Deep Learning partitioning given user-item identifiers only in Recommender Systems

Joshua Uwaifo

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2018

Abstract

In this thesis, a deep learning procedure for partitioning the user and item-space in a Recommender Systems problem has been proposed. The beauty of this procedure is that it only needs user and item ids. Furthermore, the experiments have been carried out within 5 epochs, implying fast performances. This procedure is firstly a trained autoencoder (2-1-2) with identity activations. Afterwards, the 2 units in the last layer are replaced with 1 unit to make predictions in just 1 extra epoch, also with identity activations. Both parts were trained using the rmsprop loss function.

In addition, other deep learning procedures have been researched and evaluated, as well as other non-deep learning models. The results from these architectures are evaluated on the quality of predictions made on a popular movie database, MovieLens, and Jester, a dataset of ratings for jokes. The metric used to evaluate is primarily the root-mean square error. To give a scope, there have been over 150 experiments carried out.

This paper shows that the deep-learning procedure proposed generates recommendations to a level at least as good as the state of the art collaborative filtering technique, Singular Value Decomposition. This is in addition to the other deep learning procedures investigated as well as simple baseline experiments like the Regression tree model.

keywords*: MovieLens, Jester, Coarse Preferences, user-item space partitioning, Recommender Systems, deep learning, SVD*

Acknowledgements

From the project's formation, a number of individuals have personal interest in the project and can be considered as stakeholders.

Firstly, the thesis supervisor, Pavlos Andreadis. This man's impact on this project cannot be understated. Apart from providing the thesis the project stems from [3], his compassion, care, and insight when offering suggestions, support and feedback throughout the project have been astonishing. In addition, there are two other students also under Pavlos' supervision approaching this project from different perspectives. These are Yukun Yuan and Owen Yangyiwen. The group conversations have been invaluable.

Furthermore, thank you so much Daniel Namu-Fetha, for helping to proofread, make corrections with writing and just be an amazing support at stages when I felt like giving up. Thank you to the Crossroads family and my brothers and sisters for your amazing love and support. I really appreciate all the help, especially with the passing away of my Grandma, amongst the many battles faced this Masters year.

Finally, a big shout out to my best-friend, HS, big bro, JC and my Father. You guys are such an unstoppable team. Thank You God for the loving friends and family You put around me. This couldn't have been done without all your help. I love you all!

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Joshua Uwaifo*)

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces this body of work. This document centres around recommendation systems. The objectives and results achieved in this recommender problem have been detailed. Finally, the structure for the remainder of the document as well as the author's background prior to starting this project has been outlined too.

1.1 Motivation

This section deals with the general idea behind the solution proposed, stemming from the prior work in Coarse Preferences. Coarse Preferences, detailed later in this section, reduced computational costs especially on-line and in complex decision-making scenarios. The results of using Coarse Preferences as a procedure [3] was good if not better than prior performances when making a recommendation.

Originally, the author of [3] created a Regression Tree model for the partitioning, which is an integral part of his approach, but there were some issues like the strong linearity assumption that the regression tree model gives. This work investigates and proposes deep learning procedures but on a different Recommender System problem with inspiration taken from [3] for a simple baseline. This Recommender System problem is given as input only unique identifiers for the item and users and nothing else.

The author hypothesises that the deep learning procedure proposed will perform better than, Singular Value Decomposition, the state of the art for the rating prediction

problem with {user id, item id, rating} tuples as input. Furthermore, to serve as a baseline, a simplified regression tree partitioning based model inspired by [3] is used.

1.1.1 Recommender Systems

With the expansion of the Internet and the growing amount of information that circulates the cybernetic web, people are exposed to a vast amount of material online[38]. Making sense of the information presented in real time is a challenge. Even harder is making the decision of which pieces of information are relevant and which can be disregarded. It was for this challenge that Recommender Systems were developed.

Mahmood and Ricci[23], define Recommender Systems as ”intelligent applications which assist users in information-seeking tasks”. This assistance is achieved by utilising ”user need and preferences” to suggest items that best suit these preferences[23].

Examples of Recommender Systems include Netflix’s show recommendations[5], Amazon Kindle’s book recommendations[21] or Asos’ shopping recommendations[24]. The users of these systems would be consumers of the respective medium and the items would be the electronic goods offered by each outlet. This information constitutes the catalogue from which the Recommender would recommend an appropriate item to a particular user.

Typically, Recommender Systems have a sizeable catalogue of items (m), a subset of which have preferences given to them by a large number of users (n).

A rating system is usually the preferred metric by which users[20] quantify their preference for some item. Ratings are commonly either binary, discrete or continuous. A binary rating scale consists of values such as good or bad. A discrete rating scale could feature a discrete number of symbols that represent the ”wholeness” of a users preference, like the 5 stars scale, for example, 3 out of 5 stars. A continuous rating scale refers to an item being rated as any decimal value within some numeric interval. An example of this is a rating of 0.3 on a rating scale of -10 to 10.

In practice, only a few items are rated by a given user. This has a huge impact on the space of all possible users and items. In the entirety of this space, only a small proportion would actually contain information about actual user preferences. This sparsity of information could cause a limited representation of a particular of item or user.

More formally, this user-item space with preferences is expressed in a rating matrix. A rating matrix is an n by m matrix, named R, where $r_{u,i}$ is the rating of user u for item i. As alluded to earlier, the small amount of information in the user-item space makes the rating matrix typically large and sparse.

ratings	item 1	item 2	item 3	item 4	item 5	...	item m
user 1	2	**5**		1	**5**	...	
user 2		2				...	2
user 3	2		5			...	
user 4	1	**3**		2	**3**	...	3
user 5	4					...	
⋮	⋮	⋮	⋮	⋮	⋮	...	⋮
user n		**2**			**2**	...	

Table 1.1: Example of a (sparse) rating matrix and a cluster of items

The input to the recommender system can be represented as a rating matrix, as seen in Table 1, or a flattened version of it. This flattened version corresponds to $\langle$user id, item id, rating$\rangle$. Whilst, the rating matrix is easier to understand at glance, the flattened version takes less space. This makes it a more appropriate input especially when the rating matrix is large. This project makes use of the flattened rating matrix.

1.1.2 Coarse Preferences

It can be advantageous to partition the space of possible users and items in the recommender system framework. The result is a smaller cluster of items where each cluster serves as a representative of the similar items. In the case of this project, each cluster (group) would contain items that received similar ratings for a given user.

Suppose a simple Recommender System where a 5-star rating system is employed to gauge users' preferences for some generic items ie $\{1, 2, 3, 4, 5\}$. The clustering of the item-space will construct various groups of items that were awarded similar scores by one particular user.

The principle behind coarse preferences is to learn such groupings from scratch. The idea, as seen in Table 1, is to group item 2 and item 5 together. This is because for

a given user the ratings for both those items are relatively close on the rating scale. This can be observed in user 1, user 4 and the n^{th} user, rating items 2 and 5 identically. This emphasises the compatibility of items 2 and 5 with those users, and how a Recommender System would recommend similar items to those particular users.

Coarse Preferences [4] is where each grouping represents a subset of the space of possible outcomes. In this case, the possible outcomes are the possible items and a grouping is a cluster of similarly rated items. The beauty in this is that each member of a group would share the same utility value (ie rating value) for a given user.

The effectiveness of this grouping is that one does not need to consider an entire space of items. Instead, one only needs to consider individual groups. When deciding on what items to recommend to the user, the Recommender System restricts its selection solely to a specific group.

This was proven [3] to reduce on-line computation costs because the Recommender System can split its view of the whole user-item space into user-item representatives for items with similar user preferences. This helps to ultimately make informed decisions using these representatives. For example, using the discrete rating system, a representative that contains only items with a score of 5 would be the only group needed for a Recommender System to recommend a new item that the user would most likely rate. This is computationally more efficient [3] that using all m items. The author of [3] implemented this idea using a (piece-wise linear) Regression Tree model.

A regression tree is a decision tree model. This model is a (piecewise) linear mapping of inputs into desired outputs. Each segment is constant. However, the issue is that not all problems assume an implicit linear relationship which is the strong assumption implied by the decision tree model. As a result, a deep neural network (DNN) procedure is proposed. Modifications of these ensure the possibility of learning non-linear and linear mappings. This would, therefore, allow for more effective embedding representations.

1.2 Objectives

This idea of intermediate clustering before making informed decisions is the inspiration. This project designs a deep learning procedure by either learning first, or si-

multaneously, an embedding which is then used to recommend items from a space consisting of possible items and users. This is especially significant as only identifiers are given to these users and items, ie user and item ids.

The final deep learning procedure chosen is based on the three options:

- Autoencoder: Normal vs Denoising vs Normal with regularised embedding
- Multi task learning vs 2 single tasks for reconstruction and prediction
- Selu vs Sigmoid vs Identity activation functions

The deep learning clustering-based procedure is investigated in more detail in the Design and Implementation sections. This is after further being introduced in the Research chapter.

The chosen deep learning procedure should combine an effective embedding for the Autoencoder reconstruction as well as make a well-informed recommendation. These recommendations are evaluated on 8 different datasets with different challenges posed. In addition, the deep learning procedure variants are optimised with prior constraints and fixed hyper-parameters in order to ensure the experiment is not biased.

Furthermore, these deep learning procedures are compared against 3 non-deep learning models that have also been investigated. The first model is the state of the art, Singular Value Decomposition (SVD), collaborative filtering model. Collaborative filtering is introduced in the Research chapter and SVD is detailed further in the Design and Implementation section. Two other models, which serve as the baselines are; a model that predicts the ratings as the mean rating from the training dataset and the regression tree. The regression tree was included due to the inspiration of the regression tree model from [3]. It must be emphasised that this regression tree is a simplified variation of the said model.

Finally, the problem investigated can thus be formulated as follows. Given a set of user id's, U, and a set of item id's, I, predict the rating, r a user, $u \in U$, would give to any item, $i \in I$.

To re-emphasise, this project does not make use of any item or user-specific features like user demographics in the problem format. All that is used are user or item identifiers in the form of integers.

1.3 Results achieved

The deep learning procedure proposed is as good as, when trained, the state of the art, SVD. The models were evaluated across 8 datasets. The good performance of the chosen deep learning procedure is shown in it outperforming SVD on 3 out of 6 testing experiments. The 2 other experiments were an exception as the model was not able to train, given the experimental constraints. In addition, the neural network performs exceptionally better when the rating metric is actually of continuous type.

Over 150 different experiments have been carried out, with the models that were able to train, achieving low root mean square errors in most experiments, evaluated on a held-out validation dataset. In addition, the held-out test performance has been documented too when compared with SVD.

These experiments were carried out across the 8 variants of the Jester and MovieLens datasets. The chosen deep learning procedure is a pre-trained autoencoder with 1 unit in the embedded layer and 2 units in the input/output layer. When the embedding is trained in just 5 epochs, the output layer is replaced with a unit for predicting the ratings. The activation function used throughout this procedure is the identity activation function.

1.4 Thesis structure

This thesis is a culmination of academic research, experimentation and project management. As a result, the project has been partitioned into research, methodology, and analysis.

1.4.1 Research

Chapter 2 focuses on the preparatory research and ethical considerations that are important for a well-rounded project, especially one with a software component. This includes a detailed breakdown of collaborative filtering, deep machine learning and learning an embedding as an unsupervised learning problem. In addition, this chapter references an existing solution which forms the basis for this thesis. As well as dealing with the potential legal, ethical and professional considerations of this project, especially with regards to the datasets chosen.

1.4.2 Methodology

The deep learning methodology proposed by this thesis and its experimentation component are the crucial contributions of this thesis. Chapter 3 calls attention to the computer constraints this body of work was under, as well as the resources and tools used. Furthermore, the design and implementation of the machine learning experiments carried out are of focus in this chapter.

1.4.3 Analysis

The last segment of this thesis evaluates the project in its entirety. In the Results and Analysis section, the reasoning behind the chosen deep learning procedure is detailed. This chapter analyses the results obtained from the experiments carried out. There is also a focuses on how the project was managed from inception to completion, in the final chapter. This chapter also highlights how the issue of Jester [14] having three datasets was solved in this project's life-cycle. Moreover, a look into the author's assessment of the project is seen. This concludes with the final chapter, reviewing the project and proposing future work.

1.5 Author's Background

The author's background in preparation for this project stems from Warwick's Discrete Mathematics undergraduate and Edinburgh University's Artificial Intelligence masters program. Focusing specifically on the master's degree, which the author is currently on, the courses taken were solely Machine Learning in nature. This was done to improve my understanding practically and theoretically to reach the required expertise needed for this project.

