UNIVERSITY OF AMSTERDAM

MSc Artificial Intelligence
Master Thesis

Using recurrent neural networks to predict customer behavior from interaction data

by
Daniel Sánchez Santolaya
11139005

July 7, 2017

36 EC
February 2017 - July 2017

Supervisor:
Dr Evangelos Kanoulas

Assessor:
Dr Efstratios Gavves

University of Amsterdam

University of Amsterdam

Abstract

MSc Artificial Intelligence

Using recurrent neural networks to predict customer behavior from interaction data

by Daniel Sánchez Santolaya

Customer behavior can be represented as sequential data describing the interactions of the customer with a company or a system through the time. Examples of these interactions are items that the customer purchases or views. Recurrent Neural Networks are able to model effectively sequential data and learn directly from low-level features without the need of feature engineering. In this work, we apply RNN to model this interaction data and predict which items the user will consume in the future. Besides exploring how effective are RNNs in this scenario, we study how item embeddings can help when there is a high quantity of different items, providing a comparison and analysis of different methods to learn embeddings. Finally, we apply attention mechanisms to gain interpretability in the RNN model. We compare different variants of attention mechanism, providing their performance and the usefulness to explain the predictions of the model.

Acknowledgements

I would like to thank Dr. Evangelos Kanoulas for supervising this thesis. His guidance and support have been very helpful and it allowed to me to learn a lot during this research. I also want to thank Ivar Siccama and Otto Perdeck from Pegasystems for their feedback and continuous motivation. Finally, I want to thank Dr. Efstratios Gavves for taking the time to read the thesis and to participate in the defense committee.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Customer interactions like purchases over time can be represented with sequential data. Sequential data has the main property that the order of the information is important. Many machine learning models are not suited for sequential data, as they consider each input sample independent from previous ones. In contrast, given an input sequence, Recurrent Neural Networks are able to process each element at a time, storing the necessary information of each element. Therefore, at the end of the sequence they keep in their internal state information from all previous inputs, making them suitable for this type of data. Moreover, as other Neural Network models, RNNs are able to capture information from very low-level features. Thus, they are a good alternative to machine learning models that work with hand-crafted features that include aggregated information from all the sequence.

1.1 Problem and motivation

Predict future customer behavior is an important task to offer them the best possible experience and improve their satisfaction. A clear example is observed in e-commerce systems, where users can avoid searching through a large catalog of products if they have a set of recommended products that they are interested in.

Consumer behavior can be represented as sequential data describing the interactions through the time. Examples of these interactions are the items that the user purchases or views. Therefore the history of interactions can be modeled as sequential data which has the particular trait that can incorporate a temporal aspect. For example, if a user buys a new mobile phone, he might purchase accessories for this mobile phone in the near future, or if the user buys a book, he might be interested in books by the same author or genre. Therefore, to make accurate predictions is important to model this temporal aspect correctly.

A common way to deal with this longitudinal data is to construct hand-crafted features which aggregate information from past time steps. For example, one could count the number of purchased products of a particular category in the last N days, or the number of days since the last purchase. Creating several hand-crafted features produces a feature vector which can be fed into a machine learning algorithm such as Logistic Regression.

Although good results can be achieved with this methodology, it has several drawbacks. First, part of the temporal and sequence relationships is ignored. Even though we include features containing information from past interactions is practically impossible to include all the information contained in the raw data. Only signals that are encoded in these features can be captured by the prediction models. Normally domain experts are needed in order to find good hand-crafted features that help to obtain a good prediction accuracy. Second, normally there is a huge set of candidate hand-crafted features to be created. Data scientists can spend a lot of time designing

and testing new features, which many of them lead to no improvement in prediction performance. Even if they can get improvements, it is hard to know if the actual set of hand-crafted features is optimal to the problem, so the process of testing and adding new hand-crafted features never stops, or stops when the algorithm reaches an acceptable level of performance which could be far from the true potential. Third, in some cases computing the hand-crafted features can lead to an expensive preprocessing of the data. For example, if we use a feature with the number of product views of a particular category in the last N days, we would need to update the value of the feature each day or compute it online every time that we need to make a prediction. When there are many features and many users, this process could be computationally expensive. Finally, different data sets may need different hand-crafted features, so for every dataset there is the need to find new hand-crafted features.

With these disadvantages, is reasonable to find ways to effectively model the temporal aspect of this interaction data. Some alternatives to model the sequential data taking into account the temporal aspect are Markov chain based models [20, 40] or Hawkes process [26, 25]. The problem with these approaches is that they can be computationally expensive when trying to capture long-term dependencies, especially in non-linear settings.

With Deep Learning receiving a lot of attention the last years, a new approach to model sequential data has been explored. Recurrent Neural Networks (RNN) have been very effective for learning complex sequential patterns, as they are capable to maintain a hidden state updated by a complex non-linear function learned from the data itself. They are able to capture information about the evolution of what happened in previous time steps. In the last years, RNNs have achieved the state of the art in problems like Language Modeling, Speech Recognition, Machine Translation or Handwriting recognition [22, 28, 39, 17]. These tasks share some similarities with the problem of predicting future actions from past interaction data, in the sense that the data is represented sequentially. For example, in language models a sentence can be represented as a sequence of words.

Due to the effectiveness of RNNs in these tasks in Computer Vision and Natural Language Processing, some research has been done to use them in other areas. A field where RNN have received a lot of attention recently is health care [2, 9, 21]. Although there are some differences between these works, the common approach is model the sequence of medical conditions of a patient using RNN to predict future medical conditions. Another area where RNN have been shown to be successful recently is recommender systems [27, 4, 41, 6].

Figure 1.1 shows the difference of the RNN approach versus the hand-crafted approach. As we see, the hand-crafted approach creates some features looking at the historical data. On the contrary, in the RNN approach we feed the raw data directly to the neural network, so the model learns all the knowledge by itself.

A direct approach to represent the interactions is to use one-hot encoding vectors, where each position of the vector encodes a different interaction. For example, if we are representing the interactions of buying different items, we can represent them with a one-hot encoding vector with one position per item. Therefore, its dimensionality grows with the number of items or interactions. As the one-hot encoding vector is connected to the RNN, the number of parameters to learn grows with the dimensionality of the vector. Consequently, if we have many different items we need more data to avoid overfitting and learn a model which is able to make accurate predictions. Furthermore, when the dimensionality of the input is very high, the model needs to compute high-dimensional matrix multiplications for every element of the sequence, which can have a computational impact when dealing with long sequences. Another

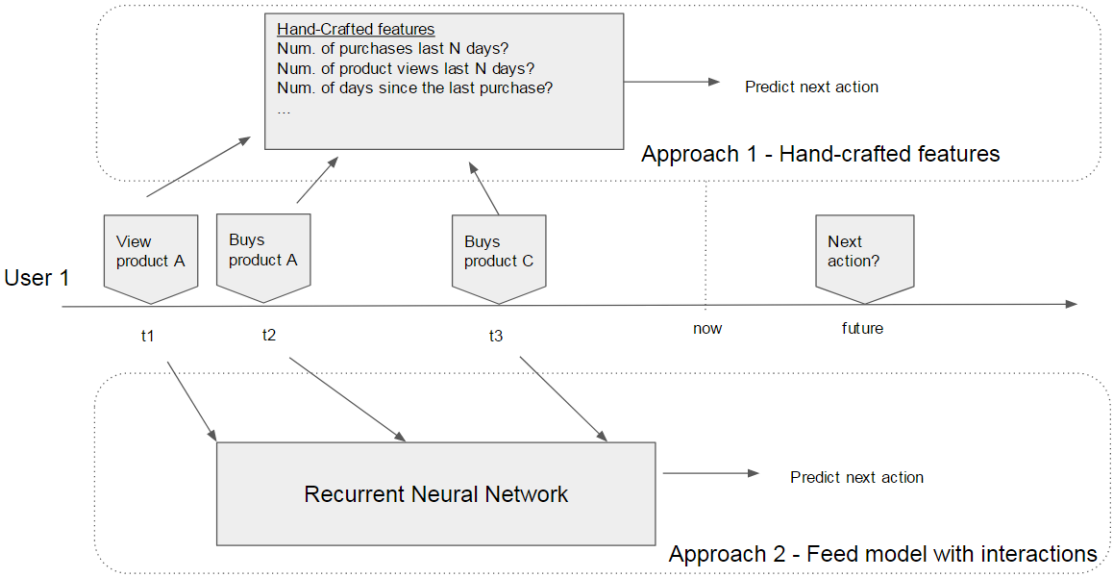

FIGURE 1.1: The method hand-crafted features vs RNN. Feature engineering creates features looking at the historical data and the resultant feature vector is fed to a ML method. With RNN the raw data is fed in the network, without the need of create hand-crafted features.

aspect of one-hot encoding vector is that they do not include information about each particular item or interaction, only the fact that the item or the interaction is different from others is encoded.

Therefore, it is necessary to find a good representation of the interactions to facilitate the learning. In NLP word2vec[12] has been applied successfully in many tasks, where words are mapped to distributed vector representations that capture syntactic and semantic relationships. These distributed vectors have much lower dimensionality than the huge one-hot encoding vectors needed to encode a whole vocabulary. Similarly, we can use this approach with items. The first benefit that we obtain is that we reduce the input dimensionality and consequently the number of parameters to learn. Second, capturing semantic relationships in the item embeddings can help to the RNN in the learning process. For example, if the items *iPhone 6* and *iPhone 7* are mapped to similar vectors, the RNN is able to capture the similarities when they are fed to the model. Two different customers that buy any of these two different items might be interested in similar items in the future. Feeding the network with any of them could produce similar predictions. On the contrary, with the one-hot encoding representation, the RNN only is able to capture that they are different items.

Another challenge of RNN is to find explanations in their predictions. The complexity of deep learning models makes hard to explain their predictions and they are often treated as black box models. However, in many domains being able to explain the predictions is an advantage. For example, in the medical context, explaining that the model predicts that the patient has a high probability of suffering a disease because he suffered some particular symptoms in the past, can provide useful information to the doctors. Recently, Attention mechanisms have been used in NLP tasks like machine translation and sentence summarization [3, 31, 10]. These attentional mechanisms focus on specific words in the sentences when making predictions for the next word, which helps to see which words are important when making the next prediction. Using attentional mechanisms with interaction data could provide which interactions

lead to the model to make a particular prediction. Moreover, in some cases, these methods can increase the prediction accuracy.

1.2 Contributions

The main contribution of this work is the study of different techniques when using RNN to predict future customer behavior. More specifically, we focus on the next two aspects:

- We study how embeddings can be used to produce useful vector item representations that help to improve the predictions with RNN. We evaluate and analyze the vector representations of different alternatives to learn item embeddings.
- We study how attentional mechanisms can help to explain the predictions of RNN models. We analyze the performance of different attentional mechanism variants and provide examples where the predictions are explained by past interactions.

1.3 Thesis contents

The thesis is organized as follows: Chapter 2 provides an overview of the related research. Chapter 3 introduces the necessary background information for Recurrent Neural Networks. Chapter 4 details the problem definition, the research questions, and the different methods used in this thesis. Chapter 5 describes the used datasets and the setup of the experiments. Chapter 6 reports and analyzes the results of the different models. Finally, Chapter 7 concludes and propose some future directions.

Chapter 2

Related work

In this section, we provide the relevant literature to this thesis. We provide the related works using RNNs to model sequential data, the related works using item embeddings, and the related works using attention mechanisms.

2.1 Recurrent Neural Network to model sequential data

As mentioned earlier, RNNs have been highly used to model sequential data in NLP [22, 28, 39]. The common approach is to represent the sequential data as a sequence of words. In Language Modeling, these words are fed into the RNN one by one and at the output, we obtain the probabilities of the predicted next word.

Recently, Zalando [34] has applied RNN to model the sequence of interactions of the users in their webshop. In their work, they use the data from the different sessions of the user to predict the probability that the user will place an order within the next seven days. As inputs, they use a sequence of one-hot encoding vector which represents past actions as product-views, cart-additions, orders, etc. The RNN model is compared against a logistic regression model with intensive feature engineering efforts over multiple months and used in production systems, achieving a similar accuracy. Additionally, they also provide a way to visualize how the predicted probabilities change over the course of the consumer's history.

In [6] collaborative filtering is viewed as a sequence prediction problem and the authors use RNN in the context of movie recommendation. Differently to most of the collaborative filtering methods, in this work is considered the time dimension. In this case, they create a sequence of ratings of movies to predict which movies the user will watch in the future. They encode the sequence of movies that the user has rated in the past as a sequence of one-hot encoding vectors. In their experiments with the Movielens and Netflix data sets, their method outperforms standard nearest neighbors and matrix factorization, which are typical collaborative filtering methods that ignore the temporal aspect of the data.

In [4] the user sessions are modeled with RNN. They consider the first click when the user enters in a web-site as the initial input of the RNN. Then, each consecutive click of the user produces a recommendation that depends on all the previous clicks. In the input sequence, they use two different representations. In the first, each element in the sequence represents a one-hot encoding vector representing the actual event. In the second alternative, each element of the sequence represents all the events in the session so far. They use a weighted sum of the past events, where the events are discounted if they have occurred earlier.

2.2 Item embeddings

Word embeddings have been applied successfully in Natural Language Processing. Traditionally, NLP systems have represented words as one-hot encoding vectors or discrete atomic symbols. The problem with these vectors is that they do not provide useful information about the relationships between different words. The basic idea of word embeddings is that words can be mapped to vectors of real numbers, where semantically similar words are mapped to nearby points. In 2013 Word2vec [12] was created allowing to map words to vector representations using neural networks. This technique allowed to represent words with vector representations in NLP tasks such as machine translation.

As we have seen previously, is common to use one-hot encoding representations to represent items. However, using embeddings to represent items has been explored recently. In [5, 30] they propose a slight variation of word2vec to represent items with vector representations. These embeddings are learned from item sequences so that items that co-occur frequently in sequences are mapped close each other in the embedding space.

In [41] RNN are used to make recommendations using only the interactions of the user in the current browsing session. In this work, the authors use an embedding layer between the input click sequences and the RNN. This allows representing the one-hot encoded click events into a vector representation. Additionally, the authors explore some techniques which are relevant to our work. First, they use data augmentation for sequences when preprocessing the data. Given a sequence of length T $S = (s_1, s_2, ..., s_T)$ they create a new training sample for all the prefixes of the original sequence $(x_1 = (s_1), x_2 = (s_1, s_2), ..., x_T = (s_1, s_2, ..., s_T))$. Second, they create a novel model where the output is the item embedding prediction instead of the probabilities of different items. This allows to reduce the number of parameters of the model, as the embedding dimension is much lower than the number of items. Moreover, we avoid to perform the softmax operation at the end of the network, which is an expensive operation when the number of items is very large. This model is tuned by minimizing the cosine loss between the embedding of the true label and the predicted embedding. The authors report that this model makes predictions using only about 60% of the time used by the models that predict item probabilities, however, the prediction accuracy performs worse. They argue that the cause of this poor performance could be that the quality of the item embeddings is not good enough.

Similarly, in [9] an embedding layer is used between the input and the RNN. In this work, Electronic health records (EHR) are used to predict diagnosis and medication categories for a subsequent visit. The input sequence represents the different visits of the patient. Each visit is represented as a multi-hot vector which represents the different medical codes that were recorded in that visit. In their architecture, this multi-hot vector is mapped to a vector representation using an embedding matrix W_{emb}. Then, embeddings are fed to the RNN. They employ two approaches to learn the embedding matrix. First, they initialize W_{emb} randomly and they learn it while training the entire model. In this case, they use a non-linear transformation to map the multi-hot vector to the embedding. In the second approach, they initialize W_{emb} with a matrix generated by the Skip-gram algorithm [12] and then fine-tune this matrix while training the whole model. In this case, they use a linear transformation to map the multi-hot vector to the embedding in the final model. This last technique is reported to outperform learning W_{emb} directly from the data with random initialization.

2.3 Attention mechanism

As we mentioned earlier, explaining predictions is a desirable feature for predictive models. In [3] an attention mechanism was introduced for machine translation. When translating a sentence, the model is able to automatically search which parts of the source sentence are relevant to predict the next translated word. Briefly, the attention mechanism works by creating a context vector with a weighted average of the different hidden states of the RNN. They achieved a translation performance comparable to the existing state-of-the-art English-to-French translation, while providing a visualization of which words in the source sentence are used to predict each of the translated words.

Recently, the attention mechanisms have been an intense area of research and they have been used further in machine translation [31] and in other NLP task such as sentence summarization [10] or hashtag recommendation [24].

In [2] the attention mechanism is applied in the medical context, where EHR from visits is used to heart failure prediction. In their model, they map the EHR data from the visits to embeddings before the RNN layer, similarly to [9]. Then, they use an attention mechanism to decide the importance of every past visit before making the prediction. Differently to [3], they build the context vector using the embeddings instead of the hidden states of the RNN. As their input is are multivariate observations (several codes can be present in a visit), they use two attention mechanism, one focusing on the whole visit, and other focusing on particular variables of the visits. The resulting model allows to interpret which past visits and variables contribute to the prediction (for instance which past symptoms recorded in a particular visit lead to predict that the patient may suffer a heart failure in the future).

Chapter 3

Background

In this section, we provide the necessary background for Recurrent Neural Networks, embeddings and attentional mechanism.

3.1 Artificial Neural Networks

Artificial Neural Networks are machine learning models which are based on the biological brain. They consist of a collection of connected neurons which can carry an activation signal. When using complex architectures, Artificial Neural Networks are able to approximate highly complex, non-linear functions. For many years, the computational resources and the data were not sufficient to allow to train Artificial Neural Networks effectively. However, the computational power and the availability of data has been growing with the years and eventually complex architectures were successfully trained to outperform any other models in some tasks such as image recognition or speech recognition.

A general architecture of artificial neural networks is shown in figure 3.1. At the input of the network we have the different feature values of the input sample (for example, the pixel values of an image). The values are propagated forward through the hidden layers using the weighted connections until the output layer is reached. In a classification task, the output generally represents the probabilities of the sample to belong to each one of the different classes (for example, the probabilities that the input image is a dog, a cat, etc.). At every node, a non-linear function can be triggered. This architecture is also called Feed-forward Neural Network.

