Master Dissertation

Bayesian Neural Bandit Using Online SWAG

June 2022

SEOUL NATIONAL UNIVERSITY

Graduate School of Data Science

WOO-SEOK JANG

Bayesian Neural Bandit Using Online SWAG

Advisor Min-hwan Oh

Submitting a master's thesis of
Data Science

June 2022

Graduate School of Data Science
Seoul National University
Department of Data Science (Data Science Major)

Woo-seok Jang

Confirming the master's thesis written by
Woo-seok Jang
July 2022

Chair	이 재 진	(Seal)
Vice Chair	오 민 환	(Seal)
Examiner	박 현 우	(Seal)
Examiner	이 상 학	(Seal)

Abstract

In this paper, we propose a Neural SWAG Bandit algorithm that combines a neural network-based bandit algorithm with Stochastic Weight Averaging Gaussian (SWAG), a Bayesian deep learning methodology. Neural Bandit is a bandit algorithm that uses the output of neural networks as an estimated reward. SWAG is a Bayesian Deep Learning method that samples parameters from the gaussian posterior distribution, which has been shown to have state-of-the-art performance and robustness compared to benchmark algorithms. By adapting SWAG into an online setting and combining it with Neural Bandit, we can leverage efficient sampling from deep neural networks while learning online. Our experiment results indicate that Neural SWAG Bandit benefits from Bayesian deep learning as well as exhibits superior performance compared to existing benchmark algorithms.

Keywords: Contextual Bandit, Bayesian Deep Learning, Neural Bandit, Stochastic Weight Averaging Gaussian(SWAG)

Student ID: 2020-21057

Contents

List of Figures

List of Tables

1 INTRODUCTION

1.1 Bandit Algorithm

This paper deals with the bandit algorithm that can be used for content recommendation. Bandit is a term that refers to a slot machine, and the bandit algorithm assumes a situation in which a single reward such as a prize is received when a single slot machine is drawn. For example, suppose you play slot machine games in a casino. Then, when there are multiple bandit machines (= slot machines), you can win the most significant money (= rewards) if you can find the best bandit machine that gives the most rewards on average.

This situation can be represented by the Multi-armed Bandit setting, one of the situations assumed by the bandit algorithm. In a multi-armed bandit situation, the bandit algorithm selects one bandit machine from among several bandit machines and pulls the arm in a single trial. Then the algorithm performs each trial several times to get every single reward. And it increasingly learns the true parameter that creates the true reward. Finally, it can find the optimal bandit machine.

Let's assume that each bandit machine is web-based content such as news, movies, or merchandise. Also, suppose that the reward you get when you pull each bandit machine is satisfaction with the content. At this time, satisfaction can be assumed in various ways. For example, it can be expressed by whether

the recommended content is clicked (= 1) or not (= 0). Then, through the bandit algorithm, it is possible to find the content (= bandit machine) that can elicit the highest average satisfaction, the most clicks (= reward), and recommend it to users.

1.2 Neural Bandit

In a simple research situation, it is assumed that the reward output from each bandit machine is calculated from a linear function. This means that the true reward output created by each bandit machine is calculated using a vector x_a containing the characteristics of each bandit machine and a certain true parameter θ^* that creates a true reward. It is equivalent to assuming that the true reward model is a linear function such as $r_a^* = x_a^\top \theta^* + b$. Since it is assumed that the true reward function is linear, we can solve the problem assuming that the estimated reward function trained based on the reward output by each bandit machine is also linear. This linear function is a good starting point for research because it can model the bandit situation most simply.

However, in an actual situation other than a research environment, it is much more likely that the true reward function is not constructed as simply as a linear function. Above all, there is a problem in that it is difficult to determine the format of the estimated reward function because the true reward function is unknown. For example, suppose the estimated reward function is set as a Neural Net model. In that case, it has the advantage that the true reward function can be inferred from a linear function and other functions such as a quadratic function and a logistic function. Zhou et al.(2020) proposed neural contextual bandits with UCB-based exploration as this concept and showed the

algorithm is empirically competitive. Therefore, in this paper, the estimated function of the bandit was adopted as a Neural Net model to implement a more realistic bandit algorithm.