Chapter 2

Research

With the problem and proposal well defined, it was important to better understand the subjects involved in the problem. The subjects up for review are recommendation techniques involved in this thesis, deep machine learning as a field and the consideration of the issues associated to datasets used and the thesis as a whole.

2.1 Recommender System Techniques

Resnick and Varian defined what Recommender Systems are in their 1998 seminal article[37] and since then Recommender Systems have proved useful as a method for suggesting items a user might prefer. Recommender Systems can be categorised based on the information source the system users. There are three information or knowledge sources here[37]:

- Collaborative based knowledge
- User-based knowledge
- Content-based knowledge

Collaborative or social based knowledge revolves around information containing other users of a given task. This information can be behavioural, based on the demographics of the users like national heritages or simply opinions generated from online reviews, tags, and ratings. Individual or user-based knowledge focus only on the knowledge of the individual user in question, which could be received from the

queries that particular makes on a search engine like Google. Content-based knowledge sources are ones which contain information about the items that are being recommended. This takes into account contextual information about the items for a certain domain. A recommender system can make use of single knowledge bases or a combination of them. The various recommendation system techniques stem from this.

A recommendation technique is an algorithmic approach to generating recommendations using a set of knowledge sources, as defined by Felfernig and Burke in 2008[12]. The common categorisation of these techniques are the following:

- Collaborative
- Content-based
- Utility-based
- Demographic
- Knowledge-based

The two techniques of focus in this thesis are collaborative and utility-based techniques. Collaborative techniques[37], commonly known as collaborative filtering, take as input ratings from users of items. The procedure is to identify similar users and derive their ratings for a given item. In this thesis, this is the technique that the deep learning procedure stems from as it has user-item-ratings as its input.

Another technique, that inspired this thesis is a utility-based technique. Utility-based techniques take as input a utility function over items based on user preferences. [12] The author of [3] where the regression tree baseline stems from used coarse preference elicitation, alluded to in the Introduction, as the utility function. The idea in [3] was to learn a computationally efficient representation of the item space without loss in recommendation quality using the coarse mapping restriction.

2.1.1 Collaborative Filtering

Collaborative filtering (CF) is a category of recommender systems techniques. These techniques approach the problem of making recommendations based on users' past

behaviour. The information presented is normally expressed as preferences of a set of users on a set of items [40].

A typical example is the 2006 Netflix Competition[17]. This was a popular competition concerning making predictions about movie preferences based on ratings from previously watched movies[40].

movie / user	Alice	Bob	Chuck	Dan
The Matrix	5	5	?	?
Blade Runner	5	?	1	?
Click	1	?	4	?
Ex Machina	?	4	?	?

Table 2.1: Movie preferences [40]

As mentioned in the Introduction, typically we have a partially observed rating matrix. Most entries will be unavailable as users will not have rated all movies. Collaborative filtering techniques are of two categories:

- Memory-based approach
- Model-based approach

2.1.1.1 Memory based CF

The memory-based approach finds similar users based on a similarity metric, such as cosine similarity. After, this similarity is then used to take a weighted average of ratings. The advantage of the memory-based approach is the easy implementation and interpretability of results. However, a drawback is that performance tends to reduce when the rating matrix is really sparse. The deep learning procedure maintains the advantages by making use of only the identifiers whilst been able to scale to large datasets.

2.1.1.2 Model based CF

The model-based approach uses machine learning to find the user ratings of items yet to be rated. The state of the art approach is Singular Value Decomposition (SVD),

detailed later in the Design section. This is emphasised in the Netflix Prize winner making use of a number of SVD models in their final solution. The benefit of the model-based approach typically revolves around dimensionality reduction. These models reduce the input dimensionality, for example, the deep learning procedure proposed reduces the 2D input to 1D (the embedding). Typically, the dimension is more than 100 but as highlighted, here there are only 2.

The downside tends to be focused around the Cold start problem [25]. This problem is that typically such models will not cope when new objects are added to the system. For example, new users can arrive or new movies are released, if referencing the Netflix problem. Also, a new movie by definition would not have ratings. This problem has been dealt with partially by the way the dataset has been partitioned. In addition, the use of identifiers only in the problem description makes the input as generalised as possible.

2.1.2 Recommendations using user-item identifiers as input

Nowadays, research in Recommendation System techniques tends to focus on taking contextual information into account rather than using the identifiers[12]. This is possibly due to the notion that simplicity won't work but as seen with the matrix factorization technique, SVD, simplicity is effective.

The work the author proposes has a unique twist but needs a state of the art to compare. The work that serves as prior work stems around matrix factorization in a collaborative filtering framework. This is because when the rating matrix is flattened, it forms the itemId-userId-rating input this problem is specified by. As a result, the only proper comparison can be the state of the art for solving matrix factorization which is the Singular Value Decomposition technique[31]. Further detail about SVD being the state of the art for matrix factorisation can be seen in chapter 3.7.

Experiments around MovieLens tend to focus on the smaller version datasets and not 20M. There are examples of papers that either focus on MovieLens 20M with the desired input format this procedure follows or a smaller version of MovieLens but without the desired input.

An example of an experiment that appeared to have results for MovieLens 20M is the Hybrid Recommender System based on Autoencoders paper, from 2016[43]. This

paper approached the Matrix factorization problem by using an end-to-end Autoencoder approach for collaborative filtering. This approach, however, used contextual information as well as approaching the problem by normalising the input thereby removing the interpretability the author, here, tries to prevent. The results although promising in this paper, but the complexity of the model and lack of interpretation is a limitation. In addition, it was not clear how the data partitioning went and if the result is from a held-out testing dataset or not.

More promisingly, a probabilistic approach to solving MovieLens has been proposed by the authors of [22]. Here, they proposed a Gaussian processes based model that aggregates the contextual information as well as the traditional input to the matrix factorization approach. This like [43] showcases promising results, however, it works on a MovieLens with 100,000 ratings instead of the 20 million (full) dataset. On the other hand, this paper furthers the idea of SVD being state of the art by making the exact claim that it is, generally, not just for the matrix factorisation instantiation of Collaborative filtering.

2.1.3 Recommendation baseline

There are different problems one could approach in a recommender system framework. One such example is the *Multi-user Coordination through set recommendation* scenario[2]. This procedure involves using an efficient intermediate representation. This representation formulating an optimisation problem by making In this scenario, a recommender system tries to adapt in real time to a single interaction from a given user. This idea of practically adjusting recommendations as a user inputs his or her information is called on-line learning. This on-line learning of user preferences from the first visit a user has is generally thought of as a complex task.

In spite of this complexity, recent research by the author of [3] showcased that the computational time to learn and make recommendations can be reduced. This method involved reducing the space of possible items by grouping items based on user preferences. The results showed that the recommendation system decreased it's time to make decisions on-line. This was also done without harming the quality of recommendations made.

The approach in Andreadis, 2018 [3], was to partition the item space using a regression tree. This is a model which assumes (piece-wise) linearity in the data. The linear

segments in this model are constant, hence why it is referred to as a linear model. Generally, a regression tree is a decision tree whose output can take on continuous values [7]. This is because the preferences fed to the linear model created were ratings. The ratings were assumed to take continuous values in his experiments. The output type being continuous, as well as the strong assumption of linearity, are two possible limitations. These help to inspire the author's approach in this paper. Specifically, as the author of [3] did, the ratings are also assumed as continuous whether originally of a discrete or continuous rating scale. In addition, a simplified regression tree is also added to the evaluation as a finishing touch.

Finally, the notion put forth by the author is that there are more generalisable ways of grouping items without strictly assuming linearity [3]. However, the deep learning procedure developed emphasise that the linearity assumption imposed by the author of [3], is an effective one.

2.2 (Deep) Machine Learning

This section deals with the field of machine learning as well as research into neural networks. Machine learning is the study of systems and algorithms such that computers can learn from data. Tom Mitchell in 1997 [26], defines a computer learning from data as follows:

"A computer program is said to learn from experience E (data) with respect to some class of tasks T, and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." [26]

There are many examples of deployed machine learning (ML) systems. For example, recommendation systems (i.e. Hulu), recognition systems (i.e. XBox Kinect), Pattern Analysis (i.e. Google search) and many more. There are three main sub-fields of Machine Learning. These are Unsupervised, Supervised and Reinforcement Learning. In this thesis, the focus will be on the former two.

2.2.1 Unsupervised Learning

In an Unsupervised machine learning framework, the computer program learns a function, f_{unsup}, that describes the hidden structure in the dataset. This dataset, unlike

supervised learning, is an unlabelled training dataset $x_n \in R^D : n = 1,...,N$. An example of an unsupervised learning problem is clustering. Here, the task is to discover the underlying structure within the data by finding groups or clusters and not predict anything specifically. The learning of the embedding in the autoencoder reconstruction is an unsupervised learning problem, in this project.

2.2.2 Supervised Learning

Supervised Learning is the paradigm where the computer program learns a function, f_{sup} that maps an observation, $x \in R^D$ to its correct target, $t \in R$. D is the number of feature or attributes associated to the observation, x. x, here, is a d-dimensional vector and t is a scalar (1-dimensional vector) quantity. The machine learning models in supervised learning learn the function, f_{sup}, by using a labelled training dataset $(x_n, t_n) \in R^D \times R : n = 1,...,N$. N is the number of samples in the dataset. The target variable can be of a discrete type, making it a classification problem, or continuous, making it a regression problem. In this project, the prediction of the ratings, after the embedding has been learnt or simultaneously, is the supervised learning problem.

The entire procedure, supervised and unsupervised segment, will be unified as a deep learning procedure. Deep learning as a concept is introduced in the next subsection through artificial neural networks.

2.2.3 Artificial Neural Networks

Artificial Neural Networks (ANN) are a category of machine learning models motivated by the biological nervous system. They model the connections between neurons and the neurons themselves[11]. In this project, multi-layer perceptron (MLP) architectures are specifically investigated.

An MLP [6] consist of dense or fully connected layers of computational units or nodes. The first layer of units is called the input layer. The final layer of units is called the output layer. The intermediary layers consist of units, called hidden units. Appropriately, these layers are called hidden layers.

Furthermore, connections in an MLP contain no cycles and are directed forward, from the input layer to the output layer. These connections are called weights. This

interconnected structure of nodes and weights form a directed acyclic graph (DAG). During the training process, ANN's employ a technique called backpropagation[29]. Backpropagation works by modifying the weights between layers, by improving the error between the model's output and the expected output, until the model lands at an optimal weight configuration.

In this paper, different activation functions in the computational units are investigated. However, to maintain consistency in comparing models, certain components of the models have been fixed, like the weight initialisation strategy. These are further detailed in the Design chapter. It should be mentioned that the depth of a neural network is determined by the number of layers in the network.

In summary, neural networks can be defined under three categories, namely; the network topology, the neuron activity rule and learning rule. These categorisations and the variants which form the experimentation are detailed further in the Design section.

2.3 Legal, Ethical and Professional Issues

This section of the project details how the possible legal, ethical and professional issues have been resolved. The British Computing Society Code of Conduct [35] states that IT professionals should carry out their professional responsibilities with due care and diligence in accordance with the Relevant Authority's requirements whilst exercising your professional judgement at all times. In addition, members should conduct themselves in a that way will not create and violate any social, ethical or legal issues within the IT profession. [35]

The resulting breakdowns are as exhaustive as possible and deal with as many pitfalls as possible. These pitfalls include; ensuring intellectual property rights are not violated, dealing with user interacted data, privacy ethics, and good programming practice.

2.3.1 Intellectual Property Rights

This section deals with legal issues specifically around Intellectual property rights (IPR). The following quote from the British Computer Society serves the importance

as to why IPR must not be violated:

"Database rights are similar to a copyright on the contents and arrangement of a database. As such the database may be considered to be a literary work and copyright rules would apply to making extracts from it. Commercial use should not be made without the approval of the copyright owner." (Intellectual property rights and software, BCS)[42]

The datasets used, namely; Jester [14] and MovieLens[16], for this project although publicly available have to be referenced appropriately. The Jester dataset terms and conditions state that out of courtesy an official email should be sent outlining how it would be used. As a result, official correspondence has occurred between the author and Ken Goldberg[14], the author of the Jester dataset.

