Bayesian Machine Learning for Online Advertising Applications

Biagio Antonelli

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

Abstract

Online advertising is a very fast-changing market; new advertisements are frequently added and it is difficult to predict their efficacy against older ads with proven effectiveness. Without sufficient exploration of new ads, advertisers risk over-exposing users to older ones. While previous literature shows the advantages of exploration via Thompson Sampling from the posterior weight distribution of Bayesian Models, empirical results are either not available or not easily reproducible. Evaluating the benefits of exploration in advertising systems is not trivial, as it is not possible to evaluate the benefits of showing ads which are not displayed. In this work, we provide a novel simulation framework which allows for fast and efficient comparison between several exploration algorithms. Deep Bayesian Models for advertising systems are evaluated for the first time with the framework and may be used as a benchmark for future research. We show the benefits of exploration with online Bayesian Logistic Regression models, while no advantage is found in using exploration with Bayesian Deep Models over greedy Neural Networks.

Acknowledgements

I would like to thank my supervisors Dr Nikolai Anokhin and Dr Sean Murphy, from the Edinburgh Amazon Development Center, for the technical guidance provided throughout the project.

Thanks to all my family, who supported me during this year. A special mention to my brother Diego, for the technical help provided when needed. And to Roberta, who has been sharing with me all my best and worst moments.

Finally, I want to thank all the people I met during this year, especially my Informatics colleagues. I learned a lot from them and I am extremely grateful for that.

Declaration

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

(*Biagio Antonelli*)

Table of Contents

Chapter 1

Introduction

1.1 Online advertising

Modern online advertising is powered by real-time bidding on advertisements, wherein advertisers take part in an auction scheme and bid for the right to place an ad based on their valuation (Varian, (2007); Edelman et al., (2007)). The value placed on an ad by an advertiser is determined based on measurable user behaviors such as clicks, views, and purchases. The ads are selected based on the bid amount and their probability of being clicked, also known as click through rate (CTR). For a set of available slots, eligible ads are chosen and each of them is associated with a bid b_i, a click probability p_i, and an expected revenue of $b_i p_i$. Effective estimation of the click through rate p_i has, therefore, a huge impact on selected ads and the resulting total revenue.

The predicted probability p_i is generated by a classification algorithm. Bayesian Logistic Regression, despite its simplicity, is widely used for CTR predictions (Chapelle et al., 2015). However, Neural Networks, with architectures as in Zhou et al. (2017), have become state-of-the-art due to their powerful feature representation.

1.2 Exploitation-Exploration dilemma

In the fast-changing market of online advertising, new ads are added every few hours. Newly added ads are likely to have a lower CTR score than older ones, some of which have already proven effective. In other words, new ads are always disadvantaged. CTR determine not only the ads to be displayed, but also which ads are included in new batches and it has direct effect on the future quality of the predictions and the

future number of clicks received. *Greedy* policies optimize 1-step returns, ignoring the long-term benefits of exploring new actions, potentially leading to over-exposure of users to some of the ads, profoundly hurting economic efficiency in the long run (Li et al., 2010). This motivates the need to address the exploration-exploitation trade-off (Sutton and Barto, 1998), which we describe in detail in chapter 2. Exploration can uncover the potential of newly added ads and adjust for changes in the probability distribution of CTR, which is non-stationary as preference of users are affected by seasonality trends and keep changing over time.

1.3 Our contribution

Thompson Sampling (Thompson, 1933) is recognized as the most effective way to perform exploration in online advertising systems. Its efficacy has been shown in many works (Graepel et al., (2010); Chapelle and Li, (2011); Chapelle et al., (2015)). However, published results are from experiments performed either on synthetic or non-public data and an empirical analysis of their results is either not provided or not easily reproducible. The effectiveness of Thompson Sampling has been proven only for Linear CTR predictor models, such as Bayesian Logistic Regression, while the effectiveness of Deep Bayesian Models is unknown. Deep Bayesian Models bring the excellent feature representation of Neural Networks while keeping a probability distribution over its parameters, which can be used for effective exploration via Thompson Sampling. These properties suggest that Deep Bayesian Models might be particularly suitable for online advertising systems and beat existing state-of-the-art solutions.

The goal of this project is to provide an evaluation framework for exploration strategies in online advertising and to evaluate Deep Bayesian models on the task of sequential ad recommendation. The framework, used successfully for our experiments, allows for the easy comparison of different algorithms subject to evaluation. It has been used to first perform baseline experiments with Bayesian Logistic Regression, and also to evaluate two variations of Bayesian Neural Networks (Gal and Ghahramani, (2016); Blundell et al., (2015)) with Thompson Sampling exploration in online advertising tasks. Although we do not find any evidence supporting our hypothesis of the potential benefits of using deep Bayesian models, the provided framework together with the performed experiments are a valid benchmark which may be used to compare newly

developed models and exploration strategies.

1.4 Thesis Structure

The work is structured as follows:

- Chapter 2 provides background notions on online advertising systems and bandit algorithms.
- Chapter 3 provides background on the Bayesian Machine Learning models that will be used in our experiments.
- Chapter 4 describes the proposed simulation framework, and the implementation details of all our models and experiments.
- Chapter 5 presents the results of our experiments.
- Chapter 6 provides conclusions for our work and sets promising new avenues for future research.

Chapter 2

Background

2.1 Online advertising

The online advertising process involves many parties. A user requests online content, which is made available from a website publisher. Together with the content, the publisher displays, depending on the available space, one or more ads. The advertising system could be managed in several ways: the publisher could be itself the advertising operator; can outsource the advertising management to a third party agency; can put for sale ad space in a bidding market, ad-exchange, where many parties interact simultaneously in real-time. In the last scenario, the user request is served by the publisher content server, which returns the desired webpage, initially without any ad. Available spaces together with user information, such as *cookies*, are sent to the publisher ad server which transmits them to a Supply-Side Platform (the publisher is supplying ad space). The platform retrieves additional data about the users such as demographics and previous purchases. Once all the relevant info are collected, the ad space is offered in the ad exchange platform. Here, demand-side platforms (DSP), who acts as an interface to agencies representing brands, have to bid, if interested in the offered space, within a few milliseconds.

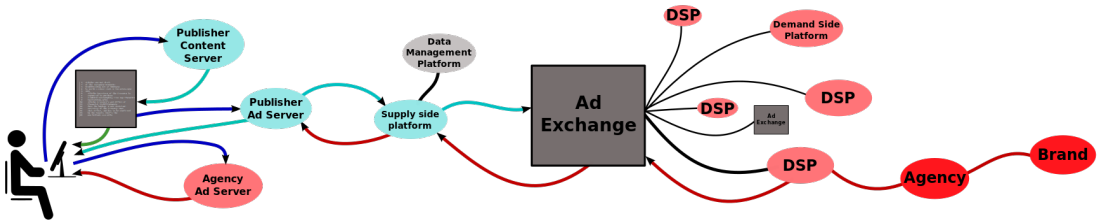

Figure 2.1: Advertsing process. Image from: `https://en.wikipedia.org/wiki/Online_advertising`

2.2 Auctions

Once received all the bids from the demand-side platforms, the ad space is allocated based on the outcome of an auction. If the auction mechanism is well designed, advertisers should bid their true evaluation of that space. An auction scheme with this property is called *incentive compatible*.

2.2.1 Keyword Auctions

In the case of keyword auction, advertisers find keywords likely to be typed when users are potentially interested in their product and make a bid for each of such keywords. The bid represent the price each advertiser would be willing to pay for each click received (Graepel et al., 2010). When one of the keywords is typed, all the advertisers who bid for it take part in an auction scheme. There exists several auction mechanisms, each one with different properties and theoretical guarantees. One of the most widely used is the Generalized Second Price auction.

2.2.2 Generalized Second Price Auctions

In a generalized second price auction (GSP), each advertiser i chooses a single value $b_i \in \mathbb{R}_{\geq 0}$. The mechanism assumes that there is a common preference for slots, each bidder prefers slot 1 to slot 2, slot 2 to slot 3 and so on. Each slot is associated with a click-through rate p_j, with $p_j \geq p_{j+1}$. Allocation is based on the rank score $p_i b_i$, which can be interpreted as expected revenue for the ad i. The item i is selected according to their ranking $p_i b_i \geq p_{i+1} b_{i+1}$. To avoid dynamic bidding behavior, the price advertiser i has to pay for the slot j c_{ij} is based on the rank score of ad $i+1$. If there are fewer bidders than slots, then the advertiser in the last position pay a reserve price r. In such an auction mechanism, truth-telling is not a dominant strategy, and as a consequence, the mechanism is not *incentive compatible*. However, in the long run, such mechanism tend to an equilibrium which is the same as the Vickrey-Clarke-Groves (VCG) mechanism (Nisan et al., 2007). This result is desirable as in the VCG mechanism truth-telling is a dominant strategy and advertisers are not incentivized to lie on their evaluation. However, VCG for multi-items auctions is an NP-hard problem, for this reason, other mechanisms with similar properties are used. Despite being not *incentive compatible*, in GSP the total revenue is at least as high as the VCG one.

2.3 Online decision-making

Having received all the bids, one could select the ad to display picking the ad i with the highest rank score $p_i b_i$. However, online decision making systems have to deal with a fundamental dilemma: *exploit* or *explore*? While in supervised machine learning we build the best possible model based on the data we have, in sequential decision-making systems the data we observe in the future and the final quality of our systems depends on our decision throughout the whole process. The problem has been extensively studied in the Reinforcement Learning literature (Sutton and Barto, 1998) and is referred to as Multi-Armed Bandits problem.

2.4 The multi-armed bandits problem

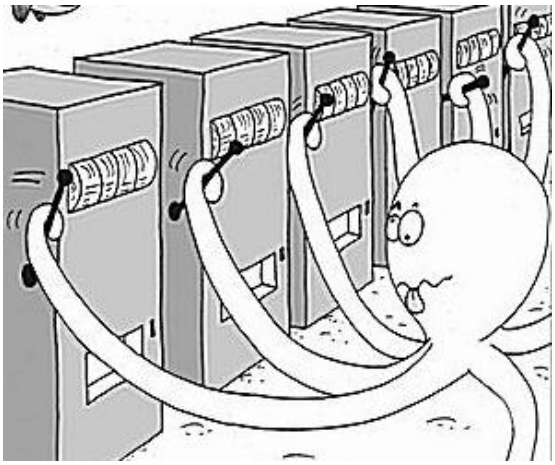

Figure 2.2: A multi-armed bandit. Figures from: `http://slivkins.com/work/bandits-svc/` .

A multi-armed bandit is a tuple $\langle \mathcal{A}, \mathcal{R} \rangle$ where $\mathcal{A}$ is a set of m actions and $\mathcal{R}^a(r) = p(r|a)$ is an unknown probability distribution over rewards. The problem of multi-armed bandits can be put as follow: an agent repeatedly faces a choice over m actions or arms $a \in \mathcal{A}$, after each action taken, it gets a rewards $r \in \mathcal{R}$ coming from a stationary distribution $p(r|a)$ conditioned on the selected action. The objective is to maximize the cumulative reward over time (Sutton and Barto, 1998). The Reinforcement Learning approach to the problem is to estimate a value function for each action a, $Q_t(a)$ at each time-step t. When the true value function $q_*(a)$ is known, at each time-step we know exactly which action to select. However, by keeping an estimation of the value function $Q_t(a)$, at each time-step there will be an action with maximum value, and

the agent can make decisions accordingly. After each action taken, the agent observes a reward r_t from the environment, which can be used to update its estimation of the action-value, iteratively improving it until it gets closer to the real one.

