Cardiff University

Discovering User Intent In E-commerce Clickstreams

by

Humphrey Sheil

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

in the
College of Physical Sciences & Engineering
Cardiff School of Computer Science & Informatics

February 2019

Declaration

This work has not been submitted in substance for any other degree or award at this or any other university or place of learning, nor is being submitted concurrently in candidature for any degree or other award.

Signed ..... (candidate) Date .....

STATEMENT 1

This thesis is being submitted in partial fulfilment of the requirements for the degree of PhD.

Signed ..... (candidate) Date .....

STATEMENT 2

This thesis is the result of my own independent work / investigation, except where otherwise stated, and the thesis has not been edited by a third party beyond what is permitted by Cardiff University's Policy on the Use of Third Party Editors by Research Degree Students. Other sources are acknowledged by explicit references. The views expressed are my own.

Signed ..... (candidate) Date .....

STATEMENT 3

I hereby give consent for my thesis, if accepted, to be available online in the University's Open Access repository and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ..... (candidate) Date .....

Cardiff University

Abstract

College of Physical Sciences & Engineering
Cardiff School of Computer Science & Informatics

Doctor of Philosophy

by Humphrey Sheil

E-commerce has revolutionised how we browse and purchase products and services globally. However, with revolution comes disruption as retailers and users struggle to keep up with the pace of change. Retailers are increasingly using a varied number of machine learning techniques in areas such as information retrieval, user interface design, product catalogue curation and sentiment analysis, all of which must operate at scale and in near real-time.

Understanding user purchase intent is important for a number of reasons. Buyers typically represent $<5\%$ of all e-commerce users, but contribute virtually all of the retailer profit. Merchants can cost-effectively target measures such as discounting, special offers or enhanced advertising at a buyer cohort - something that would be cost prohibitive if applied to all users. We used supervised classic machine learning and deep learning models to infer user purchase intent from their clickstreams.

Our contribution is three-fold: first we conducted a detailed analysis of explicit features showing that four broad feature classes enable a classic model to infer user intent. Second, we constructed a deep learning model which recovers over 98% of the predictive power of a state-of-the-art approach. Last, we show that a standard word language deep model is not optimal for e-commerce clickstream analysis and propose a combined sampling and hidden state management strategy to improve the performance of deep models in the e-commerce domain.

We also propose future work in order to build on the results obtained.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

E-commerce is transforming every avenue of our online lives [8]. Transacting over $2.3 trillion globally in 2017 [9] and growing at an average rate of 10% annually globally, consumers are increasingly purchasing all types of goods and services online. By contrast, many retailers are reporting year-on-year declines in older, physical distribution channels [8]. The migration is only beginning, with just 3% of global commerce being conducted online [10]. Since 1995, the capabilities of online retailers have been driven by parallel advances in all fields of computer science and electronics: information retrieval, networking, faster and more scalable hardware have all contributed to the online explosion. More recently, statistical and machine learning on large datasets has come to the fore, enabling users to discover content using recommender systems and search functions to become more relevant. Machine learning holds enormous potential to further enhance the next wave of e-commerce systems.

1.1 What is E-commerce?

E-commerce is the purchasing of goods and services - both physical and virtual - online. In its earliest form, merchants focused on low-risk items that users were willing to purchase without seeing them first - books and CDs. From there, online retailing expanded to include insurance, leisure and business travel, automotive and property. Today, anything can be purchased online, from fresh groceries to clothing. For merchants, the attraction of online is reduced cost of distribution - fewer physical stores and sales staff are needed. For users, e-commerce provides convenience, the ability to compare prices and features easily and the flexibility to shop anytime and from any location with a connection to the internet. Increasingly, machine learning is used to help users to pro-actively find what they need.

1.2 Evolution of E-commerce

In our opinion, the progression of e-commerce can be divided into five main phases:

1. Infrastructure build out - the creation of the basic software and hardware building blocks enabling e-commerce (1996 - 1999).

2. Content discovery and search - providing simple search interfaces which exposed product inventory to users (2000 - 2004).

3. Improved discovery and search - rapid innovation in user interface design and information retrieval techniques to improve the overall user experience (2005 - 2009).

4. Mobile enablement - supporting the browsing and purchasing of content on small-screen devices (2010 - 2015).

5. Enhanced functionality using machine learning - modifying system behaviour to reflect user needs using individual and group user implicit feedback (2015 - now).

The first phase of e-commerce focused on enablement and infrastructure build-out. Retailers invested in basic systems to sell inventory online - relational databases, inventory management systems and websites - creating *supply*. Concurrently, datacentres, networks and ISPs provided connectivity to entire populations to create a complementary *demand*. Capacity was grown in parcel delivery services and business processes adapted to cater for the selling of goods online.

The second phase of e-commerce switched focus to providing content discovery tools to users, along with better user interfaces. Retailers deployed early versions of systems to guide users to products they are more likely to purchase and to the most popular content - the Amazon "users who viewed this item also purchased.." feature [11] is a good example of this. In this phase, most e-commerce systems used SQL to query rigidly defined database schemas and thus were unable to apply information retrieval techniques such as gazettes or fuzzy keyword matching to provide a more natural search interface.

In the third phase, building a better search capability became the focus, resulting in search engines which provide better search results to users, even when nondescript terms and phrases are provided. In this phase, a significant overlap developed between traditional search / information retrieval and recommender systems [12].

In the fourth phase, mobile device enablement became a key focus, with retailers deploying both HTML and native shopping applications for users. The challenge with mobile e-commerce is to retain functionality that users expect, when operating on a constricted budget - screen size, processing power and network access. Users actively prefer to browse on their mobile device for some e-commerce segments, particularly where they do not need a large screen and / or have purchased the same item before.

We are currently in the fifth phase of e-commerce - continuing to employ machine learning to enhance the experience of shopping online, through improvements to recommendations, aids to navigation, personalised user interfaces and much more. *Machine Learning* describes software which learns to improve its performance on a given task over time by minimising an error metric or maximising a reward.

It is important to note that although we have (albeit artificially) time boxed the phases here, it is entirely possible for some merchants to be far behind the capabilities of others. In fact this is often the case. Leading retailers like Amazon, Netflix, eBay and Microsoft have superior capabilities to smaller retailers due to their size, access to development teams and massive investment in IT and machine learning infrastructure.

Figure 1.1 illustrates elements of this machine learning focus on the user. Individual panels or tiles are populated by machine learning modules such as recommender systems or simple statistical calculations such as popular products with a high conversion rate or a time limited special offer.

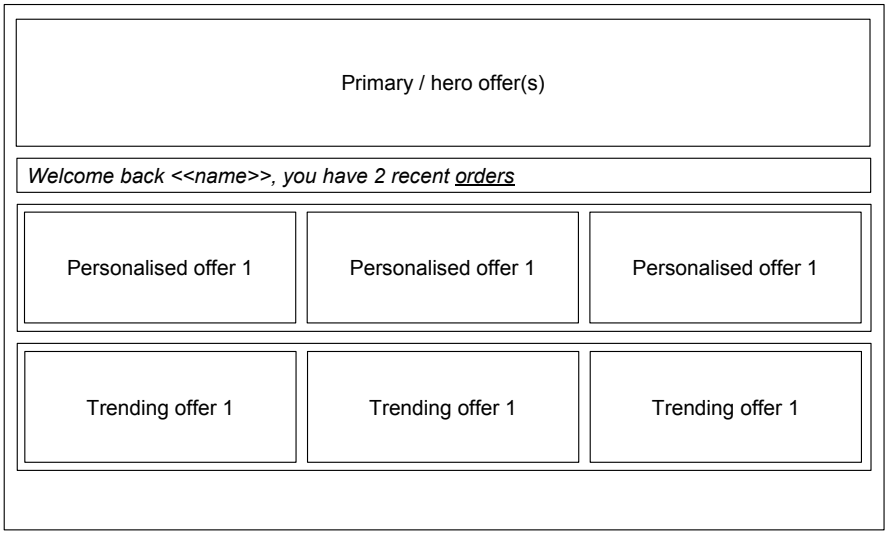

FIGURE 1.1: Typical organisation for a main or homepage of an e-commerce website.

In this thesis, our focus is on the fifth phase - the intersection of e-commerce with machine learning to improve outcomes for all participants, both users and merchants. There are many ways that machine learning can be used to improve the lot of users, including:

1. Recommender systems - helping users to discover new content based on their preferences (both implicit and explicit), similarity to other users and content popularity.

2. Price / product alerts - helping users to change the timing of their purchases to minimise cost or maximise value, or to know when a scarce item is in stock and should be purchased.

3. My account functionality (saving preferences, important dates etc. and then proactively interacting with users to advise on timely purchases etc.).

1.3 Current E-commerce Landscape

The advent of e-commerce utterly transformed traditional commerce, and remains disruptive at its core today - creating new winners and losers as user expectations change rapidly. The newspaper industry is a good example, with a 50% decline in UK newspaper sales since 2005 [13]. People now expect to get their news for free and delivered to their mobile device on the move - not in a newspaper that they must purchase or have delivered to one physical location.

Another disruptive e-commerce theme across multiple retail domains is the comparison website - products that are mostly fungible can be aggregated and compared, enabling users to save significantly in areas such as insurance, travel, energy supply, mobile phone contracts and broadband internet [7]. Consumers benefit from this price transparency, and where products are truly fungible and interchangeable e-commerce has created a more efficient marketplace, but for products that are not fungible or where post-sales service can be skimped on (or descriptions are less than truthful), the consumer can regret simply purchasing the cheapest option, while more suitable merchants miss out on sales.

The structure of e-commerce has changed significantly since its inception. Control of distribution or access to users has coalesced into a small number of portals - for example Google Shopping, eBay, Amazon and Taobao [14]. Only medium and large retailers can afford the significant, ongoing investment in e-commerce systems and all merchants regardless of size, pay very significant marketing and advertising costs to online partners via programs such as Google AdWords, Bing Ads, Facebook Ads and

Twitter Ads. In 2010 [15], an important ruling was passed allowing merchants to bid for display when competitor brand names, terms and keywords are searched on portals such as Google. The positive impact of this is that any merchant (as long as they have sufficient marketing budget) can compete with an established merchant with even the strongest brand equity and recognition. The negative impact is that overall advertising costs rise, as more companies compete when bidding for a constrained resource [16, 17].

Large portals face multiple challenges - content discovery, information retrieval and search results ranking chief among them. Continual advances in information retrieval have helped, but user sessions are often short both in time and number of interactions, meaning that portals want to present relevant results quickly - in some cases even before the user has provided any input. Portals increasingly use recommender systems to present the best (most relevant) mix of content to users.

These portals can use recommender systems with little downside - but for the merchant who has no product reviews or ratings, or is not the cheapest, recommender systems can have a catastrophic effect on trading and profit if they fall foul of the ranking algorithm [12]. This can occur in two main ways:

1. Popularity bias - the merchant is new or is selling a new product which has no user reviews or interactions - in this case the new entity is said to suffer from the cold start problem. Although well-understood and mitigating solutions exist, it still occurs in practice and penalises new merchants and products.

2. Filter bubbles or lack of diversity - where users are only recommended items or products related to their previous activity. Even popular items which are not related to previous activity can suffer under this regime.

The current e-commerce landscape then, is one dominated by extremely large portals, who use a combination of traditional information retrieval techniques and recommender systems to help users find what they are looking for.

1.4 What's Missing?

We argue that the online merchants are more often than not ignored by the field of Computer Science / Software Engineering. As an example, the large research field of recommender systems is focused on providing end-user utility (for example, suggesting the best movie to rent or the book we think you will like best) - but not merchant utility (here is the best customer for you to spend advertising on, here is the optimal price to set for this product). A large amount of work has been conducted [3, 12, 18] to agree the best metrics to measure user satisfaction (e.g. dwelltime per item, Mean Squared Error (MSE), Discounted Cumulative Gain or DCG), but little work has been conducted to measure merchant or retailer satisfaction.

It can be argued that a satisfied user implies a satisfied merchant but we rebut this. A happy user is a user who gets the best value deal - their goal is set directly against the goal for the merchant, which is to maximise their profit.

Our focus on the merchant or retailer rather than the end-user is not altruistic, one-sided or short-sighted. Retailers pay for the e-commerce internet - not end-users and not the portals. The vast majority of Google's revenues for example come from merchants who advertise with Google. If merchants cannot compete or pay too much for advertising and distribution, then they will go out of business, and competition will reduce, ultimately resulting in a less efficient marketplace. Consumers will pay more and receive less in return.

A healthy e-commerce ecosystem requires three sets of actors:

1. Portals / aggregators / agents of discovery.
2. Consumers who use these portals or visit retailers directly.
3. Retailers / merchants who sell or re-sell goods and services to consumers.

Figure 1.2 shows the interplay between the three sets of actors. Consumers can visit merchant websites directly, but increasingly discover merchants and products via portals and either purchase directly on that portal (who then claims a distribution or finders fee from the merchant), or then connect to a merchant. The growth of portals represents an opportunity and a threat to merchants. While they gain access to larger audiences of potential customers, they must also pay handsomely for this access. To underline our earlier point that merchants rely on portals, recently the European Union has opened an investigation into unfair practices against retailers by one of the most dominant e-commerce platforms, Amazon [19].

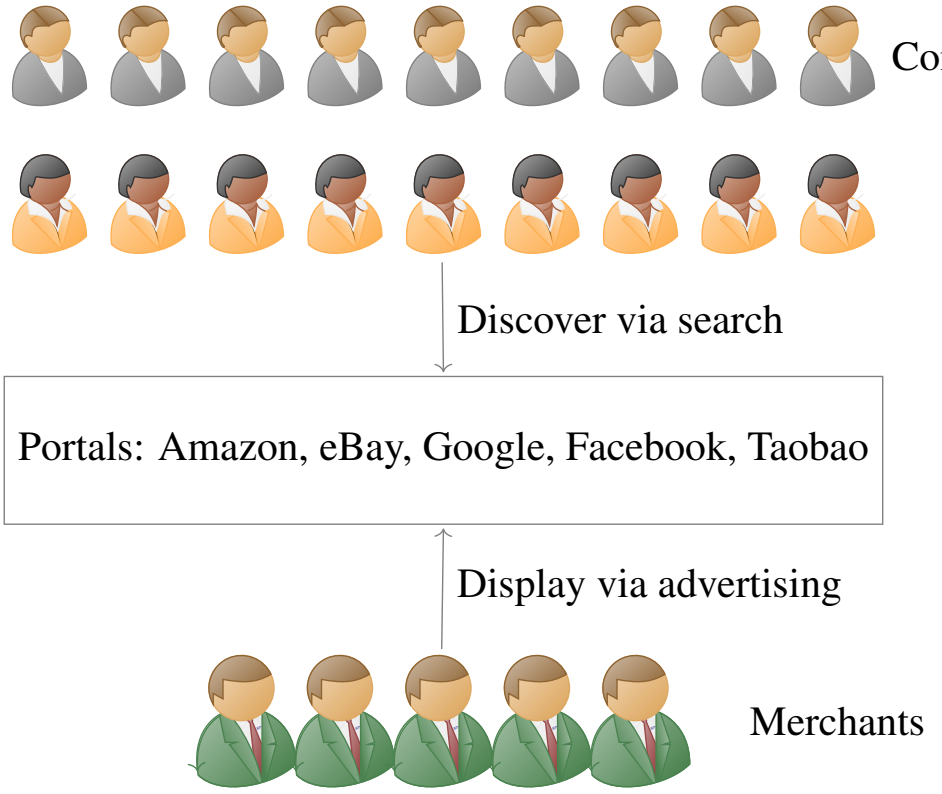

FIGURE 1.2: The primary actors in an e-commerce system and how they interact with each other.

1.5 Practical Impact of Machine Learning for Merchants

As we will see in Chapter 2, there are many different ways in which machine learning can be employed in the e-commerce ecosystem. Therefore we should consider where best to apply machine learning in order to address a pressing need that can be reasonably implemented. Commercially, merchants care deeply about their cost of advertising. Large portal owners such as Amazon, eBay, Google, Yahoo, Facebook et al. all already operate closed advertising systems which they control [16, 17]. However, directly running optimisation experiments in these environments would be prohibitively expensive, even though we acknowledge that merchants spend an inordinate amount of their operational budget on advertising. Even the best machine learning model will struggle to compete in a closed and opaque system such as Google AdWords. Despite protestations

to the contrary, AdWords is naturally run for the benefit of Google, not merchants, and currently contributes over 85% of Google revenue ($26.6 billion of the $31.1 billion revenue for Q1 2018) [20].

However, there is one task which can provide valuable input into the following merchant challenges:

1. Improve bidding strategy (timing, amounts, semantics) on advertising - the largest single area of operational spend.

2. Improve stocking and product catalogue management - ensuring that popular items are always in stock.

3. Improve pricing strategy - maximising profit margins where user demand supports the price and reducing price to grow volume where user demand is weak.

That task is discovering user intent. In information retrieval, *intent* signifies what information the user is searching for based on their queries. Queries can be further segmented into three main types - informational, navigational and transactional [18].

1.5.1 Problem Statement

In an e-commerce clickstream setting however, we propose a more focused definition in Equation 1.1:

$$\forall s_i \in S, prob(type(e_i) = e^b) = f(e_0, .., e_n) \tag{1.1}$$

where:

S is the set of all user sessions

s is a user session containing n events $\{e_0 \dots e_n\}$

e_i is a tuple comprised of $\{timestamp, itemid, itemcat\}$

T is the set of event types $\{t^c \dots t^b\}$, with t^c and t^b representing the click and buy event

$\forall e \in s \in S_{test}, type(e_i) = e^c$

f is the model implementation (with multiple candidates)

The user intent task is tractable and attractive for a number of reasons:

1. User activity datasets (e-commerce clickstreams) are readily available.
2. Accuracy of user intent prediction is measurable to help refine models over time.
3. Near real-time prediction of user intent represents high-value actionable business intelligence that merchants can use to only spend time, money and resources on users who are likely to purchase.

1.6 Proposition

Our proposition is that machine learning can be used in the e-commerce domain for the direct and measurable benefit of the *merchant*, and not just the end-user. Merchant benefit or utility can be provided in multiple ways - predicting which users are more likely

to purchase than others, assisting the merchant in pricing goods to optimise revenue and profit. Additionally, we can measure the impact of our proposed merchant benefit directly, rather than use consumer utility or satisfaction as a proxy measurement.

In order to test our theory, e-commerce datasets containing user activity are required. Then we will either need to construct good features to correctly classify user intent or use a technique which can construct a good representation by itself. Our goal is to use data which is readily accessible and does not violate any privacy legislation or require private data from end-users. In recent years, research has been conducted on incorporating multiple signals from social networks into ML e-commerce models [21] but this approach is increasingly frowned upon and difficult to implement. Users do not want to share their private data and merchants do not want the overhead of storing and processing it in a GDPR-compliant (General Data Protection Regulation) manner.

Let us consider a counter-argument to make the previous point clearer and illustrate the trade-off between persuasion and transparency [22]. Imagine that we constructed a model M_0 which required the user to log in with their Facebook, Twitter or Instagram credentials before a search query could be executed in order to provide more relevant search results (the M_0 model uses features which leverage a user's friend graph and other private items). Most users would baulk at these requests, even if they believed that the search results would be better than those provided by a model M_1, which simply used current, anonymous session interactions.

Lastly, we favour approaches which are straightforward to implement and can provide near real-time decision support to merchants.

1.7 Open Questions

There are three questions that we aim to answer in this thesis:

1. In the user intent discovery task, what is the performance difference between classic machine learning techniques using explicit features when compared to deep learning models using a learned representation?

2. What are good representations for e-commerce concepts such as product or user that can be learned by an appropriate machine learning technique?

3. Can deep neural networks (DNNs) transfer from their traditional applications of computer vision and natural language understanding to the user intent discovery task without significant modifications?

1.8 Contribution

Our contribution to these questions is as follows:

1. We conducted a detailed analysis of explicit features showing that four broad feature classes enable a classic model to infer user intent.

2. We constructed a deep learning model which recovers over 98% of the predictive power of a state-of-the-art approach.

3. We show that a standard word language deep model is not optimal for e-commerce clickstream analysis and propose a combined sampling and hidden state management strategy to improve the performance of deep models for the task of user intent detection in the e-commerce domain.

The work conducted in this thesis was published in the following papers:

1. [23]: Classifying and Recommending Using Gradient Boosted Machines and Vector Space Models, Humphrey Sheil and Omer Rana, Advances in Computational Intelligence Systems. UKCI 2017.

2. [24]: Predicting purchasing intent: Automatic Feature Learning using Recurrent Neural Networks, Humphrey Sheil and Omer Rana and Ronan Reilly, ECOM workshop at ACM SIGIR 2018.

3. [25]: Understanding E-commerce Clickstreams: a Tale of Two States, Humphrey Sheil and Omer Rana and Ronan Reilly, Deep Learning Day workshop at ACM SIGKDD 2018.

1.9 Thesis Roadmap

The following chapters are organised as follows:

- Chapter 2 reviews the current landscape of e-commerce and machine learning to set the scene for our contributary work.

- Chapter 3 sets out the system architecture used to construct, train and benchmark the classic and deep models used in this work.
- Chapter 4 provides a detailed analysis of a particular classic model, Gradient Boosted Machines (GBM), and the importance of different feature classes in the e-commerce domain.
- Chapter 5 constructs and benchmarks a competing deep learning model with further sections focused on model interpretability and model comparison.
- Chapter 6 explores how standard deep word-based language models must be modified to function effectively in the e-commerce domain for the user intent discovery task.
- Chapter 7 reviews our hyperparameter tuning efforts.
- Chapter 8 provides a critical assessment of our work and places it in context to existing work.
- Chapter 9 previews suggested future work and summarises the thesis contribution.

1.10 Summary

E-commerce is a domain where machine learning has already been applied successfully, but much remains to be done - in particular, serving the needs of merchants more fairly to ensure a healthy e-commerce ecosystem.

In addition, the selection of a target e-commerce problem in practice also depends on the availability of good data, the opportunity to directly impact and measure the performance of the machine learning model and ideally, a computationally fast model which can generate outcomes in near real-time to be effective in an e-commerce context. In the next chapter, we will set out a short overview of machine learning and how it is currently utilised in the the e-commerce domain.

Chapter 2

Background and Related Work

In our opinion, the intersection of e-commerce and computer science is an area of active research for a number of reasons:

1. Clear, significant commercial interest - e-commerce is a real world problem with immediate applicability.
2. The ready availability of datasets suitable for analysis - from user activity logs to product catalogues.
3. A well-defined mechanism (A/B testing) to test new implementations and theories in an online and offline setting.

The diagram below illustrates the broad anatomy of an e-commerce website from a user / functional perspective. We use this diagram to ground our further discussion of how machine learning is currently used across this structure.

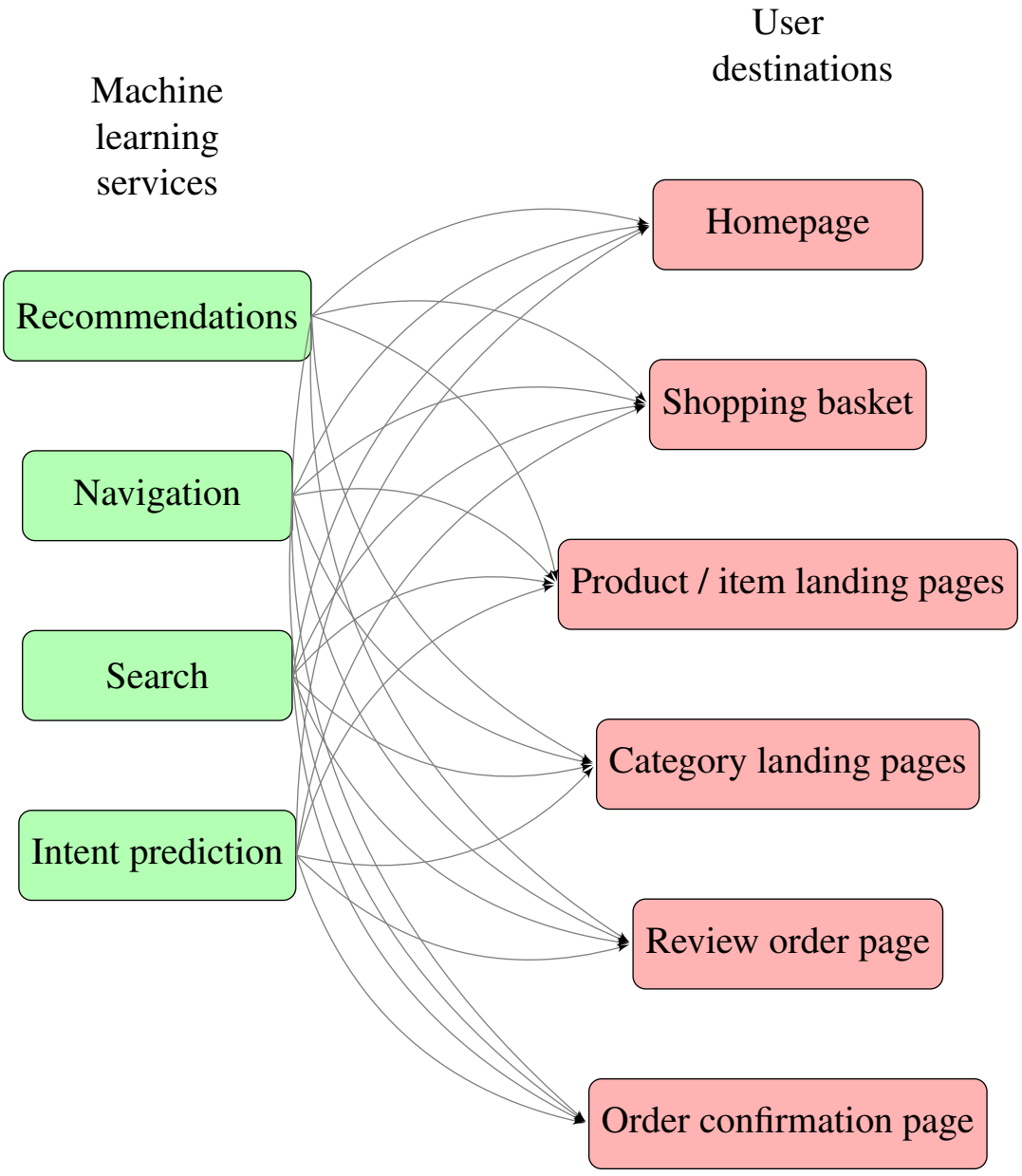

FIGURE 2.1: Broad anatomy of an e-commerce website from an end-user perspective, providing a reference map for future discussions on how ML is applied to e-commerce.

2.1 Background

In the following sections, we provide an overview of machine learning, in order to ground our overview of how it is used in e-commerce currently, and to inform more detailed discussions in Chapters 3, 4 and 5.

2.2 Machine Learning

Machine Learning precedes the field of e-commerce significantly - and if we include statistics as a natural complement / pre-cursor to machine learning, the joint fields are older still.

E-commerce is an ideal domain in which to train and deploy machine learning models since good datasets are common and relatively easy to gather, while tasks are often automated. For example, it is the norm for user sessions to be modelled as a tuple of $\{userid - eventtype - itemid\}$ and for these tuples to be grouped into user sessions based on a maximum time window (for example 30 minutes). Web server logs are regularly processed to extract these datasets and these datasets can then be fed into multiple machine learning model candidates.

In Chapter 1 we stated that machine learning (ML) represented a class of software which can learn to improve its own performance from past experience. Figure 2.2 illustrates the primary or basic components of any machine learning system.

A minimal ML system consists of [26]:

1. A dataset - the input dataset with either output target labels (desired outputs) or target mappings.

2. A model - the abstraction of the problem to be solved, or the mathematical structure by which the prediction y_i is made from the input x_i.

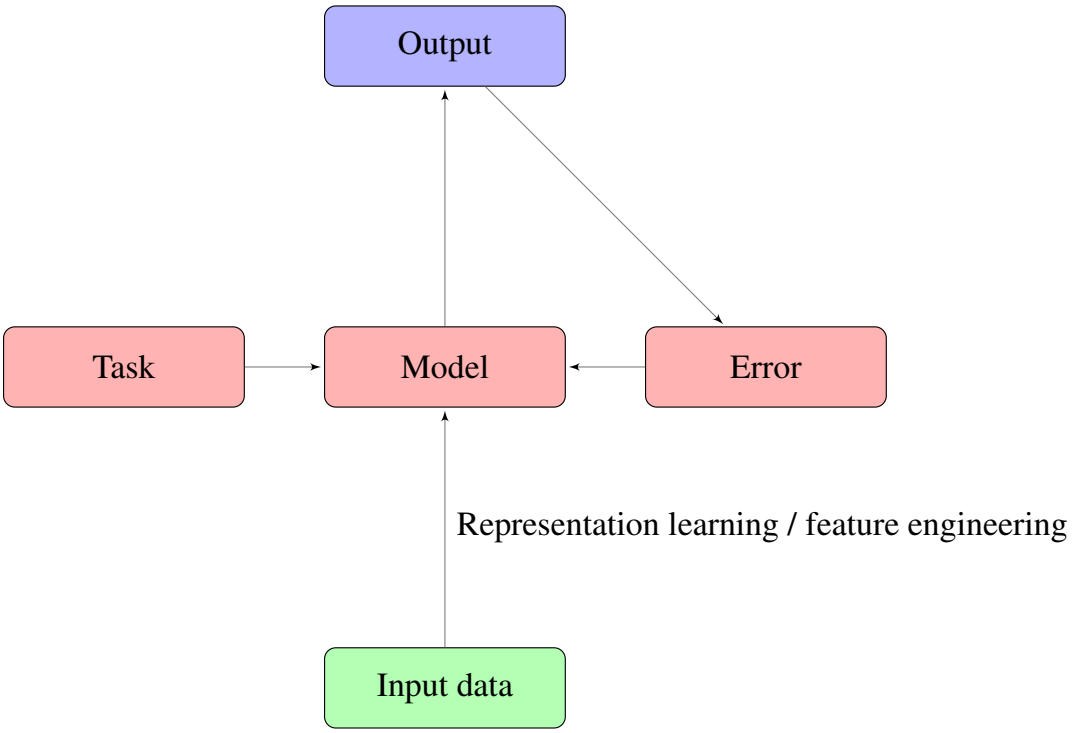

FIGURE 2.2: The primary components of a supervised machine learning system where we have labelled data and thus can calculate an error as the difference between the expected and actual output from the model.

3. A loss or error function which measures the difference between the actual model outputs and desired outputs (we assume that our data is labelled and thus we are training in a supervised setting).

4. A training or optimisation algorithm - using the loss function to modify the model in some way so that future iterations provide better outputs for the task at hand.

Each of these items may be considered as a pluggable component where the implementation can be replaced to best fit specific needs (simplicity, speed, scalability, interpretability).

