UNIVERSITAT DE BARCELONA

Facultat de Matemàtiques i Informàtica

DOBLE GRAU DE MATEMÀTIQUES I ENGINYERIA INFORMÀTICA

Treball final de grau

Self-Explaining Recommendation

Autor: Andreu Vall Hernàndez

Directora:	Dra. Maria Salamó Llorente
Codirector:	Alejandro Ariza Casabona
Realitzat a:	Departament de Matemàtiques i Informàtica

Barcelona, 10 de juny de 2024

Abstract

Recommender Systems are increasingly becoming more and more part of our everyday life, from helping us choose what videos to watch to recommending news articles and friends. However, their architecture is really complex which makes it difficult to explain their suggestions.

The goal of this project is to study in depth how does a Recommender System work, which generates automatically, in natural language, explanations on their recommendations. Furthermore, its details will be studied with different settings. It will be evaluated, analysed and discussed with different datasets.

Resum

Els sistemes de recomanació estan esdevenint cada cop més part del nostre dia a dia, des d'ajudar-nos a triar quins vídeos veure fins quines notícies llegir o amics trobar. Tot i així la seva arquitectura és molt complexa i això dificulta la justificació de les seves suggerències.

En aquest projecte el meu objectiu és estudiar en profunditat com funciona un sistema de recomanació que genera, en llenguatge natural automàticament, explicacions de les recomanacions. A més a més, s'analitzaran cadascuna de les seves parts amb diferents configuracions. Es farà una avaluació amb diferents conjunts de dades i s'analitzaran i es discutiran els resultats obtinguts.

2020 Mathematics Subject Classification. 68T07, 68T35, 68T50

Acknowledgements

I am deeply grateful to my parents, friends and family for encouraging me to continue the career and finish this thesis despite the difficult last year.

Naturally, I would also like to mention my mentor Dra. Maria Salamó and co-mentor Alejandro Ariza for all their dedication and valuable guidance, as well as all the teachers who have contributed to my academic growth and development.

Finally, I would like to thank all the amazing researchers who have dedicated part of their life to advancing science and published their code and results for anyone to use.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Around 50 million years ago, the ancestors of monkeys and apes split from other primates and became social creatures. Natural selection [1] favored bigger brains and early hominids emerged as bipedal creatures around 4 million years ago. While it is not completely certain it is believed that it was not until 100,000 years ago that language emerged, into already modern humans genetically speaking.

Earliest forms of long-lasting symbolic creativity were cave paintings, with the oldest known one being in Spain and dating to, at least, 64,000 years ago [18]. Writing was only invented around 5,000 years ago [5].

Nowadays, with the invention of devices that can record and reproduce information in the form of sound, images and video, alongside the creation of the Internet, a global network and protocol to share information, a new way of getting stimulus and information has been forged.

We are no longer restricted to owning physical content in able to enjoy it. This is very useful and contributes to the freedom of information, but has produced an information overload problem.

Typically, when we wanted to get recommendations of anything, we would ask a relative or a friend for suggestions, or simply look at other creations of authors we had previously liked. However, nowadays more content is created than any human being could interact with within a lifetime. For example, as of December 2022, more than 500 hours of video are uploaded to YouTube every minute [28].

Recommender Systems [26] can help us navigate this content overflow by showing us only elements that might be relevant to us, based on the previous elements that we have interacted with. They can be really helpful in some cases, but if they are not implemented carefully they can perpetuate existing biases, prejudices and foster radicalization [31].

In order to study their decisions, there are usually two main problems: reproducibility and explainability.

First of all, Recommender Systems implemented in major applications are often closed source, without specifying anywhere what type of data is being collected about the users, how they have been trained and what they are exactly trying to maximize. The only possible approach that could be done in this case would be to define fairness metrics to evaluate the platforms, but as end users, the only choice that you usually have is whether you use the platform or not.

Secondly, even if the data and training were to be open sourced, these models' architecture tends to be very complex and there is not an easy way to explain why certain suggestions were made instead of others. There are other possible architectures like Decision Trees [11] that are more transparent and interpretable, but it is usually at the cost of a lot of accuracy. Other possible solutions include post-hoc explainers [23], which consists in training an independent model that only generates explanations. They are versatile and can be used with any architecture, but they aim to justify a decision only after it has been made and having no influence on the decision process whatsoever.

The current main building block of Artificial Intelligence (AI) is Machine Learning and Deep Neural Networks, which are basically really long non-linear functions that have a lot of parameters that can be automatically adjusted in order to minimize some loss goal. This deep structure makes them useful to be able to learn interesting patterns in the data, but also makes it difficult to interpret their results.

Recently, there has been a lot of progress and hype within the field of Generative AI and Natural Language Generation, with tools like ChatGPT reaching a lot of global usage. A new more parallelizable architecture was proposed [16] which has lead to a lot of more useful computer generated content.

With the realization of this project, I will study whether these advances in Natural Language Generation can be used in the field of Recommender Systems for Explainable Recommendation [25].

The main goal will be to generate natural text explanations on the reason why some items are being recommended or not, similar to what a human might say, formalizing it mathematically and analyzing the whole process.

If successful, this kind of models could be used as a bridge in communication between real users and black box algorithms, being able to directly tell the computer exactly what you desire.

1.1 Project objectives

These are the main goals of this thesis:

- Choose a well-known paper on explainable recommendation
- Understand the key parts and formalize the mathematics involved
- Identify areas of improvement
- Discuss and implement potential solutions
- Validate the changes

1.2 Project organization

This thesis will be divided into six chapters. Chapter 1 is this introduction where the problem has been presented and some ambitious goals have been set.

Chapter 2 will be the Scientific background, containing mathematical explanations on how the recommendation and text generation tasks are mathematically formalized and how they can both be combined.

The rest of the chapters will describe the practical part, with Chapter 3 explaining the choice of an interesting State-of-the-Art paper and its most relevant ideas.

Chapter 4 will explain some limitations of this State-of-the-Art model and propose alternative approaches that could lead to improvements.

Chapter 5 will contain the experiments performed on real data to empirically analyze the effects of these changes.

Finally, Chapter 6 will conclude with a reflection on what it has been achieved and future directions of self-explaining AI, while realizing its limitations.

1.3 Project schedule

A Gantt chart is presented below to provide a visual representation on how almost 4.5 months of 7-days a week full time work, have gone by:

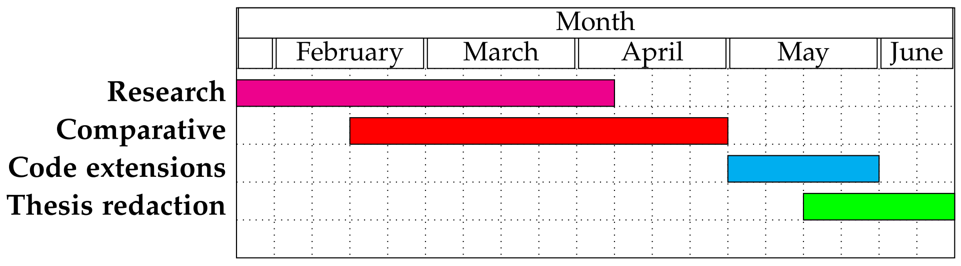


Figure 1.1: Gantt Chart

This Gantt chart as well as the backbone of the memory has been based on Adrian's latex thesis [37].

Initially, the goal of the project was a comparative of multiple codes, but as I was trying to do it, I ran into a lot of problems due no to not having enough computing resources (a lot of GPU memory is needed), and found out that it was not completely aligned with neither my interests nor values, and lacked a bit of depth. For this reason, the goal of the project was changed into an analysis in detail of just one chosen procedure, which made it easier to justify the computer science and mathematics work.

Chapter 2

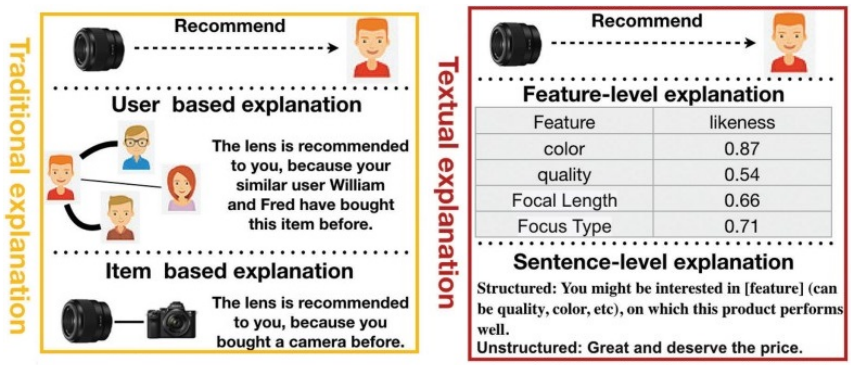
Scientific background

Explainable Recommendation is a subfield of Artificial Intelligence which attempts to develop models that generate not only high-quality recommendations but also intuitive explanations [23] to let people understand why certain elements rather than others are recommended [25].

