POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Zero-shot product retrieval with contrastive learning

Author: Lorenzo CRAVERO

Advisor: Giuseppe RIZZO
Co-Advisor: Lorenzo BONGIOVANNI

April, 2023

Abstract

While developing a platform that compares product prices and characteristics across various Italian e-commerce websites, the need to retrieve similar products across the different and ever changing websites arose. In this thesis, we approach the challenge by setting up a task of retrieval of similar products given a starting product, where each product is defined solely by its textual attributes. Particular focus is put on the zero-shot scenario, where the model performance is evaluated on both products and websites that have not been seen during training. Our goal is to produce a model that can generalize well and be easily implemented in a real-world use case.

This thesis leverages two language models that were pre-trained on extensive general corpora (BERT and MPNet), and demonstrates the benefits of applying a supervised contrastive learning objective during the fine-tuning stage for the purpose of retrieving new and unseen products in a zero-shot fashion. In addition, a more in-depth analysis reveals that this approach has the ability to enrich product embeddings by improving both their alignment and distribution uniformity. Finally, these product embeddings can be retrieved quickly using KNN, ANN or more optimized embedding search algorithms such as FAISS.

Overall, this thesis aims to present a valid approach to the challenge of product retrieval in a zero-shot context. It also provides valuable insights that may be relevant to real-world data integration scenarios.

Acknowledgements

A heartfelt thank you to Professor Giuseppe Rizzo for his punctual and precise advice, which has conveyed value with commendable efficiency. I also express my sincere gratitude to Lorenzo Bongiovanni for the valuable time he devoted to guiding me, for making me feel immediately at ease and, ultimately, for the numerous coffees he kindly offered, which played an indispensable role in maintaining my productivity. You both have my profound gratitude.

I want to thank the entire team at Trust In Food. Thanks to my experience with you, I realized that there is a whole world around me that I was missing.

Thanks to the team at Clevi because, literally, none of this would have been possible without you.

A special thanks to Gabbo, Jaco, and Gabri. The Via Maltesi. To the diamond we never reached.

A huge thank you to the Caimans of Borgo San Dalmazzo. Thanks to Andrea, Gabri, Jaco, Luca, Mario, Nico, Ogge, Pette, Recchi, Sam, Samma, Simi, Simo and the rare female specimen of Caiman, Vicky. Because you are good people and I am happy that a significant and indelible part of me is due to you.

My deepest thanks goes to my brother and my parents. For that memory of a distant summer and for the strength you give me in the present.

Lastly I want to dedicate this milestone to Riea, glimpse of beauty.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The work in this thesis was motiveted by our effort to develop a recommendation-like app for supermarkets products. Data integration is the process of combining data from multiple sources into a unified, coherent view. This process has become increasingly relevant in recent times as most supermarkets have transitioned from the traditional "brick and mortar" business model to a more sophisticated "click and mortar" model, which incorporates both online and offline operations. This shift is primarily driven by the fact that over 50% of shoppers now use the Web to research, compare and purchase products [1]. For businesses, data integration enables consolidation and standardization of data from multiple sources, providing a comprehensive view of the enterprise that can inform strategic decision-making, enhance operational efficiency, and reveal new growth opportunities. On the other hand, consumers benefit from data integration by gaining access to more extensive and accurate information about products, services, and market trends, which can enable them to make better-informed purchasing decisions. Often, consumers are faced with the challenge of comparing product prices and features across multiple websites. To address this issue, together with a team of colleagues, we came up with the idea of developing a platform to compare product prices between different italian supermarket e-commerce. To expand the concept a little further, we allow the user to compare the same cart between different online supermarkets obtaining three different outputs: the most similar cart to the one they composed, the cart that maximizes the saving and, lastly, the cart that is shipped more rapidly. In practical terms, the user enters the delivery address, assembles the cart on their preferred supermarket's page, selects which products cannot be replaced, proceeds to checkout, and then lands on a page where they can see the following:

- Prices proposed by other supermarkets for the same cart.

- The minimum cost obtainable for that same cart by replacing all present products with similar, less expensive products (such as white-label products).
- The cart with the same products but faster home delivery.

A mockup of the platform is presented in Figure 1.1. We believe that this solution could help users save time and money.

The research thesis is inspired and motivated by the described platform but independent from it.

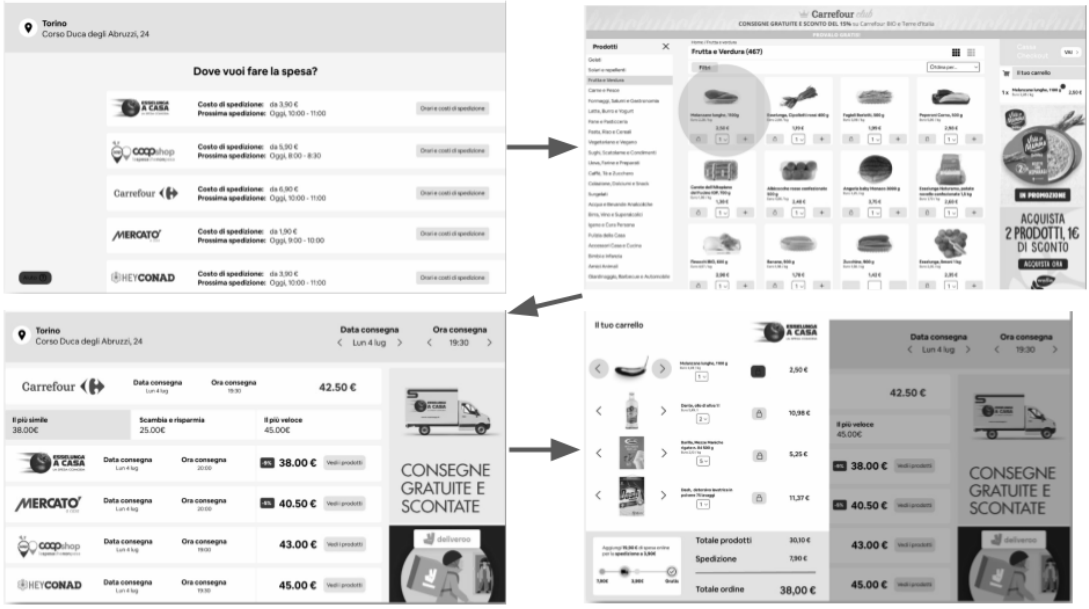


Figure 1.1: The flow of the platform. All markets available for the user address are plotted (1^{th}*screenshot*). The user chooses the one he prefers and it is presented with all the products of the market, from which he can compile its cart (2^{th}*screenshot*). When it is satisfied with its cart it proceeds to the comparison page, where three kinds of alternative carts are presented for each available supermarket (3^{th}*screenshot*). Before proceeding to the checkout with one of the proposed carts he can still refine them by replacing some of the products with suggested alternatives (4^{th}*screenshot*).

1.2 Objectives of the thesis

In order to establish the functionality of the platform outlined in the previous section, a crucial issue must be addressed: how to retrieve similar products across

the diverse and constantly evolving range of websites. This is the core challenge this thesis aims to tackle.

The solution must not only be accurate, but also capable of scaling to accommodate an extensive inventory of product entries and potentially hundreds or thousands of concurrent users requests. Given that each product entity is accompanied by textual attributes such as its description, brand, categories, sales denomination, and so forth, a particularly promising approach is to leverage a pre-trained NLP model to encode these attributes, thereby generating a corresponding embedding for each product. If the reader is not familiar with the concept of "leveraging a pre-trained NLP model", refer to Appendix A. Subsequently, the similarity between two products is determined based on the proximity of their corresponding vectors in the vector space. Specifically, products represented by vectors that are close together in the vector space are considered similar, while products represented by vectors that are distant from each other are considered semantically different. An overview can be seen in Figure 1.2.

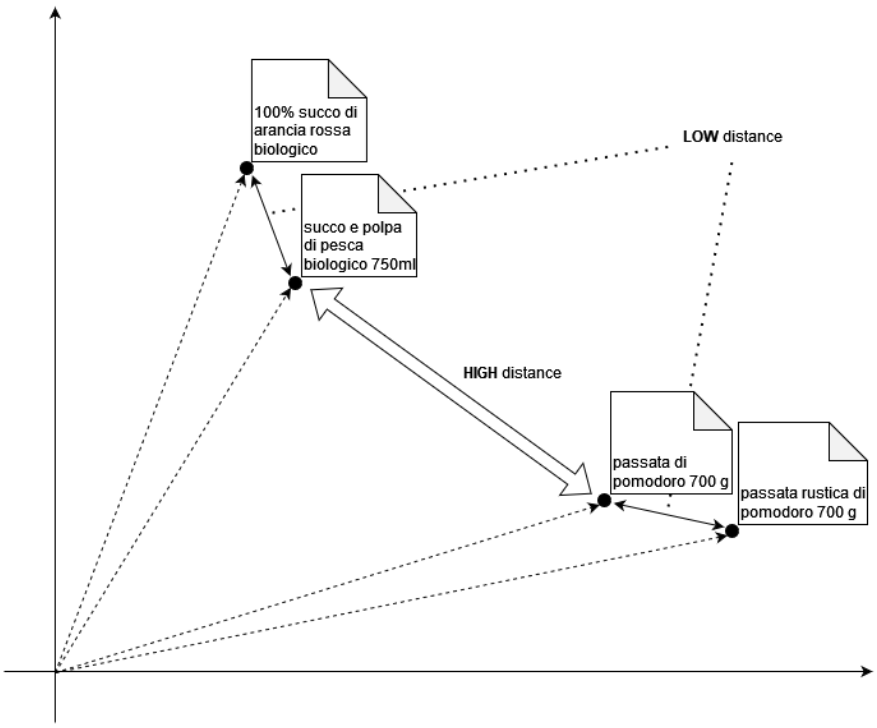


Figure 1.2: How the similarity among products is computed.

However, in order to be practically applicable, a further consideration must be taken into account. Labeling a dataset is a time-consuming and arduous process,

and the number of potential products is vast. Even if a substantial proportion of products were labeled, the introduction of new markets to the platform or the addition of new products to existing markets would inevitably result in the model confronting new and previously unseen product entities. To address this challenge, we have conducted all our experiments in a zero-shot fashion. Consequently, the models are evaluated using products different from those on which they were trained (zero-shot will be further explained in Section 2.4).

Our objective is to evaluate the performance of pre-trained NLP models in product retrieval and to assess their potential applicability to real-world scenarios. To achieve this goal, we first investigate the results obtained without any fine-tuning step. We then examine how a fine-tuning step can enhance the process and which technique is most effective. Among the techniques we explore, contrastive learning appears to be the most promising, particularly given its significant impact in generating semantically rich embeddings [2, 3]. We believe (and aim to confirm with this thesis) that such embeddings can be especially advantageous in a zero-shot setup (for further information on contrastive learning, refer to Section 2.3).

To address the research questions we have raised, we begin by defining the task. Unfortunately, in the literature, we have found little material that specifically pertains to the "e-commerce products" domain. Since the more general task of retrieving the most similar entities to a given input entity is referred to as entity retrieval, we have opted to designate the task of our interest as product retrieval. There are differences not only in the domain but also at the application level. Entity retrieval is typically employed for tasks such as document retrieval, entity disambiguation, or named entity recognition, all of which are usually referred to as entity linking tasks that are quite not our goal.

The limited coverage of the literature extends not only to the domain and the application but also to the relatively small number of experiments conducted in the zero-shot context. All of these considerations present challenges in terms of identifying valid datasets and finding benchmarks that can serve as a basis for comparison.

We have chosen to "adapt" the more extensively researched task of product matching (which boasts a wider array of benchmarks and datasets) to our area of interest. Product matching involves identifying which products refer to the same real-world product. In other words, two products match if they represent the same real product but from different sources. This represents the first contribution of this thesis: we derive the concept of a "relevant" product with respect to the one given as input from the notion of matching products and, in doing so, we propose the first publicly available zero-shot product retrieval benchmark. For a better understanding, refer to Figure 1.3. The specifics of how we transform publicly available datasets for the product matching task to treat them in a product retrieval context are discussed in Chapter 4.

As stated previously, to emulate a zero-shot scenario, the training dataset must be distinct from the one employed for evaluation. In this manner, the assessment is conducted on previously unseen products during training. Another noteworthy contribution of this thesis is the assessment of publicly available datasets for the task of product matching in a zero-shot manner. In addition, we replicate previous findings from the literature to facilitate comparison, thereby presenting a concise survey on the subject.

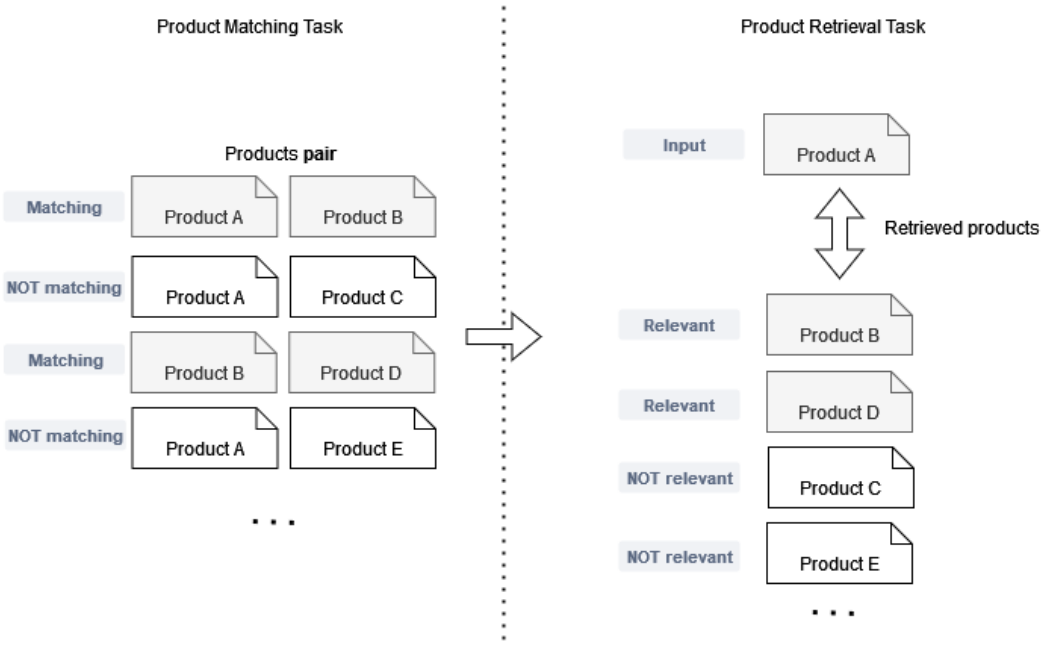


Figure 1.3: Product matching (on the left) adapted to the product retrieval (on the right) task. In a retrieval task we want to rank relevant items first. In our setup products entities that refer to the same real world product with respect to the input product are considered relevant items. It's important to note that, in this figure, product D is relevant to product A for transitive property.

1.3 Overview of the proposed approach

We aim to compare two pre-trained language models, namely BERT [4], trained as auto-encoder on the Masked Language Model task, and MPNet [5], trained with an autoregressive paradigm on the Masked and Permuted Language Model task. BERT has been pre-trained on BookCorpus, a dataset consisting of 11 038 upublished books and English Wikipedia (excluding lists, tables and headers), while MPNet is

pre-trained on the same amount of data of BERT and has been further fine-tuned on over one billion sentence pairs from a semantic textual similarity dataset. For contextual information regarding these models, please refer to Appendix A. This is done in order to evaluate how a model that adopts an autoregressive learning paradigm and it is already fine-tuned on STS task behaves compared to starting from model that is trained as autoencoder on general Masked Language model.

Subsequently, our focus shifts towards the fine-tuning stage. We begin by evaluating the pre-trained models without fine-tuning, which serves as our baseline for assessing the outcomes obtained after zero-shot fine-tuning. We explore solutions proposed in literature for the product matching task, and apply the same fine-tuning processes for the product retrieval task. Additionally, we conduct experiments using contrastive learning as our fine-tuning setup, specifically supervised contrastive learning [6]. We present the results for both zero-shot product retrieval with contrastive learning and zero-shot product matching with contrastive learning in Chapter 5.