2.2.1 Datasets

The input data, combined with the task we wish our model to learn, affects the selection of all other components in the ML system. Data can be virtually anything - from images to video, audio, handwriting and so on. For the scope of this thesis however, the datasets under consideration are e-commerce-related clickstreams, and thus are overwhelmingly structured text. In addition, we can make the following generalisations about e-commerce clickstreams:

1. They are large - typically with millions of entries or samples to consider.
2. They are heavily imbalanced - the class label or target we are interested in (buyers, fraudsters) is far smaller than the dominant label (clickers, non-fraudsters).
3. Time plays an important role - users spend time dwelling on items, user sessions have a length measured in events and time, and the entire dataset can often be considered and processed as a time series.

2.2.2 Models

There are many types of model implementations available. While there is no single agreed taxonomy to organise all of the model families, the following categories are proposed in [26]:

1. Linear models.
2. Tree models.

3. Distance-based models.

4. Probabilistic models.

In the following sections, we focus on the subset of models relevant to the user intent discovery task.

2.2.3 kNN - k Nearest Neighbours

kNN is an example of a distance-based model. It is simple to train - the model iterates over and memorises the training data. Then at inference time a certain number - $k \geq 1$ - of the points closest to the input data are used to vote and predict the majority class. A variety of distance metrics (e.g. euclidean distance, Jaccard index, cosine similarity) are used depending on the input data to create clusters of similar exemplars.

The training data consists of feature vectors x_i and label pairs y_i for each example $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ and training time is fast - $O(n)$ while inference performance at run-time is slow - also $O(n)$, since all examples need to be compared to the input x_i to find the closest k matches. kNN models have high variance since the decision regions of the model are created directly from the training data.

2.2.4 Gradient Boosted Machines / Decision Trees (GBM / GBDT)

Tree models are very popular in Machine Learning - high quality implementations exist and are readily usable [27–29]. They are fast to train and considered more interpretable

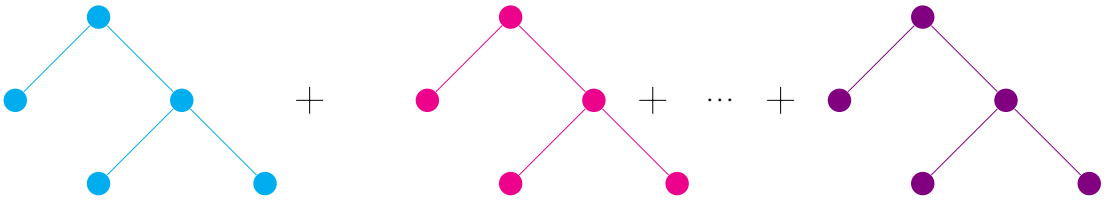

FIGURE 2.3: In GBDT, each tree is a weak learner, and when combined these separate trees form a single strong learner.

than other black box approaches (although complex trees or collections of trees are certainly more difficult to understand). Decision trees used in data mining are of two main types:

1. Classification tree analysis - used when the desired outcome is to predict the class to which the data belongs.

2. Regression tree analysis - used when the predicted outcome can be considered a real number (e.g. the price of a house, or a patient's length of stay in a hospital).

The term Classification And Regression Tree (CART) analysis is an umbrella term used to refer to both of the above procedures [26]. As their name suggests, trees are made up of nodes which represent *tests of supplied features* (for example, test $if dwell\_time > 100$) and edges which control the path to be followed based on the outcome of these tests.

Trees used for regression and trees used for classification have some similarities but also some differences, such as the procedure used to determine where to split. Ensemble methods construct more than one decision tree: boosted trees incrementally build an ensemble by training each new instance to emphasise the training instances previously misclassified.

Gradient Boosted Decision Trees (GBDT) do exactly this and attempt to iteratively minimise the *residual error*, i.e. the error remaining after the most recent tree (weak learner) has been added to the current ensemble. Figure 2.3 illustrates the concept of additive trees which combine to form a single strong learner. We use colours to illustrate that the individual trees constructed vary by structure and features used, and are combined to minimise the overall error.

Different implementations of GBDT [27–29] use different heuristics and optimisations to build trees (for example handling categorical variables differently, adding a regularisation term and / or favouring simpler trees to guard against overfitting) but all implementations share a common goal: at each iteration, minimise the remaining residual error. Regardless of the implementation chosen, the performance of the trained model is heavily reliant on the quality of the supplied feature set.

GBM / GBDT hold multiple state of the art benchmarks in e-commerce and we investigate the performance of GBM in detail on the user intent discovery task in Chapter 4.

2.2.5 Markov Chains / Hidden Markov Models

Markov chains and their derivative - hidden Markov models, are both examples of probabilistic machine learning models. A Markov chain is a model that represents the probabilities of sequences of random variables. The central assumption in a Markov chain is that all future states of a sequence can be predicted using only the current state, and no

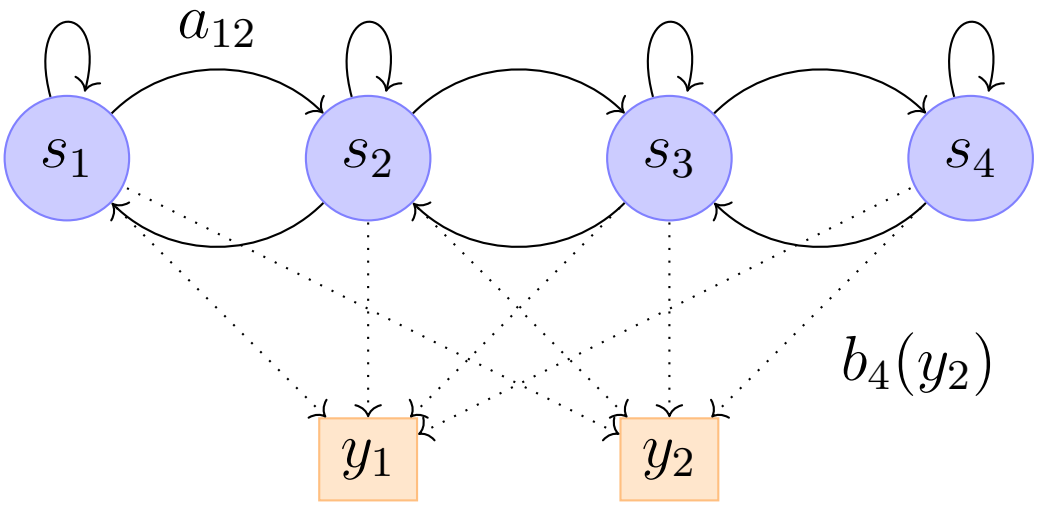

FIGURE 2.4: This example HMM with 4 states can emit 2 discrete symbols y_1 or y_2. a_{ij} is the probability to transition from state s_i to state s_j. $b_j(y_k)$ is the probability to emit symbol y_k in state s_j.

states prior to it. Figure 2.4 illustrates a very simple 4-state HMM. In this constrained HMM, states can only reach themselves or their adjacent (prior and subsequent) state.

A hidden Markov model (HMM) is an extension to the standard Markov chain where some events of interest are hidden or not directly observable. Chapter 8 of [3] (3rd edition, currently under construction) provides standard terminology for Markov chains and models as follows:

$S = s_1, s_2, \ldots, s_n$ A set of N states

$A = a_{11}, a_{12}, \ldots, a_{n1}, \ldots, a_{nn}$ a transition probability matrix A, each a_{ij} representing the probability of moving from state i to state j

$\pi = \pi_1, \pi_2, \ldots, \pi_N$ an initial probability distribution over states. π_i is the probability that the Markov chain will start in state i.

At first glance, HMMs seem like an ideal candidate to construct a model of user purchase intent. Beneath a given complexity threshold, they are intuitively appealing and

interpretable - much like probabilistic graphical models [30]. Moreover, they can accommodate inputs of variable length so can model clickstreams naturally. However, they suffer from some limiting drawbacks [31]:

1. They cannot express dependencies between hidden states - therefore long-range correlations or connections cannot be modelled by a single HMM. In fact, only a small subset of possible sequences can be modelled by a reasonably constrained HMM.

2. HMMs model discrete states, and exclude the possibility of encoding other states.

3. If states *should* depend on multiple states, then the number of states increases rapidly, e.g. N^2 states are required if each state depends on two states.

In direct contrast to HMMs, neural network models (see below) can capture and model continuous states which is a meaningful advantage. Empirically, a specific type of neural network model (Long Short Term Memory which we will cover in detail in Chapter 5) began to outperform HMM in the domain of speech recognition in 2012 [32] and since then in multiple other domains.

In [31], the concept of hybrid HMM - neural network models are proposed to combine the best of both models however it is fair to say that there is still much work to be done in this area. Much later in Chapter 8 we will cover a related concept to HMMs - causal inference - which aims to provide interpretable, more powerful models which can reason using facts and counter-factuals.

2.2.6 Neural Network Models

Artificial Neural networks (ANNs) and in particular Deep Neural Networks (DNNs) are well suited to problems that are non-linear and where the data contains nuances and patterns not readily obvious to a human. Put another way, deep feedforward neural networks are associated with complicated, non-convex objective functions that simpler models cannot solve or approximate.

Artificial Neural Networks are complex, nonlinear and parallel computers composed of very simple computing units or neurons. A neuron receives signals from other neurons, combines them and depending on some threshold and an activation function, will either fire or not. Neurons are organised in layers - input, hidden and output layers. Input layers directly receive input data, output layer values are the final values from the network after all processing has completed. Hidden layers are not strictly speaking hidden - their values can be examined at any point during training or inference, but what they are processing, i.e. the abstractions coded into one or more hidden layers, is always open to interpretation, hence the name.

Figure 2.5 illustrates the canonical view of a simple neural network.

We note that neural networks fell dramatically out of fashion from 1989 - 2009, primarily as a result of overpromising - the so-called (second) AI winter. They came back into vogue from 2010 onward due to the proliferation of large datasets and highly parallel hardware architectures on which to run them.

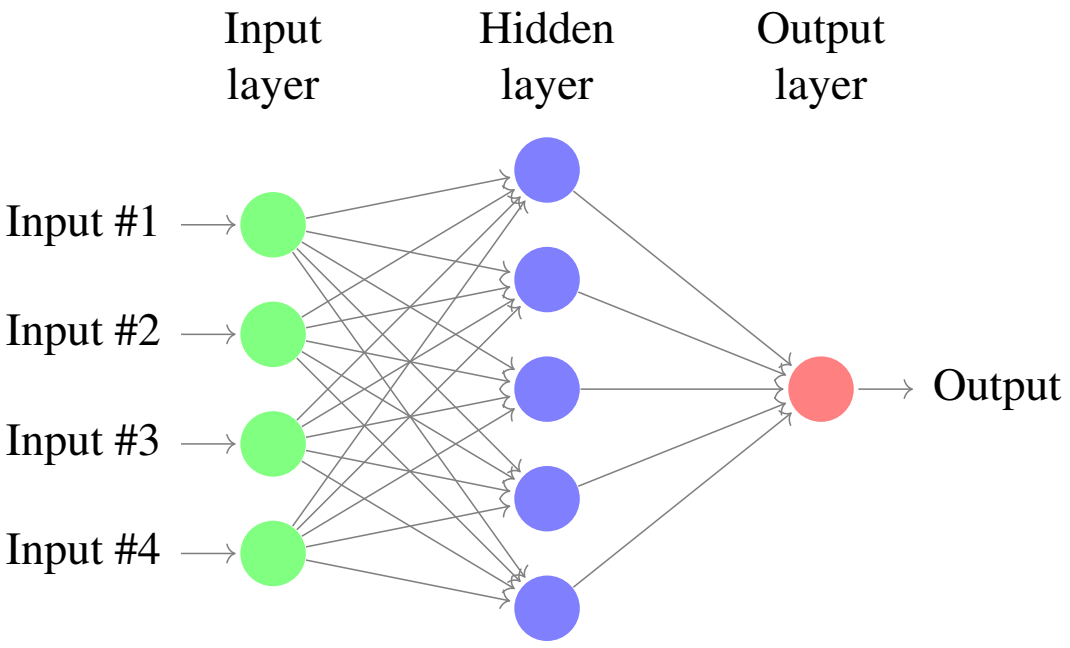

FIGURE 2.5: A simple neural network. Units are divided into three types (input, hidden, output) and organised into layers. In this example the network is fully connected in a feedforward manner.

2.2.6.1 Deep Neural Networks

Since 2010 *deep* neural networks (DNNs) [33, 34] hold state of the art performance on domains such as language translation, handwriting recognition and varied computer vision tasks including sequence to sequence translation [35] (relevant for clickstream analysis). DNNs are closely related to ANNs, and it is largely the joint advent of a hardware architecture (general purpose GPU or GPGPU) and open datasets such as MNIST and ImageNet that enabled DNNs to achieve their performance.

The exact meaning of *deep* in DNN is not defined, but colloquially most researchers take it to mean 10 or more hidden layers. DNN variants such as Highway Networks [36] can have hundreds of layers. Figure 2.2.6.1 shows the deep model from Microsoft Research used to win the 2015 ImageNet competition with 152 layers. The underlying

trend throughout the ImageNet competition was the increase in number of layers used, from GoogLeNet, AlexNet and finally ResNet.

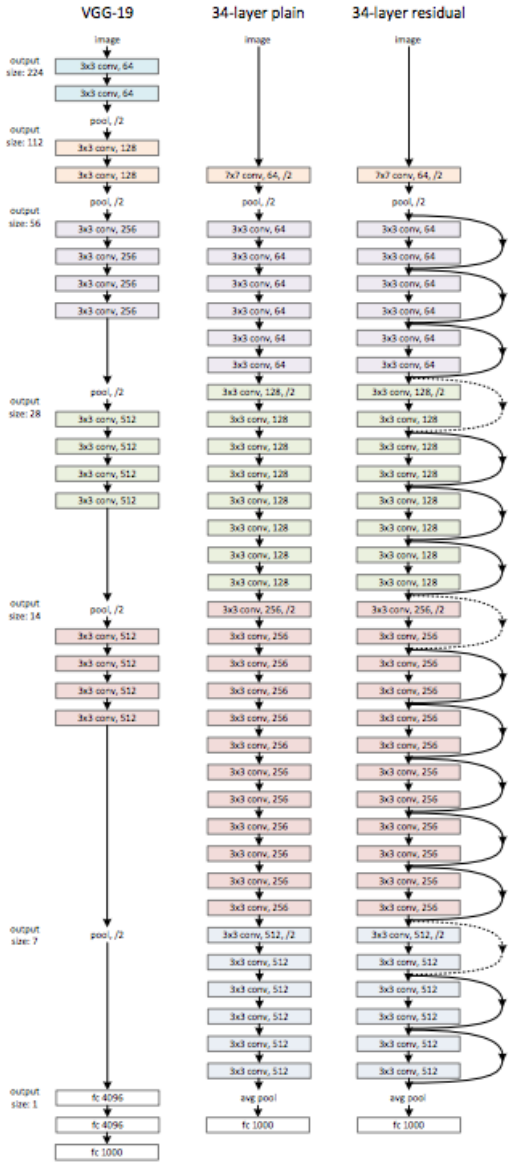

FIGURE 2.6: The winner of the 2015 ImageNet competition from Microsoft Research - ResNet [1], depicted alongside the VGG19 model.

DNNs must be trained in order to work effectively, and it is backpropagation combined with gradient descent that is most often used to train DNNs. In supervised learning back-propagation can be seen as the chain rule applied to the error derivatives from the forward pass through the network [33]. Figure 2.7 illustrates how the forward and

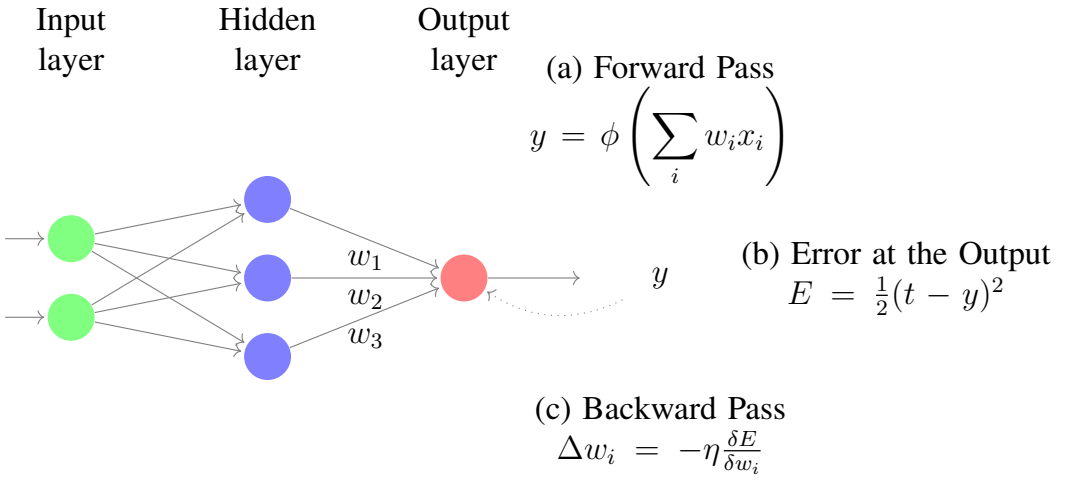

FIGURE 2.7: From [2]. Overview of backpropagation.

reverse passes are combined using back-propagation and stochastic gradient descent to train a network. To start, a training pattern is fed forward, generating corresponding output. Next, the error between actual and desired output is computed. Finally, the error propagates back through the network, through updates where a ratio of the gradient ($\frac{\delta E}{\delta w_i}$) is subtracted from each weight. x_i, w_i, Φ are the inputs, input weights, and activation function of a neuron. Error E is computed from output y and desired output or target t. η is the learning rate.

Due to the large number of parameters, DNNs are particularly susceptible to overfitting, and can create models which are brittle or behave badly when presented with mildly perturbed or unexpected data - for example samples never seen before during training, or samples drawn from a slightly different distribution [37].

A wide range of countermeasures have been designed to address overfitting, including dropout [38] and regularisation [39].

2.2.6.2 Recurrent Neural Networks

Recurrent neural networks [40] (RNNs) are a specialised class of neural networks for processing sequential data. A recurrent network is deep in *time* rather than space and arranges hidden state vectors h_t^l in a two-dimensional grid, where $t = 1 \dots T$ is thought of as time and $l = 1 \dots L$ is the depth. All intermediate vectors h_t^l are computed as a function of h_{t-1}^l and h_t^{l-1}. Through these hidden vectors, each output y at some particular time step t becomes an approximating function of all input vectors up to that time, $x_1, \dots, x_t$ [5]. We will see RNNs in more detail in Chapter 5.

2.2.6.3 Embeddings

Neural networks typically deliver their best performance not when consuming explicit hand-crafted features, but instead learning the best representation through training. In this regime, concepts in the input data (for example items or categories in the e-commerce domain) are modelled as words and each word is then transformed into a low-dimensional distributed representation - the embedding or word vector [41]. A good vector space model will map semantically similar words close together. We will cover embeddings in more detail in Chapter 5.

2.2.7 Loss / Objective Functions

A loss provides a numerical measure of the difference between the desired output of a model and the actual output. Although this may sound obvious, it is a fundamental design decision for any machine learning model. The loss function is also known as the

objective function and this name gives a truer account of its purpose - it is measuring the ability of a model to carry out the task asked of it. In other words, it is framing the learning component.

Intuitively, we want outputs that are more different to be penalised more, and actual outputs that are closer to desired outputs to be penalised less. The loss is then used to update the model in some way by the training algorithm - the essential part of machine learning.

2.2.7.1 Categorical Cross-entropy

The categorical (also known as multinoulli) distribution is a K class generalisation of the two class Bernoulli [42]. Generally the prediction is a vector of probabilities for each class, so the target y_t is a class in the one-hot representation as a vector of length $K, k = 0 \dots K$, where K is the number of classes. We will see this particular loss in more detail in Chapter 5.

2.2.7.2 Auxiliary Losses

In deep learning, a useful augmentation to a primary loss is to augment it with another associated loss - the so-called auxiliary loss. The immediate effect of this is to increase the size of the loss and thus the size of the gradients and in some cases this can ameliorate the vanishing gradient problem, especially in a sequence processing setting with recurrent neural networks (see Chapters 4 and 5).

2.2.8 Training and Optimisation

The process of training an ML model involves providing an ML algorithm (that is, the learning algorithm) with training data to learn from. The term ML model refers to the model artifact that is created by the training process. Some models are created before training, in particular deep learning models, while some training algorithms create the model architecture from the training data e.g. Gradient Boosted Machines (GBM).

Different regimes of machine learning are clearly discernible - supervised, semi-supervised and unsupervised, but in all cases the goal of training a machine learning model is to provide the model with example data so that it learns to *minimise* some error metric (or maximise some reward if reinforcement learning is used) and then to *generalise* to unseen data.

In a supervised setting where labels are known, training is most often implemented as gradient descent, for example Stochastic Gradient Descent or SGD. Various enhancements have been proposed for gradient descent, including Nesterov momentum etc.

Although it may seem obvious that the goal of gradient descent is to find a global minimum in the gradient landscape, i.e. co-ordinates where the overall error is lowest, in practice this is not the case - in fact a global minimum would represent significant overfitting on the training set, preventing good generalisation to *unseen* data and the overtrained model would perform poorly in practice. Techniques such as dropout [38, 43], L1 and L2 regularisation have also been developed to guard against overfitting. In fact, [39] lists over 50 ways in which a model can be regularised.

2.2.9 The Bias-Variance Tradeoff

Updating a model during training to minimise a specified error (for example mean squared error) is a key step in machine learning. Models are affected by three types of error [44]:

- Bias - aka underfitting, where the model cannot associate between features and outcomes.
- Variance - aka overfitting, where the model cannot generalise from the training set to unseen data.
- Noise - or the irreducible error (unpredictable changes or measurement errors), which cannot be explained by any model.

The bias-variance trade-off essentially states that we can have a model with no bias or no variance, but not both. Combining the three terms as per [45] we can see in equation 2.1:

$$E(y_0 - \hat{f}(x_0))^2 = Var(\hat{f}(x_0)) + Bias(\hat{f}(x_0))^2 + Var(\epsilon) \tag{2.1}$$

where:

E = the error or difference between target and actual value

x_0 = a given value of X

y_0 = target value for input x_0

$Var(\epsilon)$ = the irreducible error or noise in the dataset

$\hat{f}(x_0)$ = actual value for input x_0 produced by the model.

We can minimise (but not eliminate) bias and variance through the selection of an appropriate training regime and training data. In addition, in some cases it may make sense to train a biased model or estimator if our goal is to minimise the mean squared error.

2.3 Machine Learning In E-Commerce

Now that we have provided an overview of machine learning, in the following sections we propose how machine learning is used in, and influences the field of e-commerce as follows:

1. As a common, visible service - used to improve the end-user experience directly, e.g. recommendations, predicting user intent, learning to rank.

2. As a common, invisible service - used to improve the end-user experience indirectly, e.g. catalogue management, fraud detection.

3. As an important e-commerce ecosystem service - for example predicting click-through rates.

Our aim is to show the breadth and depth of the e-commerce domain, coupled with the pervasive application of machine learning throughout the domain. An important point to note is the inherent scale of e-commerce - any successful model must be able to train

on very large datasets and produce predictions on unseen data in near real-time. Dean and Barroso provides an insight into the varied techniques used to ensure that virtually all users are happy with system performance even at massive scale.

2.4 Predicting Ad Click-through Rates

Given that the dominant business model of the web is offering a free service complemented by monetisation through advertising, it comes as no surprise that predicting and increasing click-through rates has received a lot of research attention [47, 48]. Predicting ad click–through rates (CTR) is a massive-scale learning problem that is central to the multi-billion dollar online advertising industry.

2.5 Content Discovery

Content discovery is one of the older applications of ML to e-commerce [11]. In this use-case, the merchant or portal has far more content than the user can browse through manually, or has noticed that users have low session time / high churn rates on the service. Recommender systems [12] are widely used to predict what content or product the user will be interested in based on their previous behaviour. Within the field of recommender systems, a wide variety of machine learning techniques are employed to achieve this goal, such as:

1. Statistical analysis (popularity) - this is often used in the cold start setting, where no user information has been gathered yet [12].

2. User similarity and item similarity using approaches such as matrix factorisation [49].

3. Sequence processing using recurrent networks [50].

If unsolved, the user - content mismatch typically manifests in a number of ways, all detrimental to the overall user experience and hence site popularity and ultimately profitability [12]:

1. User paralysis - the sheer number of options cause the user to freeze and never consume content [11].

2. The needle in the haystack - the user cannot find what they are looking for, either by searching or navigation [11].

Recommender systems are designed to directly address this problem, by proposing content or products to users, after observing only a very small number of interactions or no interactions at all (the so-called cold start problem). In some domains, recommender systems work very well, for example 75% of all content consumed on the Netflix platform comes from their various recommendation algorithms. From figure 2.1 we can see that recommendations are commonly used across an entire e-commerce website - to suggest new content, up-sell and cross-sell.

However, recommender systems also suffer from a number of drawbacks [51]. Feedback loops caused when training data is harvested from a production deployment where users are already exposed to recommendation engines - causing confounding. Other

measures such as low diversity of recommendation, high homogeneity of users and the ability to create filter bubbles are well-known issues in the recommender community.

2.6 Predicting purchase propensity

Knowing what a user intends to do clearly has value. Search results can be made more relevant, concrete calls to action can be targeted towards the user to help them discover, browse or purchase new content or to persuade them to complete their purchase.

Most of the data used to infer user intent is *implicit*, not explicit. It is difficult to get users to give feedback on their site experience, so instead we use their known behaviour (i.e. their clickstream) to infer it. Joachims led the way in using implicit signals in the form of clickthrough data to re-rank documents returned for a given query so that the most relevant entries are at the top of the list. In this work, a Support Vector Machine (SVM) was used to implement the ranking algorithm. Although the main purpose of [52] is to improve the ranking function for a search engine, it is also inferring user intent from implicit signals, as it is only when user intent is known that more relevant results can be prioritised. Inferring user intent can also be situated as a special case of personalised search. In [53], the authors utilise variability in user intent as measured by several click-based measures (click entropy, potential for personalisation curves) to show that different users will find different results relevant for the same query.

Another line of work relevant for user intent is change point detection in user preferences over time. In [54], Hidden Markov Models (HMM) are used to identify these states based on an empirically-selected threshold of error. The common link here is

using clickstreams as a proxy for user intent, i.e. an implicit signal. This is a concept that we leverage heavily in the future chapters.

Forecasting whether or not a user will purchase an item is closely related to, but separate from that of predicting content interactions. For example if user u_k clicks on item i_m and we know that i_m is both popular and frequently purchased, this allows our model to be more confident in predicting that u_k has a higher propensity to purchase than the median or mean.

But many more local and global variables also need to be taken into account - the user's dwelltime or think time per item, the time of day and season, what other users are currently doing on the site, merchant sales events or special offers [12, 55, 56].

However, the investment involved is worth it to the merchants and portals. Specific actions can be directed to potential buyers to convince them to commit: targeted advertising, special incentives such as time-limited discounts and so on.

The problem of user intent or session classification in an online setting has been heavily studied, with a variety of classic machine learning and deep learning modelling techniques employed. [55] was the original competition winner using one of the datasets considered in later chapters using a commercial implementation of GBM with extensive feature engineering and is still to our knowledge the state-of-the-art implementation for this dataset.

Hidasi et al. uses RNNs on a subset of the same dataset to predict the next session click (regardless of user intent) so removed 1-click sessions and merged clickers and buyers, whereas this work remains focused on the user intent classification problem.

[58] compares [57] to multiple models including kNN, Markov chains and association rules on multiple datasets and finds that performance varies considerably by dataset. [59] extends [57] with a variant of LSTM to capture variations in dwelltime between user actions. User dwelltime is considered an important factor in multiple implementations and has been addressed in multiple ways. For shopping behaviour prediction, [60] uses a mixture of Recurrent Neural Networks and treats the problem as a sequence-to-sequence translation problem, effectively combining two models (prediction and recommendation) into one. However only sessions of length 4 or greater are considered - removing the bulk from consideration. From [61], we know that short sessions are very common in e-commerce datasets, moreover a user's most recent actions are often more important in deciphering their intent than older actions. Therefore we argue that all session lengths should be included. [62] adopts a tangential approach - still focused on predicting purchases, but using textual product metadata to correlate words and terms that suit a particular geographic market better than others. Broadening our focus to include the general use of RNNs in the e-commerce domain, Recurrent Recommender Networks or RRNs are used in [50] to incorporate temporal features with user preferences to improve recommendations, to predict future behavioural directions, but not purchase intent. [63] further extends [57] by focusing on data augmentation and compensating for shifts in the underlying distribution of the data.

In [64], the authors augment a more classical machine learning approach (Singular Value Decomposition or SVD) to better capture temporal information to predict user behaviour - an alternative approach to the event replication or unrolling methodology used in this paper.

Using embeddings as a learned representation is a common technique. In [65], embeddings are used to model items in a low dimensional space to calculate a similarity metric, however temporal ordering is discarded. Learnable embeddings are also used in [66] to model items and purchase confirmation emails are used as a high quality signal of user intent. Unrolling events that exceed an arbitrary threshold to create a better input representation for user dwelltime or interest is addressed in [67]. In [68], Convolutional Neural Networks (CNNs) are used as the model implementation and micro-blogging content is analysed rather than an e-commerce clickstream.

Predicting user intent using a variety of machine learning methods is a central focus of this thesis. In Chapter 4 and Chapter 5 we will examine classical and deep learning methods for inferring user intent from their anonymous clickstream data.

2.7 User Interface (UI)

Whether the user is browsing a site on a mobile device, laptop or desktop - the user interface (UI) presented plays a critical role in their perception of the system overall. Non-functional characteristics such as speed / responsiveness and stability matter, but increasingly advanced UIs are tailoring or personalising themselves around the user based on previous interactions [69].

The starting point for all e-commerce UIs is the F-shape or “golden triangle” - the phenomenon in eye tracking studies which shows that most users focus on the top left quadrant of a display. Figure 2.8 illustrates the effect clearly. The hot colours represent

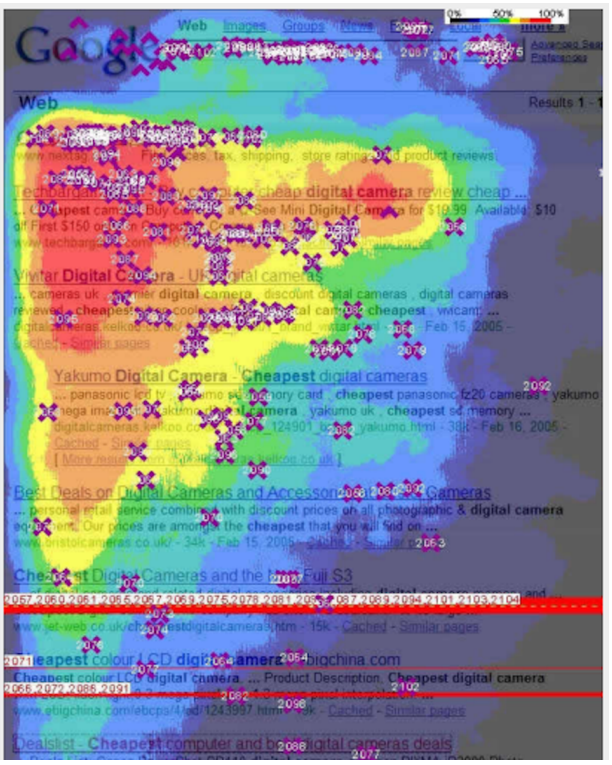

FIGURE 2.8: The golden triangle pattern of user eye-tracking clusters. Hotter (red, yellow) colours signify more focus by the end-user, while cooler shades signify low interest and user focus.

the most valuable segment of the screen - both commercially and from an information retrieval / visualisation point of view.