Although this architecture has been successfully applied in many tasks, it does not take into account the temporal aspect that characterizes sequential data. Each sample is assumed to be independent of previous data samples, therefore, the network is not able to keep a current state which stores information of historical data. In other words, they are not a natural fit when next data depend on previous data. Recurrent Neural Networks solve this problem by connecting the hidden layer with itself. They are able to maintain an internal state which allows them to exhibit dynamic temporal behavior and use the hidden state as an internal memory. In the next section, we explain in detail the Recurrent Neural Networks architectures.

3.2 Recurrent Neural Networks

Figure 3.2 shows the basic RNN model, where the rectangular box contains the nodes of the hidden layer. On the right side, we can see the unfolded version of the RNN, which can be seen as a deep feed-forward neural network with shared parameters between layers. U, W, and V are the weight matrices that are learned. The input at time step t is x_t, which is connected to the hidden state h_t of the network through

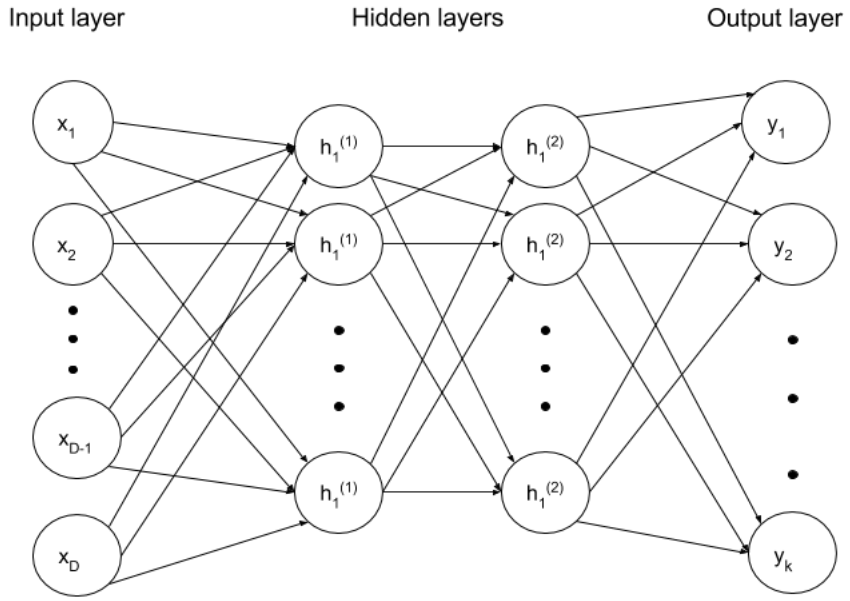

FIGURE 3.1: General architecture of feed-forward neural networks.

the weights U. The hidden state h_t is connected to itself via W and to the output y_t via U. The network can be described with the next two equations:

$$h_t = f(Ux_t + Wh_{t-1}) \tag{3.1}$$

$$y_t = g(Vh_t) \tag{3.2}$$

Where $x_t \in \mathbb{R}^D$, $h_t \in \mathbb{R}^H$ and $y_t \in \mathbb{R}^K$. The parameters learned are the matrices $U \in \mathbb{R}^{H \times D}$, $W \in \mathbb{R}^{H \times H}$, and $V \in \mathbb{R}^{K \times H}$. f is usually a non-linear and differentiable function applied to the hidden nodes, such as $tanh$, and g is a function which may depend on the task, for example for a classification task with only one valid class the softmax function is normally used.

Given an input sequence $x = (x_1, x_2, ..., x_{T-1}, x_T)$ of length T, we can feed the network with each element x_t one by one. The function of the network is to store all relevant information through the different time steps in order to capture temporal patterns in sequences. The RNN maps the information contained in the sequence until time step t into a latent space, which is represented by the hidden state vector h_t. Therefore, RNNs can solve the problem of modeling the temporal aspect of the data. In the next section, we discuss how we can train Recurrent Neural Networks.

3.3 Training RNN

Similarly to Feedforward Neural Networks, RNN can be trained with an adaptation of the backpropagation algorithm and gradient descent, where the weights are updated in the opposite direction of the gradient to minimize the loss function.

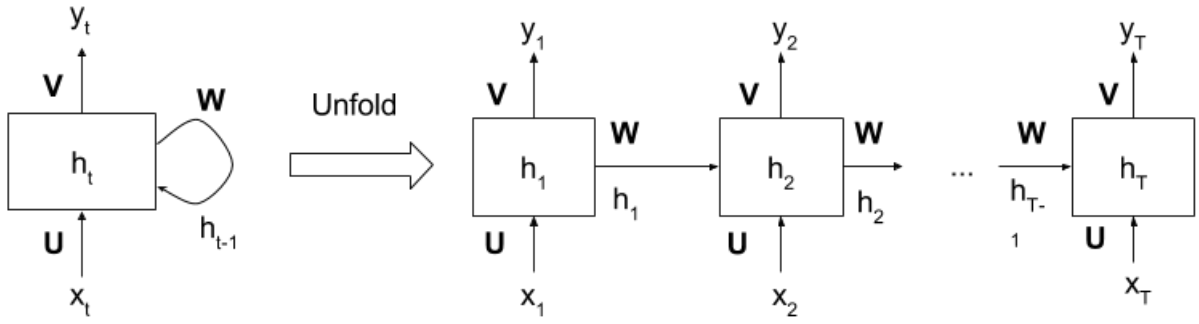

FIGURE 3.2: General RNN model. Left: folded version of the RNN. Right: unfolded version of the RNN.

3.3.1 Backpropagation Through the Time algorithm and problems

As shown in figure 3.1, we can represent an RNN as a Feedforward Neural Network where we have a layer with shared weights for every time step. Therefore, we can train RNN with the backpropagation algorithm used to train Feedforward neural network, but with the important difference that we need to sum up the gradients of the weights for every time step. However, as explained in [35], when the sequences are long the vanishing and exploding gradient problems can appear. The derivatives of the tanh or other sigmoid functions used in neural networks are very close to 0 at both ends of the function. This can create a problem since when several layers have gradients close to 0 the multiple matrix multiplications make the gradient values to shrink very fast. After a few time steps, the gradients can vanish completely, leading to no learning. Similarly, we find that if the gradients have large values, the matrix multiplications with many time steps can cause the exploding gradient problem, which leads to a situation where the values of the network parameters are unstable.

Usually, the exploding gradient problem can be solved by clipping the gradient at a threshold. However, the vanishing problem is more complex to solve. To approach this problem, gated RNN units were proposed.

3.3.2 Gated RNN Units

To face the gradient vanishing problem, the long short-term memory (LSTM) block was proposed [19]. The LSTM is composed by a memory cell (a vector) and three gate units. The gates are vectors where sigmoid functions are applied, making their values to be between 0 and 1. These vectors are then multiplied by another vector, so the gates decide how much of the other vector we obtain. Conceptually, each gate has a different function. The input gate decides which information is relevant in the current input to update the hidden state. The forget gate defines which information of the previous hidden state is no longer needed. Finally, the output gate controls which information of the new computed hidden state goes to the output vector of the network. Thanks to the gate mechanism, the LSTM block helps to combat the vanishing and exploding gradient problems and produce better results than the standard RNN.

This LSTM block is not the unique model using gates. In [16] an LSTM variant was introduced using peephole connections, so the previous internal cell state of the LSTM is connected to the gates. Gated Recurrent Units (GRU) [11] are a simplified

version of LSTM. They combine the forget and input gates into a single update gate. The model is simpler than the standard LSTM and has fewer parameters, which has lead in some cases to a better performance. There are other alternatives, but most of them contain only small variances.

Here we expose the details of the model used in this work, which is based on [19]. We denote C_t the cell state at time t, which is internal to the LSTM block, and h_t the final hidden state at time t at the output of the block. We consider x_t the vector of the input sequence at time step t. We start computing the forget and input gate vectors as follows:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.3}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{3.4}$$

where $[\cdot, \cdot]$ represents the concatenation between two vectors. Then, we compute a vector of new candidate values that can be added to the cell state:

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{3.5}$$

The next step is to update the new cell state C_t with the new information and the computed vectors at the gates:

$$C_t = f_t \cdot C_{t-1} + i_t * \tilde{C}_t \tag{3.6}$$

Then, the output gate decides what parts of the cell state will be at the output.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{3.7}$$

Finally, we compute the hidden state at the output using the output gate vector and the cell state:

$$h_t = o_t \cdot tanh(C_t) \tag{3.8}$$

As explained, Recurrent Neural Networks can be trained using the Backpropagation Through the Time algorithm. Using the LSTM blocks alleviates the vanishing and exploding gradient problems explained in the previous section and produces better results than standard RNNs [19, 18, 42]. In the next section, we will expose some techniques which help in the gradient descent optimization.

3.3.3 Gradient Descent Optimization methods

Recurrent neural networks can be trained with Backpropagation Through the Time algorithm and Gradient descent. The basic idea is that an objective or loss function is minimized by updating the parameters in the opposite direction of the gradient with respect the parameters. Therefore, in every iteration of the gradient descent, we take a step to a minimum of the objective function. The size of the step is determined by the learning rate η.

When we make the update using the samples one by one, the method is called Stochastic Gradient Descent, which can be summarized with the next equation:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta, x^{(i)}, y^{(i)}) \tag{3.9}$$

where θ are the parameters of the model, J is the objective or loss function to minimize, $x^{(i)}$ is the input vector of the training sample i and $y^{(i)}$ is its label. Once

we make the update for all the samples in the data set one by one, we have completed one epoch and we can continue the update process if the model has not converged. Updating the parameters using only one sample at the time has some problems. Using a small learning rate the model usually converges to a local minimum, but the training procedure can be very slow. Using a bigger learning rate can make the loss function to fluctuate and never converge. As the model is trained using the samples individually, we can update the parameters using noisy samples that the data set may contain, leading to a high variance.

To solve some of these problems, the mini-batch gradient descent makes every update by averaging the gradients of a group of samples:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta, x^{(i:i+n)}, y^{(i:i+n)}) \tag{3.10}$$

Here n denotes the batch size. Updating the parameters by taking several samples at the same time has some advantages. First, the model is more robust to noisy samples and there is less variance in the parameter updates. This leads to more stable convergence. Second, when making matrix operations with several samples we can make use of optimized libraries. Nonetheless, the method still requires to choose a learning rate that may be not easy to select. Moreover, the same learning rate could be not optimal in all the phases of the training and for all the different parameters.

Adagrad [13], Adadelta [43] and Adam [23] are variants of the gradient descent which try to solve these problems. Although we do not expose the details of these methods, the basic idea is to adapt the learning rate to the parameters during the different phases of the learning process. In most of the problems, we can obtain faster convergence by using these methods instead of the standard gradient descent.

3.3.4 Regularization

Due to their capacity to model complex non-linear functions and complicated relationships overfitting can be a serious problem in neural networks. In this section, we briefly review some of the techniques used to prevent overfitting in neural networks.

Weight decay with L2 regularization adds the L2 norm of the weights to the loss function to minimize. That encourages to the model to learn small weights, which usually implies simpler models that generalize better. L2 regularization is widely used in many machine learning models.

The early-stopping technique combats overfitting by interrupting the training before the model overfits. The basic idea is to create a validation set separated from the training set and test set. This validation set is evaluated every N steps of gradient descent or every epoch. When the performance on the validation set stops improving, the training procedure is stopped, as it might be a signal that the model is overfitting the training data. This technique avoids the need to tune the hyper-parameter number of epochs or steps.

Dropout was introduced in [37] and has been widely used to train deep neural networks recently. During training time some units in the network are removed (along with their connections) from the neural network. By doing this we avoid that the neurons learn complicated relationships which become noise when the dataset is not big enough, therefore we obtain a more robust and generalizable model.

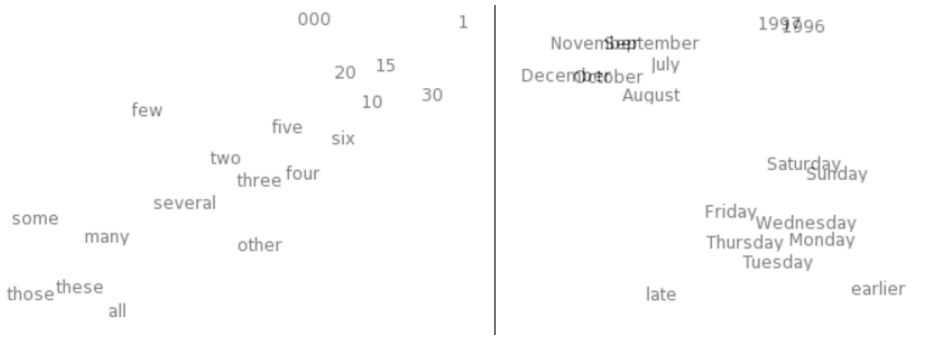

FIGURE 3.3: t-SNE of word representations from [32]. Left: word representations of the number region. Right: word representation of the job region.

3.4 Embeddings

In natural language processing is common to represent words by word embeddings. These embeddings are created by mapping other representation of words like ids or one-hot encoding vectors to a continuous vector space with much lower dimensionality. This has some advantages. First, in natural language processing we find huge vocabularies. By using one-hot representations with one dimension per word leads to vectors of huge dimensions. Therefore, by mapping the word to an embedding, we reduce the dimensionality of the vector considerably. Second, effective embedding representations of words can map words semantically similar close to each other in the embedding space. We can see this fact in figure 3.3 from [32], where embeddings of 50 dimensions are represented by reducing them to 2 dimensions with t-SNE [38]. On the left, we see word representations in the number context. We can see different groups like digit numbers or cardinal numbers. On the right side, we see word representations of years, months, and days of the week, which are grouped together in the embedding space. Therefore, we not only reduce the dimension, but we also include some information in these vector representations.

These word representations are very powerful in many models. Intuitively, we can see this with an example of sentiment analysis. Given the sentences "The movie is horrible", "The movie is awful" and "The movie is amazing" we want to create a model able to classify good reviews and bad reviews. Using one-hot representations the model does not have a clue about the difference between the sentences. The difference between "horrible" and "awful" is the same than "horrible" and "amazing", which complicates to the model to learn about the semantic meaning of the sentence. However, by mapping the words to word embeddings, the words "horrible" and "awful" are mapped close each other in the embedding space and distant to the word "amazing". That facilitates to the model to infer that the first two sentences have similar semantic meaning and different from the third sentence. Therefore representing the data correctly help the training of machine learning algorithms.

Word2vec [12] is a very effective method which uses a neural network to learn word embeddings. Word2vec contains several variations (CBOW/skip-gram, with/without negative sampling, etc.) but the basic concept is to capture the semantic meaning of the words by looking which words are likely to co-occur in a dynamic windows size. In other words, it tries to learn the semantic meaning by looking at the context which the words appear.

Although embeddings have been mainly applied to words, they are not the exclusive application. By treating a sequence of any type of items as a sentence and applying word2vec, these items can be transformed to embeddings. This allows to represent items by meaningful vector representations. For example, if applied to books, vector representations of mystery books can be represented together in the embedding space and distant from art books.

3.5 Attention mechanism

Recently, attentional mechanism has been a very powerful technique to explain predictions and improve accuracy. Given an input sequence $x = (x_1, x_2, ..., x_T)$ an attentional neural network model decides which elements of the sequence are important to make the next prediction.

One of the firsts successful applications of attention mechanisms is machine translation [3, 31]. When we translate a sentence, we pay special attention to the word that we are translating and other words that might influence the translation of this word. Intuitively, the attention mechanism in machine translation tries to imitate this behavior.

More formally, the attention mechanisms in machine learning translation works as follows: given a source sentence $x = (x_1, x_2, ..., x_T)$ each element is fed into a RNN to produce the hidden states $h = (h_1, h_2, ..., h_T)$. Each hidden state h_t contains information about the whole input sentence until t with a strong focus on the parts surrounding the i-th word of the sentence. Then, the attention weights α are computed:

$$e_i = f(h, h_i) \tag{3.11}$$

$$\alpha_i = softmax(e_1, ..., e_i) \tag{3.12}$$

for $i = 1, ..., T$

where f can be a linear or non-linear function. The set of weights $\alpha_1, \alpha_2, ..., \alpha_T$ indicates how much attention to focus on each one of the words of the input sentence. Then, a context vector c is created by using these weights and the hidden state of the RNN of the respective input:

$$c = \sum_{i}^{T} \alpha_i h_i \tag{3.13}$$

The translation of the next word can be predicted by using this context vector. This allows to the model to encode the needed information using the different hidden states found in the sequence instead of carrying all the relevant information of the sequence into the final hidden state, which may be difficult for long sequences. Additionally, the weights show the importance of every element in the sequence in the current prediction, therefore it makes the model more interpretable, a desirable feature which is hardly found in deep learning models.

The equations shown here are generic and there are variations, as using linear or non-linear functions to create the weights, or create the context vector by attending to embeddings instead of hidden states.

Chapter 4

Problem Definition and Methodology

In this section we show the problem definition, the proposed research questions and the methods used to answer the research questions.

4.1 Problem definition

Given a user u_n we denote as $x^{(n)} = (x_1, x_2, ..., x_T)$ the set of past interactions of the user. Each element of the sequence $x_i \in \mathbb{R}^D$ represents an interaction or set of interactions. In this work, we limit the interactions to purchases and elimination of items in the portfolio, or item ratings, but we could represent other interactions such as product views. Given the input sequence $x^{(n)}$ we are interested in predicting the next action $y^{(n)}_{T+1}$, which can be the product purchased in the next time step. The predicted output vector $\hat{y}^{(n)}_{t+1} \in \mathbb{K}$ represents the predicted probabilities for each possible action that the user u_n will perform in the next time step. In this case, we limit the outputs to the purchases of the different items. Given the set of predicted probabilities $\hat{y}^{(n)}_{t+1}$ we can sort them in order to create a set of recommendations for the user. We will consider multi-class classification where we classify each instance into only one label, and multi-label classification, where multiple target labels can be assigned to each instance. In the next sections, we will omit the index (n) indicating the user unless it is necessary. In section 5 we explain in detail the meaning of the input vectors x_t and y_t for each one of the data sets used in this thesis.

4.2 Research questions

Based on the exposed problem definition in this section, we explore the potential of RNN modeling sequential data. More concretely, we study the use of embeddings and attentional mechanism in this context, proposing the next research questions:

- RQ1: How can the embeddings be used to improve the performance when using RNN to model sequences? Which methods to create item embeddings lead to better prediction accuracy in our task when dealing with data with high dimensionality? What are the features that the different item embedding methods are capturing? We propose the next methods to create the embeddings: Learn embeddings separately with a specific method to create embeddings, learn embeddings jointly with the classification model, learn the embeddings separately (pre-train) and fine-tune while learning the classification model, and learn embeddings separately and predict embeddings at the output of the classification model. We detail these methods in the next section.

