Industry and Applications Paper

Entity Matching with Transformer Architectures - A Step Forward in Data Integration

Ursin Brunner
Zurich University of Applied Sciences
Switzerland
ursin.brunner@zhaw.ch

Kurt Stockinger
Zurich University of Applied Sciences
Switzerland
kurt.stockinger@zhaw.ch

ABSTRACT

Transformer architectures have proven to be very effective and provide state-of-the-art results in many natural language tasks. The attention-based architecture in combination with pre-training on large amounts of text lead to the recent breakthrough and a variety of slightly different implementations.

In this paper we analyze how well four of the most recent attention-based transformer architectures (BERT[6], XLNet[33], RoBERTa[17] and DistilBERT [23]) perform on the task of entity matching - a crucial part of data integration. Entity matching (EM) is the task of finding data instances that refer to the same real-world entity. It is a challenging task if the data instances consist of long textual data or if the data instances are "dirty" due to misplaced values.

To evaluate the capability of transformer architectures and transfer-learning on the task of EM, we empirically compare the four approaches on inherently difficult data sets. We show that transformer architectures outperform classical deep learning methods in EM[7, 20] by an average margin of 27.5%.

1 INTRODUCTION

Entity Matching (EM) is the task of determining if two data instances refer to the same real-world object. As a simple example consider the entries in Table 1 and Table 2. We need to match two entities based on attributes like names, addresses or product information while dealing with various formats and differing, missing or wrong values. Given this uncertainty, the challenge of entity matching is to identify which existing entity pairs have the highest probability of being a match.

The task of entity matching is crucial especially for the process of data integration and data cleaning [4]. Due to the large amount of data produced every day, the challenge of data integration and therefore entity matching becomes more urgent than ever for corporations and large institutions with data stemming from multiple sources [27].

While early solutions often relied on rule-based approaches and hand-crafted heuristics, entity matching by now is mainly based on machine learning (ML) approaches. Large projects like *Magellan*[14] provide tools and libraries for the whole EM-pipeline with good results on structured and semi-structured data.

However, what if the data consists of large textual instances, such as product descriptions, posts on Reddit, Quora, Stackoverflow, or company descriptions? What if the data has structure, but attributes are dirty, e.g. the attribute "name" consists of a given name and a last name, while "given name" is empty? In those cases, traditional EM approaches provide only mediocre results or require large efforts in hand-crafted features [2, 13]. Therefore and due to the recent advances of deep learning in natural language processing (NLP), several papers suggest end-to-end deep learning architectures [7, 20, 34] for EM.

Table 1: Database A - structured product information, with *description* being a text-blob.

Title	Brand	Description	Price
iPhone XS	Apple	The brand new iPhone now available in white, red and silver.	899.99
ZenFone 4 Pro	Asus	Thin and light, yet incredibly strong, the ZenFone 4 Pro (ZS551KL) features an expansive 5.5-inch, Full HD AMOLED display	530.00

Table 2: Database B - textual product information.

Product description
Apple's new iPhone XS - a masterpiece of design. Available now with 64 or 128GB storage and in three colors: white, silver and red
A smart device for a decent price - Nokia's Pure View 9, powered by a pure android, is the gift you need for Christmas. With it's incredible robust design and a battery duration of two days under heavy load, you will love it from day one.

The above-mentioned deep learning architectures for EM apply different modern NLP techniques: [7] is based on a bi-directional LSTM and word embeddings, [20] uses an attention mechanism on top of an LSTM and [34] introduces the power of pre-trained models.

While these deep learning approaches already lead to massive performance improvements on difficult data sets [20], none of them use modern transformer architectures for EM. Transformer architectures, largely based on the influential "Attention is All You Need" [30] paper, have proven to be effective on a large variety of NLP tasks. Especially in combination with pre-training on large text corpora, they almost always beat earlier deep learning approaches based on recurrent neural networks (RNNs) on popular NLP benchmarks [6].

The contributions of our paper are as follows:

- To the best of our knowledge, we are the first to compare four of the most recent transformer architectures, namely BERT[6], XLNet[33], DistilBERT[23] and RoBERTa[17] on the task of entity matching. We run experiments with all of them on five different EM data sets. Since the authors

of [20] have shown that most of the traditional, structured EM data sets are not challenging for advanced deep learning architectures anymore, we focus on "dirty" data sets and data sets with large textual data instances.
- We compare the results of modern transformer architectures with more traditional deep learning architectures [7, 20, 34]. Our experiments demonstrate that all transformer architectures clearly outperform classical deep learning methods in EM. On challenging datasets, the best transformer outperforms DeepMatcher[20] by an average margin of 27.5%.
- Finally, we analyze how much training is required to achieve good results on the EM tasks with heavily pre-trained transformers. To elaborate on this, we compare the results before fine tuning on the data set (zero-shot learning) and after each epoch. The results indicate that transformers reach already good results after only one epoch of training, while after 2-3 epochs they converge towards their optimum solution.

The paper is organized as follows. Section 2 reviews the related work on entity matching and gives a brief overview on transformer technologies that appear to be a valid technology for entity matching. Section 3 provides the background knowledge on attention-learning and the transformer architecture. Section 4 describes the four architectures and their pre-training algorithms in detail and explains the intuition behind each approach. In Section 5 we compare the four architectures based on their results on five selected EM data sets. Finally, we present our conclusion and future work in Section 6.

2 RELATED WORK

We can divide the work related to this paper roughly in two categories, namely *entity matching* and *transformer architectures*. We will discuss each of them in more detail.

2.1 Entity Matching

The entity matching process as a whole has been tackled over the years by many works [4, 7, 13, 14, 20], with [4] providing an excellent overview over the major challenges of entity matching. Recent projects like *Magellan*[14] focus not only on research, but provide a set of practical tools to solve entity matching in data integration processes. For a long time the field of entity matching focused on rather structured data records [8], with each attribute of the data records containing only relatively short amounts of text.

To calculate the similarity of such textual attributes, a variety of similarity functions has been developed [4]. Many of them focus on specific data structures as e.g. the *Jaro-Winkler* distance [11], known to work well on person names. The similarity values of such functions where then used as features in a binary classification problem (*match*/*no match*)[4].

With the emergence of deep learning in NLP, several papers [7, 20] started to apply new techniques such as *word embeddings* and *attention* to the task of entity matching. The advantages of such deep learning models is that they remove the need for handcrafted features. While recent work [34] introduced the power of *pre-training* for entity matching, we are to our knowledge the first paper approaching entity matching with modern *transformer architectures*.

2.2 Transformer Architectures

Based on the well-known paper "Attention is all you need"[30], transformers - a special type of neural networks - started by mid 2017 to become one of the most interesting techniques for NLP. BERT [6], combining the transformer architecture with massive unsupervised pre-training was the first paper to achieve state-of-the-art results in a large number of NLP tasks. Succeeding works by [16, 17, 33] achieved even higher results on said NLP benchmarks.

While transformers undeniably are one of the largest achievements in the 2018/2019 NLP landscape, they also started a controversy about the *more data/larger models/more computational power* mentality. Therefore, in addition to increasing model sizes like the authors of [26] did, research started also to develop more lean transformer models, which can be used on mobile devices or non-GPU servers at inference time [23].

Transformers have traditionally been used for NLP-tasks. However, we will apply this technology for entity matching - an important aspect of data integration.

3 ATTENTION AND TRANSFORMERS

In this section we provide detailed background information on a special type of neural networks called transformers. We will use this technology later on for entity matching.