Conditioned by consistent search engine UIs, users expect to see relevant results at the top, and then work their way down the list. So the ranking method and algorithm is what drives the UI experience for traditional search. But new UIs drive new user behaviour - for example the traditional golden triangle assumption does not hold for image search, or for browsing shopping results where a user's attention is spread more equally across a grid of results. A user can reasonably expect to find the best search result high up for traditional search, and in the middle of the page (and in a central column) for e-commerce search.

In [70], the authors propose reinforcement learning (Q-Learning) to infer how best to

rank and dynamically display results based on a users preference, using permuted Discounted Cumulative Gain (DCG) rather than normalised DCG as a reward signal. The method ranks both the documents and positions, rather than just the positions. On-demand video providers such as Netflix and Amazon also use UI layouts where the best match may be multiple rows below the top row, and not even in the first position for that row. The larger the product catalogue (and the more categories contained within it), the more UI innovation is needed to help users find what they are looking for.

2.7.1 A/B Testing

A/B testing is the primary mechanism whereby the effectiveness of a new approach or technique is tested by online experimentation. A statistically significant proportion of traffic / users is diverted to the new variant (the B variant), while the remainder of users continue to use the current variant (A). After enough data has been gathered, the new variant either becomes the default or is discarded because it under-performs the incumbent variant for a specific goal. Multi-variate (> 2 variants) testing can also be used, as long as the mutations are all of the same base feature.

A large body of empirical work exists to support A/B testing [71] and all technology companies such as eBay, Netflix, Google use A/B testing as a matter of course to guide future product direction. At scale, Google and Microsoft regularly run hundreds or thousands of experiments concurrently.

A/B testing suffers from a number of weaknesses which should be addressed during experiment design [72]. A more nuanced criticism is that A/B testing can only directly

measure short, not long-term gains (A/B testing is a greedy optimisation strategy, simply selecting the variant with the most clicks when a more nuanced metric may offer better long-term results), although [73] claims to address this flaw.

2.7.2 Guided Navigation

In guided navigation, machine learning is employed within the UI widgets normally used to traverse the hierarchical structure of the e-commerce site - which is itself a mirror of the product or item catalogue which underpins the site. Normally, as the catalogue grows the navigation menu grows, until it is unfeasible to render it on a smartphone or tablet, and / or unreasonable to expect a user to scroll past unwanted options in their search for what they really need.

The optimisation of navigation and menus while minimising un-needed change has been widely studied [74], [75] and multiple solutions have been proposed. In [76], an innovative display widget is combined with a learning component to reduce item selection time, while minimising the disruption factor to the user (users prefer static menus as a remembering aid, but dislike very large / unwieldy static menus).

2.8 Information Retrieval

In this section, we move away from the user interface and focus on servicing user requests, i.e. providing answers to user queries posed as efficiently as possible. Information Retrieval (IR) operates at the very heart of search engines which are the dominant

mechanism used to index and query e-commerce product catalogues.

2.8.1 Search

In e-commerce, search queries remain the dominant mechanism used to find content.

In classical search, users enter one or more terms or keywords and receive the top matching index entries matching those terms. Calculating the top matching or most relevant documents is an active area of research. One of the simplest and most widespread ways is to use the well-known tf-idf mechanism [18]. In tf-idf, term frequency (tf) and inverse document frequency (idf) are combined to produce a composite weight for each unique term in each document. The tf-idf weighting scheme assigns to term t a weight in document d given by:

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t.$$

In other words, $\text{tf-idf}_{t,d}$ assigns to term t a weight in document d that is highest when t occurs many times within a small number of documents (thus lending high discriminating power to those documents); lower when the term occurs fewer times in a document, or occurs in many documents (thus offering a less pronounced relevance signal); lowest when the term occurs in virtually all documents.

Two open-source search engines (Solr [77] and Elasticsearch [78]) dominate e-commerce usage and both use the same component to perform document ranking - Lucene [79]. In a practical setting, various heuristics are used to further increase or boost relevance, for

example boolean conditions (must have / must not have), tf-idf and vector space models which encapsulate domain-specific concepts.

The central concepts used in IR are a dictionary of terms, and an inverted index which stores documents - term occurrences [18].

Other well-known IR algorithms are Ponte-Croft's language model and BM25, which we describe in the next section.

2.8.1.1 BM25

A widely-known and empirically successful term probabilistic relevance framework is the Okapi Best Match 25 Model, commonly referred to as BM25 [80]. The model computes local weights as parameterised term frequencies and global weights as RSJ (Robertson Sparck Jones) weights. The original BM25 algorithm omitted the structure of documents in the weighting process, clearly an important signal when we consider that HTML documents are almost always organised using tags such as head, body, title, description, paragraphs, divisions, forms, etc. To address this issue, the original BM25 researchers proposed a simple BM25 extension for weighting terms present in multiple fields that they referred to as the BM25F Model [81]. To underline how important BM25 is to day-to-day e-commerce search, since Apache Lucene 6.0.0 was released (2016), BM25 is the default similarity function used.

2.8.1.2 Ponte-Croft

A language modelling (LM) approach to information retrieval was first proposed in [82]. In contrast to approaches such as BM25, in [82] Ponte and Croft argued strongly for the effectiveness of the term weights that come from the language modelling approach over traditional tf-idf weights. The difference between LM and PRF is that instead of overtly modelling the probability $P(R = 1|q, d)$ of relevance of a document d to a query q, the LM approach instead builds a probabilistic language model M_d from each document d, and ranks documents based on the probability of the model generating the query: $P(q|M_d)$.

Increasingly however, it is becoming clear that e-commerce search is not the same as normal web search. The user behaviour is different - dwelltimes are longer, a user may be happy / satisfied with search results but still continue to search as they are comparing products. In [83], the authors call out the difference between e-commerce search compared to generic search and propose a framework to model the different stages of the conversion journey.

Looking beyond the current state of e-commerce search, we believe that there are three open challenges to improving search in the future - these are not new challenges, but they do sum up the current research focus:

- Improving results relevance.
- Increasing user satisfaction.
- Enabling discovery of new content.

As the field of e-commerce matures and grows, the limitations of traditional search are being explored and addressed. In [84], the authors are motivated by the restrictions imposed by "bag of words" search queries to propose a simplified query ontology that is optimised explicitly for e-commerce (allowing users to quickly find specific concepts such as brands and products). Visual search is also becoming popular, and in [85] the authors propose an extension to standard inverted-index search techniques by encoding image feature vectors or embeddings into a collection of string tokens in a way such that more similar vectors will share more string tokens in common. Enabling this type of search means that users no longer need to know exact terms such as manufacturer, model number - simply uploading a photograph taken from a smartphone camera will suffice to locate the product in a catalogue.

2.8.2 Natural Language Processing

Parsing and tokenisation of product descriptions is a key problem to solve in e-commerce - without this step, the inverted index and associated postings data structure used in the search process cannot be created correctly, resulting in less relevant answers to queries.

In earlier generations of e-commerce systems, users became accustomed to providing the system with exact-matching keywords in order to obtain search results. However, users are increasingly less likely to do this and expect to receive relevant search results when more natural terms are used.

2.8.3 Question Answering (QA)

The advent of accurate speech-to-text systems has opened up an entirely new way of searching and navigating while shopping: dialog or question answering [3].

Here is a concrete example - imagine a user wants to purchase an inexpensive camera to begin learning photography. Typical search queries would be “cheap camera”, “best starter camera” etc. In question-answer mode however, the user can naturally ask “what is the best value camera for a beginner” and expect a relevant result - a good camera to start out with at a reasonable price point.

Servicing these requests is not trivial. Just some of the problems that require solutions to be in place are Named Entity Recognition, word disambiguation and anaphora (an expression whose interpretation depends upon another expression) resolution. Within the domain of question-answering, there is a wide range of tasks. Jurafsky and Martin lists some common question topologies or categories such as abbreviation expansion, description, entity, human, location and numeric. Additionally, well-curated product taxonomies need to be in place to help situate questions and answers.

Table 2.1 contains a cross-vertical sampling of questions and answers from [6]. The dataset used in [6] is a 1.4 million set of question-answer pairs from Amazon, thus representing a good cross-section of the kinds of questions, concepts, entities and numeric values we can reasonably expect any competent e-commerce QA system to learn. In the table, ’?’ indicates an open-ended answer, i.e. where a clear binary yes or no

does not suffice. An ellipsis indicates a long question or answer. It is clear that for e-commerce QA, specific challenges exist such as understanding and incorporating product attributes, SKUs, quantities, weights, sizes as well as subjective properties such as 'cheap' or 'large'.

Category	Question	Answer	Type
Video Games	Will it work with Windows 8?	Yes	Y
Appliances	Does this come with power cord and dishwasher hook up?	It does not come with a power cord. It does come with the dishwasher hookup.	?
Appliances	Is it compatible to replace my Maytag UKF8001 Pur Refrigerator Water filter?	Yes, of course. The Woder Fridge Filter fits behind the fridge or anywhere along the water line. Therefore..	Y
Beauty	can you fit make up brushes in the trays	yes it comes with adjustable dividers..	Y
Automotive	Are these cables made of copper or aluminum?	Coleman's website does indeed say copper equivalent..	Y
Grocery	Is there sugar added to this product? Thank you...	The best of my knowledge there is no added sugar. There..	N
Home and Kitchen	I'd like to use this for a 34 oz capacity teapot. Is it large enough?	Sorry, no. It makes one really strong cup of tea or two "normal" strength.	N
Sport and Outdoor	Could you please confirm that this is made in the USA? Thank you!	Yes this product is made in the USA. The manufacturer is in Texas.	Y
Pet Supplies	What is the smallest or lowest weight dog it will fit on?	It doesn't work too well on my dog, he's a mix breed 10-12 lbs it needs to be about a 15-20 lb at least I would say.	?
Clothing, Shoes and Jewellery	How long does it take to fully charge the battery?	About two hours from the computer.	?
Musical Instruments	Anyone tried this in an effect loop, without amp or cabinet modelling?..	It works totally fine man, pretty amazing sound for the cheap price.	?

TABLE 2.1: Sample questions and answers from [6] spanning a selection of common categories.

In fact, the problem of Question-Answering illustrates clearly how traditional information retrieval techniques must be augmented by machine learning to provide an adequate solution. Probabilistic or language model approaches cannot infer user intent, or provide a deep enough semantic search capability. Framed as a machine learning problem, multiple techniques and approaches are brought together to understand user questions and provide good answers:

- Deep Learning to learn embeddings (allows semantic matching between questions and snippets) [3].
- Loss functions such as Siamese contrastive loss [86] that maximise difference in class scores.

Jurafsky and Martin characterises question-answering systems as either IR-based (text-based) or knowledge-based. IR-based QA systems can rely on the tried and tested methods such as tf-idf, BM25 etc. to try and find the most relevant snippet that answers a user question. Knowledge-based systems however, attempt to construct a semantic understanding of both the query and potential answer candidates to provide a suitable answer.

In practice, current best empirical results are obtained from hybrid systems using both IR and knowledge base components. Figure 2.9 shows the major components of a question-answering system. We extend the original diagram and include e-commerce specific steps such as integrating a curated product catalogue to cross-reference products and also user purchase intent prediction to help rank answers (a user just entering

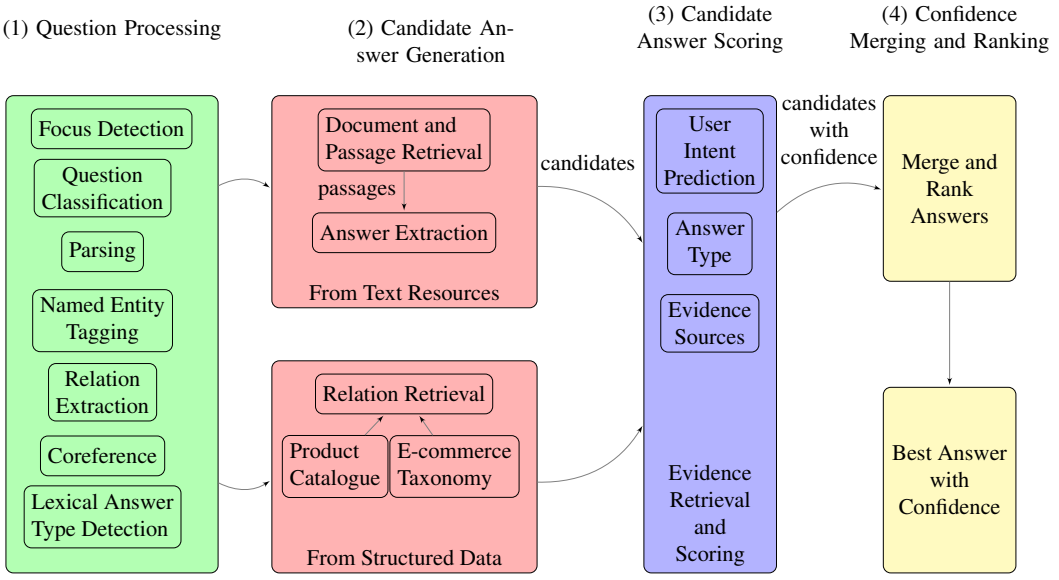

FIGURE 2.9: After [3]. Main components of a hybrid question-answering system (IBM Watson in this case from 2011), organised as a processing pipeline with distinct phases: question processing, passage retrieval, answer processing.

the research phase may prefer longer, more involved answers whereas a user close to purchasing may prefer short, unambiguous answers).

2.8.3.1 Churn Prediction

For merchants who rely on recurring revenue from users (e.g. telecommunications providers, insurers), predicting (and then trying to dissuade) users from leaving is an important topic. A large telecoms provider made a significant dataset available in [87] to predict churn as well as buy new products or services (appetency), or buy upgrades or add-ons proposed to them to make the sale more profitable (up-selling).

2.8.4 Customer Relationship Management (CRM)

The discipline of Customer Relationship Management or CRM attempts to build up a multi-modal, long-term view of the customer for the mutual benefit of the customer and the merchant. Therefore rather than focusing on short-term user intents such as purchase prediction or content interaction, CRM applies a more holistic approach. An example would be understanding the user's longer-term and / or recurring motivations as well as shorter-term actions.

2.9 Research vs Real World Usage

The e-commerce domain is inherently empirical - any new research technique or claim will be tested with real users and products and rejected if it fails to meet expectations. But there are also nuances in the commercialisation of e-commerce research (i.e. how research permeates into production systems), particularly as it relates to hybrid ML research.

The largest e-commerce retailers (Amazon, eBay, Taobao et al) build, maintain and extend their own systems, leveraging open-source components heavily but retaining full control over the software development stack.

In contrast, middle-tier operators (and certainly small operators) do not do this due to lack of expertise and high cost - most of these operators standardise on a commercial offering such as ATG Web Commerce [88], IBM Websphere [89] or SAP Hybris [90].

Therefore their ability to incorporate ML into their e-commerce presence is limited by two factors:

1. How quickly and how well their selected platform vendor adds ML capabilities to the platform.
2. How open the platform is, enabling the operator to integrate ML capabilities directly.

As the saying goes, “the future is here, it is just not evenly distributed”.

2.9.1 Automated Spend

A unique characteristic of e-commerce environments is that real spend is immediately measurable. All major portals such as Amazon, eBay, Google, Taobao, Facebook, Twitter and Instagram offer real-time APIs that can be used to place bids for targeted advertisements to reach specific subsets of users. Through these APIs, substantial spends can be entirely automated and placed in the control of a trained model. However, given the current lack of interpretability and good visualisations for trained models (see Chapter 7), it is a very brave e-commerce director who would choose to do this. At the very least, a skilled human administrator will still be used to monitor and supervise model selections before money is spent on advertising bids.

2.10 Summary

In this chapter, we have briefly described the e-commerce domain, and how machine learning has been applied to it. The fields of information retrieval and recommender systems have figured heavily in our review since search and content discovery are so important to e-commerce. We have identified the problem of deciphering user intent from anonymous clickstreams as one which remains important and acts as a cross-cutting concern affecting many other e-commerce tasks. In the following chapters we will construct and test various models to solve the user intent task - in Table 2.2 we compare the models covered in this chapter and our logic for selecting the candidates chosen for detailed examination. No single model is entirely suitable for the task at hand, but some models (HMM, kNN) are disqualified outright due to significant flaws which cannot be remedied. This analysis shows that deep neural networks and gradient boosted decision trees hold the most promise as candidate models for the user intent task. In our opinion the most important model capabilities are:

1. The ability to process streams of data, i.e. data which loses fidelity when expressed as a single row or entry.

2. The ability to reconstruct the target function, which is non-linear and complex.

3. Interpretability - allowing an end-user to understand model reasoning underpinning outcomes.

4. Scalability - as previously noted, e-commerce datasets are at least reasonable in size (tens of millions of events).

5. Simplicity - we prefer models that are more simple to train and perform inference on as these models can be promoted into production environments in a straightforward manner.

Characteristic	Deep Neural networks	Boosted Decision Trees	kNN Models	Hidden Markov Models
Process streaming data	Y	N	Y	Y
Expressive power for user intent task	Y	Y	Y	Y
Interpretable	N	Y	Y	Y
Scalable to large datasets	Y	Y	N	N
Straightforward to train	N	Y	Y	N

TABLE 2.2: The main model candidates covered in this chapter compared using requirements from the e-commerce domain.

In the next chapter we commence our detailed evaluation of machine learning approaches focused on the user intent detection task, beginning with a system architecture overview.

Chapter 3

System Architecture and Implementation

In this chapter we describe the practical implementation details of our system - encompassing data storage, pre-processing and feature construction, model training, hyperparameter tuning and evaluation. The experiments conducted in Chapters 4-6, as well as the hyperparameter searches we employed to select the optimal model configuration stem directly from design and implementation decisions documented in this chapter. In particular, the code we authored combined with selected third party libraries enabled:

- Formal measurement of e-commerce clickstream feature importance in Chapter 4 to discover user intent using explicit features.
- Tuning of hyperparameters including model architecture in Chapter 5 to recover close to state-of-the-art performance using learned, rather than explicit features.

- Experiments relating to training set sampling strategy and hidden state management in Chapter 6.

Figure 3.1 illustrates the major system components in the architecture. Hyperparameter tuning was provided by Spearmint [91, 92], which integrated with our Torch (Lua), Python and Java code by invoking operating system processes with command-line arguments.

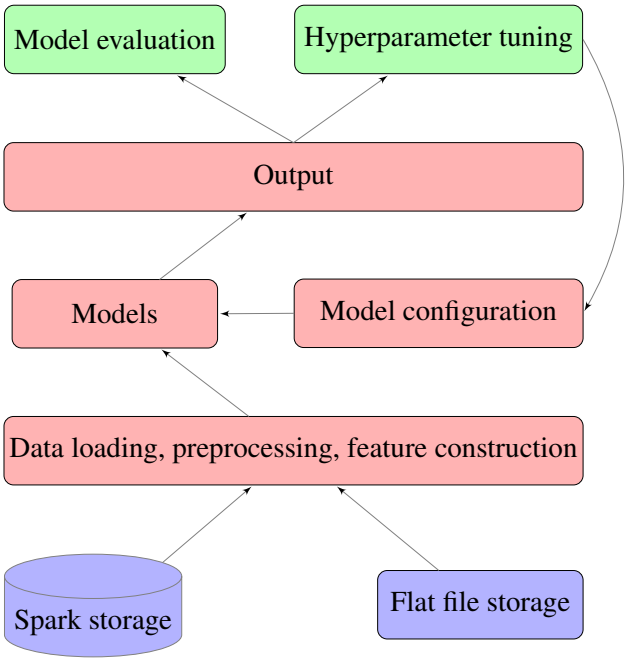

FIGURE 3.1: All of our constructed systems share common components, as illustrated here: a data loading and transformation layer, configurable models with a trainer, and finally an evaluation module.

3.1 Data storage

We used an Apache Spark database [93] as a queryable store for the clickstream datasets used. For our Gradient Boosted Machines (GBM) model, feature builders / extractors were written in the Java programming language, while our RNN models were coded in

Python using the PyTorch [94] deep learning library. While Spark itself is designed to be a fast and general-purpose cluster computing system, we used it in our architecture as a data store, using the Spark SQL dialect (which supports most but not all of the ANSI SQL specification) to query the clickstream data to calculate session-level and global feature values. The Spark project also provides implementations of machine learning classification and regression algorithms such as logistic regression, decision trees and gradient-boosted tree regression, however we did not use these elements of Spark in our implementation. Instead we used a dedicated GBM implementation [28] due to its widespread use in the recommender systems community [56, 95, 96] - enabling straightforward comparison between our work and that of the wider research community. Spark does not provide any deep learning model or training algorithm implementations itself, although it is straightforward to load data from Spark and create input tensors for any deep learning framework such as Keras, Tensorflow or PyTorch.

3.2 Datasets Used

The RecSys 2015 Challenge [97] and the Retailrocket Kaggle [98] datasets provide anonymous e-commerce clickstream data well suited to testing purchase prediction models. Both datasets are reasonable in size - consisting of 9.2 million and 1.4 million user sessions respectively. These sessions are anonymous and consist of a chronological sequence of time-stamped events describing user interactions (clicks) with content while browsing and shopping online. The logic used to mark the start and end of a user session is dataset-specific - the RecSys 2015 dataset contains more sessions with a small item catalogue while the Retailrocket dataset contains less sessions with an item

catalogue 5x larger than the RecSys 2015 dataset. Both datasets contain a very high proportion of short length sessions ($<= 3$ events), making this problem setting quite difficult for RNNs to solve. The Retailrocket dataset contains much longer sessions when measured by duration - the RecSys 2015 user sessions are much shorter in duration. In summary, the datasets differ in important respects, and provide a good test of model generalisation ability.

For both datasets, no sessions were excluded - both datasets in their entirety were used in training and evaluation. This means that for sequences with just one click, we require the trained embeddings to accurately describe the item, and time of viewing by the user to accurately classify the session, while for longer sessions, we can rely more on the RNN model to extract information from the sequence. This decision makes the training task much harder for our RNN model, but is a fairer comparison to previous work using GBM where all session lengths were also included [23, 55, 56]. Table 3.1 provides a brief comparison of the main characteristics for each dataset.

	RecSys 2015	Retailrocket
Sessions	9,249,729	1,398,795
Buyer sessions	5.5%	0.7%
Unique items	52,739	227,006

TABLE 3.1: A short comparison of the two datasets used - RecSys 2015 and Retailrocket.

3.2.1 Data Preparation

The RecSys 2015 challenge dataset consists of 9.2 million user-item click sessions. Sessions are anonymous and classes are imbalanced with only 5% of sessions ending in one or more buy events. Each user session captures the interactions between a single

user and items or products : $S_n = e_1, e_2, .., e_k$, where e_k is either a click or buy event. An example 2-event session is:

SID	Timestamp	Item ID	Cat ID
1	2014-04-07T10:51:09.277Z	214536502	0
1	2014-04-07T10:57:09.868Z	214536500	0

TABLE 3.2: An example of a clicker session from the RecSys 2015 dataset.

Both datasets contain missing or obfuscated data - presumably for commercially sensitive reasons. Where sessions end with one or more purchase events, the item price and quantity values are provided only 30% of the time in the case of the RecSys 2015 dataset, while prices are obfuscated for commercial reasons in the Retailrocket dataset. Therefore these elements of the data provide limited value.

SID	Timestamp	Item ID	Price	Quantity
420374	2014-05-27T10:03:09.277Z	214537888	12462	1
420374	2014-05-27T10:05:09.277Z	214537850	10471	1

TABLE 3.3: Examples of the buy events from a buyer session .

The Retailrocket dataset consists of 1.4 million sessions. Sessions are also anonymous and are even more imbalanced - just 0.7% of the sessions end in a buy event. This dataset also provides item metadata but in order to standardise our approach across both datasets, we chose not to use any information that was not common to both datasets. In particular we discard and do not use the additional "addtobasket" event type that is present in the Retailrocket dataset. Since it is so closely correlated with the buy event (users add to a basket before purchasing that basket), it renders the buyer prediction task trivial and an AUC of 0.97 is easily achievable for both our RNN and GBM models.

3.3 Data Pre-processing and Transformation

The primary tasks encountered when dealing with clickstream data are:

- Conduct exploratory data analysis [99] - for example to count the number of unique entries in a given column which is then used to select the width of the relevant embedding for LSTM, or as input into a feature design for GBM.
- Remove (filter out) unwanted sessions in the data - e.g. in the Retailrocket dataset [98], sessions that are too long will cause the GPU to run out of memory when they are loaded into a training batch and the entire batch is zero-padded to the longest length (long sessions can be thousands of events in length once unrolled).

The data pipeline must support two use-cases depending on the model being trained:

- For GBM, instantiate feature builders, load and transform the raw clickstream data to features and output the constructed features in the LIBSVM file format required by our chosen GBM implementation.
- For RNN / LSTM, load the raw clickstream data, transform the data to lookup IDs for use with the embedding layer and maintain the session grouping in a 2-dimensional tensor.

Figure 3.2 illustrates the colour-coded structure of the data pipeline used for the primary models in this thesis - GBM and RNN / LSTM, as well as the third-party libraries (PyTorch and XGBoost) used in the implementation. For both the GBM and RNN

models, the underlying core libraries (i.e. XGBoost and PyTorch) were pre-installed on the training server. Additional dependencies necessary for GPU training of RNN-based word language models were also pre-installed - CUDA and cuDNN. Our chosen implementation of GBM [28] loads LIBSVM flat files from disk, while our RNN / LSTM models use 2D in-memory tensors which are then converted to 3D tensors when passed through an embedding layer - the pipeline supports both outputs. The colour coding is as follows: green represents code assets written as part of this thesis. Blue represents an output type from the system. Red represents a third-party library used by the code.

In the case of GBM, new feature extraction code was deployed to the server while for the RNN models, new model code and configuration was deployed. The code file transfer mechanism used in both cases was rsync and code version control was provided by git. In the future we would like to explore the use of containers to distribute well-defined code packages as a way to exploit multiple servers during training. Containers were not necessary in this work as we were able to train both GBM and LSTM models on a single server.

3.4 Code Structure

Our design ethos for the Python code was to separate the main ML tasks and so the structure is:

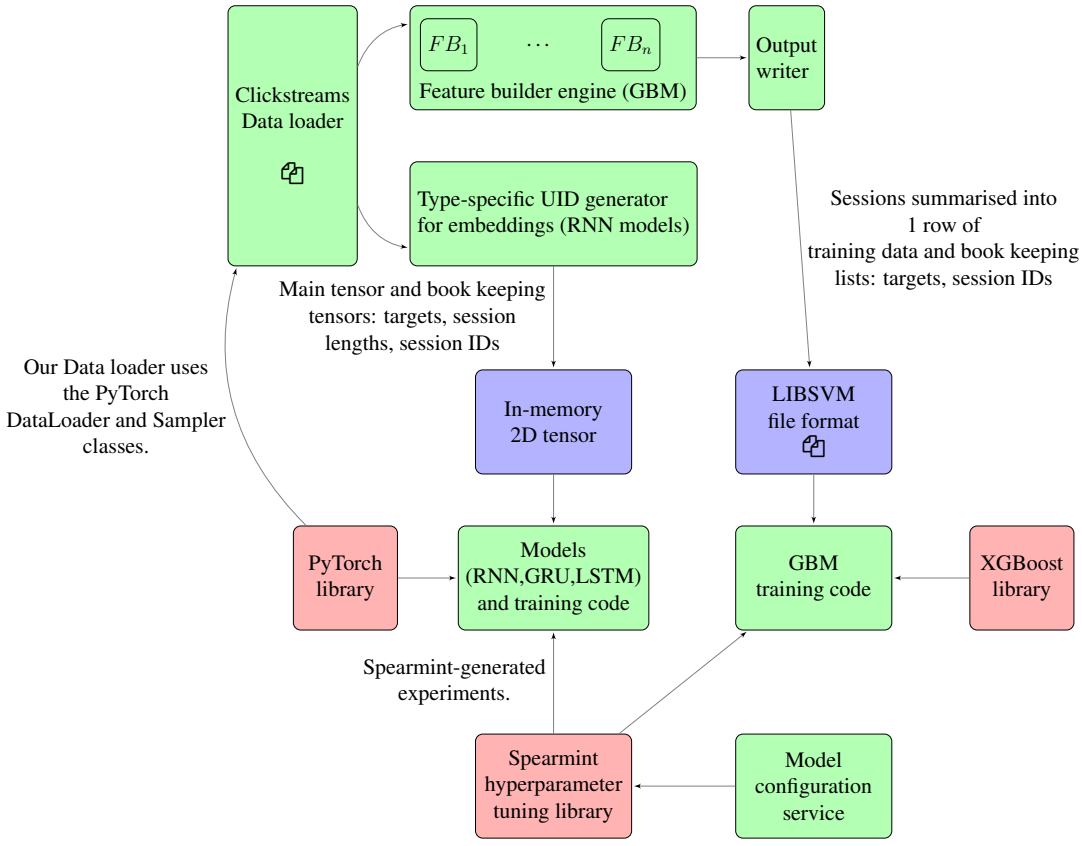

FIGURE 3.2: The end-to-end system constructed, including the data load and transform pipeline.

- session_runner.py: invoked directly from the command line or by Spearmint [91] and used the argparse library heavily to configure the system, especially to tune hyperparameters.

- session_trainer.py: contains the main training and evaluation loops, as well as initiating the initial data load and model creation.

- session_data.py: load data and transform it to a 2D tensor ready for training. Relies heavily on the DataLoader framework from PyTorch for structure and logic.

- session_model.py: Creates the model architecture being trained and also contains the implementation of the forward pass which is then reversed by the PyTorch auto-differentiation sub-framework for error back-propagation.

This split reflects some practical considerations in deep learning - in particular the realisation that models change more frequently than data loading or training code. This code organisation allows us to make model changes quickly (and controlled by configuration flags so that they can be tested in both on / off configurations and with different hyper-parameters) while insulating the rest of the codebase from these changes. The Python code described here was deployed as the module titled "Models (RNN, GRU, LSTM) and training code" in Figure 3.2.

3.5 Previous Iterations

All of the results in this thesis were generated using version 2 of the system. The first implementation had a number of drawbacks which we addressed in the second version:

- The use of WEKA [100] to provide data loading and model implementations. We found that while WEKA is full-featured, it did not scale to our datasets, nor does it contain a robust RNN or GBM model implementation.
- Our chosen RNN implementation - Torch - contained a robust RNN model implementation but was deprecated during our development and replaced by PyTorch.
- Our code structure required us to calculate standard statistical metrics such as mean, median, variance in Java, which lacks good implementations of both ML and numerical analysis libraries when compared to Python.