Explore vs Exploit. We only observe the rewards for actions we select, as the agent drive the generative process. If it always selects the action with the highest value, *greedy* action selection, the agent is said to be *exploiting* its current knowledge. Exploitation is the strategy which maximizes the 1-step expected reward. However, *exploration* may allow a better long-term reward as it grants a better knowledge of the environment. Whether it is better to explore or exploit depends on different factors such as the current time step, values estimation and uncertainty.

2.5 Contextual Bandits

In the contextual bandit setting, at each time step we receive a context $x_t \sim \mathcal{X}$ and select an action based on a model $f(a,x) = p(r|a,x)$. We then observe the reward r_t and update the internal state of f based on the tuple $\langle a_t, x_t, r_t \rangle$. At each timestep t:

- The environment generates a context $x_t \in \mathcal{X}$
- The agent select an action $a_t \in \mathcal{A}$
- The environment generates a reward r_t

In this case the *action-value* function depends also on the context $Q_t(x,a) \approx \mathbb{E}[r|x,a] = q_*(x,a)$. It is possible to approximate the *action-value* function with a parametrized model. As an example, in the case of linear regression, by indicating with $\phi(x,a)$ a feature vector representing the context-action interaction, we can calculate the value function as:

$$Q_t(x,a) = \phi(x,a)^T \mathbf{w} \tag{2.1}$$

As we acquire tuples $\langle a_t, x_t, r_t \rangle$, we can build the matrices:

$$\Phi_t = \begin{bmatrix} \phi(x_1, a_1)^T \\ \phi(x_2, a_2)^T \\ \vdots \\ \phi(x_T, a_t)^T \end{bmatrix} \qquad r_t = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_t \end{bmatrix} \tag{2.2}$$

Finally, we compute the parameters exactly with the Normal Equation:

$$\mathbf{w} = (\Phi_t^T \Phi_t)^{-1} \Phi_t^T r_t \tag{2.3}$$

Of course, it is possible to use any regression model including Deep Neural Networks and fit the parameters with gradient-based optimization. In our work, the value function is approximated with both linear and deep models. The value function represents the CTR and the actions selected are the ads with the highest probability of being clicked.

Regret. The regret is defined as the opportunity loss in a single time-step:

$$l_t = \mathbb{E}[V_t^* - Q_t(x, a)] \tag{2.4}$$

where V_t^* is the maximum reward achievable at time t. Minimizing the total regret over time is equivalent to maximize the cumulative reward.

Online advertising systems. In online advertising systems, the context **x** is represented by the webpage and user information, while the set of eligible ads represents the possible actions (arms). We can represent each interaction (context-ad) as feature vector $\phi(x, a)$. When ads are displayed, we observe an outcome $y \in \{\text{click}, \text{non-clicked}\}$ and update our CTR predictor.

2.6 Exploration strategies

The goal of exploration strategies is to minimize the total regret. From the Lai and Robbins Theorem (Lai and Robbins, 1985), we know that the asymptotic total regret is at least logarithmic with the number of steps. Therefore we seek for an exploration algorithm able to achieve such lower bound.

2.6.1 ε-greedy

ε-greedy is the most popular exploration strategy in Reinforcement learning as it is straightforward to implement. It consist of selecting with probability $1-\varepsilon$ an action a such that $a = argmax_{a\in\mathcal{A}}\ \hat{Q}(a)$, while with probability ε picking an action at random. Unfortunately, ε-greedy total regret is linear with the number of steps.

2.6.2 Upper Confidence Bound

ε-greedy forces exploratory action to be taken indiscriminately, it would be preferable to select exploratory actions based on their potential for being optimal, considering both how they are close to being maximal and their uncertainty, one way of doing this is to select actions according to:

$$a_t = argmax_{a\in\mathcal{A}} \left[Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}} \right] \tag{2.5}$$

where $N_t(a)$ indicates the number of times the action a has been selected before the current time-step t. The square root term provides an indication of our uncertainty about that action and $c > 0$ controls the amount of exploration.

2.6.3 Thompson Sampling

Thompson Sampling requires that one can sample over a posterior distribution over plausible problem instances, such as regret or rewards. In the case of contextual bandits, we have a distribution over the value function parameters and exploration is performed by sampling a weight vector $\mathbf{w} \sim p(\mathbf{w}|D)$ and select a *greedy* action based on that sample.

Thompson Sampling algorithms achieve the Lai and Robbins lower bound leading to a near-optimal cumulative regret. Furthermore, the randomized predictions from Thompson Sampling do not affect the GSP mechanism, which is still incentive compatible (Meek et al., 2005).

Algorithm 1 Thompson Sampling

Input: prior distribution over model parameters $p(\mathbf{w})$
Input: $\mathcal{D} = \emptyset$
for $t = 1, .., T$ **do**
    Receive context $\mathbf{x}_t$
    Draw $\mathbf{w} \sim p(\mathbf{w}|\mathcal{D})$
    Select $a_t = argmax_a \mathbb{E}_r(r|\mathbf{x}_t, a, \mathbf{w}_t)$
    Observe reward r_t
    $\mathcal{D} = \mathcal{D} \cup \langle \mathbf{x}_t, a_t, r_t \rangle$
    Update posterior with $\langle \mathbf{x}_t, a_t, r_t \rangle$

2.7 Previous work

The effectiveness of Thompson Sampling has been proved several times both in theory (Agrawal and Goyal, (2012); Russo et al., (2017)) and in practice. Previous practical implementation of Thompson Sampling for online advertising applications include the work of Graepel et al. (2010), who proposes the use of online Bayesian Probit regression. The author showed the advantages of having a posterior distribution over the model parameters both for drawing samples for the Thompson Sampling procedure and as an informative prior for future updates. However, no empirical analysis of such results is provided. Chapelle and Li (2011) and Chapelle et al. (2015) propose a similar method, the authors use an online Bayesian Linear Regression model, where the posterior is calculated through Laplace approximation (Bishop, 2006). They provide empirical evidence of the advantage of using Thompson Sampling over *greedy* and other bandit policies, using an offline simulation. The probability of click for a displayed ad is modeled as a $\sigma(-\mathbf{w}^*\mathbf{x})$, where $\sigma(a) = (1 + exp(-a))^{-1}$ is the Logistic function, such distribution is then used to generate clicks for unobserved context-ad interactions and evaluate different policies. Li et al. (2011) propose an alternative evaluation algorithm. A *replayer*, which uses randomized exploration data to produce an offline estimator of the evaluated policy. Unfortunately, the algorithm does not work well when, as in online advertising applications, the number of arms is large.

For deep Neural Networks, it is generally not possible to access the full posterior. However, recent methods allow sampling from an approximate posterior directly (Blundell et al., 2015) or to obtain samples via Monte Carlo Dropout (Gal and Ghahramani, 2016). Such newly developed techniques for approximate inference in Deep Models

have not been tested for Thompson Sampling in practical scenarios. Some of these models have been mostly evaluated on toy datasets for Reinforcement Learning applications. As an example, Dropout (Gal and Ghahramani, 2016) was tested in a simulated 2D environment where a 9 eyes agent has to collect red circles to maximize its reward. On the other hand, Blundell et al. (2015) evaluates Thompson sampling on the UCI Mushroom dataset (Bache and Lichman, 2013), casting it as a bandit task similar to Guez (2015). In both these works, exploration with Thompson Sampling outperforms *greedy* policies. In addition, recently, Riquelme et al. (2018) developed a benchmark to test several Bayesian deep bandit models on multiple Reinforcement Learning tasks. They noticed some weakness of Deep Bayesian models in online decision-making scenarios, highlighting the need for adapting the slowly converging uncertainty estimates to the online setting.

Chapter 3

Bayesian Machine Learning

Deep learning has been a revolution for machine learning. However, these models are not able to correctly representing uncertainty by leveraging probability theory tools. Due to recent failures in modeling uncertainty, which caused significant engineering and social failures (Kendall and Gal, 2017), there has been an increasing amount of interest in Bayesian techniques from the machine learning community.

The first applications of Bayesian machine learning can be dated back to 1990. The works of Neal (1995), MacKay (1992), and Dayan et al. (1995) provided the first tools to reason about models uncertainty. Such tools did not adapt themselves to the big data explosion and need for high computing performances. Whereas, a significant part of the success of Neural Networks models is due to their ability to be trained fast on massive datasets, with stochastic optimization (Bottou, 2010) and the backpropagation algorithm (Rumelhart et al., 1986). This, together with innovations (Srivastava et al., (2014); Nair and Hinton, (2010)), and improvement in the hardware, lead to great successes in speech recognition (Hinton et al., 2012), Natural Language Processing (Sutskever et al., 2014), and Computer Vision (Krizhevsky et al., 2012).

From elementary decision theory, we know that any decision rule which is not Bayesian can be strictly improved (Cox and Hinkley, 1974). Given any parametrized model, we can see it as a Bayesian inference problem by placing a prior distribution over the weights vector $p(\mathbf{w})$, after observing a set of data $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$, we can compute the posterior belief with Bayesian inference.

$$p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})} \tag{3.1}$$

Bayesian Machine Learning models use Bayes rule to update their beliefs given new data. In this chapter, we provide background for the Bayesian Machine Learning models we use in our experiments.

3.1 Bayesian Linear Regression

To give an example of how we can threat a simple linear machine learning problem as a Bayesian inference problem, let's consider first the case of Bayesian Linear Regression. As shown in Murphy (2012), given a data sample $\mathbf{x}$ and a model weights vector $\mathbf{w}$, the response can be expressed as function of the input in the form $y = \mathbf{w}^T\mathbf{x} + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. It is possible to see linear regression as a probabilistic model and write its predictive distribution as:

$$p(y|\mathbf{x}, \theta) = \mathcal{N}(y|\mathbf{w}^T\mathbf{x}, \sigma^2) \tag{3.2}$$

Maximum Likelihood Estimation (MLE) provides the same solution as ordinary Linear Regression. If σ is constant, given a set of observation $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$:

$$-logp(y|\mathbf{X}, \mathbf{w}) = -\sum_n \log\ p(y^{(n)}|\mathbf{x}^{(n)}, \mathbf{w}) = \frac{1}{2\sigma^2}\sum_n (y^{(n)} - \mathbf{w}^T\mathbf{x}^{(n)})^2 + \frac{N}{2}log(2\pi\sigma^2) \tag{3.3}$$

Minimizing the negative log loss (NLL) is equivalent to minimizing the square error.