The field of NLP has been dominated for a long time by architectures using recurrent neural networks (RNNs) at its core. The most popular approaches for machine translations were so-called *seq2seq*[29] or *encoder/decoder*[3] architectures, which basically consist of two RNNs, one for the encoder, and one for the decoder. Incrementally improved versions of these architectures are used in most translation software packages. For instance, Google Translate started using such models in late 2016. Even though the RNN architecture seems to be well-suited for building representations of sequential text data, several issues were discovered:

- The so-called *bottleneck problem*[1] makes it hard to transfer knowledge about the source sentence to the target sentence. The bottleneck is the context vector, which is basically all the decoder network gets as an input. By conditioning only on this single vector, the decoder then has to produce an output sequence of length N. It is intuitive to understand that the longer the sequence, the harder it is to get all necessary information of the source sentence in one single vector.
- An RNN is, due to its sequential nature, harder to parallelize and therefore takes longer to train than a simple feed forward network.
- Long range dependencies are hard to learn with RNNs, even though LSTMs and GRU (gate recurrent unit) architectures theoretically allow it. The authors of [30] describe this effect by the path length between two tokens (e.g. two words in a sentence), which is the number of steps the signal has to flow through the network. The longer the path, the harder to learn a dependency.

3.1 Attention

To overcome the bottleneck problem, [1] and [18] came up with a new technique called *attention*. Let us first elaborate on the idea of *self-attention*, to describe the intuition: A word can be represented as a weighted combination of its neighborhood. As an example, take the word *"it"* in Figure 1. For the human reader it

is intuitive that the word *it* in this sentence can be represented by the words *The* and *animal*. By applying an attention mechanism, we can train a language model to pay attention to relevant words in its neighborhood in a similar way.

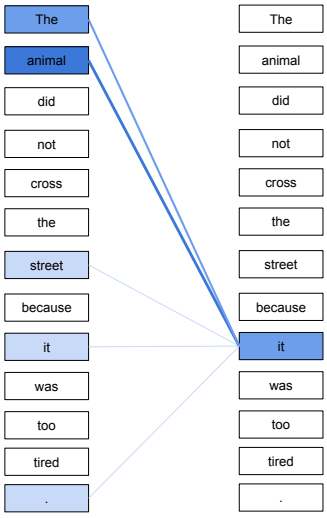

Figure 1: Self-Attention: The token "it" (see right side) can refer to the tokens "The", "animal", "street" etc. (see left side).

A sightly different version of attention was used by the first seq2seq model with attention mechanism[1]. This approach applied attention between the target sentence and the source sentence. So instead of only transferring knowledge between encoder and decoder via one context vector at the end of the encoding, the decoder has access to all the hidden states of the encoder.

Assume that we want to translate the English sentence *"The cat is sleeping"* to the German counterpart *"Die Katze schläft"*, as in Figure 2. It is helpful for the decoder to only consider information of the encoder's hidden state for *is* and *sleeping*, when the decoder is trying to produce the word *schläft*. By applying attention, the decoder will learn to focus on the important hidden states of the encoder at each given time step.

3.2 The Transformer

In 2018, Vaswani et al. published a paper with a more radical approach [30]: an architecture that completely relinquishes RNNs with the idea to build a model with attention only. The core of this paper, the so-called transformer, can be found in Figure 3. While the transformer still retains the classical encoder/decoder structure, it replaced the RNN by a so-called Multi-Head-Attention sub-layer. There are three of those attention sub-layers in each of the N stacked layers. The first two Multi-Head Attention sub-layers implement the self-attention mechanism. During the training, they will independently learn dependencies in the source and target sentence respectively, as previously shown in Figure 1. It is important to understand that in the decoder, the self-attention mechanism will only have access to all the words produced by the decoder so far, while all the words that still need to be processed are masked. Masking words with high negative values will basically "hide" them from the learning algorithm.

While the encoder contains only one Multi-Head Attention sub-layer per layer, the decoder contains a second one. This sub-layer performs attention over the output of the encoder stack.

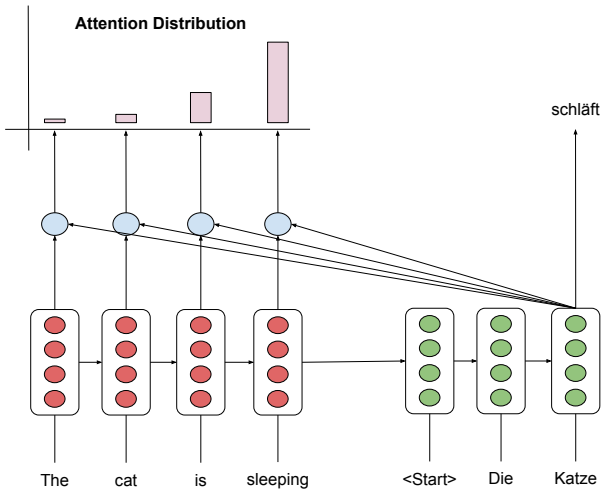

Figure 2: Sequence-to-sequence model with attention. The decoder (green) has access to all hidden states of the encoder (red) and learns how much attention to pay to each hidden state. To predict the word *schläft*, the decoder pays most attention to the hidden state of *is* and *sleeping*.

From a conceptual point of view, this sub-layer performs a similar task to the attention mechanism of classical seq2seq architectures (see previously discussed Figure 2). As a consequence, the Multi-Head Attention sub-layer allows the decoder to pay attention only to the relevant words of the source sentence.

Since a transformer model does not contain any recurrence, the authors had to introduce the idea of ordered, sequential inputs in a different way. They do this by using so-called *positional encoding* for each input word. This allows the model to make use of both the position of a word and the relative distance between two words. The authors of [30] use a combination of the functions sine and cosine with different frequencies to implement positional encodings.

A last detail to mention is the *residual connection* [9] around each sub-layer. The residual connection will, on the one hand, improve training performance, but even more importantly, it allows the positional encoding to flow untouched up to the higher layers.

4 BERT AND FRIENDS

The transformer architecture appearing first in 2017 triggered a wave of new papers based on this idea. With BERT [6] being the most popular one and setting a new state-of-the-art for many NLP tasks in 2018, other papers followed quickly and further improved performance on many tasks. In this section we will have a look at the four approaches used in our experiments: BERT, XLNet, RoBERTa and DistilBERT. The order of the subsections corresponds to the publication date of the papers.

4.1 BERT

BERT is a universal language model, pre-trained on large amounts of text data with the intention of fine-tuning it on downstream tasks (e.g. entity matching) in a supervised manner with relatively little data.

The abbreviation BERT stands for *Bidirectional Encoder Representations from Transformers*, with the emphasis on *bidirectional*.