1.3 Bayesian Deep Learning

It is important to understand what deep learning does not know and how much it does not know. An example related to the idea that 'there may be things that even deep learning does not know' is the case of Tesla's autonomous driving accident. This is an example in which an autonomous vehicle erroneously judged the side of a huge trailer to be the sky and collided with the trailer. In this accident, the deep learning model had no idea what a trailer was. If the deep learning model had known that the trailer was not well known, the decision to boldly crash the vehicle into the trailer would not have been made.

This problem occurs because of the over-confidence in deep learning-based decision-making. There are various reasons why this over-confidence problem occurs. One of the causes is that the training data itself is made to over-confidence. For example, if the result of the deep learning model for some input data was a dog, it means a dog with a 100% probability or a cat with a 0% possibility. However, if the deep learning model for some input data predicts that this data is a dog with a 70% chance and a cat with a 30% probability, a more sophisticated decision can be made. Such sophisticated decision-making can prevent accidents and support better decision-making in systems that are highly sensitive to uncertainty, such as autonomous driving, finance, and healthcare.

1.4 SWAG(Stochastic Weight Averaging Gaussian)

SWAG is a Bayesian Deep Learning method that samples deep learning model parameters θ from a Gaussian distribution $\mathcal{N}(\theta_{SWA}, \frac{1}{2} \cdot (\Sigma_{diag} + \Sigma_{low-rank}))$. SWAG, as the name suggests, is a method of creating a Gaussian distribution that can sample model parameters by utilizing the stochastic weight averaging technique. SWAG approximates the actual posterior distribution well. In addition, compared to other benchmark methods such as MC dropout, it has shown excellent performance in various tasks, including sample detection, calibration, and temperature scaling.

1.5 Contributions

Our main contributions are as follows. By adapting SWAG into an online setting and combining it with Neural Bandit, we can leverage efficient sampling from deep neural networks while learning online. Since our algorithm samples parameters of neural networks from the same Gaussian distribution, it is possible to estimate parameters more efficiently than general methods. Our experiment results indicate that Neural SWAG Bandit benefits from Bayesian deep learning and exhibits superior performance compared to existing benchmark algorithms.

2 BACKGROUND & RELATED WORK

2.1 Bandit Algorithm

2.1.1 Multi-armed Bandit

Multi-armed bandit algorithm A proceeds for trial $t = 1, 2, 3, ...$ At this time, in individual trial t, the situation proceeds as follows. First, the algorithm observes the feature vector $x_{t,a}$ containing information about each user or arm of the set $\mathcal{A}_t$ of the arms. Next, Algorithm selects one $a_t \in \mathcal{A}_t$ based on the payoffs observed in previous trials and receives the payoff r_{t,a_t} as a result. At this time, unchosen arms $a \neq a_t$ do not receive feedback. Finally, the algorithm improves the arm-selection strategy based on the new observations $(x_{t,a_t}, a_t, r_{t,a_t})$.

In the above processes, the total T-trial payoff of A is defined as $\sum_{t=1}^{T} r_{t,a_t}$. Similarly, the optimal expected T-trial payoff is defined as $\mathbb{E}[\sum_{t=1}^{T} r_{t,a_t^*}]$. In designing the multi-armed bandit algorithm, we aim to find an optimal arm-selection strategy that minimizes regret. At this time, T-trial regret $R_A(T)$ for Algorithm A is defined as follows. First, each trial's optimal true reward value is calculated based on the true reward parameter θ^*. Next, based on the same true reward parameter θ^*, the estimated reward value for the arm selected by the algorithm in the trial is calculated. In this way, regret can be calculated

by finding the expected values of the true reward and the estimated reward for trial $1 \sim T$ and the difference between them.

$$R_A(T) \stackrel{def}{=} E\left[\sum_{t=1}^{T} r_{t,a_t^*}\right] - E\left[\sum_{t=1}^{T} r_{t,a_t}\right]$$

2.1.2 Contextual Bandit

The algorithm observes the set $\mathcal{A}_t$ of the current user u_t and arm a_t. We then combine them to observe the feature vector $x_{t,a}$ for $a \in \mathcal{A}_t$. At this time, $x_{t,a}$ summarizes the information of both the user feature u_t and the arm feature a, called context. By using the context $x_{t,a}$ that compactly contains relevant information about users and articles, the Bandit algorithm can make more accurate recommendations to individual users and transfer relevant information from one article or other users to another. It can be generalized by extending it to articles or users.

