FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

David Nepožitek

Recommender systems for fashion outfits

Department of Software Engineering

Supervisor of the bachelor thesis:	Mgr. Ladislav Peška, Ph.D.
Study programme:	Computer Science
Study branch:	General Computer Science

Prague 2020

I would like to thank my supervisor Mgr. Ladislav Peška, Ph.D. for his continuous guidance, suggestions and advice that helped me to finish this thesis. I would also want to thank my family and friends for their support during the studies.

Title: Recommender systems for fashion outfits

Author: David Nepožitek

Department: Department of Software Engineering

Supervisor: Mgr. Ladislav Peška, Ph.D., Department of Software Engineering

Abstract: Outfit recommendation is a task of suggesting fashion products that complement a given set of garments. Traditional recommender systems rely primarily on similarities between items or users; however, that is not sufficient for a recommendation of complementary products. Thus, outfit recommendation systems use machine learning techniques to learn more subtle relations between items. In this thesis, we explore the possibility of employing recent natural language processing approaches in outfit recommendation. We propose a novel architecture based on the Transformer, and we evaluate the model on standard datasets. We show that our approach is capable of learning some relations between items. However, its performance does not exceed the state-of-the-art models.

Keywords: recommender systems, fashion, natural language processing

Contents

Introduction

Outfit recommendation is a task that can be defined as follows: find such fashion products that they match a given set of garments, and together they form a matching outfit. That can be useful for online retailers or as a tool that suggests items from personal wardrobes. Consider a user that is looking for shoes at an online shop. If they have already some products in the shopping cart, we can improve their experience by recommending shoes that complement these products.

Traditional recommendation approaches rely primarily on similarities between users or items; however, that is not sufficient for a recommendation of compatible items. Therefore, recent fashion recommender systems explore the capabilities of machine learning techniques to recognise more delicate relations between items. As fashion products are typically described only by their size and colour, the models focus mainly on visual traits in product images. The visual features can be extracted to a vector representation (embeddings) using a convolutional neural network (CNN); the particular recommendation techniques differ in the way of utilising these embeddings.

Some of the initial methods utilise explicit visual attributes (such as colours or patterns) extracted from the images [Yamaguchi et al., 2015]. However, most methods employ the vector representations directly in combination with various neural networks such as long short-term memory (LSTM) networks [Han et al., 2017] or graph neural networks [Cui et al., 2019, Cucurull et al., 2019, Singhal et al., 2020]. The state-of-the-art approaches compute compatibilities between two items in several embedding subspaces that correspond to some conditions of the input (e.g. categories of the items) [Vasileva et al., 2018, Veit et al., 2017, Tan et al., 2019, Lin et al., 2019].

In this thesis, we explore the possibility of adapting recent natural language processing (NLP) models for outfit recommendation. Specifically, we explore the capabilities of a mechanism called self-attention that is used in the Transformer proposed by Vaswani et al. [2017]. This goal is based on the intuitive similarities between a word in a sentence and a fashion product in an outfit (e.g. both a word and a fashion product may have a different meaning in different contexts). On the other hand, we have to overcome the differences such as order-independence of outfits or finite and discrete nature of vocabularies used in NLP.

The goals of this thesis are:

- to analyse the current state of outfit recommendation,
- to elaborate on the possibilities of modifying the Transformer for outfit recommendation, and
- to design and evaluate an outfit recommendation model based on the Transformer.

Organization

First, we describe the main concepts and basic methods of recommender systems in Chapter 1. In Chapter 2, we describe the state-of-the-art NLP models that are based on the self-attention mechanism. We discuss the prior work in outfit recommendation in Chapter 3. We propose our approach in Chapter 4 and describe its implementation in Chapter 5. In Chapter 6, we evaluate the model and discuss its performance. Finally, in Chapter 7, we describe our unsuccessful attempts to improve the model and share our thoughts on the future research.

1. Recommender systems

People are often forced to choose from a wide range of products without sufficient knowledge about them. Recommender systems should guide users through this process and suggest the most useful items. [Resnick and Varian, 1997] Those suggestions can help users buy a notebook best suited to their needs, find songs they are in a mood for, suggest a movie they are going to enjoy and so on.

1.1 Main Concepts

1.1.1 Problem Definition

It is commonly referred to an object that is being recommended as an *item* and a *user* denotes an individual to whom is the recommendation presented. The main principle of recommender systems is gathering feedback from users and then leveraging it in the prediction of their preferences.

The original way of defining the recommendation problem was the following. Let $\mathcal{U}$ be the set of all users and let $\mathcal{I}$ be the set of all items. Then we define function $r : \mathcal{U} \times \mathcal{I} \to \mathcal{R}$, where $\mathcal{R}$ is a totally ordered set. For a user $u \in \mathcal{U}$ and an item $i \in \mathcal{I}$, $r(u, i)$ represents the utility (sometimes referred to as rating) of the item i for the user u. The goal of the recommender system is then to find such item i_u that it has the maximum utility for a user u. The difficulty is that the system usually knows only part of the function r. That is why we define the recommendation problem as a prediction of the utility of a particular item for a particular user. In practice, when the prediction of the utility function is made, the best k items are presented to the user as the recommendation. [Adomavicius and Tuzhilin, 2005]

As the actual goal is to suggest top-k items for a particular user, sometimes the recommendation problem is considered a ranking problem. The main reason is that focusing on utility prediction may lead to suboptimal ranking performance. Consider we have an item i and an item j. Assume the real values of the utility of those two items are 3 and 5 respectively. Then let a model A predict the utilities in respective order as $(6, 4)$ and let $(1, 7)$ be the prediction of a model B. The accuracy of utility prediction of the model A is better judging by the absolute deviation from the real values. However, the model B preservers the order of the values which may be beneficial in practice. [Liu and Yang, 2008]

Recommender systems can process other data than feedback, as well. The inputs may include information about the items such as price, colour, size and other attributes. We refer to this input as to *content*. We may also utilize knowledge of the domain and constraints defined by the user. This approach is called *knowledge-based*. Other relevant information that can help us provide more accurate suggestions is called *context*. That can include time, location of the user, current season, users' profiles, etc.

1.1.2 Goals and Challenges

From the perspective of users, the main role of the recommender system is quite clear - to help them find the most useful items. In the context of the service provider, the intention behind recommender systems is usually to increase the profit of the application where the system is deployed. Respecting that, there are many secondary goals that a recommender system should try to fulfil. Some of them aim to improve the overall user experience, and others address rather the implementation side. It is important to note that every application has different needs and therefore, must focus on different properties of the system as many of them come with some trade-offs. The importance of particular goals should be figured experimentally. Some of the common pursued properties are the following:

- *Relevance* prediction accuracy is one of the keys to a well-designed recommender system. Despite that, it might be advantageous to favour some properties listed bellow at the expense of recommending slightly less relevant items.

- *Novelty:* A recommended item is novel when the user did not know about it before the recommendation. For example, a recommender system on a news server should primarily recommend articles that bring some new information to the users. On the other hand, for a song streaming service, it may not be that important as users probably want to listen to their favourite songs more than one time.

- *Trust* indicates how much confidence people place in the quality of recommendations. One of the ways how to build trust is to provide recommendations together with an explanation of why those particular items were chosen. How to leverage trust is further described by O'Donovan and Smyth [2005].

- *Diversity* of recommended items helps users to explore a wider range of items and categories. It might be undesirable to show only the top-10 predicted items as they may be very similar. Low diversity of the recommended items could cause unnecessary narrowing of choice.

- *Coverage* reflects the proportion of items known to the system that the system can recommend. Some recommender systems might be able to recommend only a small part of the available items, for example, only the rated ones.

- *Scalability* expresses the ability to grow with an increasing number of items, users and ratings. Many models try to optimize their training time, prediction time and memory requirements in order to be usable in large scales [Sarwar et al., 2002, Takács et al., 2009].

- *Serendipity* of recommendations means including items that are novel and surprisingly relevant for the user. In contrast to novelty, a serendipitous recommender suggests items that might not have been discovered by the user on their own. [Herlocker et al., 2004]

1.1.3 Feedback

Most of the recommendation systems are highly dependent on users' feedback as it is the main input of the preference prediction. By feedback, we usually mean some kind of rating of an item. Schafer et al. [2007, p. 293] divide ratings into these categories:

- *Scalar* ratings may be either numerical for example in the form of a five-star rating or ordinal such as a choice from good, bad and neutral ratings.
- *Binary* ratings signify either positive or negative preference.
- *Unary* ratings could, for example, signalize that the user visited a page with a particular item, but no other information is available.

The feedback may be explicit, meaning that users are directly asked for ratings. However, this kind of feedback requires significant efforts from users. Therefore, some recommender systems also utilize implicit feedback which is much easier to obtain [Nichols, 1998, Hu et al., 2008]. Implicit feedback is mostly gathered by the application where the recommender system is utilized. For example, on a website, it is possible to measure the time spent on a particular page, track clicks, record history of visited pages, etc. This data then acts as a proxy to users' real preferences. The difficulty that comes with implicit ratings is the need for accurate interpretation.

1.2 Basic Recommendation Methods

1.2.1 Collaborative

The idea of collaborative methods (collaborative filtering) is to predict the rating of a particular user for a particular item based on ratings of other users and items that are somehow similar to the ones being predicted. The methods are based on the assumption that users that had similar preferences in the past will have similar preferences in the future. The basic approaches also assume that users have stable preferences. In practice, this might be a problem as we observe that user preferences change in time [Koren, 2009]. The only input of pure collaborative models is a matrix of user-item ratings recorded by the system.

The advantage of collaborative approaches is that they do not need any additional information about items such as descriptions or attributes. In order to make accurate recommendations for a particular user, collaborative methods need some initial ratings from this user. In other words, they are unable to recommend items for a new user. This issue is usually called a user *cold start* problem. Collaborative methods also struggle to recommend new items. As the only input is user-item interactions, these methods require a sufficient amount of ratings of an item to recommend this item reliably.

Collaborative methods are often divided into *neighbourhood-based* (memory-based or heuristic-based) and *model-based*. Neighbourhood-based models exploit ratings of similar items or users and use the rating matrix directly in the prediction of unknown ratings. On the contrary, model-based approaches utilize user feedback to build and maintain some kind of learned model. And then only this model is used for the prediction itself. Methods from machine learning and data mining such as neural networks [He et al., 2017], autoencoders [Sedhain et al., 2015] or decision trees can be used in model-based approaches.

Neighbourhood-based collaborative filtering

We distinguish two categories of neighbourhood-based methods - user-based and item-based. The idea of user-based neighbourhood methods is to infer the rating of a user u for an item i from ratings of users that rated the item i and rated other items similarly as user u. We refer to those users with similar ratings as to neighbours. Item-based collaborative approaches predict the rating of user u for the item i from ratings from user u given to items similar to i. By similarity in the context of collaborative methods, we mean similarity between rows or columns of the input matrix.

Neighbourhood-based methods are convenient because of their simple implementation. As the approach is intuitive, the recommendations are usually easily interpretable. The problem of pure neighbourhood-based models is that they do not scale well. One of the reasons is that they require the rating matrix to make predictions (thus sometimes called memory-based models). Also, they have quadratic time complexity either in the number of items or in the number of users.

As this approach stayed at the begging of recommender systems and still is one of the most successful [Su and Khoshgoftaar, 2009, Resnick et al., 1994], let us present an example of the user-based version. Let $\mathcal{U} = \{u_1, \ldots, u_m\}$ be the set of users and let $\mathcal{I} = \{u_1, \ldots, u_n\}$ denote the set of items. For the ratings matrix $R \in \{1, \ldots, 5\}^{m \times n}$, $r_{i,j}$ is the rating of the user i for the item j. The set of k nearest neighbours of the user i will be denoted by $\mathcal{N}_k(i)$. The predicted rating $\hat{r}_{a,b}$ is computed as an aggregation of ratings from the users most similar to a that have rated the item b. Some examples of the aggregation functions are the following [Adomavicius and Tuzhilin, 2005]:

$$\hat{r}_{a,b} = \frac{1}{|\mathcal{N}_k(a)|} \sum_{n \in \mathcal{N}_k(i)} r_{n,b} \tag{1.1}$$

$$\hat{r}_{a,b} = \frac{1}{|\mathcal{N}_k(a)|} \sum_{n \in \mathcal{N}_k(i)} \mathrm{sim}(a, n) \cdot r_{n,b} \tag{1.2}$$

$$\hat{r}_{a,b} = \overline{r_a} + \frac{1}{|\mathcal{N}_k(a)|} \sum_{n \in \mathcal{N}_k(i)} \mathrm{sim}(a, n) \cdot (r_{n,b} - \overline{r_n}) \tag{1.3}$$

where $\overline{r_i}$ denotes the average rating of the user i for all items rated by this user. And sim stands for a similarity function of two vectors of ratings. Common similarity function are the Pearson correlation and cosine similarity [Su and Khoshgoftaar, 2009]. The most simple case (1.1) is computed just as the average of ratings from the neighbours. A weighted sum as in (1.2) can be introduced to reflect the degree of similarity of particular users. These two functions do not take into consideration that different users might have different rating behaviour (e. g. some of them may rate all items with higher values than other users). For that reason the example (1.3) uses deviations from users' average ratings instead of the ratings themselves.

Model-based collaborative filtering

Model-based collaborative filtering takes advantage of a training phase when it builds the model used for prediction. During the prediction, the original rating matrix is not needed. Instead, only the learned model is used. That solves some problems with the basic neighbourhood models. Namely, the learned model is usually much smaller than the rating matrix. Therefore, model-based systems are usually faster.

Originally, the methods included approaches such as Bayesian networks or cluster models [Breese et al., 1998, Ungar and Foster, 1998]. One of the most popular approaches is matrix factorization. The results of the Netflix Price competition showed that matrix factorization performs better than standard neighbourhood-based models [Koren et al., 2009]. The main principle of matrix factorization is to characterize both user and items in shared latent space. Those latent factors are derived from the rating matrix. The model then recommends those items whose latent factors are the most similar to those of the user. More on matrix factorization can be learned from Koren et al. [2009]. Recently, neural networks and deep learning are used to improve the performance of matrix factorization [Dziugaite and Roy, 2015, Xue et al., 2017]. Some model-based approaches use classification models such as support vector machines [Zhang and Iyengar, 2002] or neural networks. The incompleteness of the rating matrix cause problems to some classifiers, because it is difficult to adjust the models to handle missing values. Fortunately, it is not a problem for unary ratings as the missing value can be determined. Other modern approaches include autoencoder-based methods [Sedhain et al., 2015, Liang et al., 2018] or graph neural networks [Sun et al., 2019].

1.2.2 Content-Based

Content-based recommender systems try to find items similar to those that the target user liked in the past. Particularly, the goal is to match a *user profile* with attributes of items. The profile is constructed from the attributes of items rated by the user in the past. The important aspect of content-based systems is that they typically focus only on ratings of one particular user and do not consider any kind of community data.

Content-based systems share the same high-level architecture. We distinguish three components of the recommendation process:

- *Content feature extraction:* The items may have arbitrary types of attributes based on the domain. The attributes can consist of textual data, images, numerical values, audio and so on. The raw data is typically not suitable for further manipulation. That is why it is needed to be transformed into a simpler representation. During this stage, the system can, for example, extract visual features from images or convert text into vectors.