Chapter 2

Background

2.1 Product retrieval

As outlined in the introduction, the exploration of product retrieval in the literature is limited. A more covered and similar concept is product search, however, a fundamental difference is that our investigation is concerned with the semantic similarity between products, independent of user shopping needs, while product search involves two distinct stages: the initial retrieval phase, followed by a re-ranking step that identifies the optimal products to present to the customer. Generally the re-ranking phase is the one more covered, while we are most interested in the retrieval phase. In [7], the authors' focus is on the search queries made by users, with their model trained on the Walmart search query log. They incorporate various product features, such as popularity, add-to-cart rate, click-through rate, and order rate, among others. Nonetheless, their approach is similar to ours, as they propose a neural retrieval model that utilizes neural transformation to acquire vector representations of products and queries separately. They then use a straightforward aggregation function to merge two vectors to predict the final score.

Another paper in which they focus on the retrieval phase is [8]. They highlight the significance of utilizing relevance functions based on semantic matches rather than relying on exact matches such as TF-IDF. The latter is frequently employed but is insufficient to achieve search result rankings that are suitable for user presentation. E-commerce sites often overlay a range of handcrafted filters (using structured data fields) and hard-coded rules for specific queries atop the legacy relevance function score.

Neither of the previously mentioned approaches on product retrieval utilized public datasets.

Although we were unable to locate other literature specific to the task of product

retrieval, we found literature that referred to more general tasks. The current literature on information retrieval primarily focuses on web search, resulting in the specific task being referred to as document retrieval. Other topics include entity disambiguation or named entity recognition. Entity retrieval is occasionally used interchangeably with entity linking. In [9], the authors demonstrate that performing entity linking is feasible by training a dual encoder (two-tower) model that encodes mentions and entities in the same dense vector space. Candidate entities are retrieved by conducting an approximate nearest neighbor search. The authors trained the model on Wikinews. In [10], the authors propose a different approach that is based on a model that retrieves entities by generating their unique names from left to right, token-by-token, in an autoregressive manner.

2.2 Product matching

In the domain of retail, advertisements and listings often provide incomplete information about products, such as the absence of a universal product code (UPC) [11] or global trade item number (GTIN) [12]. This inadequacy is explored in greater detail in [13], which identifies six challenges associated with the efficient organization and management of e-commerce metadata. Among these challenges is the insufficient use of unique product identifiers, which are vital for enabling efficient product data integration. These codes and numbers aid in identifying products, making them more easily discoverable for consumers, and enhancing the consumer experience for retailers by providing relevant advertising, complete product listings, and better product recommendations. In conclusion, clear product identification from diverse sources benefits consumers, advertisers, and product aggregators.

Identifying products can be challenging due to incomplete, dirty, and unstructured data. Such issues can hamper the extraction of information from product listings, rendering it difficult for consumers to locate what they are seeking. Retailers may also organize products into non-interoperable taxonomies, making it challenging to extract information from the meta-structure of the product data. Furthermore, there are concerns about scalability as the size of datasets expands, and managing the inconsistencies that can arise from retailers categorizing products differently.

Product matching is a rapidly developing field, with numerous new publications since 2018 [14, 15, 16, 17, 18, 19, 20, 21]. This trend can be attributed to several factors, including greater access to data and the development of strategies and tools to handle big data. DeepER [21] and DeepMatcher [20] were the first to demonstrate that deep learning models can effectively handle entity matching, which is like product matching but extended to other entity's domains. They

utilize recurrent neural networks [22], integrated with attention modules [23, 24], to encode pairs of entities in multi-dimensional vectors and create a binary classifier based on the similarity of these embeddings to generate a matching decision.

Although these architectures are tailored for the task and perform well, they are outperformed by more recent Transformer-based models, particularly in "dirty" datasets (where the entities from a pair are structured records with the same schema, but some attribute values may be "injected" under the wrong attribute or may be missing) or "textual" datasets (where all attributes correspond to raw text entries). The release of BERT [4] sparked a revolution in the NLU field, as pre-trained language models became increasingly important in numerous tasks (as discussed in Appendix A), including entity matching (and product matching accordingly). Researchers have demonstrated that with a simple fine-tuning step, the BERT architecture outperforms previous state-of-the-art models [14, 15].

In [15] they experimented with 3 optimization techniques:

- **Leveraging domain knowledge**: it consists in pre-processing the input sequences to allow domain knowledge to be injected into *Ditto*. Two main methods are proposed. The first one called **span typing** that helps the model recognize a certain type of span surrounding it with a specific token. For example, a phone number "(866)246-6453" may be replaced with "(866)246-[LAST]643[/LAST]". The second method is **span normalization** with which two different but equivalent spans are rewritten into the same string. For example both "5 %" and "5.00%" can be rewritten as "5.0%".

- **Summarizing long entries**: Usually in transformer-based pre-training there is a limit on the sequence lenght of the input. Often sequences that don't fit into the limit are simply truncated. Here they tried to summarize them using a *TF-IDF-based* summarization algorithm. This tecnique can be beneficial in textual datasets with long text descriptions for each entity. For the purposes of the thesis, where the focus is on *products* entities that usually are characterized by short descriptions, this technique can be overlooked.

- **Augmenting training data**: this technique has proved itself beneficial in the image vision field. In the language one it is a little bit more controversial. The issue is that an augmentation almost always preserve the image labels, while a sequence can easily be distorted so much that the label become incorrect. In the paper different operators are tried like *span_del*, *span_shuffle*, *attr_del* and *entry_swap*. Moreover, a sophisticated interpolating strategy is introduced in order to reduce the "distortion" effect of the augmentation. A last thing that must be said is that the *dropout* noise during training inherent to Transformer encoders can be regarded as soft data augmentation in the embedding space,

since two embeddings of the same product offer will likely never look exactly the same during training (as analyzed in [3]).

These techniques improve results on some specific dataset. However, particularly in the ones that will be treated in the thesis belonging to the *products* domain (see Chapter 3), there is not really an appreciable improvement.

The first paper mentioned [14] doesn't use serialization techniques. For textual datasets (composed of a single long description), it directly tokenize it as it is, while for entites "fragmented" in attributes it simply concatenate all of their values together, for instance *name + brand + description + price*. *Ditto* [15] uses a more sophisticated approach using special tokens to help the model recognize different attributes. For examples the key/value pairs:

- **title**: instant immers spanish dlux 2
- **modelno**: NULL
- **price**: 36.11

are concatenated together in the string "*[COL] title [VAL] instant immers spanish dlux 2 [COL] manf./modelno [VAL] NULL [COL] price [VAL] 36.11*". It is not clear if this last method brings a noticeable improvement, however, [25] suggests that data encoding does not have on average a real impact on the results.

An interesting approach to the task is given by [16]. Unfortunately it is tested on a proprietary dataset so it is difficult to reproduce. They interpret product matching as a similarity learning task allowing them to cast the problem as a zero-shot learning problem in order to obtain semantically rich embeddings. To do so they exploited the triplet loss [26], introducing a notion of similarity between pairs of products:

$$L(p, p^+, p^-) = max(0, m + d(\mathcal{E}(p), \mathcal{E}(p^+)) - d(\mathcal{E}(p), \mathcal{E}(p^-)))$$

Where p is a product, p^+ a matching product (positive), p^- a non-matching product (negative), $\mathcal{E}$ a transformer-based encoder (accordingly, $\mathcal{E}(p)$ is the embedding of a product p) and d is a distance measure (the cosine distance in this case). Lastly m is the margin and it's treated as hyper-parameter.

2.3 Contrastive learning

In our work, we have opted to adopt a Contrastive Learning approach. Specifically, we have decided to employ the loss function introduced in [6], called SupCon loss. As explained in the paper, there has been a recent resurgence of interest

in contrastive learning, which has resulted in significant advancements in self-supervised representation learning. The underlying concept of these works is to bring an anchor and a "positive" sample closer together in the embedding space while simultaneously pushing the anchor away from multiple "negative" samples. As labels are not available, a positive pair generally consists of data augmentations of the sample, while negative pairs are formed by the anchor and randomly selected samples from the minibatch. The authors propose a supervised learning loss that builds upon the contrastive self-supervised literature by utilizing label information. Normalized embeddings from the same class are brought closer together than those from different classes. In our use case, two products belong to the same class if they refer to the same real product entity. These positives are drawn from samples of the same class as the anchor, as opposed to being data augmentations of the anchor, as is done in self-supervised learning. This is particularly beneficial in NLP tasks, since unlike image vision where an augmentation nearly always preserves the image labels, a sequence can easily become distorted to the point where the label becomes incorrect. In [3], self-supervised contrastive learning is employed on sentence embeddings, and it is discovered that dropout acts as minimal data augmentation and is more effective on its own than other text augmentations like span deletion, span shuffle, attribute deletion, and entry swap.

Since we have the information of matching and not matching products, in this thesis we adopt the supervised contrastive loss, that is expressed as

$$L_{out}^{sup} = \sum_{i \in I} L_{out,i}^{sup} = \sum_{i \in I} \frac{-1}{|P(i)|} \sum_{p \in P(i)} log \frac{exp(z_i \cdot z_p \tau)}{\sum_{a \in A(i)} exp(z_i \cdot z_a \tau)} \tag{2.1}$$

Where we have a batch of N products, $i \in I \equiv \{1...N\}$ is the i-th product in the batch, referred to as the anchor and $A(i) \equiv I/\ \{i\}$ is the set of all products except i. $P(i) \equiv \{p \in A(i) : \widetilde{y}_p = \widetilde{y}_i\}$ is the set of indices of all positives in the batch distinct from i, and $|P(i)|$ is its cardinality. $\tau \in R^+$ is a scalar temperature parameter. z_p refers to positive product embeddings with respect to the anchor, while z_a refers to all embeddings except the embedding of i. It can be seen as a generalization of both the Triplet and N-pair losses [26, 27], since the former uses only one positive and one negative sample per anchor, while the latter uses one positive and many negatives. The SupCon loss in Equation (2.1) brings positives closer together, while negatives are pushed apart, as represented in Figure 2.1.

Extending the self-supervised batch contrastive approach to the fully-supervised setting allows us to effectively leverage label information. Clusters of points belonging to the same class are pulled together in embedding space.

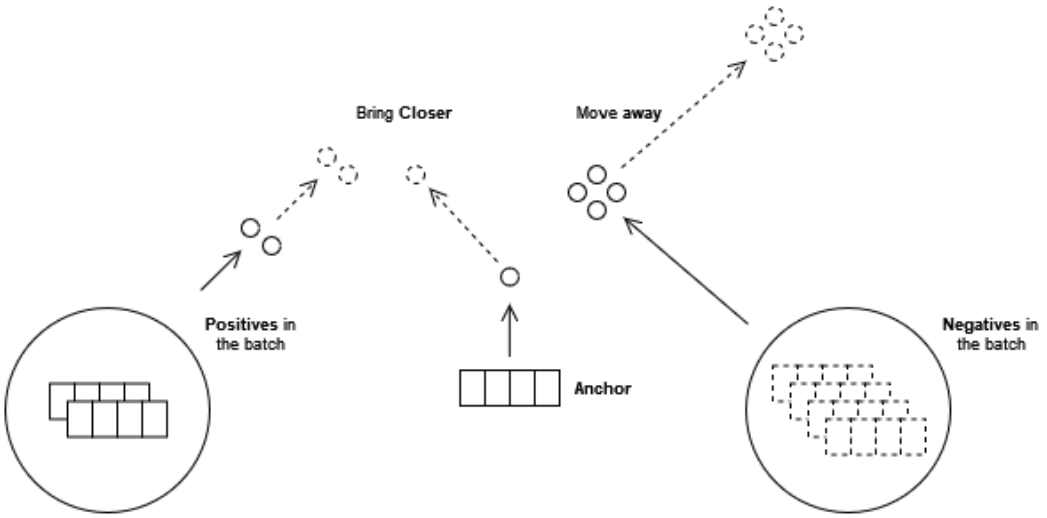


Figure 2.1: The supervised contrastive loss. Points belonging to the same class of the anchor product (i.e. matching products in our use case) are pulled together while negatives are pushed apart.

2.4 Zero-shot scenario

Zero-Shot Learning (ZSL), in its original definition, aims to identify unfamiliar categories by drawing upon their semantic correlations with known categories. These semantic connections may manifest within human-annotated attributes, lexical embeddings, textual portrayals, and so on. In practice, ZSL is performed by firstly learning an embedding space where semantic vectors and visual features are interacted. Then, within the learned embedding space, the best match among semantic vectors of unseen classes is selected for the visual features of any given sample of the unseen classes.

In the context of these thesis, we adapt the concept of Zero-Shot Learning to zero-shot product retrieval task, where each product is assigned to a cluster, and our objective is to recover products that belong to the same cluster.

Chapter 3

Data

We were unable to locate publicly accessible datasets that were explicitly tailored to the product retrieval task. Consequently, we rely on the datasets designed for product matching. In Chapter 4, we will elaborate on the processing techniques we employ to adapt these datasets to our focused task. These public-available datasets are valuable to researchers, as they lower the entry barrier for evaluating novel methodologies and linkage strategies. Moreover, the use of public data guarantees result reproducibility, particularly for projects with open-source code. The consistent use of identical public datasets across various researchers also facilitates comparisons between different approaches. In our study, the analysis of the product matching task, and the replication of the outcomes achieved in the state-of-the-art approaches, served as the foundation for our work. It is also the reason why we conducted an analysis of the product matching task. We first examined the zero-shot scenario in the product matching task and then proceeded to the zero-shot scenario in the product retrieval task.

The first structured benchmark for the product matching task was introduced by [20]. They provided three of the four datasets that we utilize, in addition to the metrics for the task, which we will deepen in Chapter 4. All subsequent literature on this task follows their benchmark. As the datasets were created for the product matching task, they comprise pairs of product entities, along with a label. This label designates a value of 1 for matching products and 0 for non-matching ones. Each product entity is defined by its textual attributes.

The data can be classified into three distinct categories:

- *Structured*: the attributes adhere to the same schema and possess clean text-based values of restricted length.
- *Textual*: all attributes correspond to raw text entries.
- *Dirty*: similar to *structured* data, except the values of the attributes may be

injected under the wrong attribute, and a varying portion of them may be null.

Four publicly available datasets have been used in our study.

- **Abt - Buy / Amazon - Google**: These datasets were initially published in the repository of the Database Group of the University of Leipzig [28].
- **Amazon - Walmart**: Made available by the University of Wisconsin-Madison's Magellan Data Repository [29].
- **WDC LSCP**: The WDC product data corpus [17] consists of 26 million product offers originating from 79 thousands websites. The offers are grouped into 16 million clusters of offers referring to the same product using identifiers, such as GTINs [12] or MPNs. The gold standard consists of 4400 pairs of offers that were manually verified as matches or non-matches. Each category has a fixed number of positive and negative pairs. The negative examples are created by selecting pairs that have high textual similarity but different IDs. From this huge corpus, different subsets of the WDC product data corpus are provided (small, medium, large, xLarge).

A summary of certain characteristics revealed by an exploratory data analysis is available in Table 3.1.

Dataset	Domain	type	Train Size	PosNeg ratio	distinct products	% with no positives
Abt - Buy	eCommerce Products	*textual*	5 743	8.3	1929	36.4%
Google - Amazon	software / electronics	*structured*	6 874	8.8	1590	54.6%
Walmart - Amazon	software / electronics	*dirty*	6 144	9.7	5126	78.2%
WDC small	eCommerce Products	*dirty*	9 038	2.9	8880	47.8%
WDC medium	eCommerce Products	*dirty*	25 567	3.6	12 349	30.6%
WDC large	eCommerce Products	*dirty*	103 411	4.5	13 719	7.1%
WDC xLarge	eCommerce Products	*dirty*	214 736	6.1	13 930	3.67%

Table 3.1: Data analysis summarization. The % of products with no positive (i.e. that never appear in a matching pair) will be relevant later on. An high percentage could cause some issues since the contrastive learning approach (as presented in Section 2.3) rely on bringing the positive products embeddings closer together and push away negative ones.

The positive negative ratio is a meaure of how much the dataset is imbalanced. WDC, for which products are extracted from numerous sources, tends to be less imbalanced.

In [16], it has been mentioned that an appropriate batching strategy is essential for selecting non-trivial pairs, failing which the loss would be zero and the model would learn nothing. However, with these public datasets, such a strategy is

not required, as the negatives are already selected from pairs with high textual similarity (hard-negatives).