As it can be seen in Figure 2.1 below, the explanations can be of any type (visual explanations, based on templates, etc). The types of explanations which I will study are sentence explanations which are not based on templates.

This Chapter's goal is to formalize the Explainable Recommendation task.

The first Section will start with a brief introduction to the most relevant concepts of Deep Learning which will be used in the project. The following Sections will introduce the Recommender Systems formalization and the Natural Language Generation task, and the last Section will conclude on how they can both be combined.



The lens is recommended to you, because your similar user William and Fred have bought this item before.

The lens is recommended to you, because you bought a camera before.

Feature	likeness
color	0.87
quality	0.54
Focal Length	0.66
Focus Type	0.71

Structured: You might be interested in [feature] (can be quality, color, etc), on which this product performs well.
Unstructured: Great and deserve the price.

Figure 2.1: Different types of recommendation explanations [23]

2.1 Deep Learning

Definition 2.1. *A **neural network** (NN) is a function $f : X \times \Theta \to Y$, with $X \subset \mathbb{R}^n$ the **input**, $Y \subset \mathbb{R}^m$ the **output** and $\Theta \subset \mathbb{R}^d$ the **parameter**.*

Definition 2.2. *An **activation function** is a non-linear function $f : \mathbb{R}^n \to \mathbb{R}^n$. It is usually differentiable almost everywhere.*

Example 2.3. The **sigmoid function** $\sigma : \mathbb{R} \to (0,1)$ [6] is a common activation function, defined as:

$$\sigma(x) := \frac{1}{1+e^{-x}} \tag{2.1}$$

Example 2.4. The **softmax function** $S : \mathbb{R}^n \to \mathbb{R}^n$ is the generalization of the sigmoid function 2.3 for $n > 1$. It is defined as:

$$S(x) := (\frac{e^{x_1}}{\sum_{i=1}^n e^{x_i}}, ..., \frac{e^{x_n}}{\sum_{i=1}^n e^{x_n}}) \tag{2.2}$$

It converts any vector $x \in \mathbb{R}^n$ into a probability distribution.

Definition 2.5. *A **neuron** or **perceptron** is the composition of a linear function with an activation function. It is a simplification of a human neuron and synapses, with the activation function being used to determine the intensity with which the neuron is activated with.*

Definition 2.6. *A **layer** l is a stack of N individual neurons, with all of them having the same input and output size.*

The linear part of the layer can be expressed as a matrix operation,

$$Y = WX + b \tag{2.3}$$

where $X \in \mathbb{R}^n$ is the layer's input, W $\in \mathbb{R}^{mxn}$ the weights, b $\in \mathbb{R}^m$ the bias and $Y \in \mathbb{R}^m$ the output; with n being the layers's input size and m the output size

Definition 2.7. *A **multilayer perceptron** (MLP) is a neural network $f : X \times \Theta \to Y$ which is the composition of neurons organized in $N \geq 3$ layers, $l_1, ..., l_N$. The parameter Θ corresponds to the weights W and bias b of each layer. The X input of the NN is fed to the first layer, and it produces an X_1 output, which is then passed to the next layer and so on and so forth, until $Y = X_N$. The layers $l_2, ..., l_{N-1}$ are called **hidden** layers.*

If not for the non-linear activation function, all MLP would be reduced to a simple linear function. There are other ways to connect the neurons besides the fully connected, uni-directional MLP flow.

Recurrent Neural Neworks (RNN) [3] have loops in their structure, which can be seen as some kind of memory, but MLP is the simplest one.

There are various forms of universal approximation theorems that define limits on what neural networks can theoretically learn [4]. Deep Learning [7] was the practical realization that, by simply increasing the number of hidden layers, this kind of models could *learn* useful patterns from data. Any form of Deep Learning has four key components [26]:

1. The **data** that we want to learn from.
2. The **model** that defines how to transform the data.
3. An **objective function** that quantifies how well the model is doing.
4. An **algorithm** that defines how to adjust the model's parameters to optimize the objective function.

Definition 2.8. *A **loss function** is a function $\mathcal{L} : Y \times Z \to \mathbb{R}$ that maps values into a real number representing some cost. Z is an optional parameter depending on the problem.*

Example 2.9. Given two vectors $x, y \in \mathbb{R}^n$, the **Root Mean Square Error** (RMSE) is an example of loss function defined as:

$$\mathcal{L}(x,y) := \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2 \tag{2.4}$$

The objective function is usually a loss function that we want to minimize. Sometimes it is called reward $\mathcal{R}$ and then we want to maximize it, but the procedure is identical. There are three main Machine Learning paradigms:

1. **Supervised learning**: the data comes in pairs (x, y). The goal is to learn to generate y from x, so the loss is of the form $\mathcal{L}(f(x), y)$ and it is a distance.
2. **Unsupervised learning**: the only available data is x, so the loss is of the form $\mathcal{L}(f(x))$. This means that we want to reach a specific goal but we do not have examples of good solutions.
3. **Reinforcement learning**: there is not any sequence of data to learn from. Instead there is an environment and an agent that can be controlled with actions a_i. At each step the model gets information about the environment and needs to do an action, with the goal of maximizing a cumulative reward $\mathcal{R}(a_1, ..., a_n)$. Reinforcement learning is the closest to how we humans and other animals learn.

Self-Supervised Learning combines Supervised and Unsupervised Learning. It learns to predict part of its input from other parts of the input. This is done by creating surrogate or pretext tasks, where the labels are generated automatically from the data itself. For example, predicting the next word from the previous words of a sentence or predicting the next frame in a video from the previous ones.

The process of **learning** in Machine Learning is the adjustment of the trainable parameters Θ in a way that the loss $\mathcal{L}$ is reduced for some sample of data, with the ultimate goal that it finds good patterns that enables it to generalize the given task and not just memorize the provided examples.

Definition 2.10. *An **optimizer** is an algorithm that modifies the learnable parameters Θ of a NN in a way that the loss function $\mathcal{L}$ is reduced.*

In supervised learning the data space $X \times Y$ tends to be infinite and have a lot of dimensions.

Let X' be a finite sample of X of size n that we will use to train our model. The goal of supervised learning is to find Θ parameters that produce a small loss for all $x \in X$, not only for $x \in X'$.

Definition 2.11. *We call **overfitting** when the loss in X' is greater than the loss in X. This means that the model has tended to memorize the examples X' that we have provided it and has not generalized the nature of the task that we wanted it to learn X.*

Definition 2.12. *We call **underfitting** when the loss in both X and X' is greater than a certain baseline which we consider unsatisfactory. This probably means that the model is not big enough to be able to capture the underlying patterns in the data.*

In order to avoid overfitting, the available data is usually split into three sets:

- **Train**: it is the split that is shown to the model.
- **Validation**: it is the split used to alter the training strategy when it starts to overfit, which can be seen when the loss reduction in this split plateaus or increases while the train loss keeps constantly being reduced.
- **Test**: this split is never used until the model has been completely trained.

A typical rule of thumb is to do a random split with relative frequencies 0.8, 0.1 0.1 for train, validation and test. This allows the model to learn from the majority of data while keeping a significant sample for the other splits.

So far, we have defined the data (which can be basically anything), the model (a NN) and the objective function (the loss) that we want to reduce. The only thing that remains is the algorithm to adjust the model's parameters Θ. There are alternative versions, but the most widely used optimizer is Stochastic Gradient Descent.

Let $g : \Omega \to \mathbb{R}$ be a differentiable function. Now, consider the optimization problem [27]

$$\underset{x \in \Omega}{arg\,min}\, g(x) \tag{2.5}$$

Let $x^0 \in \Omega$ be a random start point. **Gradient Descent** is the recurrence $k \geq 0$:

$$x^{k+1} = x^k - \eta \frac{\nabla f(x^k)}{\|\nabla f(x^k)\|} \tag{2.6}$$

where $0 < \eta \leq 1$ is called the **learning rate**

A NN is a composition of differentiable (almost everywhere) functions. **Backpropagation** [3] is a method that efficiently computes the derivatives of the loss function with respect to every trainable parameter Θ in the NN.

Stochastic Gradient Descent [2] is a stochastic approximation of the Gradient Descent optimization. Since X can be infinite and we do not have all the points, we choose a **batch** $Y \subset X$ and do one step of Gradient Descent using backpropagation to update all the trainable parameters. In this case, the loss function is defined for the whole batch.

Finally, **hyperparameters** are parameters that define the details of the learning process.

Definition 2.13. *An* ***epoch*** *is a complete pass of the model for all the training data.*

The **training** of a NN is the process of iteratively realizing epochs for a fixed number of times or until the model stops learning, which can be evaluated analyzing the evolution of the validation loss.

2.2 Recommender Systems

Let's start with formalizing the Recommender Systems.