Generally, the author does not state or imply any endorsement from the University of Minnesota or the GroupLens Research Group for the MovieLens dataset. In addition, the user acknowledges the two groups involved namely; GroupLens Research Group and the University of Berkley. In addition, this project does not create a software that displays data to the public and the data is not being redistributed. The deliverable from this project is a developed research method for applying deep learning to recommendations. The only user interaction comes from how the data sources [14] [16] were generated. This is not an issue that relates specifically to this project. The preceding points have ensured that the author does not violate any intellectual property rights.

2.3.2 User Privacy Ethics

This section deals with the ethical issue of maintaining that a user remains anonymous. The datasets [14] [16] have been given anonymous unique identifiers to each user. Despite, this privacy could still be a problem as seen with the Netflix privacy fall-out [30].

Similar to the datasets used here, there was a rating matrix of user-item pairs with anonymous identifiers. However, the problem that existed was an adversary correlated the dataset with another publicly available dataset. This other dataset had sensitive information which combined revealed peoples ratings of pornographic content. This caused a deep intrusion of privacy, according to the author of [30]. However, a

court case result [34] said that Netflix did not violate privacy, emphasizing the different perspectives.

However, from investigating, it appears that no other dataset has been available that does so with the MovieLens and Jester datasets. Although this has not been a problem it is important to note the possibility of it being problematic.

2.3.3 Good Programming Ethics

This section deals with professional issues, specifically, the use of good programming practices. Potential users of the deliverable from this project include academic researchers, machine learning developers and any organization that intends to do recommendations.

Focusing on the machine learning development, the software experiments follow the BCS code of conduct [35] as well as good programming methodologies[36]. These methodologies include, namely; debugging and optimizing the code, ensuring the code compiles, ensuring the code is readable and well maintained by looking deeply into the APIs used. Thereby, ensuring a professional approach[35] to the code by delivering one that is well documented and maintains good programming practices i.e. the use of good naming conventions to variables[36] .

This ensures that software engineers in machine learning, could for example, easily transfer the idea and application the author has developed in this project, to their projects. Finally, the use of official Application Programming Interfaces (APIs) ensures keeping in line with data industry standards. An API can be defined as a collection of software functionalities created by developers that enable other developers to make use of such tools. For example, this thesis makes use of data processing tools using Pandas [33] and Numpy [32] api's. Thus, ensuring, that the data processing is in line with the industry standard.

Given these considerations made, the author has tried as best as possible, to ensure that legal ramifications have been sorted out. Thus, the author can proceed further into the experimentation aspect of the project. This is important because problems in this section would have caused changes to be made before starting the experimentation e.g. different datasets to be used.

Chapter 3

Design and Implementation

This chapter describes the design and implementation details of the project. These details are served from a practical and theoretical perspective.

3.1 Problem description

Before going into the deep learning procedure, the recommender system problem is described, alongside how the partitioning is learned. The recommender system problem of focus here is to fill up the rating matrix, by creating an effective model that takes all the preferential information available, user-item preferences, and generalise predictions of ratings for any user and item. This can be expressed as the following:

Given a set of user id's, U, and a set of item id's, I, predict the rating, r a user, $u \in U$, would give to any item, $i \in I$. The items, here, are jokes for Jester and movies for MovieLens. The ratings define a given user's preference for a joke or movie item. The format of the input data, to be precise is (user-id, item-id) as data features with the corresponding rating the label for the input tuple.

The beauty of the deep learning procedures proposed is that these are done by first or at the same time learning the partitioning. The partitioning learnt has been implied through the quality of the embedding learned in an autoencoder-based structure and also how well the embedding helps to make correct predictions about the ratings. The problem of coarse preference as indicated in the introduction has been mirrored to the quality of embedding produced, as determined by the quality of reconstruction of the input.

The model that learns the embedding first before learning to make predictions about ratings has been referred to as a “2 Single Tasks network”. The model that learns both the embedding and the rating predictions at the same time is referred to as the “Multi-task network”.

3.2 Resource and tools

This section will briefly detail the computing constraints, this thesis was subject to. The experiments for this thesis are implemented in Python 2.7[13]. Python is a popular general-purpose programming language with an expansive library of tools. This project uses Python for data pre-processing and cleaning, data analysis and (deep) machine learning.

The datasets are primarily transformed and loaded using the Pandas[33] package into NumPy arrays [32]. The non-deep learning models were created using two Python scikit libraries; scikit-learn and scikit-surprise. The SVD model was implemented using Surprise[45]. Surprise is a scipy[10] toolkit for building and testing recommender systems. Scikit-learn is a [44] library built on NumPy and SciPy, containing tools for data analysis and data mining. The Regression Tree and dummy regressor were built using Sci-kit learn.

Software experiments with computational complexities like Deep Learning tasks are confined to the hardware capacity of the machine it is utilized on. The deep learning procedures are built and optimised using Keras[9]. Keras is a high-level deep learning library written in Python. Keras in this thesis is built on Tensorflow[15]. Tensorflow, a Python Library, is an open source machine learning framework providing high-performance numerical computation.

The author’s primary computer for this project was the DICE machine provided at the University of Edinburgh’s, School of Informatics. The capability of this machine is seen in Table 6.1. The data extraction, transforming and loading process alongside the machine learning aspect of the project requires significant computational processing as well as memory storage. The author, thus, ensured that good programming conventions were maintained in accordance with Patrick Guio’s, Good Programming Practice [36].

Name	Edinburgh University DICE Machine
Operating System	Scientific Linux Release 7.4
Processor	Quad Core
Memory	16 GB
Disk Space	50GB + 1TB External Hard Drive

Table 3.1: Computer System Specifications

3.3 Tasks in Experiment phase

The project's experimentation phase can be split into tasks. These tasks are categorised as core or additional. In addition, included in this chapter are the dependencies between objectives. These tasks were alluded to in the project proposal [47].

1. Extract, with permission, MovieLens 20M dataset from GroupLens into 2 datasets[16] (core)
2. Extract, with permission, the 3 unique versions of the Jester dataset from Berkeley's education portal into 6 datasets[14] (core)
3. Transform and load data for cleaning (core)
4. Clean and pre-process data (core)
5. Partition data into train, validation and testing data splits (core)
6. Research, build and evaluate deep-learning procedures (core)
7. Research, build and evaluate the state of the art models (core)
8. Research, build and evaluate some baseline models (additional)
9. Choose model that generally outperforms other models across the 8 validation datasets (core)
10. Report and critique performance of the chosen model on held-out test datasets against state of the art (core)

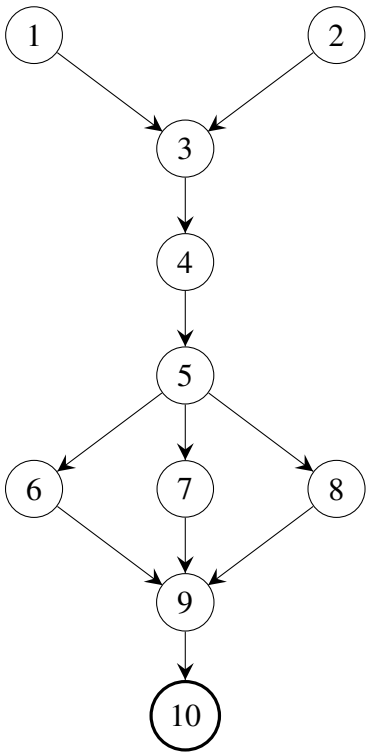

Figure 3.1: Relationship between tasks

3.4 Datasets

The datasets chosen for this project were not selected based on availability. Instead, they were selected to cover different recommendation scenarios. The reason for this is to ensure that the final procedure presented is adaptable to different tasks. After careful consideration, the datasets chosen were the MovieLens 20M [16] and Jester [14] datasets. Here is a summary:

1. MovieLens 20M
    - one dataset
    - discrete rating scale originally
    - matrix density of 0.52%
2. Jester
    - three datasets (v1, v2, v3)
    - continuous rating scale originally
    - matrix density of 31.5%

3.4.1 MovieLens 20M

MovieLens is a collection of user preferences to movies. These were gathered by the University of Minnesota's, GroupLens research group [16]. MovieLens comes in different sizes. there is a 1, 10 and 20 million version type. These numbers refer to the number of ratings made by users. These are appropriately called MovieLens 1M, 10M or 20M. This project used the 20M version. This version has around 140,000 users and 27,000 movies. The data has been collected over 20 years and makes use of the discrete rating scale $\{0, 0.5, ..., 4.5, 5\}$. As alluded to in the Introduction, the rating matrix is sparse. The sparseness of a rating matrix is measured by its density. If a rating matrix is full (or not sparse) it has a density of 100%, if it is empty it has a density of 0%. MovieLens 20M has a density of 0.52% and this extreme sparsity is it was chosen.

3.4.1.1 MovieLens preprocessing

In the preprocessing stage, a 2-dimensional feature vector is constructed for each user and item preference. The first feature is the user's identification represented as an integer and the second is the item identification, also represented as an integer. As this is already the input representation for MovieLens no further feature preprocessing was done.
MovieLens 20M was stored as a single comma-separated valued (CSV) file. Using Pandas the raw file was loaded. As the author of [3] assumed the ratings were continuous, the author did the same. The discrete rating scale had 11 intervals and was first digitized as such to result in $\{1, 2, ..., 10, 11\}$ using numpy's digitize function. Following, this the ratings were then converted into a continuous rating scale between 1 and 11. This dataset is called MovieLens 20M "continuous". The implementation can be seen in the file labelled "Data extraction...MovieLens20M.ipynb" in the Appendix section.

In addition, the author investigated the effect of binarizing the ratings. The way this was achieved was by initially thresholding the discrete ratings around the value 3. 3 is chosen as this is the median of $\{0, 0.5, ..., 4.5, 5\}$. After binarizing the values were they assumed continuous as before resulting in a continuous rating scale between 0 and 1. This dataset is called MovieLens 20M "binary". Similarly, the implementation is seen in the "Data extraction...MovieLens20M.ipynb" datafile.

3.4.2 Jester

The Jester dataset was chosen to counteract the extreme sparsity of MovieLens. Jester is a dataset developed at the University of California, Berkeley by Ken Goldbergs' research group[14]. This is a dataset originally of around 6 million ratings of 150 jokes by around 124,000 users. Jester is a more dense matrix with a density of 31.5%, which is denser than MovieLens 20M. Also, unlike MovieLens, Jester makes use of a continuous rating scale. This is scale between and inclusive of -10 and 10." Finally, unlike MovieLens, Jester came with three version; Jester version 1, 2 and 3.

3.4.2.1 Jester preprocessing

Jester v1 (version 1) came with three excel (xls) files. The first step, here, was to merge all these files into one file using Pandas' concat functionality. Jester v2 came as a generic data file (.dat). Jester v3 came in the form of an excel file. Each of these formats was readable by Pandas.

To ensure comparable experiments, a similar idea as posed in MovieLens was applied here. This idea is to obtain a "binary" and "continuous" dataset. The difference here is that the goal would be to get these 2 formats for each of the 3 Jester versions resulting in 6 extra datasets.

Each version of Jester was processed into a "binary" labelled file by thresholding at 0. As with MovieLens, they were assumed to be continuous resulting in a continuous rating scale between 0 and 1. The "continuous" datafile for each version of Jester was simply the original input as it already came with continuous rating scale between -10 and 10. These preprocessing steps are implemented in the Jupyter notebook files labelled "Data extraction...Jester-v(num).ipynb", where num refers to each version number.

3.4.3 Data partitioning

Partitioning the dataset is important when, namely; training machine learning models, evaluating different models and displaying a generalisation score for the chosen model(s). The data partitioning method used is the double holdout method.

The double hold-out method works as the following. Firstly, the labelled dataset is split into ”training” and testing sets. The testing set is used for computing the generalisation score of the chosen model. Secondly, the ”training” set is actually split into an actual training and validation set. The validation set is what is used to evaluate between different model. The training set, appropriately called such, is used for training the model.