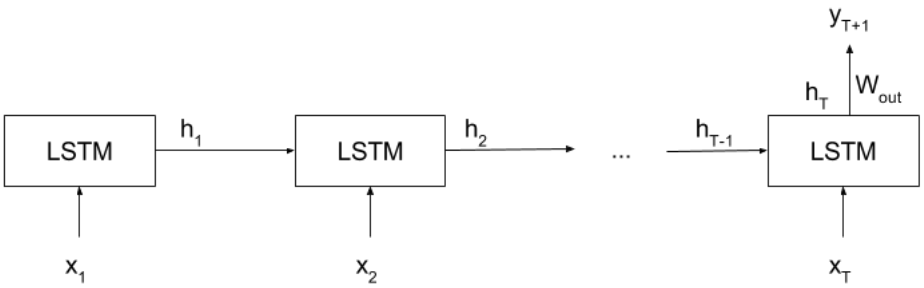

FIGURE 4.1: `RNN-Baseline` model.

- RQ2: Can attentional mechanism be effectively used to explain which interactions are important when making the predictions? Can they improve the prediction accuracy? Our goal is to study the performance of different attention mechanism methods and explore their utility to explain the predictions of the models.

In the next section we expose with detail the methods used in this work.

4.3 Methods

4.3.1 Baseline RNN

We start with an RNN model without embeddings and attentional mechanism, which we call `RNN-Baseline`. The model is depicted in figure 4.1

Each element x_t of the input sequence $x = (x_1, x_2, ..., x_T)$ is a one-hot or multi-hot encoding vector. Each element is fed to a LSTM block, creating a hidden state vector h_t:

$$h_t = LSTM(x_t, h_{t-1}) \tag{4.1}$$

The hidden state vector is obtained with the equations 3.3 to 3.8. After processing the whole sequence and obtaining the last final hidden state vector h_T, we obtain the predicted probabilities as follows:

$$\hat{y}_{T+1} = g(W_{out} h_T + b_{out}) \tag{4.2}$$

When dealing with a multi-class problem with one single valid class for each sample, g is the softmax function. Then, we optimize the weights of the model W_f, b_f, W_i, b_i, W_o, b_o in the LSTM block and W_{out} in the final layer by minimizing the cross-entropy of the correct item of the sample:

$$L = \frac{1}{N} \sum_{n=1}^{N} y_{T+1} \log(\hat{y}_{T+1}) \tag{4.3}$$

As we use the minibatch gradient descent described in equation 3.10, for each iteration of the Gradient descent N represents the batch size, which corresponds to the number of users in the minibatch.

Table 4.1: Different methods using embeddings.

Name	Description
RNN-Emb-Word2vec	Embeddings learned separately with Word2vec
RNN-Emb-Jointly-Lin	Embeddings learned jointly with the classification (linear)
RNN-Emb-Jointly-Nonlin	Embeddings learned jointly with the classification (non-linear)
RNN-Emb-Word2vec-Finetune	Embeddings learned separately and then fine-tuned jointly
RNN-Emb-Output	Predicting embeddings. Embeddings learned separately.

When dealing with a multi-label classification problem with multiple valid classes per sample, g is the sigmoid function and we optimize the weights by minimizing the cross-entropy, averaging the loss of all classes:

$$L = \frac{1}{N} \sum_{n=1}^{N} y_{T+1} \log(\hat{y}_{T+1}) + (1 - y_{T+1}) \log(1 - \hat{y}_{T+1}) \tag{4.4}$$

4.3.2 Embeddings methods

In this section, we describe the models used to answer the research question RQ1. Taking `RNN-Baseline` as a basis, we create some variations by adding embeddings to the model. Table 4.1 summarizes the different embedding methods that we explore and provides a brief description.

Embeddings learned separately with Word2vec

As we have seen word embeddings have been successfully applied in many NLP tasks. As in our work we treat with sequences of items, we can apply the same idea to create item embeddings. The motivation of using embeddings is two-fold.

First, if we have a large number of items the dimensionality of each item x_t when using one-hot encoding vectors can be huge. The input sequence is connected directly with the LSTM block. As a result, the number of weights in the gates W_i, W_f, and W_o grows with the number of items. Consequently, the model has many parameters to learn and requires more data to be successfully trained. Moreover, the huge matrix multiplications can make the training and predictions slow. Mapping the item to a real value vector of much more dimensionality instead of a huge one-hot encoding vector can help to solve this problem.

The second reason is that the one-hot encoding does not provide any semantic or similarity information about the input. By using effective methods we can represent item embeddings which contain semantic information about the item, which may help to increase the prediction performance as we are including additional information about the item.

An additional advantage of using embeddings is that if the number of items increases, the dimension of the embedding representation does not change.

Inspired by [5] we adapt word2vec to create item embeddings. To apply the method to items we replace the sequence of words (sentences) used in word2vec by sequences of items that a specific user has consumed. The rest of the method is applied as described in the original word2vec[12], concretely with the Skip-gram with Negative Sampling method.

In [5] the spatial information is removed by considering the window size as the same size of the whole user item sequence and giving the same weight to all the items.

Although we tried this approach, the method of considering a fixed context window size of minor size than the sequence led to better results in the first experiments. For this reason, we continued with a fixed context window size. This implies that we consider the proximity of co-occurrences of items to create the embeddings. Intuitively, it is reasonable to fix this window size in our data set. The reason is that in some cases it contains long sequences and the last items might be no related with the first ones.

Therefore, the item embeddings using the Skip-gram with the Negative Sampling method are created as follows:

Given a sequence of items $(i_i)_{i=0}^{T}$ that the user consumed, the objective is to maximize:

$$\frac{1}{K}\sum_{i=1}^{K}\sum_{-c\leq j\leq c, j\neq 0}\log p(i_{i+j}|i_i) \tag{4.5}$$

where c is the context windows size and $p(i_j|i_i)$ is defined as:

$$p(i_j|i_i) = \sigma(u_i^T v_j)\prod_{k=1}^{N}\sigma(-u_i^T v_k) \tag{4.6}$$

where $\sigma(x) = 1/1 + exp(-x)$ and N is the number of negative examples to draw per positive sample. A negative item i_i is sampled from the unigram distribution raised to the 3/4rd power. The vectors $u_i \in U(\subset \mathbb{R}^m)$ and $v_i \in V(\subset \mathbb{R}^m)$ correspond to the target and context representations for the item i_i, respectively, and m is the embedding size.

To combat the imbalance between rare and frequent items, each item i_i in the input sequence is discarded with probability by the formula:

$$p(discard|i_i) = 1 - \sqrt{\frac{t}{f(i_i)}} \tag{4.7}$$

where $f(i_i)$ is the frequency of the item i_i and t is a chosen threshold. Once optimized by maximizing 4.5, u_i is used as embedding of the item i_i.

Once we obtain the items embeddings, the model can be described as the `RNN-Baseline` model, with the difference that each element x_i of the input sequence $x = (x_1, x_2, ..., x_T)$ represents the corresponding item embedding u_i. We denote M the dimension of the embeddings and K the total number of items. As $M << K$, then the dimensionality of the weight matrices W_i, W_f and W_o is much lower than in the previous model. Therefore, we obtain a model with fewer parameters to learn. Moreover, we avoid to compute matrix multiplications of high dimensionality. We will refer to this model as `RNN-Emb-Word2vec`.

Embeddings learned jointly with the classification model

While learning embeddings with a separated method can produce high quality vector representations, they might be not optimized for the particular task to solve. For this reason, we propose to learn the embedding representations jointly with the classification model. This approach has been already attempted in other works [9, 2, 41]. Figure 4.2 shows the model.

As in the `RNN-Baseline` model each element x_t represents a one-hot or multi-hot encoding vector. In this case, the element is mapped to an embedding by the weight

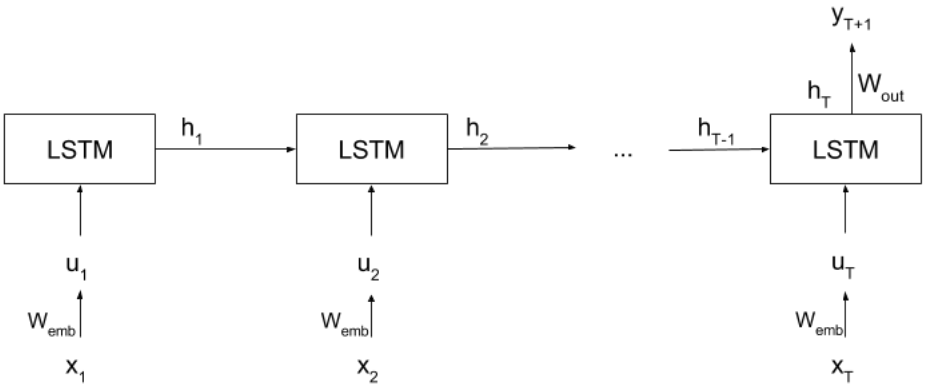

FIGURE 4.2: `RNN-Emb-Jointly-Lin` and `RNN-Emb-Jointly-Nonlin` models. The item embeddings are learn jointly with the classification model

embedding matrix W_{emb}. Here we consider two options. Learn embeddings by a linear function of the input:

$$u_t = W_{emb} x_t \tag{4.8}$$

And learn embeddings by a non-linear function of the input:

$$u_t = \tanh(W_{emb} x_t + b_{emb}) \tag{4.9}$$

The resulting embedding u_t is used as input of the LSTM. The rest of the model continues as `RNN-Baseline`. As `RNN-Emb-Word2vec` the weight matrices W_i, W_f and W_o have much lower dimensionality than `RNN-Baseline`, but there is a new weight matrix W_{emb} of dimensionality $M \times K$ to learn. We will refer to this model as `RNN-Emb-Jointly-Lin` when using linear embeddings and `RNN-Emb-Jointly-Nonlin` when using non-linear embeddings.

Embeddings learned separately and then fine-tuned jointly

Learning high quality embeddings jointly with the classification model from scratch can be a complex task. In this method, we try to use the best of the previous models: we pre-train high quality embedding representations with the method described in section 4.3.2 and then we fine-tune them for the task to solve. This technique has been successfully applied in [9] in the medical context.

The model can be represented as the `RNN-Emb-Jointly-Lin` with linear embeddings model shown in figure 4.2. The difference is that the embedding matrix W_{emb} is initialized with the pre-trained embeddings obtained with the Skip-gram method. We will refer to this model as `RNN-Emb-Word2vec-Finetune`.

Predicting embeddings. Embeddings are learned separately

In the previous models, we used the embeddings only before the LSTM block. In [41] a novel model was introduced where the final prediction of the model is an embedding. In the previous models, the weight matrix W_{out} is connected from the hidden state of the LSTM to the output of the network, where we have the probability of each one of the different items. If K is the number of items, and H is the number of nodes in

the hidden state, then the number of parameters of W_{out} is $K \times H$, which grows with the number of items.

As in [41] we propose a model to predict the embedding of the next consumed item, instead of predicting the probability of consuming each one of the items. If M is the dimensionality of the embeddings, this method reduces the number of parameters of W_{out} that reduces the number of parameters to $M \times H$, with $M << K$. Besides reducing the number of parameters, we avoid to compute a large multiplication matrix and the softmax operation, therefore the prediction time can be reduced as shown in [41].

The model is trained by minimizing the cosine loss between the embedding of the true output and the predicted embedding. One of the requirements to achieve a good accuracy is to use good quality embeddings. In [41] the model was reported to perform poorly compared with models without embedding prediction in terms of predictive accuracy. The authors believe that the performance could increase by using better quality embeddings.

The model can be represented as the `RNN-Baseline` in figure 4.1. In this case, each element x_t of the input sequence represents the embedding of the item, which is connected directly to the LSTM in the same way than the model `RNN-Baseline`. Each hidden state h_t is obtained via the LSTM equations. Finally, the predicted embedding is computed as follows:

$$\hat{y}_{T+1} = W_{out} h_T + b_{out} \tag{4.10}$$

The dimensionality of y_{T+1} is the embedding size M in this case.

The loss function minimized is the cosine loss between the predicted embedding $\hat{y}_{T+1}$ and the real embedding y_{T+1}:

$$L = \frac{1}{N} \sum_{n=1}^{N} \left(1 - \frac{\hat{y}_{T+1} \cdot y_{T+1}}{\|\hat{y}_{T+1}\|_2 \cdot \|y_{T+1}\|_2} \right) \tag{4.11}$$

We will refer to this model as `RNN-Emb-Output`.

4.3.3 Attention mechanism

As mentioned previously, attentional mechanisms can be used to interpret the model predictions, showing the importance of the different past elements of the input sequence when making each prediction. Moreover, in the previous models, we make the predictions using only the last hidden state, meaning that the model has to learn to pass all the necessary information through all the sequence to make the prediction, which can be difficult with long sequences. With attentional mechanism, we create a context vector by attending different elements of the input sequence, which avoids the necessity of encoding all the information in the last hidden state.

For the attention models, we take as a basis the base the model `RNN-Emb-Word2vec--Finetune`, which contains a layer of embeddings between the input and the LSTM. We will see in Chapter 6 that all the embeddings methods achieve a similar performance at the end of the training. However, `RNN-Emb-Word2vec-Finetune` converges faster. For this reason, we use this method as a basis for the attentional models. Table 4.2 summarizes the different attentional methods that we explore.

In the next sections we expose with detail each one of the methods.

TABLE 4.2: Different methods using attentional mechanisms.

Name	Description
RNN-Att-HS-Lin	Attention to the RNN hidden states with linear attention weights
RNN-Att-HS-Nonlin	Attention to the RNN hidden states with non-linear attention weights
RNN-Att-Emb-Lin	Attention to the embeddings with linear attention weights
RNN-Att-Emb-Nonlin	Attention to the embeddings with non-linear attention weights

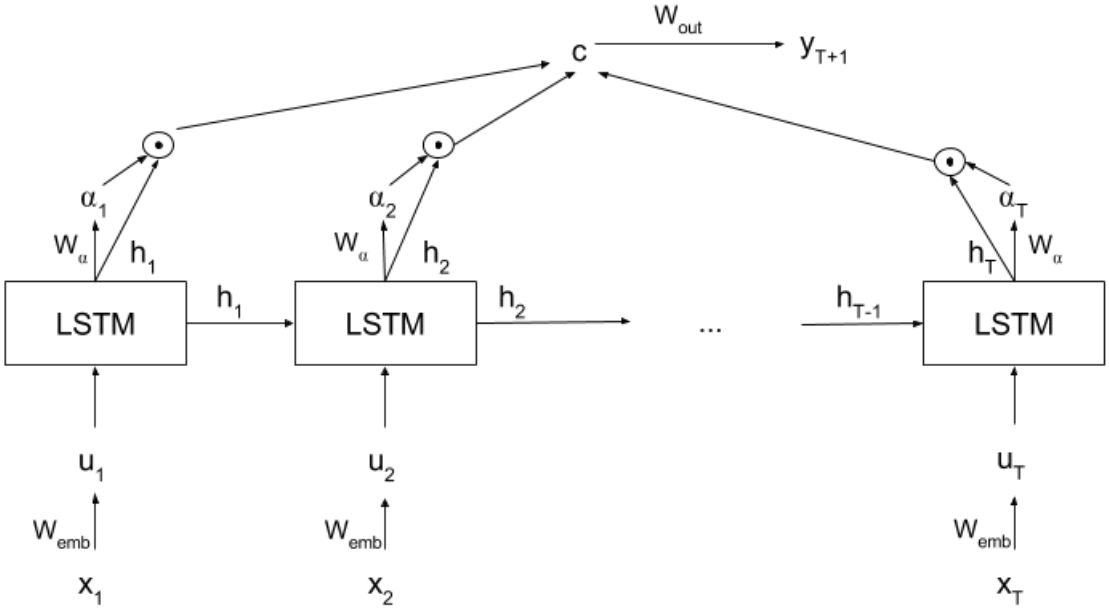

FIGURE 4.3: `RNN-Att-HS-Lin` and `RNN-Att-HS-Nonlin` model. LSTM with attentional mechanism to the hidden states

Attention to the RNN hidden states

Recently, attention to the hidden state has been successfully applied in some works [3, 31, 44]. The idea is to create a context vector by attending to the different hidden states created while processing the input sequence. The model is shown in figure 4.3. The difference with respect `RNN-Emb-Word2vec-Finetune` is that we create a weight α_t for every hidden state h_t. These weights represent how much the model focus in every hidden state to create the context vector c, which is used to predict which items will be consumed next.

Each weight α_t is created as follows. We first compute an energy e_t, where we consider a linear function of the hidden state:

$$e_t = W_\alpha h_t + b_\alpha \tag{4.12}$$

and non-linear function using tanh:

$$e_t = \tanh(W_\alpha h_t + b_\alpha) \tag{4.13}$$

Then, we proceed to compute the weights α_t by using the softmax function with all the elements e_t:

$$\alpha_t = \frac{\exp e_t}{\sum_i^T e_i} \tag{4.14}$$

To create the context vector c we use the weights α_t to compute a weighted average of the hidden states:

$$c = \sum_{t=1}^{T} \alpha_t h_t \tag{4.15}$$

Finally, the context vector is used to predict the label for the input sequence:

$$\hat{y}_{T+1} = g(W_{out} c + b_{out}) \tag{4.16}$$

As in the previous models, g is the sigmoid function for multi-label classification with multiple valid classes per sample and the softmax function for multi-class classification with only one valid class. We use the same loss function described in the `RNN-Baseline` model. We denote this model as `RNN-Att-HS-Lin` when using a linear function to compute e_t and `RNN-Att-HS-Nonlin` when using a non-linear function.

Attention to the embeddings

Attention to the hidden state is the most common form in the literature. Recently, [2] proposed an attentional method where the attention weights are used with the embeddings. Inspired by them, we propose the method shown in figure 4.4. The difference with the `RNN-Att-Emb-Lin` and `RNN-Att-Emb-Nonlin` models is that the context vector is created by focusing the attention to the embeddings instead of the hidden states. This model is simpler than the model used in [2], where two RNNs are trained to create 2 different sets of attention weights, one for the visit level, and another for the variable level.