Let $U = \{u_1, ..., u_n\}$ be a set of users and $I = \{i_1, ..., i_m\}$ a set of items. A user u is any entity that can make decisions, but typically humans. The items i can be anything that the user can interact with, usually some form of human generated creative content: YouTube videos, NetFlix shows, Instagram posts, anime series, books, movies, etc.

Definition 2.14. *An* ***interaction*** *is a tuple $\alpha = (u, i, f)$ with $u \in U$ the user, $i \in I$ the item and $f \in X$ the feedback. Notice that a user can make multiple different interactions with the same item.*

Let A be a set of interactions. The feedback can be anything: a numerical rating, a textual explanation, a video, an image, a timestamp, a utility value, a binary 1 indicator, or the combination of multiple of these things.

Definition 2.15. *A **Recommender System** (RecSys) is a system made with the aim to reduce the user's effort and time required for searching relevant items.*

There are two basic approaches [30]:

- **Content-based Filtering**: uses known properties of the items, like the title, duration, time of publication, etc. to group together similar items. Then it recommends to the user items that are similar to the ones that it has recently liked or interacted with.
- **Collaborative Filtering** (CF): uses instead data of interactions between users and items to extract information. It can be either implicit or explicit. Implicit data is when the system is tracking the user usage of a platform (like clicks on content, spent time), while explicit data is when the user manually inputs the actions to the system (like marking something as watched or rating a series in a website).

Usually, most modern Recommender Systems follow an hybrid of these two approaches. Collaborative filtering tends to give better results, but it suffers from the cold start problem for new users and items (it needs many users' interactions to be able to extract meaningful information about them).

In this project, we will use only Collaborative Filtering as it is simpler and having new users or items will not happen. CF is based on two assumptions:

1. Not all the users have the exact same tastes.
2. Taste of the users is not random, meaning that there are correlations in taste patterns between different users.

On the one hand, in relation to assumption 1, if we all were to be exactly equal, we could simply generate a list of the best items and every user should just interact with these ones. This is not the case and it would be a boring world to live in. However, popularity and average rankings can also provide meaningful information.

On the other hand, the second assumption is also a reasonable one. It is natural that if you coincide with a user in a lot of opinions, then their opinion of new items that you have not yet interacted with is more likely to be better aligned with your own than that of a random user.

There are two basic tasks for any Recommender System:

- Generate new (u, i) possible interactions called **candidates**
- Predict the feedback f, or a subset $g \subset f$ for these interactions (u, i)

RecSys are typically implemented from the view point of the user, as it is the one that can do the actions. This means that the goal of a RecSys is to do the previous tasks for a particular user u.

While the Internet has allowed the sharing of an absurd amount of information, humans still have a finite lifespan. Therefore, some kind of **ranking** is needed in order to only show to the user the better items. Theoretically, it could be simply sorting by the predicted rating if available, but there are other metrics like diversity and simply not interacting with the same thing over and over again which are also important.

Definition 2.16. ***Rating prediction*** *is when the subset of feedback g that is predicted is a utility function that represents the user's u opinion of the item i in a numerical way.*

In the particular case that there can be no duplicates in the data interactions (which is quite common), the real rating can be denoted with $R_{u,i}$ and the predicted rating $\hat{R}_{u,i}$. In this case we could save the ratings in a matrix $\mathrm{R} \in \mathbb{R}^{mxn}$ and then the rating prediction could be viewed as a matrix completion task. But this representation is not much useful in a real case, as it would occupy a lot of memory and take a lot of time to fill it. This would be an extreme case where the candidates where every single interaction, which is impractical.

In this project we are only interested in the motive behind each one of the predicted ratings. For this reason candidate generation and ranking will not be studied.

2.2.1 Collaborative Filtering

Collaborative Filtering in its most simple form is implemented with the composition of two *embeddings* with a MultiLayer Perceptron (MLP) [15].

Like previously let's consider U a set of users, I a set of items and A a set of interactions 2.14 with $X = \mathbb{R}$, meaning that the feedback is just a real value representing a utility value.

Definition 2.17. *Given E a set of entities, an* ***embedding*** *is a function $\varphi : E \rightarrow Z \subset \mathbb{R}^n$. Z is called the* ***latent*** *space. It is usually differentiable so that it can be trained with Stochastic Gradient Descent.*

Let $\varphi_u : U \to \mathbb{R}^n$ be an embedding for the users U, $\varphi_i : U \to \mathbb{R}^m$ an embedding for the items I. Let's define $X := \mathbb{R}^n \times \mathbb{R}^m$ the input, $\Theta \subset \mathbb{R}^d$ the parameter.

Then $f : X \times \Theta \to \mathbb{R}$ is a trainable MLP 2.7, with the embedding functions being part of the parameter. This means that we will also adjust these functions in a way that the final loss is minimized.

2.3 Natural Language Generation

Text understanding and especially text generation are complicate tasks for computers because they can only work with numbers and do numerical operations on them.

Therefore, before doing anything we will need to convert the Natural Language Generation (NLG) task into a mathematical process that computers are able to do. We will start with a formalization of the task.

2.3.1 Formalization

Definition 2.18. *A **letter** l is a symbol representing usually one of the sounds used in speech. It is a human construction and varies between different languages.*

Definition 2.19. *An **alphabet** A is a set of letters.*

Definition 2.20. *A **token** t in a certain alphabet A is a finite tuple of letters, $t = (l_1, ..., l_n), n \in \mathbb{N}, l_i \in A$. It is usually simply written with the letters concatenated and it is the smallest unit with meaning. It can refer to both concrete or abstracts concepts.*

Definition 2.21. *A **vocabulary** V is a set of tokens that a community has agreed upon the meaning of each one of them in order to be able to communicate between different individuals. Alongside a set of rules, it is usually called a **language**.*

Given a finite tuple of tokens $(t_1, ..., t_n)$ in a certain vocabulary V, the **next token prediction task** aims to predict the token t_{n+1} that would be a reasonable continuation to the sequence.

Definition 2.22. *A **next token value function** for a vocabulary V is a function $f : V^n \to \mathbb{R}^v$, where $v = \#V$ the size of the vocabulary and $n \in \mathbb{N}$.*

The values of each token in the vocabulary can be converted to probabilities using the softmax Equation 2.2 defined in Example 2.4. The next token prediction task can be treated as a composition of the next token value function and a softmax function. Once you have a probability distribution on all the tokens, several decoding strategies can be used.

Example 2.23. Greedy decoding chooses always the token with the greatest probability out of them all. If the NN is no longer trained, it leads to deterministic outputs.

Example 2.24. Top-k sampling selects the k values with the greatest probability. Then it randomly chooses one of these tokens by sampling their relative distribution. If $k = 1$, it would be greedy decoding.

Definition 2.25. *A model's* ***context window*** *is the maximum number of tokens that at a model can process and remember at any given time.*

With all of this, we can finally do the **text completion task** by simply repeating a next token prediction task until an special <eos> (End-Of-Sentence) token is generated or we reach the model's maximum context window.

This parameter is defined in the model's architecture and can't be changed. If we wanted to generate longer text without training the model again we could do a "text resume step" in order to compact the information, or simply ignore the text from long ago.

Example 2.26. ChatGPT 3.5 (the free version) has a context window of 4,096 tokens, while the paid one doubles it. Gemini has one of 128,000 tokens and Google is studying to enlarge it up to 1 million tokens.

With all of these definitions, we have formalized the text completion task. However, we have not yet designed a way on how can a computer learn to predict reasonable tokens. To convert text to numbers and give meaning to these numbers, a lot of steps need to be done:

1. **Tokenization**: break a text into smaller units
2. **Embedding**: convert the tokens to vectors
3. **Next token prediction**: define a way on how to generate the next token based on all the previous ones

2.3.2 Tokenization

The first step necessary for a computer to be able to work with any kind of text is to break it into smaller parts, the previously defined tokens 2.20.

The simplest form of tokenization is called word tokenization and it is just to break the text into individual words (splitting it by the space character). This is really easy to do, but if the text is not previously processed by uncapitalizing all the letters and splitting special characters and numbers from the words, it will produce a lot of vocabulary. For example, it would consider as completely different vocabulary: "Human", "human", "humans" and "human.". Having a lot of vocabulary is unproductive, because each of them is treated as a completely different concept and needs to be learned individually.

On the other extreme, we could split the text into single individual characters, solving the large vocabulary problem. But characters alone usually do not have meaning on its own and in this case we would be wasting time and resources on just trying to generate real words.

The better approach is to break the text into subwords, which can be either full words or fragments. The previous example on "humans" could be broken into <human> <_s>, where the <> symbols are usually used to denote the individual tokens. This way it would not interpret the plural of a word as a completely different thing, just the same root and an additional <_s> plural terminator.

This process of splitting the text into subwords is usually done by an already trained tokenizer algorithm, as any kind of text in a unique language tends to always follow the same patterns.

Example 2.27. BERT [17] uses WordPiece [13] with a vocabulary of 30,000 tokens. This is an algorithm that given a desired vocabulary size and a lot of text in a certain language, it learns which combinations of characters are most commonly used together and automatically learns to do the token partitions.