As indicated, there are 2, MovieLens 20M labelled datasets, and 6, Jester labelled datasets. The result of the preceding data preprocessing applied to the 8 labelled datasets is the following, with statistics included.

dataset	samples	binary rating scale	continuous rating scale
Jester v1	2, 481, 816 - train 872, 272 - validation 872, 272 - test	[0 , 1]	[-10 , 10]
Jester v2	1, 056, 863 - train 352, 288 - validation 352, 288 - test		
Jester v3	4, 258, 128 - train 141, 936 - validation 141, 936 - test		
MovieLens 20M	12, 000, 157 - train 4, 000, 053 - validation 4, 000, 053 - test		[1 , 11]

Table 3.2: Train, validation and testing splits

An important consideration that follows relates to how to split up the data into the different datasets. The idea is to use as much data for training as possible[29]. However, there needs to be enough data for the validation and testing phases in order to choose a model and report an unbiased model performance[29]. After careful consideration, the author decided to use 60% of the data for training, 20% for validation and 20% for testing. This is also commonly referred to as a 60:20:20 data split. The data

partitioning was implemented for each of the datasets in the files previously mentioned using the train test split functionality in sci-kit learn. In addition, to ensure reproducibility the random seed was fixed as well, across all the 8 datasets.

To summarise, the reasons for these splits is not to get good results but to ensure a controlled experimental setting. The fact that good performance was also achieved is an added benefit, that just further emphasises the performance of the chosen deep learning procedure proposed.

3.4.4 Data Preprocessing extra considerations

This final data subsection deal with the considerations made to not do extra preprocessing steps. Firstly, the initial idea was to read the input as a rating matrix. However, the size of the rating matrix was problematic. During the extracting, transforming and loading of the dataset, the MovieLens dataset could not be loaded and stored. This was a problem as the machine learning models, had not even been applied yet, emphasising the need for a different approach. The approach chosen was to simply flatten the rating matrix and have 2 features (user-id, item-id) as input. Also, as the datasets used are state of the arts in Recommendation Systems, the cleaning process to deal with missing values after flattening the input did not need to occur.

Binary datasets	distribution of labels	
	class 1	class 2
Jester v1	42%	58%
Jester v2	34%	66%
Jester v3	8%	92%
MovieLens 20M	39%	61%

Table 3.3: Class imbalance: binary datasets

Furthermore, table 3.2 shows the class imbalance of ratings towards class 2. Here, class 2 refers to the ratings greater than the threshold set. This caused the author in the IPP [47] to think about making use of the ROCAUC metric which includes metrics that helps when comparing class-imbalanced datasets. However, as the author of [3] assumed continuous labels, the author here, decided to assume the labels (though binarised) as continuous.

Finally, deep learning procedures (ie Neural Networks) tend to learn better when the input features are close to zero in value. Initially, this inspired the thought of standardising the input features. However, the author intended the procedure to remain interpretable throughout. Standardisation, which subtracts the mean of the input features and divides the result by the standard deviation would cause the input to have id's that were not integers. The author decided against this but for future work, one could consider this and other normalisation techniques.

3.5 Deep Learning: 2 Single tasks Network

This section focuses on the 2 Single task deep learning procedure architecture. This architecture is comprised of two neural networks. The first neural network reconstructs the original inputs as its output. The second neural network uses the internal representation from the first neural network as input and then aims to make corresponding rating predictions a given user would make for an item.

3.5.1 Reconstruction network

The idea of reconstructing an input, without making use of labels is an unsupervised learning task as indicated in the Research section. Autoencoders solve this exact this of reconstruction[29].

Autoencoders are neural networks which in this problem description are given a 2-dimensional feature vector, $\mathbf{x} = \langle \text{user id, item id} \rangle$, as input. This neural network then aims to find a setting of values for the weights comprising the network architecture so that the output is approximately the input:

$$f(x) \approx x : f(x) \neq x \tag{3.1}$$

The autoencoder architectures in focus in this thesis are, namely; normal autoencoder, denoising autoencoder, regularised embedding autoencoder. Before describing these in detail, it is important to note that an autoencoder has an encoder and decoder segment. The encoder takes the input and processes the embedding, internal representation. The decoder takes this representation and aims to reconstruct the input. An example is seen in Figure 3.2. To clarify, the encoder is the weighted sum of the inputs and fed into an activation function, resulting in the embedding. Similarly, the decoder is a weighted sum of the embedding fed into some activation function producing the reconstruction.

The autoencoder, like any neural network, needs a specified network structure (number of layers, units per layer), choice of activation functions as well as the learning rule used to optimise the parameters.

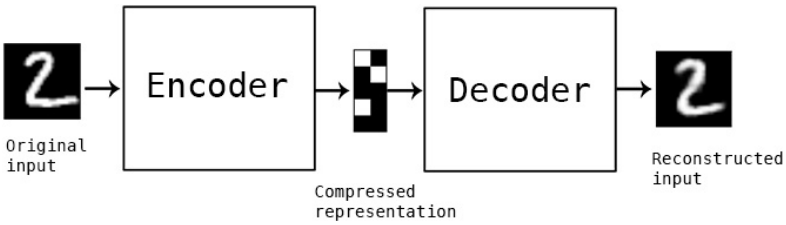

Figure 3.2: Input - Encoder - Embedding - Decoder - Reconstruction[18]

3.5.1.1 Network Structure

The network structure is a 2-1-2 structure. This means 2 units in the input layer, 1 unit (the embedding) in the hidden layer and 2 units in the output layer. The weights connecting consecutive layers have been initialised to the default keras' setting of Glorot Uniform with the biases initialised to zero. In addition, the random number generator has been fixed so that each network, when comparing, start from the same points.

3.5.1.2 Activation Functions

Information is fed to and processed in the hidden layers and output layer through linear or non-linear functions called activation functions. Activation functions perform a certain operation mathematically to its input. Certain input values can cause activation functions to "activate" or "deactivate". Activation results in a high valued output beyond some threshold whereas deactivation results in a low valued output.

Choosing an activation function is important, as it determines how effective learning is in the network. Saturated regions, regions where the gradient of the activation is almost zero, does not encourage learning. A zero-valued gradient causes the updates of weights dependent on such computational units to be set to zero, implying no change and therefore no update to weights.

Bearing this in mind, three activation functions are investigated, namely; selu, sigmoid and the identity function. Each experiment involves keeping one fixed activation function across the network to ensure effective comparison.

Scaled Exponential Linear Unit - SELU

SELU[19] has the following mathematical formulas for the forward propagation and its backpropagation stages respectively:

$$selu(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha(exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

$$\frac{\partial}{\partial x} selu(x) = \lambda \begin{cases} 1 & \text{if } x > 0 \\ \alpha exp(x) & \text{if } x \leq 0 \end{cases}$$

The SELU activation function encourages a smooth transition from 0 to $\lambda\alpha$ for negative or zero valued inputs. This smoother transition is caused by the exponential function present. This function is computationally expensive and can cause disadvantages in certain settings, like floating point errors causing blow ups in experiments as seen in "DNT" mentioned in the Results chapter. However, this activation does not have the problem of vanishing gradients as other activation functions like Relu, tend to do when the gradients are zero. A non-zero gradient gives Selu a chance to recover when learning and effectively update appropriate weights, in theory.

Sigmoid

The function and its derivative for the sigmoid activation function is the following:

$$sigmoid(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\frac{\partial}{\partial x} sigmoid(x) = \frac{\partial \sigma(x)}{\partial x} = \sigma(x) \cdot (1 - \sigma(x))$$

Sigmoid or logistic functions[27], unlike Selu, have saturation in positive regions which causes it to slow down which can be problematic. In addition, the lack of a proper thresholding alongside the presence of the computationally expensive, exponential function, creates a more computationally expensive function in theory. However, sigmoid can sometimes result in surprisingly good performances and as a result, has been introduced here as a possible activation function of choice for the deep learning procedure.

Identity

Finally, a simpler computationally, activation function has been added. This is a linear function and due to the performance of SVD, as well as the linear model pro-

posed by the author of [3], the activation function was included. The identity activation function, also known as the linear activation function has the following functions:

$$identity(x) = x$$
$$\frac{\partial}{\partial x} identity(x) = 1$$

This function serves as a fast but simple linear function. Unlike, the preceding non-linearities, this function is linear which poses a big assumption. The assumption posed is that the relationship between the input and the desired output has to be linear for this function to be effective. The experiments, carried out in the next chapter will decide whether that is the case.

3.5.1.3 Learning Rule

A traditional learning rule used in Neural Networks for learning and backpropagating the error to the weights is stochastic gradient descent (SGD). It works in the following way. For a given time update, t, SGD updates the weights in the direction of the error functions negative gradient ($g_i(t)$). The error function is simply the difference between the output of the neural network ie f(x) and the true label of the predictive task, y.

$$Error : E = f(x) - y \tag{3.2}$$

SGD uses a fixed learning rate to do the backpropagating of error. This means that the learning does not take into account the time taken to learn or the quality of learning. As a result, there is a need for an adaptive learning rule. This is a rule which decreases or increases, the rate of learning, depending on how close the weight setting is the "locally" optimal weight setting. As this task deals with root-mean-squared error as the error function, the author decided to use Keras' RMSProp learning rule.

RMSProp

Building on SGD, RMSProp, works by having an adaptive learning rate when updating each weight. The fixed learning rate from SGD, here, is normalised by taking the square **r**oot of the moving (adaptive) **m**ean of the **s**quared gradients. This is why it is called **RMS**Prop [28].

The moving average is controlled by A decay rate β, is a parameter setting which controls the moving average mentioned previously. The default setting in Keras for this decay rate is and remains for these experiments 0.9 [28]. Normally, ε is a small value added to take care of when $S_i(0) = 0$, however in Keras this is set to zero.

Algorithm 1 RMSProp

squared gradient: $g_i(t)^2$

Initially: $S_i(0) = 0$

$\beta = 0.9$

$\varepsilon = 0$

moving average: $S_i(t) = \beta S_i(t-1) + (1-\beta) g_i(t)^2$

square root (normalised) update: $\frac{-\eta g_i(t)}{\sqrt{S_i(t)+\varepsilon}}$

new weight setting after update: $w_i(t+1) = w_i(t) - \frac{\eta g_i(t)}{\sqrt{S_i(t)+\varepsilon}}$

The activation functions, network architecture with the appropriate learning rule has been implemented in the "(DatasetName)(Autoencoder Type).ipynb" notebooks, seen in the appendix.

3.5.2 Rating Predictions network

The rating predictions network is similar in architecture to the reconstruction network. The only difference is that the decoder reconstruction segment is replaced with weights going to one unit which computes the prediction of the ratings. The procedure for this segment is to first get the input to embedding segment learned from the reconstruction and then train for just one more epoch, now with the ratings as output instead. The same activation functions are used, alongside the learning rule and now the architecture is 2-1-1 instead of 2-1-2.

3.5.3 Autoencoder Types

Up to this point, the architecture described has been for a general normal autoencoder based structure. The author also investigated two further autoencoder types; the denoising autoencoder and a regularised embedding autoencoder. These different architectures are implemented as they are regularised versions of the normal autoencoder.

Regularisation is any modification one makes to a machine learning algorithm with the intention of reducing its generalisation error but not its training error. Analogously, regularisation can be thought of as imposing an internal competition between certain parameters with the goal of restricting their magnitude, thereby being less computationally expensive. One way this can be done is by controlling the magnitude of the parameters and output of a computational unit. Specifically, the single embedding computational unit in the hidden layer has had this procedure done to it using the L2 regularisation technique. This architecture is appropriately named regularised embedding autoencoder.

The other regularisation technique adopted here was adopting an implicit regularisation through making the inputs noisy by adding mean zero, variance one, Gaussian noise to the inputs. This technique encourages the network to be more robust as it needs to reconstruct the actual original input from a corrupted version. This technique is called a denoising autoencoder as the network aims to remove the noise present in the input to obtain its true form.