On the other hand, Bayesian inference needs a prior, we assume a Gaussian prior over weights $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}, 0, \sigma_w^2\mathbf{I})$ and are interested in computing the posterior:

$$p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})} \propto p(\mathcal{D}|\mathbf{w})p(\mathbf{w}) \tag{3.4}$$

The posterior can also be written as:

$$p(\mathbf{w}|\mathcal{D}) \propto p(\mathcal{D}|\mathbf{w})p(\mathbf{w}) \tag{3.5}$$
$$\propto \mathcal{N}(\mathbf{w}, \mathbf{0}, \sigma_w^2 \mathbb{I}) \prod_n \mathcal{N}(y^{(n)}; \mathbf{w}^T \mathbf{x}^{(n)}, \sigma^2) \tag{3.6}$$
$$\propto \mathcal{N}(\mathbf{w}, \mathbf{0}, \sigma_w^2 \mathbb{I}) \mathcal{N}(y; \mathbf{X}\mathbf{w}, \sigma^2 \mathbb{I}) \tag{3.7}$$

Which can be computed exactly (Murphy, 2012) as $\mathcal{N}(\mathbf{w}; \mathbf{w}_N, \mathbf{V}_N)$, where :

$$\mathbf{V}_N = \sigma^2(\sigma^2 \mathbf{V}_0^{-1} + \mathbf{X}^T \mathbf{X})^{-1} \tag{3.8}$$
$$\mathbf{w}_N = \mathbf{V}_N \mathbf{V}_0^{-1} \mathbf{w}_0 + \frac{1}{\sigma^2} \mathbf{V}_N \mathbf{X}^T y \tag{3.9}$$

where $\mathbf{V}_0 = \sigma_w^2 \mathbb{I}$ and $\mathbf{w}_0 = \mathbf{0}$. Bayesian Linear Regression prediction can be expressed as:

$$p(y|\mathbf{x}, \mathcal{D}) = \int p(y, \mathbf{w}|\mathbf{x}, \mathcal{D}) d\mathbf{w} \tag{3.10}$$
$$= \int p(y|\mathbf{w}, \mathbf{x}) p(\mathbf{w}|\mathcal{D}) d\mathbf{w} \tag{3.11}$$

which can be computed in closed form.

3.2 Bayesian Logistic Regression

In Bayesian Logistic Regression, given a data sample $\mathbf{x}$ and a model weights vector $\mathbf{w}$, the prediction can be computed as:

$$p(y = 1|\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}} \tag{3.12}$$
$$\tag{3.13}$$

The predictive distribution is:

$$p(y|\mathbf{x}, \mathcal{D}) = \int p(y, \mathbf{w}|\mathbf{x}, \mathcal{D}) d\mathbf{w} \tag{3.14}$$
$$= \int p(y|\mathbf{w}, \mathbf{x}) p(\mathbf{w}|\mathcal{D}) d\mathbf{w} \tag{3.15}$$

which in the case of Logistic Regression is intractable.

3.3 Approximate Inference

3.3.1 Laplace Approximation

There are several approximate inference techniques which aim to provide a solution for the integral in Eq. 3.15. MAP approximation, find an approximation of the posterior parameters by minimizing an **energy function** (Murphy, 2012):

$$E(\mathbf{w}) = -logp(\mathbf{w}, \mathcal{D}) \tag{3.16}$$

The mode of the posterior can be fitted by minimizing the energy function, usually using gradient based optimization:

$$\mathbf{w}^* = argmin_{\mathbf{w}} E(\mathbf{w}) \tag{3.17}$$

while the *Hessian* matrix elements give an indication on how sharp is the distribution across different directions:

$$H_{ij} = \left. \frac{\partial^2 E(\mathbf{w})}{\partial w_i \partial w_j} \right|_{\mathbf{w}=\mathbf{w}^*} \tag{3.18}$$

The posterior can then be computed as $p(w|D) = \mathcal{N}(\mathbf{w}; \mathbf{w}^*, H^{-1})$.

3.3.2 Stochastic Variational Inference

Laplace approximation is not always the best way to approximate the posterior. Variational inference provides an alternative. Variational inference aims to fit the posterior distribution by minimizing a distance measure between the true posterior and a simpler parametrized distribution $q(\mathbf{w}, \theta = \{\mu, V\})$. This is done by minimizing the Kullback-Leibler (KL) distance between the two distributions.

$$D_{\mathrm{KL}}(q(\mathbf{w},\theta) \,||\, p(\mathbf{w})) = \int q(\mathbf{w},\theta) \log \frac{q(\mathbf{w},\theta)}{p(\mathbf{w}))} \, \mathrm{d}\mathbf{w} \tag{3.19}$$

$$= -\int q(\mathbf{w},\theta) \log p(\mathbf{w})) \, \mathrm{d}\mathbf{w} + \int q(\mathbf{w},\theta) \log q(\mathbf{w},\theta) \, \mathrm{d}\mathbf{w} \tag{3.20}$$

By replacing $p(\mathbf{w}|\mathcal{D})$ with Bayes rule, the following equation can be derived[1]:

$$D_{\mathrm{KL}}(q(\mathbf{w},\theta)\,||\,p(\mathbf{w})) = \underbrace{\mathbb{E}_q[\log q(\mathbf{w})] - \mathbb{E}_q[\log p(\mathcal{D}|\mathbf{w})] - \mathbb{E}_q[\log p(\mathbf{w})]}_{J(q)} + \log p(\mathcal{D}) \tag{3.21}$$

$J(q)$ is called **variational free energy** in statistical physics and is the expected negative log loss plus a penalty (Murphy, 2012), which depends on how far away the the approximate posterior falls from the prior:

$$J(q) = \mathbb{E}_q[\log q(\mathbf{w}) - \log p(\mathcal{D}|\mathbf{w}) - \log p(\mathbf{w})] \tag{3.22}$$

$$= \mathbb{E}_q[-\log p(\mathcal{D}|\mathbf{w})] + D_{\mathrm{KL}}(q(\mathbf{w},\theta)\,||\,p(\mathbf{w})) \tag{3.23}$$

Recently Evidence Lower Bound (ELBO) has become a popular term to define $-J(q)$. The variational posterior can be fitted by minimizing the lower bound.

3.4 Gaussian Processes

The optimal approach to modelling uncertainty in machine learning models it to infer a distribution over functions given our data $p(f|\mathbf{X},y)$ and use it to compute predictions given a new input $\mathbf{x}_*$ (Murphy, 2012):

$$p(y_*|\mathbf{x}_*,\mathbf{X},y) = \int p(y_*|f,\mathbf{x}_*)\,p(f|\mathbf{X},y))\,\mathrm{d}f \tag{3.24}$$

Gaussian Processes (Rasmussen, 2004) is the most traditional and desirable machine learning approach to model uncertainty, and are often used as a benchmark for comparing the uncertainties of novel Bayesian machine learning models (Gal and Ghahramani, (2016); Osband et al., (2016)). The prohibitive $O(N^3)$ cost, where N is the number of samples, needed to fit such models, has drastically limited their adoption in most of the real-world applications, despite lots of efforts (Seeger et al., (2003); Quiñonero Candela et al., (2010)) have been put in reducing it. Nevertheless, GPs are very popular in several areas of machine learning such as for Bayesian hyperparameter optimization (Snoek et al., 2012).

[1] derivation from the MLPR 2017 class note: http://www.inf.ed.ac.uk/teaching/courses/mlpr/2017/notes/

We can use GPs for regression by placing a GP prior over functions:

$$f(x) \sim GP(m(\mathbf{x}),\ \kappa(\mathbf{x},\mathbf{x}^{'})) \tag{3.25}$$

where $m(\mathbf{x})$ is the mean and $\kappa(\mathbf{x},\mathbf{x}^{'})$ the kernel or covariance function.

$$m(\mathbf{x}) = \mathbb{E}[f(x)] \tag{3.26}$$
$$\kappa(\mathbf{x},\mathbf{x}^{'}) = \mathbb{E}[(f(\mathbf{x}-m(\mathbf{x})(f(\mathbf{x}^{'}-m(\mathbf{x}^{'})^{T}] \tag{3.27}$$

Prediction in Gaussian Processes can be computed by considering that the joint distribution has the following form:

$$p\left(\begin{bmatrix} f \\ f_* \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} f \\ f_* \end{bmatrix}; \begin{bmatrix} \mu \\ \mu_* \end{bmatrix}, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right), \tag{3.28}$$

where $K(X,X)_{i,j} = \kappa(\mathbf{x}_i,\mathbf{x}_j)$ is $N \times N$ and $\mu = (m(\mathbf{x_1},...,m(\mathbf{x}_N))$ is $N \times 1$ and the observation are assumed to be noise free. The conditional can be then computed by standard rules of conditional Gaussians:

$$p(f_*|f,X,X_*) = \mathcal{N}(f_*;\mu_*,\Sigma_*) \tag{3.29}$$

where:

$$\mu_* = \mu(X_*) + K(X_*,X)K(X,X)^{-1}(f-\mu(X)) \tag{3.30}$$
$$\Sigma_* = K(X_*,X_*) - K(X_*,X)(K(X,X))^{-1}K(X,X_*) \tag{3.31}$$

3.5 Bayesian Neural Networks

Similarly to what we showed for Bayesian Linear and Logistic regression, it is possible to place a prior over a Neural Network parameters and update the model with Bayesian inference. In addition to provide information on the uncertainty in our model, Bayesian approaches to Neural Networks are also less prone to overfitting and offer several other advantages such as learning the stability of the trained model by analyzing the weights variance, allowing transfer learning with informative priors and model pruning by removing the weight with low signal to noise ratio (Blundell et al., 2015).

Bayesian inference for Neural Network is intractable. Early approaches attempted to fit an approximate posterior with Laplace approximation (MacKay, 1992), Hamiltonian Monte Carlo (Neal, 1995) and variational inference (Hinton and Van Camp, 1993). However, these approaches lack scalability and did not see widespread adoption in practice. Graves (2011) proposed a first scalable variational inference approach, but the method performs poorly in practice due to the noise in the Monte Carlo approximation of the stochastic gradient computation (Hernández-Lobato and Adams, 2015). Recent advances in variational inference techniques, such as *sampling-based* variational inference and *stochastic* variational inference (Kingma and Welling, 2013), have been used to improve approximate inference in Bayesian Neural Network, such as in Blundell et al. (2015).

3.5.1 Bayes by backpropagation

Given a sample input $\mathbf{x}$, we can see a Neural Network as a probabilistic model $p(y|\mathbf{x},\mathbf{w})$ and assign a probability to each possible output $y \in \mathcal{Y}$, where for classification tasks $\mathcal{Y}$ is a set of classes, while for regression $\mathcal{Y}$ is $\mathbb{R}$.

The predictive distribution for a new sample $\mathbf{x}$ is given by:

$$p(y|\mathbf{x}) = \mathbb{E}_{p(\mathbf{w}|\mathcal{D})}\left[p(y|\mathbf{x},\mathbf{w})\right] \tag{3.32}$$

Taking the expectation under the posterior distribution over the weights is equivalent to using an ensemble of an infinite number of Neural Networks. We can fit a simple approximate posterior distribution $q(\mathbf{w}|\theta)$ of the true posterior $p(\mathbf{w}|\mathcal{D})$ by minimizing the **variational free energy**:

$$\mathcal{F}(\mathcal{D},\theta) = D_{\mathrm{KL}}(q\,||\,p) - \mathbb{E}_{q(\mathbf{w}|\theta)}\left[\log p(\mathcal{D}|\mathbf{w})\right] \tag{3.33}$$

The cost function in equation 3.33 is divided in a data-dependent part and a prior-dependent part. Blundell et al. (2015) name the data-dependent part as the *likelihood* cost and the prior-dependent part as the *complexity* cost. The cost function is, therefore, such that there is a trade-off between satisfying the complexity of the data $\mathcal{D}$ while not moving too far away from the simplicity of the prior $p(\mathbf{w})$.