2.3.3 Token Embedding

Tokenizing a text is just to split it in smaller parts. Now we need to somehow give meaning to each of the tokens. As humans we interpret texts and vocabulary using concepts, an intellectual interpretation of abstract or real entities.

Nevertheless, computers can only work with numbers. Therefore, we need a way to convert the tokens to numerical values (vectors). This is done by defining a numerical vector space of a certain dimension where we will represent all of the tokens in the vocabulary. This numerical space may have a lot of dimensions, and the idea behind is that each dimension should represent a different latent factor of the tokens (like the type of token, whether it is plural or singular, etc), and similar tokens should have similar representations. This is called an *embedding* and it is similar to the Collaborative Filtering user and item embedding defined in 2.17.

If we convert the tokens to vectors, we can directly operate with the embeddings and apply math functions to them. There is a famous example that it is a bit of an exaggeration but is useful to picture what an embedding really is. If we denote the embedding function with ϕ, then:

$$\phi(king) - \phi(man) + \phi(woman) \approx \phi(queen)$$

It is also worth mentioning that the embedding function is not directly reversible, so if we were to do numerical operations within the latent space, then the resultant vector would probably not be exactly the representation of any token. This means that if we want to give meaning to an arbitrary vector in this latent space, "unembedding" it, we will possibly have to look for the tokens that have a closer representation to this vector using a certain distance, and depending on the goal you could reverse embed it to the most similar or the top-k most similar tokens.

This embedding function needs to be learned from human text. It is usually randomly initialized and then is iteratively adjusted by minimizing some loss that favors that similar words used in similar ways have close representations.

Instead of training the embeddings from random representations, we could use the representations learned by another model to initialize the weights. This is called **transfer learning**. There are monolingual models and multilingual models, depending on which languages they have been trained on.

2.3.4 Next token prediction

The previous steps are useful to give meaning to individual pieces, but we still need a way on how to build new pieces of text. As humans we think in ideas and concepts and use the language to express this thoughts in a structured manner.

For Natural Language Generation models, generating new text is simply reduced to the task or predicting the next token, based on all the previous ones until now. This could seem a bit of an oversimplification, but with a big enough architecture and a lot of training text it can lead to useful results. So the only task that all Natural Language Generation models do is just this, predicting the next token based an all the previous ones.

2.3.5 Attention & Transformers

Everything up to now has been simple and straightforward, so why did meaningful progress in Natural Language Generation not happen till recent years? The answer is just the design of a more parallelizable architecture that can benefit of having a lot of individual GPU's.

Until recently, the State of the Art models used Recurrent Neural Networks to modelate this Natural Language Generation task. Similar to how we humans read and write text, it interpreted tokens in a sequential manner. This meant that it took a lot of time for a computer to read an entire peace of text, and also lead to exploding or reducing gradients (which meant forgetting things [19]).

This new architecture is based on the attention mechanism [9] and its application in the Transformers Neural Networks [16]. This is a completely new approach on how to combine the knowledge of all previous information. Its architecture can be seen in Figure 2.2 below. It "reads" the whole text in parallel and with an attention mechanism it intelligently combines the information from previous tokens in order to generate the next token. This means for example, if sufficient GPU's are available, it can be trained it with a really long context. It could be seen as "reading" a book really fast by combining the knowledge of independent units into a single output.

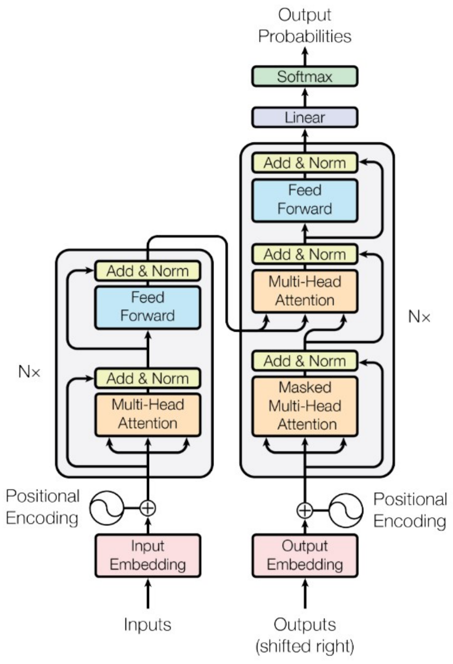


Figure 2.2: Attention architecture [16]

The only "flaw" that it has is that every "module" that reads a token (a query), in order to give meaning to this token it needs the whole context (keys), meaning that this method is inherently quadratic $O(n^2)$ in time, but if you were to have $O(n)$ GPU's then it would be only $O(n)$ time.

This architecture can be expressed with the equation:

$$Attention(Q, K, V) = softmax(\frac{QK^{\intercal}}{\sqrt{d_k}})V \tag{2.7}$$

where Q is the Query (the last token that we want to predict the next), K is the Key (all the previous tokens in the sentence) and V the value (the meaning). $\sqrt{d_k}$ is a normalization term with the size of the keys.

2.4 Combining Recommender Systems and Natural Language Generation

Once that both Recommender Systems and Text Generation have been explained, we need to combine both individual parts in a unique pipeline that will be aware of both the user, the item and the text tokens in order to be able to generate the personalized explanations for each user and item.

Training Large Language Models (LLM's) is a really time and resource intensive task, so before training anything from scratch it is worth analysing whether we can use existing models to base our model upon, as downstream tasks and finetunning models is really common in Natural Language Generation.

2.4.1 Directly using trained models

The simplest approach would be to directly use an existing LLM to analyze whether it is able to give decent results to our desired task. After all, ChatGPT has memorized hundreds of pages of popular books [33], so maybe it is capable of recommending items.

If the used model were to be closed source like ChatGPT then we would not be able to edit any of the parameters, so it is not possible to teach it anything that it has not already learned during its training. This means that the only choice for getting better results is making better prompts, which is more of an art than science. This has been studied in some previous work, but it tends to lead to poor and repetitive results which are not really personalized [35].

2.4.2 Finetunning pure text generation models

The next most simple approach would be to finetune an existing open source text generation model to generate better text for our desired task. The preferred choice for this approach is usually the model T5 [21], a text-to-text transformer trained by Google in 2020 that closely adheres to the original attention architecture that was presented in Figure 2.2.

P5 [29] is the usage of T5 in the particular case of Recommender Systems. It is a way on how to use T5 for Recommendation, alongside a collection of prompt templates that will be used to fine-tune the T5 model for recommendation. P5 stands for Pretrain, Personalized Prompt, and Predict Paradigm. A prompt is the text input that is fed to the model.

It is a framework finetuned on a multitask which is an ensemble of different recommendation tasks: rating prediction, sequential recommendation, explanation generation, review related and direct recommendation. Like T5, it is a pure text-to-text transformer that only knows to do one task: given a text, predict the next token:

$$\mathcal{L}_{\theta}^{\text{P5}} = -\sum_{j=1}^{|\mathbf{y}|} \log P_{\theta}\left(\mathbf{y}_j \mid \mathbf{y}_{<j}, \mathbf{x}\right) \tag{2.8}$$

This can lead to better results than directly using an already trained model because you can modify the weights to better do your desired tasks, but it still has a major drawback. The T5 architecture was designed as a pure text-to-text as it can be seen in Figure 2.3 below, so it models all possible tasks as pure text. This means that if we use it we can only pass it the exact same tokens that it was trained with. This means that like the ChatGPT approach, the model only naturally works with tokens.

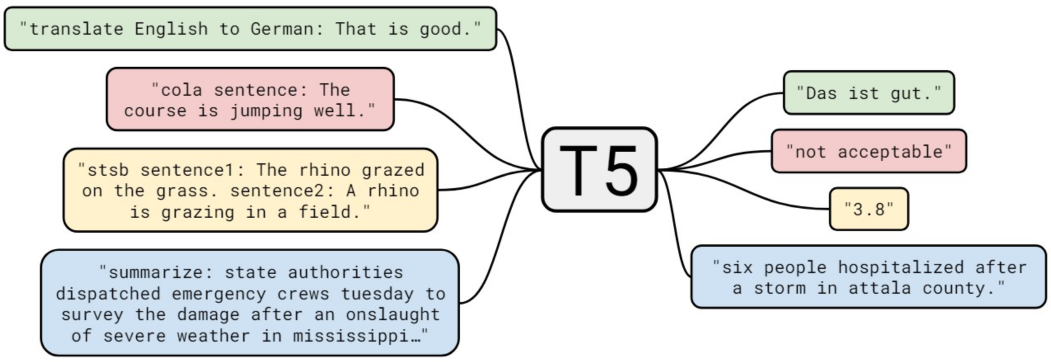

"translate English to German: That is good."
"cola sentence: The course is jumping well."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi…"
T5
"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."