We remedied these weaknesses in the second implementation - our codebase allowed us to use Python libraries such as scikit-learn, pandas, numpy, scipy, and matplotlib. For

example, by moving to use the Python Pandas project [101], the need for Spark storage was removed and the data could be loaded directly from the file system without any reduction in speed. We ported our Torch Lua code to PyTorch Python code - this task was relatively simple to complete for two reasons:

- PyTorch still relies on the old Lua core libraries but exposes their functionality via a new Python API - thus we did not encounter any missing or different functionality issues.
- Even the earliest PyTorch release came with some code samples and API documentation.

For the reasons outlined previously, in the second iteration of our system we moved to use XGBoost [28] over the decision tree implementations we used previously in WEKA.

3.6 Model Training

Each model was trained according to its own specific algorithm. For GBM, there is a single training algorithm and the metric to optimise for is changeable - we used AUC (Area under the ROC curve). The GBM training algorithm however requires significant hyperparameter tuning which we cover below.

In contrast to GBM, for RNNs the training setup is more complex. In order to train an RNN we require:

- A model with randomly-initialised weights (but carefully initialised and drawn from a small range).
- A criterion which implements our desired loss function.
- A forward pass which propagates the inputs all the way through the model and generates outputs.
- A backward pass which propagates the criterion error or loss all the way through the model.

The RNN framework used here supported Automatic Differentiation (AD) [94], therefore only the forward pass needed to be implemented in our code.

3.7 Summary

In this chapter we described the implementation of the system used to generate and record the experiments carried out in Chapter 4, Chapter 5 and Chapter 6. We also described the datasets used to carry out experiments in those chapters.

Having designed and built two iterations of a machine learning system, we would add the following improvements to a third iteration:

- Provide support for deep learning frameworks other than PyTorch, for example MXNet [102] and Tensorflow [103]. This would allow us to test different implementations concurrently, and provide access to more training algorithms and

RNN variants as paper authors typically provide implementations in one favoured framework.

- Provide support for Gradient Boosted Machine frameworks other than XGBoost, for example CatBoost [29] and LightGBM [27] which have recently experienced a surge in popularity in the recommender system and e-commerce analysis research communities and claim superior training speed and test accuracy over XGBoost.

- Automate experiment management further - including automatic syncing of file system, data, code and github issue tracking.

- Incorporate Tensorboard (a visualisation component of Tensorflow) for improved visualisation of error rates during training.

- Preserve expensive data transformations using caching to speed up initial experiment start times.

Overall, we believe that the second iteration of the system architecture has been successful in supporting our research and experiments. It possesses flexibility where necessary, and incorporates widely-used GBM and RNN implementations which are widely cited, permitting us to compare our research and techniques with other researchers and groups.

In the next chapter we use Gradient Boosted Machines (GBM) to conduct a thorough review of feature importance in the explicit feature setting.

Chapter 4

Traditional Machine Learning and User Propensity

4.1 Introduction

In the first two chapters, we provided an overview of the machine learning and e-commerce fields, as well as illustrating how machine learning - initially through improving information retrieval has become indispensable to providing users with a good experience when they browse and shop online. We also identified that the application of machine learning to e-commerce has created a "squeezed middle" - merchants who face steeply increasing advertising costs and price pressures in a race to the bottom as portals make it easy for customers to compare similar products by attribute - but especially price.

These merchants can tilt the balance back in their favour by focusing limited budget and resources on the small subset of all e-commerce users who possess an intent to buy. In this chapter, we analyse the problem of analysing user sessions to infer intent, including the twin problems of model selection and feature design.

As previously stated, deciphering user purchase intent from website clickstreams and providing more relevant product recommendations to users remains an important challenge in e-commerce. We outline our approach to the twin tasks of user classification and content ranking in an e-commerce setting using an open dataset. Design and development lessons learned through the use of gradient boosted machines are described and initial findings reviewed. We also describe a novel application of word embeddings to the dataset chosen to model item-item similarity which we build on in later chapters.

4.2 Problem Domain: Overview

The primary method used to gather data in the e-commerce domain is to log browser requests for web pages ordered temporally and grouped by a session ID. These logs are then used to train models which classify users by their intent (clicking, browsing, buying) and what items those users are most interested in. Our motivation is to predict the intent of web users using their individual and group prior behaviour and to select from a potentially large set of available content, the items of most interest to match with a specific user. Correctly identifying user intent and matching users to the most relevant content directly impacts retailer revenue and profit [11].

4.2.1 RecSys Challenge

This work focused on an open dataset from the ACM RecSys 2015 conference challenge [97]. The challenge ran for nine months, involved 850 teams from 49 countries, with a total of 5,437 solutions submitted. The winners of the challenge scored approximately 50% of the maximum score. A variety of linear and non-linear classifiers were employed as ensembles and two of the top three accepted submissions [55] and [56] relied heavily on Gradient Boosted Machine (GBM) classifiers, with [56] employing both Neural Networks and GBM.

The challenge dataset is a snapshot of web user activity where users mostly browse and infrequently purchase items from a catalogue. The data is:

1. Reasonable in size - containing 34,154,697 events grouped into 9,249,729 sessions. The sessions comprise events over 52,739 items distributed over 338 categories, with 19,949 of the items purchased.

2. Imbalanced - buyer sessions represent just 5.51% (509,696) of the total.

3. Incomplete - for example 49.5% of the clicks do not contain a category ID for the item clicked.

The objective function to maximise is:

$$Score(Sl) = \sum_{\forall s \in Sl} \begin{cases} if s \in S_b \rightarrow \frac{|S_b|}{|S|} + \frac{|A_s \cap B_s|}{|A_s \cup B_s|} \\ else \rightarrow -\frac{|S_b|}{|S|} \end{cases} \tag{4.1}$$

where:

Sl = sessions in submitted solution

S = all sessions in the test set

s = session in the test set

S_b = buy sessions in the test set

A_s = predicted buy items in session s

B_s = actual bought items in session s

The top submission for the competition achieved a score of $63,102$, 47% of the maximum score attainable: $135,176$, underlining the difficult nature of the task. Moreover, virtually all submissions achieved a negative score on the first component of the score (separating buyer sessions from clicker sessions), again demonstrating the difficulty in discerning between true and false positive cases.

The score is maximised by correctly classifying true buyers while minimising the number of false buyers (i.e. clickers). This is followed by the recommendation or ranking task, where for each buyer the exact items purchased are predicted from the click set for that buyer (a buyer can purchase just one item clicked, all or some).

4.2.2 Wider Applicability

Classification and ranking in order to recommend are not specific to the e-commerce domain. Multiple other domains such as security, finance and healthcare apply similar techniques to solve domain-specific problems. Our intent is to generalise our framework and approach to multiple domains. However, different domain problems will have substantially different objective functions - Table 4.3 shows that in the e-commerce domain high false negative and false positive scores are inevitable but that model confidence

grows as the session length increases. It is easy to imagine a problem in the finance, security or healthcare domains where better classification performance is required (but equally better distinguishing data must also be available).

4.3 Implementation

We described Gradient Boosted Machines (GBM) in Chapter 2 and selected GBM here as the initial model for a number of reasons. The trained models are interpretable in terms of feature usage, gain and coverage. A robust, fast and scalable implementation of GBM is available in [28]. GBM is also straightforward to train as it iteratively grows an ensemble of Classification and Regression Trees (CART) to learn an objective function with regularisation applied to promote generalisation on unseen data. New trees are iteratively added during training to better model the objective function and correct for the errors made by earlier trees. Figure 4.1 illustrates the data flow through the primary modules of the system. Calculated features are saved in LIBSVM format (label:value) and consumed by GBM to build a forest of CART trees. word2vec receives all sessions (S_c and S_b) and is used to calculate two distinct similarity embeddings - modelling items that are frequently clicked together, and items that are frequently bought together. The item model is trained on buyer sessions S_b only, as only buyer sessions can contribute towards the item component of the target score.

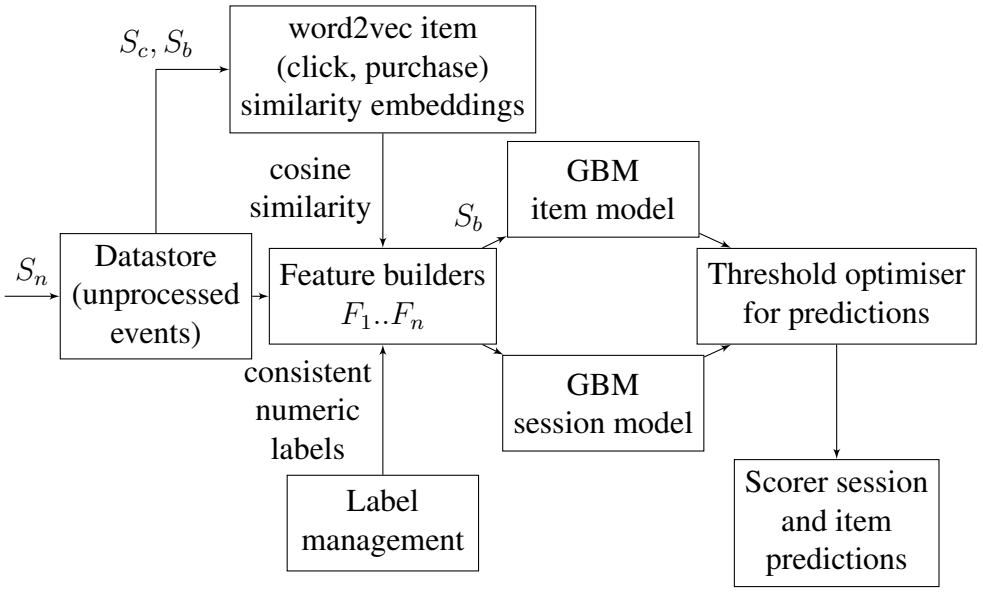

FIGURE 4.1: Primary modules of the end-to-end system implementation. The system currently contains 10 feature builders calculating 68 features in total.

4.3.1 Framework

During the implementation, specific functions and attributes to enable efficient and rapid progress were identified and coded into a re-usable machine learning management framework which we cover in more detail in Chapter 7. The main properties of this framework are:

1. *Reproducibility*. Threshold optimisation and hyperparameter values have a significant impact on the accuracy obtained and thus final score. The framework supports version control of code, data, logs and configuration relating to each experiment.

2. *Composition and Combinability*. Currently we combine homogeneous models together to solve a target task, however the framework also admits heterogeneous models.

3. *Rich data querying capabilities*. Spark SQL is used to enable rapid, iterative data analysis to test feature accuracy and to suggest new feature designs.

4. *Consistent Feature generation across models*. Feature re-use across models promotes code re-use across models and experiments.

5. *Labelling and aggregation*. The framework stores labels at session and item level through all segments of the transformation, training and scoring pipeline.

4.3.2 Features

For the session model, 40+ features and a one-hot item vector for the top 5,000 most popular items were calculated from the dataset. For the item model, 20+ features were calculated. The features overlap significantly with other competition submissions ([56], [55]). The most common features used are well understood information retrieval metrics - Table 4.1 and Table 4.2 describe the top ten features for the session and item models, graded by their feature importance score. Importance values are lower here than for the item model in table 4.2 due to the number of features used in the session model. GBM models are interpretable, allowing feature importance scores to be easily calculated. Features are grouped into four types - Temporal, Counts, Similarity and Price. In our opinion, temporal features model user engagement, price features model item competitiveness (item prices rise and fall over time), count features model popularity statistics over the dataset and similarity features model user intent - casual versus focused browsing.

TABLE 4.1: The top ten session features after training for 7,500 rounds, ordered by most important features descending.

Description	Relative importance	Type
Max time spent on an item (millisecs)	0.049	T
Global buys (last item) / Global clicks (last item)	0.048	C
Session duration (millisecs)	0.044	T
Global clicks (last item)	0.043	C
Min item price in session	0.041	P
Max value of(buys / clicks) in the session	0.04	P
Cross entropy of dwelltime across items	0.038	T
Max item price in session	0.038	P
Max click similarity in session	0.037	S
Click similarity standard deviation	0.037	S

The click and buy similarity metrics carry significant weight in this model, resulting in a focused effort to improve them - beginning with a simple count-based Jaccard similarity, progressing to matrix factorisation using Alternating Least Squares, to the current best solution - using pair-wise cosine similarity on embeddings or vectors calculated for each item. The current embeddings are of length 300, with each vector co-ordinate representing a latent variable modelling the item set. Similarity-based features are strongly represented in table 4.2, showing the effectiveness of the current similarity implementation.

4.3.3 Models and Training

The two terms of the objective function are independent, so a divide and conquer approach is a rational strategy [55, 56]. Two models were trained in parallel - the session predictor and the item predictor. During the training phase, the quality of models generated by GBM is sensitive to values chosen for some key values: the tree depth

TABLE 4.2: Top ten item features after training for 5,000 rounds, ordered by most important first.

Description	Relative importance	Type
Summed buy similarity	0.099	S
Max click similarity in session	0.097	S
Summed click similarity	0.097	S
Buy similarity standard deviation	0.096	S
Std deviation of buy similarity / click similarity x num clicks	0.091	S
Summed buy similarity / click similarity x num session clicks	0.083	S
Item dwelltime in this session	0.065	T
Global clicks for this item	0.064	C
Global item buys / global item clicks	0.063	C
(Global buys / global clicks) x num session clicks for this item	0.054	C

(max_depth), learning rate (eta) and breakpoint for new tree nodes (min_child_weight). We currently use sensible values for these parameters as suggested by [56], with a hyperparameter search planned in future work.

It quickly became apparent that the classification task is more difficult to learn than the recommending task - Area under the curve (AUC) is used to measure training progress on a validation set and the best session classification AUC is 0.853 vs 0.895 for the item prediction task. This is due to the imbalanced nature of the dataset and because some of the most common sessions comprise those with lengths between one and three, removing valuable context from some of the global features (for example cross-entropy, click similarity and buy similarity). We partly mitigated the class imbalance issue by down-sampling clickers by 50% and this resulted in a small score increase. Thus with appropriate hyperparameter selection, GBM appears to be reasonably resistant to overfitting on the dominant class in an imbalanced setting.

4.3.4 Inference - Initial Results

The model confidence in predicting user behaviour and recommending items increases based on session length. Therefore the thresholds (the probability value used to separate clickers from buyers) were selected at a session-length level, instead of using a one fits all value. Thresholds for both models were selected using grid search with a stepsize of 0.01 after training, using threshold start and end ranges known empirically to bracket the optimal thresholds. As shown in table 4.3, it is necessary to reduce the probability thresholds for session selection to an average of just 0.09 (0.069 if clickers are not under-sampled), compared to an average of 0.47 for item selection. This low session threshold value demonstrates the difficulty of the session classification task. In general, it is important to use dynamic confidence thresholds predicated on session length to maximise both the session and item components of the overall score.

The current implementation would have placed 6th or 7th (the conference did not contain a paper from the second placed team) on the competition leaderboard out of 850 and makes 99.4% of the GBM-only target score (58,442 vs 58,820 in [56]). The code for the original scoring methodology used in the competition is no longer available [104], however we reverse-engineered and validated the scoring methodology using three input sources - the solution file provided after the competition ended, the solution file provided by the authors of [55] and our own solution file.

TABLE 4.3: Session and item thresholds by session length with scores for the current models, showing the increase in model predictive confidence as the number of events per session grows.

Session Length	Session Threshold	Item Threshold	Session Score	Item Score
1	0.05	0.4	-3276	3344
2	0.06	0.6	-11224	21620
3	0.07	0.54	-6393	14421
4	0.07	0.51	-4328	11246
5	0.08	0.53	-2620	8222
6	0.09	0.49	-1701	6537
7	0.11	0.44	-1067	4942
8	0.11	0.43	-777	3993
9	0.1	0.44	-601	3125
10	0.08	0.44	-485	2530
11	0.1	0.47	-322	2076
12	0.1	0.42	-252	1702
13	0.08	0.46	-212	1335
14	0.14	0.45	-119	1104
15+	0.14	0.42	-554	6175
Totals			-33931	92373

4.3.5 Optimising Click and Buy Item Similarity Features

The optimal similarity measure discovered to date is unique in the competition we believe. Multiple similarity implementations were evaluated including Jaccard similarity and Alternating Least Squares (ALS) matrix factorisation - a staple technique in the recommender community. Currently, items are modelled as words, sessions as sentences and each word is transformed into a low-dimensional distributed representation - the embedding or word vector [41]. This feature is trained by maximising its log-likelihood on the training set:

$$J_{\text{NEG}} = \log Q_\theta(D = 1|w_t, h) + k \mathop{\mathbb{E}}_{\tilde{w} \sim P_{\text{noise}}} [\log Q_\theta(D = 0|\tilde{w}, h)]$$

where:

$Q_\theta(D = 1|w, h)$ = the binary logistic regression probability

h = the context (user session) D = corpus of all sessions

w = word (item ID)

θ = learned embedding vectors

A good vector space model will map semantically similar words close together and this feature exploits this property by calculating item-item similarity using pair-wise cosine similarity. Further experimentation for these features can also be carried out, focusing on document ordering, parameters such as embedding length (currently 300), context (currently 15 words) and the best internal word2vec model to use - Skip Grams vs Continuous Bag Of Words (CBOW).

4.4 Summary

In this chapter we showed how a homogeneous GBM implementation can learn to solve the user intent classification problem on a well-known e-commerce dataset - competing well with more advanced heterogeneous [56] and proprietary [55] solutions. In our experiments, GBM functioned consistently well as a robust classifier, therefore we posit that the score achieved relies substantially on careful feature engineering.

Given the preponderance of click / user event datasets in the e-commerce and recommender domains, we expect the work completed so far to generalise well to other datasets in the same domain but we also note the strong possibility for some engineered

features to be domain or dataset-specific and also the significant feature engineering effort required

In the next chapter we continue to focus on the problem of deciphering user intent, but now our approach will change significantly. From this chapter, we can see that representation learning adds value in modelling e-commerce items, when used in conjunction with hand-crafted features. Our modelling approach in the next chapter will use representation learning *exclusively*, combined with a suitable model, to remove the feature engineering burden associated with gradient boosting.

The material in this chapter draws substantially from [23], published at the 17th Annual UK Workshop on Computational Intelligence.

Chapter 5

Predicting purchasing intent: End-to-end Learning using Recurrent Neural Networks

In the previous chapter, we demonstrated the importance of different types of features in allowing the Gradient Boosted Machine model to accurately separate e-commerce user sessions or clickstreams into our two target classes - clicker or buyer. One feature type which is particularly effective is to build word vectors for clicked items based on session co-occurrence using word2vec [41] and then to apply a simple similarity metric to these vectors - cosine similarity in our case. Recall from Chapter 4 that learned word vectors provided our best item similarity feature. In this chapter we build on this principle, and extend it to the entire dataset, and progress from GBM to a machine learning model which consumes word vectors in a more natural fashion - Recurrent Neural Networks or RNN.

5.1 Introduction

We present a neural network for predicting purchasing intent in an e-commerce setting. Our main contribution is to address the significant investment in feature engineering that is usually associated with state-of-the-art methods such as Gradient Boosted Machines. We use trainable vector spaces to model varied, semi-structured input data comprising categoricals, quantities and unique instances. Multi-layer recurrent neural networks capture both session-local and dataset-global event dependencies and relationships for user sessions of any length. An exploration of model design decisions including parameter sharing and skip connections further increase model accuracy. Results on benchmark datasets deliver classification accuracy within 98% of state-of-the-art on one and exceed state-of-the-art on the second without the need for any domain / dataset-specific feature engineering on both short and long event sequences.

In the e-commerce domain, we propose that merchants can increase their sales volume and profit margin by acquiring better answers for two questions:

- Which users are most likely to purchase (predict purchasing intent).
- Which elements of the product catalogue do users prefer (rank content).

By how much can merchants realistically increase profits? Table 5.1 illustrates that merchants can improve profit by between 2% and 11% depending on the contributing variable. In the fluid and highly competitive world of online retailing, these margins are significant, and understanding a user's shopping intent can positively influence three out of four major variables that affect profit. In addition, merchants increasingly rely on

and pay advertising to much larger third-party portals (for example eBay, Google, Bing, Taobao, Amazon) to achieve their distribution, so any direct measures the merchant group can use to increase their profit is sorely needed.

	McKinsey	A.T. Kearney	Affected by shopping intent
Price management	11.1%	8.2%	Yes
Variable cost	7.8%	5.1%	Yes
Sales volume	3.3%	3.0%	Yes
Fixed cost	2.3%	2.0%	No

TABLE 5.1: Effect of improving different variables on operating profit, from [7]. In three out of four categories, knowing more about a user's shopping intent can be used to improve merchant profit.

As we saw in Chapters 2 and 3, e-commerce systems can be thought of as a generator of clickstream data - a log of {item - userid - action} tuples which captures user interactions with the system. A chronological grouping of these tuples by user ID is commonly known as a *session*.

Predicting a users intent to purchase is more difficult than ranking content for the following reasons [23]: Clickers (users who only click and never purchase within a session) and buyers (users who click and also purchase at least one item within a single session) can appear to be very similar, right up until a purchase action occurs. Additionally, the ratio between clickers and buyers is always heavily imbalanced - and can be 20:1 in favour of clickers or higher [83]. An uninterested user will often click on an item during browsing as there is no cost to doing so - an uninterested user will not *purchase* an item however. In our opinion, this user behaviour is in stark contrast to other settings such as predicting if a user will "like" or "pin" a piece of content hosted on a social media platform after viewing it, where there is no monetary amount at stake for the user. As

noted in [60], shoppers behave differently when visiting online vs physical stores and online conversion rates are substantially lower, for a variety of reasons.

When a merchant has increased confidence that a subset of users are more likely to purchase, they can use this information in the form of proactive actions to maximise conversion and yield. The merchant may offer a time-limited discount, spend more on targeted (and relevant) advertising to re-engage these users, create bundles of complementary products to push the user to complete their purchase, or even offer a lower-priced own-brand alternative if the product is deemed to be fungible.

However there are counterweights to the desire to create more and more accurate models of online user behaviour - namely user privacy and ease of implementation. Users are increasingly reluctant to share personal information with online services, while complex machine learning models are difficult to implement and maintain in a production setting [105].

We surveyed existing work in this area [55, 56, 106–108] and found that well-performing approaches have a number of factors in common:

- Heavy investment in dataset-specific feature engineering was necessary, regardless of the model implementation chosen.
- Model choices favour techniques such as Gradient Boosted Machines (GBM) [106] and Field-aware Factorisation Machines (FFM) [107] which are well-suited to creating representations of semi-structured clickstream data once good features have been developed [55, 56, 108].

In Chapter 4, an important feature class employed the notion of item similarity, modelled as a learned vector space generated by word2vec [41] and calculated using a standard pairwise cosine metric between item vectors. In an e-commerce context, items are more similar if they co-occur frequently over all user sessions in the corpus and are dissimilar if they infrequently co-occur. The items themselves may be physically dissimilar (for example - headphones and batteries), but they are often browsed and purchased together.

However, in common with other work our model in Chapter 4 still requires a heavy investment in feature engineering. The drawback of specific features is how tied they are to either a domain, dataset or both. The ability of deep learning models to discover good representations without explicit feature engineering is well-known [109]. In addition, artificial neural networks (ANNs) perform well with distributed representations such as embeddings, and ANNs with a recurrence capability to model events over time - Recurrent Neural Networks (RNNs) - are well-suited to sequence processing and labelling tasks [110].

Our motivation then is to build a good model of user intent prediction which does not rely on private user data, and is also straightforward to implement in a real-world environment. We address the following issues:

- What performance can RNNs with an appropriate input representation and end-to-end training regime achieve on the prediction of purchasing intent task?

- Can this performance be achieved within the constraint of only processing anonymous session data and remaining straightforward to implement on other e-commerce datasets?

5.2 Our Approach

As demonstrated in the previous chapter, classical machine learning approaches such as GBM work well and are widely used on e-commerce data, at least in part because the data is *structured*. GBM is an efficient model as it enables an additive expansion in a set of basis functions or weak learners to continually minimise a residual error. The weakness of GBM is a propensity for overly wide or deep decision trees to overfit the training data and thus record poor performance on the validation and test set due to high variance. GBM also requires significant feature engineering effort and does not naturally process the sequence in order, rather it consumes a compressed version of it (although it is possible to provide a one-hot vector representation of the input sequence as a feature). Our approach is dual in nature - firstly we construct an input representation for clickstream / session data that eliminates the need for feature engineering. Second, we design a model which can consume this input representation and predict user purchase intent in an end-to-end, sequence to prediction manner.

5.2.1 Embeddings as item / word representations

Natural Language Processing (NLP) tasks, such as information retrieval, part-of-speech tagging and chunking, operate by assigning a probability value to a sequence of words.

To this end, language models have been developed, defining a mathematical model to capture statistical properties of words and the dependencies among them. Noting the parallels between language modelling and user intent prediction with documents=clickstream sessions and words=items, we adapted a word language model [111] to our target task.

Learning good representations of input data is a central task in designing a machine learning model that can perform well. An *embedding* is a vector space model where words are converted to a low-dimensional vector. Vector space models embed words where semantically similar words are mapped to nearby points. Popular generators of word to vector mappings such as [41] and [112], operate in an unsupervised manner - predicting similarity or minimising a perplexity metric using word co-occurrence counts over a target corpus. We decided to employ embeddings as our target representation since:

- We can train the embeddings layer at the same time as training the model itself - promoting simplicity.
- E-commerce data is straightforward to model as a dictionary of words.
- Embedding size can be increased or decreased based on dictionary size and word complexity during the architecture tuning / hyper parameter search phase.

Unlike [41] and [112], we chose not to pre-train the embeddings to minimise a perplexity error measure. We observed that our model was less accurate under this regime (which also required embedding weights to be frozen at training time).Instead we allow the model to modify the embedding weights at training time by back-propagating the

loss from a binary classification criterion. Algorithm 1 outlines how embeddings are created from the input data. A concrete example serves to illustrate how the algorithm operates: if we consider the item ID 214536502 in our session from Chapter 4, then 214536502 is mapped onto the item lookup ID 1 and this ID is linked to the embedding $[-0.075, \ldots, +0.075]$.

Algorithm 1 The embedding creation algorithm.

1: **for all** $fields \in S$ **do**
2: $id_i \leftarrow unique(field^i_{type})$ *//Return a unique lookup ID for the input field. IDs are unique per type (datetime quantile, item ID, item category).*
3: $embedding_i \leftarrow uniform(min, max, size)$ *//An initialised 1-D vector of the specified* $size$ *drawn from a normal distribution bounded by* min *and* max*.*
4: $save(id_i, embedding_i)$ *//Link and store the unique ID - embedding pair for future use.*
5: **end for**

5.2.2 Recurrent Neural Networks

Recurrent neural networks [40] (RNNs) are a specialised class of neural networks for processing sequential data. A recurrent network is deep in *time* rather than space and arranges hidden state vectors h^l_t in a two-dimensional grid, where $t = 1 \ldots T$ is thought of as time and $l = 1 \ldots L$ is the depth. All intermediate vectors h^l_t are computed as a function of h^l_{t-1} and h^{l-1}_t. Through these hidden vectors, each output y at some particular time step t becomes an approximating function of all input vectors up to that time, $x_1, \ldots, x_t$ [5].

5.2.2.1 LSTM and GRU

Long Short-Term Memory (LSTM) [113] is an extension to standard RNN units designed to address the twin problems of vanishing and exploding gradients during training [114]. Vanishing gradients make learning difficult as the correct (downward) trajectory of the gradient is difficult to discern, while exploding gradients make training unstable - both are undesirable outcomes. Long-term dependencies in the input data, causing a deep computational graph which must iterate over the data are the root cause of vanishing / exploding gradients. Goodfellow et al. explains this phenomenon succinctly. Like all deep learning models, RNNs require multiplication by a matrix W. After t steps, this equates to multiplying by W^t as shown in Equation 5.1 [109].

$$W^t = (Vdiag(\lambda)V^{-1})^t = Vdiag(\lambda)^t V^{-1} \tag{5.1}$$

Eigenvalues (λ) that are not more or less equal to 1 will either explode if they are > 1, or vanish if they are < 1 as $t \to \infty$. Gradients will then be scaled by $diag(\lambda)^t$ and are similarly affected, tending towards ∞ for very large values of $diag(\lambda)^t$ or approaching 0 for very small values.

LSTM solves this problem by possessing an internal recurrence, which stabilises the gradient flow, even over long sequences. However this comes at a price of complexity. For each element in the input sequence, each layer computes the following function as shown in equation 5.2.

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hc}h_{(t-1)} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \\
c_t &= f_t * c_{(t-1)} + i_t * g_t \\
h_t &= o_t * \tanh(c_t)
\end{aligned}
\tag{5.2}
$$

where:

h_t is the hidden state at time t,

c_t is the cell state at time t,

x_t is the hidden state of the previous layer at time t or $input_t$ for the first layer,

i_t, f_t, g_t, o_t are the input, forget, cell, and out gates, respectively,

σ is the sigmoid function.

Figure 5.1 provides a graphical representation of a single LSTM cell and its components.

Gated Recurrent Units, or GRU [115] are a simplification of LSTM, with one less gate and the hidden state and cell state vectors combined. In practice, both LSTM and GRU are used interchangeably and the performance difference between both cell types is often minimal and / or dataset-specific.

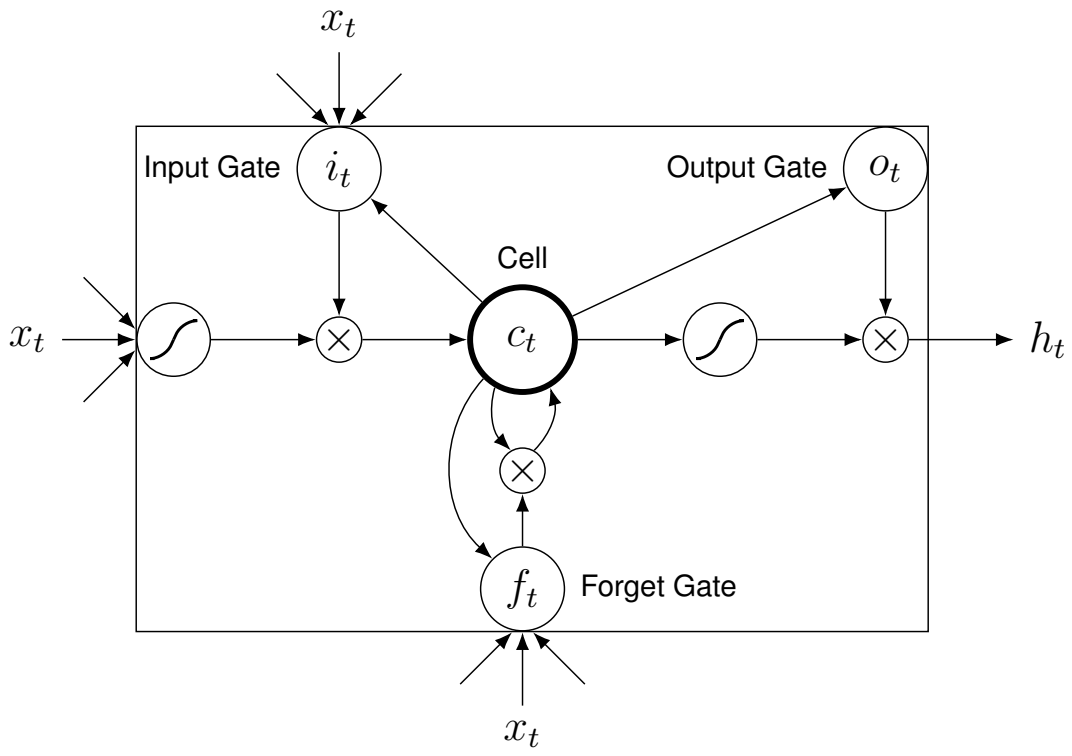

FIGURE 5.1: A single LSTM cell, depicting the hidden and cell states, as well as the three gates controlling memory (input, forget and output).

