Final Year Project

GanRecommender

Daniel Hand

Student ID: 16405352

A thesis submitted in part fulfilment of the degree of

BSc. (Hons.) in Computer Science

Supervisor: Dr. Aonghus Lawlor

UCD School of Computer Science

University College Dublin

May 21, 2020

Table of Contents

Abstract

As products and services are increasingly procured through the Internet, companies have recognised the need to help customers find the correct item amongst the vast options offered. Recommender Systems (RSs) utilise available data to create personalised item suggestions for users, allowing companies to maximise customer satisfaction. This data may be in the form of clicks, purchases, ratings etc., with numerous models developed for each setting. Common to all is the problem of data sparsity, to get enough data for a RS you need a lot of users and much of this data is session-based data (e.g. news websites) with no link between a user's past sessions. To generate meaningful recommendations the system must rapidly learn from a user's characteristics, therefore previously successful approaches such as Content-based Filtering and Collaborative Filtering fall short. Generative Adversarial Networks (GANs) have shown promise in producing realistic data samples in the fields of image and text analysis, therefore, the goal of this project is to propose a design for a RS using GAN with reinforcement learning that can improve recommendation relevancy for session-based data over current state-of-the-art approaches.

Chapter 1: **Statement of Changes**

Outline of Report Structure:

Interim Report Sections:

1. *Abstract
2. *Project Specification
3. *Related Works and Ideas
4. *Data Considerations
5. *Outline of Approach
6. *Project Work Plan
7. *Summary and Conclusions

Additional Final Report Sections:

1. Data & Context
2. Core Contribution
3. Evaluation
4. Bibliography

*None of the Interim Report Sections were modified.

Chapter 2: **Project Specification**

The plan for this project is to implement a recommendation model using Generative Adversarial Networks (GANs) and reinforcement learning as the core recommendation strategy. There have been a few models developed recently, such as SeqGAN, and RecGAN which use combinations of a Recurrent Neural Network (RNN) as a generator and a Convolutional Neural Network (CNN) for the discriminator.

The performance of the GAN recommender system model will then be evaluated against current state-of-the-art.

The final aspect will be to assess the results on a small group of users.

Chapter 3: Introduction

3.1 Recommender Systems

Historians may classify the current era as The Data Age as it is estimated annual global IP traffic will reach 4.8 ZB per year by 2022[1]. To continue to manage this data we need more advanced systems for filtering information and modelling users that allow the extraction of knowledge. RSs are a subclass of information filtering that look to garnish some useful information from the overwhelming, unrelenting sea of data streamed across the internet each day. Content-based Filtering, Collaborative Filtering, hybrids of the two, along with many others are the most frequently used methods to accomplish this. These approaches all suffer from their own set of difficulties, but common to all is an inability to create associations when data is session-based, i.e. no user profile is created so each session is like a new user. Many existing state-of-the-art recommender systems rely on the users' feedback, either explicitly, such as ratings given by the users; or implicitly, such as clicks. This paper will discuss the approaches taken to construct a RS using a GAN to simulate users.

3.1.1 Content-based Filtering

Content-based filtering (CBF) extracts meaningful features from a user's preferences or historical ratings to rank suggestions. To rank an unrated item a correlation is calculated between its features and the extracted features, but this method may result in over-specification. For example, features commonly extracted in the movie domain are genre, director, actors etc. which may have no relevance to how a user rates a movie. An early application that relied solely on CBF resulted in "slightly more than one out of two" relevant recommendations [2]. A common problem all RS models face is data sparsity which is particularly persistent in CBF - users rarely take the time to rate an item with newly introduced items having no ratings at all. This class of problem is frequently called the "cold-start" problem and is also prevalent when there are few/no users or when a new user is registered.

3.1.2 Collaborative Filtering

Collaborative filtering (CF), a model of RS based on users' past behaviour, deals with some of the issues present in CBF models by finding similarities between either users or items. In CF, we form a ratings matrix (users X items), as shown in Fig 3.1. We use this matrix to find the most similar users (neighbours) to the target user. The previous (observed) preferences of the neighbours are then used for making recommendations. (Note: This definition is for user-based CF, item-based CF uses a similar method to compare item similarity).

In the literature, many different methods are used to extract neighbours, such as K-nearest neighbour (kNN)-based or matrix factorisation (MF)-based approaches. For example, in Fig 3.1, a ratings matrix, X, is created where an entry, X_{ij} represents how user i rated item j. To infer the relevance for a user of an unrated item x, the model must learn latent features, k, that can factorise the ratings matrix. In real-world examples the ratings matrix would be a sparse matrix

with few ratings so these processes (matrix factorisation etc.) decompose the ratings matrix into a user-topic matrix, U, and a topic-item matrix V. The features of these matrices are learned by the model by minimising the error, giving a representation of the underlying patterns of a user and item. Methods to compute the similarity between items or users, such as k-nearest neighbours, group items and users with similar latent features. We can then use these neighbours to make an inference about how a user would rate an unseen item.

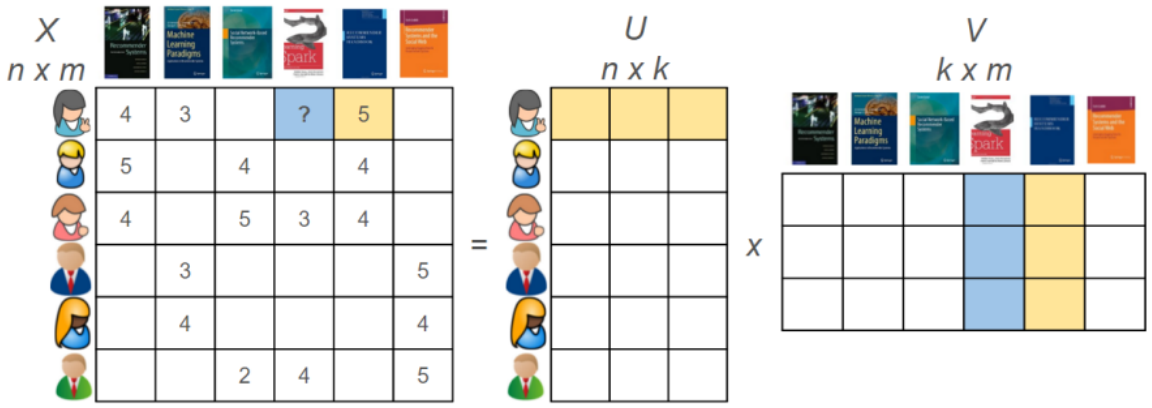


Figure 3.1: [3] Matrix factorisation can decompose a user-item matrix into a user-feature matrix multiplied by an item-feature matrix, where k is some set latent features of the user or item.

Some shortcomings of this method are the algorithms used, such as k-nearest neighbour, require massive computation[4] and again, the cold-start problem continues as new items cannot be suggested until at least one user rates the item. A variation of CF, called model-based CF, takes as input a user-item matrix and builds a model to estimate items based on association rules. This can mitigate some of the performance issues but as many users do not ever rate an item, association rules can be absent.

Combining CBF and CF into a hybrid model may eliminate some of the inadequacies in each approach while retaining the advantages of both. Alternatively, CF may be combined with one or more other methods such as Demographic Filtering where demographically similar users are grouped but acquiring demographic data can be problematic [5].

Since the introduction of the EU General Data Protection Regulation [6], privacy has become much more important. Websites now require consent to set persistent cookies that were previously used to track sessions, leading to an overall reduction in cookie setting behaviour [7]. As the above-mentioned methods fail to link previous sessions of the same user, we hope to offer an alternative RS that can emulate a user and alleviate some privacy concerns.

3.2 Machine Learning

RSs may be categorised by their category of Machine Learning (ML). Early hybrid RSs using Unsupervised Learning methods (K-means clustering) overcame the cold-start problem [8], though in this paper I will discuss Supervised Learning (SL)-based and Reinforcement Learning (RL)-based techniques as they are the most widely used.

3.3 Artificial Neural Networks

Supervised Learning (SL) algorithms have been applied to various topics from handwriting recognition to database marketing, with a wide range of algorithms available. The objective of any SL algorithm is to find a mapping from the set of inputs to the correct output. The "No Free Lunch" theorem derived by Wolpert and Macready[9] states that no one learning algorithm works best for every problem. RSs use an assortment of SL algorithms, from Support Vector Machines (SVM) to decision trees. In the RS domain, the state-of-the-art methods use Artificial Neural Networks and recently researchers have started using RL methods.

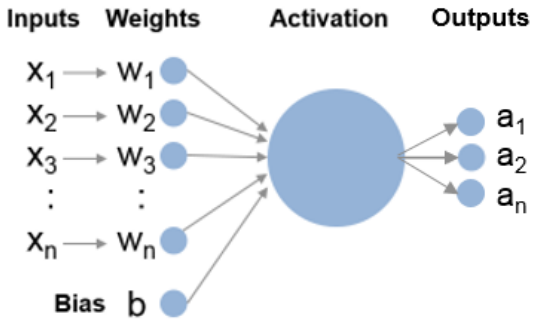


Figure 3.2: [10] High-level overview of an Artificial Neural Network.