Figure 2.3: Text-to-Text Transfer Transformer [21]

Therefore, we need to somehow insert the user and item in a vector space that was used to represent meaning of tokens. This is a bit weird since naturally users and items cannot be represented with any existing token.

The general approach to achieve this is by using some kind of handcrafted text, like "user_27" or "item_7391". The T5 tokenizer breaks it into $< user,_, 23 >$ and $< item,_, 73, 91 >$, and their proposed solution to let the model "understand" that they are unique entities is to put a whole word embedding so that the model can understand that it should be a unique word and have a representation of the user.

Another problem of this kind of approaches is that they have to directly deal with the hallucination problems, where the model generates text that it is not true. In particular it can even hallucinate items, which is quite weird for a real Recommender System to do.

Another problem of P5 is that it was slow on inference because a lot of words were needed to explain each desired goal, even though it was always doing the same few tasks that it was trained on.

POD's [34] solution was doing Prompt Distillation at the train step, meaning that it learnt directly the latent space representation of all the possible tasks as it can be see in Figure 2.4 and so it was then not needed to be computed again.

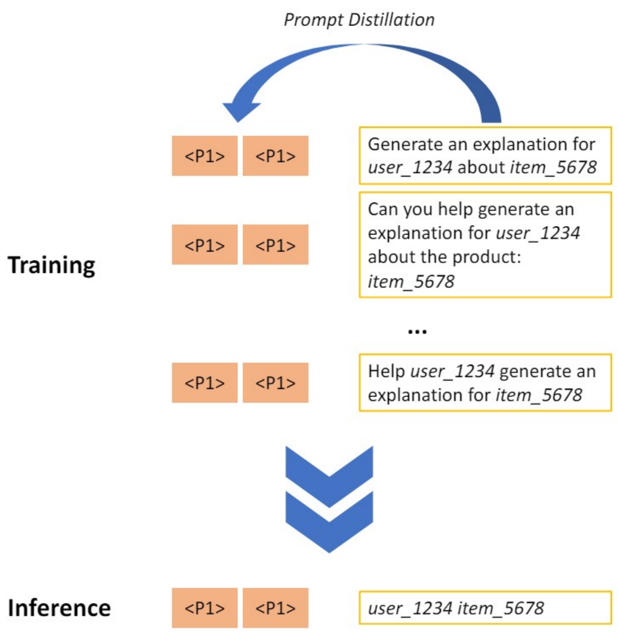

Prompt Distillation
Training
<P1> <P1>
Generate an explanation for user_1234 about item_5678
Can you help generate an explanation for user_1234 about the product: item_5678
...
Help user_1234 generate an explanation for item_5678
Inference
user_1234 item_5678

Figure 2.4: Prompt Distillation [34]

Initially POD algorithm was one of the chosen algorithms to compare and I had already spent some time understanding and executing the code, but there were a lot of downsides that ultimately made me discard this approach and center my efforts in another method that is one of the most used in the literature:

- When using the checkpoints that the authors published and doing inference on a Amazon toys dataset [10], it only had learnt to generate the same phrase 'I bought this for my 3 year old son for Christmas' for every single user and item.
- Even though I spent a lot of hours on it, I could not use the trained checkpoints of my local computer due to incompatibility problems and deprecated dependencies
- I estimated the time that it would take to reproduce the trainings that were done on the paper, and every single one of them was around 24 hours.
- I tried to train it in Google Colab as this was suggested to me as the preferred way of doing the training, but Google Colab's free version is only intended

for interactive usage and when using a lot of GPU it expels you quite frequently (every 3h when training but only 20 minutes when infering), which makes it impractical for doing exhaustive testing.

2.4.3 Training better suited architectures

A more efficient and natural approach is to define an architecture from scratch that knows how to represent both users, items and tokens. The only difference with respect to the previous ones (P5 and POD) is that it will differentiate between the three and use three different spaces to understand each one. This was pioneered by PETER [24] and will be the code with which I will base all my code.

SEQUER [32] is a PETER modification that was extended with an additional task of next-item prediction. Ultimately I decided not to include comparisons with SEQUER as my main goal was understanding PETER well and SEQUER complicated my task, but I still used it as an alternative code to help me better understand PETER's code.

Chapter 3

Analysis of PETER

In order to deeply understand how Self-Explaining Recommendation works, a major time focus on this project has been put into understanding, simplifying and extending an existing recent (2021) code base, PETER [24]. PETER stands for PErsonalized Transformer for Explainable Recommendation, which highlights the goal of the project: using a transformer architecture in order to generate textual explanations which are personalized.

3.1 Problem formulation

Like the notation for Recommender Systems, let U be a set of users and I a set of items. For a user $u \in U$ and an item $i \in I$, let $R_{u,i}$ be the rating that user u has given to item i. It's not necessary that all users have rated all the items. Ratings by users are usually integers in the range $[1, 5]$, but any other utility function could be used, even binary ones indicating whether there has been an interaction. Let $E_{u,i}$ be an explanation in natural language (text) on why user u has rated item i a score of $R_{u,i}$. An example case could be putting a textual review to something that a user has purchased, but any form on explanation could be used.

The goal of this project is to generate $\hat{R}_{u,i}$ a rating prediction on what user u would give to item i, alongside $\hat{E}_{u,i}$ a predicted explanation in natural language on why user u would give this item i this rating $\hat{R}_{u,i}$ if it were to interact with it.

3.2 Key step

The key step that PETER does to achieve personalization in the transformers architecture is to add a prefix of two extra elements in the source input, corresponding to user and item IDs. It's exactly the same as a regular transformer with a causal attention mask, only with the extra prefix of user and item that can always

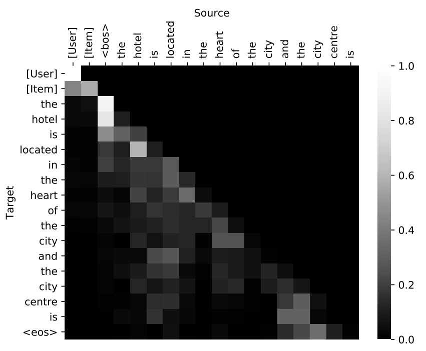


Figure 3.1: Standard Transformer model, where the user and the item have no contribution to each generation step

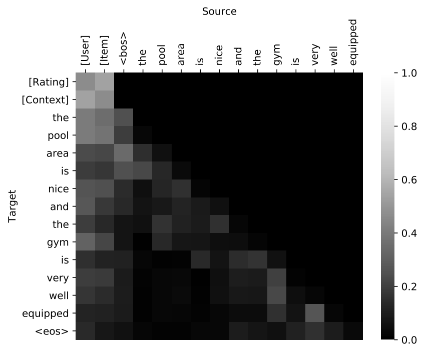


Figure 3.2: PETER model, where the user and item IDs play significant roles in the generation steps [24]

be attended to. In Figures 3.1 and 3.2 we can see the comparison on regular causal mask with the proposed PETER mask.

Python torch's implementation of the attention does not allow the attention mask to be non-square. For this reason, the target elements aligned with user and item need to be ignored in order to create a square matrix.

Instead of leaving two hidden useless states, PETER decides to use the last layer hidden[0] and hidden[1] states as input for a rating prediction task and a newly invented context prediction task, which is basically trying to predict the most likely words that will appear in the explanation in any order. This is used as some kind of hint of what will later the explanation say.

3.3 Multi-task learning

The main goal in PETER is to generate a natural language explanation, but also needs to predict the rating. All the tasks are done independently and then the loss is summed between all of them:

$$\mathcal{J} = \lambda_e \mathcal{L}_e + \lambda_c \mathcal{L}_c + \lambda_r \mathcal{L}_r \tag{3.1}$$

where λ_e, λ_c and λ_r are regularization parameters and $\mathcal{L}_e$, $\mathcal{L}_c$ and $\mathcal{L}_r$ the losses associated to the explanation, the context and the rating prediction.

At each step it performs all three tasks, computes their respective losses and aggregates them, effectively learning to do them at the same time.

3.3.1 Rating prediction

It uses the hidden[0] state to train a Multilayer Perceptron (MLP) in order to predict the rating. This hidden state attends to both the user and item embedding. This MLP is just a simple Feed Forward NN that can be expressed as:

$$y = \sigma(xW_1)W_2$$

where y is the predicted rating, x is the $hidden[0]$ state that has information relative to the user u and item i, σ is a sigmoid activation, W_1 is a emsize x emsize weight matrix, W_2 a (emsize x 1) matrix and *emsize* an hyperparameter which means embedding size and is usually 512.

It is trained with the Mean Square Error as loss function:

$$\mathcal{L}_r = \frac{1}{|\mathcal{T}|} \sum_{(u,i)\in\mathcal{T}} (r_{u,i} - \hat{r}_{u,i})^2 \tag{3.2}$$

where $\mathcal{L}_r$ is the loss associated to the rating, $\mathcal{T}$ is the set of pairs of user u, item i interactions, $r_{u,i}$ the real rating and $\hat{r}_{u,i}$ the predicted rating.