- *Profile learning:* In order to recommend relevant items to a particular user, the user profile must be built. This stage utilizes recorded explicit and implicit feedback of this particular user together with the attributes of the items. For example, Liu et al. [2010] predicts users' news interest from their clicks using a Bayesian network.

- *Filtering and recommendation:* This component takes item representations and a user profile as its input. It suggests items that match the user profile of the target user. The match can be computed, for example, using a similarity metric. This component is the only one that has to act at the time of the recommendation; for that reason, it must be effective.

Content-based methods are able to recommend items that are not rated by any user. That is a significant advantage over collaborative approaches as those have a problem to handle both new users and new items. Content-based systems only need users to rate some items or initialize their profile in some other way. Also, it is possible to explain the recommendations based on item attributes. That can increase trust in the system and its transparency.

Nonetheless, content-based models have also some disadvantages. One of the main problems is called overspecialization. It means that the recommender system suggests items that are highly similar to those already rated. This can reduce the overall usefulness of the recommendations, and it greatly reduces the serendipity of the system. Another problem is the already mentioned inability to recommend items for new users. A sufficient number of ratings must be collected in order to understand user preferences and to make reliable suggestions.

1.2.3 Knowledge-Based

In some situations, knowledge-based methods help us to overcome problems of other approaches. The main characteristic of knowledge-based methods is that they allow users to specify their needs. So no historical data is needed. On the other hand, knowledge-based approaches heavily rely on item attributes. Especially in domains where items are not bought on a regular basis, neither collaborative-based nor content-based approaches might be applicable. As discussed earlier, both of these approaches require a significant amount of ratings in order to work correctly. In addition, these methods may have problems with items that are complex such as cars, real estates, computers and so on. All of these items are represented by a great number of various attributes, so it can be challenging to infer user preferences solely from ratings. One of the reasons is that it may be challenging to find similar items due to their complexity. Also, users might want to specify some attributes explicitly. In addition, in some domains such as computers, the role of time spans of ratings is significant. Ratings of an old computer can be completely irrelevant for current recommendations.

We distinguish knowledge-based recommender systems by the form of user input to the system. *Constraint-based recommender systems* let users explicitly define constraints on item attributes. On the other hand, in *case-based recommender systems*, users specify their preferences relatively to some anchor items.

Constraint-based recommender systems

The formalism of constraint satisfaction problem [Tsang, 1993] is usually used when describing and designing a constraint-based recommender system [Felfernig and Burke, 2008]. The problem can be defined by a set of variables V, a set of domains D of these variables and a set of constraints C which declares possible combinations of values assigned to the variables. The solution to this problem is all possible instantiations of the variables such that all the constraints are satisfied. We define:

- *Customer properties* V_C is a set of variables that serves for all possible customer requirements. For example, one of the properties can be a “maximum price", and its instantiation is the actual upper bound of price defined by the customer.
- *Product properties* V_{PROD} denotes the properties of products such as price, colour, or weight.
- *Compatibility constraints* C_R represent some shared, immutable restrictions on possible instantiations of customer properties. For example, it is not possible to require a laptop that is both intended for gaming and is lightweight.
- *Filter constraints* C_F define relationships between customer requirements and product properties. An example might be that if a user requires a gaming laptop, only laptops with dedicated graphics card are shown.
- *Product constraints* C_{PROD} represents all available products. One product can be described by a conjunction of all its properties. A disjunction of these conjunctions can define the whole available product assortment.
- *Customer constraints* C_C is a set of unary constraints that assign actual values to the customer properties.

With these definitions we set $V = (V_C \cup V_{PROD})$ and $C = (C_R \cup C_F \cup C_{PROD} \cup C_C)$. A standard constraint solver can then solve this task, and the result contains all possible instantiations of the variables. With those instantiations, the model can recommend appropriate items.

Case-based recommender systems

In case-based systems, users describe their preferences with a help of some anchor items (cases). Firstly, users specify some initial requirements on the attributes or select an initial anchor item to start. In both cases, items most similar to the initial query are retrieved. Then, a repeated process of *critiquing* begins. Users select one or more items from the previous step and describe which attributes should change and how should they change. Based on these adjustments, a new set of items is suggested by the system. This interactive process continues until an item with the desired properties is found.

Domain-specific similarity metrics must be defined for the attributes; otherwise, this method will not work. The advantage of this method is the guidance of users through the process. This approach can help users to express their requirements even in a domain with complex item attributes. On the other hand, the prerequisites are that users know what they want, and they are capable of expressing it using item attributes. Also, this approach struggles in situations where more items with mutually exclusive qualities match a user’s preference. For example, a user might want a laptop with a powerful graphics card. However, a laptop with worse graphics card but fast storage and powerful processing unit may also satisfy their needs.

1.2.4 Hybridization

All three basic classes of recommender systems utilize a different kind of input data and approach the recommendation problem differently. Consequently, they have diverse strengths and flaws. For example, collaborative filtering suffers from cold start problems, and content-based approaches require well-defined item attributes. The goal of hybrid recommender systems is to combine these approaches in order to exploit more types of data, overcome some problems and combine the advantages of these fundamental approaches.

There exist various strategies on how to build a hybrid recommender. Burke [2007] characterizes seven types of hybridization strategies. Let us describe three general categories of hybridization design: monolithic, parallel and pipeline.

Monolithic design contains only one main recommendation component which utilizes more than one recommendation paradigms, as shown in Figure 1.1. The component can process several types of input data. It may group items based on an item attribute and gain additional information from ratings given to this group by a particular user. The system can, for instance, distinguish a user that likes fantasy films and recommend some new films that other fantasy fans liked. Burke [2007] refers to this strategy as to *feature combination*. Work from Zanker and Jessenitschnig [2009] may serve as an example of this approach. The second type of monolithic designs is called *feature augmentation*. In feature augmentations models, output features of secondary recommender components are used as the input of the main recommender. This approach can improve both the quality and quantity of input data for the primary recommender. Feature augmentation is used by Melville et al. [2002], who shows how to incorporate content-based data into collaborative filtering.

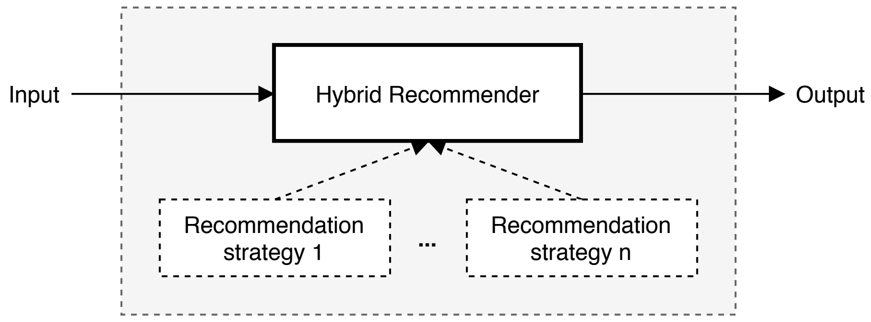

Figure 1.1: Monolithic hybrid recommender system (adapted from [Jannach, 2011, p. 128])

Parallel design uses several recommender systems side by side. Each system rates the items or produces a list of recommendations. The outputs of all those components are then aggregated. This hybridization strategy is the least invasive as it uses the recommendation components as black-boxes. All the hybridization is done in a separate component of the system as depicted in Figure 1.2. Burke [2007] further divides this class into categories distinguished by the type of the aggregation into *weighted*, *switching* and *mixed*.

Pipeline design consists of more than one successive components. The output of each component serves as an input to its successor, as shown in Figure 1.3. Optionally, each stage can also use the original input of the system. The compo-

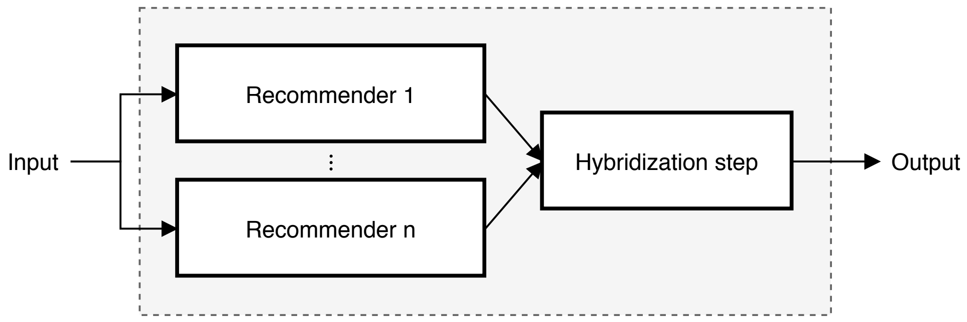

Figure 1.2: Parallelized hybrid recommender system (adapted from [Jannach, 2011, p. 129])

nents may simply refine a recommendation list provided by its predecessor. Burke [2007] calls this variation a *cascade hybrid*. The other variation is named *meta-level hybrid*. In a meta-level hybrid system, one recommender builds a model that is later used by the next component. In contrast to data augmentation, the input model of the recommender component is solely the output of the previous stage.

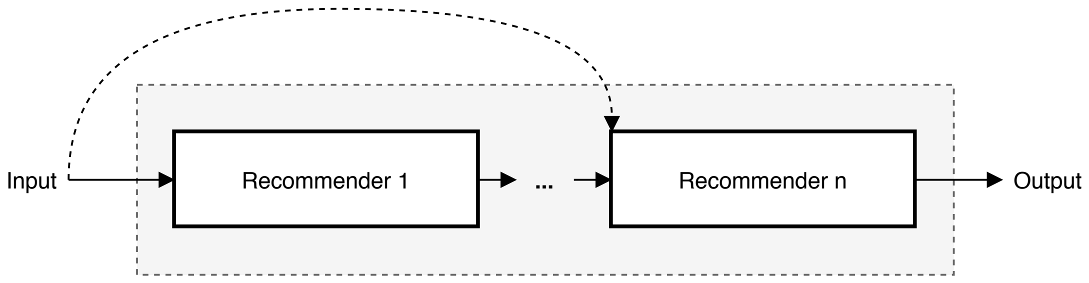

Figure 1.3: Pipelined hybrid recommender system (adapted from [Jannach, 2011, p. 129])

2. Natural Language Processing

Natural language processing (NLP) is a field concerned with computational techniques for analysis and representation of human language [Hirschberg and Manning, 2015]. NLP employs knowledge primarily from computer science and linguistics to solve a broad range of tasks, for example, related to syntax, semantics or speech. Let us name some of them: transforming words to their root form, parsing a sentence according to a grammar, automatic translation of a text, natural language generation, question answering or sentiment analysis. In this chapter, we are going to focus primarily on encoder-decoder models that are the base of our outfit recommendation approach.

2.1 Attention and Transformer-based Models

When training a model to solve NLP tasks, it is often required to generate some sort of general representations that can be later used for the particular task. This problem was historically approached by word embedding methods such as Word2Vec [Mikolov et al., 2013] or Glove [Pennington et al., 2014]. These methods learn a mapping from words to vectors that is global, meaning that the representation of one particular word is the same regardless of its context. However, some words have more than one meaning, and this approach is not able to capture the difference. That is why contextualized word embeddings were introduced.

Originally, the methods typically employed recurrent neural networks (RNNs) or convolutional neural networks (CNNs) to incorporate the context into the embeddings [McCann et al., 2017, Peters et al., 2017, 2018]. Vaswani et al. [2017] proposed the Transformer, which is an encoder-decoder model relying solely on the mechanism called attention. Many models based on the Transformer proved that this approach is suitable for various NLP tasks [Radford, 2018, Devlin et al., 2019, Raffel et al., 2019].

In this section, we elaborate on the theory behind the Transformer model. Then we describe one particular modification of the Transformer, BERT [Devlin et al., 2019], which was one of the main inspirations of our approach.

2.1.1 Encoder-Decoder Models

Encoder-decoder is an architecture that is used to solve various NLP tasks, such as machine translation, summarization or question answering. It was introduced to the NLP field by Sutskever et al. [2014], Cho et al. [2014] and Kalchbrenner and Blunsom [2013] who employed this architecture for machine translation. However, the general architecture can be used to solve arbitrary sequence-to-sequence task. That is a task where both input and output are sequences of variable lengths.

The architecture has two main components: encoder and decoder. The encoder processes the input and creates a fixed-length representation (context) of the input sequence. Then, the decoder uses the context to generate the target sequence. The decoder usually acts auto-regressively. That is, each output of the decoder is the input to the next step of the decoder. RNNs or CNNs can

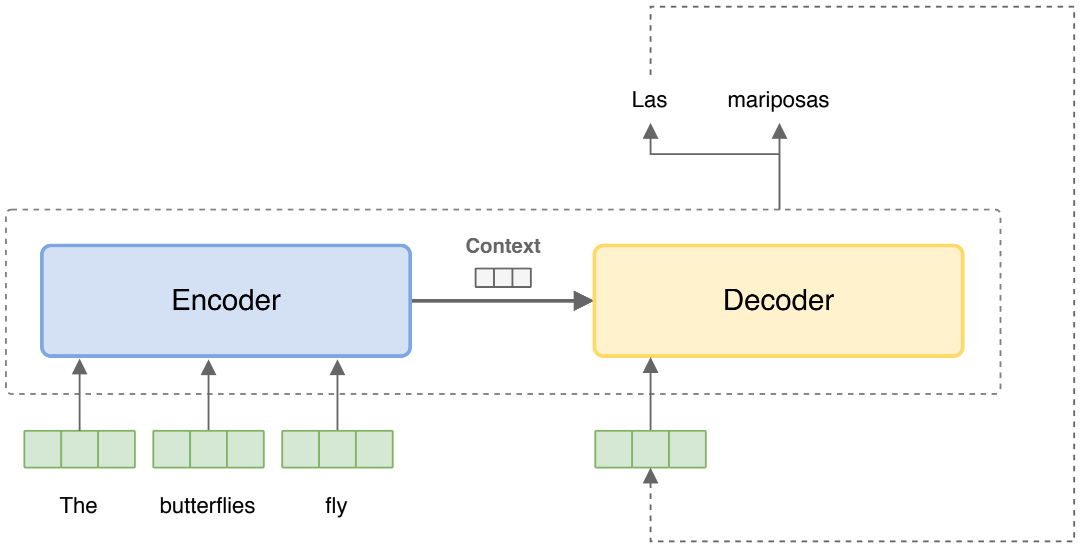

Figure 2.1: An example of an encoder-decoder model used for translation from English to Spanish. The figure depicts a moment of generating the word “mariposas”.

realize both components. Typically, each component consists of a stack of several identical blocks.

Let us elaborate on machine translation as an example of an encoder-decoder application. The task is to translate a sentence from a source language into a target language. The input sentence is first transformed into a sequence of word embeddings. This sequence is given to the encoder. The encoder computes the context and passes it to the decoder. Finally, the decoder generates the target sequence one word per step, as shown in Figure 2.1.

2.1.2 Attention

Attention is a mechanism that aims to eliminate the bottleneck of the encoder-decoder architecture caused by the context vector. Traditionally, encoder and decoder consisted of specialized RNNs such as LSTM networks [Hochreiter and Schmidhuber, 1997] or gated recurrent unit (GRU) networks [Cho et al., 2014]. RNNs take two inputs: a hidden state from the previous step and the actual input vector. The last hidden state of the encoder recurrent network was originally used as a context vector. It turned out that this part is a bottleneck of the architecture because the model was not able to capture the contextual information properly. That is, some relevant information for a particular decoding step was not encoded in the context. Bahdanau et al. [2015] address this problem with a mechanism called attention that helps to prioritize the relevant contextual information.