Artificial Neural Networks, like the one shown in Fig 3.2, learn to map a set of inputs to an output and consist of simple, highly connected input nodes/neurons (x_n). Designed to imitate neurons of the human brain, each neuron represents some input (or set of inputs) with the interactions between neurons exhibiting some global behaviour of the network. A simple network may sum up these neurons, or a subset of them, with more complicated ANNs assigning *weights* (w_n) and biases (b) to the connections, allowing a neuron to be "turned on or off" to different degrees.

Each neuron's output, called its activation, is controlled by an Activation Function (AF) which may include a bias neuron. Below are some commonly used AF's:
the logistic sigmoid (left), hyperbolic tangent (middle) and ReLU (Rectified Linear Unit).

$$f(x) = \frac{1}{1+e^x} \quad , \quad f(x) = tanh(x) = \frac{e^x - e^- x}{e^x + e^- x} \quad \text{and} \quad f(x) = x^+ = max(0, x)$$

which normalise to the range [0,1], [-1,1] and [0,∞) respectively. ReLU is more commonly used as it is idempotent and can offer significant performance improvements over sigmoidal functions [11] with numerous variations for different applications (LeakyReLU). AFs produce a non-linear transformation of the input and a simple Perceptron may use a step function as an AF. In a Multilayer Perceptron, the AF of the first layer is fed forward to other hidden layers that use the preceding layer's output as input. The complexity of the decision making increases with each layer until the output of the final hidden layer in this *feed-forward* network is delivered to the output layer.

The learning component of an ANN is focused on minimising a cost function. A cost function is a measure of the inconsistency between predicted values and the corresponding correct values in the training data. Gradient Descent is an algorithm that makes small steps along a function to find a local minimum. By calculating the derivative of the cost function, we choose values such that the overall change in the cost function is negative, leading us closer to a local minimum of the function. This score is transmitted back through the network in a process called backpropagation (Fig 3.3), that involves calculating partial derivatives of the output (y), with respect to the input (x). By iteratively going backwards from the output, using the chain rule of calculus, the gradient of the cost function with respect to the weights and biases can be computed.

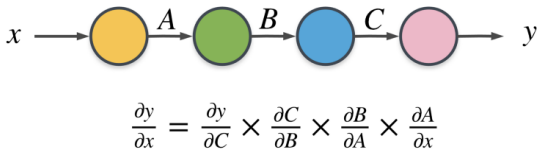


Figure 3.3: [12]Backpropagation using partial derivatives of the total derivative.

Backpropagation transformed SL algorithms by removing the need for a set of pre-classified data to train off, which may not be available. Training is instead achieved by setting initial weight and bias values randomly and continually (i) minimising the cost function, and (ii) propagating the changes back to the weights and biases in the network.

3.3.1 Reinforcement Learning

Reinforcement Learning algorithms consist of an agent that receives rewards by interacting with an environment. Rewards are issued to promote actions that move the agent from a state, s_i, to a more favourable state s_{i+1}. The agent's action selection is modelled as a function called a policy and the goal of the agent is to maximise the expected reward. RL requires choosing a trade-off between exploration and exploitation. Exploitation refers to taking actions that come from the current, best version of the learned policy—actions that we know will achieve a high reward. Exploration refers to the gathering of information to aid in decisions and can be implemented in numerous ways, from randomly sampling the entire space of possible actions to modelling the environment to compute the optimal action based on its expected reward.

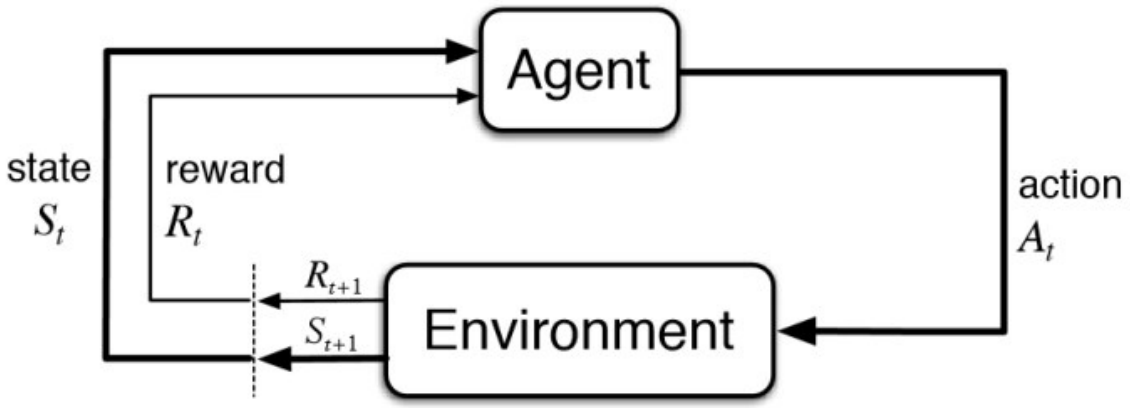


Figure 3.4: [13] The image above illustrates the basic principle of reinforcement learning.

An important intuition of RL is the Markov Property that states "the future is independent of the past given the present", or more formally:

$$P[S_t|S_{t-1}] = P[S_t|S_1, ..., S_t]$$

RL algorithms can be modelled as a Markov Decision Process or Partially Observable MDP depending on how observable the environmental state is - fully or partially, respectively. In either case, the goal remains to find an optimal policy to maximise the sequence of expected rewards. Many techniques are used to solve these processes from Dynamic Programming to Monte Carlo methods and value iteration methods such as Q-Learning. RL algorithms operate when no training data exists and have formed the foundation of deep RL systems that can beat world-class human players at games like chess and Atari [14].

The exploration-exploitation trade-off dilemma of RL is exemplified in the multi-armed bandit (MAB) problem. Named after one-armed bandit slot machines popular in casinos, the MAB problem can be imagined as a gambler at a row of slot machines deciding which machine to play and for how long. At each instance, the gambler may decide to exploit the machine that has the highest expected reward or explore the other machines and gain more information about their expected reward. Many solutions to the bandit problem exist with most relying on some assumption of the underlying distribution. Optimal solutions exist for special cases[15] but are typically difficult to compute and fail to generalise to realistic distributions. A heuristic is instead often employed such as randomised probability matching (Bayesian bandits) but more popular are semi-uniform strategies such as the epsilon-greedy strategy and UCB1[16].

The contextual multi-armed bandit is a variant of the MAB problem in which the bandit is shown some context from the environment before making each decision. Recommendation problems have incorporated contextual multi-armed bandits[17] and many show promise in dealing with two fundamental problems in RSs, the cold-start problem and diversity [18].

Deep Reinforcement Learning (DRL) models have become the new state-of-the-art methods in making recommendations [19] and promise to offer significant performance improvements by modelling dynamic aspects of users via a Deep Q-Learning framework. By considering current and future reward simultaneously, this framework has shown significant improvements in recommendation accuracy and diversity.

3.3.2 GAN

Building on the success of gradient descent applied to feedforward networks, Generative Adversarial Networks (GANs) [20] consist of a generative model trained to fool a feedforward classifier (discriminator). The discriminator attempts to classify all samples from the generative model as fake, while correctly recognising true samples from the training set.

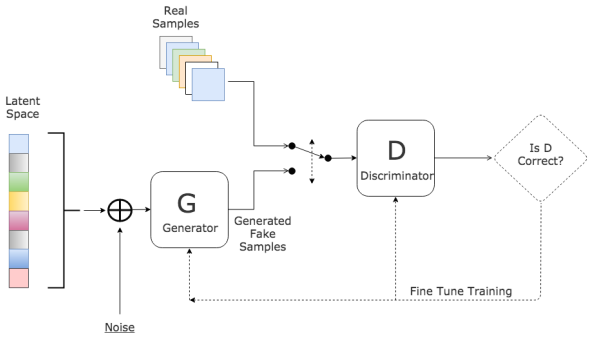


Figure 3.5: [21]A GANs structure consists of two ANNs that interact and learn from each other.

As we can see from the overview of the GAN framework in Fig 3.5, the discriminator is given real samples alongside the generated samples. The generator uses the output of the discriminator to adjust the model and create better samples from the latent space of features. In this sense, GAN is equivalent to a two-player minimax game, in which the generator is to maximise the probability of the discriminator making a mistake i.e. classifying a generated sample as real. A unique solution exists when the discriminator assigns a probability of 0.5 to both the generated and real samples meaning it can no longer distinguish the difference.

Chapter 4: Related Work and Ideas

4.1 IRGAN

Information Retrieval GAN (IRGAN) is the model proposed by Wang, Jun, et al.[22] in which a generative retrieval model is used to generate relevant documents from a candidate pool for a given query. A discriminative retrieval model aims to discriminate relevant document selections from non-relevant documents for a given query. A minimax game between the two models is constructed, optimising both until the generator can fool the discriminator by generating examples indistinguishable from the true examples. IRGAN was applied to three scenarios: web search, item recommendation and question answering showing promising results with "performance gains as much as 23.96% on Precision@5[1] and 15.50% on MAP[2]".