2.1.3 Exploration & Exploitation tradeoff

The most fundamental difficulty in the bandit problem in balancing exploration and exploitation. It is called exploitation to show only the articles that a particular user will like the most. On the other hand, it is called exploration to show other options that users do not like the most but are expected to enjoy. Although exploration can increase short-term regret, Algorithm A can be further refined by collecting reward information for other arms. This way, we can better solve the bandit problem by mixing exploitation and exploration properly.

2.1.4 Existing Bandit Algorithms

ϵ-greedy

ϵ-greedy is a method of making a random choice with a small probability of ϵ. Exploitation is performed with a chance of $1-\epsilon$, and exploration is completed with a possibility of ϵ. Since the arm is randomly chosen with a probability of ϵ, it is possible to prevent a situation in which only a specific arm is continuously recommended. Therefore, if ϵ is large, more exploitation is performed, and if ϵ is small, more exploration is performed.

UCB

UCB is an algorithm that searches using the upper bound of the confidence interval. That is, an algorithm selects the arm with the highest upper confidence bound among each arm. Usually, in the initial trial, arms with high uncertainty have high upper bounds. Therefore, if a high upper bound is used as a standard, it is possible to explore arms with high uncertainty, which can affect exploration. In UCB, $a_t = \text{argmax}_i(\text{UCB}) = \text{argmax}_i(\mu_i + P_i)$ is selected from t every hour. In this expression, various UCB algorithms such as UCB1 and UCB2 are defined according to how the P_i term is defined.

Thompson Sampling

Thompson Sampling is a Bayesian method that samples θ while continuously updating the posterior distribution for the model parameter θ. Because θ is continuously sampled randomly from the posterior distribution, it can have an effect of exploration. Outputs the output values for each arm using $\theta_t \sim P(\theta|D)$ sampled from each trial t. Among them, arm $a_t = \text{argmax}_a \mathbb{E}_r(r|x_t, a, \theta^t)$ which

outputs the largest expected value is selected and the reward r_t is received. After that, the dataset is updated to $D = D \cup (x_t, a_t, r_t)$.

2.2 Neural Bandit

2.2.1 Neural Bandit

A neural Bandit refers to Bandit that uses Neural Net as a reward prediction model. In Neural Bandit, θ is updated for every trial using the Neural Net model $f(x;\theta)$ that can output the input x as the reward r. The method in which the parameter θ is updated in Neural Bandit is basically the same as the general bandit algorithm, except that the reward is calculated with a neural net. Neural Bandit has the advantage that the true reward function can be used even if it is not in a fixed form, such as a linear function or a logistic function.

2.2.2 Existing Neural Bandit Algorithms

NeuralUCB

It is an algorithm that applies Neural Net to the UCB algorithm. In NeuralUCB, we use a neural network to learn the unknown reward function and then follow the UCB strategy for exploration.

NeuralTS

It is an algorithm that applies Neural Net to the Thompson Sampling algorithm. In NeuralTS, we also use a neural network to learn the unknown reward function and follow the TS exploration strategy. In the paper that

proposed NeuralTS (Zhang et al., 2020), a normal distribution is used to construct a posterior distribution for Thompson sampling. At this time, variance is built using the neural tangent kernel using the neural tangent features of the corresponding neural network.

2.3 Bayesian Deep Learning

2.3.1 Bayesian and Uncertainty

Bayesian deep learning is a method that grafts the Bayesian perspective to deep learning and uses the Bayes rule to quantify the deep learning model output as a probability distribution with uncertainty. In other words, quantification of uncertainty in Bayesian deep learning is implemented through variance, and the key to Bayesian is that the probability distribution containing uncertainty is updated. In the context of finding parameters of a deep learning model, frequentism aims to find a fixed set of optimal parameter values. On the other hand, in Bayesian, the probability distribution that produces each parameter value is found, and several parameter numerical sets derived from it are considered.

2.3.2 Bayes Rule

The probability distribution that Bayesian deep learning is looking for can be found using the Bayes rule, as shown below.