The attributes that are present in the Abt-Buy, Google-Amazon, and Walmart-Amazon datasets are as follows:

- **Name**: a brief text snippet describing the product.
- **Description**: a longer written description of the product.
- **Brand/manufacturer**: the name of the company responsible for creating the product.
- **Modelno**: an alpha-numeric sequence usually used to identify a product from other products created by the same manufacturer.
- **Price**: the cost of the product. This is the only numeric attribute that arises in any of the datasets.

Regarding the WDC dataset, it lacks the attribute modelno and includes two additional attributes:

- **SpecTableContent**: key/value-pairs (as one string) from the product specification tables that may exist on the offer pages.
- **Category**: one of 25 product categories the product was assigned to.

Each dataset is split into the training, validation and test sets using the ratio of 3:1:1.

The techniques utilized to concatenate the textual attributes together are treated with more details in the Chapter 4. However, an initial analysis is necessary to determine the maximum length of a single text blob containing all the attributes. This is due to transformers' maximum sequence length constraint in terms of token count, which varies depending on the architecture. For instance, BERT has a max sequence length of 512, while MPNet has 384. In cases where the dataset's sentences tend to exceed these limits, a truncation strategy, as implemented in [15], must be defined. It is worth noting that the limit is quite generous, and the only dataset (categorized as textual) that may raise concerns is Abt-Buy. Fortunately, no sentence in the dataset exceeds the limit.

To gain a better understanding of the dataset, Table 3.2 reports random samples of concatenated textual attributes values from the left and right products in the pair, labeled as "Text left" and "Text right," respectively.

Text left	**Text right**	**Label**
Abt-Buy Avg text length: 57		
sony white 8 ' portable dvd player dvpfx820w sony dvp-fx820 white 8 ' portable dvd player dvpfx820w swivel flip screen with dual sensor for remote control control buttons on screen bezel 12 bit video dac with 108 mhz processing removable , rechargeable battery car adapter included white finish	sony dvp-nc800h / s dvd player dvp-nc800hs dvd + rw , dvd-rw , dvd + r , dvd-r , cd-rw dvd video , cd-da , jpeg , mp3 , svcd , video cd playback 5 disc (s) progressive scan silver 128.69	0
delonghi oil filters fk8 delonghi oil filters fk8 made for use with d895ux 3 pack 18.0	delonghi 6-pc . replacement filters for deep fryer	1
Amazon-Google Avg text length: 16		
emc retrospect 7.5 professional for windows upgrade dantz	microsoft office and windows training professional 29.99	0
microspot macdraft professional (mac) microspot ltd. 349.99	microspot interiors (ints3 .6 sb) 95.95	0
printmaster 17 gold by encore software encore software 30.66	encore software 10478 encore printmaster v. 17.0 gold complete product creativity application 1 user (s) complete product standard pc 17.97	1
Walmart-Amazon Avg text length: 31		
da-lite da-plex deluxe rear projection screen - 78 x 139 hdtv format da-lite 7452.99 electronics - general 27573	da-plex base rear projection screen - 40 1 4 x 53 3 4 video format projection screens da-lite 1525.99	0
lg 42 class lcd 1080p 60hz hdtv 42lk450 electronics : flat panel tv 579.0 lg 42lk450	lg 42lk450 42-inch 1080p 60 hz lcd hdtv lg tvs 42lk450	1
case logic trend compact camera case mp3 accessories case logic 214336 15.19	case logic qpb-201 eva molded compact camera case magenta qpb-201magenta 9.66 cases bags case logic	0
WDC Avg text length: 36		
"Apple - 12.9- Inch iPad Pro with Wi-Fi 32 GB Silver"@en-US	"Apple 12.9-inch iPad Pro Wi-Fi - tablet 32 GB 12.9"" " Apple 12.9" ML0G2LL/A Tablets CDW.com	1
"Used Canon Rebel T3i Body D - Good"@en " Used DSLRs | Unique Photo "@en	"Canon EOS 6D(N) Digital Camera (Body Only)"@en Only) - Fumfie.com"@en	0
"Cartier" "CartierCartier Pasha C Steel Black Unisex Watch W31079M7"	"Cartier" "CartierCartier Tank Series Unisex Watch W2603556"	0

Table 3.2: Samples from the datasets. Beside the dataset name is reported the average lenght of the *Text* attribute.

Chapter 4

Methodology and implementation

In this Chapter, we aim to investigate the performance of pre-trained language models, specifically MPNet and BERT, on the task of product retrieval in a zero-shot scenario. To accomplish this, we begin with the data presented in Chapter 3, which was originally designed for the product matching task (presented in 2.2). The first step is to process the data to fit the product retrieval task (presented in 2.1). To reproduce a zero-shot scenario (2.4), we experiment various combination fine-tuning on a dataset and testing on a different one (the combinations are listed in Figure 4.5).

Building upon the findings in [30], we note that BERT induces a non-smooth, anisotropic semantic space of sentences that negatively affects its performance on semantic similarity. “Anisotropic” means that the word embeddings occupy a narrow cone in the vector space. Although various methods have been proposed to alleviate this issue (see [30, 31]), our focus is primarily on investigating the contrastive setup. As stated in [2], the contrastive loss optimizes both the alignment (closeness) of features from positive pairs and the uniformity of the induced distribution of features on the hyper-sphere. This has been shown to have positive effects on downstream tasks in the image vision domain, and similar benefits for semantic similarity in the natural language domain have been demonstrated in [3]. Furthermore, [3] demonstrates a correlation between low alignment and uniformity loss and higher average STS (Semantic Textual Similarity [32, 33, 34, 35, 36]) performance. As stated and demonstrated by the paper: "the contrastive learning objective flattens the singular value distribution of the sentence embedding space, hence improving uniformity."

As introduced in [2], the alignment loss is defined as the expected distance

between positive pairs:

$$L_{align}(f) = \underset{(x,y)\sim p_{pos}}{\mathbb{E}} ||f(x) - f(y)||^2 \tag{4.1}$$

The uniformity loss is instead defined as the logarithm of the average pairwise Gaussian potential:

$$L_{uniform}(f;t) = log \underset{(x,y)\sim p_{data}}{\mathbb{E}} e^{-2||f(x)-f(y)||^2} \tag{4.2}$$

where $e^{-2||f(x)-f(y)||^2}$ is the Gaussian potential kernel (a.k.a. Radial Basis Kernel) and t is a fixed parameter. Figure 4.1 helps to visualize the concept of uniformity.

Since our objective in both the product retrieval and product matching tasks is to enhance similarity between positives and reduce it for negatives, we measure alignment and uniformity loss in conjunction with other metrics (to be introduced shortly).

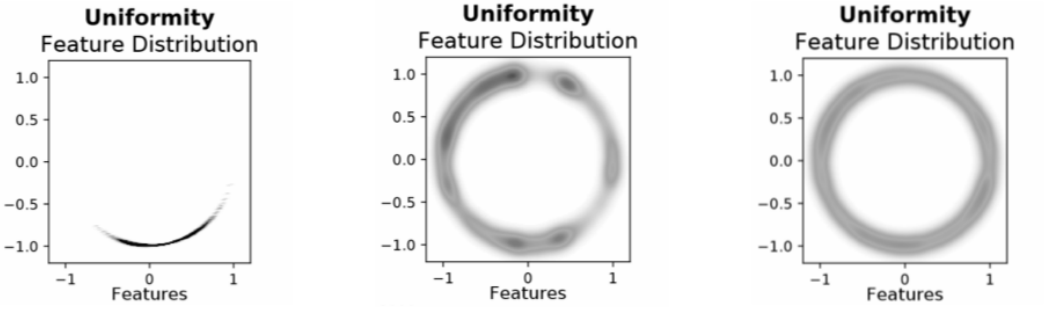


Figure 4.1: From left to right progressively more uniform representations. A lower uniformity loss translates to evenly distributed embeddings on S^1. Credits to [2].

To obtain the embeddings, we begin by concatenating the textual attributes of each product. The attributes vary depending on the dataset (see Chapter 3 for reference). Several approaches are possible, including some that are more straightforward as done in [14], where no serialization techniques are used. For

textual datasets composed of a single long description, it is directly tokenized as is, while for entities "fragmented" into attributes, all attributes are simply concatenated together. In [15] they use a more sophisticated approach using special tokens to help the model recognize different attributes as described in Section 3.

Once the attributes are concatenated, the resulting string is tokenized and then fed to the NLP model (BERT or MPNet in our case). The model outputs a fixed-size vector for each token. Various techniques exist to compress the granular token-level representations into a single fixed-length representation that is supposed to reflect the meaning of the entire sequence. In [37] *mean pooling* is suggested as the best approach, which aggregates token-level embeddings by taking their element-wise arithmetic mean. Figure 4.2 provides an overview of the entire process.

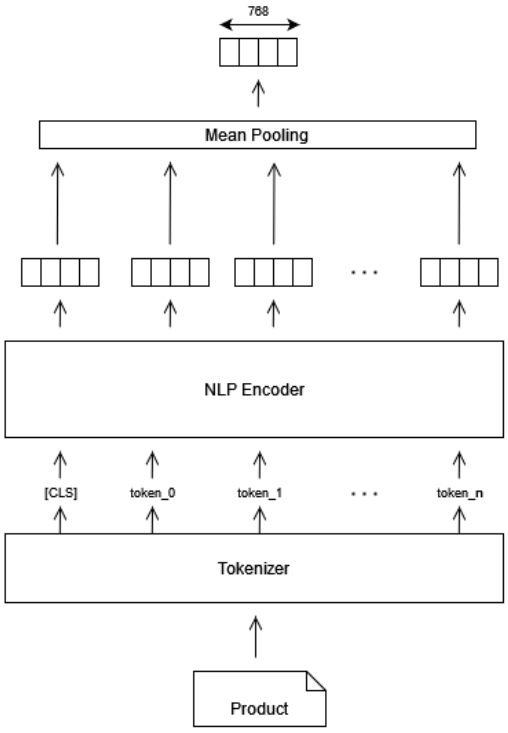


Figure 4.2: The encoding of a product entity. The textual representation of a product is first tokenized, then each token is processed by the NLP Encoder and, finally, computing element-wise arithmetic mean of the token embeddings we get the product embedding of 768 elements.

4.1 Product retrieval zero-shot setup

The concept behind our use-case is straightforward. We aim to develop a system capable of retrieving the most similar products from a given corpus. If the corpus contains products that refer to the same real-world item as the input product, these entities should rank highly. A product is defined as the representation of its textual characteristics, as discussed in the previous section. Refer to Figure 4.3 for a visual overview of the process.

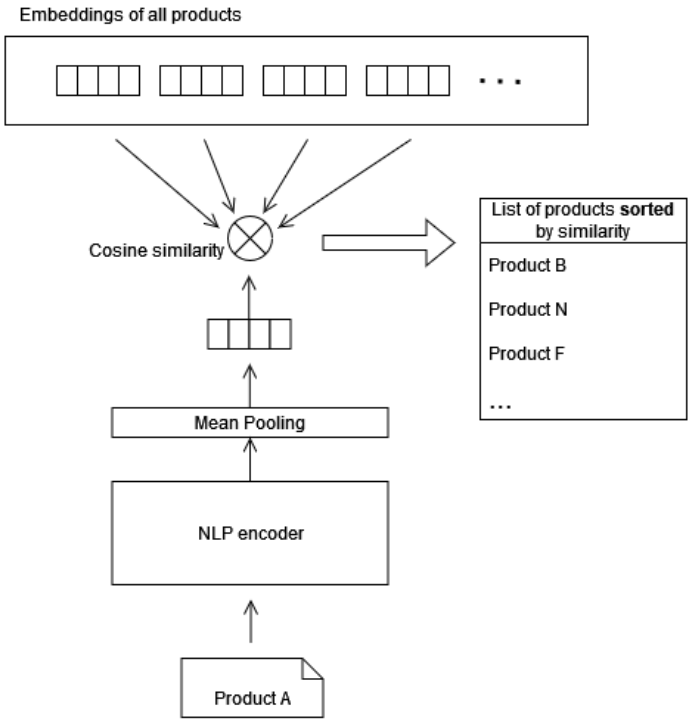


Figure 4.3: Product retrieval schematized. "Product A" is the input product. It is encoded, then we compute the cosine similarity with each embedded product in the corpus. The sorted list of products is the output.

We are interestered in 3 metrics:

- **Normalized Discounted Cumulative Gain (nDCG)**: This commonly used evaluation metric measures the relevance of entities returned by retrieval systems. nDCG returns high scores when highly relevant entities appear earlier in the results. In our context, each product embedding represents an entity. Our query is an input product from which we want to retrieve the most

similar one. Therefore, a relevant entity is a matching product, i.e. a product belonging to the same cluster as the input one. Discounted Cumulative Gain (DCG) is defined as

$$DCG_n = \sum_{i=1}^{n} \frac{rel_i}{log_2(i+1)}$$

where n is the number of products in the corpus and rel_i is the graded relevance of the result at position i. rel_i is 1 for a matching product and 0 for a **non**-matching product. The term at the denominator penalize relevant products that appear lower in the search.
To avoid a product scoring higher than another only because there are more relevant products in the corpus for its specific cluster, a normalization term is added, resulting in

$$nDCG_n = \frac{DCG_n}{IDCG_n}$$

$IDCG_n$ is the ideal discounted cumulative gain, obtained sorting all the relevant products in the corpus producing the max possible DCG for that product.

- **Recall@K, Precision@K and F1@K**: This set of metrics investigates if the system is able not only to retrieve but also recognize relevant items. We use the concept of a recommended item as a product embedding for which the cosine similarity with the input product is above a certain threshold. We try different thresholds (from 0.6 to 0.9) and keep only the best results. The definition of a relevant item remains the same as presented for nDCG. We define Recall@K as the proportion of relevant items found in the top-k recommendations:

$$Recall@K = \frac{\#\ of\ recommended\ items\ @K\ that\ are\ relevant}{\#\ of\ relevant\ items\ in\ the\ corpus}$$

Precision@K is the proportion of recommended items in the top-k set that are relevant:

$$Precision@K = \frac{\#\ of\ recommended\ items\ @K\ that\ are\ relevant}{\#\ of\ recommended\ items\ @K}$$

Lastly, we compute *F1@K* as the harmonic mean of *Recall@K* and *Precision@K*:

$$F1@K = 2 * \frac{Precision@K * Recall@K}{Precision@K + Recall@K}$$

We tested with $k = 1, 3, 5, 10$.

As shown in Chapter 3, the data is presented in pairs of products, each pair accompanied by a label that denotes whether two products refer to the same real-world entity (i.e. they are matching) or not (i.e. they are **not** matching).

To facilitate the product retrieval task, we need to convert this format into a multi-label setup, where each class corresponds to a unique real-world product. To achieve this, we draw inspiration from the formatting of data in *WDC LSPC*. Even if it is designed for the product matching task, each pair in this dataset is accompanied by a "cluster_id" field that identifies a cluster of matching products. Therefore, we create similar clusters for the Abt-Buy, Google-Amazon, and Walmart-Amazon datasets by iterating over all product pairs and encoding each distinct product in an embedding. To ensure that each product is uniquely identified across different sources, we concatenate the name of the source from which it originates to its ID (e.g., a product with $id : 15$ in *Abt* is transformed in $id : 15\_abt$). We then iterate through each pair, inserting two products with a label of 1 into the same cluster, provided that neither of them has already been added to a cluster. If one of the products has already been added to a cluster, then they are inserted into the already existing cluster. Conversely, if the pair has a label of 0, the products are added to two different clusters. The cluster ID that we obtain is later used as the label. Each product is encoded (as described at the beginning of this chapter) and saved. For an overview the processing, refer to Figure 4.4.

In a real-world use-case, these embeddings could be efficiently saved and retrieved using a vector database. Vector databases are designed to efficiently store and retrieve large collections of high-dimensional vectors, such as sentence embeddings, image features, or audio signals. To do this, they typically use indexing techniques that partition the vector space into smaller subspaces, which can be searched more efficiently than the entire space. One popular indexing technique used in vector databases is k-nearest neighbors (KNN) search, which involves dividing the space into a set of non-overlapping regions, or cells, and then searching for the k closest neighbors to a query vector within these cells. Another commonly used technique in vector databases is approximate nearest neighbor (ANN) search, which uses algorithms such as locality-sensitive hashing (LSH) and hierarchical navigable small world (HNSW) to speed up the search process. These algorithms work by mapping the high-dimensional vectors to a lower-dimensional space, where distances are more easily computed, and then searching for nearest neighbors within this lower-dimensional space. Libraries such as FAISS, NMSLIB (Non-Metric Space Library), and SPTAG (Space Partition Tree And Graph) provide efficient implementations of these algorithms.