Table 4.1: **IRGAN results against Precision@N, Normalised Discounted Cumulative Gain (NDCG@N), Mean Average Precision (MAP) and Mean Reciprocal. Ranking (MRR).**

Model	Dataset	User	Item	Pool	Results
IRGAN (Web Search)	MQ2008-semi (LETOR)	784 queries	5 correct documents each	1k unlabelled docs. each	23.96% [1] & 15.50%[2]
IRGAN (Item Rec.)	MovieLens & Netflix	943 users 480k users	1,683 movies 17,770 movies	100k ratings 100m ratings	8.82% [2] & 14.38%[1]
IRGAN (QA)	InsuranceQA	12,887 questions	1 correct answers each	21,325 answers	2.38% [3] & 1.75%[4]

[1] - Precision@5, [2] - MAP, [3] – test set 1 & [4] – test set 2

- The web search scoring function was modelled with a 2-layer NN using features of query-document pairs on the LEarning TO Rank (LETOR) dataset.
- A matrix factorisation model on latent vectors of users (query) and items (documents) modelled the scoring function for item recommendation on the Movielens and Netflix datasets.
- A CNN was used to transform the question answering into a query-document scoring problem that measured their cosine similarities on the InsuranceQA dataset.

Across the board IRGAN outperformed previously successful algorithms such as maximum likelihood evaluation (MLE), RankNet and LambdaRank but the generative model gets improved more than the discriminative model in the pointwise version with the opposite being true for pairwise. The authors noted the discriminative model could be enhanced via strategic negative sampling which may increase its ability to handle more subtle negative examples. Empirical evidence has shown several improvements to IRGAN[23] but its influence in the field of GAN driven RSs makes it an important starting point for this project.

4.2 RecGAN

Inspired by RRN[24] and IRGAN, Recurrent Generative Adversarial Network (RecGAN) operates on latent features observed from short-term and long-term user behaviour along with exploiting the sequence of unlabelled generated items to better estimate relevancy. The authors use a combination of a recurrent neural network (RNN) with a custom gated recurrent unit (GRU) and a GAN to operate on two datasets, MyFitnessPal (MFP) and Netflix. The generative model is adopted to learn the rating of items for users. Ratings are both explicit, movie ratings or implicit, diet logs of repeated foods where a rating represents the likelihood of a user consuming a certain food. The discriminative model establishes the probability that a generated sample is from the true underlying distribution of user preferences. As with IRGAN, a minimax game is played between the generator and discriminator until the probability assigned to all generated samples by the discriminator converges to 0.5 (reaching a Nash equilibrium), meaning it cannot distinguish between real samples and generated samples.

We can see from Table 4.2, the addition of a GAN improved performance across the board. As it is an implicit density model[20], it captures relevant information from both high-volume and low-volume data distributions. This makes GAN an ideal candidate for both users who rate a lot and those who rate infrequently, as in the movie domain. Unlike IRGAN that ignored the temporal aspects of latent features, RecGAN captured the temporal patterns in varying detail via their modified GRU. The authors discuss how further research into variants of this GRU along with the addition of convolutional layers may improve the performance of the RecGAN model.

Dataset	Evaluation Metric	Vanilla GRU without GAN	RecGAN1 GRU + GAN	RecGAN2 GRU + GAN
Netflix 100m (user ID, item ID, rating & timestamp)	MAP@3	0.33	0.40	**0.41**
	MAP@5	0.30	0.37	**0.38**
	NDCG@3	0.46	0.55	**0.57**
	NDCG@4	0.46	0.53	**0.55**
	MRR	0.34	0.36	**0.39**
MFP (587,187 food diary records from & 9,896 users)	MAP@3	0.33	0.38	**0.41**
	MAP@5	0.30	0.37	**0.39**
	NDCG@3	0.37	0.47	**0.50**
	NDCG@4	0.37	0.44	**0.47**
	MRR	0.35	0.38	**0.42**

Table 4.2: **Comparison of different versions of RecGAN on test data. Vanilla GRU denotes a plain GRU based RNN without the GAN framework. Higher is better.**

4.3 SeqGAN

GANs are designed to operate on real-valued samples, the gradient of loss from the discriminator is used as a guide for the generator to slightly modify its parameters. Applying the GAN framework to sequential data makes little sense as there may not be a method to slightly modify a sequence of discrete tokens, as is the case in generating text [25]. Moreover, as GANs returns a score for an entire sequence once generated, partial sequences may not be accurately evaluated, the model may generate the best candidate token for the current sequence that is a poor candidate token for the entire sequence [26].

Sequence Generative Adversarial Nets (SeqGAN)[27], deal with the problem of partial sequences by incorporating the minimax game between the generator and discriminator with an expected-outcome model. This model, a Monte Carlo search, samples the remaining unknown tokens allowing the generator to take an action that may not have the highest immediate value. This is similar to the way a chess player might sacrifice immediate rewards (a pawn) to obtain a long-term reward (a king).

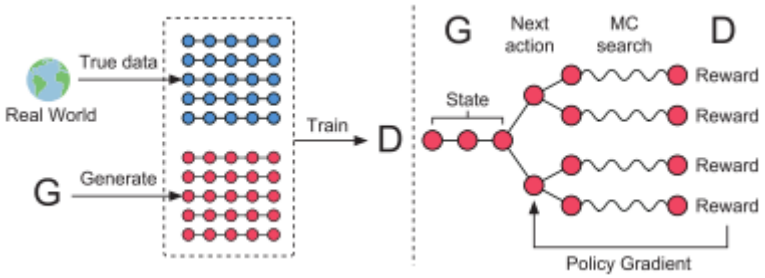


Figure 4.1: Diagram from SeqGAN describing the Generator (G) and Discriminator (D).

Fig 4.1 shows the design of SeqGAN with the generator comprising of a RNN and the discriminator, a CNN. Particular importance was assigned to the training process as the authors, along with Goodfellow [20], had noted that the discriminator must be optimal to ensure the generative process is effective and stable. As such, negative samples were combined with the generated samples allowing the discriminator to continually improve.

This ground-breaking paper showed how GANs can be applied to sequential data, which was previously deemed impossible due to the difficulties in backpropagating small changes in discrete data. The authors' use of a Long Short-Term Memory (LSTM) cell and Monte Carlo-based policy gradient allowed for adjustments to be made to the generators model with impressive results shown in the learning curve of SeqGAN compared to baseline models (Fig 4.2).

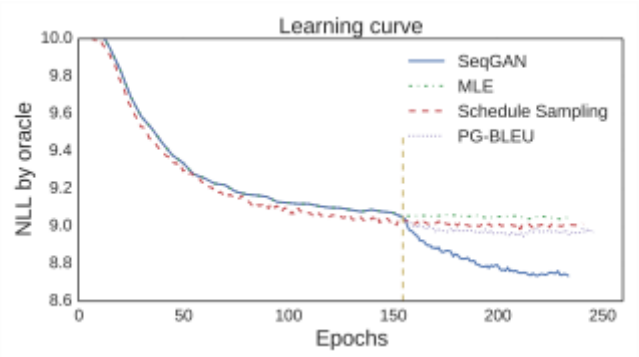


Figure 4.2: Negative log-likelihood convergence w.r.t. the training epochs. The vertical dashed line represents the end of pre-training for SeqGAN, SS and PG-BLEU.

Chapter 5: Data Considerations

In considering the data for this project, we consulted the frameworks discussed in Chapter 3 to guide our choice of dataset. IRGAN produced significant performance gains across the board but the learning curve for item recommendation on the MovieLens and Netflix Challenge datasets were not as stable as in web search and the temporal components of the ratings were not considered. RecGAN and SeqGAN incorporated these temporal components to varying degrees. SeqGAN, in particular, performs "comparably to real human data" at generating Chinese poems. In deciding a suitable dataset we considered three collections of data with an accompanying temporal component, MovieLens, Adressa and Criteo.

5.1 MovieLens

MovieLens is a collection of datasets from the GroupLens Research Project at the University of Minnesota consisting of movie ratings from users between 9th January 1995 and 31st March 2015. The format of the contained files are:

- **movies** - movieId, title, genres
- **ratings** - userId, movieId, rating, timestamp
- **links** - movieId, imdbId, tmdbId

with links *imdbId* and *tmdbId* referencing the relative movie on websites IMDb and The Movie DB respectively. The MovieLens 20m Dataset has 20 million ratings (density 0.53%) on a scale of 0.5 stars to 5 stars, across 27,278 movies by 138,493 users (only users with 20 or more ratings are included). As timestamps are included, a session may be analysed for each user. As this dataset uses explicit ratings, it does not provide the correct structure for building a CTR prediction model.