The attention weights α_t are computed in the same way than the models `RNN-Att-HS-Lin` and `RNN-Att-HS-Nonlin`. In this case, the context vector c is created by computing the weighted average of the different embeddings in the input sequence:

$$c = \sum_{t=1}^{T} \alpha_t u_t \tag{4.17}$$

The rest of the model is the same than `RNN-Att-HS-Lin` and `RNN-Att-HS-Nonlin`. We will refer to the model as `RNN-Att-Emb-Lin` when using a linear function to compute e_t and `RNN-Att-Emb-Nonlin` when using a non-linear function.

Focusing on the embeddings allows to create the context vector by focusing exclusively on the different elements of the sequence, as each embedding only contains information about itself. On the contrary, the different hidden states may contain information from previous inputs. Hence the attention to the embeddings could be a better method to know how much the network takes into consideration each one of the previous inputs when making predictions.

4.4 Preprocessing sequential data

When dealing with sequential data, we can think different ways to create data samples.

Considering the sequence of 4 elements $s = (s_1, s_2, s_3, s_4)$ we consider the next three options:

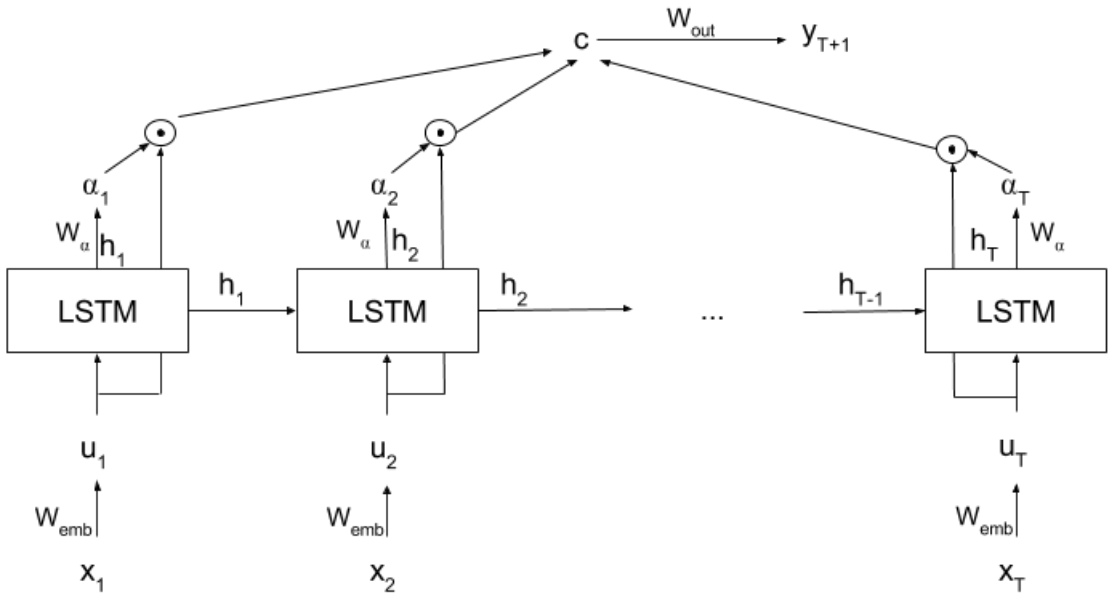

FIGURE 4.4: `RNN-Att-Emb-Lin` and `RNN-Att-Emb-Nonlin` models. LSTM with attentional mechanism to the embeddings

- Create one sample per complete sequence, considering the loss of the final element. It creates the sample $x = (s_1, s_2, s_3)$ with label $y = s_4$. In this case, the model can learn that after the sequence (s_1, s_2, s_3) comes the element s_4, but it does not learn about the intermediate elements.

- Create one sample per complete sequence, but considering the intermediate elements. It creates the sample $x = (s_1, s_2, s_3)$ with labels $y = (s_2, s_3, s_4)$. In this case, the model can learn with the intermediate elements by backpropagating the error at every time step.

- Create one sample for every prefix of the sequence. This method was used in [41] and was called data augmentation. It consists of creating a different sample for each of the prefixes of the sequence. In the example, we create the next samples:
  $x = (s_1), y = (s_2)$
  $x = (s_1, s_2), y = (s_3)$
  $x = (s_1, s_2, s_3), y = (s_4)$
  As the previous option, this allows to the model to learn with the intermediate elements.

We tried the three options in the initial experiments. As the third option produced the best results, we used this preprocessing method for the next experiments, included the ones reported in chapter 6

Chapter 5

Experiments Setup

In this section, we expose the datasets used in this work, the setup of the experiments performed, and the evaluation metrics to measure the performance of the models.

5.1 Data sets

5.1.1 Santander Product Recommendation

This dataset was used for the Kaggle competition *Santander Product Recommendation - Can you pair products with people?* [1] in 2016. The data includes purchases and elimination of financial products in the product portfolio of customers from January 2015 to May 2016. Examples of this products are Savings Account, Mortgage, or Funds. The number of different products for the dataset is 24. In the data set, the data is sampled regularly for each month, so for every month we have the products that the customer has in that moment. Although the dataset contains customer profile data such as age or country of residence, we do not make use of this data as we want to focus on the interactions of the user.

In table 5.1 we can see an example of the data for a concrete customer. When the data goes from 0 to 1 the customer has purchased a product. Similarly, when the product goes from 1 to 0, the customer has removed the product.

The data from table 5.1 can be seen as the next interaction history:

- February 2015: Customer purchases product 2
- ...
- May 2015: Customer purchases product 24 and removes product 2

[1] https://www.kaggle.com/c/santander-product-recommendation

Table 5.1: Example of data in the Santander dataset for a user.

Timestamp	**Product 1**	**Product 2**	**...**	**Product 23**	**Product 24**
2015-01-28	1	0	...	1	0
2015-02-28	0	1	...	1	0
2015-03-28	0	1	...	1	0
...	...	...	...	...	...
2016-04-28	1	1	...	1	0
2016-05-28	1	0	...	1	1

Note that we can have more than one interaction on the same month, and we can have months without any interaction. Given this interaction history, we can build an input sequence $x = (x_1, x_2, ..., x_T)$ where each x_t represents an interaction. Each x_t is then a multi-hot vector of dimension 48 which encodes the different interactions. The first 24 dimensions encode the purchases of products and the last 24 encode the elimination of products. We only add an element x_T when there is an interaction. If during the 17 months there is only 3 months with interactions, then we only have 3 elements in the sequence $x = (x_1, x_2, x_3)$. T denotes the number of elements in the input sequence.

We set the same goal as in the Kaggle competition, which consists in predicting the products that the customer will purchase in the last month, given the data from previous months. As the user can purchase more than one product in the same month, we treat this problem as a multi-label classification. While in the Kaggle competition the evaluation is done by predicting the products purchased in June 2016, which is not public data, we use the last month of the provided training data as a test set, May 2016. Therefore, our setup proceeds as follows:

- Training: we use the data from January 2015 to April 2016 to create the training samples. For each user u_n we create one data sample as mentioned in section 4.4, i.e. every month that the user purchased a product we add a data sample with the previous interactions of this user, where the labels are the items purchased by the user that month. The final training data contains 282680 data samples for a total of 23119 different users.

- Test: the goal in our test set is to predict the added products of the users in May 2016. Given a user u_n we create an input sequence x_n with the interactions of the user from January 2015 to April 2016. Then, given x_n we predict the purchased products in May 2016 for the user u_n. As the data set contains many users which do not purchase products in May 2016, we only add in the test set the users who added products on that month. The final test set contains 21732 different users.

As the number of different products is low, we do not experiment the embedding models exposed in section 4 for this data set. Therefore, we only experiment with the models `RNN-Baseline`, `RNN-Att-HS-Lin` and `RNN-Att-HS-Nonlin`. In this case, in the attentional models we do not use the embedding layer shown in figure 4.4. Additionally, we create two baselines:

- Frequency Baseline: The model always makes the predictions according to the item purchase frequency in the whole dataset, independently of the user.

- Logistic Regression: In order to compare with the RNN model, we create an input vector with the same information of the input sequence used in the RNN models. Therefore, we create a vector $T \times 48$, where $T = 16$ is the maximum number of interactions considered and covers most of the cases in the data set. This vector contains all the interactions of the user, although the model does not know the order when they occurred.

5.1.2 Movielens

The second dataset used is the MovieLens 20M Dataset. The data set consists of the history of ratings of movies for different users. The ratings contain a time stamp, so we know the order in which the user rates the movies. In the data set, the rating

scores are between 1 and 5. However, we only use the fact that the user rated a movie and not the score. An example of user interaction history can be described as follows:

- 2010-01-10: User u_n rated movie 5
- 2010-01-15: User u_n rated movie 7
- ...
- 2010-03-12: User u_n rated movie 22

In this case, the goal is to predict which movie the user will rate next given the past ratings of the user. For a user u_n, we create the input sequence $x_n = (x_1, x_2, ..., x_T)$ where each x_t represents a movie rating. Each x_t is a one-hot encoding vector which encodes the movie rated by the user at time t, or the embedding representing that movie when using embedding methods. Given the input sequence x_n we predict the label for $T + 1$, which represents the next movie to be rated. Therefore, in this case, we treat the problem as multi-class classification with only one valid class.

In our experiments, we use the data from January 2009 onwards. We filter out all the data belonging to movies with less than 20 ratings in this period. The way to create the train set and test set is as follows:

- Train set: We consider the data from January 2009 to March 2014. For each user u_n we create one data sample as mentioned in section 4.4. This means that for every movie rating, we create a data sample where the input sequence contains the previous ratings until that moment, and the true label is the one-hot encoding or the embedding of the movie rated in the actual moment. We only create samples when there are at least 5 movie ratings before the actual rating, and we limit the number of ratings per sample to 100. Therefore, if the user has rated more than 100 of ratings, we only consider the latest 100. The final training set contains 4053420 data samples for a total of 32901 different users.

- Test set: We focus on predicting the ratings of the users from April 2014 until April 2015. We will refer to this period as the test period. Although we optimize the model to learn which movie will be rated next (short-term prediction), we also measure which movies the user will rate eventually (long-term prediction). Therefore, for each user, we create the true labels $y_n = (y_{n1}, y_{n2}, ..., y_{nP})$ where y_{nt} is the t-th movie rated by the user u_n and P is the number of movies rated by the user in the test period. To predict the rated movies, we create the sequence $x_n = (x_{n1}, x_{n2}, x_{nT})$, which contains the movies watched by the user u_n before the test period. We limit the movies of the input sequence to the last 100 movies rated by the user. The final test data contains 3669 different users.

To create the embeddings with word2vec we use the same training set. For each user, we create a sequence of the movies that this user rated from January 2009 to March 2014. Then, these sequences are fed to the word2vec model to create the item embeddings.

After preprocessing, we obtain a total of 10057 different movies. In this case, we evaluate this data set with all the methods explained in section 4.

5.2 Evaluation

5.2.1 Santander Product Recommendation

As we have explained in the previous section, the goal is to predict which products the user will purchase in May 2016. Therefore, the model predicts a set of probabilities $P(i_1), P(i_2), ..., P(i_k)$, where each $P(i_j)$ represents the probability of purchasing the financial product i_j. Given this set of probabilities, we can sort each item by the probabilities of being purchased. Then, we measure the model using the Recall@k.

The Recall@k cuts-off the sorted items at k. Then, it looks how many of the total purchased items are in the top k predictions. It can be defined by the next formula:

$$\text{recall@k} = \frac{\#\text{ of true positives in the top k predictions}}{\#\text{ of true positives}} \tag{5.1}$$

For this dataset, we make the cut-off at 3 (Most users have no more than 3 purchased products on the last month). Then, we evaluate the model by computing the mean of the recall of all users in the test set.

5.2.2 Movielens

In this case, we evaluate the short-term predictions (the first movie to be rated in the test period), and long-term predictions (all the movies rated in the test period).

Given an input sequence of rated movies before the test period, we obtain the predicted probability for each movie to be rated next. Similarly to the Santander dataset, we can sort each movie by the predicted probabilities of being rated. Then, we can measure the performance of the model by the "sequence prediction success at k" ($sps@k$). This measure was introduced in [6] and measures how good the model is in short-term predictions. For a given user, the $sps@k$ is 1 if the first rated movie in the test set is in the top-k predicted movies, and 0 otherwise. Concretely, we use the $sps@10$ and the reported results in chapter 6 correspond to the average over all the users.

We also report a long-term prediction measure, but in this case, we do not cut-off the recall at a fixed position k. The reason is that the number of rated movies has a lot of variance between different users. Some users only rate a few movies during the test period, while others can rate hundreds of movies. If we make the cut-off at 10, for a user that has rated 100 of movies we can only obtain a recall of 0.1 as maximum. This makes this measure not ideal to know how good is the model. Therefore, we use the R-Precision as a measure. The R-Precision makes the cut-off at the number of movies that the user has rated. That means that the cut-off varies from user to user. If the user rated 5 movies in the test period, we look at the top 5 predictions, if the user rated 100 movies in the test period, we look at the top 100 predictions. Therefore, if R is the number of rated movies in the test period for a given user, then, the R-Precision is computed as follows:

$$\text{R-Precision} = \frac{\#\text{ positives top R}}{\text{R}} \tag{5.2}$$

That makes that the measure will be between 0 and 1 for every user. Then, we average the measure for all the users.

5.3 Implementation and hyperparameters

We implemented the models using the Deep Learning framework TensorFlow [1], which allows to create and train deep neural networks without the necessity of providing the expressions of the gradients, as they are computed automatically. For the models that the embeddings were computed with the Skip-gram method, we used the python package Gensim [33].

To find the correct hyperparameters to train the models, we tried a small set of values. However, due to the long time needed for each experiment with the Movielens dataset, we did not perform a complete search in the parameter space. Therefore, is likely that the used hyperparameters are not optimal for the problem.

To make a fair comparison between models, the number of hidden neurons in the LSTM block and embedding size (only applicable in the Movielens dataset) were set to the same values between the different models. The number of hidden neurons was set to 128 in the Santander dataset and 256 in the Movielens dataset. The embedding size to represent the different movies was set to 64.

For all the models, the network was optimized by the learning method Adam with a batch size of 128. The learning rate was set to 0.001 for both datasets. The rest of the options for the Adam method were set as default in the Tensorflow package. We tried dropout and L2 regularization, but it did not lead to better results.

To avoid the model to overfit the training data because of excessive training epochs, we used the Early stopping technique mentioned in 3.3.4, where we train the model until the evaluation on a separated validation set not used to train stops to improve.

We also tried other RNN blocks, such as GRU or different LSTM variants, but we did not obtain better accuracy. Finally, we tried to stack more than 1 layers of LSTM blocks, but we did not have success to improve the performance. However, for the Movielens dataset adding more layers increased the amount of time of the training, therefore we were not able to train the model long enough to confirm that no improvement is obtained by adding more layers. Moreover, we could need to adapt the other hyperparameters for this configuration.

Chapter 6

Results and Analysis

In this section, we show the results obtained with the different models and provide a qualitative analysis of the embeddings and attention mechanism. Table 6.1 shows a summary of all the methods that we compare with the link to the section where the models are detailed.

6.1 Overview

Table 6.2 shows the results for the Santander dataset. We observe that `RNN-Att-HS-Lin` obtained the best performance, with results very similar to the `RNN_Baseline`. The Logistic Regression baseline also performed similarly to these models. Finally, the `RNN-Att-HS-Nonlin` performed poorly compared with the others. A possible reason could be that we did not find the right hyperparameters for this model, as we did not perform an extensive hyperparameter tuning. Although we see a small difference between the RNN model and Logistic Regression, with the experiments on this dataset we confirm the effectiveness of RNN to model sequential customer interactions. We perform a more extensive analysis in the experiments with the Movielens dataset, as it provides a bigger challenge and allows to compare all the methods.

Table 6.3 shows the results for the Movielens dataset. The model `RNN-Att-HS-Nonlin` achieves the highest sps@10 and R-Precision. Contrary to the Santander dataset, the `RNN-Baseline` achieves the lowest sps@10 and R-Precision and its performance is worse than the Frequency Baseline. This shows the importance of the embeddings in the scenario with a high number of items. Another observation is that the Attention to the hidden state has higher sps@10 than the attention to embeddings, while the Non-linear attention methods have higher R-Precision than the linear attention weights. We observe that the results are much lower than in the Santander dataset, due that the task of predicting between 10000 of items is much harder.

Table 6.1: Summary of the different methods.

Name	**Description**
RNN-Baseline	RNN Baseline without embeddings and attention mechanism
RNN-Emb-Word2vec	Embeddings learned separately with Word2vec
RNN-Emb-Jointly-Lin	Embeddings learned jointly with the classification (linear)
RNN-Emb-Jointly-Nonlin	Embeddings learned jointly with the classification (non-linear)
RNN-Emb-Word2vec-Finetune	Embeddings learned separately and then fine-tuned jointly
RNN-Emb-Output	Predicting embeddings. Embeddings learned separately.
RNN-Att-HS-Lin	Attention to the hidden states with linear attention weights
RNN-Att-HS-Nonlin	Attention to the hidden states with non-linear attention weights
RNN-Att-Emb-Lin	Attention to the embeddings with linear attention weights
RNN-Att-Emb-Nonlin	Attention to the embeddings with non-linear attention weights

Table 6.2: Results Santander dataset.

Model	**Recall@3**
Frequency Baseline	0.5610
Logistic Regression Baseline	0.9162
RNN-Baseline	0.9263
RNN-Att-HS-Lin	0.9279
RNN-Att-HS-Nonlin	0.7999

Table 6.3: Results Movielens dataset.

Model	**sps@10**	**R-Precision**
Frequency Baseline	0.04234	0.05368
RNN-Baseline	0.03897	0.05094
RNN-Emb-Word2vec	0.07606	0.08947
RNN-Emb-Jointly-Lin	0.07681	0.08708
RNN-Emb-Jointly-Nonlin	0.07531	0.08686
RNN-Emb-Word2vec-Finetune	0.07718	0.08846
RNN-Emb-Output	0.06557	0.08928
RNN-Att-HS-Lin	0.08055	0.08903
RNN-Att-HS-Nonlin	0.08393	0.09511
RNN-Att-Emb-Lin	0.07718	0.08356
RNN-Att-Emb-Nonlin	0.07756	0.09305

We observe that there are not big differences between the different methods, so from the results shown in table 6.3 is hard to know if some methods perform better than others. For this reason, we execute a statistical significance test. Here, we evaluate a null hypothesis that the difference in mean performance between two methods is zero (two-sided case):

$$H_o : M_0 = M_1 \tag{6.1}$$

$$H_a : M_0 \neq M_1 \tag{6.2}$$

where M is some evaluation measure, in our case the sps@10 or the R-Precision. In general, the process consists in computing a p-value and reject the null hypothesis if it is sufficiently low (usually less than 0.05). Then, we can conclude that the methods perform differently.