5.3 Implementation

We prepared each column in the dataset as follows:

- Session IDs are discarded (of course we retain the sequence grouping indicated by the IDs).
- Timestamps are quantised into bins of 4 hours in duration.
- Item IDs are unchanged.
- Category IDs are unchanged.
- Purchase prices are unchanged. We calculate price variance per item to convey price movements to our model (e.g. a merchant special offer).

- Purchase quantities are unchanged.

Each field is then converted to an embedding vocabulary - simply a lookup table mapping values to integer IDs. We do not impose a minimum occurrence limit on any field - a value occurring even once will be represented in the respective embedding. This ensures that even long tail items will be presented to the model during training. Lookup tables are then converted to an embedding with embedding weights initialised from a range $\{$-0.075, +0.075$\}$ - Table 5.2 identifies the number of unique items per embedding and the width used. The testing dataset contains both item IDs and category IDs that are not present in the training set - however only a very small number of sessions are affected by this data in the test set.

This approach, combined with the use of Artificial Neural Networks, provides a learnable capacity to encode more information than just the original numeric value. In our opinion for example, an item price of $100 vs $150 is not simply a numeric price difference, it can also signify learnable information on brand, premium vs value and so on.

Data name	Train	Train+Test	Embedding Width
Item ID	52,739	54,287	100
Category ID	340	348	10
Timestamp	4,368	4,368	10
Price	667	667	10
Quantity	1	1	10

TABLE 5.2: Data field embeddings and dimensions, along with unique value counts for the training and test splits of the RecSys 2015 dataset.

Dataset	Events before	Events after	% increase
RecSys 2015	41,255,735	56,059,913	36%
Retailrocket	2,351,354	11,224,267	377%

TABLE 5.3: Effect of unrolling by dwelltime on the RecSys 2015 and Retailrocket datasets. There is a clear difference in the mean / median session duration of each dataset.

5.3.1 Event Unrolling

In [67], a more explicit representation of user dwelltime or interest in a particular item i_k in a sequence $e_{i_1}, \ldots, e_{i_k}$ is provided to the model by repeating the presentation of the event containing the item to the model in proportion to the elapsed time between e_{i_k} and $e_{i_{k+1}}$. In the example 2-event session displayed previously, the elapsed time between the first and second event is 6 minutes, therefore the first event is replayed 3 times during training and inference ($\lceil 360/150 \rceil$). In contrast to [67], we found that session unrolling provided a smaller improvement in model performance - for example on the RecSys 2015 dataset our best AUC increased from 0.839 to 0.841 when using the optimal unrolling value (which we discovered empirically using grid search) of 150 seconds. Unrolling also comes with a significant cost of increasing session length and thus training time - Table 5.3 demonstrates the effect of session unrolling on the size of the training / validation and test sets on both datasets.

5.3.2 Sequence Reversal

From [23], we know that the most important item in a user session is the last item (followed by the first item). We also know that sequence reversal has been reported to improve model performance in the sequence to sequence translation setting [35]. To

capitalise on this, we reversed the sequence order for each session before presenting them as batches to the model. Re-ordering the sequences provided an increase in test AUC on the RecSys 2015 dataset of 0.005 - from 0.836 to 0.841.

5.3.3 Model

5.3.3.1 Model Architecture

The data embedding modules are concatenated and presented to a configurable number of RNN layers (typically 3), with a final linear layer combining the output of the hidden units from the last layer. This output value represents the models confidence probability in class membership. Figure 5.2 illustrates the model architecture.

Given that we wish to distinguish user intent into two main classes, the model is trained by minimising an unweighted binary cross entropy loss as shown in Equation 5.3. Although the classes are heavily imbalanced, we do not rescale the weight of the under-represented class to compensate as we found this approach slightly reduced performance.

$$l_n = -\left[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)\right] \tag{5.3}$$

where:

x_n is the output label value from the model [0..1]

y_n is the target label value $\{0, 1\}$.

We conducted a grid search over the number of layers and layer size by RNN type, as indicated in Table 5.4 below. The State of the Art baseline for comparison is 0.853. For

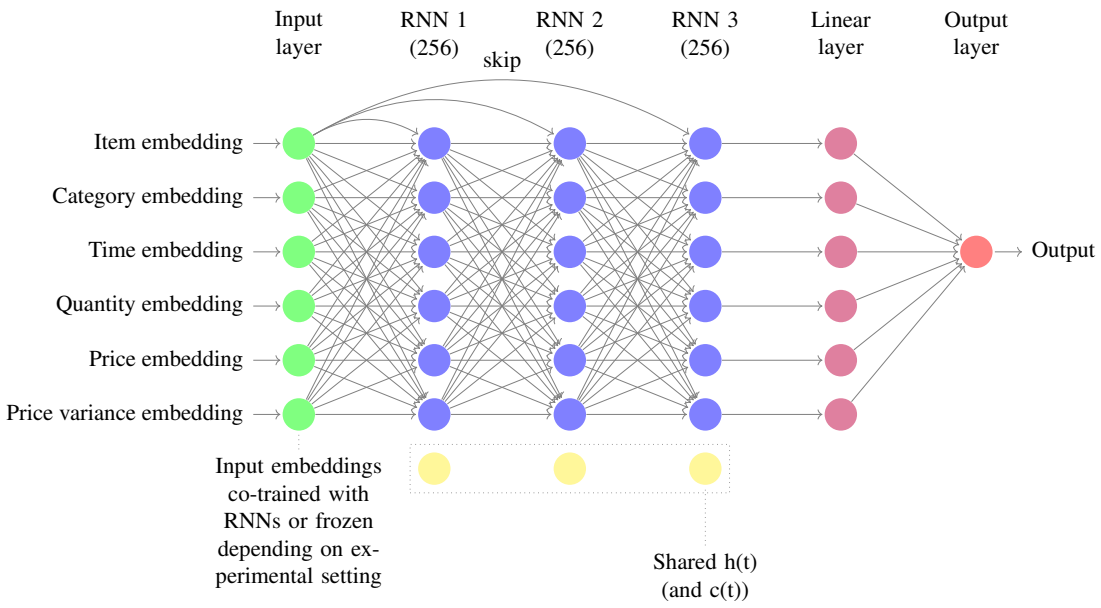

FIGURE 5.2: Model architecture used - the output is interpreted as the log probability that the input represents either a clicker or buyer session.

all cell values, we trained the model in question for 4 epochs. For the standard RNN model variant with 512 cells, training became unstable after 2 epochs, which explains why these AUC values are slightly lower than expected. We limited the layer search to 3 as we noted no discernible improvement above this level.

	RNN (RELU)			GRU			LSTM		
Layers	1	2	3	1	2	3	1	2	3
Layer size									
64	0.828	0.835	0.836	0.830	0.834	0.836	0.829	0.835	0.836
128	0.831	0.835	0.838	0.832	0.837	0.838	0.833	0.838	0.838
256	0.831	0.836	0.838	0.835	0.838	0.839	0.835	0.840	0.840
512	0.828	0.834	0.834	0.836	0.839	0.839	0.836	0.840	**0.841**

TABLE 5.4: Model grid search results for number and size of RNN layers by RNN type on the RecSys 2015 dataset.

5.3.3.2 Skip Connections

Skip connections are a device used to preserve inputs across model layers and reduce signal attenuation during training. Li et al. show that skip connections simplify the loss surface for some tasks, enabling quicker convergence at training time. In essence, we are re-presenting the original inputs to each successive model layer along with the output from the previous layer. The intuition here is that the model is more easily able to solve the objective task by seeing both the original inputs as well as per-layer abstractions at the same time.

5.3.3.3 Sharing Hidden Layer State Across Batch Boundaries

One model design decision worthy of elaboration is how hidden state information (and cell state for LSTM) is shared between training batches. We found that best results were obtained by not re-using any hidden state across RNN layers, in conjunction with a randomised sampler for selecting batch candidates. For e-commerce datasets, trying to connect batches together by sharing hidden state and deploying chronological / sequential sampling is not the best training approach to use - indeed far from it. This insight led to a very significant improvement in AUC, increasing from 0.75 to 0.84 and this finding is expanded on in Chapter 6.

5.4 Experiments and results

In this section we describe the experimental setup, and the results obtained when comparing our best RNN model to GBM, and a comparison of different RNN variants (standard, GRU, LSTM).

5.4.1 Training Details

Both datasets were split into a training set and validation set in a 90:10 ratio. We chose this ratio empirically to maximise the number of buyer sessions presented to the model at training time as buyer sessions are relatively rare compared to clicker sessions. The model was trained using the Adam optimiser [117], coupled with a binary cross entropy loss metric and a learning rate annealer. Training was halted after 2 successive epochs of worsening validation AUC. Table 5.5 illustrates the main hyperparameters and setting used during training.

Dataset split	$90/10$ (training / validation)
Hidden units range	$128 - 512 : 512$ optimal
Embedding width	$10 - 300 : 100$ optimal for items
Embedding weight	-0.075 to $+0.075$
Batch size	$32 - 256 : 256$ optimal for speed and regularisation
Optimiser	Adam, cyclic learning rate (1-cycle policy: $1e^{-5}$ - $1e^{-3}$)

TABLE 5.5: Hyper parameters and setup employed during model training.

We tested three main types of recurrent cells (standard RNN, GRU, LSTM) as well as varying the number of cells per layer and layers per model. While a 3-layer LSTM achieved the best performance, standard RNNs which possess no memory mechanism are able to achieve a competitive AUC. The datasets are heavily weighted towards

shorter session lengths (even after unrolling - see Figure 5.4 and 5.9). We posit that the power of LSTM and GRU is not needed for the shorter sequences of the dataset, and standard recurrence with embeddings has the capacity to model sessions over a short enough sequence length.

5.4.2 Overall results

The metric we used in our analysis was Area Under the ROC Curve or AUC. AUC is insensitive to class imbalance, and also the raw predictions from [55] were available to us, thus a detailed, like-for-like AUC comparison using the test set is the best model comparison. The organisers of the challenge also released the solution to the challenge, enabling the test set to be used. After training, the LSTM model AUC obtained on the test set was 0.841 - 98.6% of the AUC (0.853) obtained by the state-of-the-art (SotA) model. As the subsequent experiments demonstrate, a combination of feature embeddings and model architecture decisions contribute to this performance. For all model architecture variants tested (see table 5.4), the best performance was achieved after training for a small number of epochs (2 - 3). This held true for both datasets.

Our LSTM model achieved within 98% of the winning GBM-based model on the RecSys 2015 dataset, and outperformed our GBM model by 1.1% on the Retailrocket dataset, as Table 5.6 shows.

	RecSys 2015	Retailrocket
LSTM	0.841	**0.843**
GBM	**0.853**	0.834

TABLE 5.6: Classification performance measured using Area under the ROC curve (AUC) of the GBM and LSTM models on the RecSys 2015 and Retailrocket datasets.

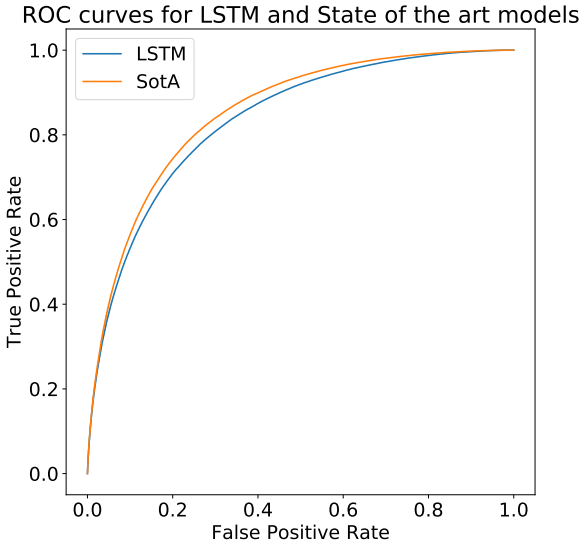

FIGURE 5.3: ROC curves for the LSTM and State of the Art models - on the RecSys 2015 test set.

5.5 Analysis

We constructed a number of tests to analyse model performance based on subsets of the test data where we can reasonably expect a divergence in model performance (for example sessions with longer item dwelltimes could favour RNNs over GBM).

5.5.1 Session length

Figure 5.4 graphs the best RNN model (a 3-layer LSTM with 256 cells per layer) and the SotA model, with AUCs broken down by session length. For context, the quantities for each session length in the test set is also provided. Both models underperform for sessions with just one click - clearly it is difficult to split clickers from buyers with such a small input signal. For the remaining session lengths, the relative model performance

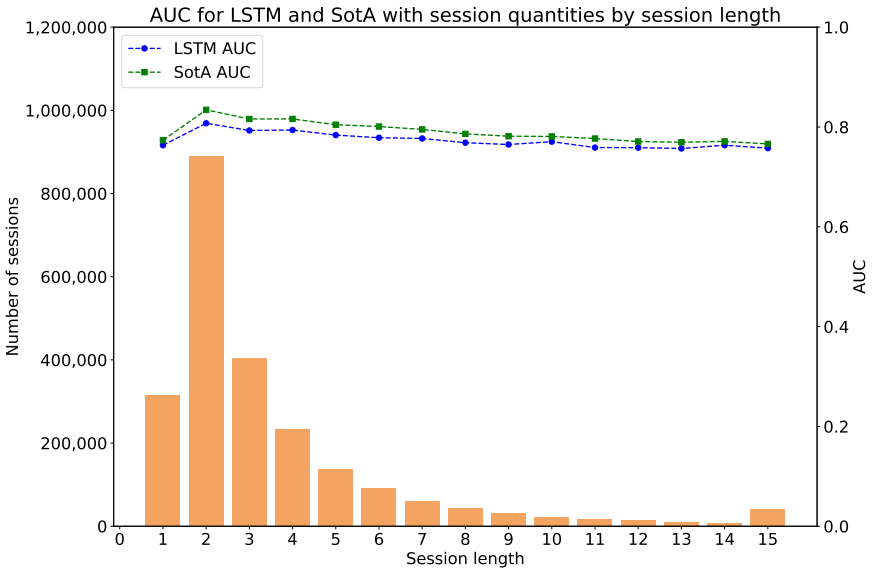

FIGURE 5.4: AUC by session length for the LSTM and SotA models, session quantities by length also provided for context - clearly showing the bias towards short sequence / session lengths in the RecSys 2015 dataset.

is consistent, although the LSTM model does start to close the gap after sessions with length > 10.

5.5.2 User dwelltime

Given that we unrolled long-running events in order to provide more input to the RNN models, we evaluated the relative performance of each model when presented with sessions with any dwelltime > 1. As Figure 5.5 shows, LSTM is closer to SotA for this subset of sessions and indeed outperforms SotA for session length $= 14$, but the volume of sessions affected $(5{,}433)$ is not enough to materially impact the overall AUC.

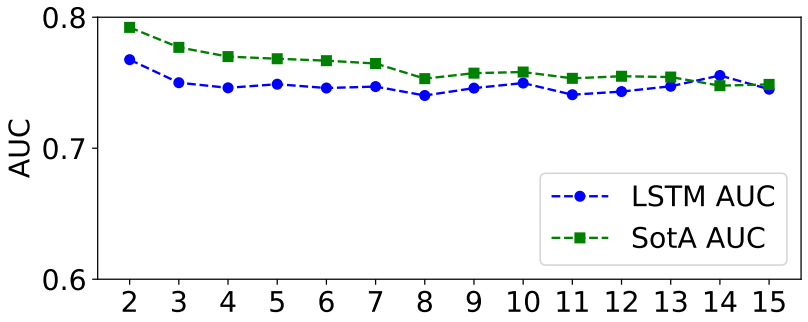

FIGURE 5.5: AUC (y-axis) by session length (x-axis) for the LSTM and SotA models, for any sessions where unrolling by dwelltime was employed. There are no sessions of length 1 as unrolling is inapplicable for these sessions (RecSys 2015 dataset).

5.5.3 Item price

Like most e-commerce catalogues, the catalogue under consideration here displays a considerable range of item prices. We first selected all sessions where any single item price was $> 10{,}000$ (capturing $544{,}014$ sessions) and then user sessions where the price was $<= 750$ (roughly 25% of the maximum price - capturing $1{,}063{,}034$ sessions). Figures 5.6 and 5.7 show the relative model performance for each session grouping. As with other selections, the relative model performance is broadly consistent - there is no region where LSTM either dramatically outperforms or underperforms the SotA model.

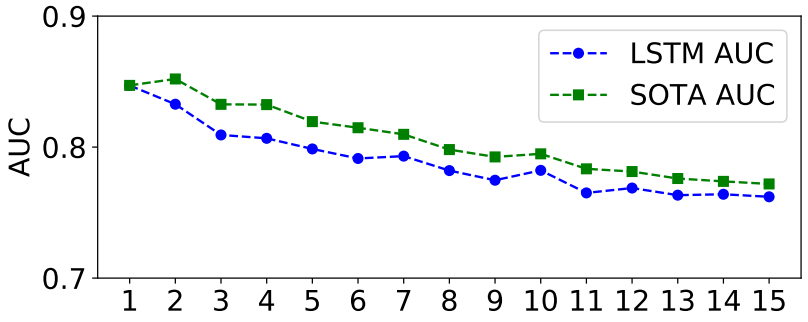

FIGURE 5.6: Model performance (AUC - y-axis) for sessions containing low price items, split by session length (x-axis) using the RecSys 2015 dataset.

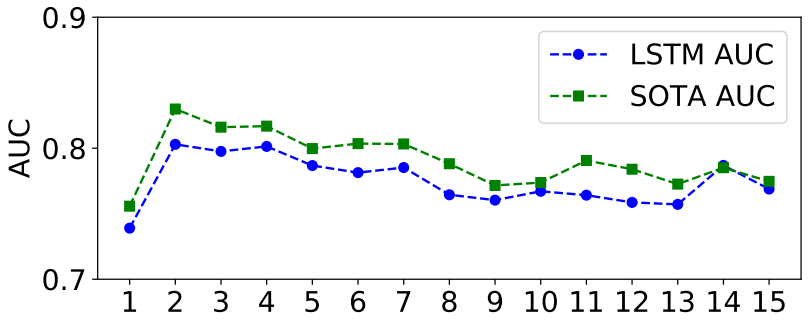

FIGURE 5.7: Model performance (AUC - y-axis) for sessions containing high price items, split by session length (x-axis) using the RecSys 2015 dataset.

5.5.4 Gated vs un-gated RNNs

Table 5.4 shows that while gated RNNs clearly outperform ungated RNNs, the difference is 0.02 of AUC which is less than might be expected. We believe the reason for this is that the dataset contains many more shorter (< 5) than longer sequences. This is to be expected for anonymised data - user actions are only aggregated based on a current session token and there is no lifetime set of user events. For many real-world cases then, using ungated RNNs may deliver acceptable performance.

5.5.5 End-to-end learning

To measure the effect of allowing (or not) the gradients from the output loss to flow unencumbered throughout the model (including the embeddings), we froze the embedding layer so no gradient updates were applied during the back-propagation phase and then trained the network. Model performance decreased to an AUC of 0.808 and training time increased by 3x to reach this AUC. In addition the number of trainable model parameters also reduces significantly. This demonstrates that depriving the model of the

ability to dynamically modify the input data representation *at training time* using gradients derived from the output loss metric reduces its ability to solve the classification problem posed.

5.5.6 Transferring to another dataset

We tested our claim that learned features should be more robust than hand-engineered features when ported to a new dataset [98] as learned features are not dataset or domain-specific. The GBM model used in [55] is not available, however we were able to use the GBM model described in [23]. Figure 5.8 shows the respective ROC curves for the RNN (LSTM) and GBM models when they are ported to the Retailrocket dataset. Both models still perform well on the second dataset, however the LSTM model out-performs the GBM model (mean AUC of 0.8434 vs 0.8338) by a clear and reproducible margin. We ran 60 experiments using different initial random seeds (30 GBM, 30 LSTM) and measured the final test AUC for each model. A two-tailed t-test was carried out on the data in Table 5.7 which indicated that the LSTM algorithm gave significantly better performance than GBM ($t = -20.372$, $df = 32.998$, $p < 2.2e^{-16}$).

A deeper analysis of the Area under the ROC curve demonstrates how the characteristics of the dataset can impact on model performance. The Retailrocket dataset is heavily weighted towards single-click sessions as Figure 5.9 shows. LSTM out-performs GBM for these sessions - which can be attributed more to the learned embeddings since there is no sequence to process. GBM by contrast can extract only limited value from single-click sessions as important feature components such as dwelltime, similarity etc. are unavailable.

Experiment	GBM (AUC)	LSTM (AUC)
1	0.8344	0.8430
2	0.8349	0.8424
3	0.8342	0.8432
4	0.8342	0.8451
5	0.8340	0.8464
6	0.8344	0.8409
7	0.8336	0.8427
8	0.8339	0.8381
9	0.8325	0.8494
10	0.8338	0.8434
11	0.8335	0.8476
12	0.8341	0.8440
13	0.8336	0.8427
14	0.8336	0.8453
15	0.8346	0.8398
16	0.8330	0.8429
17	0.8338	0.8456
18	0.8333	0.8432
19	0.8337	0.8428
20	0.8340	0.8431
21	0.8333	0.8436
22	0.8338	0.8414
23	0.8332	0.8448
24	0.8349	0.8377
25	0.8349	0.8428
26	0.8337	0.8424
27	0.8331	0.8421
28	0.8324	0.8432
29	0.8346	0.8455
30	0.8343	0.8464

TABLE 5.7: AUC comparisons for $n = 30$ experiments carried out using random seeds for the GBM and LSTM models on the Retailrocket dataset.

5.5.7 Training time and resources used

We used PyTorch [94] to construct and train the LSTM models while XGBoost [28] was used to train the GBM models. The LSTM implementation was trained on a single Nvidia GeForce GTX TITAN Black (circa 2014 and with single-precision performance of 5.1 TFLOPs vs a 2017 GTX 1080 Ti with 11.3 TFLOPs) and consumed between 1

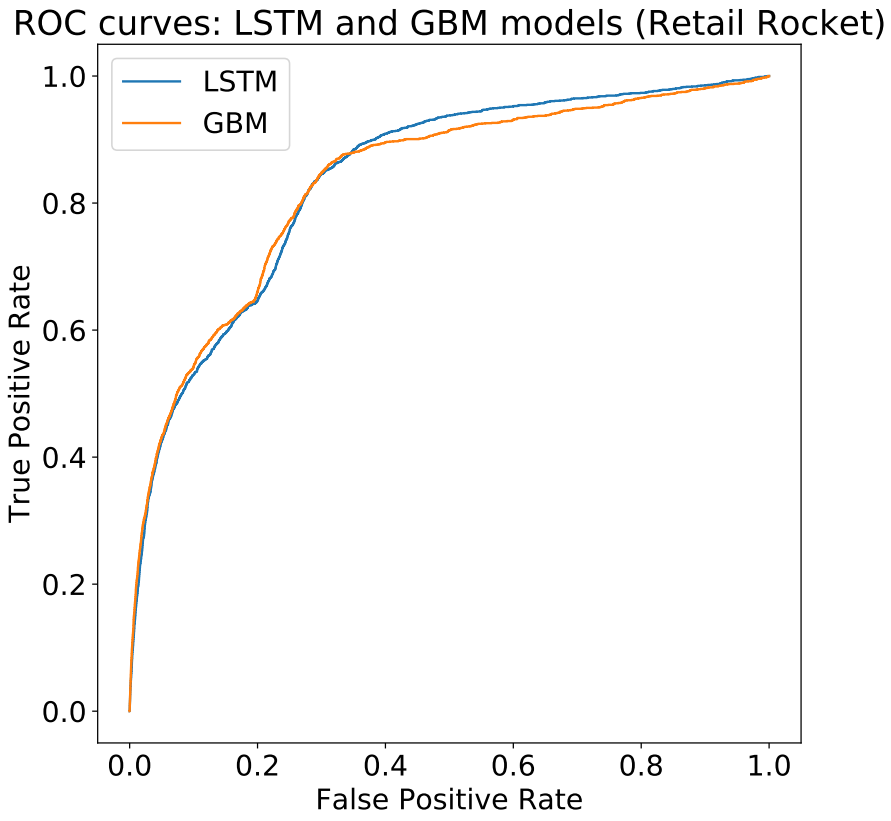

FIGURE 5.8: Area under ROC curves for LSTM and GBM models when ported to the Retailrocket dataset. On this dataset, the LSTM model slightly outperforms the GBM model overall.

and 2 GB RAM. The results reported were obtained after 3 epochs of training on the full dataset - an average of 6 hours (2 hours per epoch). This compares favourably to the training times and resources reported in [55], where 150 machines were used for 12 hours. However, 12 hours denotes the time needed to train two models (session purchase and item ranking) whereas we train just one model. While we cannot compare GBM (which was trained on a CPU cluster) to RNN (trained on a single GPU with a different parallelisation strategy) directly, we note that the hardware resources required for our model are modest and hence accessible to most commercial or academic labs. In

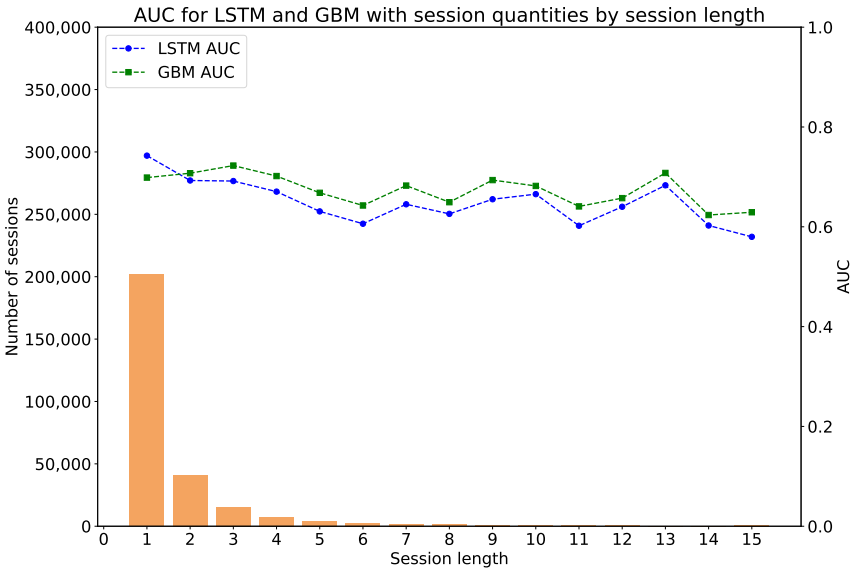

FIGURE 5.9: AUC by session length for the LSTM and GBM models when tested on the Retailrocket dataset. The bias towards shorter sessions is even more prevalent versus the RecSys 2015 dataset.

addition, real-world e-commerce datasets are large (thousands of concurrent users generating billions of events over time) [118] and change rapidly, therefore usable models must be able to consume large datasets and be re-trained readily to cater for new items / documents.

5.6 Interpretability

The RNN / LSTM model under discussion has low interpretability, whereas the GBM model admits at least some introspection of the decision tree created. In particular, the importance of the different features can be gauged, and the ranking of features for different datasets can also be analysed.

Understanding model behaviour is fast becoming a key requirement before deploying to a production setting - good performance on a test set is no longer enough. We used LIME [119] to construct simpler proxy models for our deep learning model, focusing on 4 specific data points of interest: a true positive, true negative, false positive and finally a false negative example. In this way we endeavour to understand what aspects of the input data cause the model to behave correctly and incorrectly.

Local Interpretable Model-Agnostic Explanations (LIME) [119] is a technique completely external to a target model which aims to improve the interpretability of highly complex models such as RNNs. The core idea of LIME (not justified formally in the LIME literature) is that even the most complex model can be reasonably approximated locally using a simpler model.

LIME generates an explanation by approximating the underlying model by an interpretable one (such as a linear model with only a few non-zero coefficients), learned by modifying the original instance (in our case by removing words from session events which are then replaced with a data not present value). The key intuition behind LIME is that it is much easier to approximate a black-box model by a simple model locally, instead of trying to approximate a very complex model globally.

For our particular model which is an RNN that consumes word embeddings as input, LIME works as follows:

1. Generate 1 seed output from RNN for a given input (so output $\hat{y}$ for input x).

2. Pass the original string representation of the user session x to LIME (i.e. not the embedding lookup IDs) which generates n perturbed samples. We used $n =$

2048, the default is $5,000$ however using this value increases the run-time of LIME significantly and we observed no noticeable reduction in output when using the lower value. LIME does warn if n is set too low as its own internal matrices become ill-conditioned.

3. For each sample in n, translate the sample back to embedding IDs and present this new, perturbed example to our RNN and return the output value to LIME.

4. LIME calculates a similarity metric for the n samples and corresponding RNN output values according to their proximity to the original input instance.

5. m features (we used $m = 10$ to maintain a balance between explanation complexity and power) are selected which best describe or cover the original model behaviour.

6. These m features are used to construct and fit a simpler model (by default LIME uses ridge regression to construct a linear model).

7. The simpler model feature weights are then used in the local behaviour explanation.

LIME is not without disadvantages however. As well as the direct overhead of LIME itself, the model under analysis will be invoked n times to generate outputs for the perturbed examples. Also LIME does not interpret the model under analysis directly since it is a model-agnostic approach. The simpler proxy model that LIME generates is intended to provide textual or visual artifacts that provide links between the input data (the user clickstream) and the model prediction of user intent, as opposed to a true equivalent to our original model.