With the attention mechanism, instead of only one vector representing a whole sequence, a vector for every item of the input sequence is passed to the decoder. At each step of the decoder, a weighted average of the encoded vectors is used as a contextual representation of the input sequence. Formally, let i be the position of a word currently generated by the decoder. Let n be the length of the input sequence and h_j be the j-th output of the encoder (representation of the j-th item of the input sequence). Then the context vector used in the step i is c_i:

$$c_i = \sum_{j=1}^{n} \alpha_{ij} h_j \tag{2.1}$$

where α_{ij} is a weight assigned to h_j for i-th decoding step. Bahdanau et al. [2015] compute α_{ij} as follows:

$$\alpha_{ij} = \frac{\exp(a(s_{i-1}, h_j))}{\sum_{k=1}^{n} \exp(a(s_{i-1}, h_k))} \tag{2.2}$$

where a is a scoring model realized by a feed-forward neural network. The score is computed from s_{i-1}, which is the previous hidden state of the decoder, and the corresponding encoder representation h_k. In other words, the weights α_{i*} are computed as a softmax function over scores obtained from the previous hidden state of the decoder and outputs from the encoder.

Attention was defined by Vaswani et al. [2017] in a more general manner as a function from a query and a set of key-value pairs to an output. Where query, keys, values and output are all vectors. The output is computed as a weighted average of the values. A weight of a particular value is obtained from a compatibility function of the query and the key corresponding to that value. Following the definition from Bahdanau et al. [2015], both the values and keys correspond to encoder representations $h_j, j \in \{1, \dots, n\}$ and the query of the i-th step is represented by the vector s_{i-1}. Finally, the value of the compatibility function of the i-th query and the j-th key corresponds to α_{ij}.

2.1.3 Transformer

Vaswani et al. [2017] proposed a model called Transformer as an alternative to RNN models which were widely used in the field of machine learning NLP at that time. The Transformer is based primarily on the attention mechanism and does not contain recurrent neural network nor convolution. The original model was designated for machine translation, but many models based on the Transformer architecture demonstrated its capabilities to perform well in other NLP tasks as well [Raffel et al., 2019, Devlin et al., 2019, Radford, 2018].

Transformer's Attention

A particular attention mechanism called scaled dot-product attention is a vital part of the Transformer. Both the queries and keys are vectors of dimension d_k, and the values have dimension d_v. To calculate the compatibility of a query and keys, we first calculate a dot product between the query and all the keys. Then we divide the results by $\sqrt{d_k}$. And finally, to get the weights, we apply a softmax function to the results. As the attention can be computed for all the queries in parallel, we can compute the outputs as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \tag{2.3}$$

where Q, K and V are matrices of queries, keys and values respectively.

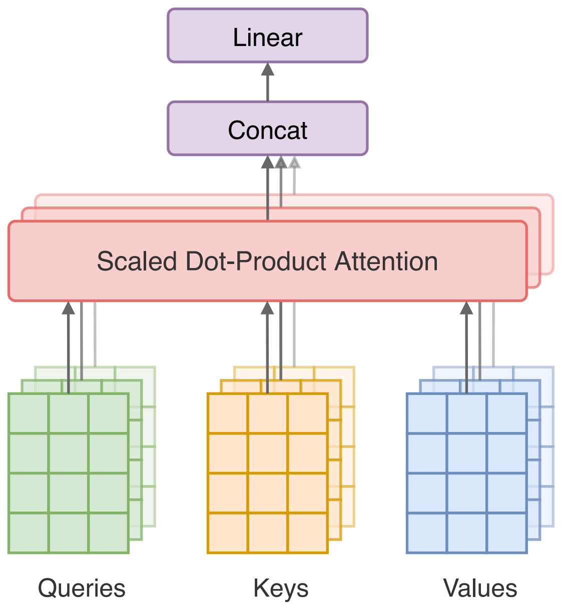

Figure 2.2: A diagram of the multi-head attention mechanism

Vaswani et al. [2017] decided to use dot-product attention as it can be optimized better than additive attention which is the alternative standard attention function. The problem of dot product attention is that it has worse performance than the additive version when the dimensions of key and query vectors get bigger [Britz et al., 2017]. Vaswani et al. [2017] assume that large values of the dot product cause the underperformance. The scaling by $\frac{1}{\sqrt{d_k}}$ was introduced to address this issue.

Instead of using the attention mechanism as is, Vaswani et al. [2017] linearly project queries, keys and values into several subspaces and perform the attention function in these subspaces. They call this technique multi-head attention where a head refers to one layer of the attention mechanism. The motivation of multi-head attention is to allow the model to attend information from different representation subspaces. Let h be the number of learned projections. The attention mechanism is applied h times simultaneously, and the resulting values are concatenated. Finally, the concatenated values are also projected. The result of this projection yields the final values. Multi-head attention is depicted in Figure 2.2.

The Transformer employs the attention mechanism in a form of self-attention in particular parts of the model. That is, all the queries, keys and values of the particular layer are computed from one input sequence. This technique allows creating contextualized word embeddings. That was typically done with RNNs in other models. Self-attention was chosen instead primarily because of its computational performance and ability to capture long-range dependencies as described in more detail by Vaswani et al. [2017].

Model

The Transformer follows the standard encoder-decoder architecture, as shown in Figure 2.3. The encoder consists of N identical encoder blocks. Each block contains two sublayers: a multi-head self-attention layer and a feed-forward network. Layer normalization [Ba et al., 2016] is applied on each sublayer, and a residual

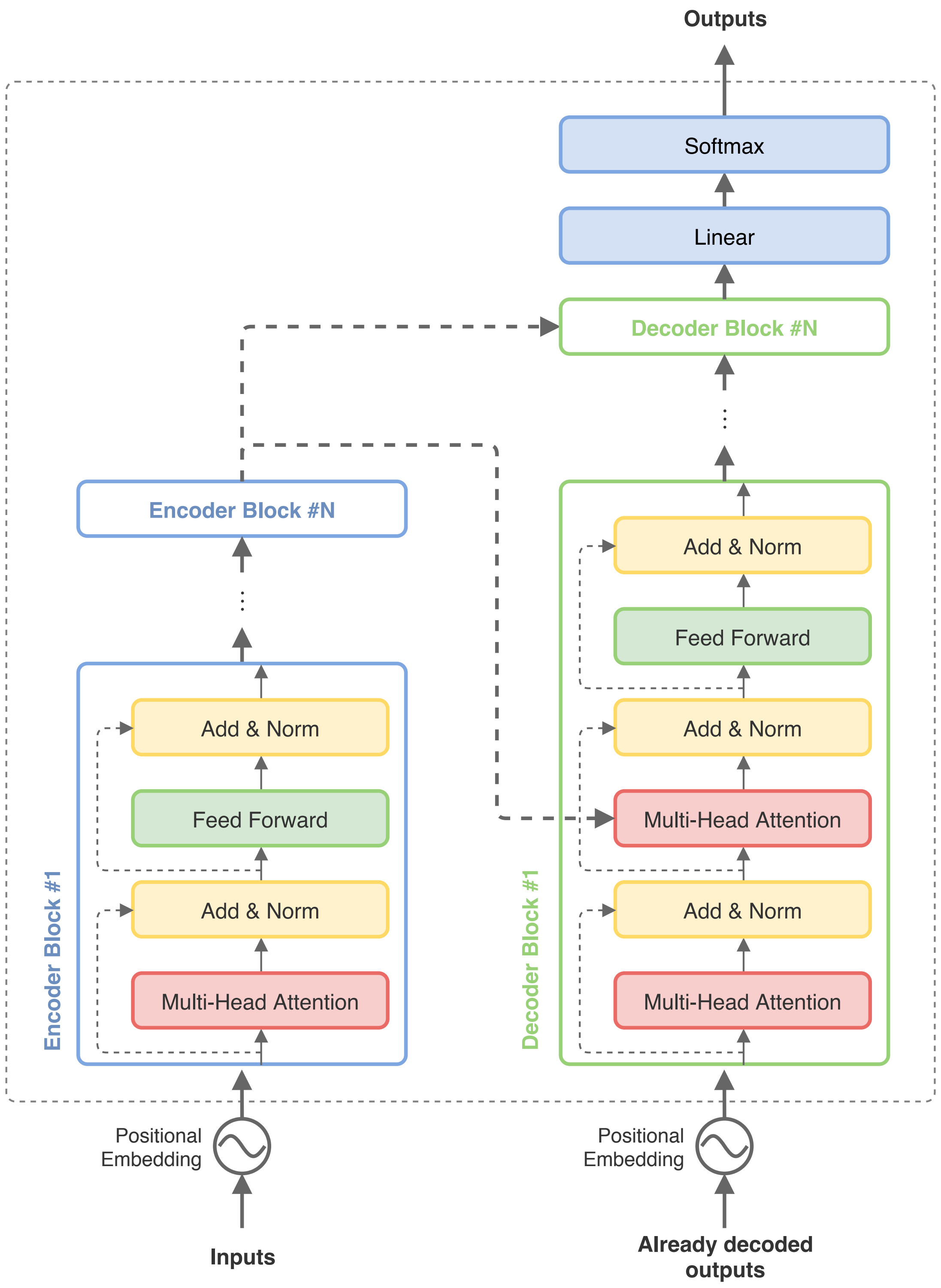

Figure 2.3: A high-level view of the Transformer

connection [He et al., 2016] adds the input of a sublayer to its output.

The decoder consists of N identical blocks as well. Encoder and decoder blocks are very similar. The only difference of a decoder block is that after the self-attention layer, there is one more attention layer which uses keys and values from the encoder. The output of the decoder is linearly projected to a vector whose elements correspond to words in the vocabulary. This vector represents non-normalized predictions of the model and it is usually called *logits*. Finally, a softmax function is applied on the logits, and one output word is chosen. The decoder is autoregressive, as usual. That is, the output words from previous steps are embedded again and passed to the decoder to generate the next word.

Since the attention mechanism is order-independent, it is necessary to provide some additional positional information. The positional encodings may be fixed or learned and relative or absolute [Raffel et al., 2019, Shaw et al., 2018, Gehring et al., 2017]. The original model adds absolute sinusoidal positional encodings to the input embeddings before given to the encoder or decoder. For the exact formula and its reasoning, we refer to Vaswani et al. [2017].

2.1.4 Bert

Many recent NLP models take advantage of pretraining to understand a text in some kind of general manner [Howard and Ruder, 2018, Radford, 2018, Yang et al., 2019, Radford et al., 2019, Raffel et al., 2019]. The knowledge learned during pretraining is meant to be transferred to particular downstream tasks. The transfer can be done by fine-tuning the model [Devlin et al., 2019, Radford, 2018] or using its pretrained features in a task-specific architectures [Peters et al., 2018]. Bidirectional Encoder Representations from Transformers (BERT) [Devlin et al., 2019] is one of the models that employ primarily the fine-tuning strategy. BERT consists of the encoder stack from the Transformer. From the architectural point of view, it does not differ from the original design. The main contribution is its approach to pretraining and fine-tuning. Both phases operate similarly and on the same architecture.

The input sequence always starts with a special classification token. The output from this token is used as a representation of the whole sequence. This representation is useful for classification tasks such as sentiment analysis. The actual textual input is embedded using sub-word embeddings [Wu et al., 2016]. Some tasks require processing two sentences with a different purpose. In this context, we use the word *sentence* to describe an arbitrary span of contiguous text. An example of this task may be question answering where it is needed to process a question and answer. In BERT, this is achieved by processing the pair of sentences together. The sentences are separated by a special separation token. Furthermore, a learned sentence embeddings are added to the subword input embeddings to distinguish the sentences. Because BERT employs self-attention, it is order-independent. That is why positional encodings are added to the input, as shown in Figure 2.4.

BERT uses two unsupervised tasks for pretraining: masked language modelling and next sentence prediction (NSP). In masked language modelling, a certain percentage of input tokens is replaced with a special mask token. The goal is to predict the original tokens. The output from the masked token is projected

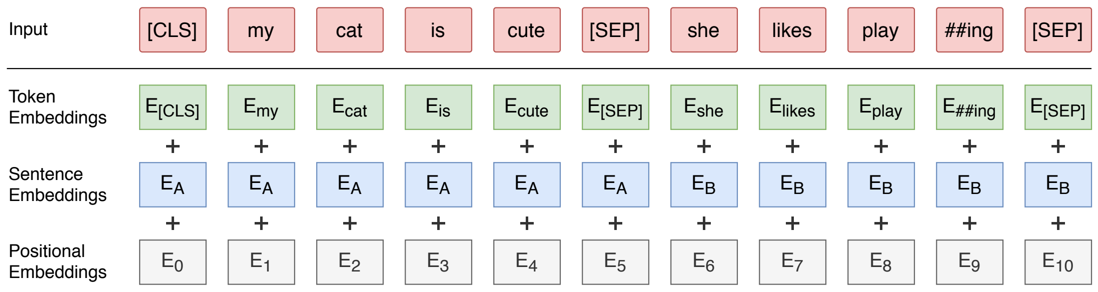
Input
[CLS]
my
cat
is
cute
[SEP]
she
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{cat} E_{is} E_{cute} $E_{[SEP]}$ E_{she} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Sentence Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Positional Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 2.4: BERT input sequence embedding. Each input item is created from three components: the token embedding, the sentence embedding and the positional embedding. (Adapted from Devlin et al. [2019].)

into the logits vector of the vocabulary. A softmax function is applied on the vector, and a token is predicted. This way, the model uses the whole context of the masked word for prediction jointly. NSP focuses on learning to capture relations between sentences. This can be useful for such tasks as question answering. For NSP, pairs of sentences are generated from a corpus. 50% of the pairs are actual consecutive sentences, and other pairs are sentences randomly sampled from the corpus. The goal is to recognize the consecutive sentences. The output from the classification token is used to solve this task.

When fine-tuning BERT, the same architecture as in pretraining is used and all the parameters of the model are trained. According to the particular downstream task, single sentences or a sentence pairs are fed into the model. For classification tasks such as sentiment analysis, the encoded classification token is passed to an output layer for classification.

BERT outperformed all the prior models and became a baseline in many NLP tasks [Wang et al., 2019a]. Various modifications of BERT were designed after its revelation, and they performed even better than the original model [Liu et al., 2019, Wang et al., 2019b]. Some variations also aimed to make BERT more effective in terms of parameters count and inference time [Sanh et al., 2019, Jiao et al., 2019].

3. Related Work

Outfit recommendation is a task that can be defined as follows: find such fashion products that they match a given set of garments, and together they form a matching outfit. This problem was approached in many various ways, but the key is exploiting visual features from images using computer vision techniques. Particular approaches differ in the way of extraction of these features and the process after the extraction. Some models are based on the extraction of visual attributes such as colours or patterns [Chen et al., 2012, 2015, Yamaguchi et al., 2015, Di et al., 2013, Kim et al., 2016]. Veit et al. [2015] and McAuley et al. [2015] learn compatibility of fashion product pairs based on co-purchases or co-views of the products. It is also possible to predict the compatibility of a set of items. Among other techniques, embedding into several embedding subspaces [Vasileva et al., 2018, Veit et al., 2017, Tan et al., 2019, Lin et al., 2019], graph neural networks [Cucurull et al., 2019, Cui et al., 2019, Singhal et al., 2020] and recurrent neural networks [Han et al., 2017] are used for fashion outfit recommendation.