3.5.4 Implementations

The different autoencoder types are investigated and implemented taking into account the aspects discussed generally. The best architectures are the one which results in the lowest reconstruction and corresponding prediction losses, based on the root-mean-squared error loss function. This is described in detail in the Results chapter. The implementations of the normal, denoising and regularised embedding autoencoder follows. They have all been implemented with the identity activation function, in a similar vein to the final chosen deep learning procedure chosen in the Results chapter.

```
import numpy as np
from keras.layers import Input, Dense
from keras.models import Model
from keras import backend as K

# helper function loss function
def root_mean_squared_error(y_true, y_pred)
    return K.sqrt(K.mean(K.square(y_pred - y_true), axis=-1))

embedding_dim = 1 # dimension of embedding

```

```
12
13 # load data inputs
14 X_train = np.load(J2_Xtr.npy)
15 X_val = np.load(J2_Xval.npy)
16 X_test = np.load(J2_Xtest.npy)
17
18 # load binary labels
19 y_train_bin =  np.load(J2_ytrBin.npy)
20 y_val_bin = np.load(J2_yvalBin.npy)
21  y_test_bin = np.load(J2_ytestBin.npy)
22
23
24
25 # Fixing random seed to ensure reproducibility
26 from numpy.random import seed
27 seed(0)
28 from tensorflow import set_random_seed
29 set_random_seed(0)
30
31 x_dim = X_train.shape[1]
32 x_input = Input(shape=(x_dim,))
33
34 # Note: Linear or Identity activation functions used throughout
35
36 hidden_enc = Dense(2, activation='linear')(x_input)
37 embedding = Dense(embedding_dim, activation='linear')(hidden_enc)
38 hidden_dec = Dense(2, activation='linear')(embedding)
39
40 # reconstructed_input is the (decoded) reconstruction of the input
41 reconstructed_input = Dense(x_dim, activation='linear')(hidden_dec)
42
43 # definition of autoencoder
44 autoencoder = Model(x_input, reconstructed_input)
45 autoencoder.compile(optimizer='rmsprop', loss=
    root_mean_squared_error)
46 autoencoder.fit(
47     X_train, X_train,
48     epochs=3, # epoch based on lowest validation reconstruction
49     batch_size=32,
50     validation_data = (X_val, X_val)
51 )
52
```

```
53
54 # part 2: Making rating predictions
55 predictor = Dense(1, activation='linear')(embedding)
56 model = Model(x_input, predictor)
57 model.compile(optimizer='rmsprop', loss=root_mean_squared_error)
58 model.fit(
59     X_train, y_train_bin,
60     epochs=1,
61     batch_size=32,
62     validation_data = (X_val, y_val_bin) # how validation score is
       produced
63 )
64
65 # how testing score is produced
66 model.evaluate(X_test, y_test_bin)"
```

Listing 3.1: Normal Autoencoder Example on Jester v2 “binary” dataset, 2 Single Tasks Network. Chosen Deep Learning Procedure

```
... AS BEFORE (with Normal Autoencoder)

# load data inputs (Normal Autoencoder)
X_train = np.load(J2_Xtr.npy)
X_val = np.load(J2_Xval.npy)
X_test = np.load(J2_Xtest.npy)

# BECOMES

# load data inputs (Denoising Autoencoder)
X_train = np.load(J2_Xtr.npy)
X_val = np.load(J2_Xval.npy)
X_test = np.load(J2_Xtest.npy)

X_train_noisy = X_train + np.random.normal(loc=0.0, scale=1.0, size=
    X_train.shape)
X_val_noisy = X_val + np.random.normal(loc=0.0, scale=1.0, size=
    X_val.shape)
X_test_noisy = X_test + np.random.normal(loc=0.0, scale=1.0, size=
    X_test.shape)

.... AS BEFORE (with Normal Autoencoder but with the input X_...
    _noisy and output X_... for the reconstruction part)
```

Listing 3.2: Denoising Autoencoder Example on Jester v2 “binary” dataset, 2 Single Tasks Network

```
... AS BEFORE (with Normal Autoencoder)

hidden_enc = Dense(2, activation='linear')(x_input)
embedding = Dense(embedding_dim, activation='linear')(hidden_enc)
hidden_dec = Dense(2, activation='linear')(embedding)

# BECOMES

from keras import regularizers

hidden_enc = Dense(2, activation='linear')(x_input)
embedding = Dense(embedding_dim, activation='linear',
    activity_regularizer=regularizers.l2(10e-5))(hidden_enc)
hidden_dec = Dense(2, activation='linear')(embedding)

```

```
15 .... AS BEFORE (with Normal Autoencoder)
```

Listing 3.3: Regularised Embedding Autoencoder Example on Jester v2 “binary” dataset, 2 Single Tasks Network

3.6 Deep Learning: Multi-task Network

There is a sub-field of machine learning, where multiple learning tasks are solved at the same time. This is referred to as Multi-task learning (MTL) [8]. When learning new tasks, one often applies the knowledge obtained from learning related tasks. As a real-life example, a baby may first learn to recognize faces, and then use this insight to recognise different objects. MTL, from a machine learning perspective, is a form of such inductive transference [39].

Inductive transfer helps improve a model by introducing an inductive bias causing a model to prefer certain hypothesis over others. L1 regularisation is a technique that leads to sparse solutions and is an example of inductive bias towards sparsity. From a perspective of deep learning procedures, multi-task learning is typically done by soft or hard sharing parameters within the hidden layers. However, in this procedure, no sharing is done in the hidden layer as there is only 1 computational unit there. Instead, the output layers are shared between the reconstruction outputs and recommendation output.

In this procedure, we are given two learning tasks, T_1 and T_2. T_1 is the reconstruction or learning of partitioning task, that results in the embedding. T_2 is the rating predictions task to produce the recommendations. The notions, here, is that the two tasks have relationships between them but not necessarily identical. Multi-task learning, in theory, helps improve the learning of task, T_i by using knowledge from both tasks.

3.6.1 Implementations

Regarding the network architecture, here the number of units in the input, hidden and output layer are 2, 1 and 3. The 3 in the output layer have two splits; 2 for the reconstruction task and 1 for the recommendation predictions task. The input to the network, as with the 2 single tasks networks is the $\langle$user id, item id$\rangle$pair. As with the single task network, there is one computational unit for the embedding in the hidden layer. In addition, as with the single-task version; similar activation functions fixed across all layers, learning rule and autoencoder types have been investigated. Regarding the loss function, when training, a loss function needs to be defined. As with the single=task models the loss function is the root mean square error. Here, we use the

same loss function for each task but optimized them jointly. The implementation of the Multi-task procedure can be seen below:

```
import numpy as np
from keras.layers import Input, Dense
from keras.models import Model
from keras import backend as K

# helper function loss function
def root_mean_squared_error(y_true, y_pred)
    return K.sqrt(K.mean(K.square(y_pred - y_true), axis=-1))

embedding_dim = 1 # dimension of embedding


# load data inputs
X_train = np.load(J2_Xtr.npy)
X_val = np.load(J2_Xval.npy)
X_test = np.load(J2_Xtest.npy)

# load binary labels
y_train_bin =  np.load(J2_ytrBin.npy)
y_val_bin = np.load(J2_yvalBin.npy)
 y_test_bin = np.load(J2_ytestBin.npy)


# Fixing random seed to ensure reproducibility
from numpy.random import seed
seed(0)
from tensorflow import set_random_seed
set_random_seed(0)

x_dim = X_train.shape[1]
x_input = Input(shape=(x_dim,))

# Note: Linear or Identity activation functions used throughout

hidden_enc = Dense(2, activation='linear')(x_input)
embedding = Dense(embedding_dim, activation='linear')(hidden_enc)
hidden_dec = Dense(2, activation='linear')(embedding)

# reconstructed_input is the (decoded) reconstruction of the input
reconstructed_input = Dense(x_dim, activation='linear')(hidden_dec)
```

```
41
42
43 # definition of Multi-task model
44 joint_model = Model(inputs=x_input, outputs=[reconstructed_input,
       predictor]) # multi-task output
45 joint_model.compile(optimizer='rmsprop', loss=
       root_mean_squared_error)
46 joint_model.fit(
47     x = [X_train],
48     y = [X_train, y_train_bin], # multi task output
49     epochs = 1,
50     batch_size = 32,
51     validation_data = ([X_val], [X_val, y_val_bin]) # how validation
       score is produced (multi-task output)
52 )
```

Listing 3.4: Normal Autoencoder Example on Jester v2 “binary” dataset, Multi-task Network

Finally, as with the 2 single tasks network, the denoising autoencoder and regularised embedding autoencoder based procedures are obtained by making the slight changes described previously.

```
1 ... AS BEFORE (with Normal Autoencoder)
2
3 # load data inputs (Normal Autoencoder)
4 X_train = np.load(J2_Xtr.npy)
5 X_val = np.load(J2_Xval.npy)
6 X_test = np.load(J2_Xtest.npy)
7
8 # BECOMES
9
10 # load data inputs (Denoising Autoencoder)
11 X_train = np.load(J2_Xtr.npy)
12 X_val = np.load(J2_Xval.npy)
13 X_test = np.load(J2_Xtest.npy)
14
15 X_train_noisy = X_train + np.random.normal(loc=0.0, scale=1.0, size=
       X_train.shape)
16 X_val_noisy = X_val + np.random.normal(loc=0.0, scale=1.0, size=
       X_val.shape)
```

```
X_test_noisy = X_test + np.random.normal(loc=0.0, scale=1.0, size=
    X_test.shape)

.... AS BEFORE (with Normal Autoencoder but with the input X_...
    _noisy and output X_... for the reconstruction part)
```

Listing 3.5: Denoising Autoencoder Example on Jester v2 “binary” dataset, Multi-Task Network

```
... AS BEFORE (with Normal Autoencoder)

hidden_enc = Dense(2, activation='linear')(x_input)
embedding = Dense(embedding_dim, activation='linear')(hidden_enc)
hidden_dec = Dense(2, activation='linear')(embedding)

# BECOMES

from keras import regularizers

hidden_enc = Dense(2, activation='linear')(x_input)
embedding = Dense(embedding_dim, activation='linear',
    activity_regularizer=regularizers.l1(10e-5))(hidden_enc)
hidden_dec = Dense(2, activation='linear')(embedding)

.... AS BEFORE (with Normal Autoencoder)
```

Listing 3.6: Regularised Embedding Autoencoder Example on Jester v2 “binary” dataset, Multi-task Network

3.7 Non-deep learning: SVD - state of the art

This model investigated is the state of the art, Singular Value Decomposition, which was a part of the best solution when solving the Netflix Prize [17]. SVD solves the matrix factorization problem. This problem asks how to model each user and item as a vector of factors, that have been inferred from the dataset.

The problem described here is the following[46]. Let q_i be the vector for item i and w_u the vector for user u. The predicted rating, $p_{u,i}$ is the dot product between q_i and w_u. These vectors are then learned from the training data in order to approximate the ratings such that $R = WQ$. W, becomes an n by f matrix of users with their latent (inferred) factors and Q is an f by m matrix of items and their latent factors. This means that the Matrix Factorization problem is factor R into W x Q . This problem is solved by the aforementioned, SVD.

The (truncated) SVD[29] finds a low-dimensional vector representing both the rows and columns of a matrix, all at once. Truncated SVD is referred to as the best known low-rank approximation of a matrix[29], which has been measured by squared error, as is with the thesis problem specification. SVD works by decomposing a given matrix, M in such a way that M, an m by n matrix becomes a product of three matrices: $M = U\ \Sigma\ V^T$. U becomes an m by m matrix of orthogonal columns, also called left singular vectors. Σ becomes a rectangular, m by n diagonal matrix, where the entries on the diagonal are called singular values. Finally V^T becomes an n by n matrix of orthogonal rows, called right singular vectors. Two vectors are said to be orthogonal if the whose dot product between them is zero, implying the two vectors are perpendicular.

Singular value decomposition has a closed format solution for obtaining these three matrices. The singular values of Σ are the square roots of the eigenvalues of MM^T. The left and right singular vectors are the eigenvectors of MM^T and M^TM respectively. As a brief comment, an eigenvector, v with eigenvalues λ of a matrix, A, is obtained from the following linear equation: $Av = \lambda\ v$.

The truncated SVD representation obtains approximate representations, by having a criteria k, such that k latent vectors or factors are used to describe the vectors rather than the larger α where $\alpha = minm,n$. Practically, vanilla SVD does not work when a matrix has missing values as the rating matrix does here, so a preprocessing step

is done to fill up the matrix by taking the average of all the values that are observed. Once there are no longer missing values, W becomes equal to $U_k \, \Sigma_k^{0.5}$ and Q becomes equal to $\Sigma_k^{0.5} \, V_k$. Here, V_k and U_k are the k singular vectors corresponding to the k largest singular values.

This procedure has been implemented using sci-kit surprise's[45] functionality. The implementation can be seen in the data files labelled "(Dataset name) SVD and Random and Regression Tree.ipynb".

3.8 Non-deep learning: Baseline Models

Following the use of the state of the art model mentioned previously, the author decided to investigate simple baseline models to effectively compare the performance of the deep learning procedure. Theoretically, these models should serve as good lower bounds for optimal performance, in the experiments carried out.