5.2 Adressa

Adressa are two datasets from Adresseavisen, a regional newspaper company in Trondheim, Norway consisting of news articles and user interactions from 1st January until 7th January 2017 (compact version) and 31st March 2017 (full version). Aside from the wealth of attributes contained in this data, one interesting aspect is the inclusion of free articles and articles behind a paywall. The interactions of users are in the form of clicks, providing a sequence of both session-based data along with subscriber data (subscribers have "pluss" in their URL). Adressa is an interesting dataset which has been thoroughly studied and may be useful to test our model on in the future.

5.3 Criteo

Criteo 1TB is an industry benchmark click-through rate (CTR) prediction dataset from the personalised retargeting company Criteo. This dataset is the largest ever publicly released ML dataset and consists of 24 files, corresponding to 24 days of data, collected from a portion of Criteo's traffic. Fig 5.1 show the structure of the dataset:

Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col10	Col11	Col12
0	40	42	2	54	3	0	0	2	16	0	1
0		24		27	5		0	2	1		3

Col13	Col14	Col15	Col16	Col17	Col18	Col19	Col20
4448	4	1acfe1ee	1b2ff61f	2e8b2631	6faef306	c6fc10d3	6fcd6dcb
10064		9a8cb066	7a06385f	417e6103	2170fc56	acf676aa	6fcd6dcb

Figure 5.1: [28] An example of the first 20 columns of Criteo 1TB.

- There are over 4 billion examples (rows) in Criteo 1TB but the semantic aspect of these 39 features is undisclosed.
- Each example is an ad served by Criteo, the the first column of each example a binary label representing if the ad was clicked or not (1 meaning the ad banner was clicked and 0 otherwise).
- Each row contains 13 features in integer values (primarily count features).
- 26 categorical features in 32-bit hashes.
- With 156 billion (dense) feature-values.
- Over 800 million unique attribute values.

The dataset is hosted by Microsoft's Azure Machine Learning, a cloud-based environment that makes it straightforward to train predictive models on this formidable dataset. With little information as to the content collected in this data, extensive pre-processing and planning will be needed in the early stages of our model's design but such a colossal dataset offers great potential.

Chapter 6: Outline of Approach

The remainder of this project will focus on the implementation of a recommendation model using GAN to produce high-quality recommendations from clickstream data. The details of this implementation are as follows:

- **Language**: Python as many libraries in existence that aid in the design and evaluation of GANs.
- **Libraries**:
  - *TensorFlow* [29], is an end-to-end open-source platform for training deep neural networks.
  - *SurpriseLib*, a flexible python framework for recommender systems, built on the foundations of the *scikit-learn* library. It also offers a set of built-in datasets and prediction algorithms.
  - *Contextual Bandits*, from David Cortes is a very useful library as it contains implementations of various methods from an assortment of literary papers.

To manage the tasks sufficiently we will break the approach into 5 categories as per the OSEMN taxonomy.

1. **Obtain** - As mentioned in Chapter 4, the Criteo dataset is hosted by Microsoft Azure ML and can be accessed via the URL `http://azuremlsampleexperiments.blob.core.windows.net/criteo/day_XX.gz`, where XX goes from 0 to 23 returning 1 of 24 days of traffic. The dataset in its entirety (all 24 days) can be downloaded via `~windows.net/criteo/day_{'seq-s','023'}.gz`. A shell script will be created to automate this process with "curl -O" to write output to a file. Direct access from Azure is also possible allowing users to run Hive queries directly via the Reader module, use the Learning with Counts module to create a set of count tables and run map-reduce jobs.
2. **Scrub** - Pre-processing or scrubbing the data will identifying and deal with missing data, inconsistencies and noise. Next, we will apply negative sampling on the dataset to deal with the immense volume of data and the natural bias within. Feature mapping will be a necessary third step to filter and remove irrelevant and associated features. A function will be designed to join users clicks by time thus creating a sequence of ads visited by each user.
3. **Explore** - Before we form and test any prediction model, we will need to explore the data. Python offers various tools to explore our data, such as *head()* function and visual functions from Matplotlib (*scatterplot()*, *histogram()* etc.). Dimension reduction by Singular Value Decomposition (SVD) or Principle Component Analysis (PCA) may provide insightful abstractions as exemplified by the Netflix prize winners[30]. Clustering, a popular unsupervised learning technique, can find structure and hidden patterns that may further our dimension reduction enabling faster computation on such a large dataset. Any features we can identify the users by will be important, IP address etc., along with a temporal component such as time since epoch.
4. **Model** - The general architecture of the proposed model is a modified version of SeqGAN, built to work on clickstream data from the Criteo dataset. The goal of any model is to correctly interpret and predict. As such, the design of our model will be contingent on

the outcome of the steps mentioned above but an overview of the two components of our model, the generator and the discriminator are as follows: Generator - a RNN with either Long Short-Term Memory (LTSM) cells or a GRU as a gating mechanism. The trick we will emulate from SeqGAN involves modifying the reward policy to work for partial sequences. In this sense, our generator will, at each timestep *t* of *T*, decide what token to generate next with respect to the sequence already generated. To do this, a Monte Carlo search will provide the token reward in terms of the overall expected reward for all possible remaining sequences. Formally, we have [27]:

$$Q_D^G(s = Y_{1:t-1}, a = y_t) = \frac{1}{N}\sum_{n=1}^{N} D(Y_{1:T}^n), Y_{1:T}^n \in MC(Y_{1:t}^n : N) \text{ for t<T} \tag{6.1}$$

where $Q_D^G(s,a)$ is the action-value function of a sequence, i.e. the expected cumulative reward starting from state s, taking action a and following the generator model G.
$D(Y_{1:T}^n)$ is the estimated probability (reward) sequence $Y_{1:T}^n$ is real where
$Y_{1:T}^n = (y_1, ..., y_t, ..., y_T,\ y_t \in Y$ and Y is the collection of candidate tokens.

5. **iNterpret** - Part of the interpretation will involve comparing the model to baseline algorithms (MLE, Mean Average Precision etc.) and interpreting the results. The advanced aspect of this project specification will be assessing the model on a small group of users. This will provide an excellent opportunity to evaluate the generated samples against new user data.

Chapter 7: Project Work Plan

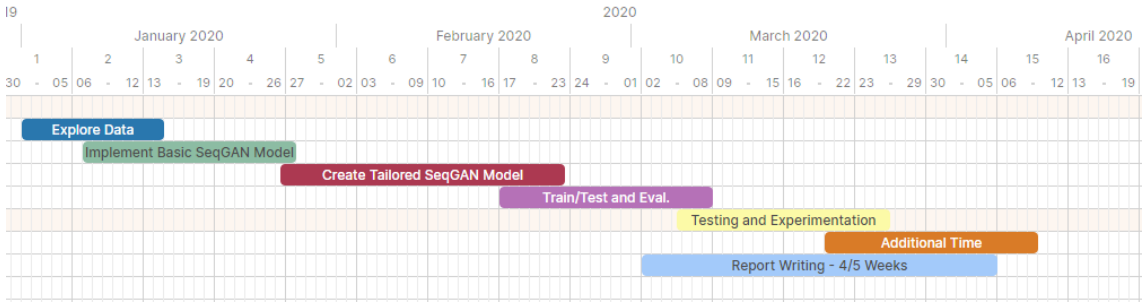


Figure 7.1: Gantt Chart of the planned implementation, testing and evaluation phase.

Explore Data - *1st - 14th January*
The Criteo dataset has vast complexity and size. As such, the initial stage will require exploration of the data to correctly identify the inherent structure. This may be straightforward but as the data is the most important aspect of any information retrieval model, I have allowed 2 weeks to complete this task.

Implement Basic SeqGAN Model - *7th - 27th January*
As the authors of the SeqGAN paper have provided source code for their model [31], this will form the backbone of our model. This task will most likely take less than the allotted 3 weeks but any extra may be used to start the next task.

Create Tailored SeqGAN model - *26th January - 23rd February*
The process of customising the basic SeqGAN model will depend on the quality of data and insights obtained in the first task. I envisage this taking 4 weeks to fully develop the model to work with the Criteo dataset with contingency time in task 6 if necessary.

Train/Test and Evaluation - *17th February - 8th March*
This task will include training and testing the model with a selected portion of the dataset. To gauge the efficacy of the model we will run baseline algorithms such as MLE to compare the results against.

Testing, and Experimentation - *5th March - 26th March*
Thorough testing will be needed to ensure the model is producing accurate and repeatable results. This task will also experiment with some of the model parameters, such as the rate of training of the discriminator and generator. 3 weeks will provide plenty of time to ensure any improvements can be recognised and acted upon.

Additional Time - *20th March - 9th April*
Additional contingency time to apportion to any task that requires it.

Report Writing - *2nd March - 5th April*
The report will require quite a bit of time so aspects of it will be formed throughout the entire semester with 4-5 weeks allocated to draft the final version.

Chapter 8: **Summary and Conclusions**

In this project, we looked at existing methods to build recommendation systems and discussed some of the advantages and disadvantages of each.