We execute the experiments in a zero-shot setup and try different combinations, as depicted in Figure 4.5. We use the BERT and MPNET models without fine-tuning as a baseline and compare the results with those obtained after a fine-tuning iteration.

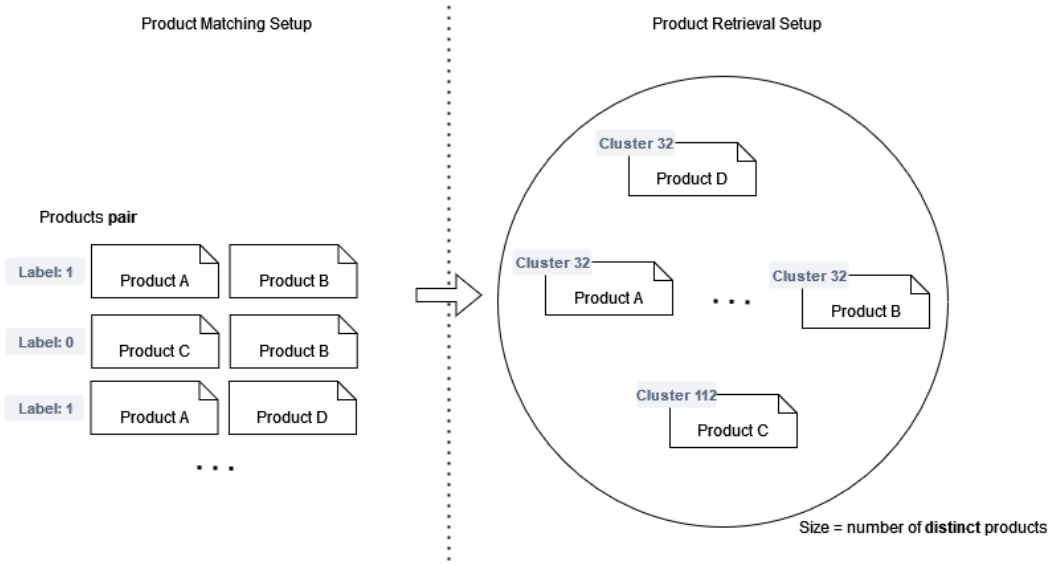


Figure 4.4: Shift data from product matching setup (on the left) to product retrieval setup (on the right). In the example product A and product B will be put in the same cluster since they have label = 1. But even product D will be put in that same cluster. Since it matches (label = 1) with product A it matches with product B too for transitive property. Product C will be put to a different cluster, since it does not match with product B and, therefore, does not match with all the others.

We are also interested in how the amount of data influences the zero-shot outcome. Therefore, for the *WDC LSCP* dataset, we fine-tune on both the small subset and the xlarge subset. To conduct a similar experiment on the Magellan datasets (i.e. Abt-Buy, Google-Amazon, Walmart-Amazon), we fine-tune on each dataset individually and on the union of all datasets.

4.2 Product matching zero-shot setup

To the best of our knowledge, we are the first to evaluate the publicly available datasets discussed in Chapter 3 for a product retrieval task. However, to ensure comparability with existing literature, in this session, we are going to assess our approach on the product matching task. As the datasets are tailored for this purpose, no pre-processing is required. Our primary metric of interest is the F1 score, and we adhere to the benchmark initially introduced in [20].

To start, we replicated the results for the current state-of-the-art methods

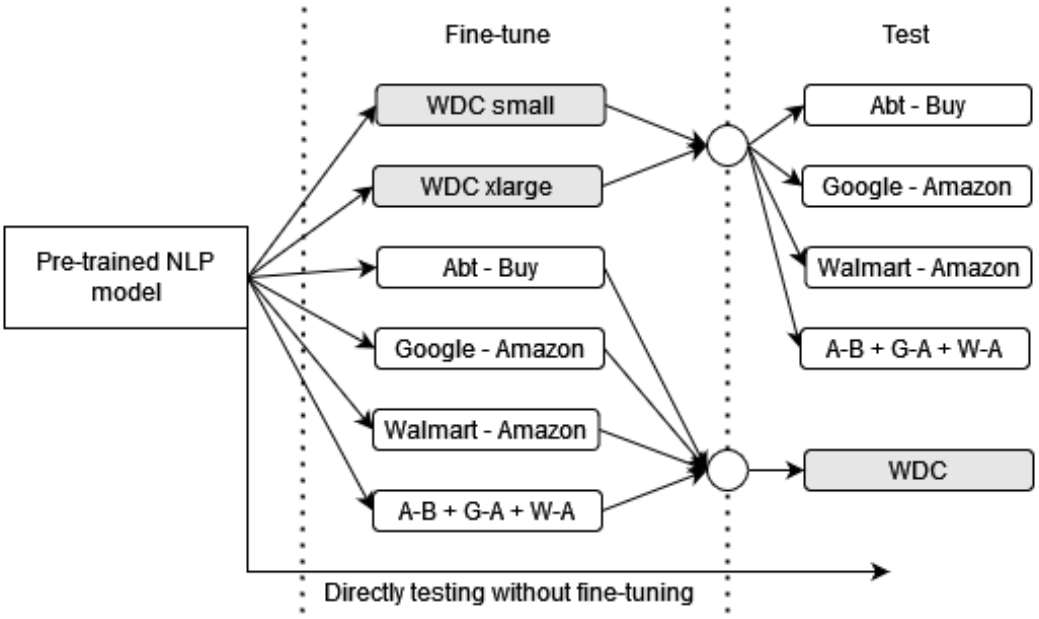


Figure 4.5: Zero-shot setup. Starting from a pre-trained NLP model, we first fine-tune it on one of the datasets in the center column. Then, depending on the dataset it has been fine-tuned on, we test on one of the datasets on the left column. As Baseline, we test the pre-trained NLP model directly without fine-tuning.

presented in [15, 14, 18] using BERT model. As depicted in Figure 4.6, two product descriptions are concatenated and tokenized, incorporating two special tokens: the classification token [CLS] and the [SEP] token (used by the network to distinguish between the two entities). The [SEP] token is employed during the "next sentence classification" task used to pretrain BERT. The resulting tokenized sentence is then input into the architecture, and the output from the initial position is taken (which corresponds to the embedding of the [CLS] token). This vector of size *hidden_size* (768 in BERT, but it varies according to the architecture) is subsequently utilized as input for a classifier, usually a simple linear classifier with a single binary label (0 when there is no match and 1 otherwise).

From now on we'll refer to this methodology as the pairwise methodology.

As the number of potential matches increases quadratically with the number of products, direct record-to-record comparison becomes eventually unfeasible. Various blocking techniques have been suggested in the literature [15] to reduce the number of potential matches. Our approach focus on developing a model that generates semantic embeddings for each product separately that can be pre-computed, which

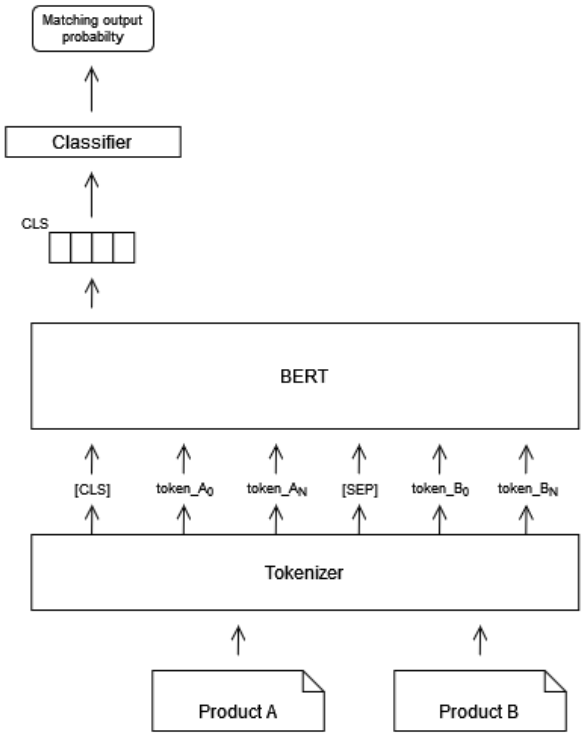


Figure 4.6: BERT fine-tuned on the *product matching* downstream task. First the two products are tokenized, then the tokens are processed by BERT model. Lastly, the embedding of the CLS token is used as input for a linear classifier, which output the probability of a match.

can then be retrieved quickly through techniques such as FAISS (Facebook AI Similarity Search) that we presented before. To execute the task by relying solely on the distance between embeddings, we first pre-train the models using contrastive learning (see the next section for details). Then, during the validation phase, two products are considered a match if the distance between their embeddings is above a certain threshold. This threshold is considered a hyperparameter to be optimized. Figure 4.7 provides an overview of our approach. There are several metrics commonly used to measure similarity between two embeddings, including cosine similarity, Euclidean distance, Manhattan distance, and Jaccard similarity. I'll describe each of these in more detail below, along with the formulas used to compute them:

- Cosine similarity: This measures the cosine of the angle between two vectors in a high-dimensional space, and is widely used in natural language processing

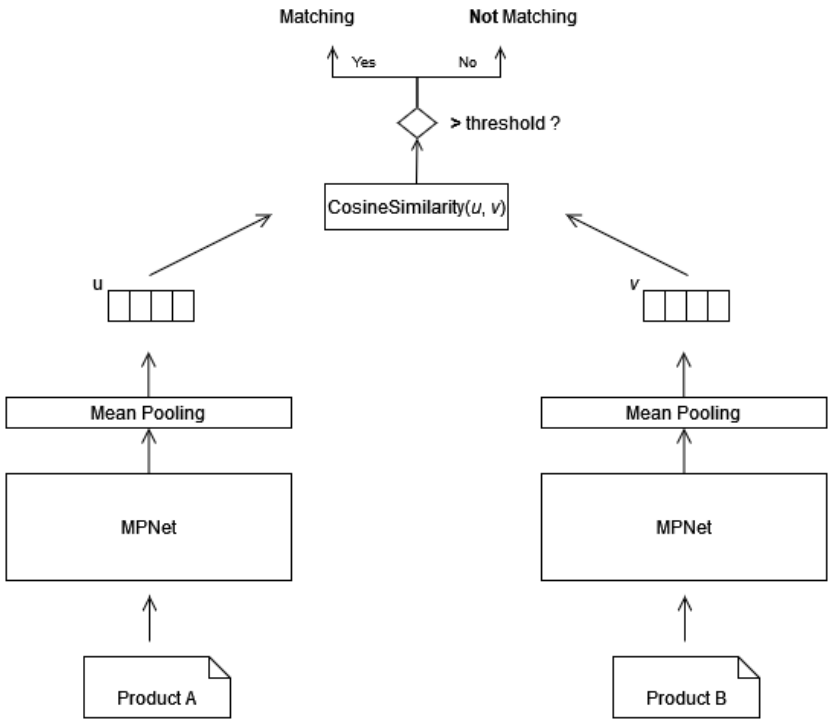


Figure 4.7: Using cosine similarity in the product matching task.

applications. The formula for cosine similarity is

$$cos\_sim(x, y) = \frac{x \cdot y}{||x||||y||}$$

- Euclidean distance: This measures the straight-line distance between two points in a high-dimensional space, and is often used in computer vision applications. The formula for Euclidean distance is:

$$euclidean\_dist(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

- Manhattan distance: This measures the distance between two points in a high-dimensional space by summing the absolute differences between their coordinates. The formula for Manhattan distance is:

$$manhattan\_dist(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

- Jaccard similarity: This measures the similarity between two sets of features, and is often used in text mining and information retrieval applications. The formula for Jaccard similarity is:

$$jaccard\_sim(x, y) = \frac{x \cap y}{x \cup y}$$

  $x \cap y$ is the size of the intersection of the two sets, and $x \cup y$ is the size of the union of the two sets.

As it is generally the most widespread, we use cosine similarity.

Despite the task being previously addressed, insufficient attention has been given to the zero-shot scenario, despite its significance in real-world applications. Therefore, we conduct experiments under the configurations depicted in Figure 4.5.

4.3 Fine-tuning BERT and MPNet through supervised contrastive learning

Leveraging the supervised contrastive loss function (refer to Section 2.3), we fine-tune BERT and MPNet models on our publicly accessible data. During the training phase, we extract batches of products from the dataset and employ the cluster ID as a label. This enables us to bring positive products closer together while pushing negative products apart, with the distance between them being computed as the dot product between embeddings. Refer to Figure 4.8 for an overview of the training process.

Subsequently, we validate the models depending on the downstream task at hand. Refer to Section 4.1 for product retrieval and Section 4.2 for product matching. Our experimentation involves varying the learning rate, weight decay, and warmup ratio. we have determined after exploring the hyper-parameter space that the most optimal configuration is to maintain a learning rate of 2e-5, a weight decay of 0.01, and a warmup ratio of 0.05.

For simplification purpose, from now on in the thesis we'll call the union of the datasets Abt-Buy, Google-Amazon and Walmart-Amazon the Magellan dataset.

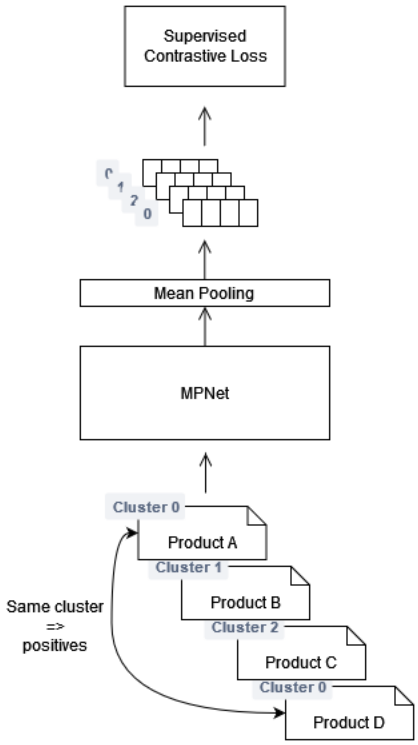


Figure 4.8: Fine-tuning a model (MPNet in this case) using a supervised contrastive loss. Positives are identified by two products with the same cluster_id.

Chapter 5

Results and evaluation

In this Chapter, we present our findings from an initial examination of the alignment and uniformity properties. Subsequently, we delve into the outcomes for our targeted tasks.

5.1 Alignment and Uniformity analysis

In the opening of the previous chapter (Chapter 4), we introduced the notions of alignment and uniformity. Alignment calculates expected distance between embeddings of products within the same cluster, while uniformity measures how well the embeddings are uniformly distributed. Table 5.1 illustrates the modification of alignment and uniformity loss due to fine-tuning using contrastive learning. The findings for both BERT and MPNet architectures are outlined. Since we are dealing with loss values, the less they are the better.

As anticipated, since MPNet has been pre-trained on sentence pairs, the resulting embeddings are more uniformly dispersed in the vector space (lower uniformity loss) across all datasets, with the exception of Abt-Buy. Conversely, the alignment loss remains mostly the same, even slightly better for BERT.

Post fine-tuning, the uniformity loss declines for both architectures, while alignment remains nearly the same. Upon comparing the outcomes for one dataset after fine-tuning on the same dataset versus fine-tuning in a zero-shot scenario (e.g. the results on WDC after fine-tuning on Magellan), the losses for both alignment and uniformity are higher in the zero-shot scenario, as expected, but not by a significant margin. However, the most crucial drop in loss is between the setup where we use the model without any fine-tuning and the one where we fine-tune with another dataset.

It is noteworthy that following finetuning on WDC_XLarge, the uniformity is notably inferior in comparison to the finetuning on WDC_small, which contains

only a fraction of the samples. Observing the value in the **non**-zero-shot scenario (i.e. target: WDC, finetuning: WDC_XLarge), the alignment loss is almost zero. While this might prove advantageous in certain contexts, it may also imply a form of degeneracy in the embeddings that occupy a restricted cone in the vector space, resulting in reduced generalization.