3.8.1 Simple Weighted average

The model used here is a regressor that makes predictions for the recommendation ratings using a simple strategy. Normally, there are different strategies one could use such as taking the median of the training dataset and always predicting the new ratings as such. However, the author decided to make use of the strategy that always simply predicts the mean. Practically, this works by taking the average of all the ratings in the training dataset, as done in the preprocessing step of SVD, and $p_{u,i}$ as this average value for all user, item pairs in the validation or testing dataset. In order to implement this model, the sci-kit learn dummy regressor[44] was used with its strategy parameter set to “mean”. The implementation of this model can be seen in the “(Dataset name) SVD and Random and Regression Tree.ipynb” files for the different experiments carried out.

3.8.2 Regression Tree

The final model investigated here is the Regression Tree model. As described in the Introduction section, this is a (piece-wise) linear model, making a strong linear assumption between the input identifiers and the recommendation outputs.

A decision tree builds regression models, hence the regression tree, into a tree form with decision nodes and leaf nodes present[41]. This is done by partitioning into linear segments the training data based on such decision nodes. A decision node has 1 or more branch, with each branch representing values from the features being tested. The leaf nodes, represent the possible recommendations as a continuous-valued output. Decision trees split in such a way that the higher up a decision node is in a tree, the better the feature is in predicting the ratings at the end. In a regression problem, the idea of a better feature is based on a splitting criterion called Standard Deviation

Reduction[41]. The standard deviation reduction procedure selects features, in each split, that results in the highest standard deviation reduction, when split on certain feature values. If the standard deviation is greater than zero then more splitting is required, if it is equal to zero, then it becomes a leaf node.

To construct a (decision) regression tree, like the one in sci-kit learn [44], the ID3 algorithm by J.R. Quinlan is used. This algorithm starts from the root (top of the tree) and proceeds to branch out downwards by choosing attributes with the highest standard deviation reduction, in each iteration. The implementation of the regression tree in this thesis has been done using the default setting from sci-kit learns, decision tree regressor library from the Tree submodule. Finally, as with the other non-deep learning procedures, the implementation can be seen in the data files labelled “(Dataset name) SVD and Random and Regression Tree.ipynb”.

Chapter 4

Results and Analysis

The purpose of this section is to first present the results of the over 150 experiments performed in this project and to discuss their significance. More specifically, the results will be used to compare the effectiveness of the different configurations of the auto-encoder inspired models. The main focus if these comparisons will be the different types of auto-encoders investigated, their various activation functions and the way the embedding was learned (multi-task manner versus separate single tasks). To aid comparison, it is useful to have similar hyperparameter settings. The ”(default)” refers to the default setting in Keras' implementation. The fixed settings for the deep learning procedures are:

- the random number generator seed
- maximum number of epochs
    1. 5 epochs for the multi-task framework
    2. 5 & 1 epochs for the 2 single-tasks framework
- each batch of user-item preferences has 32 sample
- (default) learning rate of 0.001
- learning rule: RMSProp
- (default) weight initialisation: Glorot Uniform
- bias initialised to zero
- same state of the train, validate and test dataset used at the start of each experiment

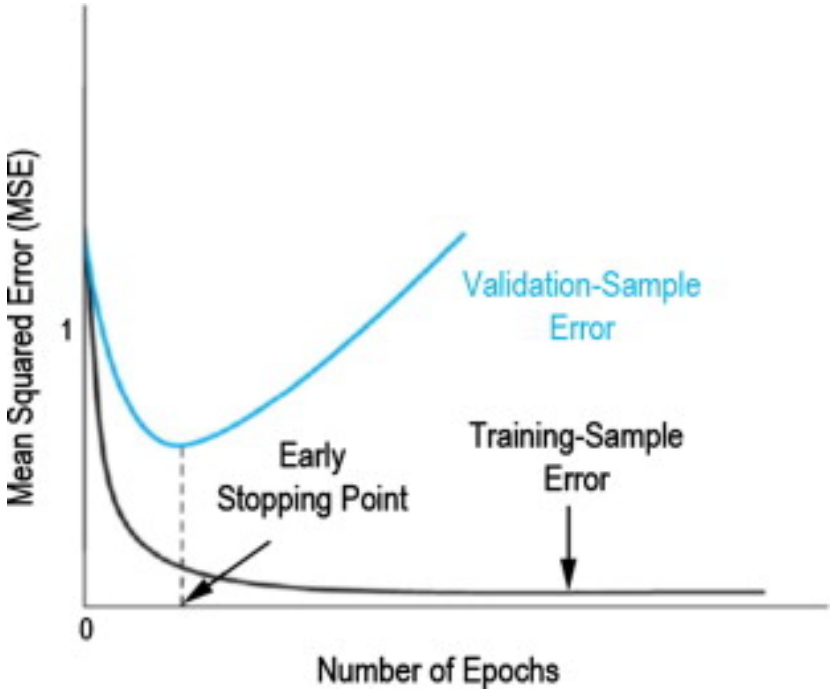

Figure 4.1: Early Stopping Point[1]

The metric used for evaluating the results is the root mean square error (RMSE). This is defined as the following, where N is the number of possible rating entries in the validation or testing dataset. For each validation/testing (user,item) pair, the user rating for a given item is predicted, $p_{u,i}$. The error is measured as the difference between the prediction and the actual rating, $r_{u,i}$ on the validation/testing dataset.

$$RMSE = \sqrt{\frac{\sum_{u,i}(p_{u,i} - r_{u,i})^2}{N}}$$

This metric effectively evaluates the results of the two main phases of the Deep Learning procedures proposed in this paper; how good the autoencoder reconstruction phase is and how good the network's recommendation predictions are. After, comparing models based on the validation performance, the model that performs comparatively better is applied to the testing dataset. This will give a generalisation performance of the model.

The number of epochs mentioned in the following tables is the result of comparing how many epochs, it takes to reach minimum validation error in the reconstruction phase. This can be seen in the early stopping point in Figure 4.1. An epoch is when a model has completely passed through all the data. The point indicated in that fig-

ure is where the model has neither over or under-fitted. In deep learning procedure, over-fitting is when the neural network fits too closely to the training dataset but does not generalise well to unseen datasets (validation or testing in this procedure). Conversely, under-fitting is when the neural network does not learn enough to do well on the training dataset itself. This balance of ensuring a model does not over or under-fit emphasises the need for the unseen datasets.

Therefore, this epoch number has been obtained because the quality of the embedding learned is of huge importance in this procedure, as well as the ratings produced. The RMSE reported is the corresponding RMSE on the validation or test datasets for the recommendation predictions. The model which has the lowest RMSE in a set of experiments is the better model.

4.1 Comparing validation performance of models

Tables 4.1 to 4.8, show the performance of the deep and non-deep learning procedures, evaluated on the 8 datasets. As alluded to in the previous chapter, these models are evaluated on the validation datasets, in order to specifically, make an evaluated choice on the final deep learning procedure.

On each of the 8 datasets, 18 neural network variations are trained and evaluated. Equally, the 3 non-deep learning procedures are also trained and evaluated. This results in 21 models trained and evaluated per dataset. It is useful to note that, during the experiments, there were some special cases were the reconstruction validation performance was not great, yet the predictions were. As the main focus was the embedding, this was not considered as emphasised in the intuition behind the epoch numbers. Finally, the results table highlights important results using the following colours:

- Green - symbolises the 3 (best) models that achieved the lowest validation root mean square error
- Red - symbolises models that did not finish training due to floating point errors in training resulting in nan values

Also, the models are labelled DNT, standing for “did not train”.

4.1.1 Jester results

model type				epoch	corresponding validation RMSE
Deep Learning models	Normal Auto-encoder	Multi-task network	selu activation	1	0.4339
			sigmoid activation	5	DNT
			identity activation	4	9.6692
		Two single tasks network	selu activation	3	0.4166
			sigmoid activation	1	DNT
			identity activation	3	0.4178
	Denoising Autoencoder	Multi-task network	selu activation	5	DNT
			sigmoid activation	5	DNT
			identity activation	4	9.4839
		Two single tasks network	selu activation	3	0.4166
			sigmoid activation	1	DNT
			identity activation	3	DNT
	Regularised embedding Autoencoder	Multi-task network	selu activation	1	0.4330
			sigmoid activation	5	DNT
			identity activation	5	1.7098
		Two single tasks network	selu activation	4	DNT
			sigmoid activation	1	DNT
			identity activation	5	DNT
Other models	Singular Value Decomposition			N/A	0.4354
	Dummy regressor: weighted average			N/A	0.4927
	Regression Tree			N/A	0.6660

Table 4.1: Jester v1 “binary” experiments

Table 4.1 shows the validation performance of the 21 models on the Jester v1 “binary” dataset. From these results, it is clear that the regularised embedding autoencoder performed sub-optimally. Generally, it did not train, an obvious reasoning for this is the regularisation imposed. This is possibly due to not enough input samples being processed due to the semi-aggressive thresholding. Surprisingly, SVD did not come in the top 5 models, coming 6^{th}. The best models, tied, were the two

single-task, selu activations for the normal and denoising autoencoder structure. Another point to note is that the regression tree model performed worse than the simple weighted average baseline which highlights it's limitation. This limitation is that the regression tree model assumes a linear relationship between the inputs and recommendation predictions which is too strong. However, this is just the first dataset, so it is a bit early to make conclusions. The results are promising though for the deep learning procedures.

model type				epoch	corresponding validation RMSE
Deep Learning models	Normal Auto-encoder	Multi-task network	selu activation	5	4.4204
			sigmoid activation	1	4.4231
			identity activation	3	13.7066
		Two single tasks network	selu activation	2	4.4148
			sigmoid activation	1	4.4231
			identity activation	3	4.3978
	Denoising Autoencoder	Multi-task network	selu activation	5	4.4223
			sigmoid activation	1	4.4231
			identity activation	4	9.4948
		Two single tasks network	selu activation	4	4.4148
			sigmoid activation	1	4.4231
			identity activation	3	4.3986
	Regularised embedding Autoencoder	Multi-task network	selu activation	4	4.4579
			sigmoid activation	1	4.4231
			identity activation	4	4.8275
		Two single tasks network	selu activation	4	4.4151
			sigmoid activation	1	4.4231
			identity activation	5	4.4434
Other models	Singular Value Decomposition			N/A	4.3584
	Dummy regressor: weighted average			N/A	5.2991
	Regression Tree			N/A	7.0728

Table 4.2: Jester v1 "continuous" experiments

The test results for the continuous and binary representations of the Jester v1 dataset appears near identical. For both sets of representations, the best deep learning models were the normal autoencoder and the de-noising autoencoder, specifically when employing the two-single tasks network. However, the activation functions were different, here, the optimal one was the identity activation function with the selu activations falling just short, tied at 4^{th}. Furthermore, being the state of the art, the SVD model performs as expected on this dataset. As mentioned previously, the regression tree model is still performing worse than the simple baseline.

model type				epoch	corresponding validation RMSE
Deep Learning models	Normal Auto-encoder	Multi-task network	selu activation	5	DNT
			sigmoid activation	5	DNT
			identity activation	3	23.1721
		Two single tasks network	selu activation	5	0.3393
			sigmoid activation	5	DNT
			identity activation	3	DNT
	Denoising Autoencoder	Multi-task network	selu activation	5	DNT
			sigmoid activation	5	DNT
			identity activation	5	21.0861
		Two single tasks network	selu activation	5	DNT
			sigmoid activation	5	DNT
			identity activation	3	0.3456
	Regularised embedding Autoencoder	Multi-task network	selu activation	3	DNT
			sigmoid activation	5	DNT
			identity activation	1	1.5825
		Two single tasks network	selu activation	5	DNT
			sigmoid activation	5	DNT
			identity activation	1	1.8507
Other models	Singular Value Decomposition			N/A	0.4148
	Dummy regressor: weighted average			N/A	0.4733
	Regression Tree			N/A	0.6328

Table 4.3: Jester v2 “binary” experiments

The top three performing models in Jester v2 “binary” seem to be a combination of ideas from the previous 2 datasets. As before, the three best models were the same autoencoders, with SVD now coming in third. In addition, the activation functions are now selu for the normal autoencoder and the identity for the denoising autoencoder. The best performing architecture was the normal autoencoder with selu activation in a two single task network architecture. From the results so far, it appears that the multi-task architecture was a theoretical advantage but not practically, at least when using 5 epochs as a constraint. The obvious thing would be to say that more epochs would improve it, however, the did not train means that it will not train with more epochs. Similarly, the regularised embedding architecture is also not performing optimally as too the aforementioned Regression Tree model.