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}, \quad p(\mathcal{D}) = \int p(\mathcal{D}|\theta)p(\theta)d\theta, \quad p(\theta|X,y) = \frac{p(y|X,\theta)p(\theta)}{p(y|X)}$$

$p(\theta)$ is a prior distribution and corresponds to a known distribution before observing new data $\mathcal{D}$. $p(\mathcal{D}|\theta)$ is the likelihood, and a quantified value indicates

how well the current probabilistic model explains the data. $p(\mathcal{D})$ is evidence and means the probability of the newly given information $\mathcal{D}$. $p(\theta|\mathcal{D})$ corresponds to the posterior distribution and is the subject to be updated from the prior distribution using the likelihood value. This way, it is possible to update the posterior distribution of the deep learning prediction values.

2.3.3 Bayesian Neural Network

Existing neural networks have fixed weight parameters after training. On the other hand, Bayesian neural networks are different in that each weight parameter is also expressed as a probability distribution with uncertainty. Since each weight parameter is a probability distribution with uncertainty, the final output y also has uncertainty. As a result, uncertainty can be considered in the final decision. From the perspective of Bayesian deep learning, it deals with how to update the prior distribution to the posterior distribution for all W and b, which are all θ values. In conclusion, Bayesian neural networks can be inferred by calculating the posterior distribution of parameters.

2.3.4 Existing Bayesian Deep Learning methods

However, as mentioned above, there is a problem that the posterior distribution cannot be accurately calculated because it is difficult to estimate the integral of the evidence. There are various methods to solve this problem, and there are two typical methods, one using variational inference and one using Monte Carlo Dropout.

Variational Inference

Variational inference means approximating the posterior distribution $p(\theta|\mathcal{D})$ in the same way as $p(\theta|\mathcal{D}) \approx q(\theta)$. In general, $q(\theta)$ uses the normal distribution as the probability distribution. Variational inference is not a perfect approximation, but it can approximate the distribution to capture the characteristics of the original posterior distribution well.

MC (Monte Carlo) Dropout

MC Dropout is a Bayesian technique that uses the dropout technique to create a variance. Dropout is a technique to give the regularization effect of deep learning by probabilistically disconnecting neurons temporarily when feedforward in a neural network. Dropout is often used as a regularization method to solve the problem of overfitting because a value is focused on a specific weight during training, and dropout is intentionally excluded from producing a consistent output during testing. However, suppose dropout is used in the test. In that case, various output values can be created, and it was induced to be approximated to the posterior distribution by finding the mean and variance using these output values.

2.4 SWAG Algorithm

2.4.1 SGD (Stochastic Gradient Descent)

The Neural Network can basically find the weight θ of the model using the SGD rule. SGD uses the gradient as follows to minimize the value of the loss function that defines the difference between the output value of the deep learning model and the actual value.

2.4.2 SWA (Stochastic Weight Averaging)

In SWA, a new parameter θ_{SWA} is obtained by averaging all parameters from SGD during a specific epoch. That is, the average θ_{SWA} of the parameter θ values obtained in every epoch up to the time point T is updated to the parameter at the time point $T+1$ after that.

2.4.3 SWAG-Diagonal: $\tilde{\theta} \sim N(\theta_{SWA}, \Sigma_{Diag})$

SWAG-Diagonal constructs a normal distribution using θ_{SWA} and Σ_{Diag} and samples the parameter $\tilde{\theta}$. The diagonal covariance Σ_{Diag} can be obtained from the diagonal components of a matrix which is created by subtracting θ_{SWA}^2 from the $\bar{\theta^2}$.

$$\theta_{SWA} = \frac{1}{T}\sum_{i=1}^{T}\theta_i, \quad \bar{\theta^2} = \frac{1}{T}\sum_{i=1}^{T}\theta_i^2, \quad \Sigma_{diag} = \mathrm{diag}(\bar{\theta^2} - \theta_{SWA}^2)$$

The posterior distribution $N(\theta_{SWA}, \Sigma_{Diag})$ can be constructed using the Σ_{diag} obtained in this way and the θ_{SWA} obtained using the SWA. And this distribution can be used to sample the model parameter θ.

2.4.4 SWAG: $\tilde{\theta} \sim \mathcal{N}(\theta_{SWA}, \frac{1}{2} \cdot (\Sigma_{diag} + \Sigma_{low-rank}))$

SWAG uses not only Σ_{diag} but also $\Sigma_{low-rank}$ to generate the normal distribution and sample θ from it. Estimating Σ_{diag} is standard in Bayesian deep learning, but it can be too restrictive. By adding $\Sigma_{low-rank}$, we can approximate covariance more flexibly.