Proposition 1(from Blundell et al. (2015). *Let* ε *be a random variable with probabiity distribution* $q(\varepsilon)$ *and* $\mathbf{w} = t(\theta, \varepsilon)$*, where* $t(\theta, \varepsilon)$ *is a deterministic function. Suppose that the marginal probability distribution of* $\boldsymbol{w}$ *,* $q(\mathbf{w}|\theta)$ *is such that* $q(\varepsilon)d\varepsilon = q(\mathbf{w}|\theta)d\mathbf{w}$*. Then we have:*

$$\frac{\partial}{\partial \theta}\mathbb{E}_{q(\mathbf{w}|\theta)}\left[f(\mathbf{w},\theta)\right] = \mathbb{E}_{q(\varepsilon)}\left[\frac{\partial f(\mathbf{w},\theta)}{\partial \mathbf{w}}\frac{\partial \mathbf{w}}{\partial \theta} + \frac{\partial f(\mathbf{w},\theta)}{\partial \theta}\right] \tag{3.34}$$

This result allows using a backpropagation like algorithm for the optimization problem in 3.33. Such an algorithm is named *Bayes by Backprop* by its authors. The exact cost (variational free energy) is approximated as:

$$\mathcal{F}(\mathcal{D},\theta) \approx \sum_{i=1}^{n} \log q(\mathbf{w}^{(i)}|\theta) - \log p(\mathbf{w}^{(i)}) - \log p(\mathcal{D}|\mathbf{w}^{(i)}) \tag{3.35}$$

where $\mathbf{w}^{(i)}$ is a Monte Carlo sample from the variational posterior.

The variational posterior parameters used are $\theta = \{\mu, \rho\}$ where ρ is related to the standard deviation $\sigma = \log\left(1 + e^{\rho}\right)$. Supposing we have a diagonal Gaussian variational posterior, it is possible to obtain a sampled weight vector as $\mathbf{w} = \sigma \circ \varepsilon + \mu$. Where $\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbb{I})$. The optimization then proceed as follows:

1. Sample $\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbb{I})$
2. Let $\mathbf{w} = \log\left(1 + e^{\rho}\right) \circ \varepsilon + \mu$
3. Let $\theta = \{\mu, \rho\}$
4. Let $f(\mathbf{w}, \theta) = \log q(\mathbf{w}^{(i)}|\theta) - \log p(\mathbf{w}^{(i)}) \log p(\mathcal{D}|\mathbf{w}^{(i)})$
5. Calculate gradients with respect to the mean:

$$\Delta_{\mu} = \frac{\partial f(\mathbf{w},\theta)}{\partial \mathbf{w}} + \frac{\partial f(\mathbf{w},\theta)}{\partial \mu}$$

6. Calculate gradients wth respect to the standard deviation parameter:

$$\Delta_{\rho} = \frac{\partial f(\mathbf{w},\theta)}{\partial \mathbf{w}} \frac{\varepsilon}{1 + e^{-\rho}} + \frac{\partial f(\mathbf{w},\theta)}{\partial \rho}$$

7. Update variational parameters:

$$\mu \leftarrow \mu - \alpha\Delta_\mu$$

$$\rho \leftarrow \rho - \alpha\Delta_\rho$$

where α is the selected learning rate and $\frac{\partial f(\mathbf{w},\theta)}{\partial \mathbf{w}}$ is the same gradient term as the usual backpropagation algorithm.

3.5.2 Dropout as Bayesian approximation

Bayesian models come with a mathematical framework to explain uncertainty in deep networks. However, this is almost always at the expenses of a prohibitive computational cost. As an example, the number of parameters required in the *Bayes by backprop* algorithm (Blundell et al., 2015), described in Section 3.5.1, is double the typical Networks, as for each hyperparameter we have to fit mean and variance.

Gal and Ghahramani (2016) show that dropout can be interpreted as a Bayesian approximation of Deep Gaussian Processes (Damianou and Lawrence, 2013). Given a Neural Network with L layers and $\{\mathbf{W}_i\}_{i=1}^{L}$ as a set of weights for the L layers. By sampling T sets of realization vectors from a Bernoulli distribution $\{\mathbf{z}_1^t,...,\mathbf{z}_L^t)\}_{t=1}^{T}$ which give the weight matrices after dropout $\{\mathbf{W}_1^t,...,\mathbf{W}_L^t\}$, we can compute the predictive mean and variance of the approximate Deep Gaussian Process as follow:

$$\mathbb{E}_{q(y_*|\mathbf{x}_*)}(y_*) \approx \frac{1}{T}\sum_{t=1}^{T}\hat{y}_*(\mathbf{x}_*,\mathbf{W}_1^t,...,\mathbf{W}_L^t) \tag{3.36}$$

where $\hat{y}_*$ is the output of the Neural Network. The variance can be computed as:

$$\begin{aligned}\mathrm{Var}_{q(y_*|\mathbf{x}_*)}(y_*) \approx \tau^{-1}\mathbb{I}_D \\ + \frac{1}{T}\sum_{t=1}^{T}\hat{y}_*(\mathbf{x}_*,\mathbf{W}_1^t,...,\mathbf{W}_L^t)^T\hat{y}_*(\mathbf{x}_*,\mathbf{W}_1^t,...,\mathbf{W}_L^t) \\ - \mathbb{E}_{q(y_*|\mathbf{x}_*)}(y_*)^T\mathbb{E}_{q(y_*|\mathbf{x}_*)}(y_*)\end{aligned} \tag{3.37}$$

where τ is a precision factor, derivation of the above formula and proof of the Deep Gaussian Process approximation can be found in the original paper. Once obtained this convenient approximation, Thompson Sampling can be conveniently computed with

Dropout Networks by merely performing a stochastic forward pass. No modification to the traditional training with dropout (Srivastava et al., 2014) is needed. Furthermore, the forward pass can be parallelized. Therefore, no extra training time is needed for training in bandits settings.

Chapter 4

Implementation

In this chapter, we describe the practical implementation of some of the algorithms introduced in section 3, the data preprocessing used and a framework for the off-line evaluation of bandit algorithms.

4.1 Dataset

We run our experiments on the Criteo dataset[1]. The training data is part of the Criteo traffic over a 7 day period, with each row corresponding to a served ad by the Criteo platform. Online advertising datasets are generally highly imbalanced as most of the users do not click on the displayed ads. Building a classifier for imbalanced datasets has been a core problem for many applications and there exist many possible ways to deal with class imbalance, Branco et al. (2015) offers an excellent survey of the topic. However, as dealing with class imbalance is entirely beyond the scope of this work, we select a version of the dataset where the positive (clicked) and negative (non-clicked) have been already subsampled at different rates, with a resulting positive rate of 25%.

Preprocessing. The original dataset comes with 13 integer features and 26 categoricals. The integer features are transformed into categoricals via *quantile binning*. We use the *hashing trick* to encode all the categorical features as proposed in Chapelle et al. (2015). Unlike classical *1-hot encoding*, the hashing trick avoids dimensionality explosion when encoding variables of high cardinality and it is easy to implement.

[1] https://www.kaggle.com/c/criteo-display-ad-challenge/data

4.2 Proposed evaluation method

In this section, we refer to each algorithm as an agent. Each agent has an internal model $f_a(\mathbf{x};\mathbf{w}) = \hat{p}(y=1|\mathbf{x})$, which provides an estimation of the click probability (CTR). The goal is to maximize the cumulative reward $\sum_{\tau=1}^{t} r_\tau$, or equivalently (see Sec. 2.5) minimizing the total regret. The agent is *greedy* if it always act in a way to maximize the 1-step reward. Whereas, it is *exploring* if it sometimes sacrifices the 1-step reward, to achieve an advantage in the long run. In Table 4.1 we provide a recap of the agents, their underlying models and prior over weights where needed.

Agent	Type	Model	Prior
BLR EO	greedy	$f_a(\mathbf{x};\mathbf{w}) = \sigma(\mathbf{w}^T\mathbf{x})$	$p(\mathbf{w}) = \mathcal{N}(\mathbf{0},\mathbb{I})$
BLR TS	explore	$f_a(\mathbf{x};\mathbf{w}) = \sigma(\mathbf{w}^T\mathbf{x})$	$p(\mathbf{w}) = \mathcal{N}(\mathbf{0},\mathbb{I})$
BNN	explore	$f_a(\mathbf{x};\{\mathbf{W}^i\}) = \sigma(\mathbf{W}^2\text{ReLU}(\mathbf{W}^1\mathbf{x}))$	$p(\mathbf{W}) = \mathcal{N}(\mathbf{0},\mathbb{I})$
VNN	greedy	$f_a(\mathbf{x};\{\mathbf{W}^i\}) = \sigma(\mathbf{W}^2\text{ReLU}(\mathbf{W}^1\mathbf{x}))$	
Dropout	explore	$f_a(\mathbf{x};\{\mathbf{W}^i\}) = \sigma(\mathbf{W}^2\text{ReLU}(\mathbf{W}^1\mathbf{x}))$	
DGT	gr. truth	$f_{gt}(\mathbf{x};\{\mathbf{W}^i_{gt}\}) = \sigma(\mathbf{W}^2_{gt}\text{ReLU}(\mathbf{W}^1_{gt}\mathbf{x}))$	
LGT	gr. truth	$f_{gt}(\mathbf{x};\mathbf{w}_{gt}) = \sigma(\mathbf{w}^T_{gt}\mathbf{x})$	

Table 4.1: TS: Thompson Sampling; EO: Exploit Only; BLR: Bayesian Logistic Regression; BNN: Bayesian Neural Network; VNN: Vanilla Neural Network (not Bayesian); LGT: Linear Ground Truth; DGT: Deep Ground Truth.
Agents recap table: For each agent, it is provided the underlying model we use for our analysis and the prior over weights, where used. Although Dropout is an exploring agent, it has no prior, but the exploration is realized via Monte Carlo Dropout (Sec .3.5.2). The posterior is approximated via Laplace approximation for BLR and stochastic variational inference for BNN.

To calculate an offline estimator of the total regret, we set up a simulated environment. From here on, we will refer to the feature vector representing the interaction between contexts and eligible ads to x. $\mathbf{X}$ is a matrix containing all the displayed ads $\{\mathbf{x}\}_{i=1}^{N}$ as its rows, ordered temporally. In the simulation, we consider part of a daily Criteo traffic, with $N = 1.000.000$ ads. At each time-step t the agent receive a subset of n_{pool} ads $\mathbf{X}^{(t)}_{pool} \sim \mathbf{X}$ and decide which subset $\mathbf{X}^{(t)}_{sel}$ of n_{sel} ads to display, based on its underlying model. A Ground Truth model $f_{gt} = p(y=1|\mathbf{x})$ generates the revenue for the agent selection r_t, the maximum achievable revenue r^*_t and clicks $y^{(t)}$. The model is finally updated with $\left\langle \mathbf{X}^{(t)}_{sel}, y^{(t)} \right\rangle$. The simulation algorithm is detailed in the next paragraph.

4.2.1 Simulation

The proposed simulation algorithm consists in the following steps:

1. Train the Ground Truth model on the full dataset.
2. Initialize agent's model parameters at random and training data $\mathcal{D}_{train} = \emptyset$. If the model is Bayesian initialize $\mathrm{p}(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \mathbf{0}, \mathbb{I})$.
3. Start simulation, for each iteration:
   - Get $\mathbf{X}^{(t)}_{pool}$, rank the ads based on their score $f_a(\mathbf{X}^{(t)}_{pool}, \mathbf{w})$, and select the n_{sel} ads with the highest score $\mathbf{X}^{(t)}_{a,sel}$. If the model is Bayesian, use $\mathbf{w} \sim p(\mathbf{w}|\mathcal{D}^{(t)}_{train})$ to compute the score, as for the Thompson Sampling procedure in Algorithm 1, Sec. 2.6.3
   - Get the reward $r^{(t)}_a = \left\| f_{gt}(\mathbf{X}^{(t)}_{a,sel}, \mathbf{w}_{gt}) \right\|_1$
   - the maximum achievable reward is $r^{(t)}_{gt} = \left\| f_{gt}(\mathbf{X}^{(t)}_{gt,sel}, \mathbf{w}_{gt}) \right\|_1$, where $\mathbf{X}^{(t)}_{gt,sel}$ is the batch we would have selected if we knew the Ground Truth.
   - Compute the regret $l^{(t)}_a = r^{(t)}_{gt} - r^{(t)}_a$
   - Generate clicks $\mathbf{y}^{(t)} \sim Bernoulli\big[f_{gt}(\mathbf{X}^{(t)}_{a,sel}, \mathbf{w}_{gt})\big]$, update training data $\mathcal{D}^{(t)}_{train} = \mathcal{D}^{(t-1)}_{train} \cup \big\langle \mathbf{X}^{(t)}_{a,sel}, \mathbf{y}^{(t)} \big\rangle$ and retrain the agent's model f_a.