It is important to notice that we compare many different methods, so we face the multiple comparisons problem. Intuitively, this problem arises because when we compare multiple systems against each other, the probability of incorrectly rejecting a null hypothesis increases. In other words, if we fix a rejection level α for each test, the number of wrong conclusions increases as the number of tests performed increases. In [8] the multiple comparisons problem is explained in detail and presents some ways to address it. The basic idea is to adjust the p-values to the number of different experiments performed.

In [36] the authors concluded that Student's, bootstrap, and randomization tests largely agree with each other when used to compare information retrieval systems. They recommend to use these tests instead of the Wilcoxon and sign tests which easily disagree with other methods.

In [8] is provided an algorithm for a randomization test where all-pairs of methods are compared (two-sided test) in MAP and the p-values are adjusted for multiple comparisons by Tukey's HSD. Because this solves our multiple comparisons problem and randomization tests have been effective for statistical hypothesis testing in information retrieval, we use this algorithm to evaluate if there is statistical significance in our methods. Instead of using the AP measure of different topics, we use the sps@10 and R-Precision of the different users.

Table 6.4 shows the corrected p-values for the sps@10 measure. In bold we see the p-values lower than 0.05, where we consider that the difference in performance in sps@10 between the two methods is statistically significant. As we see, all the methods outperform the Frequency Baseline, except the `RNN-Baseline` which does not contain embeddings. This model is also outperformed by all the models with embeddings. Then, we observe that only between the `RNN-emb-Output` and `RNN-Att-HS-Nonlin` the performance is statistically significant, as the former has one of the lowest sps@10 and the latter has the highest sps@10. Between any of the other methods, we cannot conclude that performs better than the others in terms of sps@10.

Table 6.5 shows the corrected p-values for the R-Precision measure. Again we show in bold the p-values lower than 0.05, which corresponds to the case when the difference in performance in R-Precision is statistically significant. As in the previous case, the Frequency Baseline is beaten for all the methods, except for the `RNN-Baseline`, which at the same time is beaten for all the embedding and attentional methods. We see again that we cannot conclude that there is statistical significance with any of the embedding methods performs differently. However, we observe some differences with the Attentional methods. Looking at the row and column of the `RNN-Att-HS-Nonlin` model, we see that it has statistical significance with all the other methods, except with

Table 6.4: Adjusted p-values for multiple comparisons by Tukey's with a randomization test. Results correspond to the SPS@10 measure.

	RNN-Baseline	RNN-Emb-Word2vec	RNN-Emb-Jointly-Lin	RNN-Emb-Jointly-Nonlin	RNN-Emb-Word2vec-Finetune	RNN-Emb-Output	RNN-Att-HS-Lin	RNN-Att-HS-Nonlin	RNN-Att-Emb-Lin	RNN-Att-Emb-Nonlin
Frequency Baseline	0.999	**0**	**0**	**0**	**0**	**0.001**	**0**	**0**	**0**	**0**
RNN-Baseline	-	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
RNN-Emb-Word2vec	-		1	1	1	0.678	0.999	0.932	1	1
RNN-Emb-Jointly-Lin	-	-	-	1	1	0.584	1	0.963	1	1
RNN-Emb-Jointly-Nonlin	-	-	-	-	1	0.768	0.997	0.886	1	1
RNN-Emb-Word2vec-Finetune	-	-	-	-	-	0.561	1	0.973	1	1
RNN-Emb-Output	-	-	-	-	-	-	0.179	**0.032**	0.561	0.488
RNN-Att-HS-Lin	-	-	-	-	-	-	-	0.999	0.999	0.999
RNN-Att-HS-Nonlin	-	-	-	-	-	-	-	-	0.973	0.982
RNN-Att-Emb-Lin	-	-	-	-	-	-	-	-	-	1

`RNN-Att-Emb-Nonlin`. As we saw in table 6.3 the `RNN-Att-HS-Nonlin` achieves the highest R-Precision, followed closely by `RNN-Att-HS-Nonlin`. This last method also outperforms the methods which learn the embeddings jointly from scratch. Therefore, we see that the non-linear attention seems to be particularly effective for long-term predictions. On the contrary, the `RNN-Att-Emb-Lin` has the lowest R-Precision of all the embedding and attentional methods and our test indicates that the model is beaten by most of the other models in terms of R-Precision.

In general, we observe that all the methods with embeddings and attention outperformed the Frequency Baseline and the `RNN-Baseline` in sps@10 (short-term predictions), but there is no statistical significance that some of these methods performed better than the others. The embedding and attentional models also outperformed the Frequency Baseline and the `RNN-Baseline` in terms of R-Precision (long-term predictions). For this measure, the attentional methods with non-linear weights seem to perform better, while the `RNN-Att-Emb-Lin` shows a lower performance. Here we want to remark that the methods with attention to the embeddings have a disadvantage which could make the comparison unfair. While all the other models make the final predictions using the hidden states (either the last hidden state or the context vector created with different hidden states), the attention to the embeddings methods use the embeddings. As we set the hidden state size 256 and the embedding size to 64 for all the methods, the attention to the embeddings methods have a last layer with less dimensionality before the output predictions.

Regarding the Research Question 1, we can confirm that using embeddings improves the accuracy in the scenario where there is a high number of different items. However, none of the embedding methods showed better performance with respect to the others in the Movielens dataset. Although all the embedding models have a similar performance at the end, we expect that the `RNN-Emb-Word2vec-Finetune` can achieve better results faster as a result of a good embedding initialization. This is

TABLE 6.5: Adjusted p-values for multiple comparisons by Tukey's with a randomization test. Results correspond to the R-Precision measure.

	RNN-Baseline	RNN-Emb-Word2vec	RNN-Emb-Jointly-Lin	RNN-Emb-Jointly-Nonlin	RNN-Emb-Word2vec-Finetune	RNN-Emb-Output	RNN-Att-HS-Lin	RNN-Att-HS-Nonlin	RNN-Att-Emb-Lin	RNN-Att-Emb-Nonlin
Frequency Baseline	0.867	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
RNN-Baseline	-	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
RNN-Emb-Word2vec	-	-	0.943	0.901	1	1	1	**0.031**	**0.018**	0.548
RNN-Emb-Jointly-Lin	-	-	-	1	1	0.966	0.986	**0**	0.579	**0.016**
RNN-Emb-Jointly-Nonlin	-	-	-	-	0.997	0.937	0.97	**0**	0.672	**0.01**
RNN-Emb-Word2vec-Finetune	-	-	-	-	-	0.999	1	**0.003**	0.116	0.181
RNN-Emb-Output	-	-	-	-	-	-	1	**0.022**	**0.027**	0.469
RNN-Att-HS-Lin	-	-	-	-	-	-	-	**0.013**	**0.004**	0.367
RNN-Att-HS-Nonlin	-	-	-	-	-	-	-	-	**0**	0.979
RNN-Att-Emb-Lin	-	-	-	-	-	-	-	-	-	**0**

confirmed in figure 6.1, where we see that the `RNN-Emb-Word2vec-Finetune` model shows a lower validation loss from the beginning, and the difference with the others shrinks over time. The model `RNN-Emb-Word2vec` also starts with a good performance compared with the models with random embedding initialization, but the fact that it cannot adapt these embeddings for the task makes the model harder to train and the error shows more variance.

Regarding the Research Question 2, we can confirm that some attention mechanism methods are able to improve the accuracy of the long-term predictions, but the improvement was not statistically significant for the short-term predictions.

In the next section, we will address the other parts of the research questions. We will look at which kind of information the different embedding methods are capturing (RQ1), and how the attention mechanisms can be used to infer which interactions are the most important for the model when making predictions (RQ2).

6.2 Qualitative Analysis

In this section, we look at the embeddings learned by the models and we show examples of predictions made by the attention mechanism.

6.2.1 Embedding Analysis

In the next section, we show analysis for the embeddings representations obtained with the different methods. The goal is to observe which features are represented in the item embeddings. Note that when refer `RNN-Emb-Word2vec` we use the embeddings obtained directly by the Skip-gram with negative sampling (word2vec), for

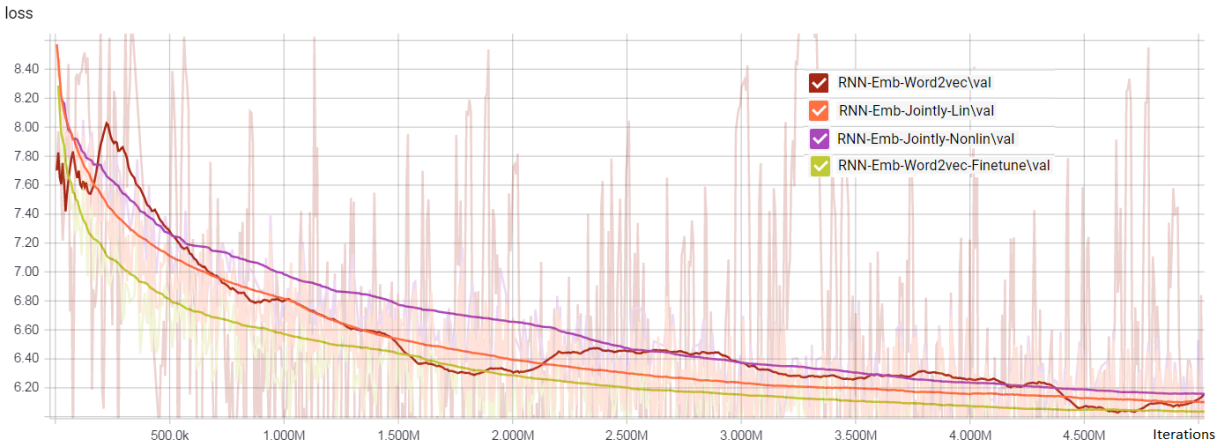

FIGURE 6.1: Loss of the validation set for the different embedding methods. The x-axis represents the number of iterations (updates)

TABLE 6.6: Top 5 most similar movies

	RNN-Emb-Word2vec	RNN-Emb-Jointly-Lin	RNN-Emb-Jointly-Nonlin	RNN-Emb-Word2vec-Finetune
Finding Nemo	Monsters, Inc.	Incredibles, The	Very Potter Musical	Incredibles, The
	Incredibles, The	Green Mile, The	Pentagon Wars	Monsters, Inc.
	Shrek	Monsters, Inc.	Neil Young	Ocean's Eleven
	Toy Story 2,	Ocean's Eleven	Devdas	Pirates of the Caribbean
	Pirates of the Caribbean	Pirates of the Caribbean	Find Me Guilty	Shrek
Star Wars IV	Star Wars: Episode V	Fugitive, The	Terminator 2	Star Wars: Episode VI
	Star Wars: Episode VI	Teenage Mutant Ninja Tur...	Wallace & Gromit	Silence of the Lambs
	Raiders of the Lost Ark	Richard Pryor: Live in...	Spirited Away	Star Wars: Episode I
	Matrix, The	Alone in the Wilderness	Ponterosa	Terminator, The
	Silence of the Lambs	Raiders of the Lost Ark	Star Wars VI	Shawshank Redemption
[REC]	Orphanage, The	Inside	Splinter	Orphanage, The
	Devil's Backbone	High Tension	Turtles Can Fly	Devil's Backbone
	Martyrs	Martyrs	Thesis	3 Extremes
	Eden Lake	Tale of Two Sisters, A	Dark Water	Battle Royale
	Descent, The	Frailty	Henry & June	Grudge 3, The
Shining, The	Clockwork Orange, A	Jaws	World According to Mons...	Stand by Me
	Full Metal Jacket	Clockwork Orange, A	The Devils	Psycho
	Taxi Driver	Trainspotting	Wal-Mart: The High Cost...	Clockwork Orange, A
	Trainspotting	Truman Show, The	Trainspotting	Trainspotting
	Psycho	World According to Mons...	Stand by Me	Repulsion

`RNN-Emb-Jointly-Lin` and `RNN-Emb-Word2vec-Finetune` we use the learned embedding matrix W_{emb}, and for `RNN-Emb-Jointly-Nonlin` we use the result of the operation $tanh(W_{emb}x_t + b_{emb})$.

Table 6.6 shows the top 5 most similar movies in the embedding space for some movies. In the first example, we see the top 5 most similar movies of *Finding Memo*. We see that all the methods except `RNN-Emb-Jointly-Nonlin` include other children movies in the top 5, like *Monsters, Inc* or *The Incredibles*. On the contrary, is difficult to find similarities to *Finding Memo* in the top 5 most similar movies for the `RNN-Emb-Jointly-Nonlin` method. The second example shows the top 5 most similar movies of *Star Wars: Episode IV*. The models `RNN-Emb-Word2vec` and `RNN-Emb-Word2vec-Finetune` seem to encode closely movies that are very related semantically, as they have more than one Star Wars movies in the top 5. Contrary to the previous case, we see relationships in some movies of the top 5 for the `RNN-Emb-Jointly-Nonlin`, like other Star Wars movie or other Sci-Fi movie *Terminator 2*. In the third example, the three models seem to position other horror/drama movies close to *[REC]*. Interestingly, in the last example the top 2 most similar movies to *The Shining* for the `RNN-Emb-Word2vec` are two movies from the same director.

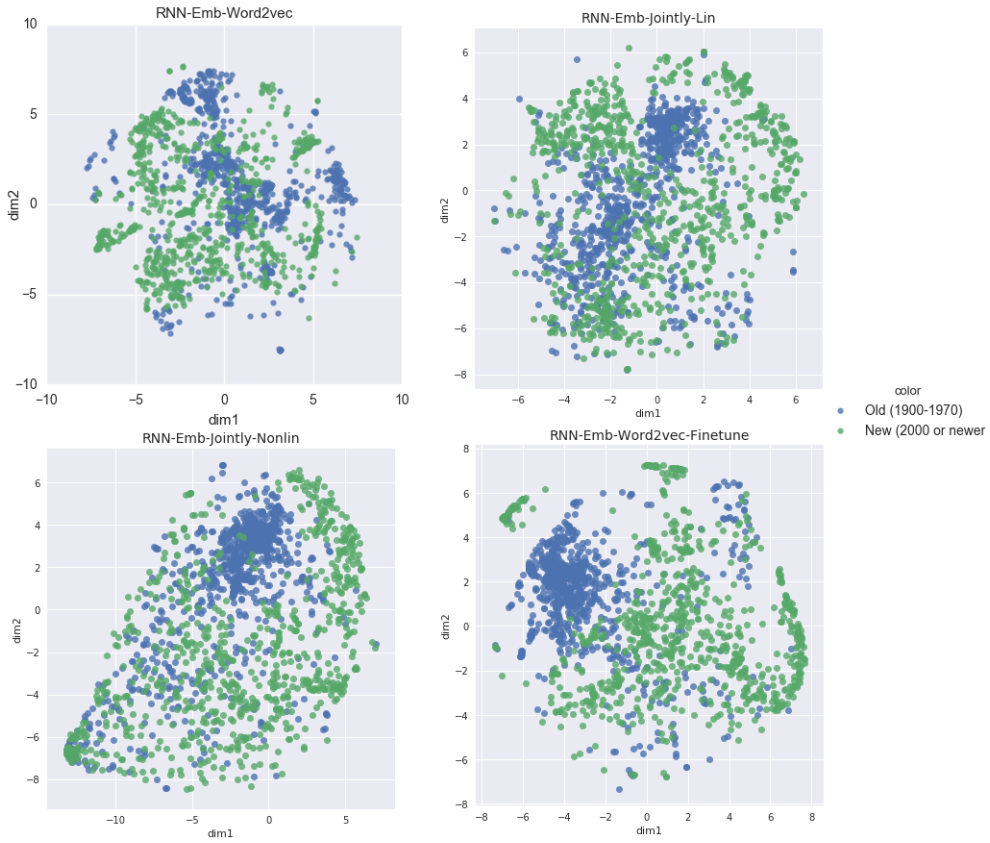

FIGURE 6.2: Embedding representation of old movies and new movies. Embeddings of dimension 64 were reduced to 2 dimensions with t-SNE. Top-Left: `RNN-Emb-Word2vec`. Top-Right: `RNN-Emb-Jointly-Lin`. Bottom-Left: `RNN-Emb-Jointly-Nonlin`. Bottom-right: `RNN-Emb-Word2vec-Finetune`

Figure 6.2 shows the embeddings of old movies (from 1900 to 1970) and new movies (from 2000 onwards) after reducing the dimensionality of the embeddings to 2 with t-SNE. For all the models we see some area where there is a high amount of old movies.

In figure 6.3 we look embedding representations of more concrete ranges of year of release. In all three methods, we see clusters by the year of release. Interestingly, in the models that learn the embeddings jointly from scratch we find that the groups of movies close in time (2010-2013, 2013-2014, 2014 or newer) are located close each other in distance, while for the other models these groups are more distant.

Figure 6.4 shows the embedding representations of movies of different genres. We see that all the methods are able to group horror movies. We see a large group of horror movies and other smaller groups, which could be specific subtypes of horror movies. The models also seem to group some children movies, specially `RNN-Emb-Word2vec` and `RNN-Emb-Word2vec-Finetune`. `RNN-Emb-Word2vec` has a clear separation between the big groups of children and horror movies, two genres quite different. On the contrary, the models which obtain the embeddings jointly have some areas where children and horror movies are mixed. There are many areas where we have romance movies, but it could be because it is a wide genre and related with many others.