CBF, CF and hybrid models have long been the de facto standard for RSs but fail to deal with many issues arising in session-based user data, notably the cold-start problem (lack of data due to new user and/or item) and their inability to operate on data without user profiles. Matrix factorisation techniques allowed for extraction of latent features in CBF models and is used in many different RS settings. The computational complexity required to perform runtime factorisation limited the scalability of CBF and CF RSs.

Recently, state-of-the-art methods use machine learning algorithms such as ANNs, provide a means to model complex data and discover underlying patterns. ANNs can learn the importance of latent user and item features by mapping a set of embeddings (input) through one or more hidden layers to an (output). Optimisation occurs by gradient descent (differentiation of a multi-variable function) to find a local minimum of that function.

Another area of approach, reinforcement learning, models the user-item scenario as a contextual multi-armed bandit problem [17] where a bandit (user) can choose from k-arms (items). The context is the latent information about the bandit and k-arms. The aim is to model a user by maximising the expected reward of the bandit, where a reward is the user selecting the arm the model suggests. The size of the state space is often so vast that this method alone can be problematic.

With advances in deep learning, more complicated (deep) neural networks like CNN's have been extended from their origin in computer vision [32] to work in RSs. RNNs use loops to act like memory, therefore, enabling more context of the user and item to be incorporated into the networks model. This has provided a basis to work with sequential data, such as natural language processing and clickstream data. The latter is of particular importance as the previous methods cannot perform well in the session-based RS setting. As sessions are often short and connecting them difficult, latent patterns of a user may be spread across multiple sessions and multiple devices.

A class of machine learning algorithms, invented by Ian Goodfellow et al.[20], called GANs acquired a lot of attention. This broad category refers to the combination of a generative neural network and a discriminative neural network that compete in a minimax game comparable to a forger and a cop. The generator (forger) is constantly learning to create brand new fake data that it can fool the discriminator (cop) into classifying as real data from a dataset. The generator continually improves the generated samples by using the output of the discriminator to guide it. GANs have been pivotal in generating user data in the RSs and are the subject of continued research with a recent implementation on sequential data generation (SeqGAN) [27] inspiring this project.

The goal of this project is to extend the fantastic work of Lantao Yu et al. on SeqGAN to a CTR prediction model that can compete with current state-of-the-art models.

Chapter 9: Data & Context

This chapter gives insights into the data, identifying issues and the steps taken to address these issues.

There are a small number of open-source sequence based dataset for recommendation. One very large advertisement click data set was open-sourced by Criteo Labs, but it is fully anonymised and is not possible to track the user who clicked on the advertisement, so there are issues using it in this project. The other dataset I investigated is a collection of song plays from Last.fm which allows us to track the users by an anonymised id and provides some information on the songs.

We describe the datasets and the processing involved to get them into the format required for the SeqGAN.

9.0.1 Criteo 1TB

The subject matter of the dataset is not pertinent but the presence of a unique user identifier was necessary for the creation of user sessions. The structure of the Criteo 1TB dataset was unclear due to the obfuscation of the features. As such, a decision was made to use a different dataset with explicit user identifiers rather than concentrating efforts on de-obfuscating the Criteo dataset.

9.0.2 Last.fm 1K

Last.fm is an online music database founded in 2002 that focuses on generating dynamic playlists for users based on songs they have previously liked or disliked. The Last.fm 1K dataset contains the listening habits of approximately 1000 users, collected by Òscar Celma[1] up until 5th May 2009. Each entry of the dataset corresponds to a single user listening to a track and was collected from Last.fm API, using the user.getRecentTracks() method. With a unique user identifier (e.g. user_001000) and timestamp for each entry, user sessions could be created.

9.0.3 Data Format

The Last.fm dataset contains two tab seperated files:

userid-timestamp-artid-artname-traid-traname.tsv ~2.53GB uncompressed

	userId	date	artist_id	artist_name	track_id	track_name
0	user_000001	2009-05-04 23:08:57+00:00	f1b1cf71-bd35-4e99-8624-24a6e15f133a	Deep Dish	NaN	Fuck Me Im Famous (Pacha Ibiza)-09-28-2007
11115	user_000001	2008-03-04 15:22:46+00:00	0f5a9adc-fcca-4d04-bf31-d7d1164133f9	Jimpster	9c35639a-0d0c-4e21-b35c-9bf6501be5a1	Square Up
11116	user_000001	2008-03-04 15:15:57+00:00	0f5a9adc-fcca-4d04-bf31-d7d1164133f9	Jimpster	94102c3d-4ffb-453c-bc3b-6cfb5bf99d8e	Dangly Panther

Figure 9.1: Example of lines in userid-timestamp-artid-artname-traid-traname.tsv, sorted by userId, for user_000001 with NaN value for track_id

[1] http://ocelma.net

userId	-	unique identifier of user (i.e. user_XXXXXX where $1 \leq X \leq 1000$)
date	-	regional time and date when the user listened to that track
artist_id	-	36 character MusicBrains ID (Universally Unique Identifier)
artist_name	-	artist/band name
track_id	-	36 character MusicBrains ID (Universally Unique Identifier)
track_name	-	track name

userid-profile.tsv ~38KB uncompressed

	#id	gender	age	country	registered
0	user_000001	m	NaN	Japan	Aug 13, 2006
1	user_000002	f	NaN	Peru	Feb 24, 2006
2	user_000003	m	22.0	United States	Oct 30, 2005
3	user_000004	f	NaN	NaN	Apr 26, 2006
4	user_000005	m	NaN	Bulgaria	Jun 29, 2006
5	user_000006	NaN	24.0	Russian Federation	May 18, 2006

Figure 9.2: First 5 lines in userid-profile.tsv with numerous NaN values for the age and sex columns

#id	-	unique identifier of user (i.e. user_XXXXXX where $1 \leq X \leq 1000$)
gender	-	gender of user (i.e. m or f)
age	-	age of user
country	-	user's country of residence
registered	-	user's date of registration for Last.fm

9.0.4 Data Statistics

The file **userid-timestamp-artid-artname-traid-traname.tsv** is formatted one entry per line

- Total Lines: 19,150,868
- Unique Users: 992
- Unique Tracks: 1,084,866
- NaN Values: 2,168,588

Exploring the data highlighted some issues:

1. No track duration
2. Many tracks had Not A Number (NaN) values for their track_id
3. Numerous tracks had consecutive plays by a user as though the track had been set to repeat. One user had 184 consecutive plays of a single track
4. A method to decide the session cut-off point (Δt) would be needed, i.e. at what interval between songs do we decide that a new session has started

Resolutions to these problems are discussed in the chapter Core Contribution.

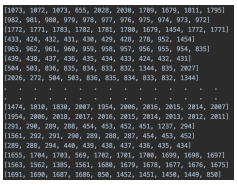
Another advantage the Last.fm dataset offers over the Criteo 1TB dataset is the nature of the data. The advanced task of this project was to evaluate generated data on a selected group of users. Using the Criteo dataset would involve presenting users with generated ad clicks. This may be harder to assess on a small group of users so the generated music sessions offer a better platform to evaluate the results.

Chapter 10: Core Contribution

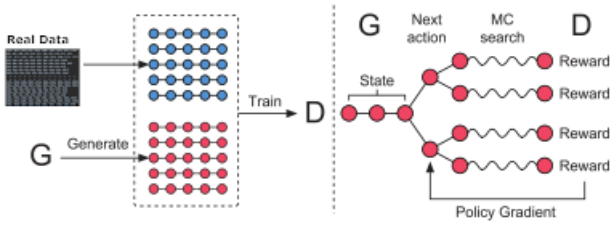
10.1 Design

10.1.1 Model

To implement the SeqGAN model, real sequence data is needed. Where the authors of SeqGAN[31] used a randomly initialised LSTM (Oracle) model, I used sequences of track plays (music sessions) as the source of real data Fig 10.1 to train the discriminator. These sessions were extracted from the individual track plays to create music sessions comprised of one or more track plays.



(a) Extracted music sessions



(b) Tailored SeqGAN model

Figure 10.1: Real data in the form of music sessions is fed to the discriminator alongside the generated music sessions

The authors of SeqGAN, based their work on that of Ofir Nachum[1]. While SeqGAN uses a randomly initialised LSTM to represent the real data, the model by Ofir Nachum uses an E-book version of Moby Dick[2] as the real data. Although using a randomly initialised LSTM as real data allows the performance to be evaluated exactly, Ofir Nachum's approach, using a Gated Recurrent Unit (GRU), is better suited as it provides the opportunity to use the Last.fm dataset as real data.