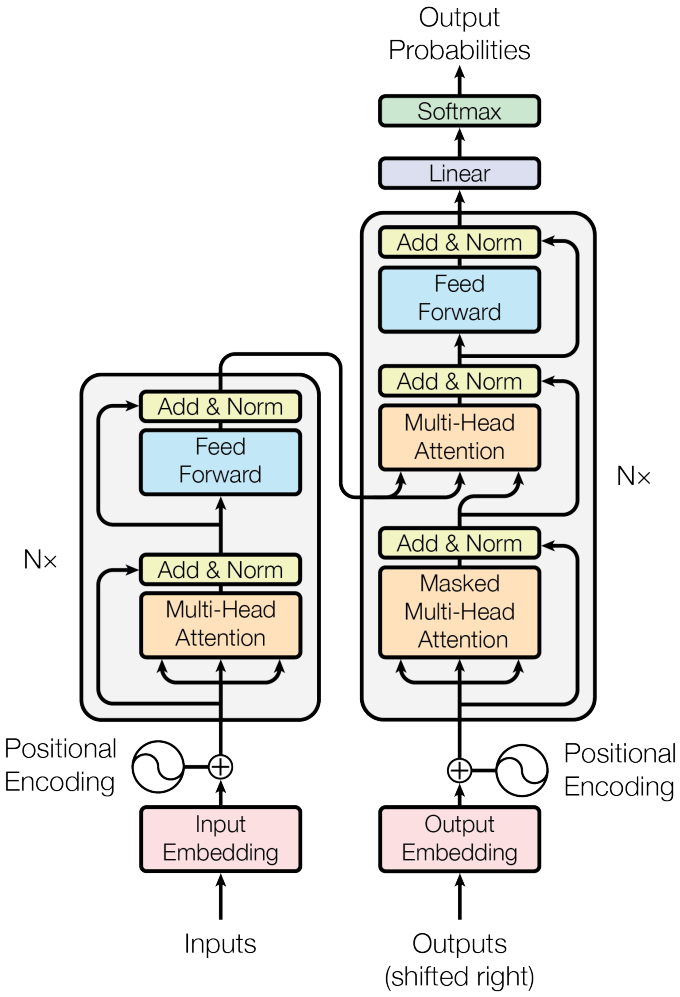

Figure 3: The original transformer architecture by Vaswani et al. as shown in the paper[30].

It is in its core a transformer language model as designed by [30], but contrary to other language models (e.g. OpenAI's GPT[21]) it is jointly conditioning on both the left- and right context of the query token during pre-training. This is somehow counter intuitive, as the most common training task of language models is to simply predict the next token (word).

Let us consider the following fraction of a sentence "[..] problems turning into banking crisis as [..]" (see Figure 4). Let us further assume that we want to predict the query token *"into"* by its left context, which is all the input to the left of the query token. With this training task one can by definition only use the left or the right context. By using both contexts together, the task would become trivial since you already know the next token.

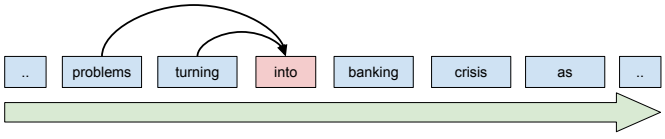

Figure 4: Left-to-right: Using the left context to predict the query token *into*.

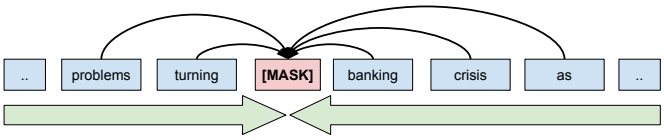

Figure 5: Bidirectional: Using both the left- and right context to predict a masked token.

To be able to condition on both the left and right context, BERT had to change the training task. Instead of predicting the next word, it tries to predict masked tokens as seen in Figure 5 by using both the left and right context. By predicting masked tokens, the BERT model is classified as an auto encoder. It learns to reconstruct the original data by restoring corrupted input, i.e. the masked tokens.

In a second training task, BERT performs *Next Sentence Prediction* (NSP). Here, the model receives two sentences as input, and has to predict if the first sentence is followed by the second one. This pre-training is necessary for all tasks which are based on the relationship between sentences. Typical examples of these tasks are *Question Answering*, *Natural Language Inference* or *Entity Matching*. It is important to understand that both training tasks are unsupervised tasks and do not require labels, but only large amounts of text data. Labeled data are only required for task-specific fine-tuning.

The ablation studies of the BERT paper demonstrate that using both the left and right context is the most important contribution of the paper. As a second contribution the BERT-team shows, that massive unsupervised pre-training on large data (BooksCorpus and Wikipedia) improves performance on a large number of tasks without the need for task-specific architectures. The BERT architecture further demonstrates to be very flexible as it allows simple fine-tuning on a range of downstream tasks such as *Question/Answering*, *Named Entity Recognition* or *Classification.*

4.2 XLNet and Transformer-XL

As seen in Section 4.1, BERT's most important contribution is its bidirectional representation. To achieve it, an auto encoder approach is necessary, with the training task to predict **[MASK]** tokens instead of predicting the next word. The XLNet [33] paper demonstrates the downsides of this approach and proposes a new architecture to solve it. According to XLNet there are two major downsides of the BERT approach:

- Predicting **[MASK]** tokens is what BERT learns during its pre-training phase. But those artificial symbols never occur in downstream tasks, which creates a pretrain-finetune discrepancy.
- BERT reconstructs all **[MASK]** tokens in parallel and independent of each other. This independence assumption is not justified as a simple example in Figure 6 demonstrates.

To overcome these flaws, XLNet returns to the more classical architecture of an autoregressive (AR) language model. In contrast to BERT's autoencoder approach, an AR model does not suffer from the downsides described above, as it does not introduce any artificial symbols and simply learns to predict the next token.

However, as we discussed in Section 4.1, an AR pre-training cannot use forward and backward contexts at the same time. To be able to capture a bidirectional context (as in BERT) and still

I went to [MASK] [MASK] and saw the [MASK] [MASK] [MASK].

*I went to **New York** and saw the **Empire State building.*** ✓

*I went to **San Francisco** and saw the **Golden Gate bridge.*** ✓

*I went to **San Francisco** and saw the **Empire State building.*** ✗

Figure 6: The independence assumption between the masked tokens is not justified.

have the advantages of an AR model, XLNet proposes a new *generalized autoregressive pre-training method.*

The core of this method is a new pre-train objective, which is called *permutation language modeling*. Instead of approaching a sequence of tokens only in a forward or backward manner, permutation language modeling takes into account all possible factorization orders of a sequence.

Assume that we have the following sentence "New York is a city". Further assume that we have already received the tokens "New" and "York". Next, we want to predict the token "is". Figure 7 shows how the model tries to predict the third token *"is"* with different factorization orders.

Let us discuss the three permutations in more detail:

- *Permutation 1* is the classical autoregressive left-to-right order. The model has access to the left side context (x_1, x_2) and tries to predict the query token x_3.
- In *Permutation 2* one can grasp the advantage of the permutation method: due to its order, the model now has access to the context x_4, x_2, x_1 to predict x_3. Be aware that in this sequence order, token x_5 is not accessible, as it appears to the right of token x_3.
- In *Permutation 3* the model has access to the tokens x_4 and x_5, as those appear to the left of token x_3 in the factorization order. By looking at this example it becomes intuitive that the token x_3 (that has to be predicted) has seen every other tokens in the sequence and will therefore, similar to an autoencoder model, capture the bidirectional context.

In addition to its core contribution, the *permutation language modeling*, XLNet includes two major improvements originally proposed in the Transformer-XL paper [5]. The Transformer-XL architecture is based on the original transformer [30], but with several improvements. The most important one, which is also implemented in XLNet, is the ability to learn dependencies beyond a fixed length without disrupting temporal coherence. This is possible due to a segment-level recurrence mechanism and a novel positional encoding scheme.

XLNet outperforms BERT on 20 tasks and achieves state-of-the-art results on 18 tasks including question answering, natural language inference, sentiment analysis, and document ranking [33].