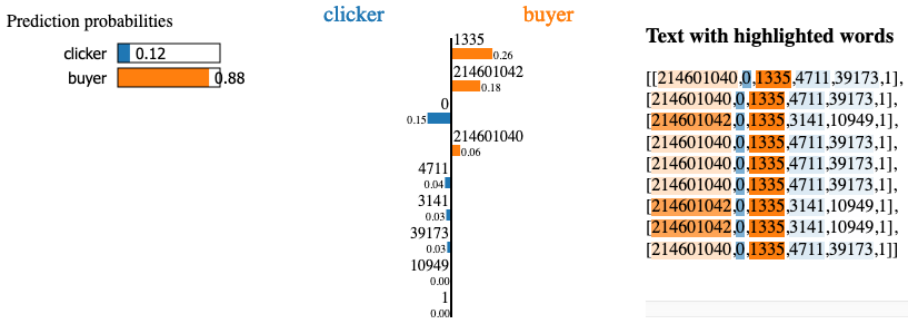

FIGURE 5.10: Strong true positive example: the model correctly has very high confidence (probability = 0.88) that the user has a purchase intention. The items clicked (214601040 and 214601042) also have a higher than average buy likelihood at 6% and 8.1% respectively.

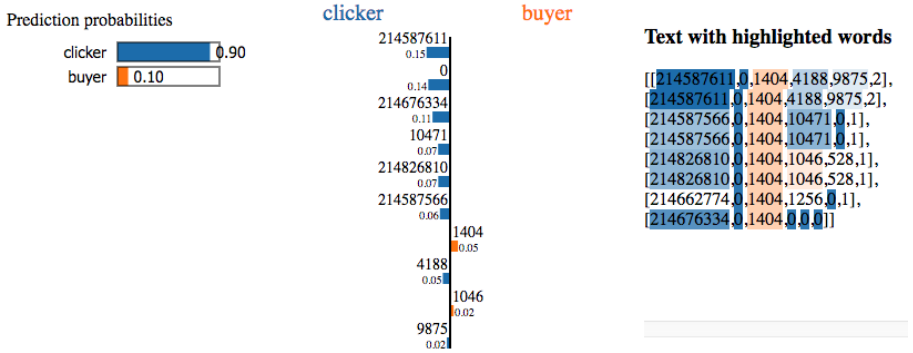

FIGURE 5.11: Weak true positive example: the model is much less sure (probability = 0.1) that the session ends with a purchase. The items clicked are not often purchased, so the model has to rely on the day / time of day and price embeddings instead.

Figures 5.10, 5.11, 5.12, 5.13 and 5.14 provide illustrations of our deep model behaviour for specific inputs. All of the sample sessions were drawn from the test set. In reading all of the figures, please note that each session is comprised of one or more sentences (separated by a newline) and each sentence is comprised of: item, category, datetime quantile, price, price variance and quantity - all separated by the ‘,’ character. For each example, we provide statistics drawn from the dataset to support our interpretation of the LIME result. Words highlighted in orange contribute to a prediction that the user has a buyer intent, whereas words in blue are associated with a clicker intent.

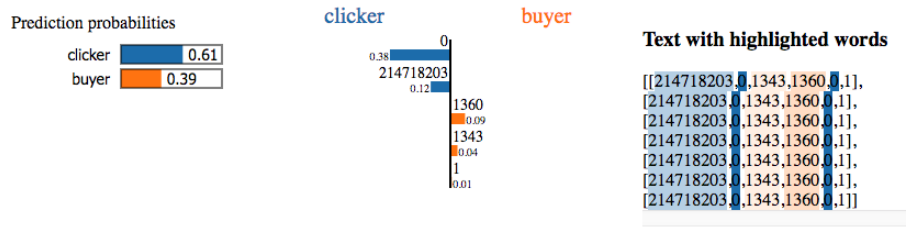

FIGURE 5.12: Strong false positive example: the model incorrectly (probability = 0.98) classified this session as a buyer. The model was relying on a reasonable item price combined with a time of day when buy sessions are more likely to make its incorrect prediction.

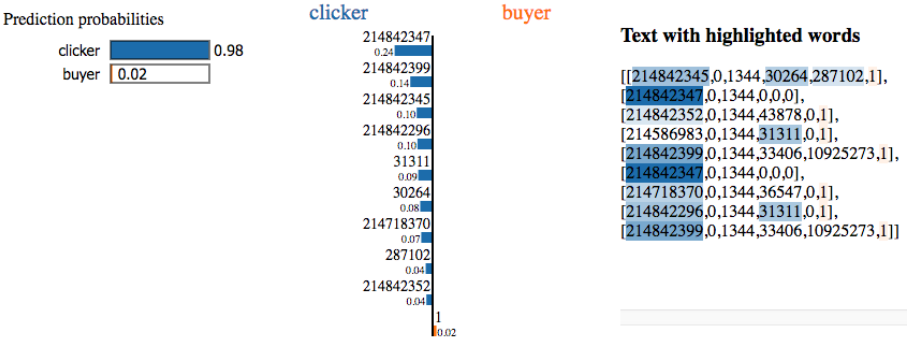

FIGURE 5.13: Strong true negative example: the model relies on the presence of unpopular items. For example, item 214842347 occurs only 1,462 times in the entire clickstream (33 million clicks) and is never purchased.

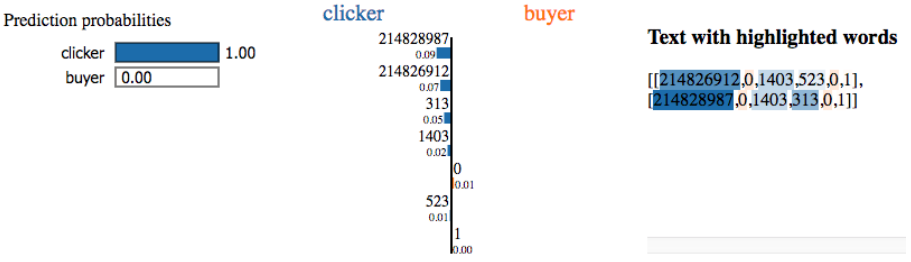

FIGURE 5.14: Strong false negative example: this session is a false negative example (probability = 0.001) - where the session ends with a purchase but our model failed to detect this. This session appears to show an inability of the model to detect long tail buy sessions - sessions containing items that are indeed purchased, but only weakly over the entire dataset. In this example, item 214826912 has a buy:click ratio of 363:21201 or a purchase rate of 1.7%. The corresponding statistics for item 214828987 are 592:23740 and 2.49%.

5.7 Conclusions and future work

We presented a Recurrent Neural Network (RNN) model which recovers 98.4% of the best GBM model performance on the user purchase prediction problem in e-commerce without using explicit features. On a second dataset, our RNN model fractionally exceeds the performance of our GBM model. The model is straightforward to implement, generalises to different datasets with comparable performance and can be trained with modest hardware resource requirements.

We applied LIME to generate interpretable models explaining how our model behaves in practice for selected data points of interest.

It is promising that gated RNNs with no feature engineering can be competitive with Gradient Boosted Machines on short session lengths and structured data - GBM is a more established model choice in the domain of recommender systems and e-commerce in general. We believe additional work on input representation (while still avoiding feature engineering) can further improve results for both gated and non-gated RNNs.

Our approach is transductive - in other words our model will perform poorly when presented with items never seen in the training set. In Chapter 9 we outline further planned improvements to the model including a proposal to address this limitation.

A surprising finding from this chapter is that the standard configuration of RNN-based word language models perform poorly when naively applied to the task of e-commerce user intent prediction. In Chapter 6 we examine this phenomenon more closely in order to document the best strategy for RNN usage for the buyer prediction task.

The material in this chapter draws substantially from [24], published at the ECOM workshop held during the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval.

Chapter 6

Hidden State Management and Sampling

In the previous chapter, we described how a deep learning model was constructed to compete with a Gradient Boosted Machine (GBM) model in discovering user intent in an e-commerce setting. During our design and implementation of this deep learning model (LSTM), we noted that the accuracy of our model suffered when we used perceived best practices around batch management and the training sampler used - in particular how *hidden state* is either re-used or discarded across batch boundaries. In this chapter, we provide more details about our findings and subsequent recommendations in this area - particularly relevant for clickstream analysis.

6.1 Introduction

Recurrent Neural Networks (RNNs) are one of the most common forms of neural networks in use today. They are well-suited to sequence processing tasks such as language modelling, tagging, translation and image captioning [110].

In Chapter 5, we applied multiple variants of the base RNN model to the problem of analysing user *clickstream* data in order to predict user intent in an e-commerce setting. In that work we tuned standard RNN hyperparameters such as dropout, batch size, learning rate, optimiser selection, sampling strategy, number and size of layers and also tested and employed less common techniques such as skip connections. However, implementing skip connections required us to apply fine grained control over each layer in the model, in particular how hidden state is passed between batches. We noticed that choosing whether or not to pass hidden state between batches had a substantial impact on model performance. In fact, deciding how to handle hidden state and order of presentation of examples to the model during training were the two most important design decisions contributing to the final predictive power of the model.

Although RNNs have been used recently to process clickstream sequences [110], more traditional work has utilised Gradient Boosted Machines (GBM) [106] and Field-aware Factorisation Machines (FFM) [120] to perform this analysis. Both of these approaches work well, especially when domain and dataset-specific features are used. Typical features constructed provide global context to the model, e.g. frequency-based measures of item popularity. RNNs by contrast do not receive this information explicitly, but can accrue it over time and training epochs.

Virtually all e-commerce systems can be thought of as a generator of clickstream data - a log of $\{item - userid - action\}$ tuples which captures user interactions with the system. A chronological set of these tuples grouped by user ID is commonly known as a **session**.

In an e-commerce context, we can think of local state as the individual story for a single user - their clicks including dwelltime (conveying interest) and items provide insight into their intent (buy vs browse). Additionally, we posit that there is a global state telling a second story about the dataset, over and above the first and most immediate local story encoded in each individual user session. Examples of global state are specific items going on sale for a short period of time and seasonality of particular items over weeks and months. It is intuitively appealing to think of both global and local patterns encoded in clickstream data that can be parsed and understood by LSTM to improve predictive performance. However, the inability of LSTM to handle very long sequences (and thus learn global patterns) is also well-known [121]. Our goal is to examine how well LSTM can learn global state in an e-commerce context.

Electing to pass hidden state between batches when training is, inasmuch as we can determine, the default setting for RNN word language models. RNN word language models are also frequently used as the starting point for new sequence processing models and applications. Colloquially, passing state is referred to as stateful LSTM while choosing not to pass state is known as stateless LSTM. However there do not appear to be any formal references to this model configuration in the literature, even as a tips and tricks entry. Moreover, these names are confusing as "stateless LSTM" would seem to indicate state is not maintained in the cells (and thus reduces to standard RNN) but

this is not the case. With stateless LSTM, the learned weights are retained during the training phase, but no knowledge is retained in the form of hidden state from previous batches.

Practitioners are also faced with another important decision - how to sample from the dataset at training time to maximise performance at inference time. In the Experiments section, we measure the performance of multiple RNN variants under different settings to propose the optimal RNN configuration for e-commerce clickstream analysis.

Sampling and hidden state strategy are very important to overall model performance. One example of the difference in model predictive power when hidden layer state is either re-used or discarded between batches is shown in Figure 6.1. When it is discarded, our best LSTM model is able to recover over 98% of the performance of a strong baseline - the GBM model for this task [55]. With re-use, the LSTM model is far less effective. Sharing hidden state and using sequential sampling represents current word language model best practices to extract maximum predictive accuracy from models. However, in an e-commerce clickstream setting, we find that it is more advantageous to use random sampling and to eliminate hidden state sharing across batches completely, since our modified model displays superior classification accuracy.

6.2 Related Work

This section focuses on two sub-problems:

- How to process and classify clickstreams effectively.

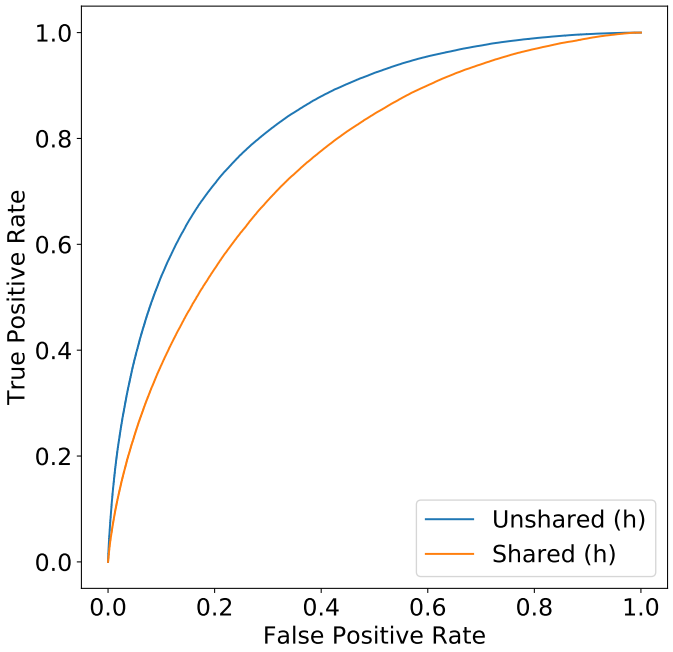

FIGURE 6.1: ROC curves for two LSTM models on the RecSys 2015 test set.

- The search for an optimal RNN model architecture.

The search for better LSTM model architectures is almost as old as LSTM itself [122], [123]. In [124], the authors found that the forget gate and the output activation function are the most critical components in the LSTM cell. In [125], the authors found that more advanced recurrent units (i.e. variants such as LSTM and GRU that incorporate gating mechanisms) regularly outperform standard recurrent units and that for the most part, LSTM and GRU provide equivalent performance. Ba et al. employs layer normalisation to reduce the training time for recurrent neural networks - an important goal given the inordinate time reported to train state-of-the-art models on larger datasets. In [127], regularisation is applied to RNNs stochastically, with the aim of improving generalisation.

That the performance of neural network-based models depends heavily on the ordering of input sequences is well known. Bengio et al. argue that careful selection of training samples can achieve better generalisation. In [129], optimal performance on a sequence generation task is achieved by preserving order during training, while [130] observes that faster convergence can be observed if batch order is randomised between epochs. These observations are seemingly at odds with each other, and serve to illustrate the requirement for domain and dataset-specific tuning of training algorithms.

6.3 Model Structure / Hidden layers

In deep learning, the term hidden layers is used to refer to any model layer where the training data does not stipulate the desired output for these layers. Instead the training algorithm is free to use these layers to construct the best approximation for the desired function f^* - in other words hidden layers contribute to the overall model capacity in the form of learnable / trainable parameters.

Recall from Chapter 5 the standard model under discussion here is as illustrated in figure 5.2 - independent of the recurrent cell type used. The cells are organised into three layers with 256 cells per layer. The first layer accepts input from a group of input embeddings (unique input values are mapped to a corresponding vector of real values), while the output of the last layer is combined using a linear layer and passed through a non-linearity to generate a session prediction.

6.3.1 Implementation

The implementation consisted of three main areas: data generation, model training and finally inference and measurement on the test set. We re-used the data generation and inference modules from previous chapters without modification, and we extended the training module to accept new runtime parameters to alternate between different samplers and hidden state management regimes.

Our chronological and random training regimes were implemented as pluggable PyTorch [94] samplers, selectable at runtime. Hidden state was initialised as a zero tensor in all experiments and then either re-zeroed between batches (the no-sharing regime), or re-used between batches in the chronological training case (the sharing regime). When training chronologically, the hidden state was detached from the computational graph after each batch to avoid the automatic differentiation from back-propagating all the way back to the beginning of each epoch.

6.4 Experiments

Our experiments focused on the two areas of RNN training where we observed significant differences in RNN performance depending on the choice taken:

- Whether or not hidden state was shared between batches.
- The sampling method used to construct batches as part of the training algorithm.

6.4.1 Desired Task

Predicting a users intent to purchase from their clickstream is a difficult task [23]. Clickers (users who only click and never purchase within a session) and buyers (users who click and also purchase at least one item within a single session) can appear to be very similar, right up until a purchase action occurs. Additionally, the ratio between clickers and buyers is always heavily imbalanced - and can be 20:1 in favour of clickers or higher. An uninterested user will often click on an item during browsing as there is no cost to doing so - an uninterested user will not *purchase* an item however. As noted in [60], shoppers behave differently when visiting online vs physical stores and online conversion rates are substantially lower, for a variety of reasons.

When a merchant has increased confidence that a subset of users are more likely to purchase, they can use this information in the form of preemptive actions to maximise conversion and yield. The merchant may offer a time-limited discount, spend more on targeted (and relevant) advertising to re-engage these users, create bundles of complementary products to push the user to complete their purchase, or even offer a lower-priced own-brand alternative if the product is deemed to be fungible.

6.4.2 Dataset Used

The RecSys 2015 Challenge [97] is a set of e-commerce clickstreams well suited to testing purchase prediction models. It is reasonable in size, consisting of 9.2 million user sessions. These sessions are anonymous and consist of a chronological sequence of time-stamped events describing user interactions (clicks) with content while browsing

and shopping online. The dataset also contains a very high proportion of short length sessions ($<= 3$ events), making this problem setting quite difficult for RNNs to solve.

No sessions were excluded - the dataset was used in its entirety. This means that for sequences with just one click, we require the trained embeddings to accurately describe the item, and time of viewing by the user to accurately classify the session, while for longer sessions, we can rely more on the RNN model to extract information from the sequence. This decision makes the training task harder for our RNN model, but is a fairer comparison to previous work using GBM where all session lengths were also included [23, 55, 56]. Lastly, the dataset is quite imbalanced - the class of interest (buyers) represents just 5% of the total number of samples.

6.4.3 RNNs vs GRU vs LSTM

Gated Recurrent Units, or GRU [115] are a simplification of LSTM, with one less gate and the hidden state and cell state vectors combined. These modifications mean that GRU is less computationally intensive to train versus LSTM. In practice, both LSTM and GRU are often used interchangeably and the performance difference between both cell types is often minimal and / or dataset-specific. RNN cells are the simplest of all recurrence cell types, with just a feedback loop from time $t-1$ to time t and possessing none of the gate structures / arrays used by GRU or LSTM.

Table 6.1 shows the impact of stateful vs stateless hidden state sharing on four widely used RNN architectures. In all cases the effect of not using non-local state is a noticeable

increase in AUC performance for a binary classification task. We use AUC to measure model performance since the classes are imbalanced.

Model	Stateful AUC	Stateless AUC
LSTM	0.75	0.841
GRU	0.756	0.839
RNN (TANH)	0.716	0.826
RNN (RELU)	0.789	0.834

TABLE 6.1: The effect of sharing hidden layer parameters on four widely used RNN model architectures - LSTM, GRU, and standard RNN units using TANH or RELU non-linear activation functions.

6.4.4 Chronological Training Regime

A common technique to prevent overfitting is to randomly sample from the training set during training and present disjoint samples to the model. However, if our goal is for LSTM to learn trends that develop over time then we must train chronologically and assume that there is a progression through time that our LSTM model can learn to discern between session outcomes. Therefore we re-order the training and testing set by time and perform both training and inference *chronologically*. As Figure 6.2 shows, the initial results are very positive with a higher training performance, but roughly 20% into the epoch, training degrades substantially and does not recover. Validation performance is also significantly reduced. Therefore we conclude, that with conventional LSTM at least, there is no clear evolution in patterns over linear time that can be used to improve performance. Even though clickstreams are at some level a time series, the best model performance is achieved when *not* treating them as time series and instead sampling randomly. This finding is closely related to the initial finding shown in Figure 6.1, where it is counter-productive to pass hidden state across batch boundaries.

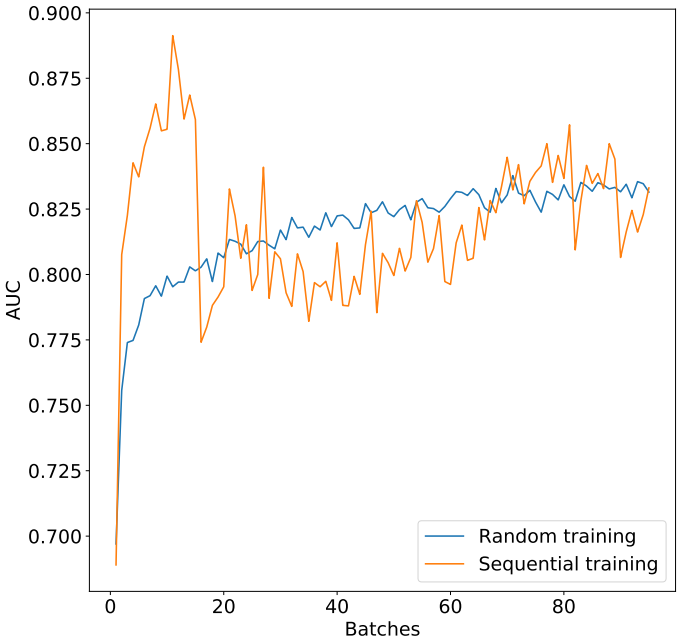

FIGURE 6.2: Training performance of LSTM under two sampling regimes: chronological (sequential) and random.

6.4.5 Parameter Reduction

Our second experiment tests the theory that by selective ablation of certain parts of the model, we can remove the model's ability to pay attention to global data features. We reduce model capacity by freezing the embeddings only and retaining all of the RNN capacity. Our motivation in doing this is that not all model parameters are created equal - the embedding for an infrequently-encountered item will have far less effect on model performance than the hidden state contained directly in the model itself. Therefore we:

- Froze the entire embeddings layer - model loss was not back-propagated to the input layer.
- Moved from LSTM to standard RNN with RELU activations - removing internal memory from the model itself.

Under normal training conditions, the model loss is back-propagated all the way into the aggregate embedding layer, in effect treating these embeddings as a trainable memory for concepts such as popular and unpopular items / categories / dates, similar versus dissimilar item pairs and so on. With the embedding layers frozen, model performance does indeed reduce, as Table 6.2 demonstrates. The restrictions imposed do reduce the predictive power of the model - for example both RNN models have 5% of the best model parameters yet are able to recover 86% and 90% of its predictive power.

These two changes resulted in a very large decrease in the number of the model parameters - from $6{,}945{,}871$ to $367{,}873$, or a reduction of 94%. The changes caused the model to under-perform our best model.

Type	Model parameters	Test AUC
LSTM: learnable embeddings	$6{,}945{,}871$	0.841
LSTM: frozen embeddings	$1{,}470{,}721$	0.74
RNN: TANH activation + frozen embeddings	$367{,}873$	0.722
RNN: RELU activation + frozen embeddings	$367{,}873$	0.762

TABLE 6.2: The effect of removing the ability of RNN and LSTM to store dataset statistics other than those directly obtainable from a session (i.e. local) by either freezing embedding weights during training or using simpler, non-gated recurrent units.

6.5 Summary and Conclusion

We presented a series of experiments using different Recurrent Neural Network types to investigate if both global and local (session-level) patterns in e-commerce datasets are used to infer user intent. The conclusion is yes, albeit with some caveats: although training results were substantially affected when the training set-up was configured to enable global training, results on the validation and test sets under-performed a training

configuration where no global patterns or context were assumed. Instead, RNNs can use trainable embeddings to learn global statistics over time to improve performance.

Our results demonstrate that careful selection and configuration of both model and training regime is necessary when applying RNNs / LSTM to the domain of e-commerce and clickstream analysis. Moreover, the current best practice in word language models of propagating hidden state between batches does not transfer to clickstream analysis and should be tested carefully on new datasets and domains.

In the next chapter, we provide details of our hyperparameter tuning approach.

The material in this chapter draws substantially from [25], published at the Deep Learning workshop day held during the 24th ACM SIGKDD Conference on Knowledge Discovery and Data Mining.

Chapter 7

Hyperparameter Tuning

In this chapter we detail our experiences with hyperparameter tuning, especially automatic tuning. Considerable time and effort was expended in this tuning effort, however we found that while hyperparameter tuning has its place, it is not a replacement for algorithmic insights and advances.

7.1 Hyperparameter tuning

The two model implementations used in this thesis - Gradient Boosted Machines (GBM) and Recurrent Neural Networks (RNN) - rely heavily on good hyperparameter selection in order to achieve good performance. A hyperparameter is any variable of the model or learning algorithm which must be selected external to the training process, i.e. it cannot be learned by the algorithm itself.

For example, GBM can train to very high levels of precision and recall if the tree depth is large and new decision and leaf nodes are created to reduce training error (max_depth, min_child_weight and gamma respectively), but then will perform badly on a validation or test set (displaying overfitting or high variance).

RNNs have equally important hyperparameters to prevent overfitting - training batch size and dropout to promote loss regularisation, architecture parameters such as the number of layers to improve performance and so on. In fact, hyperparameter tuning for neural networks is often likened to a dark or magic art, and depends on the domain, dataset and model architecture employed [131].

We invested considerable time and effort in automating the selection of optimal hyperparameters. Initially we used default hyperparameter values as per [111, 117] since our model is inspired by the word language model and we used Adam as our optimiser, and then expanded our search for the optimal values to use for clickstream analysis. Spearmint [92] was used to partially automate the discovery process of the best combination of hyperparameters to use. Spearmint is a Bayesian optimiser which can be used to find machine learning hyperparameters that minimise a loss function and are expressible as a float, integer, boolean or enumeration (a discrete range of values). A Bayesian prior is initialised and updated after every experiment to converge on the best combination of parameters. Algorithm 2 provides an overview of the algorithm used in Spearmint - the real-world code makes some simplifications and approximations that are not described in the Spearmint literature.

Our experience of Spearmint was mixed. We found that for large combinations of hyperparameters with wide ranges, Spearmint wasted time exploring hyperparameter

Algorithm 2 The Spearmint hyperparameter tuning algorithm.

1: $init\_experiments()$ *//Create 2 experiments. Size is configurable, purpose is to initialise the population to draw from.*
2: $g \leftarrow build\_grid(20000)$ *//Creates a Sobol grid with 20,000 cells. A Sobol sequence is a low discrepancy quasi-random sequence. This models the potential space of solutions (in theory unlimited, in practice set to 20,000).*
3: $g \leftarrow populate\_grid()$ *//Place valid values in each cell subject to user-provided constraints.*
4: $g \leftarrow overlay\_previous()$ *//Add previously visited points to the grid. Ensures Spearmint does not make the same recommendation twice.*
5: **while** true **do**
6: $fit()$ //Fit using known information to maximise Expected Improvement (EI).
7: $s \leftarrow suggestion()$ *// Retrieve the next suggestion to try*
8: $s_{cb} \leftarrow current\_best()$*//Compute the current best suggestion.*
9: $mean = find\_mean()$ *//Compute the Gaussian Process (GP) mean.*
10: $mean_{min}, mean_{argmin} \leftarrow optimise\_gp\_mean()$ *//Find the min and argmin of the GP mean*
11: $mean_{min} \leftarrow un\_normalise(mean_{min})$ *//Un-normalise the min of mean to original units*
12: $s_b \leftarrow best()$ *//Compute the best value seen so far*
13: $s_{bm} \leftarrow best\_model()$ *//Return best value according to model, not observed*
14: $c \leftarrow candidates(s_{bm})$ *//Add some extra candidates around the best so far*
15: $ei_g \leftarrow expected\_improvement(g)$ *//Compute Expected Improvement (EI) on the grid*
16: $ei_{highest} \leftarrow highest\_ei(g)$ *//Find the points on the grid with highest EI*
17: **end while**

combinations that we knew were sub-optimal. An exhaustive exploration of a large hyperparameter space would require very large amounts of compute capability as well as wall clock time. However Spearmint can run in reasonable time when the number and range of hyperparameters are reduced.

We illustrate our experience with Spearmint using a positive and negative example. Our positive example relates to the dropout rate hyperparameter - the rate we used in chapters 4 and 5 was discovered by Spearmint and is considerably lower than our original value (0.004873 vs 0.1 / 0.2). During our experiments we increased the training batch size from 32 to 256 and this increase acted as a regulariser on our model preventing overfitting, removing the need for dropout to perform the same function.

Our negative example is learning rate. Originally we used Spearmint to select a learning rate for the Adam optimiser of $1e^{-3}$. Using this learning rate value for the model grid search exercise reported in table 5.4 results in the test AUC values reported in 7.1. Clearly, the results in Table 5.4 from Chapter 5 which use a 1-cycle cyclical learning rate policy [4] are better than those in Table 7.1.

Like any automated hyperparameter tuning mechanism, Spearmint can only explore values in a specified range, using a specified step-size. It cannot discover a new learning rate scheduler.

Figure 7.2 shows the key idea behind the 1-cycle cyclical learning rate policy - although simple in conception and implementation, Spearmint cannot discover this when probing a linear range of candidate values. Using the technique detailed in [4] we selected the minimum and maximum values for the learning rate by gradually increasing the learning rate over a single epoch and plotted the average loss against the log learning rate. Our goal is to select a minimum learning rate close to where the loss begins to reduce considerably, and set the maximum learning rate before the loss increases dramatically, indicating that the model is diverging, and not converging. Figure 7.1 demonstrates the selection strategy for the RecSys dataset.

In [132], hyperparameter auto-tuning is expanded to include the entire model architecture itself. Good results are reported, however again we note the inordinate amount of compute resources required. In our opinion, algorithmic improvements and targeted heuristics such as cyclical learning rates [4] offer better prospects for model improvement, while spare compute capacity should be used in an automated continual refinement manner to fine-tune hyperparameters.

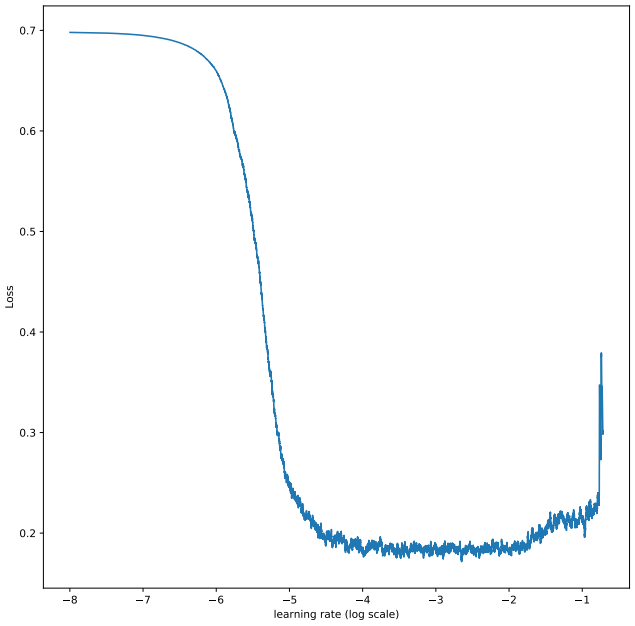

FIGURE 7.1: Following the doctrine in [4], we trained for 1 epoch with an initial rate of $1e^{-8}$ and gradually increased this to 0.1. The graph clearly intimates values for the minimum and maximum learning rates of $5e^{-5}$ and $5e^{-3}$.