$\Sigma_{low-rank}$ can be calculated by constructing a deviation matrix $\hat{D}$ using only $i = T-K+1, ..., T$ trials corresponding to the last part of the total

T trials. The deviation matrix $\hat{D}$ can be obtained from D, which is a total deviation matrix consisted of $D_i = (\theta_i - \bar{\theta_i})$. Since we cannot access the value of θ_{SWA} during training, we use $\bar{\theta_i}$ that estimates θ_{SWA}. $\bar{\theta_i}$ is an estimate of the average of parameters during $t = 1, ...i$.

$$\Sigma = \frac{\Sigma_{i=1}^{T}(\theta_i - \theta_{SWA})(\theta_i - \theta_{SWA})^\top}{T-1} \approx \frac{\Sigma_{i=1}^{T}(\theta_i - \bar{\theta_i})(\theta_i - \bar{\theta_i})^\top}{T-1} = \frac{DD^\top}{T-1}$$

$$\Sigma_{low-rank} = \frac{1}{K-1}\hat{D}\hat{D}^\top$$

Finally, we can construct the normal distribution $\mathcal{N}(\theta_{SWA}, \frac{1}{2}(\Sigma_{diag} + \Sigma_{low-rank}))$. It is possible to sample θ from the distribution. In this way, the parameter $\tilde{\theta}$ sampled by SWAG can be calculated as follows. d is the number of parameters in the network.

$$\tilde{\theta} = \theta_{\text{SWA}} + \frac{1}{\sqrt{2}}\Sigma_{\text{diag}}^{\frac{1}{2}} z_1 + \frac{1}{\sqrt{2(K-1)}}\hat{D}z_2, \text{ where } z_1 \sim \mathcal{N}(0, I_d), z_2 \sim \mathcal{N}(0, I_K)$$

3 THE NeuralSWAG ALGORITHM

The key idea of NeuralSWAG is to use a neural network $f(x;\theta)$ with the Bayesian SWAG method to predict the reward of content x. As the trial goes, NeuralSWAG can train the Gaussian distribution of SWAG. From the distribution, the algorithm samples parameters of each neural network layer to calculate the output value, which can reduce total cumulative regrets. NeuralSWAG conducts the recommendation process in two steps: First, it chooses the arm with the highest estimated reward value. Second it updated its SWAG parameters through calculating θ_{SWA} and Σ_{diag}. With these SWAG parameters, it is possible to sample estimated parameters of neural networks and reduce overall cumulative regrets.

Algorithm 1 Neural SWAG Bandit

Input: Number of total rounds T; Number of random sample rounds τ; step size η; base model $f_{\text{base}}(x)$; SWAG model $f_{\text{SWAG}}(x)$

Step1. Choose Arm

for $t = 1, 2, 3, ..., T$ **do**

  Observe features of all arms $a \in \mathcal{A}_t : x_{t,a} \in \mathbb{R}^d$ **foreach** $a \in \mathcal{A}_t$ **do**

    if $t < \tau$ **then**

      randomly choose $a_t \in [K]$ for $t \in [\tau]$

    end

    else

      if *swag_started* **then**

        Compute $\hat{y}_{t,a} = f_{\text{SWAG}}(x_{t,a}; \theta_{t-1})$

        Let $a_t = \text{argmax}_{a \in [K]} \hat{y}_{t,a}$

      end

      else

        Compute $\hat{y}_{t,a} = f_{\text{base}}(x_{t,a}; \theta_{t-1})$

        Let $a_t = \text{argmax}_{a \in [K]} \hat{y}_{t,a}$

      end

    end

  end

end

Step2. Update

Play a_t and observe reward r_{t,a_t}

Train $f_{\text{base}}(x)$

if *swag_started* **then**

  Train $f_{\text{SWAG}}(x)$

  $$\bar{\theta} \leftarrow \theta_o,\ \bar{\theta^2} \leftarrow \theta_o^2$$

  for $i = 1, 2, 3, ..., T$ **do**

    $$\theta_i \leftarrow \theta_{i-1} - \eta \nabla_\theta \mathcal{L}(\theta_{i-1})$$