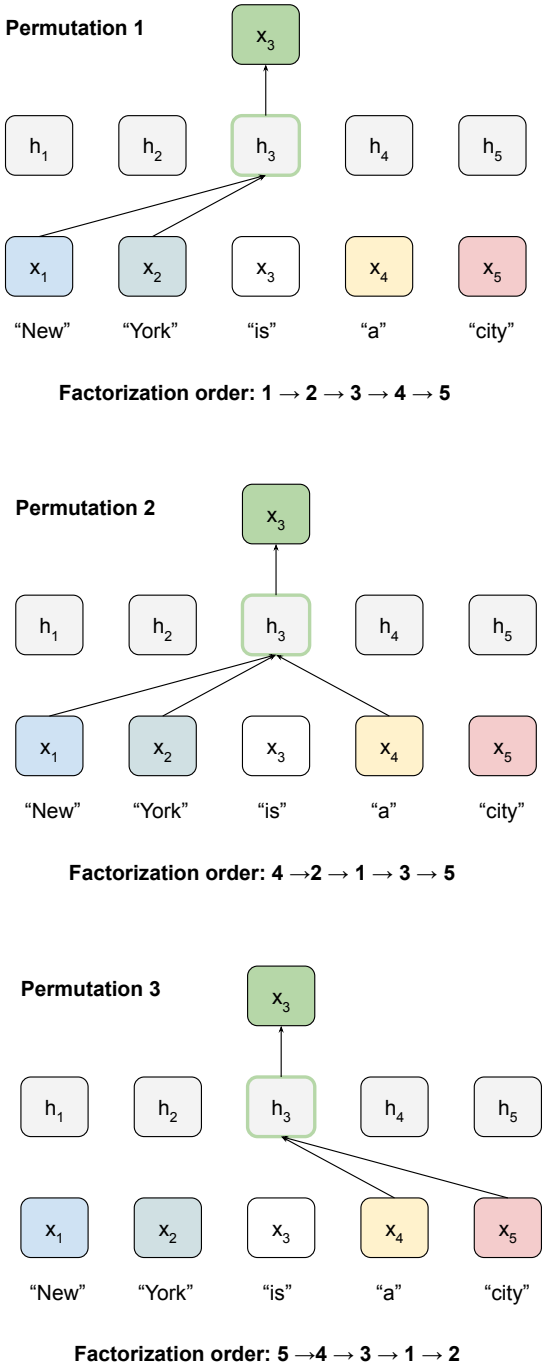

Figure 7: Permutation language modeling: predicting x_3 given the same input sequence but with different factorization orders.

4.3 RoBERTa

RoBERTa [17], a paper released at the end of July 2019, differs from other all the other papers as it does not come up with a new transformer approach, but new insights on BERT. The authors claim that BERT, without any major changes, can match or exceed every published model after it by just using the right hyperparameters and enough training data.

The authors find that BERT was significantly undertrained and propose the following modifications for maximal performance:

- **More training data.** The original BERT was trained on the BookCorpus and English Wikipedia, covering around 16 GB of text. Many of the subsequent papers used much larger data sets, some of them publicly available, others

not. RoBERTa uses five English-language corpora with a total size of over 160 GB of text.

- **Longer training.** RoBERTa evaluates three training durations with 100K, 300K and 500K steps. They show that maximal training duration, together with the additional data, results in best performance.
- **Larger batch size.** While the original mini-batch size of BERT is 256, RoBERTa further evaluates batch sizes of 2,000 and 8,000 samples per mini-batch. The experiments on several downstream tasks indicate that a batch size of 2,000 is the best choice, given the learning rate is increased appropriately.
- **Get rid of the next sentence prediction (NSP) objective.** In contrast to [6], the authors of RoBERTa show that by removing the NSP loss, they achieve slightly better downstream task performance. They also show though that it is crucial to use the full attention span (the model input size, max. 512 tokens) during pre-training in order for BERT to learn long-range dependencies.
- **Change the masking pattern of training data dynamically.** While BERT uses a relatively static masking procedure applied during preprocessing, RoBERTa suggests dynamically masking of each sample during training before feeding it to the model.

With all these modifications, RoBERTa's performance is evaluated on the GLUE, SQuAD and RACE benchmarks. RoBERTa achieves state-of-the-art results on all three challenges, with slightly better results than XLNet (its closest competitor) on most challenges and clearly better than the original BERT [17].

4.4 DistilBERT - Smaller, Faster, Cheaper

While all three models discussed so far focused on improving the state-of-the-art, the authors of DistilBERT [23] pursue another objective. Instead of better results, DistilBERT aims for a smaller and faster models which can be used in real-world projects. Indeed, the size of recent transformer models is impressive:

- A BERT-Large model uses 340M parameters.
- RoBERTa works with 355M parameters while using 160 GB data for pre-training.
- NVIDIA trained a transformer language model called *MegatronLM*[26], using 8300M parameters.

The authors of DistilBERT see the challenge of modern transformer models mainly at inference time, when a model should be small and fast, so it can work on mobile devices or non-GPU servers.

There are several approaches to reduce model size while approximating performance, namely *quantization*, *weights pruning* and *distillation*. The authors of DistilBERT focus on knowledge distillation.

4.4.1 Knowledge Distillation. The core idea of knowledge distillation is to train a smaller model (the *student*) which learns from the original model (the *teacher*). The method has been generalized by [10] and is sometimes also referred to as *teacher-student learning*.

The challenge of knowledge distillation is to learn the so-called *dark knowledge* of a model. Let us assume the example in Figure 8, where a language model predicts the last token in a sentence. While one or two predictions usually have a high probability, there will almost always be a long tail of tokens with low probability. This knowledge, even though the probabilities might be low, is crucial for a model to generalize. The challenge of knowledge distillation is therefore not only to learn the one prediction with high probability, but also the near-zero probabilities on other classes. In terms of distillation loss function, this near-zero probabilities are called *soft targets*.

Input:

'I', 'think', 'this', 'is', 'the', 'beginning', 'of', 'a', 'beautiful', **[MASK]**

Predicted token (top 16):

#1 token: 'day'	**p:** 0.213	**#9** *token:* 'summer'	**p:** 0.016
#2 *token:* 'life'	**p:** 0.183	**#10** *token:* 'adventure'	**p:** 0.013
#3 *token:* 'future'	**p:** 0.062	**#11** *token:* 'dream'	**p:** 0.012
#4 *token:* 'story'	**p:** 0.058	**#12** *token:* 'moment'	**p:** 0.012
#5 *token:* 'world'	**p:** 0.049	**#13** *token:* 'night'	**p:** 0.011
#6 *token:* 'era'	**p:** 0.045	**#14** *token:* 'beginning'	**p:** 0.010
#7 *token:* 'time'	**p:** 0.032	**#15** *token:* 'season'	**p:** 0.009
#8 *token:* 'year'	**p:** 0.017	**#16** *token:* 'journey'	**p:** 0.006

Figure 8: Predicting a masked token with BERT: The first two tokens show a high probability, followed by a long tail of near-zero possibilities.

4.4.2 Loss Objective. To motivate a model to learn both high and near-zero probabilities, DistillBERT suggests a composite loss function as follows:

- **Distillation Loss** The student learns the probabilities of the teacher (*soft targets*) with $L = \sum_i t_i * \log(s_i)$ where t_i is a probability estimated by the teacher and s_i the probability estimated by the student. To control the smoothness of the output distribution, a further technique from [10] called *softmax-temperature* is used.
- **Masked Language Modeling (MLM) Loss [6]** As the teacher model used in DistilBERT is the original BERT model, the second loss function is the original MLM, with the goal to predict the masked tokens.
- **Cosine Embedding Loss** A rather technical loss to align the directions of the student and teacher hidden states vectors.

4.4.3 Student Model Architecture. While using the original BERT architecture as teacher model, the student model architecture is a purged version of the BERT model. *Token-type embeddings* and the *pooler* have been removed and the number of layers have been reduced by factor 2. This reduces the overall size of the model by 40% [23].