3.3.2 Text prediction

The text prediction task is implemented as a classification problem over all the vocabulary tokens. For this reason, it is trained with the Negative Log-Likelihood (NLL) loss function:

$$\mathcal{L}_c = \frac{1}{|\mathcal{T}|} \sum_{(u,i)\in\mathcal{T}} \frac{1}{|E_{u,i}|} \sum_{t=1}^{|E_{u,i}|} -\log c^{e_t} \tag{3.3}$$

where $\mathcal{L}_c$ is the loss associated to the context, $E_{u,i}$ the explanation for user u and item i.

3.4 Architecture

PETER's architecture, as can be seen in Figure 3.3 in the next page is a transformer with L layers. The first elements of the input correspond to the contextualized embedding of user and item, and all the layers can attend to it.

The output of the first neuron of the L-th transformer layer is used to predict the rating. The second one is used to predict the context (the most probable words of the review in any order). The next ones are used to decode the explanation.

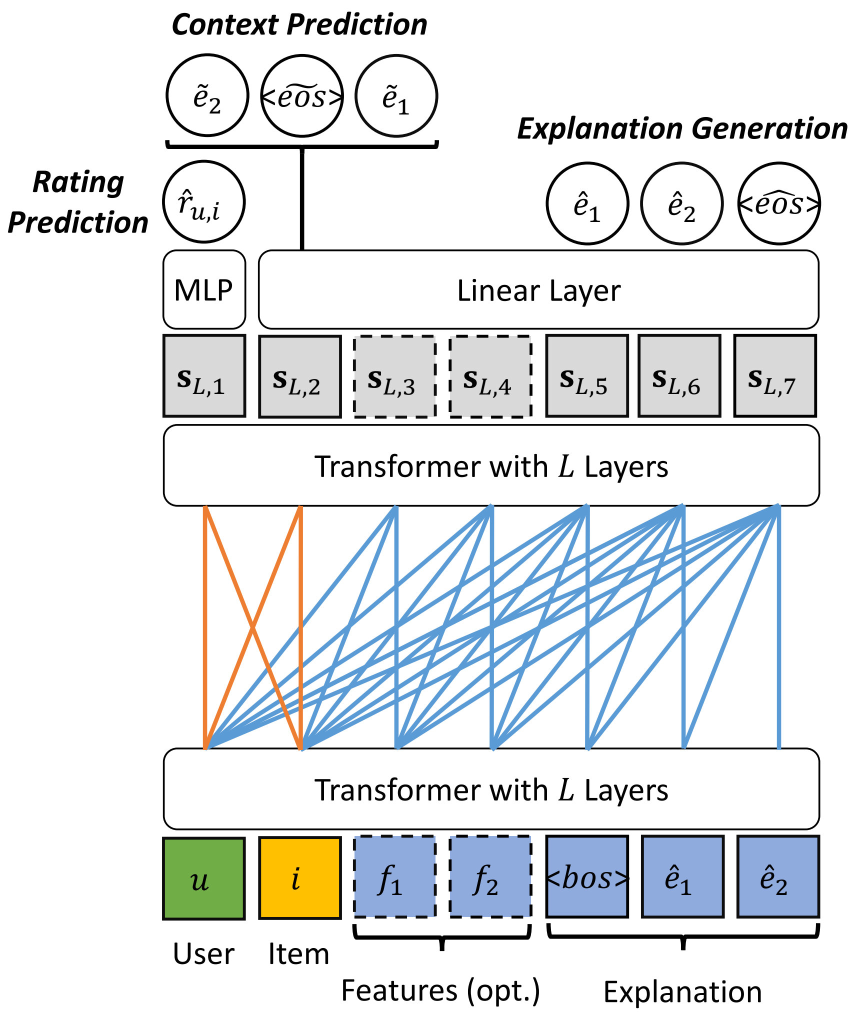


Figure 3.3: PETER's architecture [24]

Chapter 4

Extensions to PETER

The goal of this project was to combine Recommender Systems and text generation into a unique pipeline that was able to learn to do both tasks together.

The original PETER's [24] architecture was really clever, but the provided source code was a bit messy and not really well organized.

To understand all the important parts I simplified, edited and reorganized a lot of the code. Ultimately, I defined my own pipeline to train a personalized transformer for any dataset, and released it publicly in GitHub at andreu-vall/tfg.

My implemented pipeline consists of the following steps:

1. **create**: create a dataset with columns user, item, rating and text
2. **split**: make a train/validation/test split, usually (0.8, 0.1, 0.1)
3. **tokenize**: split the text using a certain tokenizer
4. **train**: train a PETER model
5. **test**: evaluate fast metrics
6. **generate**: generate new text using a certain strategy
7. **analysis**: human analysis of the text quality

In the following sections I will detail only the relevant functional changes with respect to the PETER original code.

4.1 Tokenize

PETER only used the word tokenizer, because for their experiments they used a dataset that had already been processed with another framework. But if we were to use raw human text, it is better to use a custom tokenizer rather that just word splitting. If there is a lot of vocabulary, the code takes more than double the time to train, probably because it is harder to converge to useful representations. And if you directly limit the vocabulary to a certain size, then a significant percentage of the text might be lost.

Instead of limiting the vocabulary size, it is better to use a more complex tokenizer. In my code, I used the bert-base-uncased tokenizer [17] from HuggingFace Transformers [22]. Any other tokenizer could be used with just specifying its HuggingFace name, but it was not a goal of this project to compare a lot of tokenizers.

To be able to compare results with PETER, I also added the option to do the space tokenization.

4.2 Transfer learning

Transfer Learning and downstream tasks are a common approach in Deep Learning and Natural Language Generation. If one trained model has already gained knowledge on a task, it is natural to take advantage out of it instead of always having to learn everything from scratch.

When using an existing tokenizer we can also load the trained embeddings for the tokens as a starting point for our token embeddings, rather than the PETER's random initialization. I added this option in my framework for the bert-base-uncased model, and exactly as before any other HuggingFace model could be used.

There was one slightly inconvenience though. PETER typically used an *emsize* of 512, but BERT's embedding size was 768. I wanted to keep my *emsize* to 512 for two reasons, to be able to compare results with PETER and because making the *emsize* 768 nearly duplicated the execution time. As I have limited resources I decided to cut the BERT's embedding size to 512.

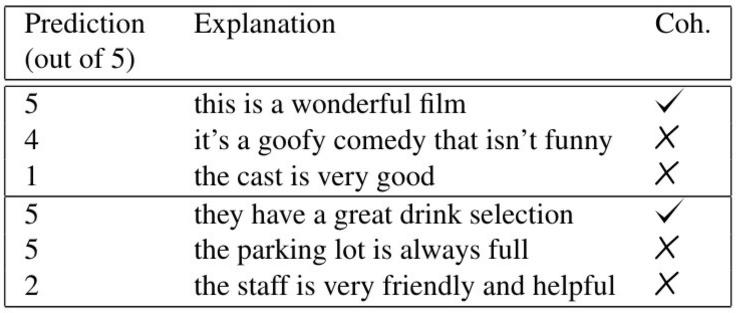
The better approach would have been to apply a PCA (Principal Component Analysis) algorithm to reduce it, but as this was not a key goal of the project I opted for simply truncating the BERT's size to the first 512 dimensions.

4.3 Train

4.3.1 Rating prediction

PETER used the first hidden output of the last layer to train their recommender, which attended to user and item.

An intereseting and more natural option would have been to follow the typical neural collaborative filtering approach and directly give the embeddings of user u and item i to the NN. Therefore, I added an option that decides which is the source for the MLP. For the PETER's approach it is the $hidden[0]$, the first element of the last layer of the transformer with size *emsize*. The other approach is by using $x = Concat(u, i)$, with u the user embedding and i the item embedding, with size



Prediction (out of 5)	Explanation	Coh.
5	this is a wonderful film	✓
4	it's a goofy comedy that isn't funny	✗
1	the cast is very good	✗
5	they have a great drink selection	✓
5	the parking lot is always full	✗
2	the staff is very friendly and helpful	✗

Figure 4.1: PETER's lack of coherence [36]

$2 * emsize$. The MLP architecture is exactly the same, only doubling the input size.

4.3.2 Better alignment between rating and explanation

One of the main problems of PETER that CER [36] described was a lack of coherence between the predicted rating and the generated text. As it can be seen in Figure 4.1, it might predict a low rating and generate a very positive explanation, or predict a high rating while generating a negative explanation. This is indeed a problem because it completely undermines the utility of the generated explanations.

Their solution was to train a rating predictor from text. Then, once a text prediction was made, it computed a rating. Furthermore, there was an additional loss that penalized the misalignment of the predicted rating from user and item and this other one, which hopefully meant that rating predictions and explanation generation should be more coherent.

