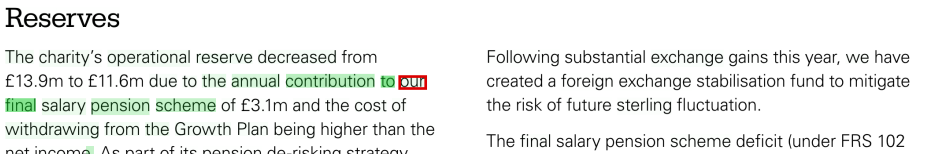
Following substantial exchange gains this year, we have created a foreign exchange stabilisation fund to mitigate the risk of future sterling fluctuation.

The final salary pension scheme deficit (under FRS 102 valuation) of £5.9m in 2014/15 has turned into a surplus of £0.7m. This follows both changes in the economic assumptions, principally the discount rate, which have decreased the liabilities in the scheme substantially and an increase in the scheme assets as we continue to contribute to the pension deficit in line with the recovery plan agreed with the Pensions Trust. The surplus in the scheme cannot be recognised in the balance sheet under FRS 102 as it is not recoverable.

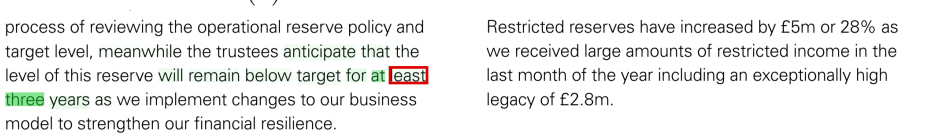
Restricted reserves have increased by £5m or 28% as we received large amounts of restricted income in the last month of the year including an exceptionally high legacy of £2.8m.

	2013/14 restated	2014/15 restated	2015/16	2015/16 change
Operational reserve	£14.0m	£13.9m	£11.6m	(17%)
Foreign exchange stabilisation fund	£0m	£0.3m	£1.2m	300%
Fixed asset reserves	£8.2m	£7.9m	£7.7m	(3%)
Other unrestricted reserves	£0.6m	£0.6m	£0.8m	33%
FRS 102 calculated pension deficits	(£2.7m)	(£6.5m)	£0m	100%
Unrestricted reserves, net of pension benefit	£20.1m	£16.2m	£21.3m	31%
Restricted reserves	£15.3m	£17.9m	£22.9m	28%
Total reserves, net of pension deficit	**£35.4m**	**£34.1m**	**£44.2m**	**30%**

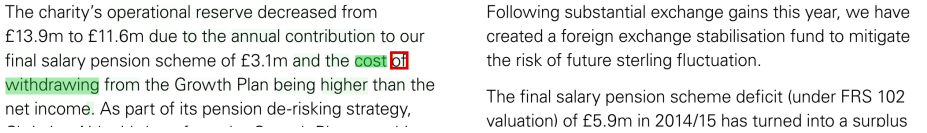
(a) Original document. Sequential token order: left body of text, table key column, right body of text and table value columns.



(b) Attention for a token at line break.

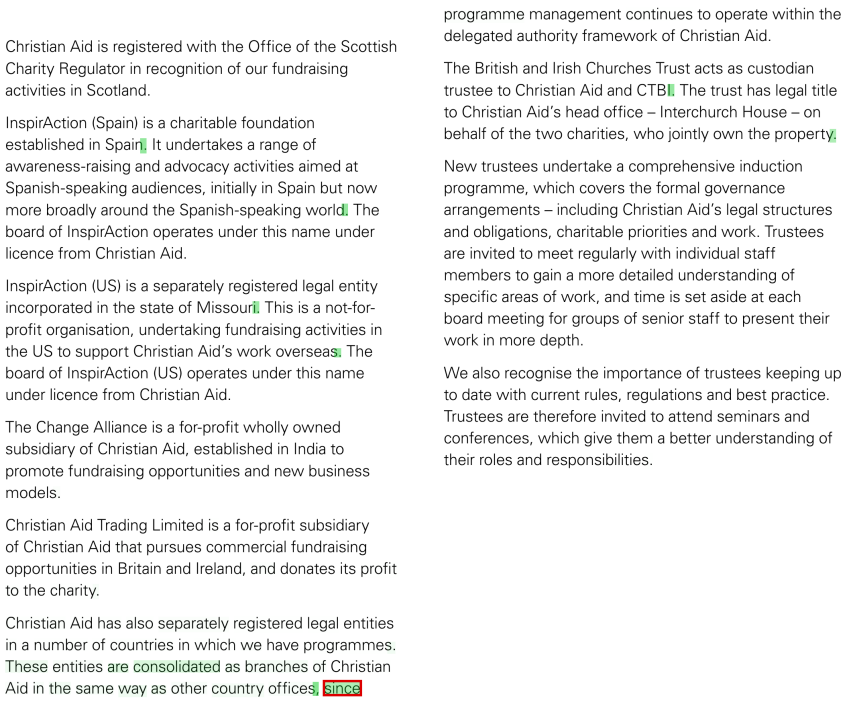


(c) Attention for a token at line break.

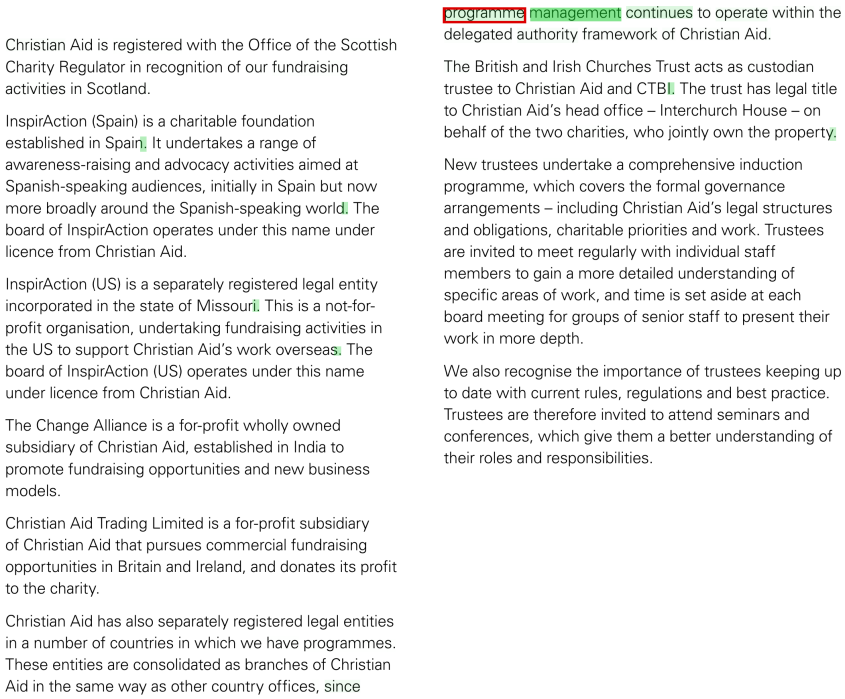


(d) Attention for a token at line break.

Figure 5: Visualizations of the attention averaged over the single layer using model with suppressed positional embeddings. Even without information about sequential token positions, the model is able to attend to correct tokens.



(a) Attention for the last token of the left text column.



(b) Attention for the first token of the right text column.

Figure 6: Visualizations of the attention averaged over the single layer using model with suppressed positional embeddings. The model is unable to attend to the parts of the sentence that are highly separated in terms of position on page.