Note that the distillation process happens on the general-purpose model, before applying fine-tuning on downstream tasks. This in contrast to task-specific distillation, which distills the model after already being fine-tuned on a specific task.

4.4.4 Performance. To compare DistilBERTs performance, it has been fine-tuned on two of the usual benchmarks (GLUE, SQuAD). DistilBERT retains 97% of the original BERT model while reducing the model size by 40% and being 60% faster [23].

5 EVALUATION

In this section, we present and discuss the results of the experiments conducted with the transformer architectures described in Section 4. In particular, we will address the following research questions with respect to entity matching:

- How well do transformer architectures perform compared to traditional, non-deep-learning ML approaches?
- How well do transformer architectures perform compared to recent deep learning approaches?
- Which transformer architecture performs best on the task of entity matching?
- Transformer architectures are complex deep neural networks. How much training is necessary to fine-tune them on the entity matching task? Is the amount of training data in a traditional EM dataset sufficient to fine-tune a transformer?

5.1 Datasets

The authors of *DeepMatching*[20] demonstrated that modern deep learning methods do not perform better in entity matching tasks on structured data than traditional approaches. Traditional ML approaches such as e.g. *Magellan*[14] perform even slightly better and training time is magnitudes shorter than for deep learning methods. In addition, the scores of *Magellan* on structured data are high - it performs on those 11 datasets with an average F1-score of 86% - without the rather textual dataset *Amazon-Google* the F1-score is even 90%. The authors of DeepER[7], a second approach using deep learning for entity matching, confirm these results with F1-scores between 88% and 100% on these structured datasets.

We therefore focus our experiments on challenging datasets where other approaches showed relatively low performance. To evaluate our transformer approaches we use both *textual* and so-called *dirty* datasets provided by the Magellan team [20]. We use all publicly available datasets except the textual dataset *Company*, as this contains text blobs with lengths between 2000-3000 tokens, which exceeds the maximal attention spans of 512 tokens for transformer architectures. There have been recent approaches to extend this attention span to 8000+ tokens by [28]. However, we leave these challenges for future work.

For our entity matching experiments, we use the following five datasets to evaluate the transformer architectures (see Table 3): *(1) Abt-Buy*: Product data from Abt.com and Buy.com. The core attribute is *description*, which is a long text blob describing the product. We use no informative attribute (e.g. the *title*), but only the noisy *description* attribute, similar to [20]. *(2) iTunes-Amazon (Dirty):* music data from iTunes and Amazon. The data has been modified to simulate dirty data as done by [20]. They suggest for each attribute other than "title" to randomly move each value to the attribute "title" in the same tuple with a probability of $p = 0.5$. *(3) DBLP-ACM (Dirty):* Bibliographic data from DBLP and ACM. The data has been modified to simulate dirty data. *(4) DBLP-Scholar (Dirty):* Bibliographic data from DBLP and Google Scholar. The data has been modified to simulate dirty data. *(5) Walmart-Amazon (Dirty):* Product data from Walmart and Amazon. The data has been modified to simulate dirty data.

To evaluate our transformer architectures, we split all five datasets into into three parts with a ratio of 3:1:1. We use the 60% split of the data for training, and the two 20% splits for validation and test. All reported numbers in this paper show results on the test split.

Table 3: Datasets used in our experiments.

Dataset	Domain	Size	# Matches	# Attr.
Abt-Buy	Products	9,575	1,028	3
iTunes-Amazon	Music	539	132	8
Walmart-Amazon	Products	10,242	962	5
DBLP-ACM	Citation	12,363	2,220	4
DBLP-Scholar	Citation	28,707	5,347	4

5.2 Setup & Methods

In this section we will describe the hardware we used for our experiments as well as the implementation of the transformers along with the hyperparameters and pre-trained models.

5.2.1 Hardware. All experiments were executed on a single Nvidia TITAN Xp GPU (12GB Memory) with Intel(R) Xeon(R) CPU E5-2650 v4 (4 cores) and 8GB memory. The experiments are implemented using PyTorch and the transformer implementations are based on [32].

5.2.2 Methods. Figure 9 shows how an entity pair consisting of `Entity A` and `Entity B` is fed into a transformer architecture. This approach is used at both training and inference time. The data feeding approach looks similar in all four transformer architectures, while having minor differences in the use of separator tokens, position embeddings and the classification representation (`CLS`) in the last layer. The detailed description of the following section refers to BERT/RoBERTa and DistilBERT, while being very similar in XLNet.

We will now describe how we use the transformer for entity matching in detail. `Entity A` and `Entity B` contain a single text blob each. In case of textual data (*Abt-Buy*), the text blob consists of the single attribute *description*. For "dirty" datasets, all attributes of a data instance are concatenated, for instance `[name + brand + description + price]`. Next, the single text blob is tokenized. For each token, the corresponding embedding is looked up (for details on tokenizing and embedding see Section 5.2.3).

Each token embedding is then summed up with a positional embedding and a segment embedding. The segment embedding is used to distinguish between tokens of `Entity A` and `Entity B`. The sum of all three embeddings is then fed into the transformer. The maximum length of the input ($A_1 - A_N$ plus $B_1 - B_M$ plus the two artificial tokens *classification* and *separator*) has been empirically defined based on the longest data rows in the training data. It lies between 128 and 265 tokens, depending on the data set.

In one pass, a pair of two entities (all tokens of `Entity A` and `Entity B`) is processed by the transformer model. This leads to major performance improvements compared to classical deep learning models based on RNNs[20], where each token is depending on the tokens appearing earlier in the sequence. The output of a pass is then represented in the hidden state at position 0, the purple `CLS` symbol in Figure 9. This hidden state (a vector of size 768) is then passed into a single classification layer (*Classifier* in Figure 9).

Keep in mind that transformer models are universal language models, capable of performing different downstream tasks (e.g. classification, seq2seq, question/answering, NER, etc.). Hence, to feed the classification output of the transformer model via a

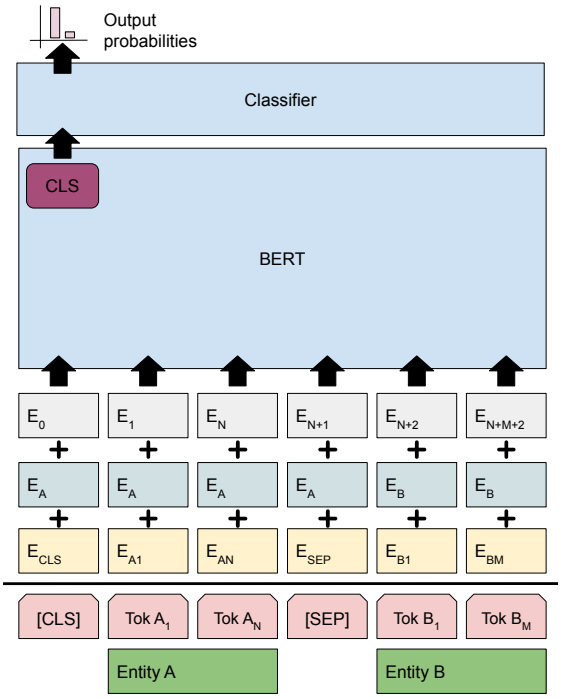

Figure 9: Transformer feeding approach. Entity A and Entity B are tokenized (e.g. `Tok A1,red`). The embedding of each token (E_{A1},`yellow`) is fed into the transformer together with positional- and segment embeddings (E_0,`gray`/E_A,`blue`). Special symbols are used to feed the output into the classification layer (`[CLS]`) and to separate the two entities (`[SEP]`)

designated representation symbol (`CLS`) into a specific classification layer, is an architectural decision of [6] to keep the model flexible. When using the model for other tasks, e.g. for NER, the ouput layer looks differently.