Table 5.1: Results of Alignment and Uniformity loss before and after having fine-tuned the model with supervised contrastive learning. These are loss values, so the less they are the "better". Results in bold corresponds to the best results for the particular metric in the row.

Target dataset	Loss	BEFORE FT		FT on wdc_small		FT on wdc_xlarge	FT on magellan	
		BERT	MPNet	BERT	MPNet	MPNet	BERT	MPNet
abt_buy	Alignment	0.3	0.3	0.33	0.47	0.37	NOT zero-shot	
	Uniformity	-0.79	-0.76	-2.43	**-3.1**	-2.7		
google_amazon	Alignment	0.25	0.43	0.41	0.42	0.36		
	Uniformity	-1.15	-2.6	-2.38	**-3.1**	-2.35		
walmart_amazon	Alignment	**0.17**	0.37	0.24	0.33	0.23		
	Uniformity	-0.9	-2.7	-2.49	**-3.1**	-2.46		
magellan	Alignment	**0.24**	0.42	0.33	0.41	0.32	0.32	-0.26
	Uniformity	-0.96	-2.8	-2.3	**-3.2**	-2.5	-3.3	-3.8
wdc	Alignment	-0.27	0.48	0.21	0.37	**0.09**	0.55	0.49
	Uniformity	-1.05	-2.85	-2.93	**-3.73**	-3.62	-3.3	-3.5

5.2 Product retrieval zero-shot

We report the resuls for the product retrieval task divided in two subsections: Baseline 5.2.1 and Zero-Shot finetuning 5.2.2.

5.2.1 Baseline

The results for the baseline, where models are used out-of-the box with no fine-tuning whatsoever, are reported in Table 5.2. These results serve as a crucial baseline for the zero-shot results. It is worth noting that MPNet outperforms BERT in terms of results on such baseline. This is consistent with the considerations made for the alignment and uniformity loss; however, BERT manages to achieve results that are not so far off.

5.2.2 Zero-Shot finetuning

The results in this section are obtained after having fine-tuned the models using contrastive learning as discussed in the previous Chapter 4. Results are reported

Table 5.2: Results for product retrieval with pre-trained models without finetuning. R / P / F1 stands for Recall, Precision and F1. Results in bold corresponds to the best results for the particular metric in the row.

Target dataset	Metric	No finetuning	
		BERT	MPNet
abt_buy	NDCG	0.28	**0.67**
	R / P / F1 @1	0,04 / 0,05 / 0,05	**0,29 / 0,30 / 0,29**
	R / P / F1 @3	0,09 / 0,03 / 0,05	**0,51 / 0,28 / 0,34**
	R / P / F1 @5	0,12 / 0,03 / 0,05	**0,59 / 0,26 / 0,33**
	R / P / F1 @10	0,15 / 0,02 / 0,03	**0,73 / 0,26 / 0,32**
google_amazon	NDCG	0.68	**0.78**
	R / P / F1 @1	0,41 / 0,42 / 0,41	**0,46 / 0,47 / 0,46**
	R / P / F1 @3	0,59 / 0,23 / 0,32	**0,61 / 0,38 / 0,44**
	R / P / F1 @5	0,66 / 0,18 / 0,26	**0,68 / 0,36 / 0,43**
	R / P / F1 @10	0,75 / 0,14 / 0,20	**0,75 / 0,35 / 0,41**
walmart_amazon	NDCG	0.75	**0.88**
	R / P / F1 @1	0,48 / 0,49 / 0,48	**0,66 / 0,67 / 0,66**
	R / P / F1 @3	0,68 / 0,23 / 0,34	**0,82 / 0,51 / 0,60**
	R / P / F1 @5	0,76 / 0,15 / 0,26	**0,88 / 0,50 / 0,57**
	R / P / F1 @10	0,84 / 0,09 / 0,15	**0,93 / 0,47 / 0,54**
magellan	NDCG	0.56	**0.75**
	R / P / F1 @1	0,31 / 0,31 / 0,31	**0.46 / 0.47 / 0.46**
	R / P / F1 @3	0,45 / 0,16 / 0,23	**0.64 / 0.37 / 0.44**
	R / P / F1 @5	0,51 / 0,11 / 0,18	**0.71 / 0.35 / 0.43**
	R / P / F1 @10	0,59 / 0,06 / 0,11	**0.80 / 0.34 / 0.41**
wdc	NDCG	0.43	**0.6**
	R / P / F1 @1	0,17 / 0,38 / 0,22	**0.22 / 0.5 / 0.30**
	R / P / F1 @3	0,26 / 0,22 / 0,22	**0.4 / 0.33 / 0.34**
	R / P / F1 @5	0,31 / 0,16 / 0,19	**0.47 / 0.25 / 0.31**
	R / P / F1 @10	0,37 / 0,10 / 0,15	**0.6 / 0.18 / 0.25**

Table 5.3: Results for zero-shot product retrieval task when finetuning is performed on WDC and the evaluation on Magellan. Results in bold corresponds to the best results for the particular metric in the row.

Target dataset	Metric	Finetuned on wdc_small		Finetuned on wdc_xlarge
		BERT	MPNet	MPNet
abt_buy	NDCG	0.84	**0.86**	**0.86**
	R / P / F1 @1	0,51 / 0,51 / 0,51	0.53 / 0.54 / 0.54	**0,61 / 0,61 / 0,61**
	R / P / F1 @3	0,60 / 0,50 / 0,52	**0.70 / 0.48 / 0.54**	0,74 / 0,44 / 0,52
	R / P / F1 @5	0,61 / 0,49 / 0,52	**0.74 / 0.47 / 0.53**	0,78 / 0,41 / 0,48
	R / P / F1 @10	0,63 / 0,49 / 0,52	**0.77 / 0.47 / 0.53**	0,82 / 0,39 / 0,46
google_amazon	NDCG	0.77	**0.88**	0.72
	R / P / F1 @1	0,47 / 0,48 / 0,47	**0,60 / 0,60 / 0,60**	0,44 / 0,45 / 0,45
	R / P / F1 @3	0,65 / 0,37 / 0,45	**0,76 / 0,48 / 0,55**	0,62 / 0,28 / 0,37
	R / P / F1 @5	0,68 / 0,34 / 0,42	**0,83 / 0,45 / 0,53**	0,68 / 0,23 / 0,32
	R / P / F1 @10	0,71 / 0,33 / 0,40	**0,87 / 0,43 / 0,50**	0,74 / 0,20 / 0,27
walmart_amazon	NDCG	0.93	**0.96**	0.91
	R / P / F1 @1	0,79 / 0,80 / 0,79	**0,81 / 0,81 / 0,81**	0,73 / 0,74 / 0,73
	R / P / F1 @3	0,88 / 0,69 / 0,74	**0,91 / 0,71 / 0,76**	0,82 / 0,56 / 0,63
	R / P / F1 @5	0,88 / 0,67 / 0,72	**0,91 / 0,70 / 0,75**	0,82 / 0,53 / 0,60
	R / P / F1 @10	0,89 / 0,67 / 0,72	**0,91 / 0,70 / 0,74**	0,84 / 0,52 / 0,58
magellan	NDCG	0.83	**0.89**	0.81
	R / P / F1 @1	0.59 / 0.59 / 0.59	**0.62 / 0.62 / 0.62**	0,54 / 0,55 / 0,55
	R / P / F1 @3	0.71 / 0.50 / 0.55	**0.76 / 0.54 / 0.60**	0,66 / 0,41 / 0,47
	R / P / F1 @5	0.73 / 0.48 / 0.54	**0.79 / 0.52 / 0.58**	0,68 / 0,38 / 0,44
	R / P / F1 @10	0.75 / 0.47 / 0.53	**0.81 / 0.51 / 0.57**	0,71 / 0,37 / 0,42
wdc	NDCG	0.89	0.91	**0.99**
	R / P / F1 @1	0,39 / 0,83 / 0,50	0,40 / 0,85 / 0,51	**0,46 / 0,97 / 0,59**
	R / P / F1 @3	0,74 / 0,73 / 0,71	0,77 / 0,73 / 0,72	**0,91 / 0,88 / 0,87**
	R / P / F1 @5	0,81 / 0,67 / 0,70	0,85 / 0,66 / 0,71	**0,98 / 0,80 / 0,85**
	R / P / F1 @10	0,86 / 0,64 / 0,69	0,9 / 0,62 / 0,69	**0,99 / 0,76 / 0,83**

in Table 5.4 and 5.3. In general, all the methods considerably improves the score compared to raw pre-trained models (seen in Table 5.2).

It is interesting to see that, in the zero-shot setup, finetuning on WDC XLarge performs very poorly, while it gives excellent results in the scenario where the evaluation is executed on the test subset of the same dataset used for training. As underlined before (see Table 5.1) the high alignment and relatively low uniformity is a detrimental combination in the zero-shot setup.

Generally, the methods that perform the best with one metric, like NDCG, tends to perform the best also on the others.

Table 5.4: Results for zero-shot product retrieval task when finetuning is performed on Abt - Buy, Amazon - Google and/or Walmart - Amazon and the evaluation on WDC.

Target dataset	Metric	Finetuned on abt_buy	
		BERT	MPNet
wdc	NDCG	0.72	**0.8**
	R / P / F1 @1	0,31 / 0,68 / 0,40	0.35 / 0.75 / 0.45
	R / P / F1 @3	0,55 / 0,51 / 0,50	0.63 / 0.60 / 0.58
	R / P / F1 @5	0,62 / 0,43 / 0,47	0.70 / 0.52 / 0.56
	R / P / F1 @10	0,68 / 0,36 / 0,42	0.76 / 0.46 / 0.51
Target dataset	Metric	Finetuned on google_amazon	
		BERT	MPNet
wdc	NDCG	0.59	**0.7**
	R / P / F1 @1	0,22 / 0,52 / 0,30	0.27 / 0.61 / 0.35
	R / P / F1 @3	0,40 / 0,35 / 0,35	0.50 / 0.44 / 0.44
	R / P / F1 @5	0,48 / 0,27 / 0,32	0.60 / 0.36 / 0.42
	R / P / F1 @10	0,58 / 0,20 / 0,27	0.71 / 0.28 / 0.37
Target dataset	Metric	Finetuned on walmart_amazon	
		BERT	MPNet
wdc	NDCG	0.72	**0.79**
	R / P / F1 @1	0,29 / 0,66 / 0,38	0.33 / 0.73 / 0.43
	R / P / F1 @3	0,51 / 0,53 / 0,49	0.60 / 0.59 / 0.56
	R / P / F1 @5	0,55 / 0,47 / 0,47	0.67 / 0.52 / 0.54
	R / P / F1 @10	0,60 / 0,44 / 0,45	0.71 / 0.47 / 0.51
Target dataset	Metric	Finetuned on magellan	
		BERT	MPNet
wdc	NDCG	0.75	**0.81**
	R / P / F1 @1	0,32 / 0,71 / 0,42	0,35 / 0,77 / 0,46
	R / P / F1 @3	0,59 / 0,53 / 0,52	0,65 / 0,60 / 0,60
	R / P / F1 @5	0,67 / 0,43 / 0, 50	0,73 / 0,50 / 0,56
	R / P / F1 @10	0,74 / 0,35 / 0,43	0,80 / 0,42 / 0,50

5.3 Product matching zero-shot

Following the same schema of the previous section, we report the results for the product matching task divided in subsections (three instead of two this time): Baseline 5.3.1, Zero-Shot finetuning 5.3.2 and Same dataset finetuning 5.3.3.

5.3.1 Baseline

Results for the baseline, where **no** fine-tuning is performed, are shown in Table 5.5. It is worth noting that MPNet outperforms BERT in terms of results on such baseline, as it was the case in the product retrieval task.

Table 5.5: Results for product matching with pre-trained models without finetuning.

Target dataset	No finetuning	
	BERT Cosine	MPNet Cosine
abt_buy	0.19	**0.34**
google_amazon	0.31	**0.4**
walmart_amazon	0.19	0.2
magellan	0.21	**0.28**
wdc	0.41	**0.52**

5.3.2 Zero-Shot finetuning

Tables 5.6 and 5.7 report the outcomes for the product matching task in the zero-shot setup. This time we also take into consideration BERT Pairs, which refers to the method of feeding the model during finetuning with pairs of product entities. For product retrieval it was not taken into consideration, since the time complexity of the retrieval operation would make it unfeasible to process all the samples pairwise. Although, on average, MPNet performs slightly better, we can see that there is no significant difference in terms of F1 score between the evaluated methods. However, it is important to remind that by utilizing contrastive learning, we can efficiently identify matching products by calculating the distance between two separately computed embeddings.

In general, all the methods considerably improves the score compared to the baseline (seen in Table 5.5).

All the considerations done for the previous task regarding the dataset size can be re-proposed. Finetuning on WDC XLarge generalizes poorly, as it shifts from excellent results when evaluating on WDC to worst results in the zero-shot setup.

Table 5.6: Results of F1 score for product matching task. The models have been finetuned on the Abt-Buy, Amazon-Google and Walmart-Amazon datasets and then evaluated on WDC.

Finetuned on magellan		
BERT Pairs	BERT Cosine	MPNet Cosine
0.68	0.66	**0.7**
Finetuned on abt_buy		
BERT Pairs	BERT Cosine	MPNet Cosine
0.68	0.65	**0.69**
Finetued on google_amazon		
BERT Pairs	BERT Cosine	MPNet Cosine
0.61	0.49	0.58
Finetuned on walmart_amazon		
BERT Pairs	BERT Cosine	MPNet Cosine
0.61	0.63	**0.7**

Table 5.7: Results of F1 score for product matching task. The models have been finetuned on wdc small and wdc xlarge and evaluated on different datasets.

Target dataset	Finetuned on wdc_small			Finetuned on wdc_xlarge	
	BERT Pairs	BERT Cosine	MPNet Cosine	BERT Pairs	MPNet Cosine
abt_buy	**0.64**	0.62	**0.64**	0.66	0.51
google_amazon	0.42	0.43	**0.51**	0.38	0.37
walmart_amazon	**0.51**	**0.51**	0.49	0.51	0.41
magellan	**0.5**	0.45	**0.5**	0.49	0.37

5.3.3 Same dataset finetuning

Although we are interested in the zero-shot scenario, results for a more conventional approach are also included in order to evaluate the effectiveness of fine-tuning with contrastive learning with regard to the methodology proposed in the literature for the datasets used. Therefore, the outcomes for the **non**-zero-shot scenario are presented in Table 5.8. We observe that the pairwise method tends to yield slightly superior outcomes. Nevertheless, for the Abt-Buy and the WDC datasets, the score is the same, which could be attributed to the percentage of product entities for which a positive is not present in the dataset's corpus. For Google-Amazon and Walmart-Amazon, where contrastive learning performs the poorest, this percentage is respectively 54% and 78%. While for Abt-Buy (38%) and wdc small (42.8%), it is relatively lower. This consideration is based on the fact that supervised contrastive learning exploits positive pairs to "align" matching products, therefore, it could be highly penalized by the absence of positive instances (see Section 2.3). This consideration seems confirmed by the fact that for wdc xlarge, where only 7% of products have no positive, contrastive learning in this scenario works very well.

Table 5.8: Results for the **non** zero-shot scenario. BERT Pairs refers to the method of feeding the model during finetuning with pairs of product entities. BERT Cosine and MPNet cosine refers to the model beeing finetuned with contrastive learning.

Finetuned on abt_buy -> evaluated on abt_buy		
BERT Pairs	BERT Cosine	MPNet Cosine
0.84	0.8	**0.85**
Finetuned on google_amazon -> evaluated on google_amazon		
BERT Pairs	BERT Cosine	MPNet Cosine
0.81	0.71	0.73
Finetuned on walmart_amazon -> evaluated on walmart_amazon		
BERT Pairs	BERT Cosine	MPNet Cosine
0.82	0.73	0.77
Finetuned on wdc_small -> evaluated on wdc		
BERT Pairs	BERT Cosine	MPNet Cosine
0.85	0.84	0.84
Finetuned on wdc_xlarge -> evaluated on wdc		
BERT Pairs		MPNet Cosine
0.93		0.97

5.4 A couple more experiments

Since the experiments up until now showed a clear correlation between the score and a low uniformity loss in both tasks, Table 5.9 shows how the dimension of the finetuning dataset affect the uniformity and alignment of the embeddings. Contrary to what you would expect, a bigger dataset, while considerably improves performances in the standard scenario, it is detrimental in the zero-shot scenario: the medium subset of the WDC dataset already brings degeneration of the embeddings degrading uniformity.