3.1 Visual Attributes

It is possible to extract some visual attributes (e.g. colours, patterns or category) from the images of fashion items [Chen et al., 2012, 2015, Yamaguchi et al., 2015, Di et al., 2013, Kim et al., 2016]. A common approach is to use a convolutional neural network for this purpose. Yamaguchi et al. [2015] employ these attributes for learning compatibility of garments. Kim et al. [2016] and Di et al. [2013] show how the attributes can be used for retrieval. Kim et al. [2016] use the extracted visual features to build an indexing scheme. Di et al. [2013] propose a fashion retrieval framework that allows users to search for fashion products based on a provided image and selected attributes.

3.2 Recommendation of Complementary Items

Many models aim to learn compatibility and relationships between fashion items [Han et al., 2017, Vasileva et al., 2018, Veit et al., 2017, Lin et al., 2019]. The motivation is to recommend items that fit into a partly created outfit. This approach is based on the assumption that users want to buy compatible products or products that form a matching outfit. This approach differs from the standard recommendation techniques as those are based primarily on similarities between items or users.

Some methods put restrictions on the size or composition of the set whose compatibility is predicted [Veit et al., 2015, McAuley et al., 2015, He et al., 2016]. However, more recent models can recommend items that are complementary to an arbitrary set of fashion products [Lin et al., 2019, Veit et al., 2017].

The common approach is to use features extracted from images and sometimes additional data such as textual descriptions. The models then learn compatibility by exploiting datasets that capture some relations between items. It is possible to use implicitly collected data from e-commerce website [McAuley et al., 2015, Veit et al., 2015], but the majority of the methods employs a dataset consisting of user-created outfits.

Product Pairs

The most straightforward approach is to predict the compatibility of fashion product pairs [Veit et al., 2015, McAuley et al., 2015]. Both Veit et al. [2015] and McAuley et al. [2015] use the dataset captured by McAuley et al. [2015] from Amazon.com. The dataset contains relationships between items based on co-purchases and co-views on the website. Veit et al. [2015] sample compatible and incompatible pairs of fashion items from different categories. Then they train a Siamese CNN to capture the compatibility of products based on visual features.

The advantage of using pairs (instead of more complex structures such as a whole outfit) is that it is significantly easier to gather training data. That is because these relations can be captured implicitly on a website. On the other hand, it is difficult to interpret implicit feedback correctly. Moreover, we might not be able to infer relations in a set of more items from this source of data.

Fixed Outfit Composition

He and Hu [2018] propose models that learn compatibility of outfits with a fixed composition. The models employ VGGNet [Simonyan and Zisserman, 2014] pre-trained on ImageNet [Deng et al., 2009] for visual features extraction. The features are then passed to a fully connected neural network for compatibility prediction. He and Hu [2018] use their own dataset crawled from polyvore.com, a website where users can create fashion outfits.

Outfit as a Sequence

Han et al. [2017] predict the compatibility of arbitrarily long outfits. They achieved this relaxation of requirements by treating the outfit as a sequence. The input sequence is processed with bidirectional LSTM network. Because of that, they had to define an order of the items in the outfit based on the categories. The model makes predictions by combining the predictions inferred from left and right context of the target category. More specifically the model follows this equation:

$$x = \arg\max_{x_c \in \mathcal{C}} (Pr(x_c | x_1, \ldots, x_{m-1}) + Pr(x_c | x_{m+1}, \ldots, x_n)) \tag{3.1}$$

where x is the prediction made for the m-th item of the outfit, $\mathcal{C}$ is the set of all items and $(x_1, \ldots, x_{m-1}, x_{m+1}, \ldots, x_n)$ is the input partial outfit.

Subspace Embedding

Several models use more embedding spaces instead of only one in order to capture more kinds of similarities and visual relations [Vasileva et al., 2018, Veit et al., 2017, Lin et al., 2019, Tan et al., 2019]. All the mentioned models are adaptations of a conditional similarity network (CSN) [Veit et al., 2017]. A learned mask is applied to the general embedding based on a property (condition) of the input

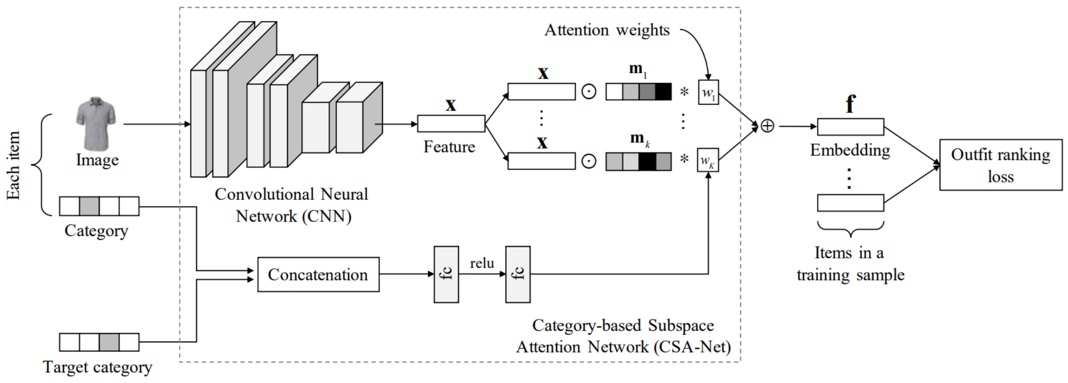

Figure 3.1: Overview of the model proposed by Lin et al. [2019]. When comparing the item pairs, the items are projected into multiple subspaces. To obtain the final embedding, a weighted sum of the subspace embeddings is computed. The weights are based only on the categories of the items of the pair.

items (such as their categories). Thanks to that, items can be compared in several representation subspaces that can capture different notions of similarity.

Vasileva et al. [2018] introduce subspaces assigned to category pairs, such as top-bottom. So when the compatibility of a top and a bottom item is computed, both embeddings are projected into this top-bottom subspace. The compatibility is then retrieved from the distance of the vectors in the corresponding subspace. In total, their model learns 66 similarity subspaces. Tan et al. [2019] do not define the conditions and the purpose of subspaces in advance. Instead, a weighted average of the embeddings from all subspaces is used. The weights are computed from the pair of input images. This approach overcomes the need for increasing the number of subspaces. Lin et al. [2019] further improve this approach and compute the weights based only on the categories of the items, as shown in Figure 3.1. This method allows simple indexing for complementary item retrieval as the embedding can be computed just from the product image and a pair of categories.

Graph-based Models

There were also attempts to represent an outfit as a graph [Cui et al., 2019, Cucurull et al., 2019, Singhal et al., 2020]. Cui et al. [2019] construct a category-level directed graph where nodes represent categories and edges represent relations between categories in outfits. An outfit is thus represented as a sub-graph with item features in corresponding nodes. Their proposed node-wise graph neural network reflects the context of each node and employs attention mechanism to calculate the outfit compatibility.

Cucurull et al. [2019] propose an item-level undirected graph. Nodes represent items, and an edge between two items exists if and only if there is an outfit containing those items. Representation of a node contains information about the item in the node and also about near items reachable from the target node. Graph convolutional network [Kipf and Welling, 2017] realizes this embedding. The compatibility of an outfit is treated as an edge prediction problem. Probability of edges between all items in the outfit is predicted. The average of the results corresponds to the predicted compatibility. Singhal et al. [2020] employ item-level graph based on the model from Cucurull et al. [2019] and extends their method.

The approaches designed by Cucurull et al. [2019] and Singhal et al. [2020] differ from all the methods we mention because they require edges between items during the prediction time. That means the models leverages more data than others. Moreover, it is not clear how they intend to handle new items in the dataset.

Transformer

At the time of writing this thesis, Prato et al. [2020] presented an approach that shares some similarities with our method. They claim to use the Transformer for compatibility prediction and outfit completion. However, the authors do not share details about the implementation nor training of the model, so it is not possible to compare the model reliably.

3.2.1 Classification of Outfit Recommenders

We can see some similarities of the aforementioned models across the particular architectures. Thus, we propose the following classes that group the models based on their general concept rather than architecture:

- *Generating* approaches predict the representation of a complementary item, as illustrated in Figure 3.2. The target item is then found as the closest one in a particular representation space. This approach is, for example, employed by Han et al. [2017] that generate the product's vector representation using bidirectional LSTM and uses dot-product to find the target item.

- *Compatibility Predicting* models find complementary items by predicting the compatibility of an outfit created from the initial set of items and a potential target item. The product that forms an outfit with the highest compatibility, is the target item. The above-mentioned models that belong into this category use the following ways to predict the compatibility:

  - *Pair-wise* approaches predict the compatibility of an outfit by aggregating pair-wise compatibilities of its items, as depicted in Figure 3.3. The compatibilities can be computed using a distance metric in an embedding space of the products. Using this strategy, it is usually possible to build database indexes from the product embeddings and find the target items using a nearest neighbour algorithm that is reasonably fast. A disadvantage might be that the input items cannot be embedded with contextual information. A Siamese network from Veit et al. [2015] or subspace embedding methods [Vasileva et al., 2018, Tan et al., 2019, Lin et al., 2019] may serve as examples of this approach.
  - *Monolithic* models process the whole input at once and use all the input items together to predict the compatibility, as shown in Figure 3.4. An advantage of these models is that they can utilize contextual information of the input. However, to find the target item, the model

has to process all the items in the database in order to compute their compatibility. An example of this approach is a model from He and Hu [2018] or graph-based models such as the one form Cui et al. [2019].

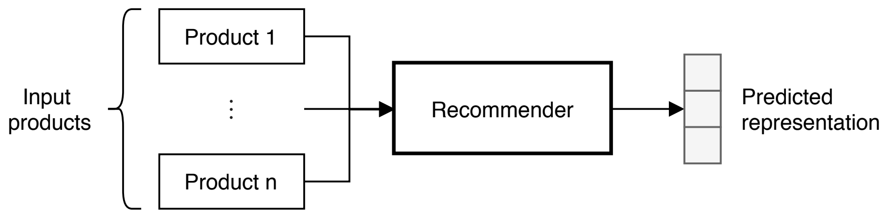

Figure 3.2: A diagram of generating outfit recommendation approach. The recommender predicts the target item's representation.

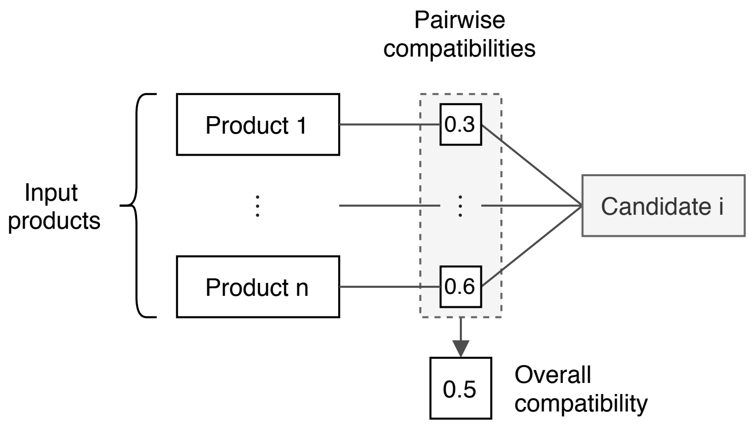

Figure 3.3: A diagram of pair-wise outfit recommendation approach that aggregates pair-wise compatibilities to compute the overall compatibility.

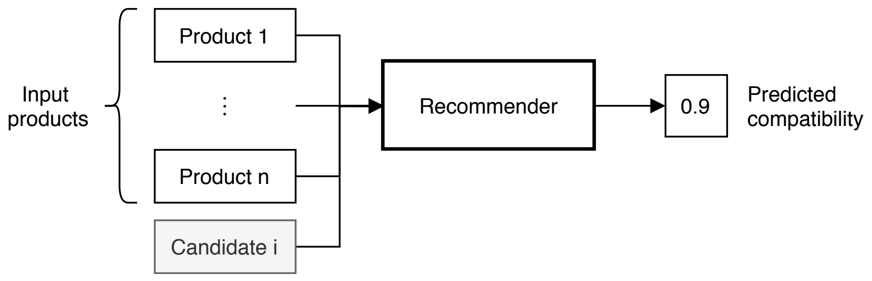

Figure 3.4: A diagram of monolithic outfit recommendation approach that predicts outfit compatibility.

3.3 Datasets

For outfit recommendation, the models have to learn not only compatibility of product pairs, but they need to take into consideration the outfit composition as well. Because of that, the models need data that captures these complex relations. The most popular datasets consist of manually created outfits. However, that is

Dataset	# outfits	# items	# categories	# high-level categories
Maryland Polyvore	21 889	164 379	380	N/A
Polyvore Outfits	68 306	251 008	153	19
Polyvore Outfits-D	35 140	152 785	153	19

Table 3.1: Comparison of datasets

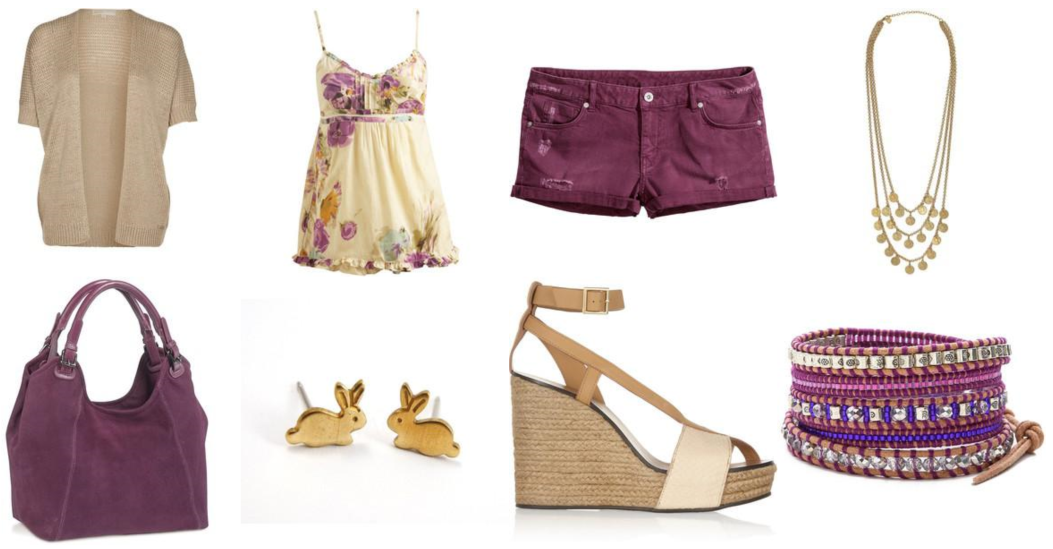

Figure 3.5: Example of an outfit from polyvore.com

quite costly and cannot be captured implicitly. Therefore, there exist only a few of these datasets, and the majority of them were crawled from the same website. A comparison of the datasets is shown in Table 3.1.