4.2.2 Simulation parameters settings

Results heavily depends on experiment parameters settings. The parameters of our proposed algorithms are:

- ω, frequency of updates
- N, number of training datapoints for the Ground Truth
- n_{pool}, size of the pool of displayed ads $\mathbf{X}^t_{pool}$ at each timestep t
- n_{sel}, size of the set of selected ads $\mathbf{X}^t_{sel}$
- Initial training size, the training size of the initial model also affect the comparison. As one would expect, exploration is more beneficial when we start from $\mathcal{D}_{train} = \emptyset$ or very limited data.

- Online batch training, in this case $\omega = 1$ and the model is updated with the selected subset of ads and the observed outcome from the environment $\langle \mathbf{X}_{sel}^{(t)}, \mathbf{y}^{(t)} \rangle$. Otherwise, at each iteration $\mathcal{D}_{train}^{(t)} = \mathcal{D}_{train}^{(t-1)} \cup \langle \mathbf{X}_{sel}^{(t)}, \mathbf{y}^{(t)} \rangle$, from here on we will refer to this as **reset training**

One of the most sensitive parameters is the frequency of updates. This has a significant impact on the number of time-steps necessary to observe the long-term benefit of Thompson Sampling exploration. We found that as the frequency of the updates increases, more iterations are needed to observe the long-term benefit. The reason is also related to the concept of *Deep Exploration* (Guez et al., (2012); Osband et al., (2016)) for which to fully benefit from Thompson Sampling exploration, the agent should commit to the sample for several time-steps. The naive application of Thompson Sampling, which re-sample at each time step, can be extremely inefficient. For this reason, we use $\omega = 10$, as it also requires less computing time and allows, in some cases, to show the benefits of exploration in a smaller number of iterations. The combination n_{sel}, n_{pool} also has a significant impact on the outcome of our experiments. We found that reasonable value to use are $n_{sel} \in \{100, 1.000\}$ and $n_{pool} \in \{1.000, 10.000\}$. In all the illustrated results we use $n_{sel} = 100$ and $n_{pool} = 1.000$.

We use *reset training* for all our experiments except for the baseline experiment, where we reproduce the experiments setting in Chapelle et al. (2015). The reason for our choice is that *reset training* allows a more fair comparison between linear and deep models. It is more common in real-world applications, where data are accumulated over time and models re-trained offline. We also tried online simulations and found that the overall outcomes are not different within models of the same class, but affect the comparison between linear vs. deep models, as deep model training in online scenarios is much slower.

4.2.3 Linear Ground Truth model

Evaluation of exploration in online advertising application is difficult because there is no way to know the reward of an action which was not chosen. Chapelle et al. (2015) suggest using a weight vector $\mathbf{w}^*$ and generate click with probability $p(y = 1|\mathbf{x}) = (1 + exp(\mathbf{w}^{*T}\mathbf{x}))^{-1}$. We use this idea to design our Logistic Regression Ground Truth. The linear Ground Truth model is trained on $N = 1.000.000$ datapoints from the Criteo dataset, with regularization constant $\lambda = 1$.

4.2.4 Deep Ground Truth model

In order to compare linear models with deep models, we also train a Neural Network Ground Truth model. The network has 1 hidden layer with 130 neurons and was trained with stochastic gradient-based optimization, using ReLU (Nair and Hinton, 2010) as activations, Adam as optimizer, with learning rate $\alpha = 10^{-3}$, $\beta_1 = 0,9$, $\beta_1 = 0,999$, and mini batches size of 64. We use dropout as a regularizer, with a dropout probability $p = 0.2$.

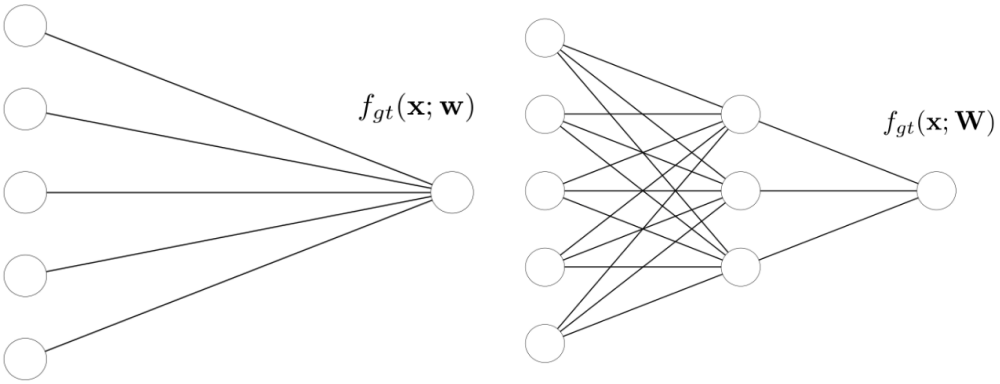

Figure 4.1: Simple representation of Linear and Deep Ground Truth models.

4.3 Bayesian Logistic Regression

The first algorithm object of our investigation is a Bayesian Linear Regression model with diagonal prior $p(\mathbf{w} = \mathcal{N}(\mathbf{w}; \mathbf{0}, \mathbb{I})$. Given n datapoint $\{\mathbf{x}_i\}_{i=1}^n$, where $\mathbf{x} \in \mathbb{R}^d$, and a set of outcomes $\{y_i\}_{i=1}^n$, where $y_i \in \{\text{clicked, non-clicked}\}$, the posterior updates in Bayesian logistic regression are exact and can be computed with Laplace approximation (see section 3.3.1), with the following update rules (Chapelle et al., 2015):

$$\mathbf{w} = argmin_w \frac{1}{2} \sum_{i=1}^{d} q_i (w_i - m_i)^2 + \sum_{j=1}^{n} \log\left(1 + exp(-y_j \mathbf{w}^T \mathbf{x_j})\right) \tag{4.1}$$

each posterior weight distribution $w_i \sim \mathcal{N}(w_i; m_i, q_i^{-1})$ can be updated conveniently as follows:

$$\begin{cases} m_i & = w_i \\ q_i & = q_i + \sum_{j=1}^{n} x_{ij}^2 p_j(1-p_j), \quad p_j = (1+exp(-\mathbf{w}^T \mathbf{x}_j))^{-1} \end{cases} \tag{4.2}$$

We implement Bayesian Logistic Regression with Laplace approximation starting from publicly available code[2] and adapting it to our simulation needs.

4.4 Deep Bayesian Models

There has been an increasing amount of interest in Deep Bayesian models over the past few years in the machine learning community. Bayesian deep models provide a desirable probabilistic interpretation. Recently developed scalable solution for the approximate posterior computation (Blundell et al., 2015) or alternative methods such as Dropout (Gal and Ghahramani, 2016) played a significant role in the diffusion of such techniques. However, such models are still in their early stage, their properties are still not well known, and, as a consequence, not yet widely adopted for real-world applications. We test the effectiveness of some of these new techniques in online advertising systems, comparing them with usual Neural Networks, which are referred to as Vanilla Neural Networks. Vanilla Neural Networks are trained in the same way as Dropout Neural Networks, which are described below, with the only difference that they do not provide any probabilistic interpretation and mean of exploration. We use our evaluation framework to evaluate the different algorithms and the weights distribution, where available, to perform Thompson Sampling exploration.

Dropout as Bayesian approximation. The first Bayesian Deep model we experiment with is a Dropout Neural Network. In Sec. 3.5.2 we showed that it is possible to approximate a Deep Gaussian Process simply by activating dropout in the test phase. Due to its simplicity, the method has been rapidly adopted in many applications (Kendall and Gal, (2017); Gal et al., (2017b)). Thompson Sampling can be obtained simply as Stochastic forward pass over the Dropout Neural Network. The architecture used is similar to the one used for the deep Ground Truth, with 1 hidden layer of 100 neurons instead of 130. We implemented Dropout in PyTorch (Paszke et al., 2017), and trained

[2] https://github.com/Valassis-Digital-Media/bayes_logistic

it via stochastic gradient descent, using Adam as optimizer (Kingma and Ba, 2014), with learning rate $\alpha = 10^{-3}$, $\beta_1 = 0,9$, $\beta_1 = 0,999$ and mini batches of size 64. The dropout rate is set to $p = 0.2$.

Bayesan Neural Network. Finally, we train a Bayesian Neural Network agent using the *bayes by backprop* algorithm. As explained in more detail in section 3.5.1, the algorithm allows to compute the approximate variational posterior by minimizing the ELBO lower bound with gradient-based optimization. We reproduce and train the *bayes by backprop* algorithm in PyTorch. As for the Dropout Neural Network, we use 1 hidden layer with 100 neurons. It should be noted that *bayes by backprop* requires twice the number of parameters of the Dropout Neural Network, as for each weight we need to fit mean and variance parameters. Additional parameters to set are the prior distribution, its mean and standard deviation. As suggested in the original paper, we use as a prior the scale mixture of two Gaussian densities:

$$p(\mathbf{w}) = \prod_j \pi \mathcal{N}(\mathbf{w}_j; 0, \sigma_1^2) + (1-\pi)\mathcal{N}(\mathbf{w}_j; 0, \sigma_2^2) \tag{4.3}$$

we set $\pi = 0,75$, $-\log\sigma_1 = 0$ and $-\log\sigma_2 = 6$. The model is trained with Stochastic Gradient Descent (SGD) with 10^{-4} as learning rate and minibatches of size 64.

Before running the two above algorithms on the Criteo dataset, we checked our implementation in two ways: we evaluated the models on the MNIST dataset (LeCun, 1998); we implemented the models with no layers and compared their performances to the Bayesian Linear Regression model.

Chapter 5

Results and Discussion

In this chapter we present the results of our experiments. The chapter is organized as follows: In Section 5.1, we present our baseline experiments, which are used as a proof of work of our methodology; Section 5.2 presents the results for the algorithms in Table 4.1 with a Logistic Regression Ground Truth model; Section 5.3 uses a Neural Network Ground Truth; In Section 5.4, we compare the performance of linear and deep algorithms with both the linear and deep Ground Truth; Finally, in Section 5.5, we provide a discussion of the results.

5.1 Baseline experiment

In our baseline experiments, we compare *greedy* and *exploring* strategies with a Bayesian Online Logistic Regression model, partially reproducing the setting in Chapelle et al. (2015). We use our proposed algorithm (Sec. 4.2) to evaluate the effectiveness of Thompson Sampling exploration in terms of total regret.