Figure 6.5 shows the embeddings of some particular sagas of movies. As we see, `RNN-Emb-Word2vec` encodes closely all the different sagas, except for one of the Rocky

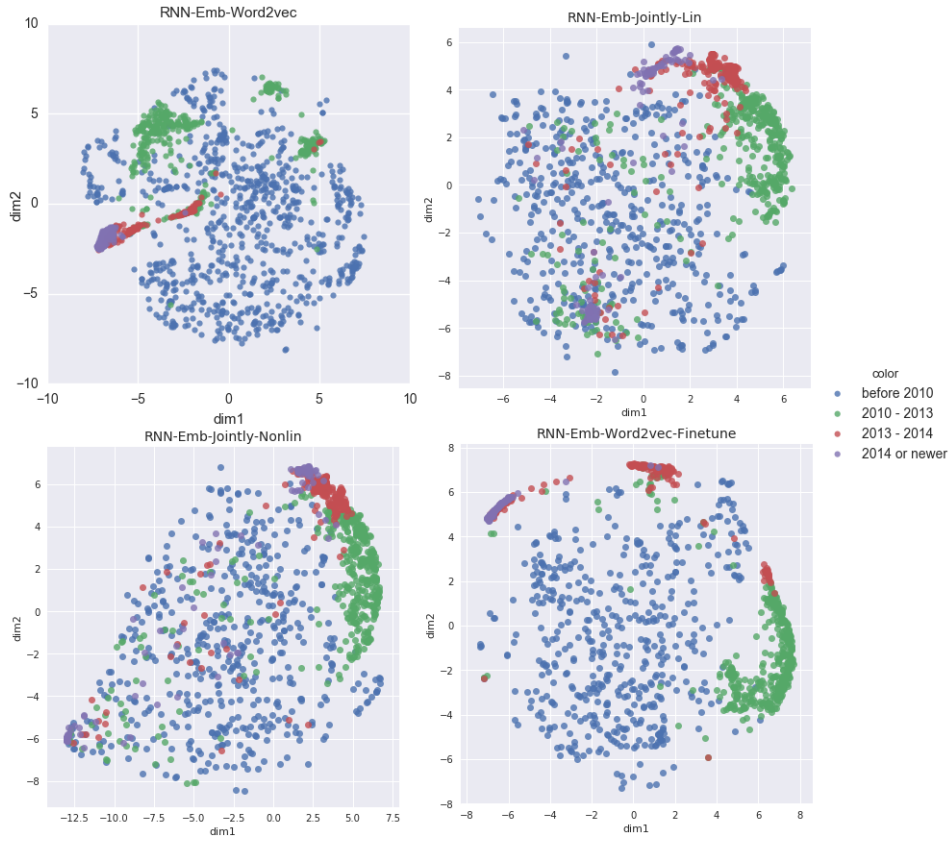

FIGURE 6.3: Embedding representation of movies released on different years. Embeddings of dimension 64 were reduced to 2 dimensions with t-SNE. Top-Left: `RNN-Emb-Word2vec`. Top-Right: `RNN-Emb-Jointly-Lin`. Bottom-Left: `RNN-Emb-Jointly-Nonlin`. Bottom-right: `RNN-Emb-Word2vec-Finetune`

FIGURE 6.4: Embedding representation of movies of different genres. Embeddings of dimension 64 were reduced to 2 dimensions with t-SNE. Top-Left: `RNN-Emb-Word2vec`. Top-Right: `RNN-Emb-Jointly-Lin`. Bottom-Left: `RNN-Emb-Jointly-Nonlin`. Bottom-right: `RNN-Emb-Word2vec-Finetune`

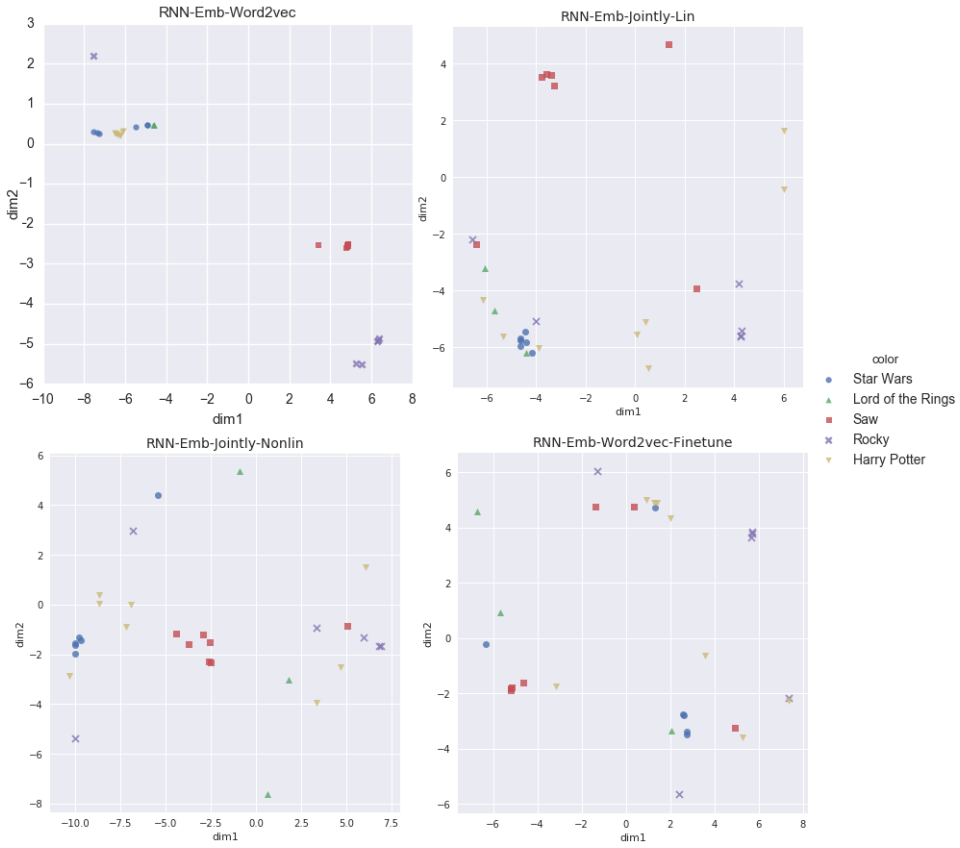

FIGURE 6.5: Embedding representation of movies of different sagas. Embeddings of dimension 64 were reduced to 2 dimensions with t-SNE. Top-Left: `RNN-Emb-Word2vec`. Top-Right: `RNN-Emb-Jointly-Lin`. Bottom-Left: `RNN-Emb-Jointly-Nonlin`. Bottom-right: `RNN-Emb-Word2vec-Finetune`

movies which is completed separated from the others. It seems that the other three method group some of the movies of each saga, but we also find that some movies of the sagas are in a different area.

In general, we observe that all the embedding representations contain some information about the items that are encoding. Word2vec learns the representations using the context of the items. As we have seen, word2vec embeddings seems to incorporate information about the period of release of the movie, the genre, or particular sagas. This means that probably there are some users which watch movies from the same period, the same genre or the same saga in a row. We have seen that the other methods are also able to incorporate this kind of information in the embeddings. These methods try to optimize the embedding representation for the task, where `RNN-Emb-Word2vec-Finetune` starts with the word2vec embedding representation. As the task consists in predicting the movies given the past movies, these embeddings representations are also learned taken into account the context. As we said, is likely that some users watch movies from the same period, genre or saga one after the other. As this is important for the prediction task, the models which learn the embeddings jointly are also able to incorporate some of this information in the embeddings. On the other hand, as the Word2vec is a method specific to learn embedding representations, it seems to represent more clearly this information, as we have seen with the sagas or the genres, although it does not seem to have an impact

on the performance.

Predicting embeddings

In this section, we analyze the predictions of the `RNN-Emb-Output` model. In figure 6.6 we show some examples of good predictions (left), and bad predictions (right). The blue dots are the embeddings of the last 10 movies rated by the user. The big red cross represents the predicted embedding. The green crosses are the embeddings of the first 5 movies that the user rates in the test period, where the big green cross is the first one. On the top-left, we see an example where many of the last rated movies by the user are located in the same area. As a consequence, the predicted embedding it is located in the same area. As the user continues watching movies whose embeddings are located around this space, some of the closest films are movies that the user rates in the test period. On the bottom-left, we see another good prediction. In this case, the predicted embedding is almost in the exact point of the embedding of the next movie, so in this case, the first movie is predicted correctly (sps@10 = 1). On the top-right, we see a typical example where the model does not achieve any correct prediction. We observe that the predicted embedding is located close to many of the embeddings of movies that the user recently watched. However, it seems that the user starts to watch different kinds of movies, or movies where the embeddings are in different areas. On the bottom-right, we see the another typical scenario where the model does not get any good prediction. In this case, the embeddings of the past rated movies are scattered around all the embedding space. Therefore, the model is not able to predict an embedding which is close to the past rated movies. Then, the user starts watching movies in some area which is not close to the prediction. This last scenario is the most challenging, as the previously rated movies are more variate. Intuitively, a good model would recommend a variety of different kind of movies in the top predictions, in order to be able to address the possible different kind of movies that the user will be interested. However, if we predict an embedding and we base our recommendations in the movies which are closest to this embedding we may lose the possibility to have variance in the predictions.

6.2.2 Attentional Analysis

As we mentioned, the attention weights can provide how much the model is focusing on each element of the sequence to make the predictions. Thus, they can be very useful to interpret predictions. In this section, we show some examples of predictions and attention weights for the different methods.

We show first the examples for the attention to the embeddings with linear weights (`RNN-Att-Emb-Lin`), as it is the method which provides higher interpretation to the model. Figure 6.7 shows the first example. On the top, we see the sequence of the last 28 movies that the user rated. The order goes from left to right, row by row, so the last row represents the latest movies. The attention weights are represented by color, where darker color indicates that the model focuses more attention. On the bottom, we see the top 10 predicted movies by the model. In this case, we see that the model has a strong attention to the last two Louis C.K. shows. Then, the movie with the highest probability to be rated is another Louis C.K. show with a high gap with the second highest probability. Moreover, we find two more Louis C.K. shows in the top 10 predicted probabilities, and the second predicted probability is *David Cross: Bigger & Blackerer* which is another comedy show. The second example is shown in figure 6.8. In this case, we see that the model is focusing mainly in children movies. Then,

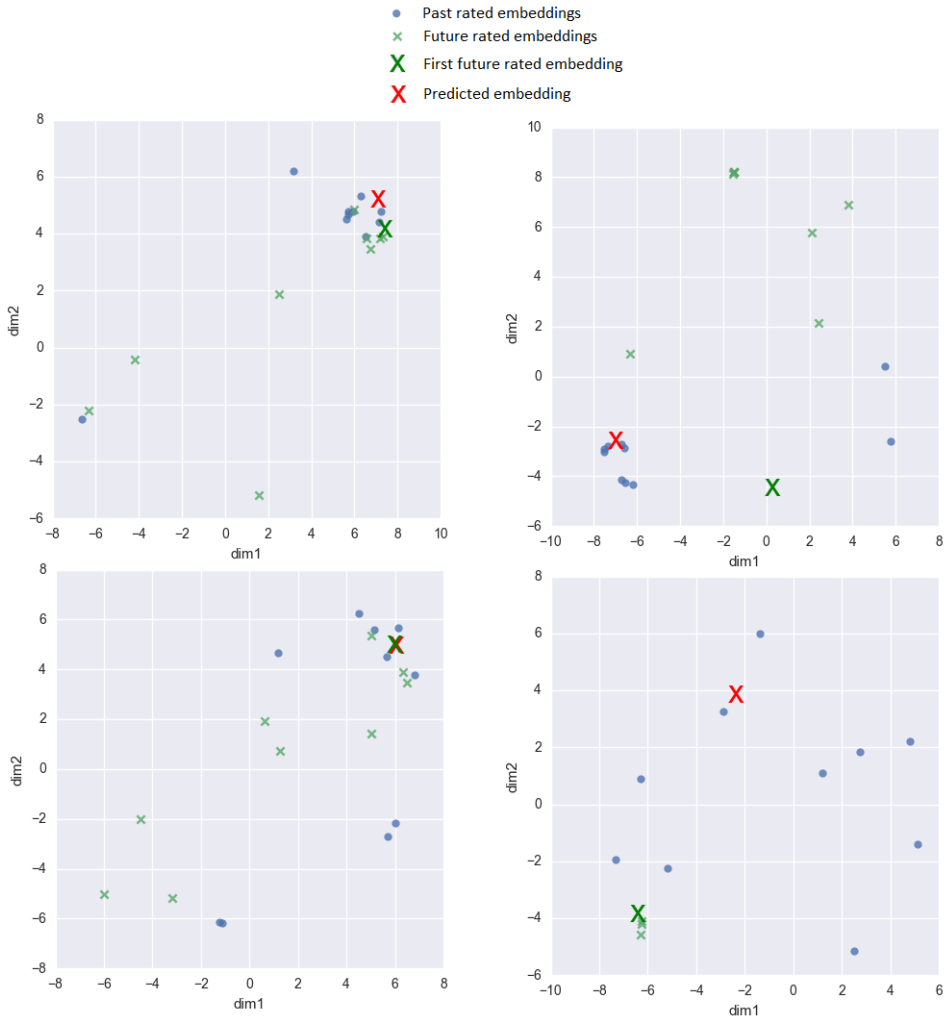

FIGURE 6.6: Examples of predicted embeddings reducing the dimension to 2 with t-SNE. The blue dots are the embeddings of the last 10 rated movies by the user. The big red cross represents the predicted embedding. The green crosses are the embeddings of the first 5 movies that the user rates in the test period, where the big green cross is the first one. Left: examples with at least one good prediction. Right: examples with incorrect predictions.

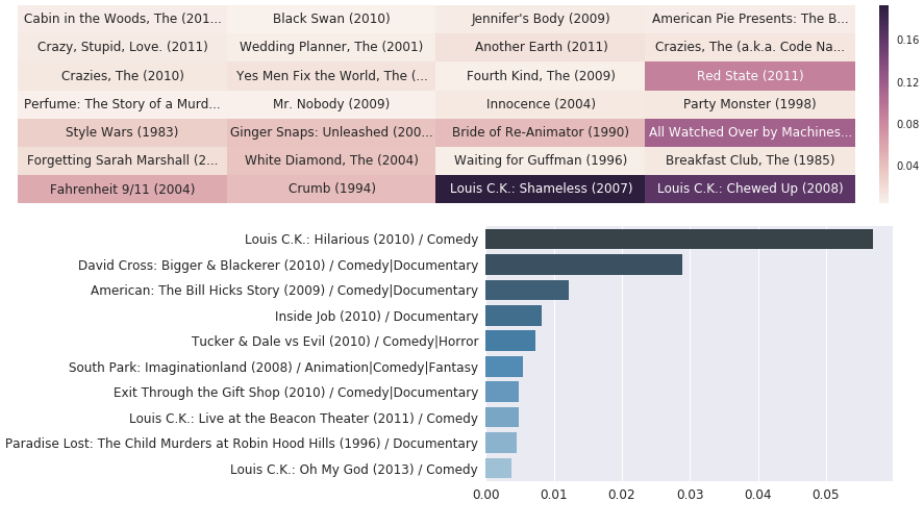

FIGURE 6.7: Attention mechanism to embeddings (linear). Top: History of rated movies by the user, the order goes from left to right, row by row. Color indicates the α weights. Darker color indicates more attention to the specific movie. Bottom: top 10 predicted movies.

we see that the top 10 is formed by children movies. As we see, all the films in the last 10 are children/fantasy movies except one, *Under the Skin*, which is considered as drama/horror movie. Interestingly, the model seems to take it less into consideration to the make the predictions. A final example is in figure 6.9, where differently to the previous examples the highest attention is not on the latest films. In this case, the movie with the highest attention is *The Expensables 3*. Then, in the top 10 predicted movies we see *The Expendables* as the movie with the highest probability, and *The Expendables 2* as the 7th movie with highest probability. In general, we observe that the attention mechanism to embeddings with linear attention seems to be effective to explain predictions, whether they are correct or not.

Figures 6.10, 6.11 and 6.12 show the same examples for the attention to the embeddings with non-linear weights (`RNN-Att-Emb-Nonlin`). We rapidly notice that the attention weights are more distributed between different movies. In the first example 6.10, the model predicts only one Louis C.K. show in the top 10 in the first example, contrary to the three predicted by the attention to the embeddings with linear weights, probably because now the model focuses on different movies. In the second example 6.11, the model focuses attention not only on children movies, so we see that the top 10 predicted movies are not all children movies now. Differently to the previous case, in most cases we do not have a particular movie or small set of movies that contributes to the predictions. Instead, we have big sets of movies with very small contributions. Consequently, in general we observe more variance in the predictions, probably because of the fact of dividing the attention in more movies. As the context vector is created using more movies, different kind of movies are taken into consideration for the final prediction.

Figures 6.13, 6.14 and 6.15 shows the examples for the attention mechanism to the hidden state with non-linear weights (`RNN-Att-HS-Nonlin`). Here we need to take into consideration that the hidden state encodes information from previous inputs. Therefore, the fact that the model is focusing on a particular hidden state cannot ensure that the model is focusing on a specific movie. However, if the input of a movie makes the network to have a strong attention to the correspondent hidden

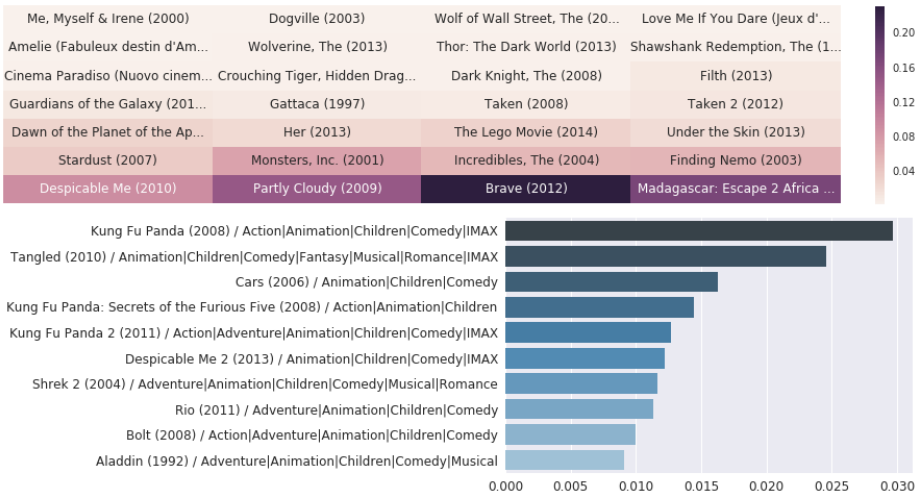

FIGURE 6.8: Attention mechanism to embeddings (linear). Top: History of rated movies by the user, the order goes from left to right, row by row. Color indicates the α weights. Darker color indicates more attention to the specific movie. Bottom: top 10 predicted movies.