Given the clear correlation between score and a low uniformity loss observed in our experiments thus far, Table 5.9 illustrates the impact of the size of the finetuning dataset on the uniformity and alignment of the embeddings. Surprisingly, a larger dataset, while enhancing performance in the standard scenario, actually harms performance in the zero-shot scenario: even the medium subset of the WDC dataset leads to degenerated embeddings that compromise uniformity.

Table 5.9: A table showing how the size of the finetuning dataset influence uniformity and alignment. The F1 score on the product matching task is also reported for comparison. Small, medium, large and xlarge refers to the subsets of WDC dataset. The best results for the zero-shot scenario (i.e. target dataset "magellan"), both in terms of F1 score and uniformity, are obtained after finetuning on the small subset.

Target dataset	Metric	MPNet				
		NO FT	FT on small	FT on medium	FT on large	FT on xlarge
magellan	F1	0.21	**0.5**	0.49	0.74	0.37
	Align Loss	0.42	0.41	0.33	**0.27**	0.32
	Unif Loss	-2.8	**-3.2**	-2.88	-2.4	-2.5
wdc	F1	0.41	0.84	0.93	0.96	**0.97**
	Align Loss	0.48	0.37	0.17	0.1	**0.09**
	Unif Loss	-2.85	**-3.73**	-3.71	-3.6	-3.62

Lastly, for a qualitative analysis of the product retrieval task, the Graphs 5.1, 5.2, 5.3, 5.4 showcase the results from one last experiment that we conducted. The retrieval outcome for a randomly sampled product from the corpus is displayed for each dataset and both architectures. This is done to further elucidate the structure of the task and how the fine-tuning with supervised contrastive learning affects the similarity between embeddings. Given an input product embedding, the histograms display the sorted similarity score of the top 10 similar product embeddings. The darker histograms represent products labeled as matching with the input product.

We can see that BERT's embeddings are clustered closely together in the vector space before fine-tuning, while MPNet is already able to differentiate between them. After fine-tuning, this gap narrows, and the qualitative analysis suggests that they

are more evenly distributed.

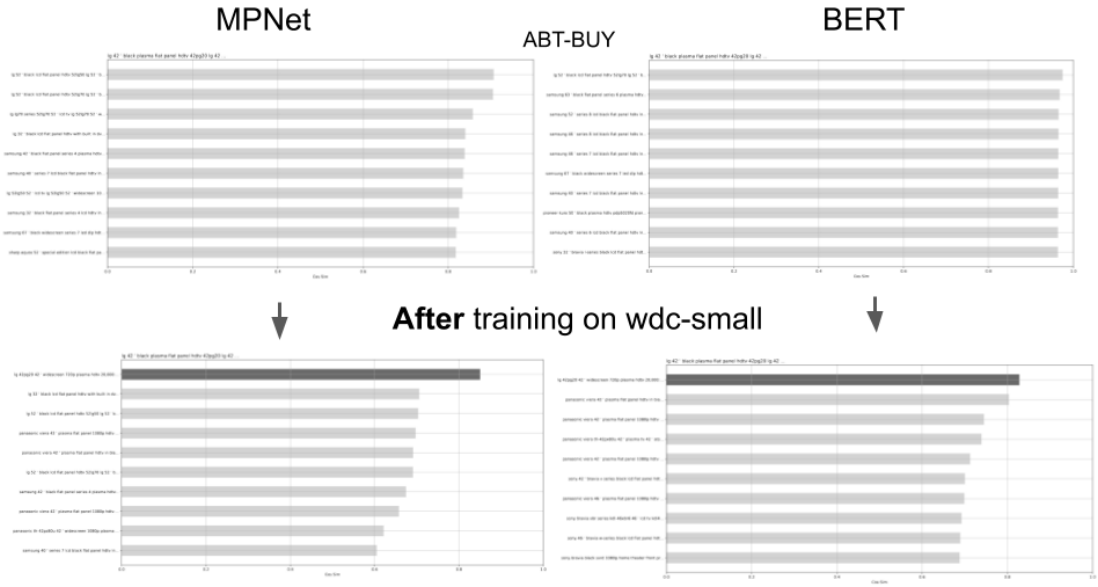


Figure 5.1: Qualitative analysis of *product retrieval* performances on Abt-Buy before and after finetuning on a different dataset. The darker histograms represent products labeled as matching the the input product.

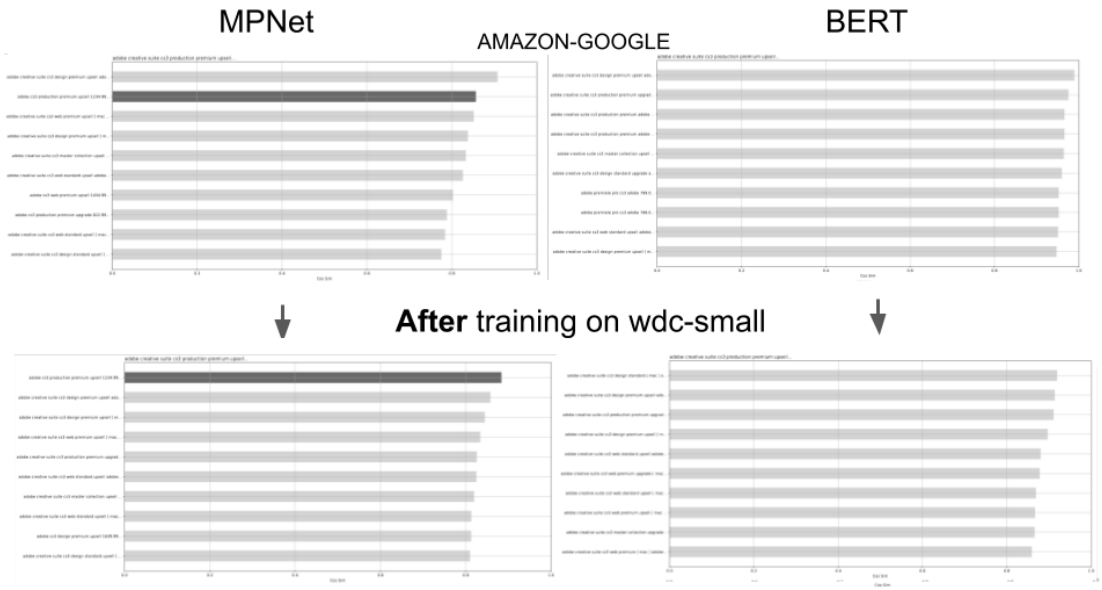


Figure 5.2: Qualitative analysis of *product retrieval* performances on Amazon-Google before and after finetuning on a different dataset. The darker histograms represent products labeled as matching the the input product.

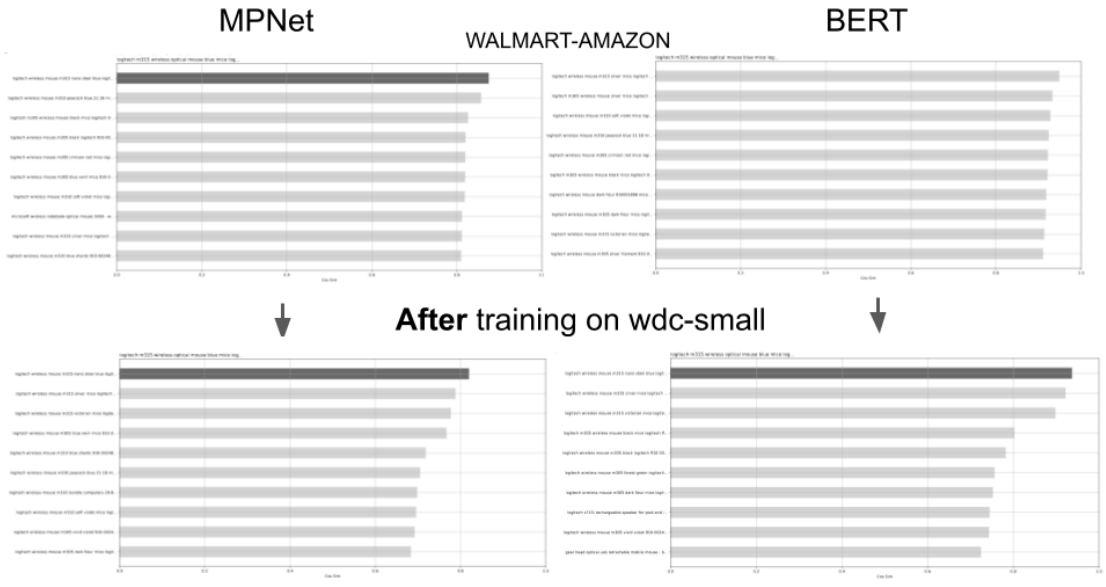


Figure 5.3: Qualitative analysis of *product retrieval* performances on Walmart-Amazon before and after finetuning on a different dataset. The darker histograms represent products labeled as matching the the input product.

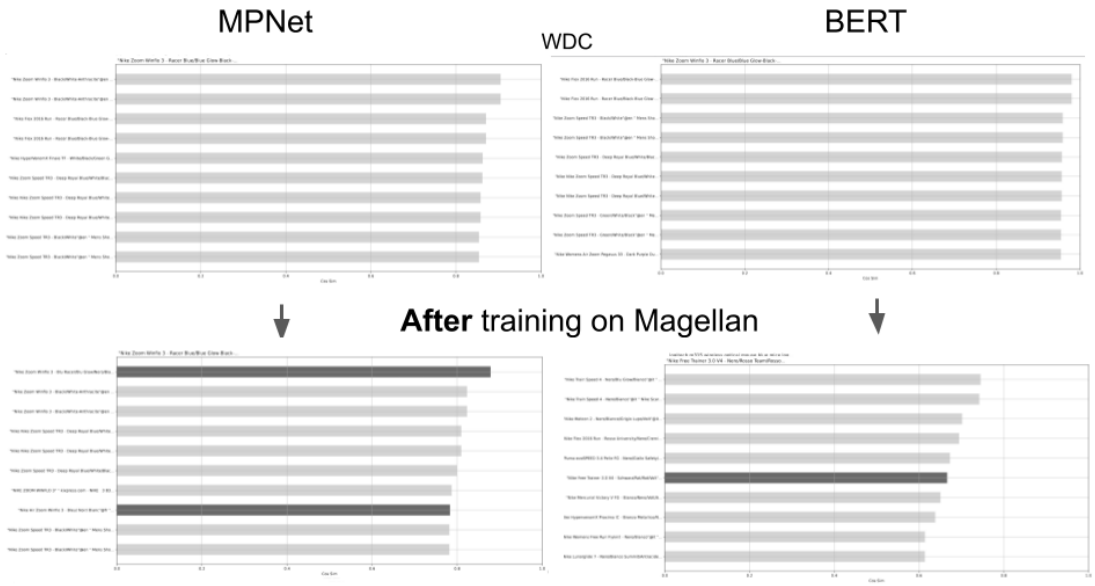


Figure 5.4: Qualitative analysis of *product retrieval* performances on WDC before and after finetuning on a different dataset. The darker histograms represent products labeled as matching the the input product.

Chapter 6

Discussion and conclusion

We believe that there are various valuable insights regarding the product retrieval task that can be obtained from our studies.

It is particularly critical, to perform well on the zero-shot setup, to achieve a low uniformity loss. On the other hand, a low alignment loss may indicate "degenerate" embeddings, which can lead to an anisotropic semantic space [30], an issue we previously discussed in the thesis. In the results, this correlation is evident, especially when examining how poorly the models perform after fine-tuning on larger datasets.

In order to obtain better uniformly distributed embeddings, as discussed in [3], the contrastive loss is a valuable instrument. However, as evaluated in [18], the unsupervised contrastive loss behaves poorly on pairs of matching products. This is because unsupervised contrastive learning is based on pulling augmented versions of the same products closer and pushing away all the others. Since the datasets are built on pairs of matching products, this has the opposite effect of what we are looking for. Each product in a matching pair is pulled closer to its augmented version but is pushed away from the other product in the matching pair. Therefore, to train the model and maintain stable alignment while achieving better uniformity, it is crucial to adopt the supervised contrastive loss [6], which exploits label information to avoid matching products being pushed away. The embeddings obtained in this way can generalize quite well, as suggested by the significant improvements in the zero-shot setup compared to the embeddings obtained through pre-trained models without fine-tuning. In [3], it was demonstrated that better uniformity of the embeddings translates to better performance in average STS (semantic textual similarity) benchmarks. This could imply that fine-tuning using supervised contrastive loss with matching and non-matching product pairs (as proposed in the thesis) can enable the model to compute more realistic similarity scores between all products. This still needs to be evaluated on a production dataset, which is what we are planning to do with the startup team. Obtaining

data for the fine-tuning step is not difficult because, in Europe, a vast portion of products is tagged with an EAN (European Article Number), which is a GTIN (Global Trade Item Number) [12], similar and compatible with the American UPC (Universal Product Code) [11]. The EAN allows determining if products from different sources match and can be used to quickly build a dataset to train the model with supervised contrastive learning. The difficult part is evaluating the performance on the more general similarity score computation task.

Another interesting insight concerns the number of samples for fine-tuning. While in a non-zero-shot scenario, the more the samples, the better the results, in a zero-shot scenario, it is crucial to find the sweet spot that allows uniformity not to start dropping.

Lastly, starting from a model that adopts an autoregressive learning paradigm and it is already fine-tuned on STS task, MPNet in our case, seems to yield better results compared to starting from model that is trained as autoencoder on general Masked Language model, i.e. BERT.

Appendix A

Deep learning in natural language processing

In this section, we present a comprehensive overview of the technologies and techniques utilized in the domain of natural language processing. This is done not only to provide the reader with background information, but also to justify our decision to employ transformer-based architectures in our experiments. Prior knowledge of neural networks is required, and there are a plethora of online resources available. For a brief yet informative introduction, the author recommends the "Neural Networks" series on the YouTube channel "3Blue1Brown".

Historically, natural language processing (NLP) has been dominated by architectures utilizing recurrent neural networks (RNNs) [22] as their core, as RNNs are particularly well-suited for processing sequences of inputs. As depicted in Figure A.1, each input token is processed by the RNN together with a hidden state that contains information from past tokens.

Despite the RNN architecture being well-suited for sequential text data, several issues were identified. The first issue is that learning long-range dependencies with RNNs is difficult. If the input sequence is particularly long, during the backpropagation phase, the high number of matrix multiplications disrupts the uninterrupted flow of the gradient, resulting in issues such as exploding or vanishing gradients [38, 39]. In order to address the issue of vanishing gradients, a change in the architecture was necessary. Long short-term memory (LSTM) was introduced to address this problem [40]. LSTMs overcome the problem of vanishing gradients by ignoring useless data or information in the network, thanks to the introduction of gates that allow the gradient to flow during backpropagation. See Figure A.2. Other solutions to learning long-range dependencies, such as gated recurrent units (GRUs), also became popular [41].

These architectures exhibit a considerable degree of flexibility, their usage and

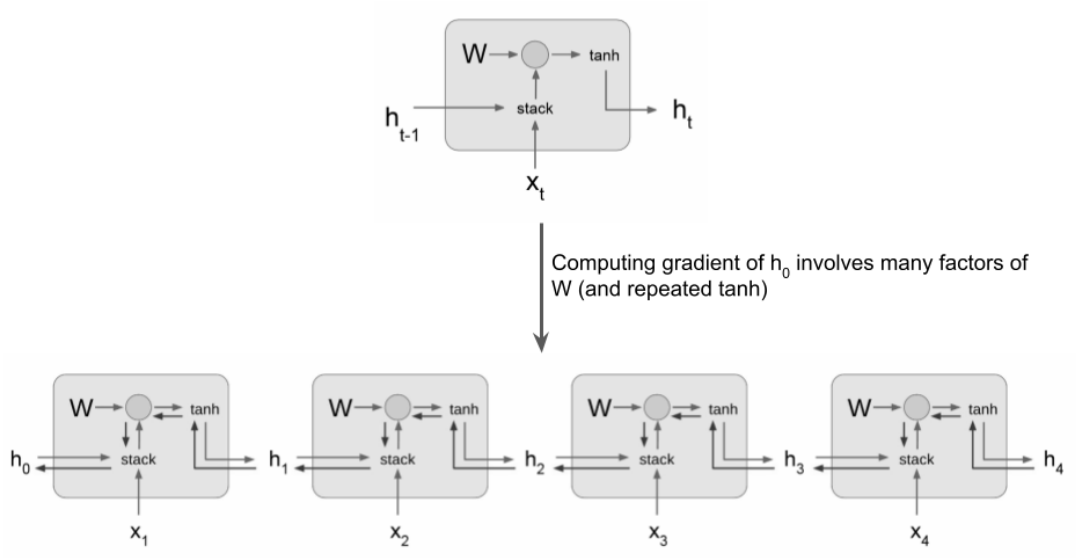


Figure A.1: Vanilla RNN.