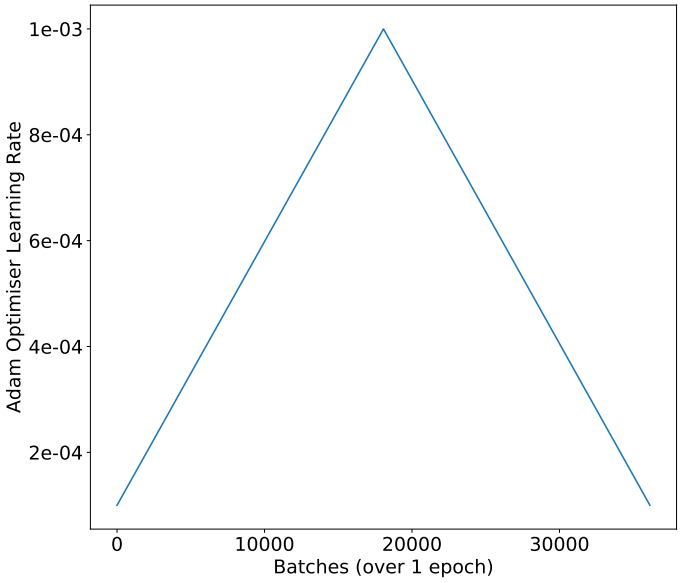

FIGURE 7.2: In the 1-cycle policy, the learning rate starts low ($1e^{-4}$) and increases gradually to $1e^{-3}$ at the mid-way point of the epoch. The rate then cools again, reaching the original starting point at the end of the epoch.

	RNN			GRU			LSTM		
Layers	1	2	3	1	2	3	1	2	3
Layer size									
64	0.72	0.81	0.81	0.741	0.832	0.833	0.735	0.832	0.831
128	0.72	0.80	0.80	0.755	0.833	0.833	0.729	0.834	0.834
256	0.71	0.80	0.80	0.732	0.834	0.834	0.724	0.834	**0.839**
512	0.69	0.80	0.77	0.746	0.832	0.833	0.759	0.835	**0.839**

TABLE 7.1: Model grid search results for number and size of RNN layers by RNN type on the RecSys 2015 dataset.

In order to make the hyperparameter search process more tractable in time and compute resources, we note that some hyperparameters are more important than others [133]. For GBM we chose to optimise tree depth and the node creation penalty (new nodes help reduce training error, but can adversely impact validation / test error).

For RNNs, we chose to optimise the number of layers in the model, batch size and dropout rate.

Normally, the learning rate used to update RNN model parameter weights would be an important parameter to include, however we chose to use the Adam [117] optimiser (the name is derived from adaptive moment estimation) which is more immune to the initial learning rate selected. Adam is different to classical Stochastic Gradient Descent (SGD), which uses a single learning rate α, for all weight updates and the learning rate does not change during training. Therefore selecting the correct / best α when using SGD is critically important to model performance.

With Adam, a learning rate is maintained for each network weight (parameter) and separately adapted as learning unfolds. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the

gradients. Kingma and Ba describe Adam as combining the advantages of two other extensions of stochastic gradient descent:

- Adaptive Gradient Algorithm (AdaGrad) that maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems).
- Root Mean Square Propagation (RMSProp) that also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). This means the algorithm does well on online and non-stationary problems (where the data itself is noisy).

In practice we found Adam to require little tuning of hyperparameters - in contrast to SGD which is very sensitive to the learning rate selected. For the datasets we used, Adam also performed comparably well to SGD, thus it became our default optimiser selection in all experiments.

Spearmint offers straightforward integration with Python code - a small glue script needs to be implemented which maps Spearmint suggestions for hyperparameters onto configurable values in the machine learning model and / or algorithm. A single experiment is run at a time, as the result is needed to update Spearmint priors / beliefs and then selecting the next experiment to run.

7.1.1 Practical Effectiveness of Automated Hyperparameter Tuning

In Chapter 5, we demonstrated that a model from the deep learning family (LSTM) combined with an appropriate input representation (word vectors), could recover over 98% of the state of the art performance, and with virtually no feature engineering used. The state of the art approach by contrast, required a very significant investment in feature engineering.

However, our deep learning approach is not without drawbacks. Although we were successful in removing almost all of the feature engineering effort, the number of model architecture parameters and training hyperparameters meant that we spent time in tuning the model architecture and associated hyperparameters. Part of this is simply because, as we showed in Chapter 6, current best practices and configuration for RNN models are very domain and dataset-specific. Moreover, there is no strong theoretical basis for choosing hyperparameters - the process is inherently empirical and time-consuming.

Therefore, the time, domain expertise and coding effort required to generate good features for GBM is partly offset by the time and computational resources needed to find the best combination of model architecture configuration and hyperparameters to use with LSTM. A brief example serves to illustrate the time required for even a cursory scan over a small number of hyperparameters:

- training batch size - one of $32, 64, 128, 256$.
- dropout rate - from $0.1 \ldots 0.5$ with 0.1 increments.

- learning rate - from $1e^{-3} \ldots 1e^{-4}$ with 0.0001 increments.

A single training epoch takes 1 hour on the RecSys dataset and at least 3 epochs are needed to evaluate the effect of hyperparameter selection. Therefore this example requires $4 \times 5 \times 10 \times 3 = 600$ hours of compute time. Moreover, the increments used are often more fine-grained ($1e^{-3}$ vs $1e^{-2}$ to tune dropout for example). Although this represents the worst case scenario of running hyperparameter tuning experiments to cover an entire grid, considerable compute time is also needed for the two major alternatives - random search and approaches using Bayesian optimisation.

In practice, we chose sensible defaults for hyperparameters for both GBM and LSTM as detailed in the relevant literature [28, 56, 111, 117] and then scheduled Spearmint sessions to fine-tune parameters within narrower ranges to reduce the potential search space.

7.1.2 The Advantage of Pluggable Implementations

While the GBM model and training algorithm is relatively monolithic, the RNN model is composed of interchangeable components, for example:

- The optimisation algorithm used to update weights after each batch - we used the Adam optimiser as already described but alternatives include SGD, Adagrad and LBFGS among others.

- The learning rate scheduler - we used a cyclic rate scheduler as detailed in [4] over more standard schedulers such as a step or plateau scheduler. We found that using

a 1-cycle cyclical learning rate policy which scaled from $1e^{-5}$ to $1e^{-3}$ improved our AUC from 0.839 to 0.841 on the RecSys challenge dataset, an improvement of 0.3%.

7.2 Experiment Setup

7.2.1 Model Evaluation Protocol

We wrote a number of scripts to evaluate model performance. Since the buyer and clicker classes are so imbalanced (20:1 in favour of clickers for the YOOCHOOSE RecSys 2015 dataset and 100:1 for the Retailrocket dataset), our primary evaluation metric is AUC-ROC, we used two independent implementations to check our calculations. Both are based on the composite trapezoidal method [134].

In statistics, a receiver operating characteristic curve, i.e. ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. Therefore, a good classifier as measured by AUC-ROC can still deliver poor performance if the threshold is poorly chosen, but the AUC-ROC curve illustrates the ability of the model to discriminate between two classes. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings and is considered to show the expectation that a uniformly drawn random positive is ranked before a uniformly drawn random negative[135].

We do not claim that the AUC-ROC curve is a perfect measure of model power, but for the binary classification problem focused on in this thesis where the dataset is imbalanced, we do believe it is the most suitable.

In general, we evaluated different model performance as follows:

- The models to compare were loaded into memory.
- Each model predicted a probability score for user intent and these outputs were then sorted by session ID.
- We calculated the overall model AUC-ROC twice using two independent approaches and compared them to ensure correctness.
- For deeper comparisons, we obtained subsets of the data (low price, high price, session length) and again compared AUC values for each model.
- The various results obtained were logged, graphed and then written to disk for further use.

7.2.2 Datasets

We used two e-commerce clickstream datasets in our experiments:

- The RecSys 2015 challenge dataset supplied by YOOCHOOSE.
- The Retailrocket dataset supplied by Retailrocket.

In Chapter 4 and Chapter 5 we provided a detailed comparison of both datasets - including size, complexity and class imbalance ratios. One important difference which affects experiment design is that the RecSys dataset contains a dedicated test set (20% of the entire dataset) while the Retailrocket dataset does not. For consistency, with each dataset we reserved 10% of the main dataset as a validation set, and we used the RecSys test set when comparing it to the original models from the competition.

7.3 Performance

The GBM training algorithm runs on a CPU, while our RNN models are trained on a GPU, therefore they are not directly comparable. However the wall clock time taken to train each model is relevant when we consider a production deployment. In generating the values for Table 5.7 in Chapter 5, the average training time for LSTM was 341.36 seconds with a standard deviation of 9.75 while the average for GBM was 752 seconds with a standard deviation of 10.21.

7.4 Conclusion

In this chapter, we described our overall approach to hyperparameter tuning and contrasted it with specific algorithmic improvements. We showed that hyperparameter tuning is most beneficial when used conservatively to explore well-defined ranges bounding specific variables. In the next chapter, we perform a critical assessment of our results so far.

Chapter 8

Critical Assessment of Results

So far our work has followed a methodical path. First we strove to understand which features in e-commerce clickstreams work best for the user intent prediction task and why Gradient Boosted Machines (GBM) is a suitable model implementation to use.

In Chapter 4 and Chapter 5, we noted that model choices such as GBM yield good results only if a significant investment in feature engineering is also undertaken. This investment is problematic for a number of reasons:

- Features are time-consuming to generate - there is no automatic way to build features.
- Features can be reliant on a single dataset.
- Features can be reliant on a single domain.

In Chapter 5 and Chapter 6 we moved on to investigate deep learning as a model substitute for GBM in the user intent prediction problem - yielding promising results when

compared to state of the art and practical contributions on how to apply deep learning to e-commerce clickstreams.

In the following sections we review the precision, thoroughness and contribution of our work with the current state of the art approach. We also illustrate the limitations of our current method when compared to potential future work.

8.1 Data-driven analysis of GBM vs LSTM

In Chapter 5, we segmented the RecSys dataset in a number of ways in an attempt to find either a segment where RNN / LSTM outperformed GBM, or the opposite - a segment where GBM significantly out-performed RNN / LSTM. However in all of the segmentations we conducted, RNN / LSTM underperforms GBM in a consistent way, demonstrating the power of designed features that exploit the characteristics of a specific dataset. Our segmentations were designed based on domain expertise as follows:

- High value items - common intuition suggests that users will browse carefully and compare often, generating more clicks and dwelltime pauses for LSTM to analyse.
- Low value items - the opposite logic to high value items suggests that users will purchase quickly, favouring GBM.
- Long dwelltime - related to high price, but a user may spend a longer time analysing an item for reasons other than price - reading terms and conditions for example. We posited that LSTM should do well on these user sessions.

As the figures in Chapter 5 illustrate, relative model performance between GBM and LSTM was stable regardless of the data segmentation used.

8.2 Interpretability

Interpretability is a key concern in the deployment and use of machine learning models in real world production settings. Although the domain of e-commerce does not require models to support (or make) potentially life and death decisions as opposed to medical care or policing, significant profits and losses can be earned.

Understanding what a model is doing and why it returns the output $\hat{y}$ for a given input x is challenging, perhaps for any model more complex than linear or logistic regression. As an example, although Gradient Boosted Machines (GBM) claim to admit analysis (the tree structure and rules can be viewed by a human assessor) for any complex problem the tree depth, width and recurrence of the same variable in the tree quickly renders the model unsuitable for human interpretation.

Deep learning models are less interpretable than Gradient Boosted Machines (GBM). Neural networks are complex dynamic systems with multiple non-linearities. Increasingly, input data x_i is translated into embeddings as part of the inference process to extract the best model performance. These embeddings muddy the waters even further, especially when we move away from sequence to sequence translation and towards classification tasks. Figure 8.1 from [5] attempts to show how some cells activate strongly for some corpus features while other features are much harder to interpret, but overall any end-user will struggle to use these visualisations in a useful manner. Text colour

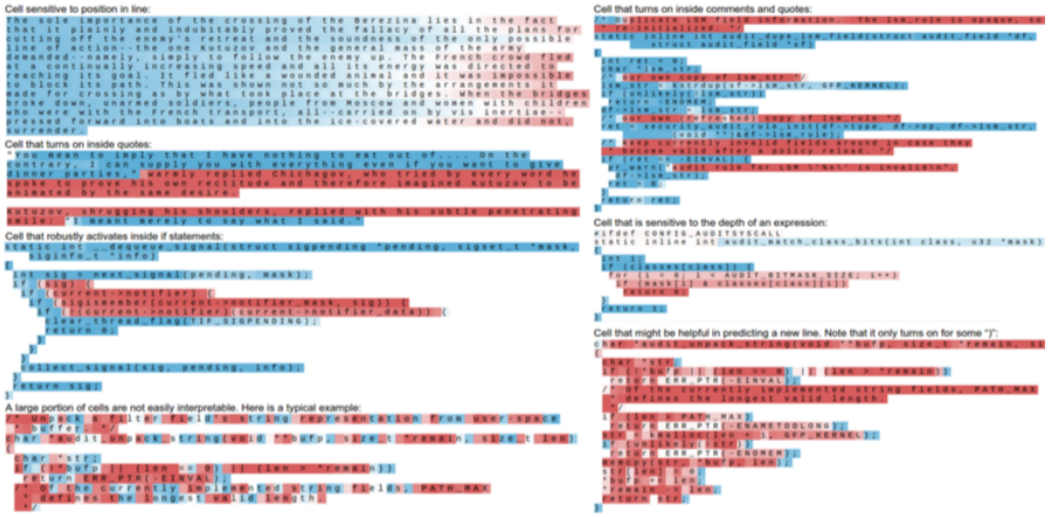

FIGURE 8.1: From [5]: Several examples of LSTM cells with interpretable activations discovered when trained on two text datasets - Linux kernel source code and War and Peace.

corresponds to $tanh(c)$, where -1 is red and $+1$ is blue. According to [5], these are interpretable, saturated gate activations with values < 0.1 named left-saturated, and values > 0.9 right-saturated. Right-saturated values are said to correspond to cells that remember values for very long time periods but this is not quantified or explained. In general, this regime can only be considered as providing partial or incomplete interpretability of RNNs. The author realises that this approach is not optimal, and in [136], provides multiple examples where this approach to interpretability yields both insightful, confusing and contradictory results.

Hinton diagrams are also used to visualise the values of a 2D array (e.g. a weight matrix or tensor) and thus the connection strength between cells or units. Positive and negative values are represented by white and black squares respectively, and the size of each square represents the magnitude of each value. Again however, we argue that these weight plots do not confer interpretability.

An alternative approach is to substitute (only approximately) the model in question

with a simpler one (e.g. a linear classifier) which is easier to understand. Then the proxy model can be used to help understand the more complex and impenetrable model, in part at least. This is the approach offered in [119], which proposes LIME (Local Interpretable Model-Agnostic Explanations). In LIME, an explanation is a *local linear approximation* of the model's behaviour. We provided a detailed overview of LIME in Chapter 5 and applied it to specific user sessions of interest to gain a better insight into the operation of our RNN model.

Solving interpretability is a key milestone in the widespread adoption of complex machine learning models in e-commerce. If users do not understand a model and its predictions, they will not trust it. Given that our thesis focuses on how merchants can improve their efficiency using machine learning, a lack of trust will remove opportunities to make savings, reduce waste and increase profits.

8.3 Causal Inference

The machine learning field has always been aware of its own limitations. Some problems are endemic - overfitting, underfitting, unnecessary complexity, lack of interpretability, opaque results achieved by hyperparameter tuning, and especially for deep learning, a sense that model performance cannot be explained theoretically. Of all the machine learning approaches, deep learning is the most empirical. Our deep learning model suffers equally from these drawbacks and will only improve as the field in general improves.

Within e-commerce, merchants can further improve their profitability if they can plausibly answer some central questions relating to volume, price and factors affecting conversion. But as we see below, questions of this ilk require advances over current machine learning approaches.

Most recently [137] (although the author has been working on it for decades [138]), the lack of *causal inference* in current machine learning approaches has been decried. Proponents of causal inference claim that without it, machine learning models are merely curve fitting. Adding the ability to understand and reason with counterfactuals for example, would improve machine learning model performance considerably. Our models can also be critiqued in this way, however we appeal to the reader that curve fitting used correctly provides real value.

To demonstrate the reasoning capability that Pearl believes is missing from current machine learning models, we note here the key ideas from [137]. Firstly, a 3-layer hierarchy where each successive layer relies on the layers beneath it being in place and solved:

- Layer 1 - association, or $P(y|x)$. e-commerce example: what is the conversion rate of item i_k.
- Layer 2 - intervention, or $P(y|do(x), z)$. e-commerce example: what is the projected conversion rate of item i_k if we double the price.
- Layer 3 - counterfactuals, or $P(y_x|x', y')$. e-commerce example: what would the conversion rate of item i_k be if we doubled its price a year ago.

Being able to answer these types of question would add significant value to e-commerce merchants when used in conjunction with our user intent discovery models - for example enabling merchants to implement optimal pricing and stocking strategies. Pearl further proposes how such an analytic capacity would be built, namely:

- Encoding causal assumptions – transparency and testability. The author prefers graphical models to achieve these two goals using the structure of the graph for transparency and d-separation (dependence separation) to test statistical predictions.

- Do-calculus and the control of confounding. Confounding (the presence of unobserved causes of two or more variables) is a major obstacle to drawing causal inference from data, however progress in the form of back-door criteria and the do-calculus has been made.

- Algorithmisation of counterfactuals, or the formalisation of counterfactual reasoning within a graphical model representation.

- Mediation analysis and the assessment of direct and indirect effects. Again the formalisation of counterfactual reasoning within a graphical representation admits this analysis and assessment.

- Adaptability, external validity and sample selection bias. These are common concerns in all machine learning settings, and the author proposes do-calculus to overcome bias due to environmental changes.

- Recovering from missing data. This is another common issue, however the author proposes a causal model of missing data, to inform conditions under which incomplete data can be used robustly.

- Causal discovery. Following on from the d-separation criteria above, a compact set of models can be inferred from the data and used to answer causal queries.

We note that at least 3 of these "seven pillars" are not unique to causal inference and are common to all branches of machine and statistical learning.

Building a system to encode, organise and reason about facts will require probabilistic graphical models (PGM) [30], structural equations and counterfactual and interventional logic.

Causal inference is not without its own challenges - for Structured Causal Models (SCM) the graph structure must be validated against domain expertise and *assumptions* (i.e. edges which are not present) are as important as facts / influences (defined edges). Additionally, confounding variables can affect the logic of the graph.

In practical terms, the performance and scalability of PGM is problematic for large feature sets. But overall, the potential benefits are significant and causal inference promises to augment interpretability and understanding in the field of machine learning.

8.4 Online Prediction

Knowing the intent of a user is only useful if that knowledge is acted upon - and promptly. Both GBM and LSTM models can be deployed to a production setting in

a straightforward manner and used in near real time to inform system actions.

"Near real time" however is a fuzzy requirement. Speed is essential in the online realm. Particularly when searching or browsing, users have been shown to be remarkably intolerant of slow systems [139], with 53% of mobile site visits abandoned if pages take longer than 3 seconds to load. Incorporating the prediction of a model into an existing recommender or search engine, and returning the rendered page to the user within 3 seconds represents a significant challenge. Further work is needed in the efficient deployment of trained machine learning models to production in a low-latency environment.

8.5 Dataset Preprocessing Using Coresets

In Chapter 6 we showed that a random sampling strategy was necessary during training to achieve the best RNN model performance. However more work in filtering and refining the training set is certainly possible. We would like to explore the idea of constructing a *coreset* [140] - a small, weighted subset of the data that approximates the full dataset to speed up model training time.

In the current work, we employed two strategies to address the class imbalance problem:

- Weighting positive labels to force the network to pay more attention to them during training.
- Removing clicker sessions to partially address the imbalance issue.

We found that positive weighting had no impact on model performance, and that it was possible to remove 50% of the clicker sessions before training without impacting test performance. However it is likely that there is even more redundancy to exploit in e-commerce datasets, enabling faster, more accurate training on a smaller dataset.

8.6 Contribution

We used our literature review to paint a comprehensive picture of the current e-commerce domain, and particularly the usage of machine learning within that domain. After focusing on the task of predicting user intent in order to provide better insight to merchants, we constructed classical and deep learning models, compared them to a strong state-of-the-art baseline and to each other.

8.6.1 Machine Learning and E-commerce

Our literature review and survey demonstrated that machine learning has much to offer the field of e-commerce. Until recently, e-commerce has relied on information retrieval techniques based primarily on statistics, such as tf-idf ranking of documents in search. Recommender systems move more into machine learning, but do not explicitly model user intent. However almost every aspect of e-commerce - user interface, results ranking, question-answering and recommendations is moving to use machine learning algorithms. Behind the scenes, machine learning is used to control stock levels and predict refill orders, as well as set optimal pricing strategies

We also noted that of all the stakeholders in e-commerce, merchants are often excluded from explicit measurements of satisfaction - it is often deemed enough to have satisfied users. But the evolution of e-commerce and coalescence into extremely large portals such as Amazon, eBay and Google undermines this theory. We argue that merchant satisfaction must be measured more directly and that an important tool to give merchants is understanding user intent.

8.6.2 Inferring User Intent Using Classical Models

In order to understand what elements of e-commerce data are important to user intent classification, in Chapter 4 we used a superset of features and the state-of-the-art models from the top scoring submissions on a well-studied and open dataset, before adding our own features and then analysing overall contribution at the individual feature level. In this work, our best-performing item similarity feature was based on word vectors / embeddings and we grouped the feature set into three categories: popularity / frequency-based, time-based and item similarity-based. The success of our embedding approach, combined with the significant effort we invested into feature engineering, led us to explore feature-free alternatives - the class of model and learning algorithm that can learn their own internal representation during training without explicit feature engineering.

8.6.3 Deep Learning Application to User Intent Prediction

In Chapter 5 we presented a Recurrent Neural Network (RNN) model which recovers 98.4% of current state-of-the-art performance on the user purchase prediction problem in e-commerce using item and session representations learned during training. In contrast, state-of-the-art approaches use machine learning techniques such as Gradient Boosted Machines coupled with comprehensive feature engineering.

On a second clickstream dataset, our RNN model exceeds GBM performance, demonstrating that explicit features designed for one dataset do not automatically translate to other datasets, even when closely related and learned features can outperform them. Our model is straightforward to implement, generalises to different datasets in the same domain with comparable performance and can be trained using modest hardware.

In Chapter 6 we demonstrated that a canonical word-based language model is not suited for e-commerce session / clickstream analysis. Specifically we show that a random sampling training regime must be used in conjunction with no hidden state sharing across training batch boundaries.

8.7 Summary

In this chapter, we placed our research to date in a wider context. In Chapter 5 we showed that although our best RNN model requires no feature engineering at all, its performance when compared to the GBM state-of-the-art model is always slightly less. In Chapter 9 we propose additional research avenues to address this.

Although the problem of user intent prediction is important and valuable, we can see that a more general and interpretable model for e-commerce such as causal inference would represent a major, positive addition to the domain.

In the next and final chapter, we outline multiple directions for future work building on the work in this thesis and summarise our main conclusions and contributions.

Chapter 9

Further Work and Conclusion

In this chapter we outline the main future avenues of research that are indicated by the research conducted to date, before summarising our main contributions and concluding the thesis.

Our work to date presages a number of ways in which our research can be extended and refined. In summary, these are:

- Incremental improvements to the existing RNN model.
- Augmenting the item representation method used to meaningfully encode previously unseen data.
- Improving the predictive performance of the deep learning model (without onerous feature engineering).

9.1 RNN Model - Incremental Improvements

The training process achieved remarkably similar results across a wide variety of mutations to the model architecture - recurrence type, number of layers. We posit that this means that the current model architecture and training regime are optimal for the classification task.

Therefore, general improvements to the model are more likely to come from improving the input representation to the model, and further refinements in converting the model output to a probability score.

The input representation at this point has not yet been exhaustively explored. Although we conducted a hyperparameter search for the optimal embedding widths to use, we believe further work is necessary in this area. Hierarchical embeddings have been explored in [141] and may add value for the item embedding, since e-commerce product catalogues have an inherently hierarchical structure.

We also note that the general set of best practices used by the deep learning community evolves continuously. As already noted in Chapter 7, we improved model performance by a further 0.3% by applying cyclical learning rates [4] during training and it is likely that further incremental improvements will be found.

It is also possible that there are additional pooling or aggregation techniques which could bolster the connection between recurrent and output layers, although we have already applied the balanced pooling technique outlined in [142] without observing any increase in model power.

In the same vein, we trained a model using a bidirectional LSTM in order to use the reverse pass as a simple self-attention mechanism, again without observing any increase in the predictive power of the model. We believe that our attempts to apply attention to our model had little effect since the sequences are much shorter than typical sentences, however more work is needed here.

9.2 Enabling Inductive Learning

Using the same terminology as [143], our current embedding approach is shallow. From [144], shallow embedding representations have the following drawbacks:

- $O(|V|)$ parameters are needed: there is no parameter sharing and every node has its own unique embedding vector.
- Inherently transductive: It is not possible to generate meaningful embeddings for nodes that were not seen during training.
- Do not incorporate node features: Many graphs have features that we can and should leverage.

Using Graph Convolutional Networks (GCN) [145], a (domain-specific) encoder is used to generate embeddings for new, unseen items based on initial similarity to existing members of an embedding set. The accuracy of this initial similarity is important to effectively including new items but offers significant advantages as datasets expand. In other words, we delegate more computational complexity to the embedding input layer,

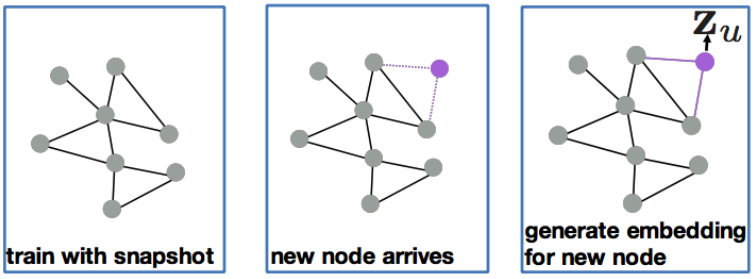

FIGURE 9.1: How Graph Convolutional Networks lend inductive capacity to a transductive model.

and allow the RNN layer to take advantage of the new, oriented embeddings at inference time.

One interesting side effect (not necessarily a drawback) of this approach is that under this regime, items will continue to affect the graph embeddings even after deletion. This is not a problem for the e-commerce domain, but may be for other domains.

Figure 9.1 demonstrates the key idea - existing nodes in the embedding graph are used to place, orient and connect new nodes so that the model is instantly able to compute over the new nodes without any training. This new data is placed using a similarity measure and then a new embedding is generated for the new item. Figure taken from [144].

9.3 Avoiding Overfitting

Overfitting is a problem that affects all supervised machine learning approaches [146], but because of the number of parameters in a neural network, they are especially prone to this problem. In the e-commerce domain, overlapping classes (also known as confusing examples) is a particular problem (buyers can be mislabelled as clickers if they

purchase elsewhere) and we plan to investigate an approach to remove these erroneous examples from e-commerce datasets using [147]. Erroneous samples are removed iteratively using a simple approach and then new models are trained on the newly pruned dataset. The goal is to roughly estimate the conditional probabilities of training set samples labels and to exclude those samples, for which $P(-y_i|x_i) > 0.5 > P(y_i|x_i)$.

9.4 Augmenting Local With Global Interpretability

In Chapter 5, we used LIME - a local interpretability approach to construct simple proxy models which explained our deep model behaviour for specific session samples of interest. This local technique provided some interpretability to our RNN / LSTM model. However, LIME is data-point specific and also generates a proxy model for our deep model - we are not analysing the deep model directly.

SVCCA (Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability) [148] is a complementary approach which analyses deep models directly through comparison of internal representations. We would like to compare our model using SVCCA as well as LIME so that we have both global and local interpretability.

9.5 Substituting RNNs With CNNs

A consistent theme in this thesis has been the natural suitability of RNNs to process sequential data. The ability of RNNs to unroll themselves in time makes for a seamless

application to datasets that have a definite temporal ordering - like e-commerce click-streams. In contrast, Convolutional Neural Networks (CNNs) [149] are traditionally employed in the field of computer vision and not for sequence processing. Traditionally, CNNs are held to outperform RNNs in extracting position-invariant features while RNNs excel at modelling sequential data.

However, recently [150], a CNN variant known as Temporal Convolutional Networks (TCNs) [150] have been used to process sequential data and the initial results are positive - comparable accuracy coupled with a lower computational complexity [151]. We note that the experiment tasks used in [150] do not bear a close resemblance to the problems set out in this thesis and given the counter-intuitive findings we note in the best training algorithm and sampling strategy to use, we would like to compare TCN with RNN in the e-commerce domain.

9.6 Data Augmentation Using Adversarial Training

Generative Adversarial Networks (GANs) [152] have recently attracted significant research interest. GANs work by using two neural network components in an antagonistic setting to improve their mutual performance. One *generator* network creates forgeries designed to trick the other network, which functions as a forgery detector or *discriminator*. In this adversarial setting, the forgery detector network becomes better at recognising valid or true samples while the generator can simulate samples with increasing fidelity.

There are multiple uses of GANs in an e-commerce setting. For example, one network would create synthetic buyer sessions in an attempt to trick the detector network, while we would expect the detector network to improve its recognition of true buyer sessions over time. In Chapter 5 we saw that our deep model is less robust when presented with rarely purchased or long tail items. In our experiments we also noted that traditional mechanisms used to address imbalanced datasets such as oversampling the smaller positive class and weighting positive class samples did not improve performance. Therefore good data augmentation techniques could help to improve model performance without artificially affecting the dataset distribution or core characteristics.

We could use GANs in an offline data augmentation role as a precursor to normal training, creating additional plausible buyer sessions which also contain these long tail items to reduce the class imbalance problem (buyers are far less represented than clickers in e-commerce clickstreams) when training.

Figure 9.2 below illustrates how the discriminator and generator components are trained in a GAN network. Our intention is to use a GAN to generate additional plausible buyer sessions and to use these when training our existing RNN model, thus addressing the class imbalance problem.

However, GANs possess their own problems which are not associated with the RNN models used in this thesis:

- Non-convergence: the model parameters oscillate, destabilise and never converge to a usable phase or attractor point.