3.3.1 Maryland Polyvore

Han et al. [2017] created the first widely used public dataset for outfit recommendation. It is a dataset obtained from polyvore.com, a former website allowing users to create outfits out of various fashion products. An example of such outfit is shown in Figure 3.5. The dataset contains 21 889 outfits consisting of on average 6.5 fashion items. The histogram of outfit sizes in this dataset is shown in Figure 3.6. The outfits are divided into training, validation and test sets each containing 17 316, 1 497 and 3 076 outfits respectively. Overall, the whole dataset contains 164 379 items that do not overlap between two splits.

Each item contains an image on a white background, title, category and some additional information such as a number of likes. There are 380 fine-grained categories that are not grouped into any high-level product types (e.g. short skirts and long skirts are two different categories that are not linked together). Although the authors state that they deleted non-fashion items from the dataset, there are still items such as lamps or furniture.

The dataset comes with two evaluation tasks: fill-in-the-blank (FITB) and outfit compatibility prediction. The tasks became a popular way of evaluating outfit recommendation models [Veit et al., 2017, Vasileva et al., 2018, Cucurull et al., 2019].

In FITB, a question is formed from each outfit of the test set by removing one item from the outfit. The goal is to select the missing item from four options that consist of the correct choice and three random products. The incorrect candidates do not always match the category of the correct choice. Furthermore, the incorrect choices may even have the same category as an item from the input outfit.

The outfit compatibility prediction task consists of outfits from the test set and 4 000 randomly created outfits. The goal is to distinguish these two types of outfits.

3.3.2 Polyvore Outfits

Vasileva et al. [2018] proposed a new dataset crawled from polyvore.com as well. Their goal was to create a bigger dataset with better annotations and more consistent evaluation tasks than the one made by Han et al. [2017]. Hence, the authors collected a dataset containing 68 306 outfits and 251 008 items. In addition to the annotations from Maryland Polyvore, the categories are grouped into 19 semantic categories such as tops, outerwear or bags. The representation of these categories in outfits is shown in Figure 3.8.

The authors provide two different versions of their dataset. In the first version called Polyvore Outfits, items from test set may appear in the training part of the dataset. However, a whole outfit never appears in two splits of the dataset. Thanks to that, the dataset is bigger and contains all 68 306 outfits. The outfits are split into 53 306 for training, 10 000 for testing, and 5 000 for validation. The outfits contain 5 items on average, and you can see the distribution of outfit sizes in Figure 3.7. The more restricted version is called Polyvore Outfits Disjoint (Polyvore Outfits-D). Vasileva et al. [2018] state that this version was created using graph segmentation algorithm that ensured that one item always appears only in one split. However, we discovered that out of 14 657 products in the validation set 3 781 items appear in the training set as well. Because of the restrictions, some items are discarded, which leads to a dataset containing 35 140 outfits and 175 485 items. The training split contains 16 995 outfits; the validation split contains 3 000 outfits and 15 145 outfits from the test set. Notice that the counts differ from the original paper as we discovered that the dataset has different properties than presented. The outfits are formed of 5 items on average and a histogram of outfit sizes in this version is shown in Figure 3.7.

The compatibility prediction task is defined the same way as in Maryland Polyvore. The FITB task is also the same with the exception that all the candidates for one question have the same high-level category.

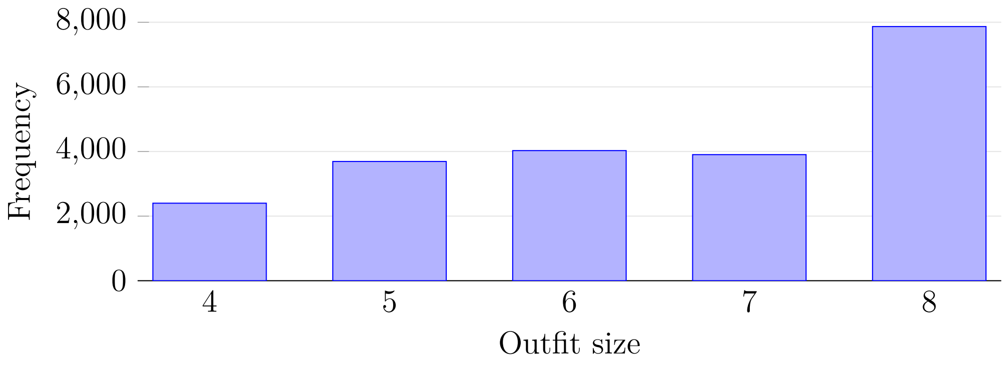

Figure 3.6: Distribution of outfit sizes in Maryland Polyvore

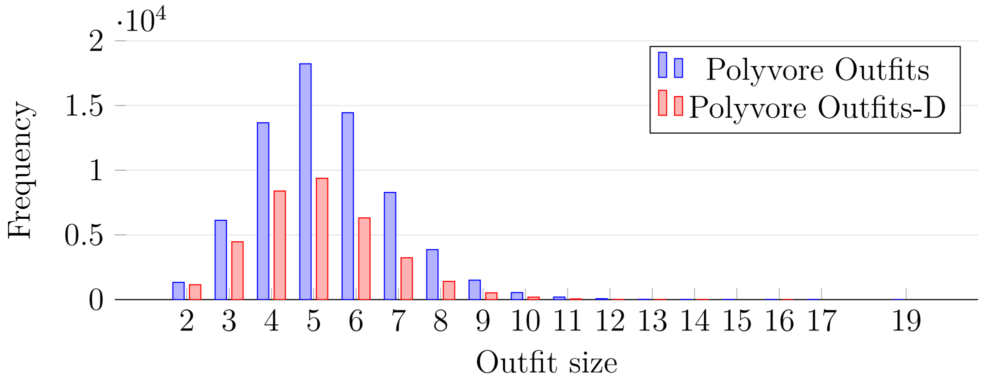

Figure 3.7: Distribution of outfit sizes in Polyvore Outfits

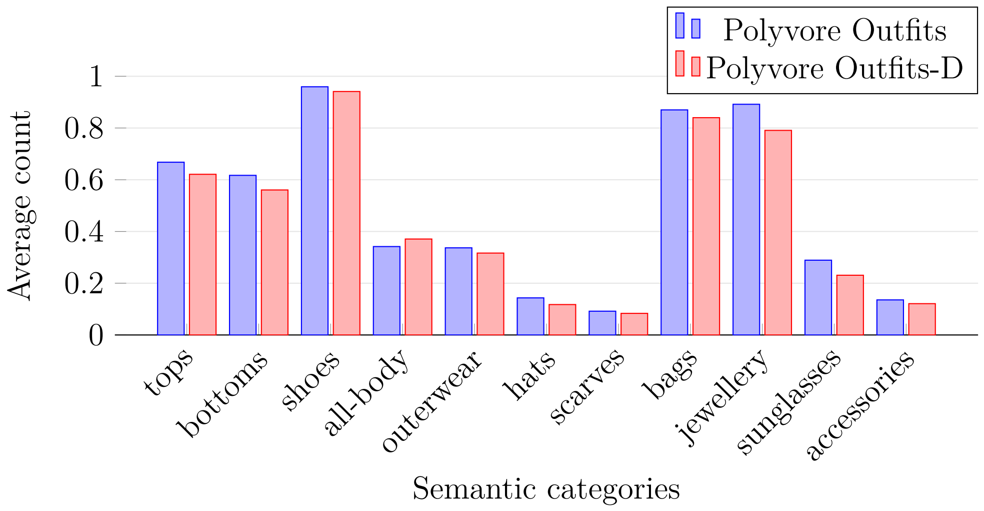

Figure 3.8: Average number of items of particular category in outfits of both Polyvore Outfits splits (disjoint and non-disjoint)

4. Our approach

Our approach to fashion recommendation is highly inspired by recent NLP models. We intended to transfer the semantics of a word in a sequence to a fashion product in an outfit. Thanks to that, we can employ NLP techniques for fashion recommendation. In particular, we use the encoder component from the Transformer [Vaswani et al., 2017], and our training process is similar to the one employed by Devlin et al. [2019].

4.1 Model

4.1.1 Architecture

The input of our model is a set of items that form a partial outfit. As the whole model operates over a set of items, the input can have an arbitrary size and its composition is not restricted in any way. Each input item must be composed of an image and a category. Let $I = \{I_1, \ldots, I_n\}$ is the set of input images, $C = \{C_1, \ldots, C_n\}$ is the set of input categories and optionally let c_t denotes the target category. We approach the outfit recommendation as learning of function $\psi(I, C)$ or $\phi(I, C, c_t)$ depending on whether the target category is known or not. The output of the function is the predicted representation of the item that complements the partial input outfit. The overall architecture is depicted in Figure 4.1.

The input images are first processed by a CNN. Then, the embeddings are passed to a fully-connected layer. The fully connected layer serves mainly for reducing the dimension. That is needed because the dimension of the CNN may be high, and the number of parameters of transformer blocks scales quickly with the dimension.

After the dimension reduction, a mask token is added to the set of embeddings. The value of this token is learned during the training phase. We propose two strategies of masking: single-token and category-wise masking. With single-token masking, only one masking token M is used for all prediction. On the contrary, with category-wise masking, we use a category-specific token $M = m(c_t)$. The output of this stage is a set $D = \{D_1, \ldots, D_n, M\}$ of item embeddings and a mask token.

We apply category embeddings on the outputs from the previous component. This stage is similar to the positional embedding in the Transformer. However, we learn the embedding vectors during the training. We merge the category embeddings with the vectors $D_1, \ldots, D_n$ using addition, concatenation or multiplication. When a category of the searched item is known, a category embedding can be applied to the mask token as well. We call the components up to this point a preprocessor as they prepare the embeddings for the encoder. The output of the preprocessor is a set of embedded items and mask $E = \{E_1, \ldots, E_n, E_M\}$.

The final and the essential part of our model is the encoder stack from the Transformer. We use the same architecture of the encoder blocks as the original Transformer model described in subsection 2.1.3. The input of the encoder is a set of embeddings produced by a preprocessor, as shown in Figure 4.1. The

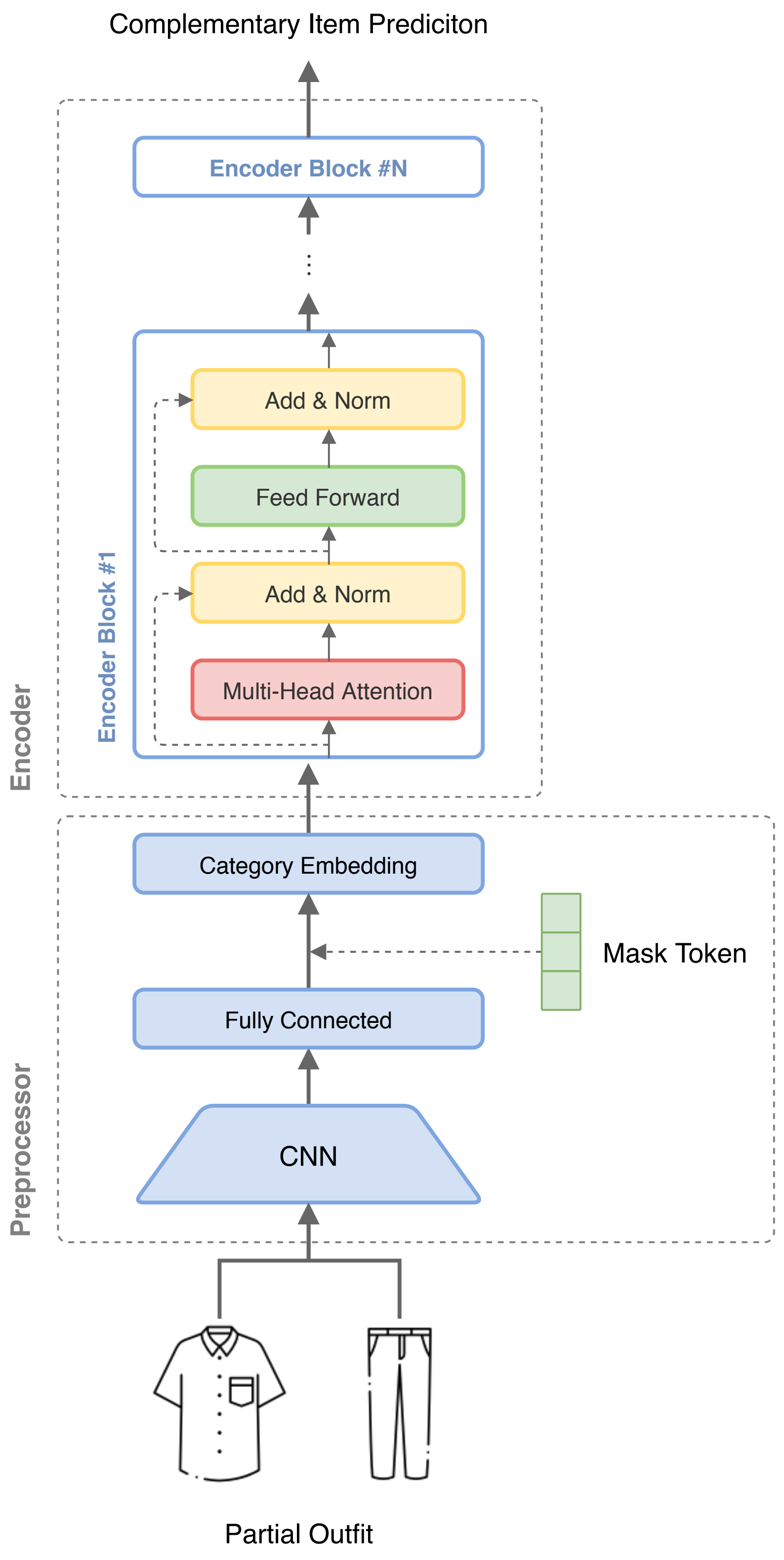

Figure 4.1: Architecture of our Fashion Encoder model. The input consists of fashion product images with product categories. A CNN first processes the images, the CNN embeddings are then enriched with category embeddings. Finally, the set of embedded fashion products is passed to the encoder.

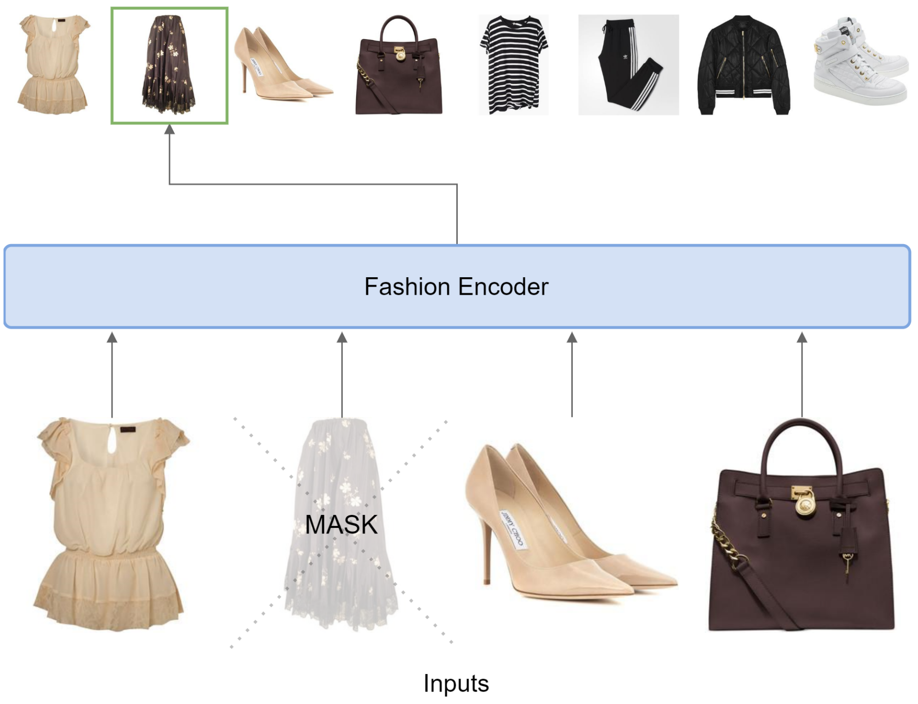

Figure 4.2: Fashion Encoder training process. The goal is to select the correct missing item from all items in the batch.

output of the encoder is a set of encoded products and the encoded mask token $O = \{O_1, ..., O_n, O_M\}$.