Table 4.4’s experiments follow a similar vein to the previous experiments and will not be repeated. Instead, this segment serves as a summary for the best performing models seen so far. The best two architectures so far are the Normal and Denoising autoencoders. Regarding the type of neural network, the optimal choice seen is the two single-task network structure. The choice of activation function is the interesting part. Regarding the denoising structure, it is favouring the identity function over selu by three-to-one. For the normal autoencoder, it is currently a tie between selu and the identity activation function.

Table 4.5 shows the validation performance on the “binary” version 3 Jester dataset. The first thing of note is that the deep learning models did not train generally. However, when they did, there were mixed results. The normal autoencoder did not train generally in this task, surprisingly. The denoising autoencoder, selu, non-multi-task architecture did train and performed extremely well. This architecture was the best performing model across all architectures on this dataset. Similarly, the SVD model performs well as has been the common theme across the datasets so far. The inverse is also similarly seen in the regression tree model performance.

Regarding the final Jester validation experiment in Table 4.6, the SVD model was the worst performing of the 21 model. This shows that event the state-of-the-art procedures have limitations. A possible reason might be the space of not just possible users and items, but the domain of the ratings. Jester v3 is the biggest of all Jester datasets

model type				epoch	corresponding validation RMSE
Deep Learning models	Normal Auto-encoder	Multi-task network	selu activation	5	4.3544
			sigmoid activation	5	DNT
			identity activation	5	8.7916
		Two single tasks network	selu activation	5	4.3281
			sigmoid activation	5	DNT
			identity activation	3	4.2729
	Denoising Autoencoder	Multi-task network	selu activation	5	4.3647
			sigmoid activation	5	DNT
			identity activation	5	10.2424
		Two single tasks network	selu activation	5	4.3281
			sigmoid activation	5	DNT
			identity activation	3	4.2783
	Regularised embedding Autoencoder	Multi-task network	selu activation	5	4.3493
			sigmoid activation	5	DNT
			identity activation	1	4.3287
		Two single tasks network	selu activation	5	4.3300
			sigmoid activation	5	DNT
			identity activation	3	10.7076
Other models	Singular Value Decomposition			N/A	4.4103
	Dummy regressor: weighted average			N/A	5.3130
	Regression Tree			N/A	0.6328

Table 4.4: Jester v2 “continuous” experiments

model type				epoch	corresponding validation RMSE
Deep Learning models	Normal Auto-encoder	Multi-task network	selu activation	5	DNT
			sigmoid activation	5	DNT
			identity activation	1	8.1636
		Two single tasks network	selu activation	4	DNT
			sigmoid activation	5	DNT
			identity activation	2	DNT
	Denoising Autoencoder	Multi-task network	selu activation	5	DNT
			sigmoid activation	5	DNT
			identity activation	1	7.1744
		Two single tasks network	selu activation	5	0.0825
			sigmoid activation	5	DNT
			identity activation	2	DNT
	Regularised embedding Autoencoder	Multi-task network	selu activation	5	DNT
			sigmoid activation	5	DNT
			identity activation	1	0.1555
		Two single tasks network	selu activation	5	DNT
			sigmoid activation	5	DNT
			identity activation	5	DNT
Other models	Singular Value Decomposition			N/A	0.2299
	Dummy regressor: weighted average			N/A	0.2736
	Regression Tree			N/A	0.3350

Table 4.5: Jester v3 “binary” experiments

model type				epoch	corresponding validation RMSE
Deep Learning models	Normal Auto-encoder	Multi-task network	selu activation	5	DNT
			sigmoid activation	5	DNT
			identity activation	5	DNT
		Two single tasks network	selu activation	4	DNT
			sigmoid activation	5	DNT
			identity activation	2	26.5623
	Denoising Autoencoder	Multi-task network	selu activation	5	DNT
			sigmoid activation	5	DNT
			identity activation	1	38.6460
		Two single tasks network	selu activation	5	DNT
			sigmoid activation	5	DNT
			identity activation	2	DNT
	Regularised embedding Autoencoder	Multi-task network	selu activation	5	DNT
			sigmoid activation	5	DNT
			identity activation	1	27.6022
		Two single tasks network	selu activation	5	DNT
			sigmoid activation	5	DNT
			identity activation	5	DNT
Other models	Singular Value Decomposition			N/A	77.5467
	Dummy regressor: weighted average			N/A	41.8713
	Regression Tree			N/A	47.8688

Table 4.6: Jester v3 “continuous” experiments

investigated. In addition, the rating scale for the “continuous” Jester v3 is much bigger than the “binary” version. The binary version has values mainly between 0 and 1 whereas the continuous version is between -10 and 10. In addition, the best performing model in these experiments is the normal autoencoder with two single task networks and identity activation functions. The spatial input disadvantage proposed for the SVD model might serve as an explanation for why the regularised embedding autoencoder architecture performs well. The imposing of competition amongst the weights as a result of the regularisation has prevented the overfitting that possibly occurred due to the large input space. This architecture finished as the second-best architecture in this experiment. Finally, at this point, the author would choose the identity activation for the normal autoencoder architecture that did not use multi-task learning. The reason for this choice is that this architecture most frequently came out top whenever able to train, apart from on one occasion coming out second.

4.1.2 MovieLens 20M results

In Table 4.7, apart from the author’s proposal as the best deep learning architecture mentioned in the previous paragraph, the deep learning procedures did not perform exceptionally. This can be seen through most of the deep learning procedures not training and also for some that did train performing worse than the regression tree model. Here the best procedure was the SVD model followed by the dummy regressor. The fact that the deep learning procedure performed a bit worse (only in this scenario) than the baseline is not a problem to the author. This is because the dummy regressor just happened to have a really good performance in this scenario so this model in this experiment should not really be thought of as a baseline.

In the MovieLens 20M “continuous” experiments, the effect of input space was not a problem unlike alluded to with Jester v3 “continuous”. This can be seen especially in SVD’s better performance. Here, SVD performed slightly better than the best deep learning procedure. However, the deep learning procedure is still outstanding. This is seen as the normal autoencoder with two single task networks and identity activations came out joint second.

model type				epoch	corresponding validation RMSE
Deep Learning models	Normal Auto-encoder	Multi-task network	selu activation	1	0.5579
			sigmoid activation	5	DNT
			identity activation	2	6.4070
		Two single tasks network	selu activation	1	DNT
			sigmoid activation	5	DNT
			identity activation	4	0.4920
	Denoising Autoencoder	Multi-task network	selu activation	1	0.5580
			sigmoid activation	5	DNT
			identity activation	5	14.8560
		Two single tasks network	selu activation	5	DNT
			sigmoid activation	5	DNT
			identity activation	3	1.0354
	Regularised embedding Autoencoder	Multi-task network	selu activation	1	0.5583
			sigmoid activation	5	DNT
			identity activation	1	7.5978
		Two single tasks network	selu activation	4	DNT
			sigmoid activation	5	DNT
			identity activation	3	DNT
Other models	Singular Value Decomposition			N/A	0.4174
	Dummy regressor: weighted average			N/A	0.4878
	Regression Tree			N/A	0.6307

Table 4.7: MovieLens 20M “binary” experiments

model type				epoch	corresponding validation RMSE
Deep Learning models	Normal Auto-encoder	Multi-task network	selu activation	1	DNT
			sigmoid activation	1	7.0515
			identity activation	3	16.5202
		Two single tasks network	selu activation	4	1.6819
			sigmoid activation	1	7.0515
			identity activation	4	1.6819
	Denoising Autoencoder	Multi-task network	selu activation	1	1.6820
			sigmoid activation	5	7.0515
			identity activation	1	17.8057
		Two single tasks network	selu activation	5	1.6819
			sigmoid activation	5	7.0515
			identity activation	4	1.6995
	Regularised embedding Autoencoder	Multi-task network	selu activation	2	1.6821
			sigmoid activation	2	7.0515
			identity activation	5	DNT
		Two single tasks network	selu activation	5	DNT
			sigmoid activation	1	7.0515
			identity activation	3	DNT
Other models	Singular Value Decomposition			N/A	1.6751
	Dummy regressor: weighted average			N/A	2.1042
	Regression Tree			N/A	2.6582

Table 4.8: MovieLens 20M “continuous” experiments

4.1.3 Chosen Deep Learning procedure

As seen through the experiments carried out, the best procedure (deep or not) is a deep learning procedure. This proposed deep learning procedure, shown as Listing 3.1 in the Design and Implementation section, had the following architecture:

- Normal autoencoder
- Two single task networks (first for reconstruction, second for rating prediction)
- Identity activation functions across the network

4.2 Test dataset: chosen procedure against state of the art

The performance of the chosen deep learning procedure is promising. This section focuses on producing a generalisation score of the model across the different datasets investigated. As an added level of comparison, the generalisation score of the state-of-the-art SVD is added too. The colour scheme here is simpler:

- Green - symbolises the (best) models that achieved the lowest testing root mean square error
- Red - symbolises models that finished second, in the situations where both models were able to fully train

From these results, there are points to be noticed. Whenever the chosen procedure trained it performed equally as well as SVD, tying overall, 3 each out of 6 occasions. However, the 2 times it did not train is a limitation. Assuming the model did not perform better in those instances then the procedure would be slightly worse than the state-of-the-art. Regardless though, the experiments show that the chosen deep learning procedure performs at least as well as the singular value decomposition model. Finally, if just focusing on the times when it did train, it tends to outperform the SVD model. This creates a point of focus moving on that, whenever implementing the procedure, as long as it is evaluated as mentioned, training is the important thing that needs to occur. Once trained, one can be confident of its performance when given just identifiers as input as this thesis shows.

dataset	model	epoch	corresponding test RMSE
Jester v1	chosen deep learning procedure	3	0.4177
	Singular Value Decomposition	N/A	0.4356
Jester v2	chosen deep learning procedure	3	DNT
	Singular Value Decomposition	N/A	0.4152
Jester v3	chosen deep learning procedure	2	DNT
	Singular Value Decomposition	N/A	0.2299
MovieLens 20M	chosen deep learning procedure	4	0.4924
	Singular Value Decomposition	N/A	0.4176

Table 4.9: “Binary” dataset held-out test performance

dataset	model	epoch	corresponding test RMSE
Jester v1	chosen deep learning procedure	3	4.3919
	Singular Value Decomposition	N/A	4.3542
Jester v2	chosen deep learning procedure	3	4.2631
	Singular Value Decomposition	N/A	4.4075
Jester v3	chosen deep learning procedure	2	26.5922
	Singular Value Decomposition	N/A	77.5337
MovieLens 20M	chosen deep learning procedure	4	1.6820
	Singular Value Decomposition	N/A	1.6763

Table 4.10: ”Continuous” dataset held-out test performance

Chapter 5

Conclusion

This piece of work set out to explore the possibilities of using a deep learning model in a way that would reduce computational expense, but retain (or improve upon) the prediction prowess of existing Recommender Systems. The project undertaken in this paper proposes the use of a model to accurately predict labels using only two inputs: an identifier for a subject (user) and an identifier for an item pertaining to that subject. To accomplish this a lot of research went into the different types of models, model architectures and activation functions that could be configured to work together and make the best predictions possible. For an effective investigation of this proposal, it was important to choose an appropriate dataset(s) for this project, and this was achieved using the MovieLens dataset and the Jester dataset. Finally, to compare and validate the results of the proposed model on the proposed datasets the Root-Mean-Squared-Error metric was used.

The final deep learning procedure developed was a 2 single tasks network, with the reconstruction task a 2-1-2 autoencoder structure to produce an embedding. This embedding implies a quality partitioning of the user-item input space. After the embedding was learned, the decoder segment of the autoencoder was replaced with a connection to one node, forming a rating prediction network with a 2-1-1 architecture. Throughout the architecture, there was a fixed activation function used, the identity activation. This procedure was chosen following 150+ validation experiments as well as the final test results which showed promising results in comparison to the state of the art, SVD model. The author proposes as possible future work further tweaking some of the fixed parameters mentioned at the Results section, alongside further researching an effective loss function that can directly or indirectly learn a partitioning

of the input space. This should be such that that Coarse Preference elicitation mentioned and alluded to in the Introduction and Research section is utilised. The other supervisees approached this problem from that perspective so a possible merge of ideas could yield more fruitful results.

5.1 Project management

The consistent interactions with Pavlos Andreadis and fellow supervisees ensured that the objectives were met and the corresponding tasks completed. An original timetable for the project was drawn out as a Gantt chart seen in the Project Proposal [47]. As the project developed, some issues began to arise.