Our experiments show that Thompson Sampling outperform the *greedy* policy in the long run (Fig. 5.1). However, unlike the results presented in Chapelle et al. (2015), where Thompson Sampling always has a better regret than the Exploit strategy, in our case, the regret of Thompson Sampling is higher at the beginning of the simulation, as it sacrifices short-term rewards for a deeper understanding of the underlying Ground Truth, which eventually brings long-term benefits.

There are a few differences between our implementation, as described in chapter 4, and the simulation setting in Chapelle et al. (2015). The author, for his simulations,

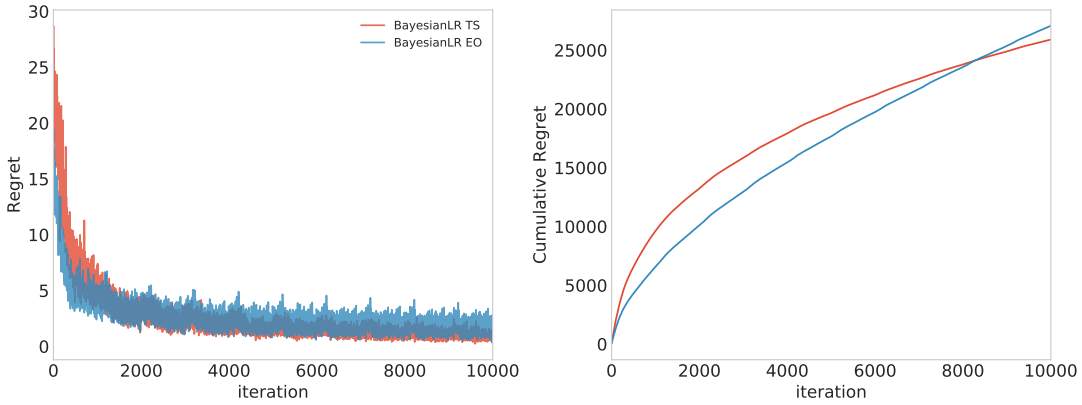

Figure 5.1: Regret and cumulative regret comparison between greedy (exploit) and exploring bayesian online logistic regression algorithms. In the long run, Thompson Sampling leads to a better cumulative regret.

uses 13.000 contexts per hour/time-step. A number of eligible ads variable from 1 to 5.910 with a mean of 1.394 are selected (in our case, we use 10.000 contexts and select 1.000 of them). Then the model is updated online each time-step with the new batch of selected (context, ads) pairs and the generated outcomes. The simulation lasts for 90 hours, which are the equivalent of 90 time-steps in our simulation, the dataset used is not publicly available and no detailed info about the robustness of the results and its behavior in the long-run is provided.

5.2 Linear Ground Truth

From here on we will use *reset training* (Sec. 4.2.2) for all our experiments. *Reset training* is much more computationally expensive than online training. Therefore, we run the simulation for 2.000 iterations instead of 10.000 as in the baseline experiment.

With reset training, the benefits of exploration is not evident with Thompson Sampling in linear models (Fig. 5.2). As both models get very close to the Ground Truth very soon and the better knowledge of the Ground Truth of the exploring agent is not enough to observe a better total regret in an observable number of iterations.

For deep models, we also do not find any evidence of the advantage of performing exploration via Thompson Sampling (Fig. 5.3). Bayesian Neural Networks have twice the number of parameters to fit (mean and standard deviation for each parameter),

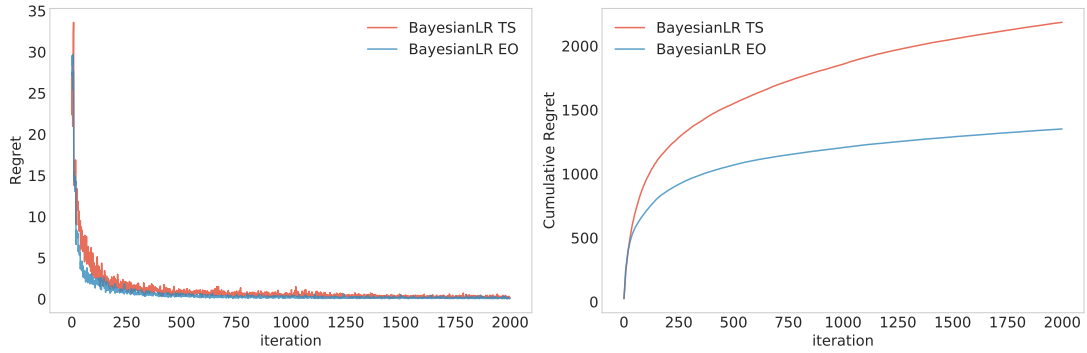

Figure 5.2: Regret and cumulative regret comparison between the two linear models with logistic regression as Ground Truth model.

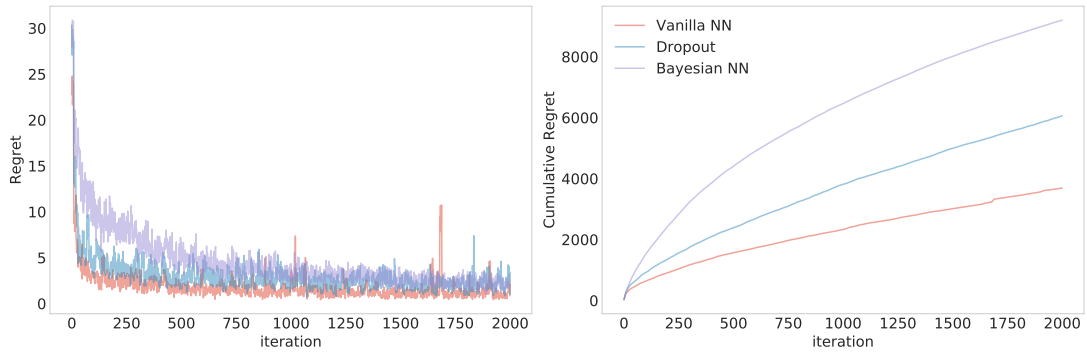

Figure 5.3: Regret and cumulative regret comparison among deep models with logistic regression as Ground Truth model. Deep Bayesian models do not outperform vanilla Neural Networks in the observed number of iterations.

and this seems to be a substantial limitation in sequential decision-making scenarios. Vanilla Neural Networks outperform the other algorithms and have a better regret beginning early in training. However, the regret per iteration, which can be used as an indication of the cumulative regret curve's slope, for Vanilla Neural Networks and Dropout Neural Network is fairly constant during the last time-steps of the simulation, while for Bayesian Neural Networks it keeps improving over time. This could be an indication of potential benefits of using this algorithm in the long run. Unfortunately, running the simulation for more than 2.000 iterations is extremely computationally expensive and might be worth exploring only depending on the application (see discussion on the number of iterations in Sec. 5.5).

5.3 Deep Ground Truth

When using deep neural Ground Truth, we observe more noise in the regret per iteration in the linear agents, as the underlying generative model is more complex. However, the final result is unchanged. As shown in Fig. 5.4, the two agents regrets are getting closer, but the Thompson Sampling total regret does not get better than the *greedy* one, in the observed iterations.

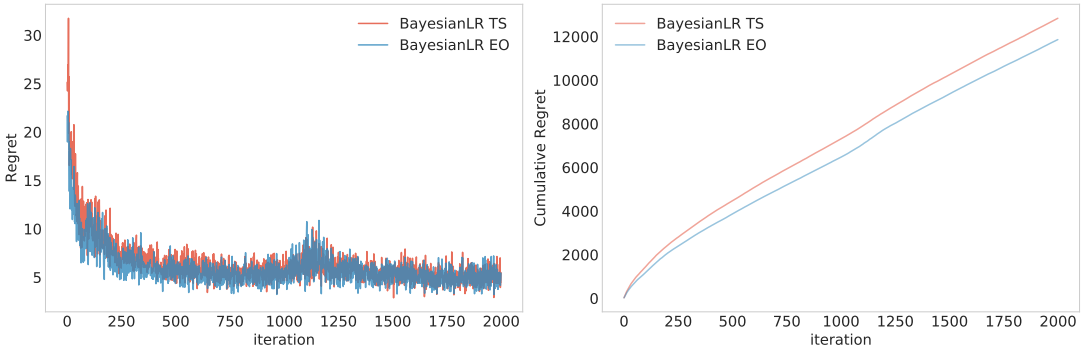

Figure 5.4: Regret per iteration and cumulative regret for the policies Exploit Only and Thompson Sampling for Bayesian Linear Regression models with Neural Network Ground Truth.

For deep models, the observed pattern is precisely the same as for the linear Ground Truth scenario, with no evidence of benefits of the exploration and Vanilla Neural Networks having the best total regret (Fig. 5.5). For a fair comparison with linear models, we also tried an online simulation on a larger number of iterations with no evidence of potential benefits of exploration also in this case. A figure of the outcome of the online simulation is provided in Appendix A.

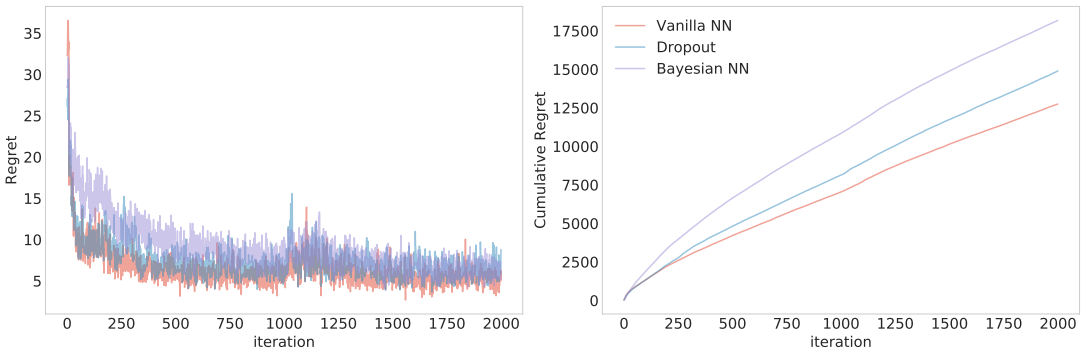

Figure 5.5: Regret per iteration and cumulative regret for the policies Exploit Only and Thompson Sampling with Neural Network Ground Truth. Also in this case, there is no evidence of the benefits of exploration with deep Bayesian models.

5.4 Model comparison

When using a linear model as Ground Truth, the linear models are highly advantaged, as they have much fewer parameters to fit, and have the same structure of the generative model. The comparison of all the algorithms with a Logistic Regression Ground Truth is shown in Fig. 5.6. Fig. 5.7 reports a comparison of the regret achieved by each algorithm at the end of the simulation. Since there is a great amount of noise in the regret, we provide an indication of such noise, reporting mean and standard deviation of the regret in the last 10 iterations. Final regret gives a good indication of the long-term performance of an algorithm, as it can be interpreted as the cumulative regret curve's slope.

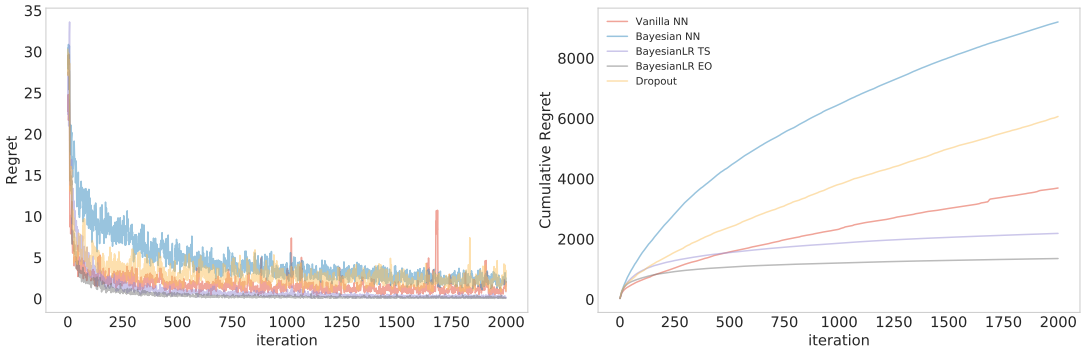

Figure 5.6: Regret per iteration for the policies Exploit Only and Thompson Sampling for all the algorithms with Logistic Regression Ground Truth.