The output of our model is a vector O_M that represents the target item. The last step is to find the target item from the dataset. Unlike NLP models that work with a vocabulary of fixed size, we can not connect the output of the encoder directly to the logits vector of the vocabulary. However, we can find the target item by finding the most similar item according to a similarity function. We use a dot-product for this purpose. Formally, let $\mathcal{Y}$ is the set of all available items, let E_y denotes the preprocessor's embedding of an item $y \in \mathcal{Y}$, and finally, let y_t is the true target item. The predicted complementary item $\hat{y}$ is the one with the highest dot-product value:

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} (O_m \cdot E_y) \tag{4.1}$$

4.2 Training

To train the model, we use a masked outfit completion task that is similar to the FITB from Polyvore datasets and the masked language modelling employed in BERT [Devlin et al., 2019]. The input is formed from an outfit with one randomly selected item taken away. The goal is to choose the missing item from all items available in the batch, as depicted in Figure 4.2.

Following the notation from the previous section, we compute the model's loss as follows. We apply softmax function on the set of dot-products between E_m and all $y \in \mathcal{Y}$ of the current batch in order to create a probability distribution.

With this distribution we compute a cross entropy loss function l for one outfit:

$$l = -\log \left(\frac{\exp(E_m \cdot E_{y_t})}{\sum_{y \in \mathcal{Y}} \exp(E_m \cdot E_y)} \right) \tag{4.2}$$

Unlike the models using conditional similarity networks [Veit et al., 2017, Tan et al., 2019, Vasileva et al., 2018], that are trained using triplets, our training task allows the model to exploit the whole outfit at the same time. The triplet training does not provide any information about the outfit composition. However, masked outfit completion allows the model to learn relations between the items of the outfit. That might be beneficial in some cases when pairwise compatibilities are not sufficient. Consider trousers that match a particular shirt, and let they also match some shoes, the three items together do not necessarily make a compatible outfit, for example, if they share the same vivid colour.

Han et al. [2017] use a similar approach to train their bi-directional model. However, the bi-directionality of their model is limited by merging the left and right context; thus, the model can not process the context as a whole.

4.3 Discussion

In this section, we would like to explain the motivation and reasoning behind our design decisions. Moreover, we want to elaborate on some properties of the model.

4.3.1 Why Self-Attention

The Transformer and the self-attention mechanism was introduced in NLP as an alternative to recurrent neural networks. The goal was primarily to reduce sequential computations and allow capturing long-range dependencies [Vaswani et al., 2017]. However, fashion outfits are typically not nearly as long as textual data that is processed in NLP. Nonetheless, the Transformer with the self-attention is also a new approach to contextual embedding; and that is the reason why we explore its capabilities in outfit recommendation.

Our expectations from the multi-head self-attention used in the encoder are the following:

- Each fashion product of the encoded outfit can be encoded differently depending on its context. That allows one item to have different “role" in different outfits.
- The self-attention mechanism can regulate the impact of the items on the searched product. That should allow the main categories (e. g. tops or bottoms) to have a stronger influence than, for example, jewellery.
- When encoding the mask token, we expect it to gather all the necessary information from the input items. Moreover, when the category of the searched item is known, the key in the attention mechanism may be category-dependent. Consider we are searching for shoes compatible with our shirt and belt, the category-specific key should have the capability to pay more attention to the belt than the shirt.

- Because the weighted averages are computed for each head separately, this approach is more flexible than using only one head. Thus, we think that the multi-head attention might allow the model to encode different aspects of the products within each head.

4.3.2 Overall Design

We decided to approach the recommendation problem as a prediction of item's vector representation. We do that by encoding a mask token. However, it is also possible to modify the Transformer to predict outfit compatibility (with monolithic architecture). This can be done by imitating the classification approach of BERT [Devlin et al., 2019]. We can add a classification token to the input together with a partial outfit and a candidate complementary item. The output of the model that corresponds to the classification token is passed to a classification layer. The output of this layer should be a probability that the input outfit is compatible. That way, the output corresponding to the classification token is a representation of the whole outfit.

The variant that predicts compatibility may achieve higher accuracy as the model processes the partial outfit as well as the candidates. However, this approach is not suitable for retrieval because the model must be executed with every product in the database during the prediction in order to find the most compatible one. On the contrary, when using our architecture for retrieval, it is only needed to embed all the items into the one shared space to create indexes; and at the time of prediction, the model has to run only once. The output of the model is a vector representation; hence we can search through the indexes of the database to find the nearest item.

4.3.3 Encoder-only Architecture

We decided to use only the encoder, which is an approach used by BERT as well [Devlin et al., 2019]. The reason is that unlike the textual data, the products that form an outfit are not implicitly ordered. Because of that, it is not clear how should the items be ordered in the decoder. Moreover, encoding the mask token is sufficient for our case as we do not need to generate continuous sequences. Thus, we decided to use only the encoder part, which is order-independent and proven to work on its own in NLP.

The encoder-decoder or decoder-only architectures might be better for generating more than one fashion product thanks to the decoder's autoregressive nature. However, in the encoder-only architecture, this can be addressed by using the model iteratively (generating one product per run).

4.3.4 Comparison with Other Approaches

Our model is an example of generating outfit recommendation system (as defined in Subsection 3.2.1). Hence, our approach is similar to the bi-directional LSTM model by Han et al. [2017]. These methods predict the missing item using the whole outfit at once. Intuitively, our approach can utilize contextual information about the input products that can not be captured by the pairwise approach.

Nonetheless, the subspace embedding approaches have the state-of-the-art performance, as shown in Chapter 6.

In contrast to the models that put some restriction on the input, such as fixed composition [He and Hu, 2018] or order [Han et al., 2017], our model can process outfits of arbitrary sizes and compositions.

In context of recommender systems, we classify the approach as a monolithic hybrid system with aspects of content-based and collaborative methods. We use content attributes (product images and categories); however, we learn the relations between the items from community data. The recommendations are not personalized, but produced from the contextual information such as products in the shopping cart or a currently viewed item.

5. Implementation

In this chapter, we discuss the implementation of our approach.

5.1 Used Technologies

The project was written primarily in *Python 3.7*. We built the model using *Tensorflow* [1], a popular library for developing and training machine learning models. We tuned the hyperparameters using *Keras Tuner* [2], a hyperparameter tuning framework for Tensorflow. To streamline experimenting and data exploration, we used *Jupyter Notebooks*[3] that allow interactive prototyping. To enhance the portability of the project, we took advantage of a package and environment manager *Conda*[4]. We ran the training code inside a *Docker*[5] container.

5.2 Module Structure

The codebase is located inside `src` module that is further dived into the following submodules:

- `data` - A module containing the functionality that deals with the input data for the model. Specifically, it contains a code that builds the datasets files. It also contains methods that build the actual input pipeline for training and evaluation.
- `models` - The code related to training and the model itself is located in this module.
- `notebooks` - Data exploration and minor experiments that were done in Jupyter notebooks are in the `notebooks` module.

5.3 Input Pipeline

Our model consumes two types of datasets: a training dataset and a dataset for FITB the task. One sample of the training dataset corresponds to one outfit. The sample consists of a sequence of product images, a sequence of categories and a position that should be masked. A sample from the FITB dataset represents one FITB question. The sample contains a sequence of images of the partial outfit with one extra placeholder image, a sequence of corresponding categories (optionally with the target category at the position of the placeholder), a sequence of candidate (target) images with categories, an index of the correct target item and an index of the placeholder. A possible composition of the FITB sample is shown in Figure 5.1. Both dataset types may contain CNN representations instead of the raw images for optimization purposes.

[1]https://www.tensorflow.org/
[2]https://github.com/keras-team/keras-tuner
[3]https://jupyter.org/
[4]https://docs.conda.io/
[5]https://www.docker.com/

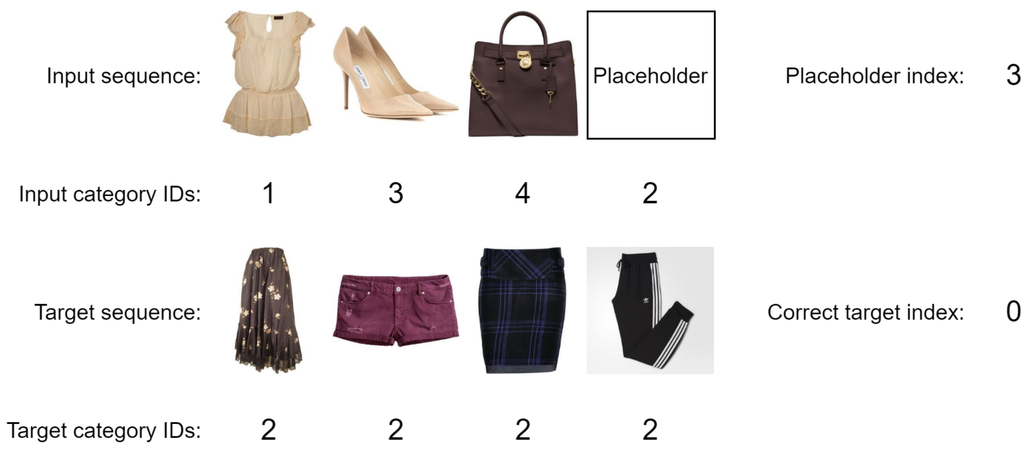

Figure 5.1: Example of one sample from the FITB dataset

We build the input pipelines using `tf.data`, which is a Tensorflow module that allows us to create complex and efficient data pipelines. First, we preprocess the datasets, so they almost comply with the above descriptions. Optionally, we extract the visual features from the images using a CNN. After this stage, we save the datasets to TFRecord files. TFRecord is a file format that allows storing sequential messages that are serialized using Protocol Buffers[6], a mechanism for efficient serializing of structured data.

The training loop consumes an instance of `tf.data.Dataset`, which we create from the TFRecord files. `tf.data.Dataset` is an abstraction of a sequence of samples that may be created from various data sources such as TFRecord files, CSV data or Python generator. Thanks to that, the input pipeline can be easily replaced with other sources of data. At this point, we add a random mask position to the training samples, and we also finalize the form of the FITB samples. To make sure that the input pipeline is not a bottleneck of the training, we take advantage of caching and prefetching the samples.

5.4 Model

The Fashion Encoder model is an instance of `tf.keras.Model` that is created using Keras functional API. We build the model from two separated parts: the preprocessor and the encoder. Thanks to that, the preprocessor can be easily interchanged, and the encoder can be used on its own. That allows us to design different methods of feature extraction that can, for example, exploit not only the product images. In the following subsections, we describe the implementation of these two components.

5.4.1 Preprocessor

We realize the preprocessor by a class `FashionPreprocessor` that is a subclass of `tf.keras.Model`. The preprocessor is a neural network that projects fashion

[6]https://developers.google.com/protocol-buffers/

products into vector representations. This network consists of four layers connected sequentially: a CNN extractor, a fully-connected network, a masking layer and a category embedding layer. However, the CNN and the category embedding layer are optional. We can modify the final shape and composition of the preprocessor via its constructor parameters.

The `CNNExtractor` is a thin wrapper around a CNN that deals with padding of the outfits when training in batches. We use a Keras implementation of InceptionV3 [Szegedy et al., 2016] pretrained on the ImageNet classification task [Deng et al., 2009] as the actual CNN. However, we also experiment with training the model without training the CNN (using the initial ImageNet weights). In that case, we use the dataset that consists of CNN embeddings; hence, the `CNNExtractor` may be omitted.

The fully-connected neural network reduces the dimension of vectors from the CNN (2048 in case of InceptionV3). It is a standard dense layer with leaky rectified linear unit (leaky ReLU) activation and dropout to reduce overfitting of the model.

The other mandatory part of the preprocessor is a masking layer. A masking layer replaces one of its inputs with a special masking token. We implement two types of masking: `SingleMasking` and `CategoryMasking`. `SingleMasking` replaces all the vectors at mask positions with the same vector which is learned during the training phase. `CategoryMasking` learns one masking token for each category and replaces the vectors with corresponding masks.

We implemented three ways of merging the category embeddings with the outputs from the CNN. Again, for straightforward interchangeability we implement the category embedding with three layers: `CategoryAdder`, `CategoryMultiplier` and `CategoryConcater`. Based on the preprocessor parameters, one or none of these layers is used.

5.4.2 Encoder

The encoder is based on the Tensorflow official implementation of the Transformer[7] that follows the original architecture from Vaswani et al. [2017]. We extracted the code of the encoder component and removed all the NLP-specific parts such as word embedding. Moreover, we implemented a functionality that allows computing key and query vectors from one-hot encoded categories instead of the full preprocessor representation.

5.5 Training

The training is implemented in a class `EncoderTask` that sets up the training process based on the constructor parameters. We train the model using a custom training loop that records all the operations performed by the model on a `GradientTape`. The gradient computed from the tape is then used to optimize the model with Adam optimizer [Kingma and Ba, 2015]. Our training loop provides a standard functionality such as loading and saving checkpoints, tracking the progress and custom callbacks. To avoid overfitting, we also implement an

[7]https://github.com/tensorflow/models

early stopping mechanism that can stop the training when the validation metric is not improving.

We utilize Keras Tuner to search for the best set of hyperparameters. It allows us to conveniently search the hyperparameter space using random search, Bayesian optimization or HyperBand algorithm [Li et al., 2018].

6. Evaluation

We evaluate our model on Maryland Polyvore and Polyvore Outfits datasets that are described in Section 3.3. Both datasets contain FITB and compatibility prediction tasks. However, there is no straightforward modification of our approach that would allow us to predict outfit compatibility. Thus, we evaluate the model only on the FITB tasks.

We compare our approach with the models mentioned in Section 3.2. All the results we present are the results reported by the authors of the models (if not stated otherwise). To distinguish the models, we are using this naming convention:

- **Type-aware**: a model that employs category pairs subspaces proposed by Vasileva et al. [2018].
- **CSA-Net**: Category Subspace Attention Network designed by Lin et al. [2019]
- **SCE-Net**: Similarity Condition Embedding Network proposed by Tan et al. [2019]
- **Trans-Net**: Transformer based model from Prato et al. [2020]
- **SiameseNet**: Siamese network implemented by Vasileva et al. [2018]
- **Bi-LSTM**: bidirectional LSTM model designed by Han et al. [2017]
- **FE**: our Fashion Encoder approach with the following hyperparameters:
  - d**D**: dimension d of preprocessor embeddings (hidden size)
  - **(ADD|MUL|CONCAT**[c]**)**: type of category embedding, c is a dimension of category embedding (c must be lower than d)
  - **CA**: the keys and queries of the self-attention mechanism are computed from one-hot encoded categories (we call it category attention)
  - f**F**: filter size f (number of units of the first dense layer of the encoder's feed forwards networks)
  - h**H**: number h of self-attention heads
  - l**L**: number l of encoder blocks (number of hidden layers)

6.1 Maryland Polyvore

FITB task from the Maryland Polyvore dataset [Han et al., 2017] contains questions that are formed by taking away one item from an outfit. The goal is to select the correct candidate from four options. The incorrect candidates are selected randomly, and their category does not have to match the correct item's category.