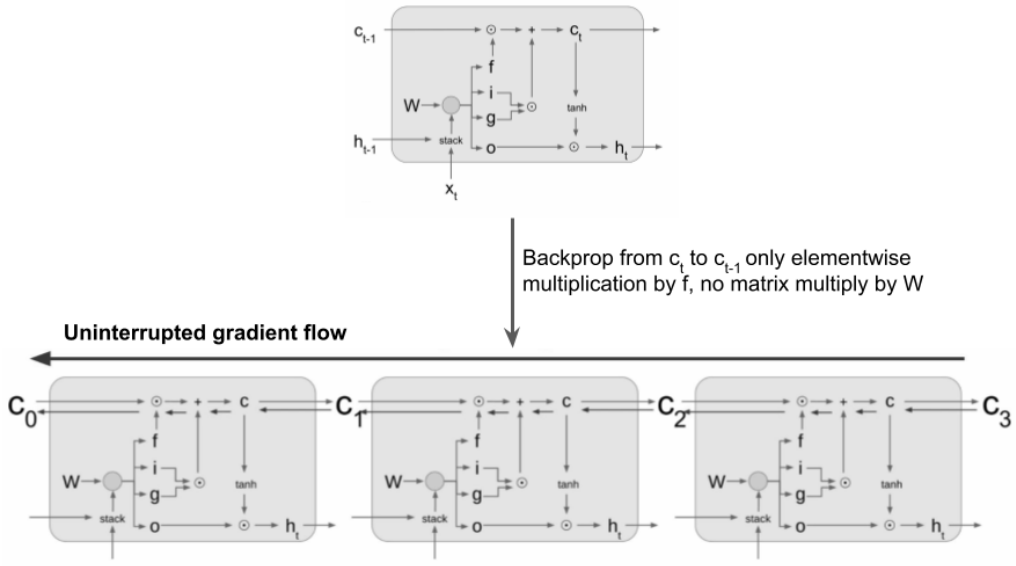
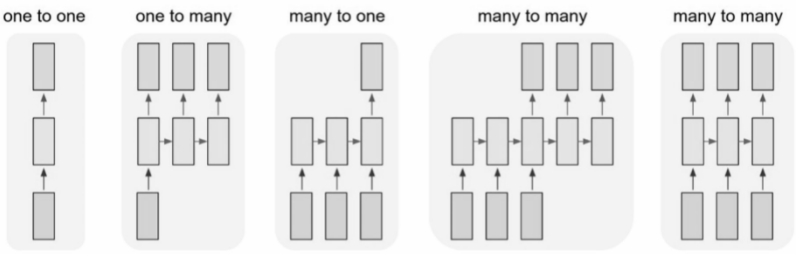


Figure A.2: LSTM.

implementation being task-specific. In this regard, tasks can be grouped into five primary categories: one to one, one to many, many to one, and many to many, as

illustrated in the following figure.



An example of a many to one task is text classification, while machine translation exemplifies the many to many category. To elucidate how these architectures are utilized and to introduce the so-called bottleneck issue, we provide an example of a machine translation task. The prevailing methods for this application, such as the seq2seq [42] and encoder/decoder architectures [43], employ two RNNs: one for encoding and one for decoding. As depicted in Figure A.3, the encoder produces a hidden state for each input token of the input sequence, according to the following equation:

$$h_t = f_w(x_t, h_{t-1})$$

The final hidden state is then used to predict the decoder's initial state, s_0, and the context vector, c. Subsequently, a starting token of the output sequence, together with the context vector and the initial state, is utilized to predict the next decoder state, s_1, which is then employed to predict the following token of the output sequence, according to this equation

$$s_t = g_U(y_{t-1}, s_{t-1}, c)$$

Unfortunately, a problem arises due to the fact that the context vector, c, is a fixed-sized vector that must encapsulate the information of all encoding steps. What if the number of steps is exceedingly high? This is where the bottleneck problem is encountered, with the context vector constituting the bottleneck.

To circumvent this problem, the concept of Attention was introduced, as elaborated in references [23, 24]. The main idea is to use a new context vector at each step of the decoder. As illustrated in Figure A.4, for each decoder state, an alignment score is computed according to the following equation:

$$e_{t,i} = f_{att}(s_{t-1}.h_i)$$

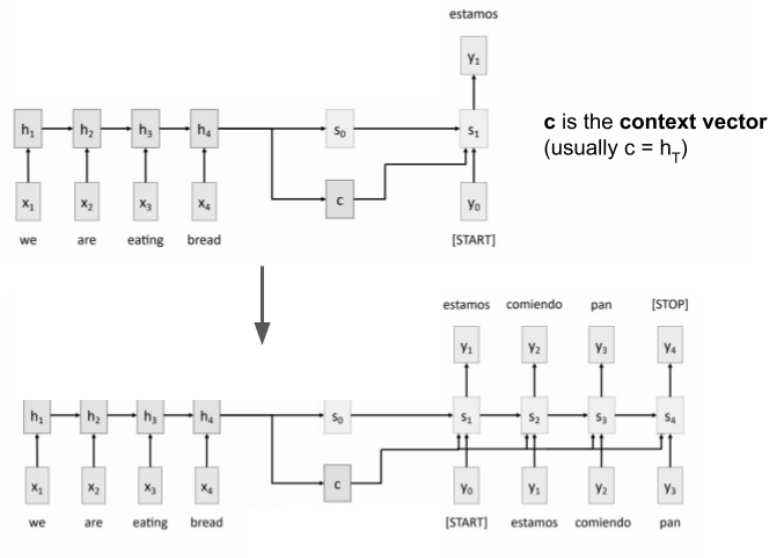


Figure A.3: Machine translation with encoder (left) and decoder (right) consising of two RNNs. The context vector c is a fixed-sized vector that bottleneck the input sequence if it is too long.

Here, f_{att} refers to a multi-layer perceptron. The score is then normalized to obtain attention weights ($0 < a_{t,i} < 1$ and $\sum_i a_{t,i} = 1$). The context vector for each step is obtained as follows:

$$c_t = \sum_i a_{t,i} h_i$$

Lastly, the context vector is used in the decoder as before, according to this equation:

$$s_t = g_U(y_{t-1}, s_{t-1}, c_t)$$

However, in contrast to the previous approach, in each timestep of the decoder, the context vector differs. This serves two purposes: first, the input sequence is not bottlenecked through a single vector, and second, at each timestep of the decoder, the context vector "looks at" different parts of the input sequence.

One final issue with RNNs is that they are difficult to parallelize because the hidden states must be computed sequentially.

In order to present a solution to this problem, a new concept needs to be introduced. Figure A.5 illustrates the transition from the attention mechanism to the self-attention mechanism. The key concept here is that the input vector and query vector are no longer separate entities, as the former is obtained simply by

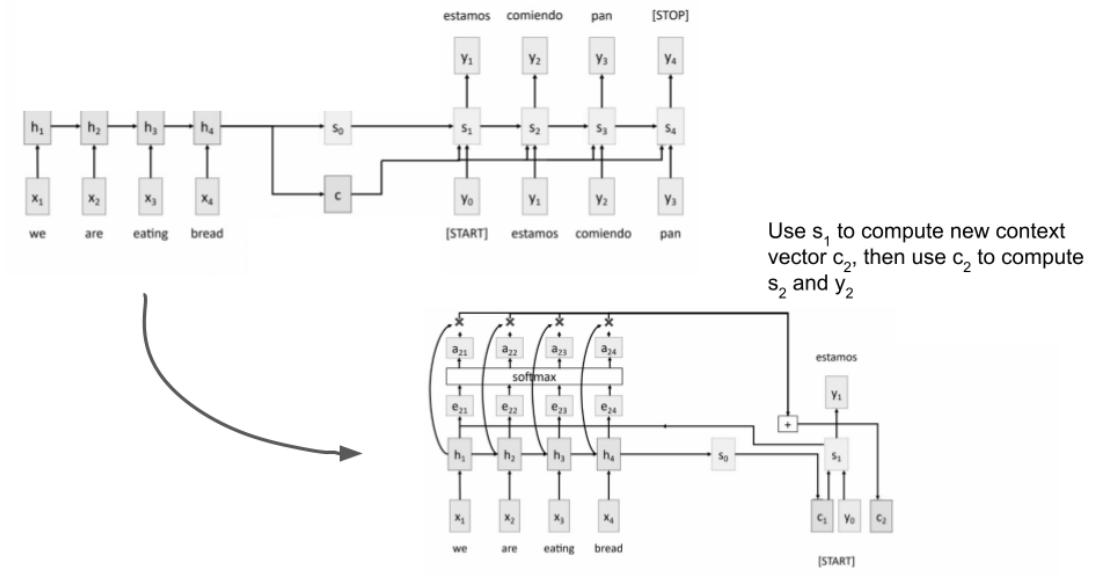


Figure A.4: From sequence to sequence with RNNs to sequence to sequence with RNNs and Attention. To better get an intuition of the role of attention weights, look at y_1: "estamos". We expect the attention weights linked to x_1: "we" and x_2: "are" (so respectively a_{11} and $a12$) to be higher than the ones linked to $x3$ and $x4$.

multiplying the input vector with a query matrix. We will refrain from delving into the intricacies of Attention, as it is a highly sophisticated, yet elegant, solution that has been extensively documented online.

The self-attention mechanism has been utilized as a fundamental block by [44] in the creation of the Transformer, which revolutionized the NLP landscape. The paper's title, "Attention is all you need," succinctly encapsulates this architecture's essence. In Figure A.6, we observe how self-attention serves as the sole interaction between vectors. It is also important to note the residual connection [45] present around each sub-layer, as it permits positional encoding to remain unaltered as it ascends to higher layers, thereby augmenting the model's performance. With the Transformer, we now have a highly scalable and parallelizable structure that is not constrained by bottlenecks.

Nevertheless, there remains one last step necessary to truly revolutionize the NLP world. BERT [4] brings the pre-training and transfer learning concepts, which are commonplace in the image vision domain, to the NLP field. Essentially composed of stacked Transformer layers, the BERT architecture provides contextual embeddings through self-attention layers during a pre-training phase on a massive corpus of text from the internet. As a result, the embedding of a word is dependent on the

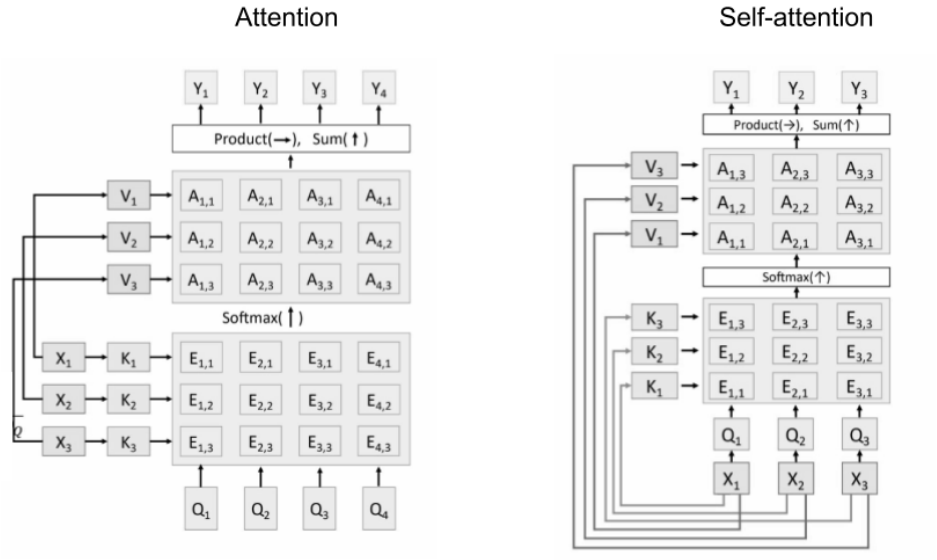


Figure A.5: From attention to self attention.

surrounding words' context. A subsequent, smaller fine-tuning phase adapts the model to a specific NLP downstream task. This concept will be further elucidated in the following section.

While transformers have been a significant breakthrough in NLP during 2018 and 2019, their high data requirements, computational power, and model size have generated controversy. To address this, researchers have created more efficient transformer models like DistilBERT [46], which can be utilized in resource-constrained environments, such as mobile devices or non-GPU servers during inference. This makes transformer-based models applicable to scenarios where resources are limited, in addition to those involving large model sizes, such as Megatron-LM [47]. In the following section, we will delve more deeply into the state-of-the-art models that use the Transformer architecture.

A.1 Transformer Architectures

In this section, the four main approaches that have built upon the transformer architecture and achieved state-of-the-art results on NLP benchmarks: BERT [4], XLNet [48], RoBERTa [49], and MPNet [5] will be examined. Each of these models has its own unique characteristics and techniques that have led to its success. The architecture, pre-training techniques, and performance of each model will be discussed in more detail.

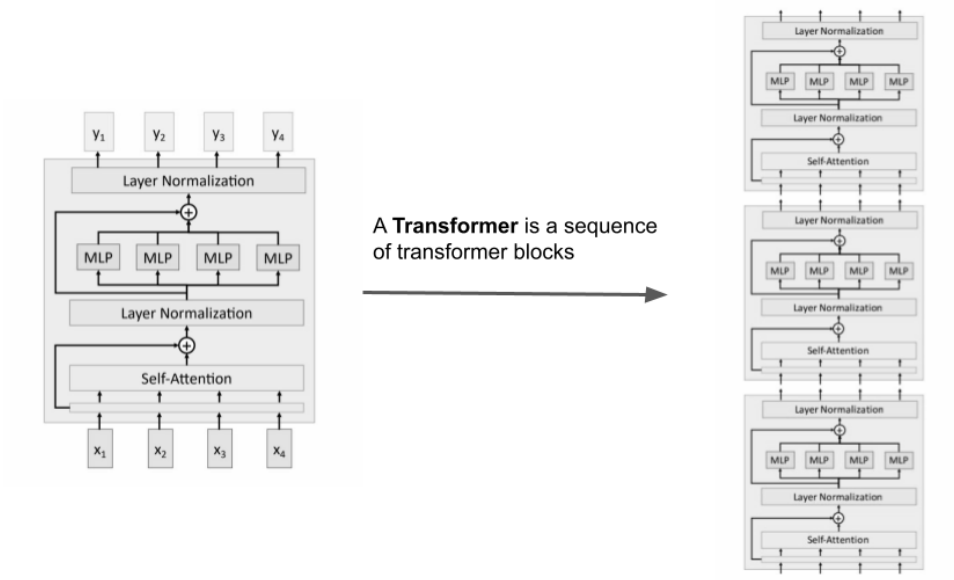


Figure A.6: The Transformer.

A.1.1 BERT

BERT [4] is an 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.

It is in its core a transformer language model as designed by [44], but contrary to other language models (e.g. OpenAI's GPT [50]) 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 [..] (example taken from [14]. 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.

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 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 overall pre-training and fine-tuning procedures are shown in Figure A.7.