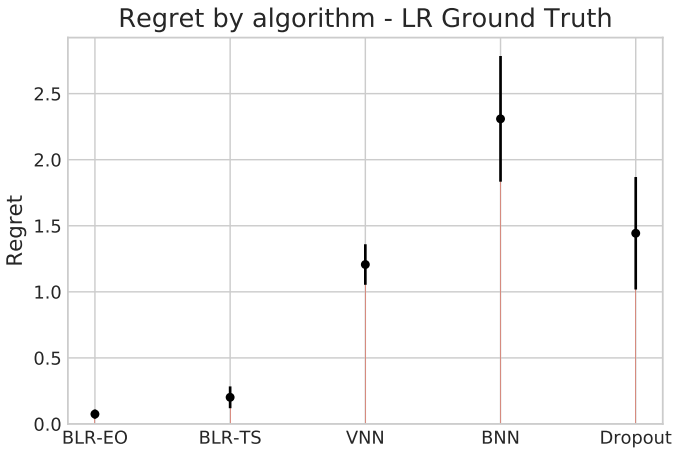

Figure 5.7: Final Regret achieved by the algorithms under a logistic regression Ground Truth.

Surprisingly, linear models perform well also with a Neural Network Ground Truth and

have performances comparable to Deep models (Figs. 5.8, 5.9). In fact, in the observed number of iterations the *exploit* Bayesian Logistic Regression model achieves the best total regret. However, as indicated by the final regret achieved, Vanilla Neural Network achieved the best result, which means it is the model which will perform the best in the long run. The results are summarized in Table 5.1 for linear Ground Truth and Table 5.2 for deep Ground Truth.

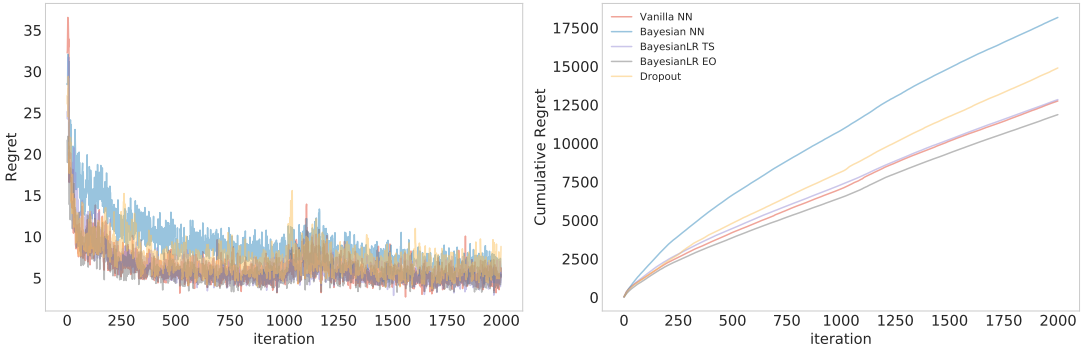

Figure 5.8: Regret per iteration for the policies Exploit Only and Thompson Sampling for all the algorithms with Neural Network Ground Truth.

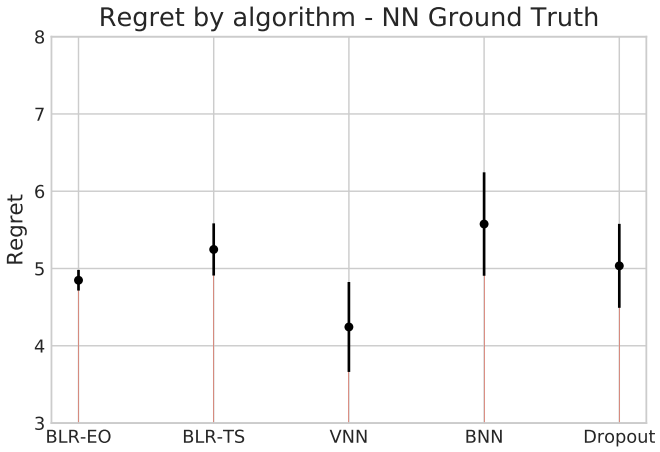

Figure 5.9: Final Regret achieved by the algorithms under a Neural Network Ground Truth.

In real-world applications, we expect the true generative model to be complex and certainly not likely to be represented merely by a linear model Ground Truth. With this assumption, the best model to adopt would be a Vanilla Neural Network, as the results, in the context of our experimental setting and with the proposed deterministic

Ground Truth model, suggest that exploration may not be needed in online advertising applications. The reasons might be numerous and in the next section, we provide an extensive discussion on the outcome of the experiments.

Agent	Final Regret	Cumulative Regret
BLR EO	$\mathbf{0,0748 \pm 0,0344}$	**1.351**
BLR TS	$0,2018 \pm 0,0825$	2.185
VNN	$1,2068 \pm 0,1534$	3.804
BNN	$2,3092 \pm 0,4756$	9.206
Dropout	$1,4433 \pm 0,4256$	6.079

Table 5.1: Models comparison with Logistic regression Ground Truth.

Agent	Final Regret	Cumulative Regret
BLR EO	$4,8481 \pm 0,2893$	**10.873**
BLR TS	$5,1408 \pm 0,0405$	11.814
VNN	$\mathbf{4,2432 \pm 0,9889}$	12.389
BNN	$5,5754 \pm 0,7029$	18.306
Dropout	$5,0345 \pm 0,8270$	14.622

Table 5.2: Models comparison with Neural Network Ground Truth. BLR EO has the best Cumulative Regret but VNN achieves the best final regret and is, therefore, the model who is going to do better in the long run.

5.5 Discussion

Overall, we found that, within the hypothesis of our framework, there is no evidence that exploration with Bayesian deep model algorithms can bring benefits in terms of cumulative regret. While we found that it may be true in the online setting when using Thompson Sampling with Bayesian Logistic Regression, even in this case, the results depend on many factors, such as update frequency, online vs. non-online training and number of iterations.

The experiments and simulation framework we used in our work are in many aspects new and unique. Therefore we want to outline several points that should be noted to better understand the outcome, the potential future use of such a framework, as well as future improvements.

Exploration in online advertising systems. Evaluation of Deep Bayesian bandits algorithms for online advertising application has not been extensively studied so far. Most of the literature on this topic comes from the Reinforcement Learning research community. However, exploration in reinforcement learning has a vital role. It is in most cases impossible for the agent to learn the optimal policy without it. This is not necessarily true in online advertising applications, where of course there could be benefits in exploring, but it is not so obvious. Sometimes the noise in the process and the dynamics in the application may be such to generate automatic exploration. Furthermore, in Reinforcement Learning, we generally have a simulated environment and different (state, action) pairs can be easily tested by evaluating the observed reward. Whereas in online advertising we cannot observe rewards for all the possible (state, action) combinations, since we do not know the outcome of ads we did not show. Therefore, evaluating different policies is not trivial. With this work we introduced a framework which could be used as a solution to such a problem, allowing to test and compare efficiently several algorithms and to establish benchmarks for the application.

Hyperparameter Tuning. Deep models are highly sensitive to hyperparameter tuning and require heavier training. In online/sequential decision-making scenarios, this is even more evident than in supervised learning applications. It is therefore not trivial to carry out a fair comparison between different algorithm and extensive tuning using Bayesian optimization (Snoek et al., 2012) or Random Search (Bergstra and Bengio, 2012) should be done before deriving conclusions.

Number of iterations. The number of iterations worth exploring depends on the application and on the specific problem to solve. We focus on providing a tool to compare the qualities of different agents in sequential decision-making problems, whether to run the analysis on more iterations, online or with reset, depends on the specific application.

Dataset. The Criteo dataset contains ads that have been displayed. Ideally, one would have access to all the potential ads/interaction and perform the comparison in a more realistic setting.

Sequential decision-making. Whether a model does well in a supervised learning scenario does not necessarily mean that it does as well also in online decision-making.

Models like the Bayesian Neural Network, which are successful in supervised learning under-perform in the task. Riquelme et al. (2018) in their recent work also found that the Bayesian Deep Model posterior estimate might not be suitable for sequential decision problems, despite their excellent results in supervised learning scenarios.

Ground Truth model The Ground Truth models proposed might be too simple and deterministic to generalize in real-world scenarios. The results obtained with such models might not hold for more complex Ground Truth models. However, the simulation setting provided is such that one can quickly try different hypotheses and test the robustness of the results under several Ground Truth model assumption.

5.5.1 Uncertainty in Deep Bayesian models

Another important factor to consider for the understanding of the results and potential future works is the uncertainty estimation provided by each algorithm, how they differ, what they are measuring and how to evaluate their quality. There has been a lot of discussion in the Bayesian Machine Learning community about which uncertainties are being modeled and which one are needed. Kendall and Gal (2017) divide the uncertainties into two main families:

1. *Aleatoric*: captures uncertainty related to the observation. This can be further divided in *homoscedastic* and *heteroscedastic* depending on whether stays constant for different inputs or not. It is possible to learn σ_x as function of the input by optimizing
$$\frac{||y-\hat{y}||_2}{2\sigma_x^2} + log(\sigma_x^2)$$
2. *Epistemic*: captures uncertainty related to the model parameters. It is therefore the uncertainty needed to correctly perform Thompson Sampling (Osband, 2016).

Aleatoric uncertainty is intrinsic in the observations, Whereas, *epistemic* uncertainty can be explained away as we acquire more data. In Neural Networks, *epistemic* uncertainty can be modeled by placing a prior over the network weights, $\mathbf{W} \sim \mathcal{N}(0, \mathbb{I})$. If we denote the random output of such network as $f^W(\mathbf{x})$, we can define the model likelihood as $p(\mathbf{y}|f^W(\mathbf{x}))$. Given a dataset $\mathbf{X} = \{\mathbf{x}_1, .., \mathbf{x}_N\}$ and $\mathbf{Y} = \{\mathbf{y}_1, .., \mathbf{y}_N\}$ we can compute the posterior $p(\mathbf{W}|\mathbf{X}, \mathbf{Y})$ with Bayesian inference. As the posterior can be computed as:

$$p(\mathbf{W}|\mathbf{X},\mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{X},\mathbf{W})p(\mathbf{W})}{p(\mathbf{Y}|\mathbf{X})}$$

In section 3.5.1 we showed that is possible to approximate the posterior distribution with the *Bayes by backprop* algorithm with Stochastic Variational Inference. Therefore, we should be able to perform Thompson Sampling effectively from the approximate posterior. However, unlike Bayesian Linear Regression, the approximate posterior computation is not exact and the quality of the approximation still not well understood.