10.1.2 Sessions

To generate sequence data the model needs a collection of sequences (or listening sessions in the case of the Last.fm dataset). Ofir Nachum extracted 100k characters from Moby Dick to use as the token stream. To create sequences from the Last.fm dataset, a cut-off point must be selected in which it is deemed a new session has begun. This cut-off point, (Δt), is the window of time between two consecutive plays of a song by a user, i.e. $\Delta t = 10$ minutes would classify the track plays in Fig 9.1 as separate sessions as 14:20 - 14:07, 13 minutes, is greater than Δt.

userId	date	artist_id	artist_name	track_id	track_name
user_000001	2009-05-03 14:20:15+00:00	ce559a88-58ba-4d8a-8456-9177412d609c	Masomenos		Eight
user_000001	2009-05-03 14:07:40+00:00	ce559a88-58ba-4d8a-8456-9177412d609c	Masomenos		Seven

Figure 10.2: Example of two consecutive track plays with NaN entries for track_id

[1] https://github.com/ofirnachum/sequence_gan
[2] http://www.gutenberg.org/cache/epub/2701/pg2701.txt

A session is comprised of one or more track plays for a specific user, represented by their track_id(s). A Pandas DataFrame, shown in Fig 10.3 holds the sessions for all users. Each entry in this DataFrame is comprised of a list of track_ids, defining the list of tracks that a user listened to. When the difference in time between two consecutive track plays is greater than the session cut-off (Δt), a new entry is started representing the start of a new session.

```
[433, 424, 432, 431, 430, 429, 428, 278, 952, 1454]
[2021]
[2020, 2019, 2025, 278]
[963, 962, 961, 960, 959, 958, 957, 956, 955, 954, 835, 504, 503, 836, 835, 834, 833, 832, 440]
[439, 438, 437, 436, 435, 434, 433, 424, 432, 431, 430, 429, 428, 272]
[504, 503, 836, 835, 834, 833, 832, 1344, 835, 2027, 2026, 272, 504, 503, 836, 835, 834, 833, 832, 1344, 835, 2027, 2026, 504, 503,
[2022, 2021, 2020, 2019, 1773]
[1474, 1810, 1830, 2007, 1954, 2006, 2018, 2017, 2016, 2015, 2014, 2007]
[1954, 2006, 2018, 2017, 2016, 2015, 2014, 2013, 2012, 2011, 2007, 1954, 2006, 2018, 2017, 2016, 2015, 2014, 2013, 2012, 2011, 2007,
[291, 290, 289, 288, 454, 453, 452, 451, 1237, 294, 293, 292, 291, 290, 289, 288, 287, 454, 453, 452, 451, 1237, 294, 293, 1237]
[294, 293]
```

Figure 10.3: Example of sessions created with $\Delta t = 20$ minutes

As most sessions are either less than or greater than 10 track plays, only sessions of length 10 or greater are selected for training. For the sessions greater than 10 tracks, a random selection of 10 tracks from the entire session is chosen. This reduces the usable data considerably and is a significant limitation of the SeqGAN model.

10.1.3 Evaluation

The evaluation metric chosen, bilingual evaluation understudy (BLEU), uses modified N-gram precision on blocks of tokens to return a score bewtween 0 and 1[33]. A perfect score of 1 indicates the candidate sequence appears exactly in the reference token stream. It has been shown to have a high correlation with human judgement for machine translations by averaging out individual judgement errors over the entire corpus.

10.2 Pre-Processing

An essential ingredient to the SeqGAN are sessions, which for the LastFM dataset are a collection of songs which are played together by the user. The LastFM dataset does not provide session data, but rather just a list of the songs and the time at which they are played. We have to process the data to extract the sessions, and we have to analyse the data carefully to figure out the best way to extract the sessions.

10.2.1 Session Cut-Off

To find the optimal value for the session length, a plot was created of the number of sessions N versus the minimum session length Δt for session lengths ranging from 5 minutes to 35 minutes. Fig 10.4 gives an indication that the optimal value for Δt is likely between 8 and 25 minutes as the number of sessions for values outside of this range are too big and too small respectively.

Graphing the frequency of the number of tracks in each session helped narrow down the optimal value of Δt. Sessions were created for all users for values of Δt ranging from 5 minutes to 35 minutes in 5 minute increments. The session lengths were then placed in 20 bins of equal distance

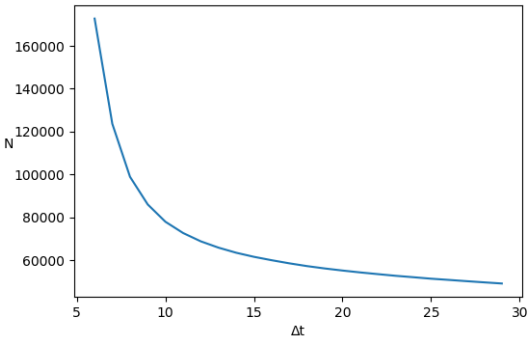
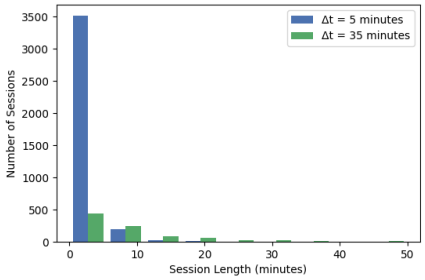
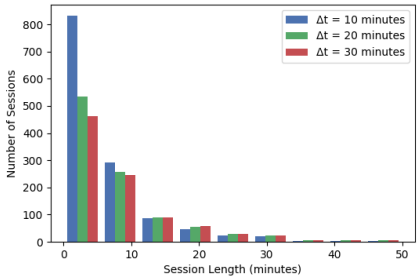


Figure 10.4: Number of Sessions (N) vs. Minimum Session Size (Δt)

so the frequencies could be plotted on a histogram.



(a) Dispersion of session lengths for the two extremes, 5 and 35 minutes



(b) Dispersion of session lengths for 10, 20 and 30 minutes.

Figure 10.5: shows the difference in dispersion of session lengths when the session size, Δt, is varied

Further calculations verified that the optimal value for Δt was 20 minutes with it containing the highest percentage of the data within one standard deviation from the mean

Session Size (minutes)	Data within one standard deviation (%)
5	73.3
10	52.7
15	53.1
20	54.7
25	53.3
30	52.6
35	57.1

10.2.2 Dataset Issues

Missing Track Duration

As the length of the tracks is not present in the data, the calculations above assume all track lengths are roughly the same. This is highly unlikely and assuming so may mean long tracks (tracks equal to or greater than the minimum session length) terminate the current session when in reality they should not. To estimate the length of each song, the time in between it and the subsequent song being played was summed and divided by the number of plays for that song. This

gave an average of the interval but was not accurate as, firstly, the song may actually appear at the end of a session and secondly, many songs had very few plays. This step was ommitted from the pre-processing but another possible option is to utilise the Last.fm API to return the track durations. This requires a Last.fm API key, which was not acquired, but would be an interesting future step.

Missing Track IDs

While pre-processing the data it was noticed that the Last.fm dataset had NaN values for various track_ids (as can be seen in Figs 9.1 & 10.2). NaN values represent missing data in a Pandas DataFrame and as the track plays would be represented using their track_id, this missing data would be problematic. To remedy this, a new track_id was assigned to each track using the Pandas DataFrame function, unique(), on the track_name column. This removed the problem of NaN values while reducing the amount of memory needed. The initial track_ids were a 512 bit but the new track_ids need only 21 bits to give each of the 1,084,866 unique tracks an id ($2^{21} > 1,084,866$).

Repeated Track Plays

Another issue discovered while exploring the data was that many sessions contained consecutively repeated tracks. These tracks were repeated various numbers of times, with one session having a track played 184 times in a row. 20,636 of the sessions contained a track played more than once consecutively. Of course, some users may like a song enough to play it numerous times in a row, but a track played 184 times in a row seemed questionable. In the end it was decided to leave these repeats in as it is not our decision to decide if this is valid data or not.

10.2.3 Creating Sessions

Using the optimal value for minimum session length, user sessions were created. In Python, the data analysis library Pandas offers many powerful methods of sorting, grouping etc. to aid in the process of creating sessions. With such a big dataset (19.1 million entries), even this pre-processing step took a great deal of time so during the early stages 1-10% of the data was sampled. Once all modifications to the code were complete, the UCD server known as theengine was utilised to execute the code on the entire dataset. With 8x 64GB RAM modules, 4x TURBO GTX 1080TI GPUs and 2x XEON 2.1GHz processors this server could use all 19.1 million examples while not suffering from memory errors when training the GAN.

10.3 Model

10.3.1 N-Gram Verification

The Sequence GAN model designed by Ofir Nachum uses a token stream as real data input with a N-gram model to verify any generated data does not exist in the real data. The N-gram method is popular in natural language processing and allows a quick check of the correctness of generated data in this scenario. Iterating through the entire generated sequence in blocks of N (with N = 3) a check was performed to verify if each block of N existed in the real data or not.