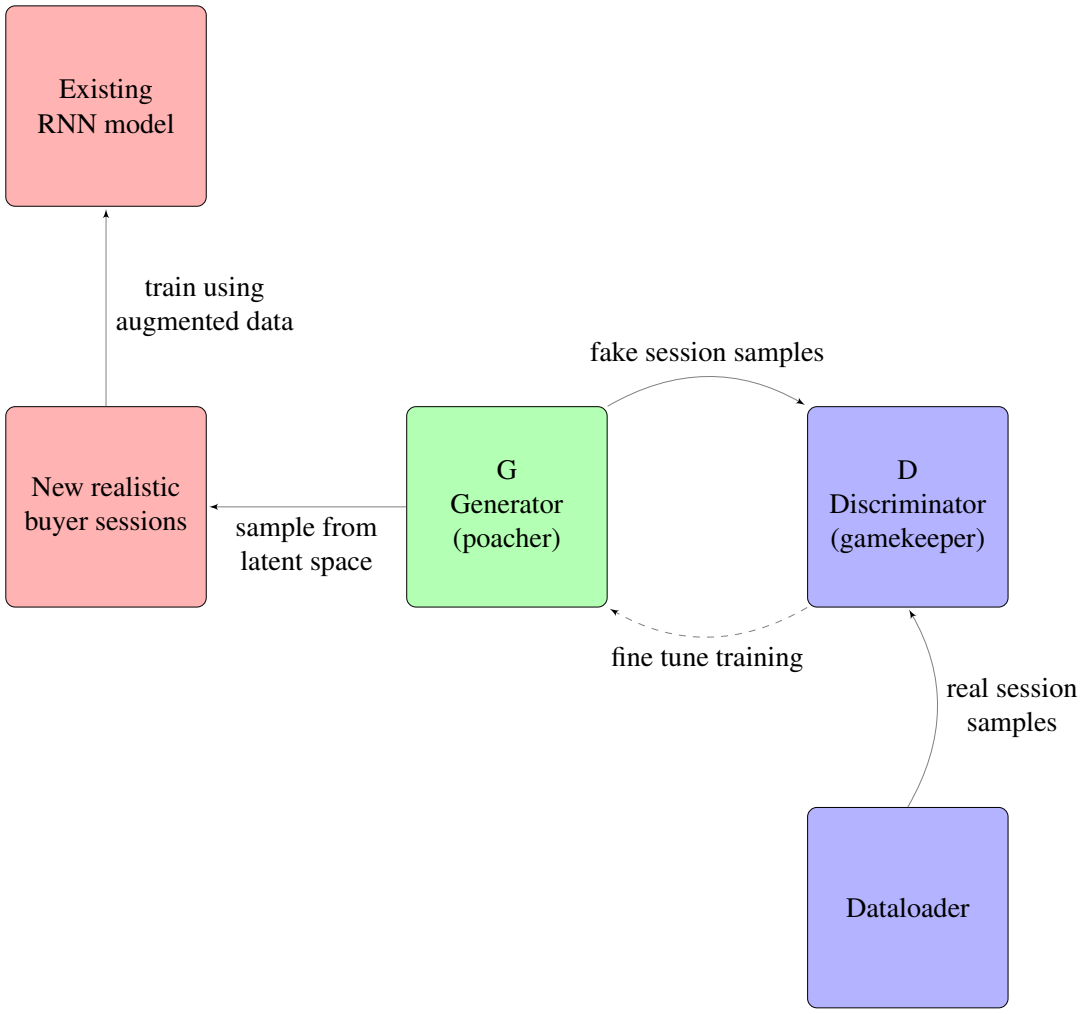

FIGURE 9.2: In generative adversarial networks, two neural networks (the generator / poacher and the discriminator / gamekeeper), duel with each other and in doing so mutually improve performance.

- Mode collapse: the generator collapses and is only able to synthesise limited varieties of samples - more clickers in our case.

- Diminished gradient: the discriminator gets too successful that the generator gradient vanishes and so no learning can occur.

9.7 Further Hyperparameter Tuning

Although we have invested considerable time and effort into hyperparameter selection for both our GBM and RNN models, it is not proven that the values we have selected are optimal. Not only must we consider very fine-grained float values for parameters such as dropout and learning rate, but also combinations of parameters. Although [133] claims that only a few hyperparameters ultimately matter, even this subset of possible combinations requires copious amounts of time and compute power. Lastly, the structure of our codebase allowed us to tune not just obvious parameters but also to modify the model architecture and training regime during experiments, for example the number of layers, sampler used, gradient descent method employed. Selecting the best combination of all of these options remains an open question, not just in our research, but for machine learning as a field.

9.8 Conclusion

In conclusion, our work shows that deep learning is competitive with classical machine learning for specific tasks in the e-commerce domain - specifically the task of inferring user purchase propensity which is an important problem in e-commerce. In addition, deep learning has the added benefit of representation learning, so in settings where new features may be required quickly, or assumptions underpinning features become invalid as the data set changes, may in fact out-perform classical techniques as we showed in Chapter 5.

Nevertheless, deep learning cannot be simply applied wholesale and unchanged to the e-commerce domain. As we showed in Chapter 6, significant modifications are required to the default training regime and hidden state management strategy in order to extract the best model performance.

Finally, further work is suggested on multiple avenues of research, ranging from better interpretability to causal inference and inductive learning.

Bibliography

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

[2] Atilim Günes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: A survey. *J. Mach. Learn. Res.*, 18(1):5595–5637, January 2017. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=3122009.3242010.

[3] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 3rd edition, 2018. ISBN 0130950696.

[4] Leslie N. Smith. No more pesky learning rate guessing games. *CoRR*, abs/1506.01186, 2015. URL http://arxiv.org/abs/1506.01186.

[5] Andrej Karpathy, Justin Johnson, and Fei-Fei Li. Visualizing and understanding recurrent networks. *CoRR*, abs/1506.02078, 2015. URL http://arxiv.org/abs/1506.02078.

[6] Mengting Wan and Julian McAuley. Modeling ambiguity, subjectivity, and diverging viewpoints in opinion question answering systems. In *ICDM*, pages 489–498. IEEE, 2016.

[7] R.L. Phillips. *Pricing and Revenue Optimization*. Stanford Business Books. Stanford University Press, 2005. ISBN 9780804746984. URL https://books.google.co.uk/books?id=bXsyO06qikEC.

[8] Forrester. US eCommerce Grows, Reaching $414B By 2018, But Physical Stores Will Live On. https://www.forbes.com/sites/forrester/2014/05/12/us-ecommerce-grows-reaching-414b-by-2018-but-physical-stores-will-live-on/#37b6c95b24e5, 2014. [Online; accessed 10-Jul-2018].

[9] Statista. Global retail e-commerce sales worldwide from 2014 to 2021. https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/, 2018. [Online; accessed 15-09-2018].

[10] Stripe. Stripe: About us page. https://stripe.com/about, 2018. [Online; accessed 27-Sep-2018].

[11] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, January 2003. ISSN 1089-7801. doi: 10.1109/MIC.2003.1167344. URL http://dx.doi.org/10.1109/MIC.2003.1167344.

[12] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. *Recommender Systems Handbook*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010. ISBN 0387858199, 9780387858197.

[13] Statista. Consumption expenditure on newspapers in the united kingdom from 2005 to 2017. https://www.statista.com/statistics/476016/expenditure-on-newspapers-in-the-united-kingdom-uk/, 2019. [Online; accessed 15-Feb-2019].

[14] Amazon to take almost 50 percent of the u.s. e-commerce market by year's end. https://www.cnbc.com/2018/07/12/amazon-to-take-almost-50-percent-of-us-e-commerce-market-by-years-end.html, 2018. [Online; accessed 15-02-2019].

[15] Norton Rose Fulbright. ECJ finds for Google on AdWords but advertisers beware. http://www.nortonrosefulbright.com/knowledge/publications/28253/ecj-finds-for-google-on-adwords-but-advertisers-beware, 2010. [Online; accessed 10-Jul-2018].

[16] Hal R. Varian. Position auctions. *International Journal of Industrial Organization*, 25(6):1163–1178, December 2007. URL https://ideas.repec.org/a/eee/indorg/v25y2007i6p1163-1178.html.

[17] Benjamin Edelman, Michael Ostrovsky, and Michael Schwarz. Internet Advertising and the Generalized Second-Price Auction: Selling Billions of Dollars Worth of Keywords. *American Economic Review*, 97(1):242–259, March 2007. URL https://ideas.repec.org/a/aea/aecrev/v97y2007i1p242-259.html.

[18] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008. ISBN 0521865719, 9780521865715.

[19] Financial Times. Brussels bids to get ahead of the curve with Amazon investigation. https://www.ft.com/content/bfac46c0-bc63-11e8-94b2-17176fbf93f5, 2018. [Online; accessed 20-Sep-2018].

[20] Statista. Advertising revenue of google from 2001 to 2018. https://www.statista.com/statistics/266249/advertising-revenue-of-google/, 2018. [Online; accessed 13-Feb-2019].

[21] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, and Joaquin Quiñonero Candela. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, ADKDD'14, pages 5:1–5:9, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2999-6. doi: 10.1145/2648584.2648589. URL http://doi.acm.org/10.1145/2648584.2648589.

[22] Tami Kim, Kate Barasz, and Leslie K John. Why am I seeing this ad? the effect of ad transparency on ad effectiveness. *Journal of Consumer Research*, page ucy039, 2018. doi: 10.1093/jcr/ucy039. URL http://dx.doi.org/10.1093/jcr/ucy039.

[23] Humphrey Sheil and Omer Rana. Classifying and recommending using gradient boosted machines and vector space models. In Zhang Q Chao F., Schockaert S., editor, *Advances in Computational Intelligence Systems. UKCI 2017.*, volume 650, pages 214–221, Cham, 2017. Springer. doi: 10.1007/978-3-319-66939-7_18. URL https://doi.org/10.1007/978-3-319-66939-7_18.

[24] Humphrey Sheil, Omer Rana, and Ronan Reilly. Predicting purchasing intent: Automatic feature learning using recurrent neural networks. In *ACM SIGIR Forum*. ACM, 2018.

[25] Humphrey Sheil, Omer Rana, and Ronan Reilly. Understanding ecommerce clickstreams: a tale of two states. In *KDD Deep Learning Day*. ACM, 2018.

[26] Peter Flach. *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. Cambridge University Press, 2012. doi: 10.1017/CBO9780511973000.

[27] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3146–3154. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf.

[28] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939785. URL http://doi.acm.org/10.1145/2939672.2939785.

[29] Anna Veronika Dorogush, Andrey Gulin, Gleb Gusev, Nikita Kazeev, Liudmila Ostroumova Prokhorenkova, and Aleksandr Vorobev. Fighting biases with

dynamic boosting. *CoRR*, abs/1706.09516, 2017. URL http://arxiv.org/abs/1706.09516.

[30] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009. ISBN 0262013193.

[31] Pierre Baldi and Søren Brunak. *Bioinformatics: The Machine Learning Approach*. MIT Press, Cambridge, MA, USA, 2nd edition, 2001. ISBN 0-262-02506-X.

[32] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, May 2013. doi: 10.1109/ICASSP.2013.6638947.

[33] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521 (7553):436–444, 5 2015. ISSN 0028-0836. doi: 10.1038/nature14539.

[34] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. doi: 10.1016/j.neunet.2014.09.003. Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

[35] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 4, 09 2014.

[36] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015. URL http://arxiv.org/abs/1505.00387.

[37] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013. URL http://arxiv.org/abs/1312.6199.

[38] Pierre Baldi and Peter Sadowski. Understanding dropout. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 2814–2822, USA, 2013. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999792.2999926.

[39] Jan Kukacka, Vladimir Golkov, and Daniel Cremers. Regularization for deep learning: A taxonomy. *CoRR*, abs/1710.10686, 2017. URL http://arxiv.org/abs/1710.10686.

[40] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*, pages 318–362. MIT Press, Cambridge, MA, USA, 1986. ISBN 0-262-68053-X. URL http://dl.acm.org/citation.cfm?id=104279.104293.

[41] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013. URL http://arxiv.org/abs/1301.3781.

[42] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The elements of statistical learning: data mining, inference, and prediction, 2nd Edition*.

Springer series in statistics. Springer, 2009. ISBN 9780387848570. URL http://www.worldcat.org/oclc/300478243.

[43] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL http://arxiv.org/abs/1207.0580.

[44] M.L. de Prado. *Advances in Financial Machine Learning*. Wiley, 2018. ISBN 9781119482086. URL https://books.google.co.uk/books?id=oU9KDwAAQBAJ.

[45] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014. ISBN 1461471370, 9781461471370.

[46] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Commun. ACM*, 56(2): 74–80, February 2013. ISSN 0001-0782. doi: 10.1145/2408776.2408794. URL http://doi.acm.org/10.1145/2408776.2408794.

[47] H. Brendan McMahan, Gary Holt, D. Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. Ad click prediction: A view from the trenches. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 1222–1230, New York, NY, USA,

2013. ACM. ISBN 978-1-4503-2174-7. doi: 10.1145/2487575.2488200. URL http://doi.acm.org/10.1145/2487575.2488200.

[48] Léon Bottou, Jonas Peters, Joaquin Quiñonero Candela, Denis X. Charles, D. Max Chickering, Elon Portugaly, Dipankar Ray, Patrice Simard, and Ed Snelson. Counterfactual reasoning and learning systems: The example of computational advertising. *J. Mach. Learn. Res.*, 14(1):3207–3260, January 2013. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2567709.2567766.

[49] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, August 2009. ISSN 0018-9162. doi: 10.1109/MC.2009.263. URL http://dx.doi.org/10.1109/MC.2009.263.

[50] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J. Smola, and How Jing. Recurrent recommender networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, pages 495–503, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4675-7. doi: 10.1145/3018661.3018689. URL http://doi.acm.org/10.1145/3018661.3018689.

[51] Allison J. B. Chaney, Brandon M. Stewart, and Barbara E. Engelhardt. How algorithmic confounding in recommendation systems increases homogeneity and decreases utility. *CoRR*, abs/1710.11214, 2017. URL http://arxiv.org/abs/1710.11214.

[52] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 133–142, New York, NY, USA, 2002. ACM. ISBN 1-58113-567-X. doi: 10.1145/775047.775067. URL http://doi.acm.org/10.1145/775047.775067.

[53] Jaime Teevan, Susan T. Dumais, and Daniel J. Liebling. To personalize or not to personalize: Modeling queries with variation in user intent. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 163–170, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-164-4. doi: 10.1145/1390334.1390364. URL http://doi.acm.org/10.1145/1390334.1390364.

[54] Farzad Eskandanian and Bamshad Mobasher. Detecting changes in user preferences using hidden markov models for sequential recommendation tasks. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, RecSys '18, New York, NY, USA, 2018. ACM.

[55] Peter Romov and Evgeny Sokolov. Recsys challenge 2015: Ensemble learning with categorical features. In *Proceedings of the 2015 International ACM Recommender Systems Challenge*, RecSys '15 Challenge, pages 1:1–1:4, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3665-9. doi: 10.1145/2813448.2813510. URL http://doi.acm.org/10.1145/2813448.2813510.

[56] Maksims Volkovs. Two-stage approach to item recommendation from user sessions. In *Proceedings of the 2015 International ACM Recommender Systems Challenge*, RecSys '15 Challenge, pages 3:1–3:4, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3665-9. doi: 10.1145/2813448.2813512. URL http://doi.acm.org/10.1145/2813448.2813512.

[57] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *CoRR*, abs/1511.06939, 2015. URL http://arxiv.org/abs/1511.06939.

[58] Malte Ludewig and Dietmar Jannach. Evaluation of session-based recommendation algorithms. *CoRR*, abs/1803.09587, 2018. URL http://arxiv.org/abs/1803.09587.

[59] Yu Zhu, Hao Li, Yikang Liao, Beidou Wang, Ziyu Guan, Haifeng Liu, and Deng Cai. What to do next: Modeling user behaviors by time-lstm. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 3602–3608, 2017. doi: 10.24963/ijcai.2017/504. URL https://doi.org/10.24963/ijcai.2017/504.

[60] Arthur Toth, Louis Tan, Giuseppe Di Fabbrizio, and Ankur Datta. Predicting shopping behavior with mixture of RNNs. In *ACM SIGIR Forum*. ACM, 2017.

[61] Dietmar Jannach, Malte Ludewig, and Lukas Lerche. Session-based item recommendation in e-commerce: on short-term intents, reminders, trends and discounts. *User Modeling and User-Adapted Interaction*, 27:351–392, 2017.

[62] Reid Pryzant, Young joo Chung, and Dan Jurafsky. Predicting sales from the language of product descriptions. In *ACM SIGIR Forum*. ACM, 2017.

[63] Yong Kiam Tan, Xinxing Xu, and Yong Liu. Improved recurrent neural networks for session-based recommendations. *CoRR*, abs/1606.08117, 2016. URL http://arxiv.org/abs/1606.08117.

[64] Yehuda Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 447–456, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-495-9. doi: 10.1145/1557019.1557072. URL http://doi.acm.org/10.1145/1557019.1557072.

[65] Oren Barkan and Noam Koenigstein. Item2vec: Neural item embedding for collaborative filtering. *CoRR*, abs/1603.04259, 2016. URL http://arxiv.org/abs/1603.04259.

[66] Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, Jaikit Savla, Varun Bhagwan, and Doug Sharp. E-commerce in your inbox: Product recommendations at scale. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1809–1818. ACM, 2015.

[67] Veronika Bogina and Tsvi Kuflik. Incorporating dwell time in session-based recommendations with recurrent neural networks. In *Proceedings of the 1st Workshop on Temporal Reasoning in Recommender Systems co-located with 11th International Conference on Recommender Systems (RecSys 2017), Como, Italy,*

August 27-31, 2017., pages 57–59, 2017. URL http://ceur-ws.org/Vol-1922/paper11.pdf.

[68] Xiao Ding, Ting Liu, Junwen Duan, and Jian-Yun Nie. Mining user consumption intention from social media using domain adaptive convolutional neural network. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 2389–2395. AAAI Press, 2015. ISBN 0-262-51129-0. URL http://dl.acm.org/citation.cfm?id=2886521.2886653.

[69] Sherman R. Alpert, John Karat, Clare-Marie Karat, Carolyn Brodie, and John G. Vergo. User attitudes regarding a user-adaptive ecommerce web site. *User Modeling and User-Adapted Interaction*, 13(4):373–396, November 2003. ISSN 0924-1868. doi: 10.1023/A:1026201108015. URL https://doi.org/10.1023/A:1026201108015.

[70] Harrie Oosterhuis and Maarten de Rijke. Ranking for relevance and display preferences in complex presentation layouts. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '18, pages 845–854, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5657-2. doi: 10.1145/3209978.3209992. URL http://doi.acm.org/10.1145/3209978.3209992.

[71] Ron Kohavi, Alex Deng, Roger Longbotham, and Ya Xu. Seven rules of thumb for web site experimenters. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 1857–1866, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2956-9. doi: 10.1145/2623330.2623341. URL http://doi.acm.org/10.1145/2623330.2623341.

[72] Luo Lu and Chuang Liu. Separation strategies for three pitfalls in A/B testing. In *UEO Workshop, KDD*. ACM, 2014.

[73] Henning Hohnhold, Deirdre O'Brien, and Diane Tang. Focusing on the long-term: It's good for users and business. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1849–1858, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3664-2. doi: 10.1145/2783258.2788583. URL http://doi.acm.org/10.1145/2783258.2788583.

[74] Gilles Bailly, Eric Lecolinet, and Laurence Nigay. Visual menu techniques. *ACM Comput. Surv.*, 49(4):60:1–60:41, December 2016. ISSN 0360-0300. doi: 10.1145/3002171. URL http://doi.acm.org/10.1145/3002171.

[75] Leah Findlater and Joanna McGrenere. A comparison of static, adaptive, and adaptable menus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 89–96, New York, NY, USA, 2004. ACM. ISBN 1-58113-702-8. doi: 10.1145/985692.985704. URL http://doi.acm.org/10.1145/985692.985704.

[76] Jean Vanderdonckt, Sara Bouzit, Gaëlle Calvary, and Denis Chêne. Cloud menus: A circular adaptive menu for small screens. In *23rd International Conference on Intelligent User Interfaces*, IUI '18, pages 317–328, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-4945-1. doi: 10.1145/3172944.3172975. URL http://doi.acm.org/10.1145/3172944.3172975.

[77] Apache. Apache Solr. http://lucene.apache.org/solr/, 2018. [Online; accessed 07-06-2018].

[78] Elasticsearch. Elasticsearch. https://www.elastic.co/products/elasticsearch, 2018. [Online; accessed 07-06-2018].

[79] Apache. Apache Lucene. https://lucene.apache.org/, 2018. [Online; accessed 07-06-2018].

[80] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, April 2009. ISSN 1554-0669. doi: 10.1561/1500000019. URL http://dx.doi.org/10.1561/1500000019.

[81] Stephen Robertson, Hugo Zaragoza, and Michael Taylor. Simple BM25 extension to multiple weighted fields. In *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management*, CIKM '04, pages 42–49, New York, NY, USA, 2004. ACM. ISBN 1-58113-874-1. doi: 10.1145/1031171.1031181. URL http://doi.acm.org/10.1145/1031171.1031181.

[82] Jay M. Ponte and W. Bruce Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, pages 275–281, New York, NY, USA, 1998. ACM. ISBN 1-58113-015-5. doi: 10.1145/290941.291008. URL http://doi.acm.org/10.1145/290941.291008.

[83] Liang Wu, Diane Hu, Liangjie Hong, and Huan Liu. Turning clicks into purchases: Revenue optimization for product search in e-commerce. In *The 41st*

International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR '18, pages 365–374, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5657-2. doi: 10.1145/3209978.3209993. URL http://doi.acm.org/10.1145/3209978.3209993.

[84] Aliasgar Kutiyanawala, Prateek Verma, and Zheng (John)Yan. Towards a simplified ontology for better e-commerce search. In *ACM SIGIR Forum*. ACM, 2018.

[85] Cun Mu, Jun Zhao, Guang Yang, Jing Zhang, and Zheng Yan. Towards practical visual search engine within elasticsearch. In *ACM SIGIR Forum*. ACM, 2018.

[86] Arpita Das, Harish Yenala, Manoj Chinnakotla, and Manish Shrivastava. Together we stand: Siamese networks for similar question retrieval. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 378–387. Association for Computational Linguistics, 2016. doi: 10.18653/v1/P16-1036. URL http://www.aclweb.org/anthology/P16-1036.

[87] Isabelle Guyon, Vincent Lemaire, Marc Boullé, Gideon Dror, and David Vogel. Design and analysis of the KDD cup 2009: Fast scoring on a large orange customer database. *SIGKDD Explor. Newsl.*, 11(2):68–76, May 2010. ISSN 1931-0145. doi: 10.1145/1809400.1809414. URL http://doi.acm.org/10.1145/1809400.1809414.

[88] Oracle. Oracle commerce experience manager. https://www.oracle.com/uk/applications/customer-experience/ecommerce/solutions/experience-management.html, 2018. [Online; accessed 07-06-2018].

[89] IBM. IBM websphere application server. https://www.ibm.com/cloud/websphere-application-platform, 2018. [Online; accessed 07-06-2018].

[90] SAP. SAP Hybris C/4HANA. https://www.hybris.com/en/products/digital-portfolio, 2018. [Online; accessed 07-06-2018].

[91] Jasper Snoek, Kevin Swersky, Richard Zemel, and Ryan P. Adams. Input warping for bayesian optimization of non-stationary functions. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1674–II–1682. JMLR.org, 2014. URL http://dl.acm.org/citation.cfm?id=3044805.3045079.

[92] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, pages 2951–2959, USA, 2012. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999325.2999464.

[93] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: A unified engine for big data processing. *Commun. ACM*,

59(11):56–65, October 2016. ISSN 0001-0782. doi: 10.1145/2934664. URL http://doi.acm.org/10.1145/2934664.

[94] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS-W*, 2017.

[95] Maksims Volkovs, Himanshu Rai, Zhaoyue Cheng, Ga Wu, Yichao Lu, and Scott Sanner. Two-stage model for automatic playlist continuation at scale. In *Proceedings of the ACM Recommender Systems Challenge 2018*, RecSys Challenge '18, pages 9:1–9:6, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-6586-4. doi: 10.1145/3267471.3267480. URL http://doi.acm.org/10.1145/3267471.3267480.

[96] Ching-Wei Chen, Paul Lamere, Markus Schedl, and Hamed Zamani. Recsys Challenge 2018: Automatic Music Playlist Continuation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, pages 527–528, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5901-6. doi: 10.1145/3240323.3240342. URL http://doi.acm.org/10.1145/3240323.3240342.

[97] David Ben-Shimon, Alexander Tsikinovsky, Michael Friedmann, Bracha Shapira, Lior Rokach, and Johannes Hoerle. Recsys challenge 2015 and the yoochoose dataset. In *Proceedings of the 9th ACM Conference on Recommender Systems*, RecSys '15, pages 357–358, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3692-5. doi: 10.1145/2792838.2798723. URL http://doi.acm.org/10.1145/2792838.2798723.

[98] Retailrocket. Retailrocket recommender system dataset. https://www.kaggle.com/retailrocket/ecommerce-dataset, 2017. [Online; accessed 01-Feb-2018].

[99] J.W. Tukey. *Exploratory Data Analysis*. Number v. 1 in Exploratory Data Analysis. Addison Wesley Publishing Company, 1970. ISBN 9780608082240. URL https://books.google.co.uk/books?id=HCsSLgEACAAJ.

[100] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009. ISSN 1931-0145. doi: 10.1145/1656274.1656278. URL http://doi.acm.org/10.1145/1656274.1656278.

[101] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[102] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015. URL http://arxiv.org/abs/1512.01274.

[103] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zhang. Tensorflow: A system for large-scale

machine learning. *CoRR*, abs/1605.08695, 2016. URL http://arxiv.org/abs/1605.08695.

[104] David Ben-Shimon. personal communication, 2017.

[105] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. Machine learning: The high interest credit card of technical debt. In *SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop)*, 2014.

[106] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

[107] Steffen Rendle. Factorization machines. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 995–1000, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4256-0. doi: 10.1109/ICDM.2010.127. URL http://dx.doi.org/10.1109/ICDM.2010.127.

[108] Peng Yan, Xiaocong Zhou, and Yitao Duan. E-commerce item recommendation based on field-aware factorization machine. In *Proceedings of the 2015 International ACM Recommender Systems Challenge*, RecSys '15 Challenge, pages 2:1–2:4, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3665-9. doi: 10.1145/2813448.2813511. URL http://doi.acm.org/10.1145/2813448.2813511.

[109] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[110] Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015. URL http://arxiv.org/abs/1506.00019.

[111] Wojciech Zaremba and Ilya Sutskever. Learning to execute. *CoRR*, abs/1410.4615, 2014. URL http://arxiv.org/abs/1410.4615.

[112] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

[113] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[114] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1310–III–1318. JMLR.org, 2013. URL http://dl.acm.org/citation.cfm?id=3042817.3043083.

[115] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014. URL http://arxiv.org/abs/1409.1259.

[116] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6391–6401. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7875-visualizing-the-loss-landscape-of-neural-nets.pdf.

[117] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

[118] Andrew Trotman, Jon Degenhardt, and Surya Kallumadi. The architecture of ebay search. In *ACM SIGIR Forum*. ACM, 2017.

[119] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1135–1144, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939778. URL http://doi.acm.org/10.1145/2939672.2939778.

[120] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. Field-aware factorization machines for CTR prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 43–50, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4035-9. doi: 10.1145/2959100.2959134. URL http://doi.acm.org/10.1145/2959100.2959134.

[121] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased LSTM: accelerating recurrent network training for long or event-based sequences. *CoRR*, abs/1610.09513, 2016. URL http://arxiv.org/abs/1610.09513.

[122] F. A. Gers and J. Schmidhuber. Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 3, pages 189–194 vol.3. IEEE, 2000. ISBN 0-7695-0619-4. doi: 10.1109/ijcnn.2000.861302. URL http://dx.doi.org/10.1109/ijcnn.2000.861302.

[123] Justin Bayer, Daan Wierstra, Julian Togelius, and Jürgen Schmidhuber. Evolving memory cell structures for sequence learning. In Cesare Alippi, Marios Polycarpou, Christos Panayiotou, and Georgios Ellinas, editors, *Artificial Neural Networks – ICANN 2009*, pages 755–764, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-04277-5.

[124] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *IEEE Trans. Neural Netw. Learning Syst.*, 28(10):2222–2232, 2017. doi: 10.1109/TNNLS.2016.2582924. URL https://doi.org/10.1109/TNNLS.2016.2582924.

[125] Junyoung Chung, Çağlar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv e-prints*, abs/1412.3555, 2014. URL https://arxiv.org/abs/1412.3555. Presented at the Deep Learning workshop at NIPS2014.

[126] Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. URL http://arxiv.org/abs/1607.06450.

[127] David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Hugo Larochelle, Aaron C. Courville, and Chris Pal. Zoneout: Regularizing RNNs by Randomly Preserving Hidden Activations. *CoRR*, abs/1606.01305, 2016. URL http://arxiv.org/abs/1606.01305.

[128] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 41–48, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553380. URL http://doi.acm.org/10.1145/1553374.1553380.

[129] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013. URL http://arxiv.org/abs/1308.0850.

[130] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *CoRR*, abs/1206.5533, 2012. URL http://arxiv.org/abs/1206.5533.

[131] Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 343–351, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL http://proceedings.mlr.press/v28/schaul13.html.

[132] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017. URL http://arxiv.org/abs/1707.07012.

[133] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(1):281–305, February 2012. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2503308.2188395.

[134] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3 edition, 2007. ISBN 0521880688, 9780521880688.

[135] Franck Dernoncourt. Interpreting the AUROC. https://stats.stackexchange.com/questions/132777/what-does-auc-stand-for-and-what-is-it, 2015. [Online; accessed 19-07-2018].

[136] Andrej Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness/, 2015. [Online; accessed 01-June-2015].

[137] J. Pearl. The seven pillars of causal reasoning with reflections on machine learning. Technical Report R-481, <http://ftp.cs.ucla.edu/pub/stat_ser/r481.pdf>, Department of Computer Science, University of California, Los Angeles, CA, 2018.

[138] Judea Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, New York, NY, USA, 2nd edition, 2009. ISBN 052189560X, 9780521895606.

[139] Doubleclick. The need for mobile speed: How mobile latency impacts publisher revenue. https://www.doubleclickbygoogle.com/articles/mobile-speed-matters/, 2016. [Online; accessed 19-07-2018].

[140] Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Geometric approximation via coresets. In *COMBINATORIAL AND COMPUTATIONAL GEOMETRY, MSRI*, pages 1–30. University Press, 2005.

[141] Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6338–6347. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7213-poincare-embeddings-for-learning-hierarchical-representations.pdf.

[142] Michael Skinner. Product categorization with LSTMs and balanced pooling views. In *ACM SIGIR Forum*. ACM, 2018.

[143] William Hamilton, Rex Ying, Jure Leskovec, and Rok Sosic. Representation learning on networks. http://snap.stanford.edu/proj/embeddings-www/, 2018. [Online; accessed 15-04-2018].

[144] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Eng. Bull.*, 40:52–74, 2017.

[145] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016. URL http://arxiv.org/abs/1609.02907.

[146] Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM Comput. Surv.*, 27(3):326–327, September 1995. ISSN 0360-0300. doi: 10.1145/212094.212114. URL http://doi.acm.org/10.1145/212094.212114.

[147] Alexander Vezhnevets and Olga Barinova. Avoiding boosting overfitting by removing confusing samples. In *Proceedings of the 18th European Conference on Machine Learning*, ECML '07, pages 430–441, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-74957-8. doi: 10.1007/978-3-540-74958-5_40.

[148] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. SVCCA: Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6076–6085. Curran Associates, Inc., 2017.

[149] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[150] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *CoRR*, abs/1803.01271, 2018. URL http://arxiv.org/abs/1803.01271.

[151] Eugene Culurciello. The fall of RNN / LSTM. https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0, 2018. Accessed: 2018-05-27.

[152] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.