The classification layer is - in contrast to the rest of the model - not pre-trained and contains a fully connected layer with 768 neurons plus two output neurons. These two output neurons finally represent the two classes of an entity matching problem: *"Entity A and B match"* or *"Entity A and B do not match"*.

We use Adam [12] for optimization in combination with a linear learning rate. The choice of optimizer, learning rate and other hyperparameters are based on best practices for similar classification tasks, e.g. the GLUE-QQP benchmark [6, 17, 23, 31, 33].

With the implementation described above, we evaluate all four transformer architectures described in Section 4: BERT, XLNet, RoBERTa and DistilBERT. To reproduce the experiments, we provide source code and hyperparameters on GitHub[1].

5.2.3 Tokenization and Embeddings. Before feeding a pair of `Entity A` and `Entity B` into the transformer, tokenizing the input and a lookup of embeddings is necessary. Transformers like almost all modern language models work with embeddings as input values. Using embeddings increases performance over simple one-hot encoded vectors massively, as demonstrated by the influential *word2vec* paper [19].

We use the following tokenization/embedding techniques:

[1]https://github.com/brunnurs/entity-matching-transformer.

Table 4: Pre-trained models used in our experiments.

Transformer	Details
BERT	12-layer, 768-hidden, 12-heads, 110M parameters. BERT-base model. Trained on lower-cased English text.
XLNet	12-layer, 768-hidden, 12-heads, 110M parameters. XLNet English model
RoBERTa	12-layer, 768-hidden, 12-heads, 125M parameters. RoBERTa is using the BERT-base architecture.
DistilBERT	6-layer, 768-hidden, 12-heads, 66M parameters. The DistilBERT model is distilled from the BERT-base model.

- **BERT/DistilBERT:** We first split the whole input textblob into single tokens by simple white space- and punctuation splitting rules. In a second step, we create Wordpiece embeddings by applying the original algorithm from[24].
- **RoBERTa:** We split the whole input into tokens by using white spaces, punctuation and special abbreviations (`'s|'t|'re|'ve|'m|'ll|'d`). We then apply Byte-level Byte-Pair-Encoding [25].
- **XLNet:** In contrast to the other approaches, we do not pre-tokenize the input into word sequences, but directly input the raw text blob into a SentecePiece[15] subword tokenizer.

5.2.4 Pre-Trained Models. Due to their model sizes, all four transformer architectures require very resource-intensive pre-training on large amounts of data. As an example, the BERT_{LARGE} model was trained on 64 TPU chips for 4 days while RoBERTa uses 1024 V100 GPUs for approximately one day [6, 17]. The transformers performance is therefore heavily dependent on the pre-trained model. In Table 4 we list all pre-trained models used in the experiments. We use the pre-trained models provided by the original papers [32] and gathered by Hugging Face[2]. Note that due to hardware constrains, we always used the smallest available pre-trained model. We expect larger pre-trained models to perform as good or even better.

5.3 Comparison with Classical Models

Table 5: F1-scores of the best performing Transformer model in comparison with Magellan (MG) and DeepMatcher (DeepM).

Dataset	MG	DeepM	T_{BEST}	ΔF_1
Abt-Buy	33.0	55.0	90.9	35.9
$\text{iTunes-Amazon}_{Dirty}$	46.8	79.4	94.2	14.8
$\text{Walmart-Amazon}_{Dirty}$	37.4	53.8	85.5	31.7
DBLP-ACM_{Dirty}	91.9	98.1	98.9	0.8
$\text{DBLP-Scholar}_{Dirty}$	82.5	93.8	95.6	1.8

Table 5 shows the performance of transformer models in comparison with Magellan and DeepMatcher. The results are measured, similar to other papers in the field [7, 20], by an F1 score where recall is the ratio of true matches predicted vs. all true

[2]https://github.com/huggingface/transformers

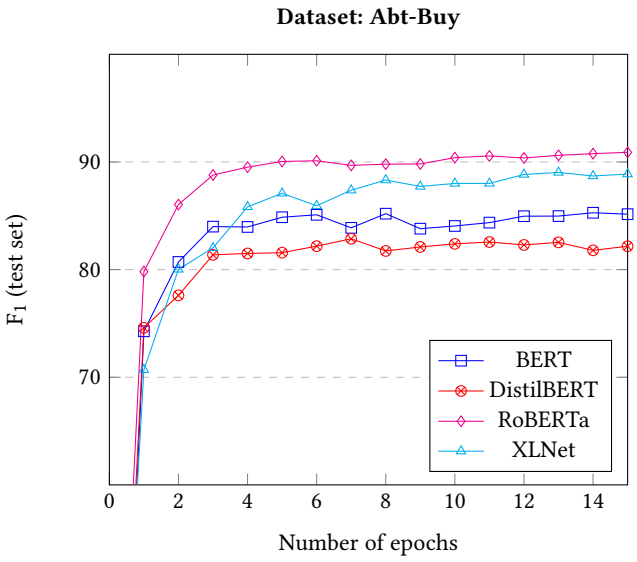

Figure 10: Performance of transformers on the Abt-Buy dataset, averaged over five runs. We only visualize F_1 scores in range 60 - 100% to emphasize the difference.

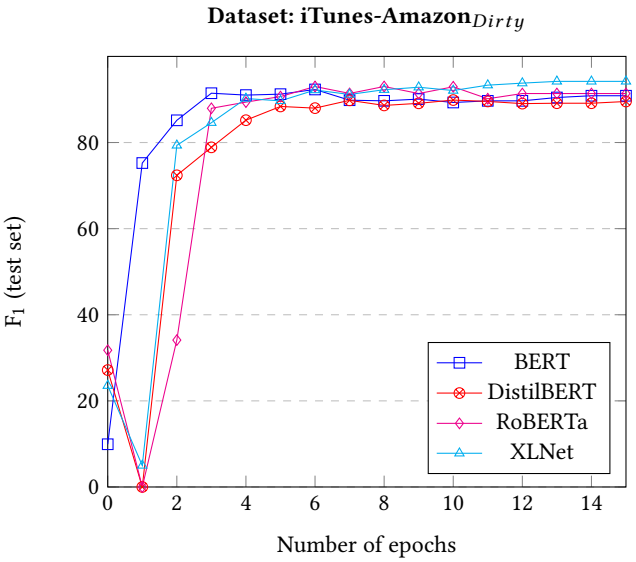

Figure 11: Performance of transformers on the iTunes-Amazon$_{Dirty}$ dataset, averaged over five runs. The effect of little training data is visible at epoch 1.

matches. For each dataset we report the best performing of the four DeepMatcher DL models, the result of Magellan and the best performing transformer. The transformer result is an average over five runs. **We see that transformer architectures outperforms both DeepMatcher and Magellan by a large margin on the challenging datasets *Abt-Buy* (35.9%), *iTunes-Amazon* (14.8%) and *Walmart-Amazon* (31.7%).**

We are also interested in how transformer architectures perform on datasets where DeepMatcher and Magellan perform very well, like on *DBLP-ACM* (DeepMatcher: 98.1%, Magellan: 91.9%) and *DBLP-Scholar* (DeepMatcher: 93.8%, Magellan: 82.5%). As expected, the transformers perform better than existing methods also on these datasets with a ΔF_1 of 0.8% and 1.8%, respectively. In comparison to the challenging datasets though, the improvements are relatively small. Since the results of DeepMatcher on these datasets are already very high (F1-score: 98.1% and 93.8%) we assume that better language models only slightly improve the performance on those tasks.