My suggestion to solve this problem was simpler. I created an additional embedding for rating and put it as a prefix. This means that all the tasks that generate text will be aware of the rating that it is generating it for. In the inference step, it is exactly identical to the autoregressiveness of typical Natural Language Generation. In a previous execution I predict a rating, and after that I feed this predicted rating to my model.

4.3.3 Train strategy

PETER decayed the learning rate by a factor of 0.25 when the epoch validation loss was worse than the best obtained validation loss.

My hypothesis is that if you wait that long the model already tends to overfit to the training dataset. The improvements on the training loss tend to be constant while the validation loss starts to plateau, maybe marginally improving.

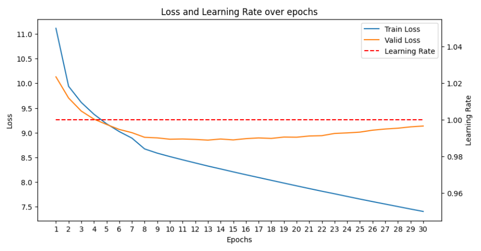


Figure 4.2: Fixed learning rate

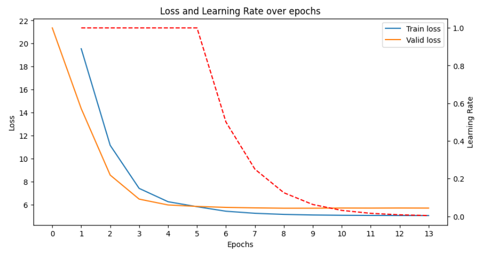


Figure 4.3: Decaying learning rate

My proposed change is to start decaying the learning rate sooner. One way could be to decay the learning rate when the validation loss percentage improvement was lower than a certain percentage, usually 5%, as can be seen in Figures 4.2 and 4.3.

4.4 Text generation

PETER only used greedy decoding to generate text. I also implemented top-k sampling, which was defined on Example 2.24 and it was simply to select the token by sampling the top-k tokens.

Beam search [14] could have also been implemented with a bit more work but I think that the usual approach of implementing it would only generate N beams pretty much identical. Diverse beam search [12] could have been an interesting alternative approach to try, but ultimately this was not one of the main goals of the project and it would complicate the metrics analysis.

Chapter 5

Experiments

In this Chapter some experiments on real data will be made in order to validate the usefulness of the framework and the effect of the proposed extensions.

Some of the things that will be tested are the following ones:

- Text source that is fed to the model
- Different tokenizers
- The effect of transfer learning

5.1 Dataset

In order to train a collaborative filtering Recommender System to predict ratings we need a dataset with users, items and ratings. If we also want to learn to generate human-like explanations on why this item is or is not relevant to the user, we will also need human generated reviews on why they liked or disliked the items.

We could use any dataset with (user, item, rating, text), but as scraping new data is not the goal of this project and to be able to compare results with other papers, we will use standard datasets that are widely used in research.

The Julian McAuley Amazon 2014 product data [10] is an unofficial 1996-2014 Amazon scrape of publicly available data by the J McAuley team. It is split by categories and filtered by 5-core (only users and items with at least 5 reviews are kept). To validate my results on different domains, I will use three categories of this dataset: *Beauty*, *Sports and Outdoors* and *Toys and Games*. Details of the datasets are shown in Table 5.1 below.

Dataset	Beauty	Sports	Toys
#Users	22,363	35,598	19,412
#Items	12,101	18,357	11,925
#Reviews	198,502	296,337	167,597
#Reviews/user	8.88	8.32	8.63

Table 5.1: Description of the datasets

5.2 Text quality metrics

Computers are yet far to be able to generate really useful or creative text, and automatic metrics cannot really accurately measure the quality of new text. The only way to really see if text generated by computer might be really useful to a human is to manually read it and evaluate it.

However, since the datasets are very large and human evaluation is subjective, there are some common metrics which are used to get approximations on the quality of the generated text. In this thesis, we will use two objective metrics that measure how similar a certain text is to an expected baseline, which is the real text that we would hope our model to produce something similar to.

BiLingual Evaluation Understudy (BLEU) is a measure that was invented to evaluate automatically translated text by comparing it with good quality translations produced by humans. It is a ratio between 0 and 1, with 1 meaning that it is exactly the same translation and 0 totally different. It is not limited to translation and can be used to compare any two texts. It is computed with N-grams, sequences of N consecutive words. This means that BLEU-N compares N-grams in the candidate text to N-grams in the reference text. In this project we will use BLEU-1 (B1) and BLEU-4 (B4), as they were the metrics that the original PETER code used.

Recall-Oriented Understudy for Gisting Evaluation (ROUGE) is another measure used to evaluate automatic summarizing and machine translation software. It is also a ratio between 0 and 1. Similar to BLEU it compares N-grams, but instead of pure precision is most focused on recall (how much of the information is preserved).

Text diversity metrics can also be used to evaluate the model, because if it generates almost exactly the same phrase for all the users and items it will probably not be very helpful in a real scenario. It will be measured with Unique Sentence Ratio (USR), the ratio of uniquely generated sentences.

5.3 Text source and tokenizer

Even with a well designed architecture, it is really difficult for a computer to be able to generate text directly from almost nothing without any kind of hint of what it should say.

As I explained before, the usual way to train a computer to generate text is by learning to do the task of next token prediction. This works really well for text summarization and question answering, as the model has already a lot of context of exactly what the user wants to be generated, but it falls short for tasks that require a little bit of reasoning.

So if we try with our transformers architecture to imitate the reviews without any kind of "thinking about what I'm going to say" step, the model will generate the most generic explanations and will not really generate personalized explanations to the user and the item, failing to achieve the task that we wanted to do.

For this reason, instead of directly trying to predict real reviews from users, PETER [24] used an automatic toolkit [20] [8] that extracts tuples of (feature, opinion, sentence, sentiment) of the reviews, and instead of trying to predict real reviews the task was reduced to producing these sentences. For an example of what it might extract, see Table 5.2.

Feature	**Opinion**	**Sentence**	**Sentiment**
hair feel	thicker	This stuff makes your hair feel thicker after ...	1
fragrance	light	you'll just have a light fragrance of some ind...	1
dish soap	lemon	but it kinda smells like lemony dish soap or l...	-1
scent	strong	It is not strong scented	-1

Table 5.2: Sentires toolkit example with Beauty dataset

This was generated by a completely different code, takes quite some time to be extracted and it is far from being factually correct, but even then it simplifies the explanation task so much that then the model produces things that are a bit more close to being remotely useful.

If we wanted our model to achieve better results we would need to improve this step of extracting knowledge from the text, as before learning to generate new things we need to understand exactly what we want to generate and put it in a simple way so that the model can learn how to generate it.

All tasks in Table 5.3 below are trying to predict the first 10 tokens from each source and all parameters are exactly the same as PETER [24] used. The most relevant ones are batch size of 128 and vocabulary size of 20,000 (for the space

Source	**Tokenizer**	**USR**	**B1**	**B4**	**ROUGE**
review	space	0.5	14.8	1.74	15.8
review	bert	1.2	<u>17.7</u>	**2.33**	**19.1**
summary	space	0.7	5.71	0.46	5.58
summary	bert	1.1	3.93	0.72	8.43
sentence	space	0.9	15.8	1.10	16.1
sentence	bert	1.2	16.3	1.39	<u>16.7</u>
feat+sent	space	<u>1.4</u>	**17.8**	<u>1.68</u>	16.5
feat+op+sent	space	**1.6**	15.5	1.35	15.5

Table 5.3: Text source and tokenizer

tokenizer, the bert tokenizer already has a much smaller vocabulary). All ratios are expressed as percentages, BN stands for BLEU-N, USR Unique Sentence Ratio and the ROUGE value corresponds to the 1-gram F1 score. The best performing values are boldfaced and the second best underlined.

Even if the automatic quality metrics from one source are greater than others, not all text sources are equally useful from a user perspective. Predicting the whole review would give the most value, as it is the most detailed explanation of the user opinion. But when restricted to only the first 10 tokens, predicting the summary would be more useful, as with a few words it usually expresses the whole review opinion.

The feature (feat), opinion (op) and sentence (sent) is a way that PETER used to simplify the user reviews so that the model can learn to generate short simple sentences. The original code only used the feature as additional input in order to have more idea on what the sentence should talk about. While this made it generate better sentences, it was at the cost of having to know this hint beforehand before even being able to generate any phrase.

This makes the generation of explanations without telling the model explicitly what should the review be about impossible, making it useless for the goal of explanation generation. To avoid this, I treated the feature and opinion as if they were simply part of the text goal to produce. This way, it is like the model learns to tell you the token that this review will comment on, then the general opinion and then it generates the full sentence that would correspond to the expected review.

If we were to analyze our results only in these metrics, it would seem that the source review plus bert tokenizer performs the best. But only comparing automatic metrics is a naive approach. To really see the usefulness of each approach, it is a better approach to read the outputs and judge whether the produced text would be a useful explanation of why would a user like or not the item.