6.1.1 Implementation Details

This dataset contains the FITB task only for the test set. Hence, we generated a validation FITB task the same way that the test task was created, but from the outfits of the validation set.

We employ early stopping in all the trainings done on this dataset. Particularly, the training is stopped when 25 epochs are surpassed, and the validation accuracy does not improve by 0.2% for 10 epochs. We run the test task with the model that reached the highest validation accuracy.

We used Bayesian optimization implemented in Keras Tuner to find the best set of hyperparameters. The model is trained with 96 outfits in each batch with Adam optimizer [Kingma and Ba, 2015] with learning rate 0.002.

As the candidates of this FITB task might have different categories, we employ single token masking in all the experiments. Also, we do not apply the category embedding to the mask token. Thus, all the predictions are made solely based on the partial input outfit.

We always train all the parameters of the model except for the CNN (we use the ImageNet weights). We found out that this approach has the best performance.

6.1.2 Results

Our best model reached the accuracy of **72.6 %** in the Maryland Polyvore FITB task. We achieved this accuracy with a model with the hidden size of 64, filter size 128, two hidden layers and concatenated category embedding that takes up half of the hidden size. We evaluated this model 5 times; the final result is the average of these executions. The training of this model took 25 minutes using NVIDIA Tesla V100. A comparison with baseline and state-of-the-art methods is shown in Figure 6.1.

Maryland Polyvore FITB task tests not only the compatibility of styles but also the compatibility of categories because the candidates may have different categories. It is questionable whether this task is designed properly as some candidates can be dismissed based only on their category as discussed by Vasileva et al. [2018]. The Siamese network has the worst performance presumably because its input does not contain category information, and by design, it is not able to distinguish categories properly. The input of the bidirectional LSTM [Han et al., 2017] model includes images, descriptions and positions in the outfit. The Bi-LSTM outperforms the Siamese network by a significant margin. However, their model is still not able to dismiss items of incorrect categories reliably. Vasileva et al. [2018] report the best performance in this task with their Type-aware Embedding network. Their model learns projections to subspaces that corresponds to category pairs (e. g. top-bottom). Thus, when the model computes compatibility between items of categories that were not present during training (such as bottom-bottom, if no training outfit contains two bottom parts), then the compatibility is computed in their general embedding space. That way, the easily recognizable incompatible categories are filtered. Thanks to that, the accuracy of their model is that high.

Our best model without category embedding significantly exceeds the performance of the Siamese network. That is probably caused by the fact that our

Method	FITB Accuracy
SiameseNet	54.2
Bi-LSTM	68.6
Type-aware	**86.1**
FE-128D-256F-32H-2L (ours)	65.2
FE-64D-128F-2L-32H-CONCAT[32] (ours)	72.6

Table 6.1: Comparison of different approaches on the Maryland Polyvore FITB task [Han et al., 2017].

model processes the whole input at the same time, and thus can utilize contextual information (such as all the categories included in the outfit). However, we achieved the highest accuracy with a model that employs category embedding concatenation. Thanks to that, the model can use both visual and explicit category information that seems to be beneficial in this task.

6.2 Polyvore Outfits

Polyvore Outfits dataset was designed by Vasileva et al. [2018] and contains similar evaluation tasks as Maryland Polyvore. The only difference is that all the candidates of the FITB task have the same high-level category. Hence, the task evaluates rather the ability to recommend visually compatible items than recommending compatible categories. The dataset contains two splits (further described in Section 3.3): Polyvore Outfits (PO) and Polyvore Outfits Disjoint (PO-D).

6.2.1 Implementation Details

We employ the same early stopping mechanism as in the experiments with Maryland Polyvore. That is, the training is stopped when at least 25 epochs have passed, and the validation accuracy does not improve by 0.2% for 10 epochs. We use the model with the highest validation accuracy to run the test tasks.

We also use similar training hyperparameters as in the Maryland Poylvore. Each training batch contains 96 outfits and the model is trained with Adam optimizer [Kingma and Ba, 2015] with learning rate 0.002. We train all the model's parameters except for the CNN.

We use the high-level categories for both the category embedding and category attention as we discovered that the fine-grained categories do not improve the model's performance. Also, during the training, we dismiss the items that do not have the same high-level category as the target item in order to better approximate the FITB task.

6.2.2 Results

We achieved the FITB accuracy of **42.6 %** for Polyvore Outfits Disjoint and **48.6 %** for Polyvore Outfits. These results were reached with a model with a hidden size of 256, filter size of 128 and 32 attention heads. The model does not use category embedding; however, the keys and queries of self-attention are computed from one-hot encoded categories. We trained the model using NVIDIA

Method	FITB Accuracy	
	PO-D	PO
Bi-LSTM	39.4	39.7
SiameseNet	51.8	52.9
Type-aware	55.2	56.2
SCE-Net	–	61.6
Trans-Net	–	62.5
CSA-Net	**59.3**	**63.7**
FE-256D-512F-32H-2L-CA (ours)	42.6	48.6

Table 6.2: Comparison of different approaches on the Polyvore Outfits FITB tasks [Vasileva et al., 2018]. Note that the results for Bi-LSTM were reported by Vasileva et al. [2018] as the model is older than the task.

Tesla V100, the trainings last 30 and 50 minutes for Polyvore Outfits Disjoint and Polyvore Outfits, respectively. The final results are the averages of 5 trainings. We compare our results with other methods in Table 6.2.

The best results for this task were reported by Lin et al. [2019] that use a subspace embedding model with an attention mechanism. Their results are followed by the Transformer based model [Prato et al., 2020] and the Similarity Condition Network Tan et al. [2019]. The Similarity Condition Network employ product images as the embedding condition; thus, the model must compute the compatibility for every item of the dataset during the evaluation. Moreover, it is not clear whether the Transformer based model is suitable for retrieval as the authors may use the classification model for compatibility prediction (hence, may have the same issue as the SCE-Net). The Type-aware model [Vasileva et al., 2018] has about 5 % lower accuracy than the SCE-Net. The Siamese Network as a baseline method implemented by Vasileva et al. [2018] reaches the accuracies of 51.8 % and 56.2 % for Polyvore Outfits Disjoint and Polyvore Outfits in respective order. Our model does not exceed the accuracy of the Siamese Network, and the difference in the accuracies is significant. The reason may be that the generating approach to outfit recommendation is not very effective considering that the Bi-LSTM model [Han et al., 2017] achieves even lower accuracies.

All the methods achieve higher accuracy for Polyvore Outfits than for Polyvore Outfits Disjoint. That is expected as the Polyvore Outfits split contains more training data, and the products from training set may appear in the test set.

6.3 Hyperparameters and Modifications

In this section, we elaborate on the impact of individual hyperparameters and modifications of the model. Note that the following experiments were usually executed only twice (due to their computational complexity); thus, the results might be little inconsistent. However, we typically performed more experiments (with various configurations) than shown in the tables in order to gather as much information about the hyperparameters as possible.

Method	FITB Accuracy		# parameters
	MP	PO-D	
FE-32D-64F-32H-1L	63.4	39.8	74 080
FE-64D-128F-32H-1L	65.0	41.1	164 544
FE-128D-256F-32H-1L	64.3	41.8	394 624
FE-256D-512F-32H-1L	65.2	41.4	1 051 392
FE-32D-64F-32H-2L	62.8	41.0	82 496
FE-64D-128F-32H-2L	65.0	41.7	197 760
FE-128D-256F-32H-2L	65.2	41.9	526 592
FE-256D-512F-32H-2L	65.1	41.3	1 577 472
FE-64D-128F-32H-1L-CA	–	41.1	197 760
FE-256D-512F-32H-1L-CA	–	41.9	1 577 472
FE-64D-128F-32H-2L-CA	–	42.3	197 760
FE-256D-512F-32H-2L-CA	–	42.6	1 577 472

Table 6.3: Different sizes of our model evaluated on the Maryland Polyvore FITB task [Han et al., 2017].

6.3.1 Model Size

First, we experimented with a model without category embedding (using only the CNN embeddings). We explored the impact of hidden size and number of encoder blocks. A summary of these experiments is shown in Table 6.3. We found out that models with the hidden size higher than 64 have comparable performance.

We also observed that, for some configurations (e. g. a model with category attention), the model with two hidden layers has higher accuracy than the model with only one hidden layer. However, for other configurations, the number of hidden layers has no notable effect. This observation suggests that the contextual embeddings sometimes have a positive impact on the performance. Let us remind that the products are not contextually embedded in a model with only one encoder block, because in this case, only the encoder inputs are used when encoding the mask token via the self-attention mechanism.

6.3.2 Category Embedding

We observed a substantial performance improvement caused by the category embedding for Maryland Polyvore dataset. However, for Polyvore Outfits we were not able to improve the performance with this component. Tables 6.4, 6.5 and 6.6 show the results for addition, multiplication and concatenation category embedding in respective order.

All three variants increase the accuracy of Maryland Polyvore FITB; however, concatenation proved to be the best approach. We associate the high impact of category embedding with the fact that FITB candidates may have different categories. Thus, the model has to predict not only visually compatible item but also an item with a compatible category. To prove this hypothesis, we executed an experiment with a model with concatenation category embedding with a hidden size of 256 and 255 category dimension (i. e. only one dimension of the representation vector corresponds to the CNN features). With this model, we were able

Method	FITB Accuracy	
	MP	PO-D
FE-32D-64F-2L-32H-ADD	70.9	41.5
FE-64D-128F-2L-32H-ADD	71.1	41.4
FE-128D-256F-2L-32H-ADD	71.5	41.8
FE-256D-512F-2L-32H-ADD	70.7	41.6

Table 6.4: Evaluation of our model with different hidden sizes and added category embedding on the Maryland Polyvore FITB task [Han et al., 2017].

Method	FITB Accuracy	
	MP	PO-D
FE-32D-64F-2L-32H-MUL	70.2	40
FE-64D-128F-2L-32H-MUL	71.7	41.9
FE-128D-256F-2L-32H-MUL	71.8	41
FE-256D-512F-2L-32H-MUL	71.3	41.2

Table 6.5: Evaluation of our model with different hidden sizes and multiplication category embedding on the Maryland Polyvore FITB task [Han et al., 2017].

to reach 69% accuracy in the FITB task of the Maryland Polyvore dataset.

Note that the results for Polyvore Outfits with category embedding were obtained with a model that applies category embedding to the mask token as well. We did not notice any significant difference between a model with mask token category embedding and the one that applies category embedding only to the input items.

6.3.3 Category Attention

We experimented with a model that computes the key and query attention vectors just from the categories of particular products (one-hot encoded categories). Only the value vectors are computed from the preprocessor's embeddings. The goal of this modification was to reduce the complexity of the model by restricting the querying of the self-attention only to the explicit category information. This model achieves around 0.5 % higher accuracy (on Polyvore Outfits Disjoint) than an equivalent model without category embedding, as shown in Table 6.3.

We employ this technique only on Polyvore Outfits datasets because we find using category embedding more effective on Maryland Polyvore.

6.3.4 Number of Heads

We also examined the number of self-attention heads. For most configurations of our model, the number of attention heads does not have a notable impact on the performance. For a few configurations, there is a difference in accuracy between using one attention head and more than one head. Specifically, using more than one attention head lead to higher FITB accuracy (usually up to 1% improvement on the Maryland Polyvore FITB). For most experiments, we use 32 attention heads as we did not notice a decrease of the accuracy caused by a

Method	FITB Accuracy	
	MP	PO-D
FE-16D-32F-2L-16H-CONCAT[8]	66.5	37.4
FE-32D-64F-2L-32H-CONCAT[16]	72.1	40.8
FE-64D-128F-2L-32H-CONCAT[32]	72.6	40.5
FE-128D-256F-2L-32H-CONCAT[64]	72.5	40.7
FE-256D-512F-2L-32H-CONCAT[128]	72.4	41.3

Table 6.6: Evaluation of our model with different hidden sizes and concatenation category embedding on the Maryland Polyvore FITB task [Han et al., 2017].

higher number of heads. Also, note that the number of attention heads does not affect the model's size.

7. Discussion

7.1 Unsuccessful Modifications

Let us describe some modifications that we tried but did not use in the final model because of their inferior performance.

Maryland Polyvore Categories

Because the categories of Maryland Polyvore dataset are not grouped into high-level categories, we created our own category groups. We divided the fine-grained categories into nine semantic categories such as tops, bottoms or outerwear. Then, we used category embedding with these high-level groups. Our motivation was to make sure that the model is able to learn proper category embedding even for categories with a few items. However, the performance of this approach was worse than using the fine-grained categories.

CNN Training

We tried training the CNN together with the rest of the model. We also tried to split the training into two stages, all the parameters except for the CNN were trained in the first stage, and the whole model was fine-tuned in the second stage. None of these approaches improved the performance of the model. The reason might be that we were forced to lower the batch size down to 10 due to the GPU memory requirements of the CNN. We observed that the model is able to encode the contextual information so the input items (the partial outfit) have lower dot-product with the prediction. This behaviour reduces the training loss; however, is surely not desirable. Nonetheless, this effect is negligible with bigger batches.

Loss and Similarity Functions

We also unsuccessfully experimented with a distance similarity metric and distance loss function. With distance metric d and the notation from Section 4.1, the predicted item $\hat{y}$ can be found as follows:

$$\hat{y} = \arg\min_{y \in \mathcal{Y}} d(e_m, e_y) \tag{7.1}$$

where d can be an arbitrary metric; however, we only experimented with Euclidean distance.

We want to make such predictions that e_{y_t} is close to e_m and all other item embeddings are far from e_m. Thus, we define the loss function accordingly, and we inspire from the approach of Lin et al. [2019]. Let $\{y_1, \ldots, y_n\}$ is a set of the wrong items' embeddings. We first aggregate the distances to the incorrect candidates using an aggregation function φ (e. g. min or average):

$$d_n = \varphi(\{y_1, \ldots, y_n\}) \tag{7.2}$$

We experimented with a loss function that encourages a distance margin m between e_{y_t} and the embeddings of incorrect candidates:

$$l = \max(d(E_m, E_{y_t}) - d_n + m, 0) \tag{7.3}$$

The Architecture

We also tried to alter the overall architecture. For example, we experimented with putting the category embedding and the masking layer in front of the input dense layer. Then, we added a dense layer after the encoder component, so the dimension of the output was the same as the CNN embeddings. None of the variations that we tried achieved better performance than the one we propose. We chose the model that has the best performance and the most straightforward architecture. Figure 7.1 depicts the above-mentioned modifications in comparison with the final design.