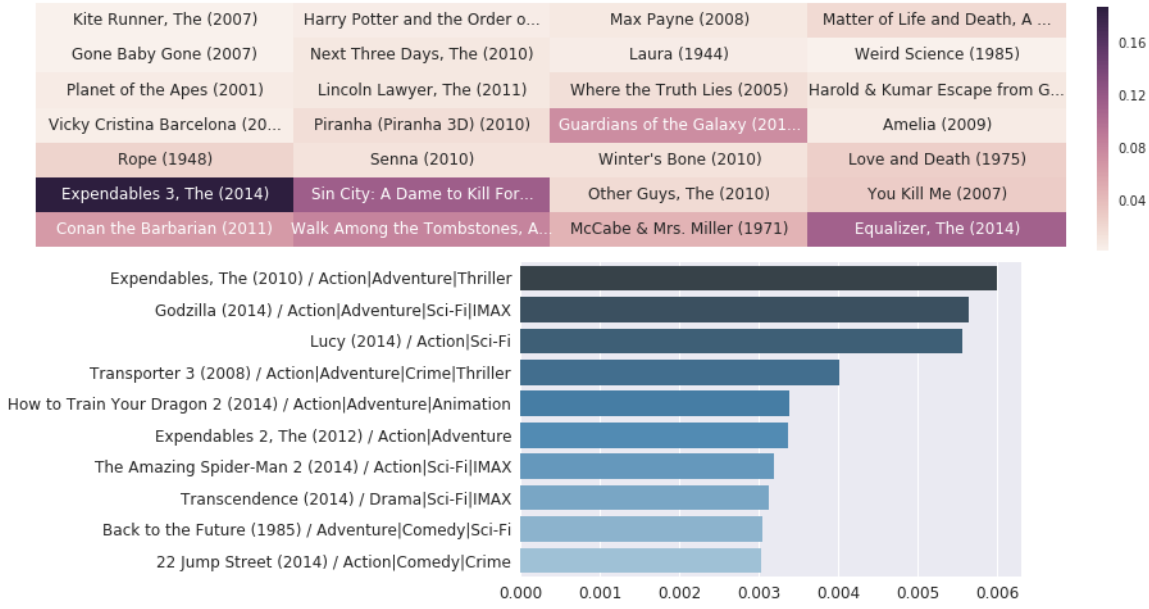

FIGURE 6.9: Attention mechanism to embeddings (linear). Top: History of rated movies by the user, the order goes from left to right, row by row. Color indicates the α weights. Darker color indicates more attention to the specific movie. Bottom: top 10 predicted movies.

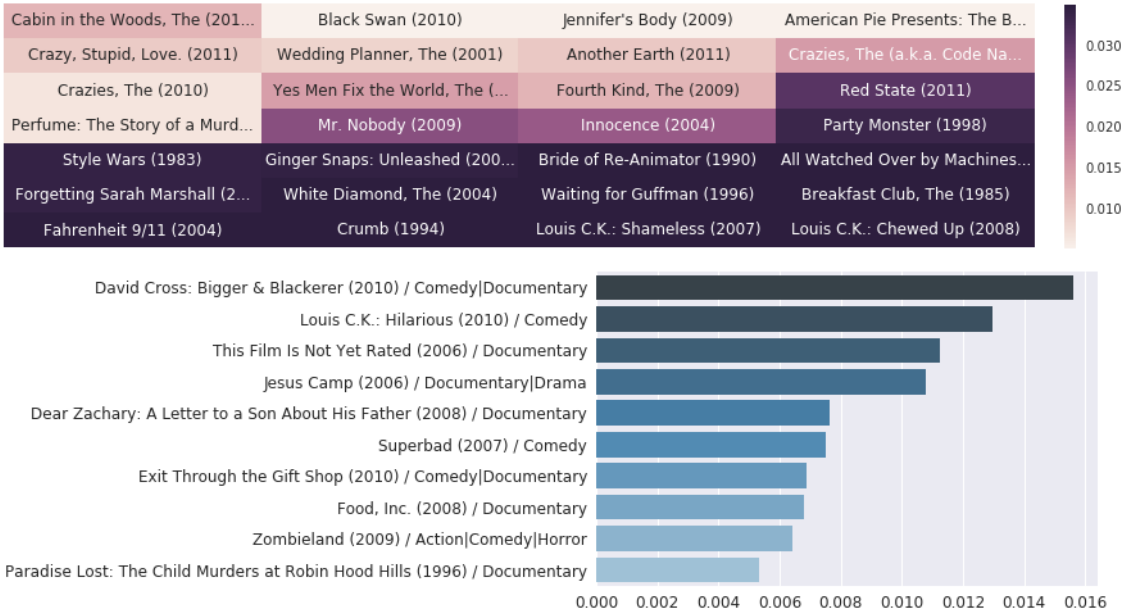

FIGURE 6.10: Attention mechanism to embeddings (nonlinear). Top: History of rated movies by the user, the order goes from left to right, row by row. Color indicates the α weights. Darker color indicates more attention to the specific movie. Bottom: top 10 predicted movies.

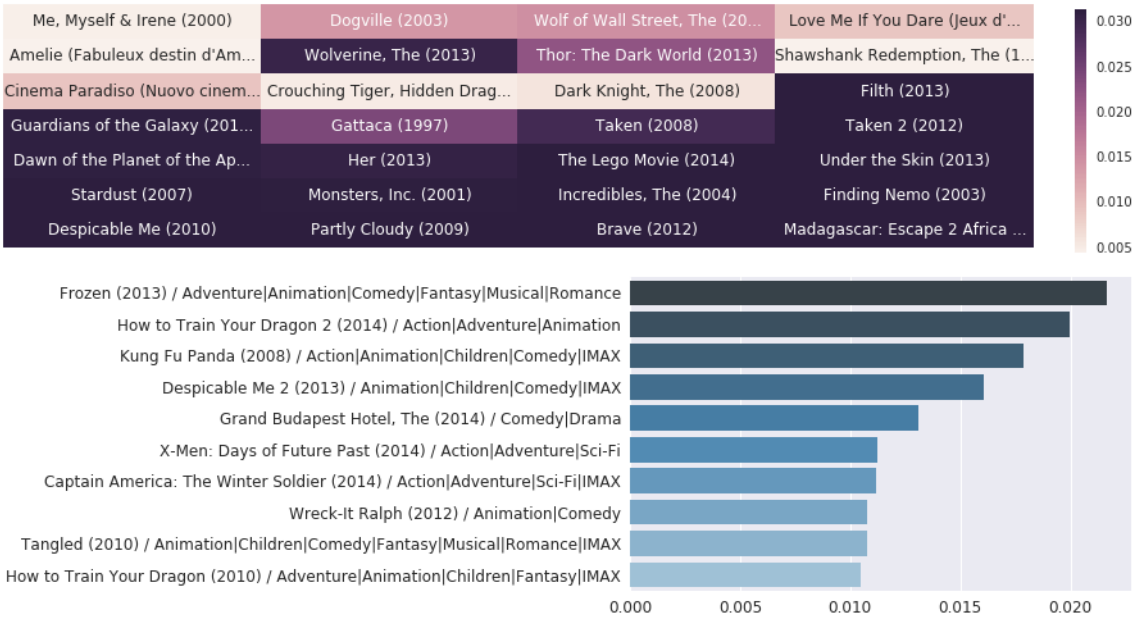

FIGURE 6.11: Attention mechanism to embeddings (nonlinear). Top: History of rated movies by the user, the order goes from left to right, row by row. Color indicates the α weights. Darker color indicates more attention to the specific movie. Bottom: top 10 predicted movies.

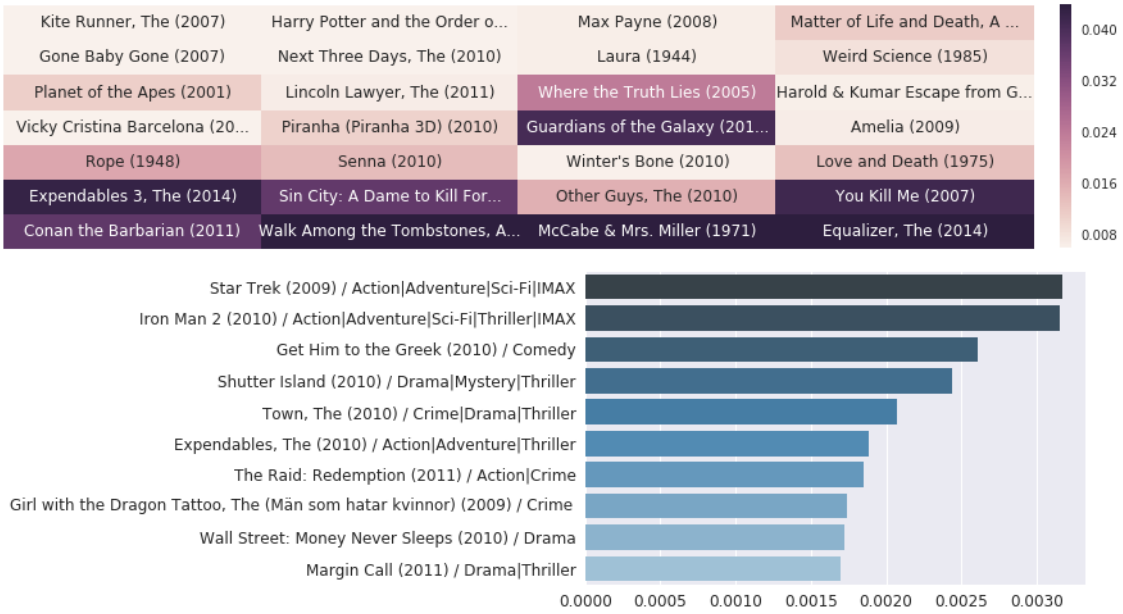

FIGURE 6.12: Attention mechanism to embeddings (nonlinear). Top: History of rated movies by the user, the order goes from left to right, row by row. Color indicates the α weights. Darker color indicates more attention to the specific movie. Bottom: top 10 predicted movies.

state, it may be a signal that this particular movie has a strong influence in the final prediction. In this case, we see similarities to the attention to the embeddings with non-linear weights, as the attention is splitted equally in several movies. That seems to lead again to more variety in the predictions. For example, there is only one Louis C.K. show in the top 10 predictions in the first example 6.13, and we can find non children movies in the top 10 predictions in the second example 6.14.

Finally, figures 6.16, 6.17 and 6.18 show examples for the attention mechanism with attention to the hidden state with linear weights (`RNN-Att-HS-Lin`). We observe that the model seems to have a tendency to focus strongly in the last hidden state, which happens in many examples. In this case, it is difficult to know if the model is learning that the last movies are more important and using these movies for the predictions, or if the model is carrying information of the whole sequence to the final hidden states. Therefore, in this case we cannot confirm that the attention weights are useful to detect which movies are more important in the prediction.

In general, we find that using attention to the embeddings with linear weights provides a good insight about why the model is predicting a set of movies as the one with highest probability. This method provides a few movies as the most important to the model which reflect similarities with the top predicted movies. The models with non-linear attention weights provide bigger sets of movies that are important for the model to make the predictions. We observe that this helps to have more variety in the predictions, although having many movies with the same attention weight can make the model less useful for the explanation purpose. We believe that having more variety on the predictions might be the reason of why the non-linear methods achieve a better precision-R. Considering just one or few movies implies that the top predictions are all highly related to these few movies. In some users, these can be good predictions in the short-term, as they could be interested in movies related with the last ones they watched. For example, if a user watches a movie related with *Artificial Intelligence*,

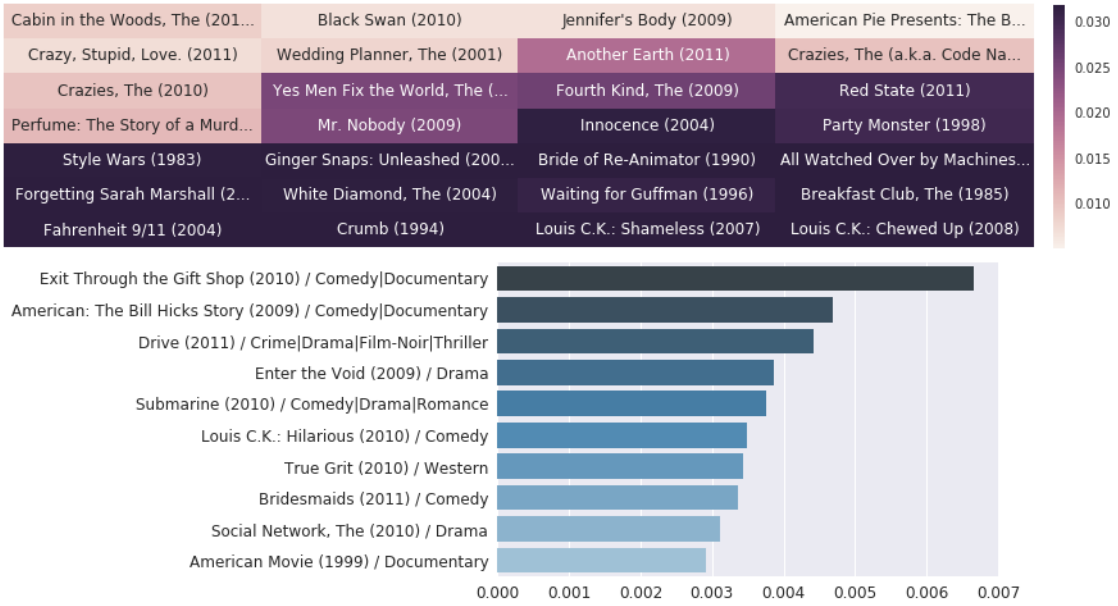

FIGURE 6.13: Attention mechanism to hidden state (nonlinear). Top: History of rated movies by the user, the order goes from left to right, row by row. Color indicates the α weights. Darker color indicates more attention to the specific movie. Bottom: top 10 predicted movies.

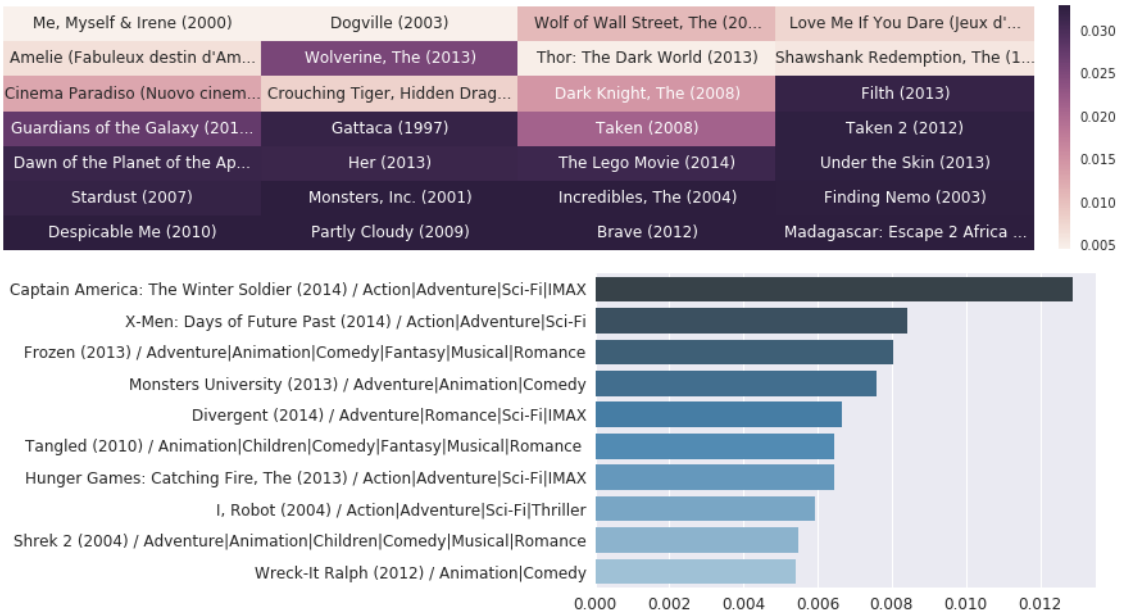

FIGURE 6.14: Attention mechanism to hidden state (nonlinear). Top: History of rated movies by the user, the order goes from left to right, row by row. Color indicates the α weights. Darker color indicates more attention to the specific movie. Bottom: top 10 predicted movies.

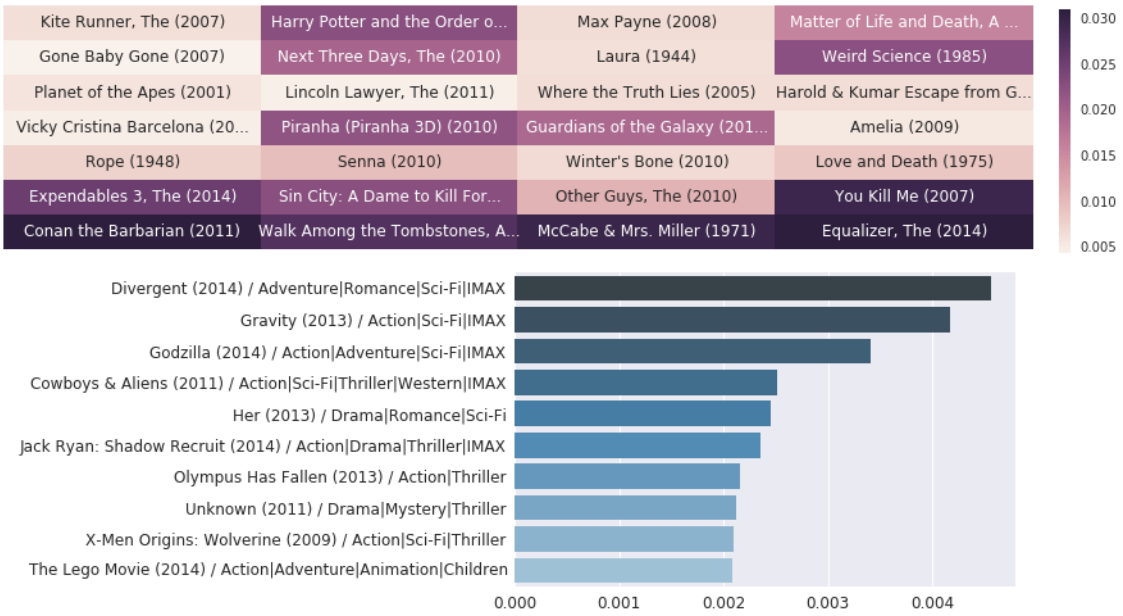

FIGURE 6.15: Attention mechanism to hidden state (nonlinear). Top: History of rated movies by the user, the order goes from left to right, row by row. Color indicates the α weights. Darker color indicates more attention to the specific movie. Bottom: top 10 predicted movies.