5.4 Transformers Architectures Head-to-Head

The performance of transformer architectures has been compared many times in recent papers [6, 17, 33], usually on large NLP benchmarks (e.g. GLUE, SQuAD [22, 31]). However, we are not aware of any related work where transformers were used for entity matching.

Figures 10 - 14 show the performance of the different transformer architectures on the datasets defined in Section 5.1. The results have been averaged over five runs. The main findings of our systematic, experimental evaluation are as follows:

- **All transformer architectures perform remarkably well on the task of entity matching.** Even the least performing transformer, *DistilBERT*, performs clearly better than classical approaches (see Section 5.3).
- After only **one epoch of training**, most experiments range within a 5% interval of their peak performance. After 3-5 epochs, almost all experiments converge to their peak performance. The effort to fine-tune a transformer on an entity matching task is therefore manageable, especially considering the training time per epoch in Section 5.5.
- Overall, *RoBERTa* performs best, also reaching high results already after only a few epochs. We conclude that the massive pre-training effort on a huge amount of data (compared to the other three approaches) gives RoBERTa a slight advantage in the task of entity matching.

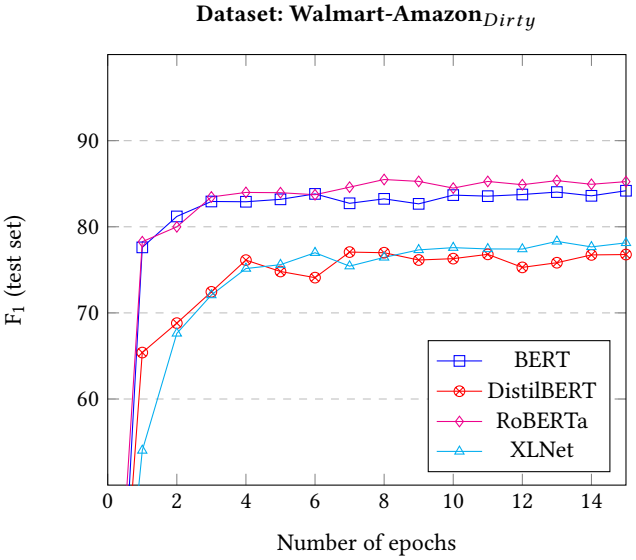

Figure 12: Performance of transformers on the Walmart-Amazon$_{Dirty}$ dataset, averaged over five runs. We only visualize F_1 scores in range 50 - 100% to emphasize the difference.

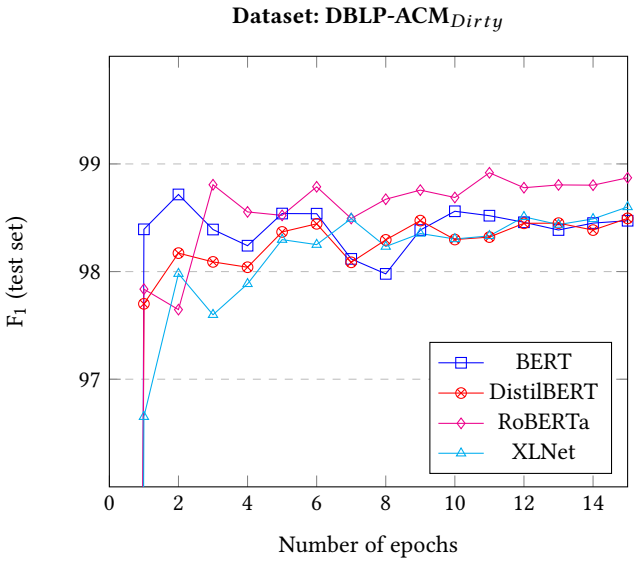

Figure 13: Performance of transformers on the DBLP-ACM$_{Dirty}$ dataset, averaged over five runs. We only visualize F_1 scores in range 96 - 100% to emphasize the difference.

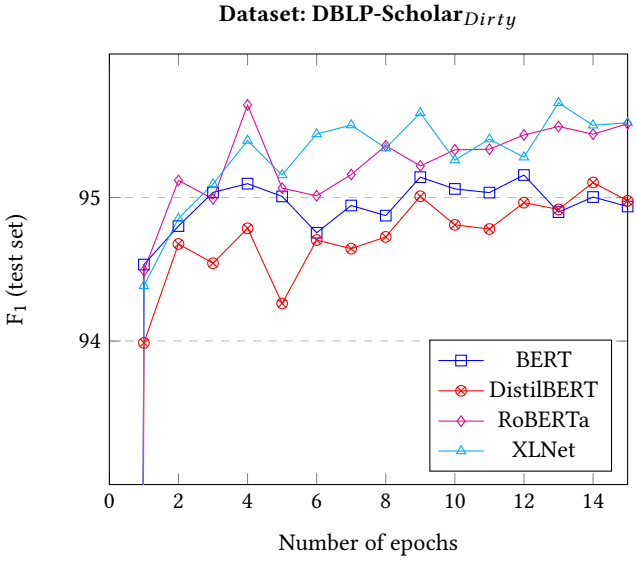

Figure 14: Performance of transformers on the DBLP-Scholar$_{Dirty}$ dataset, averaged over five runs. We only visualize F_1 scores in range 93 - 96% to emphasize the difference.

- *DistilBERT* performs worst, averaged on all tasks. This is explainable due to the comparably small model of DistilBERT. Still though the results of DistilBERT are close to its "big brothers" and higher than classical approaches (see Section 5.3).
- *XLNET* requires, on average, longer training times to reach peak results. We conclude the *permutation language modeling* eventually results in high performance (XLNet performs often almost equally well as RoBERTa), but due to the large amount of permutations, it requires more training time.

Table 6: Training time per epoch on each data set.

Dataset	BERT	XLNet	RoB.a	D.BERT
Abt-Buy	2m 42s	6m 15s	2m 43s	1m 22s
iTunes-Amazon$_{Dirty}$	7s	12s	7s	3.5s
Walmart-Amazon$_{Dirty}$	1m 41s	2m 29s	1m 41s	52s
DBLP-ACM$_{Dirty}$	2m 24s	4m 9s	2m 24s	1m 13s
DBLP-Scholar$_{Dirty}$	4m 5s	5m 57s	4m 13s	2m 6s

- The iTunes-Amazon$_{Dirty}$ dataset is extremely small containing only 132 matching records . As we see in Figure 11, this is influencing the training process. After epoch 1 almost all experiments have an F1-score of 0%, in comparison to the other datasets where after epoch 1 the results are already close to peak performance. We see, though, that even with this small amount of training data, the fine-tuning converges after 4-6 epochs of training.

5.5 Training Time

Transformer models are known to be resource-intensive architectures with millions of parameters. For instance, DistilBERT$_{SMALL}$ consists of 66 million parameters and RoBERTa$_{LARGE}$ has even 355 million parameters. As we have seen in Section 4, training a transformer is split in two tasks: *(1)* unsupervised *pre-training* on large amounts of unspecific data and *(2)* supervised *fine-tuning* on downstream tasks (e.g. entity matching) with task-specific data. While pre-training is doing a major part of finding these parameters (see Section 5.2.4), it is still a large model to fine-tune on a specific downstream task like entity matching.

Table 6 shows the training time for fine-tuning each transformer on the given datasets. The times reported are per epoch on the training set (which is roughly 60% of the total dataset).