Source	Tokenizer	Example output
review	space	I love this product. It is a great product
review	bert	i love this product . i have been using it
summary	space	Great for the price
summary	bert	great for dry skin
sentence	space	I have very fair skin and this is a great
feat+sent	space	hair I have very thick hair and this product is
feat+op+sent	space	hair dry I have dry hair and this is not

Table 5.4: Real examples for each source and tokenizer

With Table 5.4 we can extract more meaningful conclusions. If the model is trained with only the first words of the reviews, it tends to simply generate really generic comments without giving any useful information. The sentences produced using the toolkit are more useful and give concrete reasons, but it would not be feasible in real scenarios and still gives generic reasons. The text trained with the summaries seems to be the most useful of them all, as it tends to give a better approximate of the real reasons the users would give.

Therefore, from now on all successive tests will be done exclusively with the summary and bert tokenizer.

5.4 Transfer learning

Once a text source and tokenizer have been selected, we can study the effect of transfer learning on the token embbeddings by initializing them from random values or using the learned embeddings of an already trained model as a starting point.

We could also use the learned embeddings of one dataset as starting point to another dataset, but this would lead to a lot of combinations. The key point is simply that the token embeddings are transferible and maybe we can speed up the training by re-using learned token meanings.

Applying transfer learning to the embeddings of the users and items would be much more difficult. We would need somehow to have some correspondence between the old entities and the new ones, and this is by no means a trivial task.

If we had an already trained model and would like to add new users and items, we could reuse all our weights and simply continue the training with these new entities.

Dataset	**Weights**	**USR**	**B1**	**B4**	**ROUGE**	**RMSE**
beauty	random	**1.05**	**3.93**	**0.72**	**8.43**	1.14
beauty	bert	0.61	3.46	0.54	7.82	**1.15**
sports	random	**1.99**	**6.24**	**1.13**	**10.3**	0.97
sports	bert	0.38	4.25	0.93	8.79	**0.99**
toys	random	**0.76**	**6.03**	**1.03**	**10.6**	**0.98**
toys	bert	0.46	5.65	0.88	9.99	**0.98**

Table 5.5: Transfer learning in token embeddings

With the obtained results of Table 5.5, it seems that training the embeddings from random values gives better results than starting with already trained ones. This is a bit weird since apparently it would seem that having already an understanding of the general meaning of words would be better. This might be caused because the used text length is really small (<= 15) and, in this case, what matters the most are some only really important words, so starting the weights from random values seems to better fit this task. The difference in the rating prediction tasks seems to be much smaller.

Chapter 6

Conclusions

Generation of text is a complicate task, and even if recently computers are starting to be able to generate text on the spot that can be helpful, they are still far from really understanding the text or being able to create anything remotely creative.

The only one advantage that they have with respect to humans is that they can process a lot of data in little time. In the following years, Artificial Intelligence will probably be able to automate some repetitive tasks, but it will probably not meet a lot of the expectations of the current hype, as the main building blocks and architectures are still very rudimentary.

Despite the lack of creativity, there is still a lot of room for improvement and even with today's simple architecture they can be really useful and provide real value in some tasks.

It has been a bit of a surprise to me that even with the exact same architecture, if the text data is not simplified to an easy format, it is more difficult for the model to learn to do the desired task, similar to how you could think when a human needs to learn to say the first words: he needs really simple words and sentences.

It is also worth mentioning that Recommender Systems are completely dependant of the architecture and data that they were trained on. This understanding of the text could also be used to directly manipulate with text instructions the recommended items, so having available open source models and studying them is really necessary.

My personal biggest accomplishment has been the understanding and development of a framework that could be further tested and analyzed to try to give more human meaning to decisions made by Recommender Systems.

It has also made me better realize that current State-of-the-Art Machine Learning is really controllable and completely dependant of the data and objective that it was trained with, so we should be really careful of platforms that collect our data and be more critical with things that can influence important life decisions.

The time distribution of the project could have been a bit better. Since the start, I had the area of research quite well defined, but I was not completely sure about my goals, which made me spend time on understanding and contextualizing important things, but that were not directly part of this written thesis.

In relation to the goals of the project all of them have been mostly resolved. I was able to understand, formalize and extend a well-known paper and code on explainable recommendation, even cleaning up the code and trying other alternatives. In the near future, I will continue developing my framework and apply and analyze it to other domains.

Bibliography

[1] Charles Darwin. On the origin of species by means of natural selection, or preservation of favoured races in the struggle for life. 1859.

[2] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[3] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[4] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[5] Stephen D Houston. The first writing: Script invention as history and process. 2004.

[6] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[8] Yongfeng Zhang, Haochen Zhang, Min Zhang, Yiqun Liu, and Shaoping Ma. Do users rate or review? boost phrase-level sentiment labeling with review-level sentiment classification. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 1027–1030, 2014.

[9] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[10] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pages 43–52, 2015.

[11] Yan-Yan Song and LU Ying. Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry*, 27(2):130, 2015.

[12] Ashwin K Vijayakumar, Michael Cogswell, Ramprasath R Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. Diverse beam search: Decoding diverse solutions from neural sequence models. *arXiv preprint arXiv:1610.02424*, 2016.

[13] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[14] Markus Freitag and Yaser Al-Onaizan. Beam search strategies for neural machine translation. *arXiv preprint arXiv:1702.01806*, 2017.

[15] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[18] Dirk L Hoffmann, Christopher D Standish, Marcos García-Diez, Paul B Pettitt, James A Milton, João Zilhão, Javier J Alcolea-González, Pedro Cantalejo-Duarte, Hipólito Collado, Rodrigo De Balbín, et al. U-th dating of carbonate crusts reveals neandertal origin of iberian cave art. *Science*, 359(6378):912–915, 2018.

[19] M Mattheakis and P Protopapas. Recurrent neural networks: Exploding vanishing gradients & reservoir computing. In *Advanced Topics in Data Science*. Harvard Press Cambridge, MA, USA, 2019.

[20] Lei Li, Yongfeng Zhang, and Li Chen. Generate neural template explanations for recommendation. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 755–764, 2020.

[21] 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. *Journal of machine learning research*, 21(140):1–67, 2020.

[22] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

[23] Yongfeng Zhang, Xu Chen, et al. Explainable recommendation: A survey and new perspectives. *Foundations and Trends® in Information Retrieval*, 14(1):1–101, 2020.

[24] Lei Li, Yongfeng Zhang, and Li Chen. Personalized transformer for explainable recommendation. *arXiv preprint arXiv:2105.11601*, 2021.

[25] Alexandra Vultureanu-Albişi and Costin Bădică. Recommender systems: An explainable ai perspective. In *2021 International conference on innovations in intelligent systems and applications (INISTA)*, pages 1–6. IEEE, 2021.

[26] Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

[27] Rafael Beaus Iranzo. Article similarities using transformers. Mathematics and Computer Science Final Degree Thesis, 2022.

[28] Oxford Economics. The state of the creator economy – assessing the economic, cultural, and educational impact of youtube in the us in 2022, 2022.

[29] Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. Recommendation as language processing (rlp): A unified pretrain, personalized prompt & predict paradigm (p5). In *Proceedings of the 16th ACM Conference on Recommender Systems*, pages 299–315, 2022.

[30] Deepjyoti Roy and Mala Dutta. A systematic review and research perspective on recommender systems. *Journal of Big Data*, 9(1):59, 2022.

[31] Qazi Mohammad Areeb, Mohammad Nadeem, Shahab Saquib Sohail, Raza Imam, Faiyaz Doctor, Yassine Himeur, Amir Hussain, and Abbes Amira. Filter bubbles in recommender systems: Fact or fallacy—a systematic review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 13(6):e1512, 2023.

[32] Alejandro Ariza-Casabona, Maria Salamó, Ludovico Boratto, and Gianni Fenu. Towards self-explaining sequence-aware recommendation. In *Proceedings of the 17th ACM Conference on Recommender Systems*, pages 904–911, 2023.

[33] Kent K Chang, Mackenzie Cramer, Sandeep Soni, and David Bamman. Speak, memory: An archaeology of books known to chatgpt/gpt-4. *arXiv preprint arXiv:2305.00118*, 2023.

[34] Lei Li, Yongfeng Zhang, and Li Chen. Prompt distillation for efficient llm-based recommendation. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 1348–1357, 2023.

[35] Junling Liu, Chao Liu, Peilin Zhou, Renjie Lv, Kang Zhou, and Yan Zhang. Is chatgpt a good recommender? a preliminary study. *arXiv preprint arXiv:2304.10149*, 2023.

[36] Jakub Raczyński, Mateusz Lango, and Jerzy Stefanowski. The problem of coherence in natural language explanations of recommendations. *arXiv preprint arXiv:2312.11356*, 2023.

[37] Adrián Saiz de Pedro. Learning with errors: a lattice-based cryptosystem. Mathematics and Computer Science Final Degree Thesis, 2024.