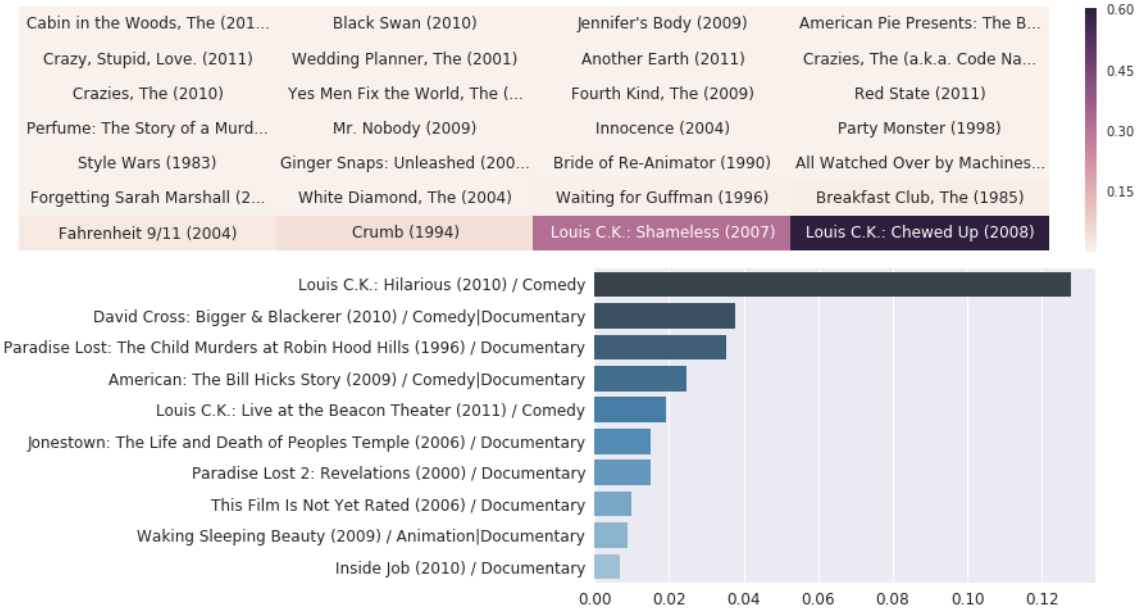

FIGURE 6.16: Attention mechanism to hidden state (linear). Top: History of rated movies by the user, the order goes from left to right, row by row. Color indicates the α weights. Darker color indicates more attention to the specific movie. Bottom: top 10 predicted movies.

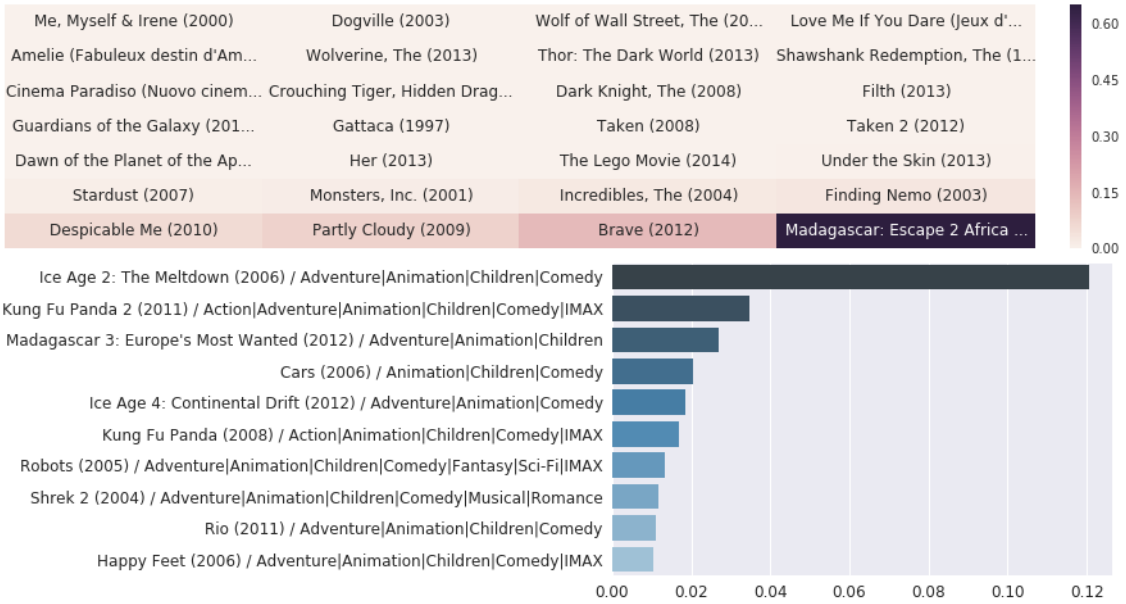

FIGURE 6.17: Attention mechanism to hidden state (linear). Top: History of rated movies by the user, the order goes from left to right, row by row. Color indicates the α weights. Darker color indicates more attention to the specific movie. Bottom: top 10 predicted movies.

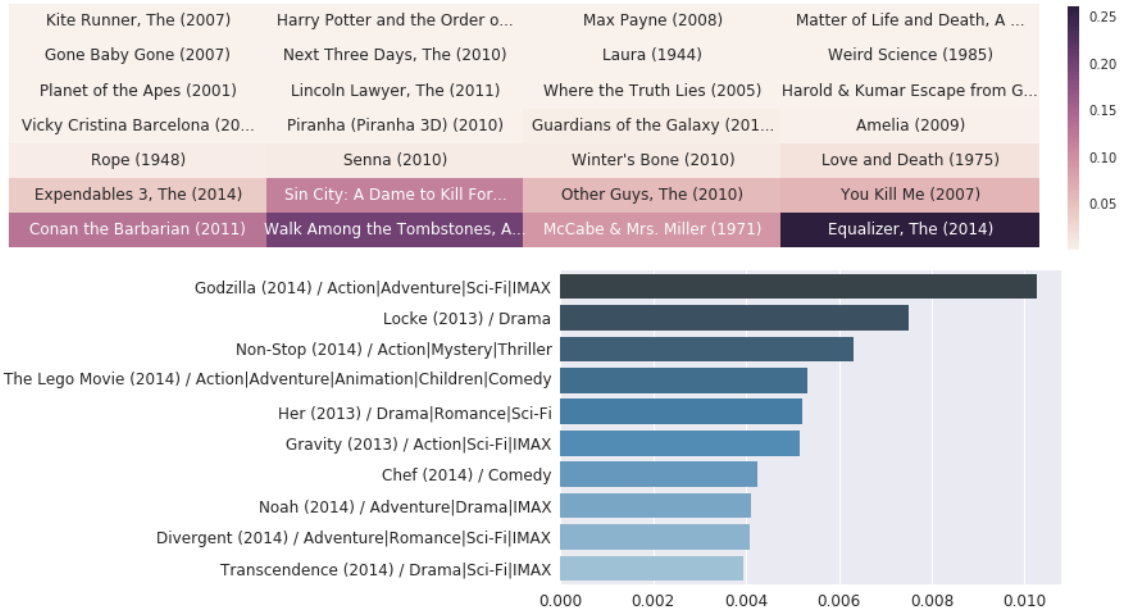

FIGURE 6.18: Attention mechanism to hidden state (linear). Top: History of rated movies by the user, the order goes from left to right, row by row. Color indicates the α weights. Darker color indicates more attention to the specific movie. Bottom: top 10 predicted movies.

he may be interested in other movies of this topic in short-term. However, after some months the interest for this topic could diminish. A top 10 based on distinct kind of movies is likely to have more matches with the real watched movies for a user than a top 10 based only on movies related with *Artificial Intelligence*. Finally, the model with linear attention to the hidden state normally focuses most of its attention to the last hidden states. There are two cases that can explain this behavior. The first is that the model is really focusing on these movies, as the last ones could be the most important to determine which movies the user will be interested. The second is that the model is just carrying all the information from previous movies, so the last hidden states contains the information about all the previous movies. In this last case, the attention weights cannot be used to know which are the most important movies for the predictions.

Chapter 7

Conclusions and future work

In this thesis, we explored the use of Recurrent Neural Networks to predict customer behavior from past interaction data. In the first experiments with the Santander dataset, we observed that RNNs can be very effective to model customer interaction data to predict items that the user will purchase in the future. In this data, the RNN obtained a slightly better mean *recall*@3 than a logistic regression model with the same information, and much better than the frequency baseline.

Then, we explored RNNs in a scenario where we have a high number of different items and where using embeddings could lead to a potential improvement. As we have seen, the embedding methods improved substantially the simple RNN model in short-term and long-term predictions. The different embedding methods seem to be able to represent some information in the embeddings such as the time when the movie was released, genres or saga. Word2vec seems to represent this kind of information slightly clearer, but the other methods are able to encode the necessary information to achieve the same performance in this task. Possible future research would be to explore methods to learn embeddings using item data (e.g. year of the movie, director, etc.) or using knowledge graph embeddings [7, 15].

We continued our research exploring attentional mechanism as a tool to explain the predictions made by the models. In our experiments with the Movielens dataset, the attention to the embeddings with linear weight provides a single movie or few movies that are most important to the model to make the prediction. As we have seen, the movies that the model focuses are related to the predictions, however, the fact of only focusing on a few movies could hurt the long-term prediction performance as it reduces the variety of the predictions. On the contrary, the attention with non-linear weights focuses on more different movies, allowing to the model to predict different kinds of movies, which seems to improve the long-term prediction performance. A possible negative aspect is that focusing on many movies could make the method less useful to explain the predictions.

An interesting area of research would be how to represent effectively time in RNNs. We have seen that RNNs can be very powerful to represent sequential data. Most of their success has been in Natural Language Processing, where the notion of time does not exist (there is no time between word by word, they are just regularly sampled one after the other). However, in tasks such as representing customer interactions, there is an amount of time between each interaction. This time could contain important information, since is not the same if an interaction happens a few seconds after the previous one than if it happens after a year. We have seen that RNNs do not contain this information, and they are only able to represent the order of the sequence of interactions. In [9] a feature is added to the input vector indicating the time from the previous visit of the patient. However, this method does not seem a "natural" way to represent the time. In [29] the authors affirm that RNN models are ill-suited to process irregularly sampled data and they introduce the Phased LSTM model, an extension

of the LSTM which adds a new time gate. This model naturally integrates inputs from sensors of different sampling rates and is reported to achieve faster convergence than regular LSTMs on tasks which require learning of long sequences. We believe that processing asynchronous sequential data and representing time effectively is an interesting area of research and can potentially lead to better models.

Related to this, another direction of research is how to model data which changes at a different rate. For example, a user can have a set of long-term preferences that varies slowly through the time, and a temporal short-term set of preferences that changes at a faster pace. In [14] a deep neural network is used to combine the long-term static and short-term temporal user preferences. Their model is composed by several components: they include two LSTM to capture daily and weekly user temporal patterns, another LSTM to capture global user interests, and a Deep Semantic Structured Model to model user static interests. The model is jointly optimized and is reported to outperform other baselines such as UserKNN.

Although we see potential for improvement, we believe that the models used in this thesis can be very effective to model customer behavior from interaction data.

Bibliography

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `http://tensorflow.org/`.

[2] J. Sun, J. Kulas, A. Schuetz, E. Choi, M. T. Bahadori and W. Stewart. "RETAIN: An interpretable predictive model for healthcare using reverse time attention mechanism". *NIPS* (2016).

[3] Kyunghyun, Bahdanau Dzmitry, Cho and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". *NIPS* (2013).

[4] Balazs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. "Session-based Recommendations With Recurrent Neural Networks". *ICLR* (2016).

[5] Oren, Barkan and Noam Koenigstein. "Item2Vec: Neural Item Embedding for Collaborative Filtering". *arXiv Preprint arXiv:1603.04259* (2016).

[6] Robin Devooght, Hugues Bersini. "Collaborative Filtering with Recurrent Neural Networks". *arXiv:1608.07400* (2016).

[7] N., Garcia-Duran A., Weston J., Bordes A., Usunier and O Yakhnenko. "Translating embeddings for modeling multi-relational data". *n Advances in Neural Information Processing Systems 26* (2013), 2787–2795.

[8] B. Carterette. "Multiple testing in statistical analysis of systems-based information retrieval experiments". *ACM Transactions on Information Systems (TOIS)* 30 (2012), pp. 1–34.

[9] M. T. Bahadori, E. Choi, and J. Sun. "Doctor ai: Predicting clinical events via recurrent neural networks". *1st Machine Learning for Healthcare Conference* (2015).

[10] Alexander M. Rush, Sumit Chopra, and Jason Weston. "A neural attention model for abstractive sentence summarization". *EMNLP* (2015).

[11] C., Cho-K., Chung J., Gulcehre and Y Bengio. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". *arXiv preprint arXiv:1412.3555* (2004).

[12] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. "Distributed Representations of Words and Phrases and their Compositionality". *ICLR* (2015).

[13] E. Duchi J., Hazan and Y Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". *Journal of Machine Learning Research* 12 (2011), 2121—2159.

[14] Yang Song, Ali Mamdouh Elkahky, and Xiaodong He. "Multi-rate deep learning for temporal recommendation". *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval* (2016), 909–912.

[15] B. Yang, W. Yih, X. He, J. Gao, and L. Deng. "Embedding entities and relations for learning and inference in knowledge bases". *arXiv preprint arXiv:1412.6575* (2015).

[16] F. Gers and J. Schmidhuber. "Recurrent nets that time and count". *Proc. IEEE-INNS-ENNS International Joint Conf. on Neural Networks* 3 (2000), pp. 189–194.

[17] A. Graves. "Generating sequences with recurrent neural networks". *Arxiv preprint arXiv:1308.0850* (2013).

[18] A. Graves and J. Schmidhuber. "Framewise phoneme classification with bidirectional lstm and other neural network architectures". *IEEE Transactions on Neural Networks* 18(5) (2005), pp. 602–610.

[19] S. Hochreiter and J. Schmidhuber. "Long short-term memory". *Neural Computation* 9 (1997), pp. 1735–1780.

[20] Matthew J. Johnson, and Alan S Willsky. "Bayesian nonparametric hidden semimarkov models". *The Journal of Machine Learning Research* (2013), pp. 673–701.

[21] C. Elkan, Z. C. Lipton, D. C. Kale and R. Wetzell. "Learning to Diagnose with LSTM Recurrent Neural Networks". *ICLR* (2016).

[22] T. Mikolov, M. Karafiat, L. Burget, J.Cernocky, S. Khudanpur. "Recurrent neural network based language model". *Proceedings of Interspeech* (2010), pp. 1413–1421.

[23] Diederik P. Kingma and Jimmy. Adam Ba. "Adam: a Method for Stochastic Optimization". *International Conference on Learning Representations* (2015), 1—13.

[24] Ting, Jing JIANG, LI Yang, LIU and Liang ZHANG. "Hashtag recommendation with topical attention-based LSTM". *Proceedings of the 26th International Conference on Computational Linguistics* (2016).

[25] Scott Linderman and Ryan Adams. "Discovering latent network structure in point process data". *In ICML* (2014), pp. 1413–1421.

[26] Thomas Josef Liniger. "Multivariate hawkes processes". *PhD thesis, Diss., Eidgenossische Technische Hochschule ETH Zurich* (2009).

[27] Young-Jun Ko, Lucas Maystre, and Matthias Grossglauser. "Collaborative Recurrent Neural Networks for Dynamic Recommender Systems". *Journal of Machine Learning Research* (2016).

[28] A. Graves, A. Mohamed, and G. Hinton. "Speech recognition with deep recurrent neural networks". *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference* (2013), pp. 6645–6649.

[29] M., Neil D., Pfeiffer and S.-C Liu. "Phased LSTM: Accelerating Recurrent Network Training for Long or Event-based Sequences". *Advances in Neural Information Processing Systems 29* (2016), 3882–3890.

[30] M. G. Ozsoy. "From word embeddings to item recommendation". *arXiv preprint arXiv:1601.01356* (2016).

[31] Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (2015), pp. 1412–1421.

[32] Joseph Turian, Lev Ratinov, and Yoshua Bengio. "Word representations: a simple and general method for semi-supervised learning". *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics* (2010), 384—394.

[33] Radim Řehůřek and Petr Sojka. "Software Framework for Topic Modelling with Large Corpora". English. *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. `http://is.muni.cz/publication/884893/en`. Valletta, Malta: ELRA, May 2010, pp. 45–50.

[34] Tobias Lang, Matthias Rettenmeier. "Understanding Consumer Behavior with Recurrent Neural Networks". *MLRec* (2017).

[35] Y. Bengio, P. Simard, and P. Frasconi. "Learning long-term dependencies with gradient descent is difficult". *IEEE Transactions on Neural Networks* 5 (1994), pp. 157–166.

[36] J., Carterette B. Smucker M. D., Allan. "A comparison of statistical significance tests for information retrieval evaluation". *Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management (CIKM)* (2007), 623–632.

[37] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting". *JMLR* (2014).

[38] L.J.P. van der Maaten and G.E. Hinton. "Visualizing High-Dimensional Data Using t-SNE". *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605.

[39] I. Sutskever, O. Vinyals, and Q. V. Le. "Sequence to sequence learning with neural networks". *Advances in neural information processing systems* (2014), pp. 3104–3112.

[40] Chun-Jung Lin, Fan Wu, and I-Han Chiu. "Using Hidden Markov Model to predict the surfing users intention of cyber purchase on the web". *Journal of Global Business Management* (2009).

[41] Yong K Tan, Xinxing Xu, and Yong Liu. "Improved Recurrent Neural Networks for Session-based Recommendations". *arXiv:1606.08117v2* (2016).

[42] R. Jozefowicz, W. Zaremba, and I. Sutskever. "An empirical exploration of recurrent network architectures". *ICML* (2015), 2342–2350.

[43] M. D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method". *arXiv:1212.5701* (2012).

[44] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. "Show, attend and tell: Neural image caption generation with visual attention". *ICML* (2015).