Chapter 11: Evaluation

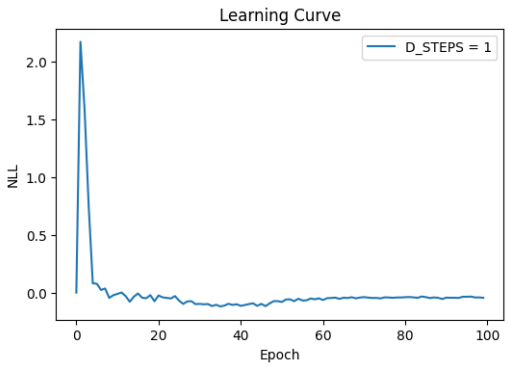
11.1 Training

The Discriminator (D) is trained using the generated music sessions alongside the real music sessions. The classification score of these sessions (0 if D classifies session as fake, 1 if D classifies session as real) is used to train the Generator (G) via policy gradient. G becomes better at generating sessions that will "fool" D i.e. attain a higher classification score.

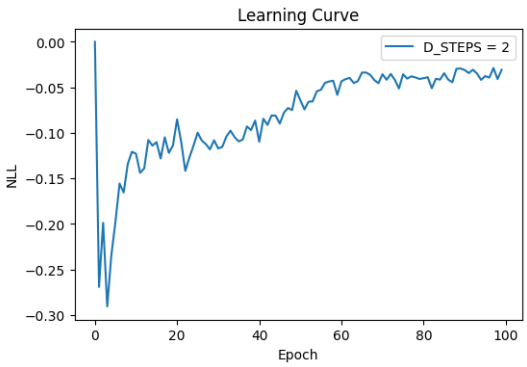
1. For the experimental evaluation we initialise the parameters of the GRU network as follows:
   - Number of Embeddings: 2,168,588 (number of unique tracks)
   - Embedding Dimension: 20
   - Hidden State Dimensions: 10
   - Sequence Length: 10
2. I then generate a number of sequences of length 10 as the training set S for the generative models
3. In the SeqGAN algorithm, the training set for the discriminator is comprised of the generated samples with the label 0 and the instances from S with the label 1.
4. The number of times to train the Discriminator per generator step is controlled by D_STEPS, while the rate at which the model moves from supervised training to unsupervised training is controlled by the $CURRICULUM\_RATE$. Using a grid-search to tune these Hyper-parameters the following values were selected:
   - D_STEPS: 3
   - $CURRICULUM\_RATE$: 0.02
5. We use the BLEU metric, a metric which measures the similarity between a generated sequence and the training data.
6. The training is shown in Fig 11.1 where we plot the negative log likelihood for each epoch for D_STEPS values of 1 to 4.

11.2 Hyperparameter Tuning

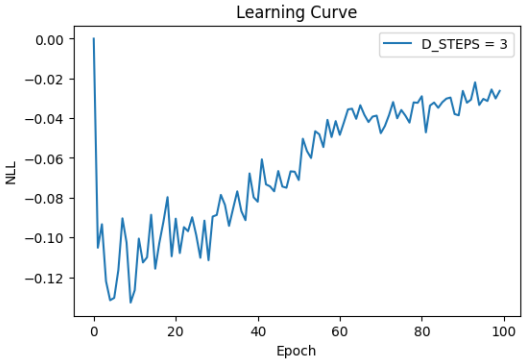
We can see that our SeqGAN algorithm is able to learn to generate synthetic sequences in a stable way. The model converges after about 50 epochs.



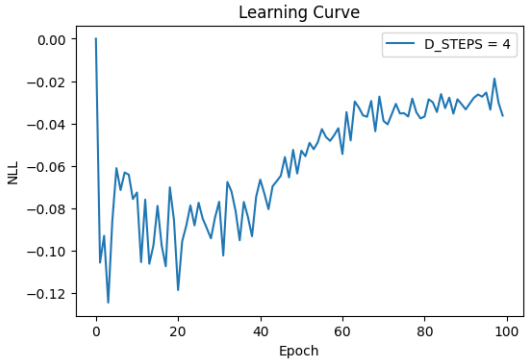
(a) NLL when 1 discriminator step for every generator step. Final NLL = -0.0447



(b) NLL when 2 discriminator steps for every generator step. Final NLL = -0.0308



(c) NLL with 3 discriminator steps for every generator step. Final NLL = -0.0262



(d) NLL with 4 discriminator steps for every generator step. Final NLL = -0.0363

Figure 11.1: With the smallest Negative Log Likelihood of -0.0262, training the discriminator 3 times for every generator training instance proved optimal giving a value of 3 for D_STEPS

11.3 Evaluation Issues

The lack of clear evaluation metrics for sequences generated by a GAN is well documented [34]. Methods such as maximum likelihood estimation (MLE), where the cross-entropy loss is minimised, suffer from various problems:

Exposure Bias - the next token is generated based on the ground-truth tokens during training. As no ground-truth labels are available during testing, the next token is generated based on the previously generated tokens, causing a bias.

KL-Divergence Direction - MLE is equivalent to minimising KL-Divergence which, while producing more diverse generated samples, produces lower quality samples than reverse KL-Divergence [35]. As Reverse KL-Divergence will produce less diverse samples, there is a trade-off between the direction of divergence.

General Responses - MLE will often only produce very general responses, thus offering little information as to the quality of generated samples [36].

With these difficulties in mind, I chose to use the BLEU metric to evaluate the quality of the generated samples.

11.4 Peer Evaluation

A small group of peers evaluated the samples by assigning a score out of 10. As the generated samples were length 10, 0/10 would mean a user thought none of the tracks belong together in a session while 10/10 meant the user thought that all of the tracks belong together in a session i.e. the session perfectly resembles a real session.

These results are displayed in Fig 11.2 alongside the BLEU metric for each of the 4 sessions generated between the 97^{th} and 100^{th} epoch inclusive.

Session 1 - Epoch 97: [1024, 145, 761, 1265, 647, 879, 857, 640, 1282, 1499]

1. Music For 18 Musicians: Section Vii - Steve Reich
2. Breaking News - Designed People
3. Mechanic - Reflection
4. Moog Dub - 2562
5. Inner Soul [Slow Version] - Roland Appel
6. Birds Of A Feather - Herbert
7. Dogbite (A1 People Mix) - The Black Dog
8. For Your Love - Benny Sings
9. Dayvan Cowboy - Boards Of Canada
10. Hidden Place - Björk

Session 2 - Epoch 98: [876, 1484, 1304, 1486, 1557, 1486, 1976, 1557, 1557, 1363]

1. Sombre Detune - Röyksopp
2. Miu - Steve Orchard
3. The Beginning Of The End (Of The End Of The Beginning) / Montage (Sensitive Mix) - Derrick May
4. Som En Film - Spinform
5. Sun Will Shine - Part Time Heroes
6. Som En Film - Spinform
7. All I Need - Radiohead
8. Sun Will Shine - Part Time Heroes
9. Sun Will Shine - Part Time Heroes
10. Heirloom - Björk

Session 3 - Epoch 99: [406, 972, 973, 974, 412, 413, 414, 1119, 355, 513]

1. Blues Walk - Lou Donaldson
2. Ti Born - Plaid
3. Tak 4 - Plaid
4. Porn Coconut Co - Plaid
5. Got To Get Your Love - Clyde Alexander
6. Can'T Fake The Feeling - Geraldine Hunt
7. Boogie Oogie Oogie - A Taste Of Honey
8. A Perfect Day - Bobby Cole
9. In A Distance - Minilogue
10. Bitter Sweet Symphony (Original) - The Verve

Session 4 - Epoch 100: [941, 938, 1611, 1373, 1612, 1613, 607, 608, 609, 610]

1. Color Of Feels - Kaito
2. We Were Born Here - Kaito
3. The Session (Kuniyukis Piano Mix) - Kuniyuki Takahashi
4. All These Things (Theo Parrish Remix) - Kuniyuki Takahashi
5. ouch - A Mountain Of One Peyote Remix - Kuniyuki Takahashi
6. All These Things (Kuniyuki Self Remix) - Kuniyuki Takahashi
7. Third Wind - Pat Metheny Group
8. (It'S Just) Talk - Pat Metheny Group
9. Last Train Home - Pat Metheny Group
10. So May It Secretly Begin - Pat Metheny Trio

	Peer 1	Peer 2	Peer 3	Peer 4	Peer 5	**Average**	**BLEU**
session 1	3/10	5/10	4/10	4/10	2/10	**3.6/10**	**34.7%**
session 2	6/10	2/10	4/10	5/10	4/10	**4.2/10**	**32.8%**
session 3	7/10	3/10	3/10	6/10	4/10	**4.6/10**	**42.5%**
session 4	9/10	6/10	5/10	5/10	7/10	**6.4/10**	**58.1%**

Using these 4 generated sessions from the final 4 epochs, I averaged the scores given for each track by 5 of my peers. These scores were compared to the BLEU scores Fig 11.2.