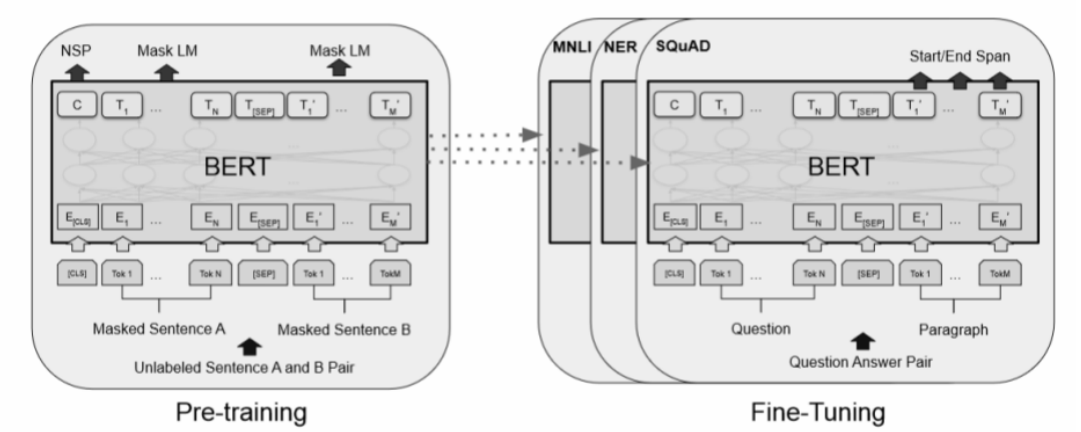


Figure A.7: Overall pre-training and fine-tuning procedures for BERT. Credits to [4]

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.

A.1.2 XLNet

BERT's bidirectional representation is achieved through a training task where the model is tasked with predicting missing tokens (designated with [MASK]) in a sentence, rather than predicting the next word. The XLNet paper [48] proposed an alternative architecture to address some of the limitations of the approach used by BERT. According to XLNet there are two major downsides of the BERT approach:

- predicting [MASK] tokens during pre-training is not representative of the downstream tasks, resulting in a discrepancy between pre-training and fine-tuning
- BERT assumes independence between the different [MASK] tokens, which may not be true in practice

To overcome these flaws, XLNet returns to the more traditional architecture of an autoregressive (AR) language model. AR models do not introduce any artificial symbols and simply learn to predict the next token in a sequence. However, unlike BERT's autoencoder approach, an AR model can only use forward context, not backward context (as can be seen in Figure A.8). To overcome this limitation, XLNet introduces a new generalized autoregressive pre-training method that allows the model to capture bidirectional context while still maintaining the advantages of an AR model. The new method is based on permutation-based training, where the model is trained to predict tokens in a sequence based on their permuted order, rather than their original order. This allows the model to learn the dependencies between all tokens, not just those that come before or after a given token, thus capturing bidirectional context.

A.8

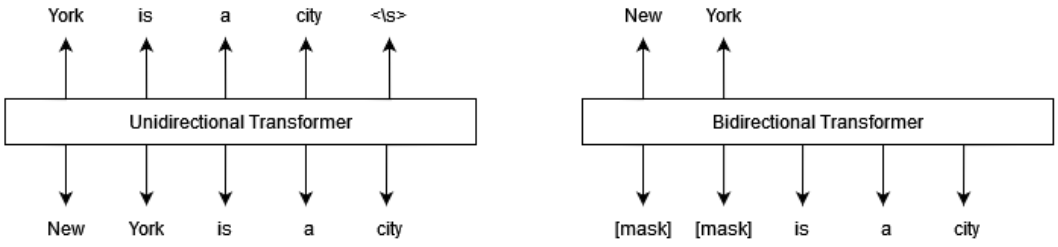


Figure A.8: The auto-regressive language modeling (left) is NOT able to model bidirectional context. The bidirectional one (right) predicts tokens that are independent of each other and the token [mask] is not used during finetuning

Assume that we have the same sentence from Figure A.8 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 given the same input sequence but with different factorization orders:

- New York is a city
- a York New is city
- city a is New York

In addition to its core contribution of permutation-based language modeling, XLNet also includes the ability to learn dependencies beyond a fixed length without disrupting temporal coherence is achieved through a segment-level recurrence mechanism and a novel positional encoding scheme. This allows the model to better handle longer sequences than the original transformer model. This improvement was originally included in the Transformer-XL paper [51].

According to the XLNet paper, it 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.

A.1.3 RoBERTa

RoBERTa [49] is a paper released at the end of July 2019 that focuses on new insights on BERT rather than proposing a new transformer approach. The authors of RoBERTa claim that BERT, without any major changes, can match or exceed every published model after it by using the right hyperparameters and enough training data. To achieve this, the authors of RoBERTa identify 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. 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 the 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 that the learning rate is increased appropriately

- removing the next sentence prediction (NSP) objective: In contrast to [4], 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

- changing 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.

A.1.4 MPNet

As shown in section A.1.2, since BERT neglects dependency among predicted tokens XLNet introduces permuted language modeling (PLM) for pre-training to address this problem. However, PLM has its own limitation: Each token can only see its preceding tokens in a permuted sequence but does not know the position information of the full sentence (e.g., the position information of future tokens in the permuted sentence) during the autoregressive pre-training, which brings discrepancy between pre-training and fine-tuning. MPNet [5] is a pre-training method that inherits the advantages of BERT and XLNet and avoids their limitations. MPNet leverages the dependency among predicted tokens through permuted language modeling (vs. MLM in BERT), and takes auxiliary position information as input to make the model see a full sentence and thus reducing the position discrepancy (vs. PLM in XLNet).

MPNet main contributions are:

- It takes the dependency among the predicted tokens into consideration through permuted language modeling and thus avoids the issue of BERT.

- It takes position information of all tokens as input to make the model see the position information of all the tokens and thus alleviates the position discrepancy of XLNet.

MPNet is pre-trained on a large-scale text corpora (over 160GB data) following the practice in [49, 48], and fine-tuned on a variety of down-streaming benchmark tasks, including GLUE, SQuAD, RACE and IMDB. Experimental results show that MPNet outperforms MLM and PLM by a large margin. Moreover, MPNet

outperforms previous well-known models BERT, XLNet and RoBERTa on GLUE dev sets under the same model setting, indicating the great potential of MPNet for language understanding.

Bibliography

[1] Aaron Smith and Monica Anderson. «Online shopping and purchasing preferences». In: *Erişim adresi: https://www. pewresearch. org/internet/2016/12/19/online-shopping-and-purchasing-preferences* (2016) (cit. on p. 1).

[2] Tongzhou Wang and Phillip Isola. «Understanding contrastive representation learning through alignment and uniformity on the hypersphere». In: *International Conference on Machine Learning*. PMLR. 2020, pp. 9929–9939 (cit. on pp. 4, 17, 18).

[3] Tianyu Gao, Xingcheng Yao, and Danqi Chen. «Simcse: Simple contrastive learning of sentence embeddings». In: *arXiv preprint arXiv:2104.08821* (2021) (cit. on pp. 4, 10, 11, 17, 41).

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. «Bert: Pre-training of deep bidirectional transformers for language understanding». In: *arXiv preprint arXiv:1810.04805* (2018) (cit. on pp. 5, 9, 47–50, 53).

[5] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. «Mpnet: Masked and permuted pre-training for language understanding». In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 16857–16867 (cit. on pp. 5, 48, 53).

[6] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. «Supervised contrastive learning». In: *Advances in neural information processing systems* 33 (2020), pp. 18661–18673 (cit. on pp. 6, 10, 41).

[7] Alessandro Magnani, Feng Liu, Min Xie, and Somnath Banerjee. «Neural product retrieval at walmart. com». In: *Companion Proceedings of The 2019 World Wide Web Conference*. 2019, pp. 367–372 (cit. on p. 7).

[8] Eliot P Brenner, Jun Zhao, Aliasgar Kutiyanawala, and Z Yan. «End-to-end neural ranking for ecommerce product search». In: *Proceedings of SIGIR eCom* 18 (2018) (cit. on p. 7).

[9] Daniel Gillick, Sayali Kulkarni, Larry Lansing, Alessandro Presta, Jason Baldridge, Eugene Ie, and Diego Garcia-Olano. «Learning dense representations for entity retrieval». In: *arXiv preprint arXiv:1909.10506* (2019) (cit. on p. 8).

[10] Nicola De Cao, Gautier Izacard, Sebastian Riedel, and Fabio Petroni. «Autoregressive entity retrieval». In: *arXiv preprint arXiv:2010.00904* (2020) (cit. on p. 8).

[11] David Savir and George J. Laurer. «The characteristics and decodability of the Universal Product Code symbol». In: *IBM Systems Journal* 14.1 (1975), pp. 16–34 (cit. on pp. 8, 42).

[12] David L Brock. «Integrating the Electronic Product Code (EPC) and the Global Trade Item Number (GTIN)». In: *White Paper available at www. autoidcenter. org/pdfs/MIT-WUTOID-WH-004. pdf* 25 (2001), pp. 2–25 (cit. on pp. 8, 14, 42).

[13] Jelena Jovanovic and Ebrahim Bagheri. «Electronic commerce meets the semantic web». In: *It Professional* 18.4 (2016), pp. 56–65 (cit. on p. 8).

[14] Ursin Brunner and Kurt Stockinger. «Entity matching with transformer architectures-a step forward in data integration». In: *23rd International Conference on Extending Database Technology, Copenhagen, 30 March-2 April 2020*. OpenProceedings. 2020 (cit. on pp. 8–10, 18, 24, 49).

[15] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. «Deep entity matching with pre-trained language models». In: *arXiv preprint arXiv:2004.00584* (2020) (cit. on pp. 8–10, 15, 19, 24).

[16] Janusz Tracz, Piotr Iwo Wójcik, Kalina Jasinska-Kobus, Riccardo Belluzzo, Robert Mroczkowski, and Ireneusz Gawlik. «BERT-based similarity learning for product matching». In: *Proceedings of Workshop on Natural Language Processing in E-Commerce*. 2020, pp. 66–75 (cit. on pp. 8, 10, 14).

[17] Ralph Peeters, Anna Primpeli, Benedikt Wichtlhuber, and Christian Bizer. «Using schema. org annotations for training and maintaining product matchers». In: *Proceedings of the 10th International Conference on Web Intelligence, Mining and Semantics*. 2020, pp. 195–204 (cit. on pp. 8, 14).

[18] Ralph Peeters and Christian Bizer. «Supervised Contrastive Learning for Product Matching». In: *arXiv preprint arXiv:2202.02098* (2022) (cit. on pp. 8, 24, 41).

[19] Juan Li, Zhicheng Dou, Yutao Zhu, Xiaochen Zuo, and Ji-Rong Wen. «Deep cross-platform product matching in e-commerce». In: *Information Retrieval Journal* 23.2 (2020), pp. 136–158 (cit. on p. 8).

[20] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. «Deep learning for entity matching: A design space exploration». In: *Proceedings of the 2018 International Conference on Management of Data*. 2018, pp. 19–34 (cit. on pp. 8, 13, 23).

[21] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. «Distributed representations of tuples for entity resolution». In: *Proceedings of the VLDB Endowment* 11.11 (2018), pp. 1454–1467 (cit. on p. 8).

[22] Jeffrey L Elman. «Finding structure in time». In: *Cognitive science* 14.2 (1990), pp. 179–211 (cit. on pp. 9, 43).

[23] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. «Neural machine translation by jointly learning to align and translate». In: *arXiv preprint arXiv:1409.0473* (2014) (cit. on pp. 9, 45).

[24] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. «Effective approaches to attention-based neural machine translation». In: *arXiv preprint arXiv:1508.04025* (2015) (cit. on pp. 9, 45).

[25] Matteo Paganelli, Francesco Del Buono, Andrea Baraldi, Francesco Guerra, et al. «Analyzing how BERT performs entity matching». In: *Proceedings of the VLDB Endowment* 15.8 (2022), pp. 1726–1738 (cit. on p. 10).

[26] Elad Hoffer and Nir Ailon. «Deep metric learning using triplet network». In: *Similarity-Based Pattern Recognition: Third International Workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015. Proceedings 3*. Springer. 2015, pp. 84–92 (cit. on pp. 10, 11).

[27] Kihyuk Sohn. «Improved deep metric learning with multi-class n-pair loss objective». In: *Advances in neural information processing systems* 29 (2016) (cit. on p. 11).

[28] Hanna Köpcke, Andreas Thor, and Erhard Rahm. «Evaluation of entity resolution approaches on real-world match problems». In: *Proceedings of the VLDB Endowment* 3.1-2 (2010), pp. 484–493 (cit. on p. 14).

[29] Pradap Venkatramanan Konda. *Magellan: Toward building entity matching management systems*. The University of Wisconsin-Madison, 2018 (cit. on p. 14).

[30] Bohan Li, Hao Zhou, Junxian He, Mingxuan Wang, Yiming Yang, and Lei Li. «On the sentence embeddings from pre-trained language models». In: *arXiv preprint arXiv:2011.05864* (2020) (cit. on pp. 17, 41).

[31] Jianlin Su, Jiarun Cao, Weijie Liu, and Yangyiwen Ou. «Whitening sentence representations for better semantics and faster retrieval». In: *arXiv preprint arXiv:2103.15316* (2021) (cit. on p. 17).

[32] Eneko Agirre, Daniel Cer, Mona Diab, and Aitor Gonzalez-Agirre. «Semeval-2012 task 6: A pilot on semantic textual similarity». In: ** SEM 2012: The First Joint Conference on Lexical and Computational Semantics–Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*. 2012, pp. 385–393 (cit. on p. 17).

[33] Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. «* SEM 2013 shared task: Semantic textual similarity». In: *Second joint conference on lexical and computational semantics (* SEM), volume 1: proceedings of the Main conference and the shared task: semantic textual similarity*. 2013, pp. 32–43 (cit. on p. 17).

[34] Eneko Agirre et al. «Semeval-2015 task 2: Semantic textual similarity, english, spanish and pilot on interpretability». In: *Proceedings of the 9th international workshop on semantic evaluation (SemEval 2015)*. 2015, pp. 252–263 (cit. on p. 17).

[35] Eneko Agirre, Carmen Banea, Daniel Cer, Mona Diab, Aitor Gonzalez Agirre, Rada Mihalcea, German Rigau Claramunt, and Janyce Wiebe. «Semeval-2016 task 1: Semantic textual similarity, monolingual and cross-lingual evaluation». In: *SemEval-2016. 10th International Workshop on Semantic Evaluation; 2016 Jun 16-17; San Diego, CA. Stroudsburg (PA): ACL; 2016. p. 497-511.* ACL (Association for Computational Linguistics). 2016 (cit. on p. 17).

[36] Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. «Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation». In: *arXiv preprint arXiv:1708.00055* (2017) (cit. on p. 17).

[37] Nils Reimers and Iryna Gurevych. «Sentence-bert: Sentence embeddings using siamese bert-networks». In: *arXiv preprint arXiv:1908.10084* (2019) (cit. on p. 19).

[38] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. «Learning long-term dependencies with gradient descent is difficult». In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166 (cit. on p. 43).

[39] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. «On the difficulty of training recurrent neural networks». In: *International conference on machine learning*. Pmlr. 2013, pp. 1310–1318 (cit. on p. 43).

[40] Sepp Hochreiter and Jürgen Schmidhuber. «Long short-term memory». In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cit. on p. 43).

[41] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. «On the properties of neural machine translation: Encoder-decoder approaches». In: *arXiv preprint arXiv:1409.1259* (2014) (cit. on p. 43).

[42] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. «Sequence to sequence learning with neural networks». In: *Advances in neural information processing systems* 27 (2014) (cit. on p. 45).

[43] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. «Learning phrase representations using RNN encoder-decoder for statistical machine translation». In: *arXiv preprint arXiv:1406.1078* (2014) (cit. on p. 45).

[44] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. «Attention is all you need». In: *Advances in neural information processing systems* 30 (2017) (cit. on pp. 47, 49).

[45] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Deep residual learning for image recognition». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778 (cit. on p. 47).

[46] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. «DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter». In: *arXiv preprint arXiv:1910.01108* (2019) (cit. on p. 48).

[47] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. «Megatron-lm: Training multi-billion parameter language models using model parallelism». In: *arXiv preprint arXiv:1909.08053* (2019) (cit. on p. 48).

[48] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. «Xlnet: Generalized autoregressive pretraining for language understanding». In: *Advances in neural information processing systems* 32 (2019) (cit. on pp. 48, 51, 53).

[49] Yinhan Liu et al. «Roberta: A robustly optimized bert pretraining approach». In: *arXiv preprint arXiv:1907.11692* (2019) (cit. on pp. 48, 52, 53).

[50] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. «Improving language understanding by generative pre-training». In: (2018) (cit. on p. 49).

[51] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. «Transformer-xl: Attentive language models beyond a fixed-length context». In: *arXiv preprint arXiv:1901.02860* (2019) (cit. on p. 52).