As fine-tuning usually converges to maximum performance after 1-3 epochs (see Section 5.4), total training takes between **10s** (DistilBERT on iTunes-Amazon$_{Dirty}$, 3 epochs) and **11m 55s** (XLNet on DBLP-Scholar$_{Dirty}$, 2 epochs). It is important to note that the experiments ran on low budget hardware (see Section 5.2.1) without any parallelization. With state-of-the-art-hardware, these times can be reduced by an order of magnitude.

If we compare these training times to the reported results of DeepMatcher [20], we can draw two important conclusions:

(1) Compared to conventional, non-deep learning EM models as e.g. *Magellan*, fine-tuning a transformer is still relatively slow.
(2) Compared to training a deep learning model as proposed in DeepMatcher, fine-tuning a transformer is fast. DeepMatcher reports training times from **10min** to **11h**, depending on the dataset and solution. This is 1-2 magnitudes slower than fine-tuning a transformer as we propose, especially as DeepMatcher uses faster hardware than we do.

The reason fine-tuning a transformer is faster than training a comparably light-weighted deep learning model[20] is clearly the resource-intensive pre-training, which improves convergence on the downstream task (entity matching) enormously.

6 CONCLUSIONS

In this work, we show for the first time the strong impact of transformer architectures and pre-training for entity matching on datasets with large textual- or "dirty" data. We show that all transformers described can be used for EM out of the box, without the need for a task-specific architecture. Our experiments on five datasets show a significant improvement of F1-scores of up to 35.9% of transformers on challenging datasets compared to state-of-the-art approaches. In addition, we demonstrate that on relatively small and clean datasets, transformers still perform slightly better than earlier deep learning approaches.

We demonstrate that fine tuning a transformer on an EM task takes relatively little time and requires no particularly capable hardware, which might be contrary to expectations due to the large size of transformer models. Regarding the question on which of the four transformer models performs best on the EM task, experiments show that all of them deliver relatively similar results. In comparison with average results, RoBERTa shows slightly better and DistilBERT slightly worse performance, which is in line with the theoretical background of these approaches.

We consider transformer approaches as a valuable technology for entity matching. Using standard transformer architectures instead of designing EM-specific architecture does not only benefit from simplicity, but profits also directly from further advances in the NLP field of pre-training and transformers.

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. http://arxiv.org/abs/1409.0473 cite arxiv:1409.0473Comment: Accepted at ICLR 2015 as oral presentation.

[2] U. Brunner and K. Stockinger. 2019. Entity Matching on Unstructured Data: An Active Learning Approach. In *2019 6th Swiss Conference on Data Science (SDS)*. 97–102. https://doi.org/10.1109/SDS.2019.00006

[3] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *CoRR* abs/1406.1078 (2014). arXiv:1406.1078 http://arxiv.org/abs/1406.1078

[4] Peter Christen. 2012. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.* Springer Publishing Company, Incorporated.

[5] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. *CoRR* abs/1901.02860 (2019). arXiv:1901.02860 http://arxiv.org/abs/1901.02860

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018). arXiv:1810.04805 http://arxiv.org/abs/1810.04805

[7] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq R. Joty, Mourad Ouzzani, and Nan Tang. 2017. DeepER - Deep Entity Resolution. *CoRR* abs/1710.00597 (2017). arXiv:1710.00597 http://arxiv.org/abs/1710.00597

[8] Lise Getoor and Ashwin Machanavajjhala. 2012. Entity Resolution: Theory, Practice & Open Challenges. *Proc. VLDB Endow.* 5, 12 (Aug. 2012), 2018–2019. https://doi.org/10.14778/2367502.2367564

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). arXiv:1512.03385 http://arxiv.org/abs/1512.03385

[10] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. *ArXiv* abs/1503.02531 (2015).

[11] Matthew A. Jaro. 1989. Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida. *J. Amer. Statist. Assoc.* 84, 406 (1989), 414–420. https://doi.org/10.1080/01621459.1989.10478785

[12] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. http://arxiv.org/abs/1412.6980 cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[13] Hanna Koepcke, Andreas Thor, and Erhard Rahm. 2010. Evaluation of Entity Resolution Approaches on Real-world Match Problems. *Proc. VLDB Endow.* 3, 1-2 (Sept. 2010), 484–493. https://doi.org/10.14778/1920841.1920904

[14] Pradap Konda, Jeff Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, Vijay Raghavendra, Sanjib Das, Paul C., AnHai Doan, Adel Ardalan, Jeffrey Ballard, Han Li, Fatemah Panahi, and Haojun Zhang. 2016. Magellan: Toward building entity matching management systems. *Proceedings of the VLDB Endowment* 9 (08 2016), 1197–1208. https://doi.org/10.14778/2994509.2994535

[15] Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. *CoRR* abs/1808.06226 (2018). arXiv:1808.06226 http://arxiv.org/abs/1808.06226

[16] Guillaume Lample and Alexis Conneau. 2019. Cross-lingual Language Model Pretraining. *CoRR* abs/1901.07291 (2019). arXiv:1901.07291 http://arxiv.org/abs/1901.07291

[17] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR* abs/1907.11692 (2019). arXiv:1907.11692 http://arxiv.org/abs/1907.11692

[18] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. *CoRR* abs/1508.04025 (2015). arXiv:1508.04025 http://arxiv.org/abs/1508.04025

[19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 3111–3119.

[20] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep Learning for Entity Matching: A Design Space Exploration. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)*. ACM, New York, NY, USA, 19–34. https://doi.org/10.1145/3183713.3196926

[21] Salimans Radford, Narasimhan and Sutskever. [n.d.]. Improving Language Understanding by Generative Pre-Training. *Technical report, OpenAI* ([n. d.]). https://www.cs.ubc.ca/~amuham01/LING530/papers/radford2018improving.pdf

[22] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100, 000+ Questions for Machine Comprehension of Text. *CoRR* abs/1606.05250 (2016). arXiv:1606.05250 http://arxiv.org/abs/1606.05250

[23] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. [n.d.]. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. ([n. d.]). https://arxiv.org/abs/1910.01108

[24] M. Schuster and K. Nakajima. 2012. Japanese and Korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 5149–5152. https://doi.org/10.1109/ICASSP.2012.6289079

[25] Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, 1715–1725. https://doi.org/10.18653/v1/P16-1162

[26] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. [n.d.]. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. ([n. d.]). https://arxiv.org/abs/1909.08053

[27] Michael Stonebraker, Daniel Bruckner, Ihab F. Ilyas, George Beskales, Mitch Cherniack, Stanley B. Zdonik, Alexander Pagan, and Shan Xu. 2013. Data Curation at Scale: The Data Tamer System. In *CIDR*.

[28] Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. 2019. Adaptive Attention Span in Transformers. *CoRR* abs/1905.07799 (2019). arXiv:1905.07799 http://arxiv.org/abs/1905.07799

[29] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 3104–3112. http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 5998–6008. http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

[31] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. *CoRR* abs/1804.07461 (2018). arXiv:1804.07461 http://arxiv.org/abs/1804.07461

[32] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *ArXiv* abs/1910.03771 (2019).

[33] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. *CoRR* abs/1906.08237 (2019). arXiv:1906.08237 http://arxiv.org/abs/1906.08237

[34] Chen Zhao and Yeye He. 2019. Auto-EM: End-to-end Fuzzy Entity-Matching Using Pre-trained Deep Models and Transfer Learning. In *The World Wide Web Conference (WWW '19)*. ACM, New York, NY, USA, 2413–2424. https://doi.org/10.1145/3308558.3313578