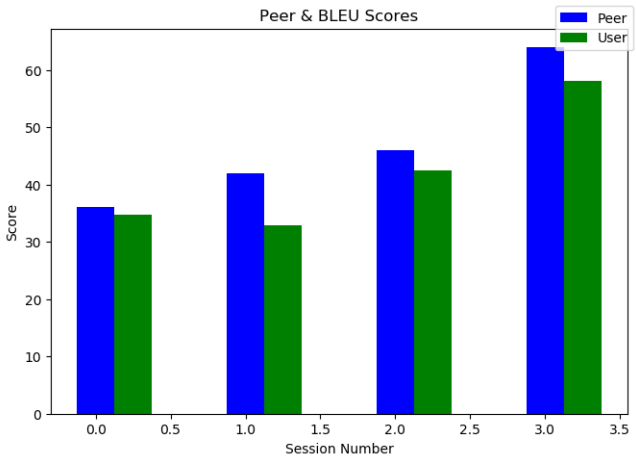


Figure 11.2: Comparison of scores by peers and using BLEU for 4 generated sessions

The results in Fig 11.2 show the BLEU metric is indeed a good representative metric for human judgement, with quite similar scores for music sessions to the scores given by my peers. It should be noted that quite a significant amount generated sessions included repeated tracks, which is representative of the many repeated tracks in the real data. Session 2 - Epoch 98 has 1 song repeated 3 times and another repeated twice. This session gained the lowest individual score of any other, 2/10 by peer 2, which raised further doubts about the presence of repeated tracks in the real data.

Chapter 12: **Conclusions**

In this project I built a synthetic sequence generation algorithm, based on SeqGAN and applied it to the problem of generating artificial session data for recommender systems datasets.

SeqGAN algorithms have been used to generate other kinds of sequence data and show great promise to assist with the training of sequence based algorithms. Many modern recommendation problems are based on sequential data, such as advertisement recommendation, shopping baskets on ecommerce sites, song and playlist recommendation and many others. Training these algorithms is hard, and often the data is lacking or not available. A synthetic sequence generation algorithm for training will allow these algorithms to be trained more efficiently and effectively.

I built a SeqGAN algorithm and trained it on a LastFM dataset to generate song sequences for training. The algorithm can learn to generate sequences in a stable way, and provides the foundation for further incorporation into more sophisticated reinforcement learning algorithms for recommendation.

I wrote a significant amount of Python code to implement my own SeqGAN model. Throughout the data exploration and pre-processing phases I learnt a great deal about the difficulties in using large data and the necessary procedures to generate usable sequence data. It was interesting to postulate then evaluate the significance of different ideas. Figuring out how to extract sessions from the Last.fm data was also very interesting, and in the future it would be important to conduct more extensive evaluations on this aspect of the work.

Future improvements could focus on generating sequences with richer features to allow better personalisation, and on incorporating the SeqGAN algorithms in a natural way into the training of reinforcement learning recommendation algorithms.

User evaluation is very difficult and in particular, when using such a small group of peers to evaluate. Future work could look at how to collect feedback from more live users in a real world setting. A smartphone app that asks users to either rate a session, or select the session they think is real, would be an excellent way to do this.

12.1 Code Repository

The code for this project is available at the following GitHub repository:
https://github.com/insight-ucd/daniel_seqgan_fyp

Acknowledgements

I would like to take this opportunity to express my profound gratitude to my supervisor, Dr Aonghus Lawlor, for his time, patience and invaluable suggestions throughout this endeavour.

I also take this opportunity to express my sincere appreciation to Makbule Gulcin Ozsoy for her keen interest, expertise and continued support.

Their immense knowledge and practical experience was essential to the work in this project.

I hereby certify that the content of this report is entirely my own.

Bibliography

1. Cisco, V. Cisco visual networking index: Forecast and trends, 2017–2022. *White Paper* **1** (2018).
2. Meteren, R. V. & Someren, M. V. Using Content-Based Filtering for Recommendation. *ECML/MLNET Workshop on Machine Learning and the New Information Age,* 47–56. ISSN: 15506606 (2000).
3. Hristakeva, M. *A Practical Guide To Building Recommender Systems* https://bit.ly/356HJa2.
4. Kataria, A. & Singh, M. A review of data classification using k-nearest neighbour algorithm. *International Journal of Emerging Technology and Advanced Engineering* **3,** 354–360 (2013).
5. Burke, R. D. *Hybrid Web Recommender Systems* in *The Adaptive Web* (2007).
6. Parliament, E. & the Council of the European Union. *REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL* https://eur-lex.europa.eu/legal-content/EN/TXT/?qid=1573120061584&uri=CELEX:32016R0679. 2016.
7. Dabrowski, A., Merzdovnik, G., Ullrich, J., Sendera, G. & Weippl, E. *Measuring Cookies and Web Privacy in a Post-GDPR World* in *International Conference on Passive and Active Network Measurement* (2019), 258–270.
8. Li, Q. & Kim, B. M. *Clustering approach for hybrid recommender system* in *Proceedings IEEE/WIC International Conference on Web Intelligence (WI 2003)* (2003), 33–38.
9. Wolpert, D. H., Macready, W. G., *et al.* No free lunch theorems for optimization. *IEEE transactions on evolutionary computation* **1,** 67–82 (1997).
10. *Understanding Generative Adversarial Networks (GANs)* https://blogs.mathworks.com/loren/2015/08/04/artificial-neural-networks-for-beginners/.
11. Maas, A. L., Hannun, A. Y. & Ng, A. Y. *Rectifier nonlinearities improve neural network acoustic models* in *Proc. icml* **30** (2013), 3.
12. *Logistic Regression with a Neural Network Mindset* https://edorado93.github.io/2018/09/07/Logistic-Regression-with-a-Neural-Networks-Mindset-9b5526c2ed46/.
13. Bhatt, S. *5 Things You Need to Know about Reinforcement Learning* https://kdnuggets.com/2018/03/5-things-reinforcement-learning.html.
14. Mnih, V. *et al.* Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).
15. Gittins, J. C. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society: Series B (Methodological)* **41,** 148–164 (1979).
16. Auer, P., Cesa-Bianchi, N. & Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Machine learning* **47,** 235–256 (2002).
17. Li, L., Chu, W., Langford, J. & Schapire, R. E. *A contextual-bandit approach to personalized news article recommendation* in *Proceedings of the 19th international conference on World wide web* (2010), 661–670.
18. Wang, L., Wang, C., Wang, K. & He, X. *Biucb: A contextual bandit algorithm for cold-start and diversified recommendation* in *2017 IEEE International Conference on Big Knowledge (ICBK)* (2017), 248–253.
19. Zheng, G. *et al. DRN: A deep reinforcement learning framework for news recommendation* in *Proceedings of the 2018 World Wide Web Conference* (2018), 167–176.

20. Goodfellow, I. *et al. Generative adversarial nets* in *Advances in neural information processing systems* (2014), 2672–2680.

21. *Understanding Generative Adversarial Networks (GANs)* https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29.

22. Wang, J. *et al. Irgan: A minimax game for unifying generative and discriminative information retrieval models* in *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval* (2017), 515–524.

23. Jain, M. *et al.* Proximal Policy Optimization for Improved Convergence in IRGAN. *arXiv preprint arXiv:1910.00352* (2019).

24. Liu, H., Chandrasekar, V. & Xu, G. An adaptive neural network scheme for radar rainfall estimation from WSR-88D observations. *Journal of Applied Meteorology* **40,** 2038–2050 (2001).

25. Huszár, F. *How (not) to Train your Generative Model: Scheduled Sampling, Likelihood, Adversary?* 2015. arXiv: 1511.05101 [stat.ML].

26. Gehring, J., Auli, M., Grangier, D., Yarats, D. & Dauphin, Y. N. *Convolutional Sequence to Sequence Learning* 2017. arXiv: 1705.03122 [cs.CL].

27. Yu, L., Zhang, W., Wang, J. & Yu, Y. *Seqgan: Sequence generative adversarial nets with policy gradient* in *Thirty-First AAAI Conference on Artificial Intelligence* (2017).

28. *Using an Azure HDInsight Hadoop Cluster on a 1 TB dataset* https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/hive-criteo-walkthrough.

29. *GitHub repository for TensorFlow* https://github.com/tensorflow/tensorflow.

30. Koren, Y., Bell, R. & Volinsky, C. Matrix factorization techniques for recommender systems. *Computer,* 30–37 (2009).

31. LantaoYu. *GitHub repository for SeqGAN* https://github.com/LantaoYu/SeqGAN.

32. Krizhevsky, A., Sutskever, I. & Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* **60,** 84–90 (2012).

33. Papineni, K., Roukos, S., Ward, T. & Zhu, W.-J. *BLEU: a method for automatic evaluation of machine translation* in *Proceedings of the 40th annual meeting on association for computational linguistics* (2002), 311–318.

34. Tevet, G., Habib, G., Shwartz, V. & Berant, J. Evaluating text gans as language models. *arXiv preprint arXiv:1810.12686* (2018).

35. Ulyanov, D., Vedaldi, A. & Lempitsky, V. *Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis* in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), 6924–6932.

36. Tuan, Y.-L. & Lee, H.-Y. Improving conditional sequence generative adversarial networks by stepwise evaluation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* **27,** 788–798 (2019).