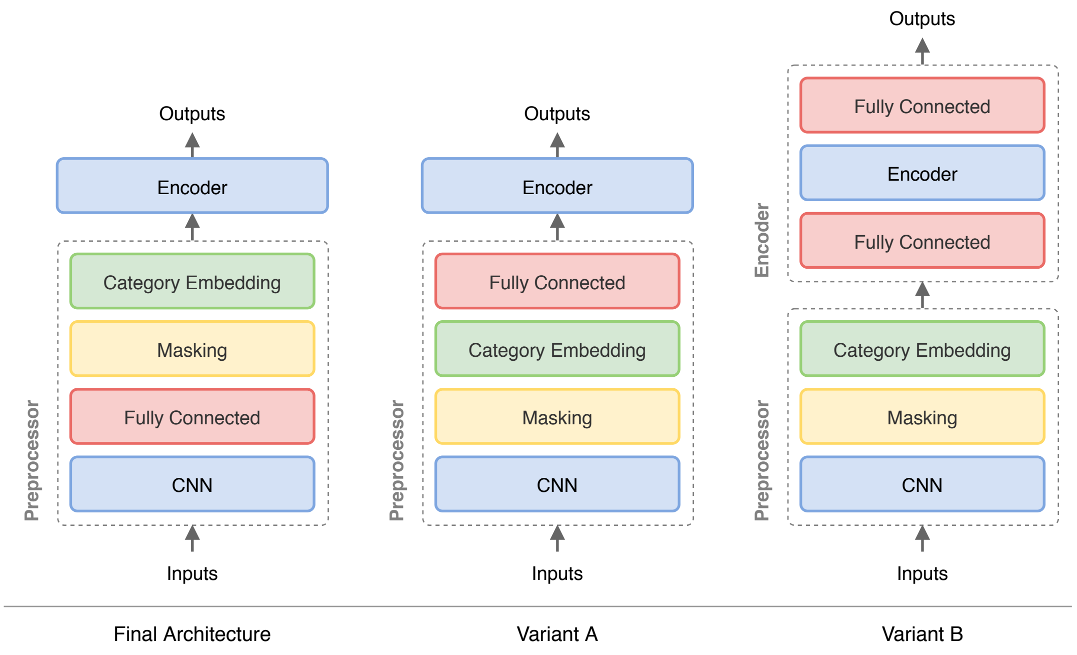

Figure 7.1: Comparison of possible Fashion Encoder architectures that we evaluated.

7.2 Future Work

Although the evaluation of our model suggests using another method, there are still some options of modifying our approach that may be interesting for future research.

Training the CNN

We were not able to improve the accuracy by training the CNN as discussed in the previous section. However, the InceptionV3 [Szegedy et al., 2016] may be replaced with a smaller network such as EfficientNet [Tan and Le, 2019] that should be implemented in Tensorflow 2.3. With a smaller network, the model can be trained with bigger batches and the accuracy may improve.

Alternatively, we can modify the training process, so it is not affected by the batch size that much. That may be done, for example, by dismissing the items of the input outfit from the possible candidates.

Enriched Features

The outfit recommendation models typically employ visual data, some of them also use category or textual information [Han et al., 2017, Lin et al., 2019, Prato et al., 2020, Vasileva et al., 2018]. The preprocessor of our model can be modified to exploit the textual data as well. With the textual data, the model may also use visual semantic embedding (VSE) that is described by Vasileva et al. [2018] and Han et al. [2017]. Both models report better performance with the introduction of VSE to their models.

Loss Functions and Similarity Metric

We restricted our experimenting mostly to the model trained with cross entropy loss function and a dot-product as a similarity function. We also tried using distance loss function with Euclidean distance; however we were not able to train the model with this configuration. Nonetheless, we neither proceeded in the analysis of this approach nor tried to alter the loss function. Hence, there is an opportunity for further improvement.

Conclusion

The goal of this thesis was to explore the possibility of employing recent NLP approaches in outfit recommendation. We showed that it is possible to create a model for outfit recommendation based on the Transformer [Vaswani et al., 2017].

To achieve this goal, we researched the essential concepts of recommender systems. Then, we examined the methods of complementary item recommendation. We explored the recent state-of-the-art approaches of natural language processing and thoroughly analysed the models that seemed to be adaptable for outfit recommendation. Based on this research, we designed a novel architecture based on the Transformer's encoder component [Vaswani et al., 2017] with a training process inspired by BERT [Devlin et al., 2019]. We examined several architectures, and we implemented various modifications to the model, such as category embedding, to improve the performance of the model. To find the best configuration, we employed Bayesian optimisation for hyperparameter tuning.

We evaluated the model on the fill-in-the-blank tasks of the standard datasets: Maryland Polyvore and Polyvore Outfits. On Maryland Polyvore we achieved the accuracy of 72.6 %. That is better than the baseline methods but worse than the current state-of-the-art approach, which has around 14 % higher accuracy. On Polyvore Outfits FITB where candidates have the same high-level category, our model reached 42.6 % and 48.6 % accuracy for Polyvore Outfits Disjoint and Polyvore Outfits, respectively. Thus, our model does not exceed the Siamese Network's baseline that is 51.8 % and 52.9 %. Moreover, the state-of-the-art method from Lin et al. [2019] has significantly higher accuracy of 59.3 % and 63.7 %.

Although the models that we examined do not achieve the necessary performance, there is still a space for experimenting with some modifications, for example, relating to the loss function or similarity metric. We believe that our work sets directions and boundaries for further research of self-attention based models in outfit recommendation.

Bibliography

Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, June 2005. ISSN 1041-4347. doi: 10.1109/TKDE.2005.99. URL `https://doi.org/10.1109/TKDE.2005.99`.

Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *ArXiv*, abs/1607.06450, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2015.

John S. Breese, David Heckerman, and Carl Myers Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *UAI*, 1998.

Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc V. Le. Massive exploration of neural machine translation architectures. *ArXiv*, abs/1703.03906, 2017.

Robin Burke. *Hybrid Web Recommender Systems*, pages 377–408. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-72079-9. doi: 10.1007/978-3-540-72079-9_12. URL `https://doi.org/10.1007/978-3-540-72079-9_12`.

Huizhong Chen, Andrew C. Gallagher, and Bernd Girod. Describing clothing by semantic attributes. In *ECCV*, 2012.

Qiang Chen, Junshi Huang, Rogério Schmidt Feris, Lisa M. Brown, Jian Dong, and Shuicheng Yan. Deep domain adaptation for describing people based on fine-grained clothing attributes. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5315–5324, 2015.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *ArXiv*, abs/1406.1078, 2014.

Guillem Cucurull, Perouz Taslakian, and David Vázquez. Context-aware visual compatibility prediction. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12609–12618, 2019.

Zeyu Cui, Zekun Li, Shu Wu, Xiaoyu Zhang, and Liang Wang. Dressing as a whole: Outfit compatibility learning based on node-wise graph neural networks. In *WWW '19*, 2019.

Jia Deng, Wenjun Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR 2009*, 2009.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2019.

Wei Di, Catherine Wah, Anurag Bhardwaj, Robinson Piramuthu, and Neel Sundaresan. Style finder: Fine-grained clothing style detection and retrieval. *2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 8–13, 2013.

Gintare Karolina Dziugaite and Daniel M. Roy. Neural network matrix factorization. *ArXiv*, abs/1511.06443, 2015.

A. Felfernig and R. Burke. Constraint-based recommender systems: Technologies and research issues. In *Proceedings of the 10th International Conference on Electronic Commerce*, ICEC '08, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605580753. doi: 10.1145/1409540.1409544. URL `https://doi.org/10.1145/1409540.1409544`.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. *ArXiv*, abs/1705.03122, 2017.

Xintong Han, Zuxuan Wu, Yu-Gang Jiang, and Larry S. Davis. Learning fashion compatibility with bidirectional lstms. *Proceedings of the 25th ACM international conference on Multimedia*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

Tong He and Yang Hu. Fashionnet: Personalized outfit recommendation with deep neural network. *ArXiv*, abs/1810.02443, 2018.

Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. *Proceedings of the 26th International Conference on World Wide Web*, 2017.

Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John Thomas Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22:5–53, 2004.

Julia Hirschberg and Christopher D. Manning. Advances in natural language processing. *Science*, 349:261–266, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *ArXiv*, abs/1801.06146, 2018.

Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272, 2008.

Dietmar Jannach. *Recommender systems : an introduction.* Cambridge University Press, 2011. ISBN 978-0-521-49336-9.

Xiaoqi Jiao, Y. Yin, Lifeng Shang, Xin Jiang, Xusong Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. *ArXiv*, abs/1909.10351, 2019.

Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *EMNLP*, 2013.

Taewan Kim, Seyeong Kim, Sangil Na, Hayoon Kim, Moonki Kim, and Beyeongki Jeon. Visual fashion-product search at sk planet. *ArXiv*, abs/1609.07859, 2016.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ArXiv*, abs/1609.02907, 2017.

Yehuda Koren. Collaborative filtering with temporal dynamics. In *KDD*, 2009.

Yehuda Koren, Robert M. Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42, 2009.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018. URL `http://jmlr.org/papers/v18/16-558.html`.

Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 World Wide Web Conference*, pages 689–698, 2018.

Yen-Liang Lin, Son N. Tran, and Larry S. Davis. Fashion outfit complementary item retrieval. *ArXiv*, abs/1912.08967, 2019.

Jiahui Liu, Peter Dolan, and Elin Rønby Pedersen. Personalized news recommendation based on click behavior. In *Proceedings of the 15th International Conference on Intelligent User Interfaces*, IUI '10, page 31–40, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605585154. doi: 10.1145/1719970.1719976. URL `https://doi.org/10.1145/1719970.1719976`.

Nathan N. Liu and Qiang Yang. Eigenrank: A ranking-oriented approach to collaborative filtering. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, page 83–90, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605581644. doi: 10.1145/1390334.1390351. URL `https://doi.org/10.1145/1390334.1390351`.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692, 2019.

Julian J. McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. Image-based recommendations on styles and substitutes. *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2015.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. *ArXiv*, abs/1708.00107, 2017.

Prem Melville, Raymond J. Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *AAAI/IAAI*, 2002.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *ArXiv*, abs/1301.3781, 2013.

David M. Nichols. Implicit rating and filtering. In *Proceedings of the Fifth DELOS Workshop on Filtering and Collaborative Filtering*, pages 31–36. ERCIM, 1998.

John O'Donovan and Barry Smyth. Trust in recommender systems. In *Proceedings of the 10th International Conference on Intelligent User Interfaces*, IUI '05, page 167–174, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1581138946. doi: 10.1145/1040830.1040870. URL `https://doi.org/10.1145/1040830.1040870`.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

Matthew E. Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. Semi-supervised sequence tagging with bidirectional language models. *ArXiv*, abs/1705.00108, 2017.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *ArXiv*, abs/1802.05365, 2018.

Gabriele Prato, Federico Sallemi, Paolo Cremonesi, Mario Scriminaci, Stefan Freyr Gudmundsson, and Silvio Palumbo. Outfit completion and clothes recommendation. *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020.

Alec Radford. Improving language understanding by generative pre-training. *OpenAI Blog*, 2018. URL `https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf`.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 2019. URL `https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf`.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *ArXiv*, abs/1910.10683, 2019.

Paul Resnick and Hal R. Varian. Recommender systems. *Commun. ACM*, 40: 56–58, 1997.

Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *CSCW '94*, 1994.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108, 2019.

Badrul Munir Sarwar, George Karypis, Joseph A. Konstan, and John Riedl. Recommender systems for large-scale e-commerce : Scalable neighborhood formation using clustering. In *Proceedings of the fifth international conference on computer and information technology*, volume 1, pages 291–324, 2002.

J. Ben Schafer, Dan Frankowski, Jonathan L. Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In *The Adaptive Web*, 2007.

Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *WWW '15 Companion*, 2015.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 2018. doi: 10.18653/v1/n18-2074. URL `http://dx.doi.org/10.18653/v1/N18-2074`.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *ArXiv*, abs/1409.1556, 2014.

Anirudh Singhal, Ayush Chopra, Kumar Ayush, Utkarsh R. Patel, and Balaji Krishnamurthy. Towards a unified framework for visual compatibility prediction. *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 3596–3605, 2020.

Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. Artificial Intellegence*, 2009:421425:1–421425:19, 2009.

Jianing Sun, Yiran Zhang, Chen Ma, Mark Coates, Huifeng Guo, Ruiming Tang, and Xiuqiang He. Multi-graph convolution collaborative filtering. *2019 IEEE International Conference on Data Mining (ICDM)*, pages 1306–1311, 2019.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *J. Mach. Learn. Res.*, 10:623–656, 2009.

Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *ArXiv*, abs/1905.11946, 2019.

Reuben Tan, Mariya I. Vasileva, Kate Saenko, and Bryan A. Plummer. Learning similarity conditions without explicit supervision. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10372–10381, 2019.

Edward P. K. Tsang. Foundations of constraint satisfaction. In *Computation in cognitive science*, 1993.

Lyle H. Ungar and Dean P. Foster. Clustering methods for collaborative filtering. In *AAAI workshop on recommendation systems*, pages 114–129, 1998.

Mariya I. Vasileva, Bryan A. Plummer, Krishna Dusad, Shreya Rajpal, Ranjitha Kumar, and David A. Forsyth. Learning type-aware embeddings for fashion compatibility. In *ECCV*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *ArXiv*, abs/1706.03762, 2017.

Andreas Veit, Balazs Kovacs, Sean Bell, Julian McAuley, Kavita Bala, and Serge Belongie. Learning visual clothing style with heterogeneous dyadic co-occurrences. *2015 IEEE International Conference on Computer Vision (ICCV)*, December 2015. doi: 10.1109/iccv.2015.527. URL `http://dx.doi.org/10.1109/ICCV.2015.527`.

Andreas Veit, Serge J. Belongie, and Theofanis Karaletsos. Conditional similarity networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1781–1789, 2017.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *ArXiv*, abs/1905.00537, 2019a.

Wei Wang, Bin Bi, Ming Yan, Chen Wu, Zuyi Bao, Jiangnan Xia, Liwei Peng, and Luo Si. Structbert: Incorporating language structures into pre-training for deep language understanding. *ArXiv*, abs/1908.04577, 2019b.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*, abs/1609.08144, 2016.

Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. Deep matrix factorization models for recommender systems. In *IJCAI*, 2017.

Kota Yamaguchi, Takayuki Okatani, Kyoko Sudo, Kazuhiko Murasaki, and Yukinobu Taniguchi. Mix and match: Joint model for clothing and attribute recognition. In *BMVC*, 2015.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*, 2019.

Markus Zanker and Markus Jessenitschnig. Collaborative feature-combination recommender exploiting explicit and implicit user feedback. *2009 IEEE Conference on Commerce and Enterprise Computing*, pages 49–56, 2009.

Tong Zhang and Vijay S. Iyengar. Recommender systems using linear classifiers. *Journal of Machine Learning Research*, 2:313–334, 2002.

List of Figures

List of Tables

List of Abbreviations

CNN	Convolutional Neural Network
LSTM	Long Short-Term Memory
NLP	Natural Language Processing
RNN	Recurrent Neural Network
GRU	Gated Recurrent Unit
BERT	Bidirectional Encoder Representations from Transformers
NSP	Next Sentence Prediction
CSN	Conditional Similarity Network
FITB	Fill-In-The-Blank
ReLU	Rectified Linear Unit
VSE	Visual Semantic Embedding

A. Attachments

A.1 Electronic Attachment

The contents of the electronic attachment are the following:

- `Framework/` – Framework for training and evaluation of the Fashion Encoder model
- `Thesis.pdf` – Text of this thesis