On the other hand, although it is claimed that the Dropout Network is also minimizing the Kullback-Leibler (KL) distance between the simple parametrized approximate posterior $q^*_\theta(\mathbf{W})$ and the true posterior $p(\mathbf{W}|\mathbf{X},\mathbf{Y})$, and can therefore be representative of the epistemic uncertainty, there are some alternative interpretations of it. Osband (2016) suggests a distinction between *risk* and *uncertainty*. To explain the concept the author gives as an example a coin with a fixed $p = 0.5$ of heads. In this case, although the probability p is fixed, every single outcome holds some risk, as a consequence, a learning agent may also be uncertain about p. Unlike risk, uncertainty captures the variability of a model posterior. High-risk states are less attractive for an agent in both exploration and exploitation setups. The author also proves that the variability in the dropout predictions does not go to zero as more data are gathered. This is also evident from the online demo provided by the author.

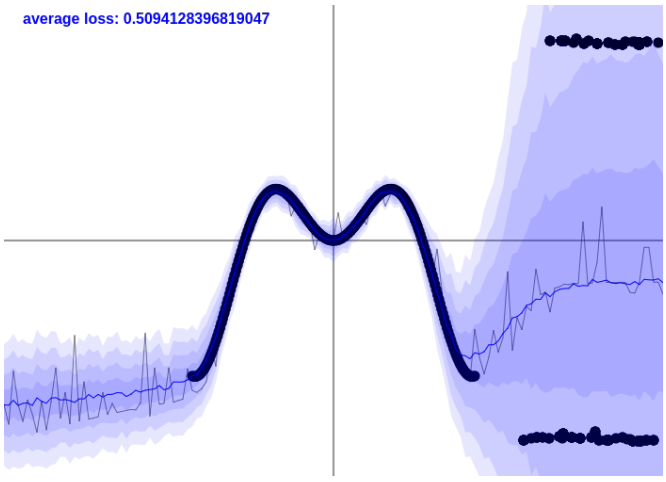

Figure 5.10: Even after observing lots of data in a specific region, the Dropout model is still very uncertain. Screen-shot from the online demo: `http://www.cs.ox.ac.uk/people/yarin.gal/website/` .

Dropout may, therefore, be modeling risk rather than *epistemic* uncertainty, and as a consequence not appropriate for Thompson Sampling. To make Dropout Neural Networks work in Reinforcement Learning settings, the author recently suggested an alternative version of it, where the dropout probability is learned during the training and goes to zero as more data are acquired (Gal et al., 2017a).

Chapter 6

Conclusion

In this chapter, we recapitulate our work, the results achieved so far and suggest potential future developments.

6.1 Our contribution

Deep Neural Network models, due to their powerful feature representations, have been widely adopted in online advertising applications. However, they lack interpretability and do not provide any uncertainty measure to be leveraged in the decision-making process. On the other hand, simple and scalable models such as Bayesian Logistic Regression, have been adopted successfully for many years and are able to provide statistical meaning. Several works on online advertising systems, such as Graepel et al. (2010), Chapelle and Li (2011), and Chapelle et al. (2015) showed that the uncertainty over the model parameters could be leveraged to perform exploration with Thompson Sampling procedure. Nevertheless, there is no work evaluating the effectiveness of sampling from a Bayesian Neural Network posterior.

We provided an overview of Bayesian machine learning techniques and showed how newly developed techniques could be integrated with the current commonly employed models in online advertising, both to add interpretability and to perform Thompson Sampling exploration. We describe in detail the hypothesis behind each of the algorithms and provide a discussion about the meaning and quality of their posterior approximation.

Evaluating the benefits of exploration in advertising systems is a hard problem as we

do not know the outcome of ads we did not show. We developed and implemented a novel evaluation framework that provided a solution for such a problem. It can be employed to test several models which can then be used as benchmarks to accelerate research and development in this area.

Finally, we used our evaluation framework to assesses the effectiveness of several models and exploration policies. There is no published work which assesses the performance of these models for online advertising applications. Overall, we found that Bayesian Deep models add no benefits in the online advertising regime in terms of total regret, suggesting that exploration for such applications may not always be needed.

6.2 Future work

There are many interesting future works and developments for the problem of exploration in online advertising systems.

6.2.1 Exploring new models

The easiest first expansion of our work would be to use our simulation framework to test newly developed models which have been proven effective for exploration in Reinforcement Learning scenarios. Below we provide a short description of two plausible models:

Concrete Dropout. Concrete Dropout (Gal et al., 2017a) is a natural improvement over the proposed Dropout Network. It is proposed as a solution for the need of well-calibrated uncertainty in deep models. It also addresses the problem of decreasing the model uncertainty to zero as new data are acquired. The dropout probability is included in the model's parameters and optimized via gradient-based optimization.

Bootstrapped Neural Networks. Bootstrapped Neural Networks (Osband et al., 2016) are an alternative approach to efficient exploration inspired by Thompson Sampling. The network consists of a shared architecture with K distinct and independent heads. Such architecture maintains some notion of uncertainty and can implement an heuristic similar to Thompson Sampling.

6.2.2 Improved simulation and new datasets

Riquelme et al. (2018), in their work, provide a benchmark where several bandits algorithm are compared against several Reinforcement Learning tasks. Benchmarks are often a research catalyst. Starting from our proposed framework, we could try to recreate a similar benchmark for online advertising applications by considering other related datasets and by fine-tuning some of the simulation's parameters discussed in Sec. 4.2.2. Additionally, we could analyze the robustness of our results under several ground-truth hypotheses. This could be achieved either by adding complexity (e.g., with more hidden layers) or adding an internal dynamics which can simulate the non-stationarities and seasonality trends of online advertising systems.

Appendix A

Supporting Material

A.1 Deep online models

In the figure below we illustrate the results of running an online simulation with deep neural models.

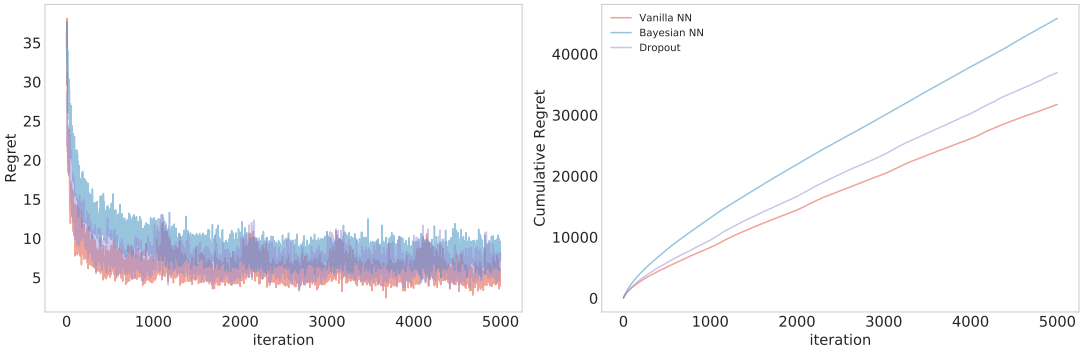

Figure A.1: Regret per iteration and cumulative regret for the policies Exploit Only and Thompson Sampling for Deep Bayesian models with Neural Network Ground Truth and online training.

Bibliography

Agrawal, S. and Goyal, N. (2012). Analysis of thompson sampling for the multi-armed bandit problem. In *Conference on Learning Theory*, pages 39–1.

Bache, K. and Lichman, M. (2013). Uci machine learning repository.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.

Branco, P., Torgo, L., and Ribeiro, R. (2015). A survey of predictive modelling under imbalanced distributions. *arXiv preprint arXiv:1505.01658*.

Chapelle, O. and Li, L. (2011). An empirical evaluation of thompson sampling. In *Advances in neural information processing systems*, pages 2249–2257.

Chapelle, O., Manavoglu, E., and Rosales, R. (2015). Simple and scalable response prediction for display advertising. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(4):61.

Cox, D. R. D. R. and Hinkley, D. V., j. a. (1974). *Theoretical statistics*. London : Chapman and Hall. Distributed in the U.S.A. by Halsted Press, New York.

Damianou, A. and Lawrence, N. (2013). Deep gaussian processes. In *Artificial Intelligence and Statistics*, pages 207–215.

Dayan, P., Hinton, G. E., Neal, R. M., and Zemel, R. S. (1995). The helmholtz machine. *Neural computation*, 7(5):889–904.

Edelman, B., Ostrovsky, M., and Schwarz, M. (2007). Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords. *American economic review*, 97(1):242–259.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059.

Gal, Y., Hron, J., and Kendall, A. (2017a). Concrete dropout. In *Advances in Neural Information Processing Systems*, pages 3581–3590.

Gal, Y., Islam, R., and Ghahramani, Z. (2017b). Deep bayesian active learning with image data. *arXiv preprint arXiv:1703.02910*.

Graepel, T., Candela, J. Q., Borchert, T., and Herbrich, R. (2010). Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. Omnipress.

Graves, A. (2011). Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356.

Guez, A. (2015). *Sample-based search methods for Bayes-adaptive planning*. PhD thesis, UCL (University College London).

Guez, A., Silver, D., and Dayan, P. (2012). Efficient bayes-adaptive reinforcement learning using sample-based search. In *Advances in Neural Information Processing Systems*, pages 1025–1033.

Hernández-Lobato, J. M. and Adams, R. (2015). Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97.

Hinton, G. E. and Van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13. ACM.

Kendall, A. and Gal, Y. (2017). What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in Neural Information Processing Systems*, pages 5580–5590.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Lai, T. L. and Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22.

LeCun, Y. (1998). The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

Li, L., Chu, W., Langford, J., and Wang, X. (2011). Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 297–306. ACM.

Li, W., Wang, X., Zhang, R., Cui, Y., Mao, J., and Jin, R. (2010). Exploitation and exploration in a performance based contextual advertising system. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 27–36. ACM.

MacKay, D. J. (1992). A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472.

Meek, C., Chickering, D. M., and Wilson, D. B. (2005). Stochastic and contingent-payment auctions. In *1st Workshop on Sponsored Search Auctions*.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

Neal, R. M. (1995). *Bayesian Learning for Neural Networks*. PhD thesis, Toronto, Ont., Canada, Canada. AAINN02676.

Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V. V. (2007). *Algorithmic game theory*. Cambridge university press.

Osband, I. (2016). Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout. In *Proceedings of the NIPS* 2016 Workshop on Bayesian Deep Learning*.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. (2016). Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pages 4026–4034.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

Quiñonero Candela, J., Rasmussen, C. E., Figueiras-Vidal, A. R., et al. (2010). Sparse spectrum gaussian process regression. *Journal of Machine Learning Research*, 11(Jun):1865–1881.

Rasmussen, C. E. (2004). Gaussian processes in machine learning. In *Advanced lectures on machine learning*. Springer.

Riquelme, C., Tucker, G., and Snoek, J. (2018). Deep bayesian bandits showdown: An empirical comparison of bayesian deep networks for thompson sampling. *arXiv preprint arXiv:1802.09127*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533.

Russo, D., Van Roy, B., Kazerouni, A., Osband, I., and Wen, Z. (2017). A tutorial on thompson sampling. *arXiv preprint arXiv:1707.02038*.

Seeger, M., Williams, C., and Lawrence, N. (2003). Fast forward selection to speed up sparse gaussian process regression. In *Artificial Intelligence and Statistics 9*, number EPFL-CONF-161318.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

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

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294.

Varian, H. R. (2007). Position auctions. *international Journal of industrial Organization*, 25(6):1163–1178.

Zhou, G., Song, C., Zhu, X., Ma, X., Yan, Y., Dai, X., Zhu, H., Jin, J., Li, H., and Gai, K. (2017). Deep interest network for click-through rate prediction. *arXiv preprint arXiv:1706.06978*.