One example was the lack of consistency across the three versions of the Jester dataset prevented a merging of all 3 Jester datasets. This problem was solved by considering each version as an individual dataset, as mentioned in the Design and Implementation chapter.

Another problem was that many deep learning procedures than initially anticipated did not finish their training. A possible resolution in future works could be more intricately tweaking some of the fixed parameters, mentioned in the Results and Analysis section. In addition, different activation functions could be investigated as well. Given more time, the author would have revised even more iterations of the project with the supervisor to further improve the final deep learning procedure.

Finally, the allocation of suitable contingency time, as seen in the Gantt chart, allowed for most problems to be solved without negatively affecting the progress of the project. In addition to this, the use of a version control system, more specifically Github enabled any interested parties to have full access to the development of the procedures and experiments carried out (See Appendix).

5.2 Author's Assessment of the Thesis

5.2.1 Thesis' technical contribution

This project involved implementing a pipeline for extracting a minimal amount of data, from a wide range of datasets, then transforming that data (ie. to remove noise

and any other inconsistencies) and ultimately generating an accurate prediction based on a trained model.

To achieve the resulting product of this project, a lot of research went into understanding and devising different techniques that would help the proposed Recommender System outperform its competition. Most importantly, it involved examining the different components that constitute an embedding; looking at how others have implemented their own embedding, and improving upon those designs.

In this paper, one of the main goals was implementing a simple input-output structure around a good embedding. This was made possible by carefully pairing a deep learning model (that assumed a non-linear relationship) with an activation function that applied the right amount of restriction on the input. To summarise, this project

is a merge of supervised and unsupervised machine learning techniques which also delivers a research-led deep learning procedure at the end.

5.2.2 Relevance and importance of thesis

Firstly, this project formalizes a and presents a new deep learning procedure for partitioning the user and item-space in a Recommender System given only unique identifiers. It also implements the procedure, runs experiments, compares effectively against well-performing methods. This context is given as it exactly refers to the completion criteria outlined by the supervisor who proposed the project. This criterion was graded on its complexity and the highest complexity tasks were achieved.

This project was proposed to the author who is on the Machine Learning pathway of the Artificial Intelligence program. As detailed, in the technical contribution section, all aspects are at the core of a machine learning project. These aspects are; designing and deploying a (deep) machine learning model applied to the field of Recommender Systems. This field is historically a technical challenge, emphasizing the importance. In addition, the research aspect that has occurred throughout this project enhances the skill that is valuable in academia and industry as a machine learning practitioner. Also, the data processing, data science pipeline, exploration, and analysis are important to machine learning processes.

In addition, given how recommender systems are ubiquitous and the fact that the chosen deep learning procedure performs as well as the state of the art, SVD model. Researchers have been playing around with different machine learning techniques that aim to outperform an SVD procedure, which has been around for a while [17]. This project builds upon various researchers' work and further progresses this line of research. This procedure simplifies the recommender system approach which normally takes into account other contextual information about users and items by using just unique identifiers. This reduces the computational expense of the task.

Finally, there is the organizational element of planning the project well enough and executing the plan at hand. As seen, through the project management section, this was done well despite difficult challenges.

5.2.3 Thesis' usefulness to subject field

This project involves a description on a recommender system procedure which academics can use. This project can appeal to researchers in the machine learning sector, data science sector, and generally the Artificial Intelligence sector. They can use this project to critique and further develop ideas which could possibly improve current research in the field.

In addition, the code and reasoning behind development are detailed in the Design and Implementation section. Also, for machine learning developers or engineers, the code has been well-described and attached to the Appendix section.

Finally, the procedure that was developed and tested in this project is inspired by a state-of-the research proposed by the author of [3]. In addition, the performance of the procedure is as good as Singular Value Decomposition, a state of the art technique for matrix factorisation.

Despite the great results obtained during the project, there is still some room for improvement. For example, the use of anonymous identifiers could be merged with contextual information regarding users and items. This context could (not necessarily should) help the machine learning procedure suggested. The fellow supervisees approached this problem by taking into account context. Therefore, given further time,

understanding, discussing and merging the ideas learned should account for a procedure with at least similar performance and hopefully better. Equally, making use of non-continuous assumptions of the output label as well as evaluating the network based on a metric like average precision which helps in class-imbalanced data, serve as further work that could possibly improve the deep learning procedure.

In conclusion, this project was an opportunity for the author to appreciate the significance of the internal workings of a Recommender System and more importantly conduct research and experiments on ways of improving upon these internal mechanisms.

Appendix

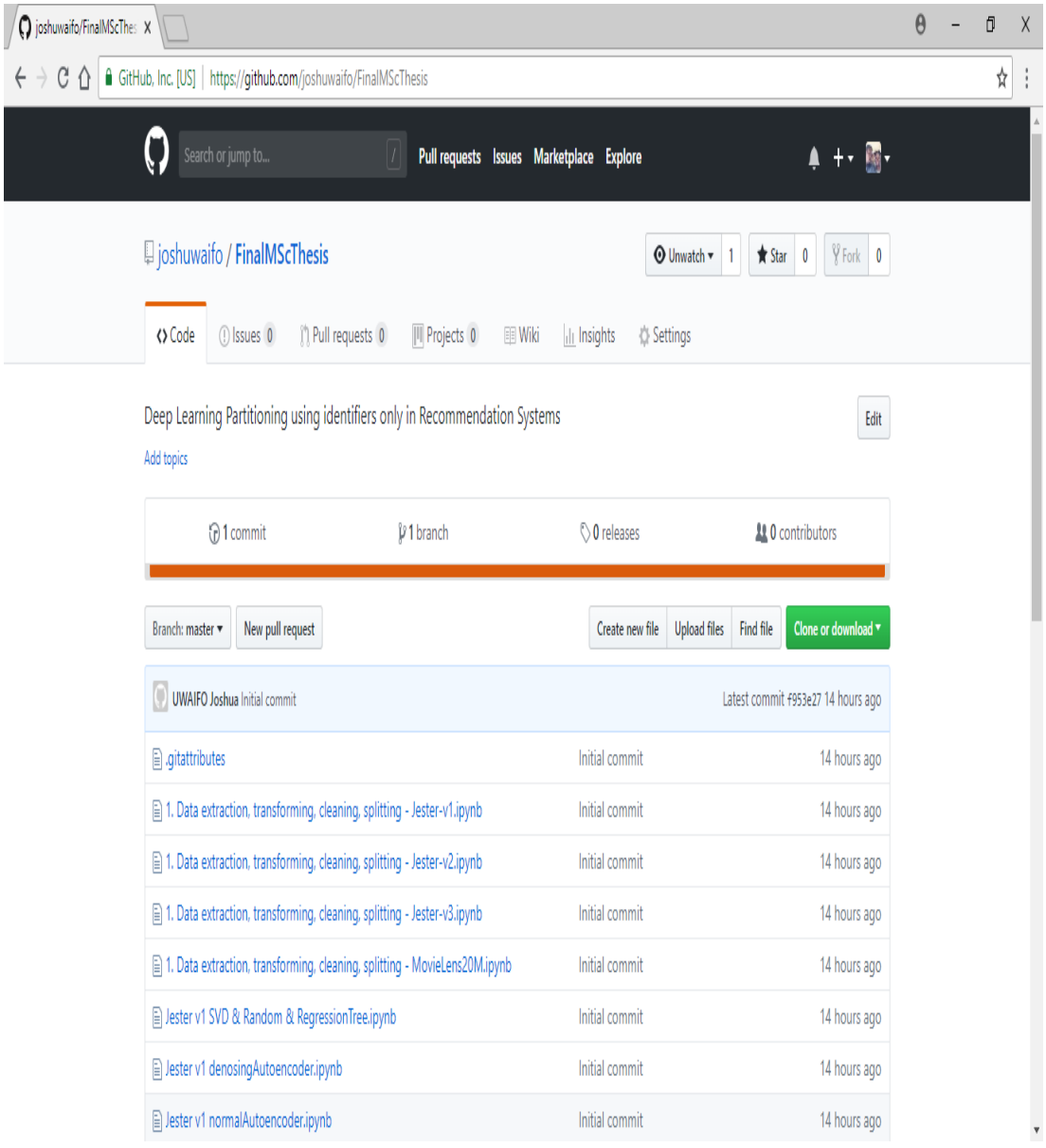

Figure 1: Appendix: Github repository for project

Bibliography

[1] A Abuomar, S Nouranian, R King, TM Ricks, and TE Lacy. Comprehensive mechanical property classification of vapor-grown carbon nanofiber/vinyl ester nanocomposites using support vector machines. 2014.

[2] P Andreadis, S Ceppi, M Rovatsos, and S Ramamoorthy. Diversity-aware recommendation for human collectives. 2016.

[3] Pavlos Andreadis. Coarse preferences: Representation, elicitation and decision making. 2018.

[4] M Aravamudan and A Rajasekharan. Methods and systems for segmenting relative user preferences into fine-grain and coarse-grain collections. 2009.

[5] J Bennett and S Lanning. The netflix prize. 2007.

[6] H Bourlard and CJ Wellekens. Links between markov models and multilayer perceptrons. 1989.

[7] L Breiman. Classification and regression trees. 2017.

[8] Rich Caruana. Multi-task learning, machine learning. 1997.

[9] F Chollet. Keras: The deep learning library, 2015.

[10] SciPy developers. Scipy: Python scientific library.

[11] Molaie et al. Artificial neural networks: powerful tools for modelling chaotic behavior in the nervous system. 2014.

[12] A Felfernig and R Burke. Constraint-based recommender systems: Technologies and research issues. 2008.

[13] Python Software Foundation. Python 2.7. 1990-2018.

[14] K Goldberg, T Roeder, D Gupta, and C Perkins. Jester dataset. eigentaste: A constant time collaborative filtering algorithm. 2001.

[15] Google. Tensorflow.

[16] F Harper and J Konstan. The movielens datasets: History and context. 2015.

[17] Netflix Inc. Netflix prize. 1997-2009.

[18] Keras IO. Keras: The python deep learning library.

[19] G Klambauer, T Unterthiner, A Mayr, and S Hochreiter. Self-normalizing neural networks. 2017.

[20] Y Koren, R Bell, and C Volinsky. Matrix factorization techniques for recommender systems. 2009.

[21] G Linden, B Smith, and J York. Amazon.com recommendations: Item-to-item collaborative filtering. 2003.

[22] Q Liu, E Chen, B Xiang, CHQ Ding, and L He. Gaussian process for recommender systems. 2011.

[23] T Mahmood and F Ricci. Improving recommender systems with adaptive conversational strategies. 2009.

[24] H McCormick and C Livett. Analysing the influence of the presentation of fashion garments on young consumers' online behaviour. 2012.

[25] L Melville and V Sindhwani. Recommender systems. 2011.

[26] Tom M. Mitchell. Machine learning. 1997.

[27] S Mitra and S Pal. Fuzzy multi-layer perceptron, inference and rule generation. 1995.

[28] MC Mukkamala and M Hein. Variants of rmsprop and adagrad with logarithmic regret bounds. 2017.

[29] Iain Murray. Machine learning and pattern recognition. 2018.

[30] Iain Murray. Netflix prize. 2018.

[31] Iain Murray. Pca and svd. autoencoder and principal components analysis. machine learning and pattern recognition. 2018.

[32] NumFOCUS. Numpy: Scientific computing python library.

[33] NumFOCUS. Pandas: Python data analysis library.

[34] United States Court of Appeals for the Ninth Circuit. Video privacy protection act.

[35] British Computer Society official website. Code of good practice. category 6030.

[36] University College London Patrick Guio. Good programming practice.

[37] P Resnick and HR Varian. Recommender systems. 1997.

[38] F Ricci, L Rokach, B Shapira, and P Kantor. Recommender systems handbook. 2010.

[39] Sebastian Ruder. An overview of multi-task learning in deep neural networks. 2017.

[40] Dino Sejdinovic. Sc4/sm8 advanced topics in statistical machine learning. 2018.

[41] Saed Seyed. Decision tree regression: Standard deviation reduction.

[42] British Computer Society. Intellectual property rights and software. content web document 2755.

[43] F Strub, R Gaudel, and J Mary. Hybrid recommender system based on autoencoders. 2016.

[44] SciPy Toolkits. Scikit-learn: A python scikit for machine learning.

[45] SciPy Toolkits. Scikit-surprise: A python scikit for recommender systems.

[46] Warwick University. Foundations of data analytics, cs910. 2017.

[47] Joshua Uwaifo. Informatics project proposal. 2018.