    if $MOD(i, c) = 0$ **then**

      $$n \leftarrow i/c,\ \bar{\theta} \leftarrow \frac{n\bar{\theta} + \theta_i}{n+1},\ \bar{\theta^2} \leftarrow \frac{n\bar{\theta^2} + \theta_i^2}{n+1}$$

      if *NUM_COLS*$(\hat{D}) = K$ **then**

        REMOVE_COL$(\hat{D}[:, 1])$

      end

    end

    return

    $$\theta_{\text{SWA}} = \bar{\theta},\ \Sigma_{\text{diag}} = \bar{\theta^2} - \bar{\theta}^2, \hat{D}$$

  end

end

4 EVALUATION METHODOLOGY

4.1 Cumulative Regret

In the bandit problem, the cumulative regret, which accumulates the regret calculated in each trial, is used as the evaluation methodology. By checking whether the cumulative regret gradually decreases sub-linearly, it can be judged whether the bandit model is well trained, and the performance can be checked by comparing it with benchmark models. Since the regret is calculated from oracle rewards, if we don't know all oracle rewards of each context vector, it is impossible to calculate regret and cumulative regret. For real-world datasets, sometimes there is no information about oracle rewards. Therefore we modified real-world datasets of K-classification tasks to set true oracle rewards for all context vectors.

5 EXPERIMENTS

5.1 Dataset

5.1.1 Simulation Dataset

We consider the stochastic K-armed contextual bandit problem with the total number of trials T. The agent observes the context vector consisting of K feature vectors at round $t \in [T]$. It can be notated like as follows: $\{x_{t,a} \in \mathbb{R}^d | a \in [K]\}$. Then the agent selects an action a_t which has the highest estimated reward and receives the reward $r_{t,a}$ from trial t. $\{x^i\}_{i=1}^{TK}$ is same notation with the collection of $\{x_{1,1}, x_{1,2}, ..., x_{T,K}\}$. With those vectors, we make assumption about reward generation: for any round t,

$$r_{t,a_t} = h(x_{t,a_t}) + \eta_t$$

where h is an unknown function including linear, logistic, quadratic and trigonometric. η_t is -sub-Gaussian noise which satisfies $\mathbb{E}(\eta_t) = 0$.

Specifically, a set of 20 article vectors with six features extracted from the normal distribution were generated for each trial. Next, one true parameter shared by arms was created. The true parameter was kept fixed during the experiment. After conducting the dot product and any other calculation of the article feature and the true parameter, the true reward functions were obtained. For logistic rewards functions, the true temporary reward was obtained by

Table 5.1 Real-world dataset statistics

Dataset	MNIST	CIFAR10	CIFAR100
Feature Dimension	784	3072	3072
Number of Classes	10	10	100
Number of Instances	60000	50000	50000

taking the sigmoid function to the dot product of the article feature and the true parameter. And then, by adding the value as Bernoulli's parameter, true feedback was calculated and used in the experiment.

$$\mathrm{x}_a = [f_1, f_2, ..., f_d]_a \quad f \sim \mathcal{N}(0, 1)$$

$$\theta^* = [g_1, g_2, ..., g_d] \quad g \sim \mathcal{N}(0, 1)$$

$$\text{true feedback} = 1 \text{ or } 0 \sim \text{Bernoulli}(\sigma(x_a^\top \theta^*))$$

5.1.2 Real-world Dataset

We evaluate NeuralSWAG on real-world datasets too. The MNIST dataset consists of handwritten digit numbers from 0 to 9. The size of each image is 28x28. CIFAR10 dataset consists of 32x32 color images in 10 classes. It contains images of airplanes, automobiles, birds, etc. The CIFAR100 dataset is similar to CIFAR10, but it has 100 labels. It contains superclasses like aquatic mammals, fish, flowers, etc. Each superclass has several sub-classes. Table 5.1 shows statistics of each real-world datasets. Since these datasets are for K-classification tasks, we changed K-classification tasks to K-armed multi-bandit tasks. We used each image's features and labels as context features and rewards of bandit settings.

5.2 Model

For simulation, a straightforward Neural Net model was created. It is a fully linked, $L = 5$ depth neural network.

$$f(x;\theta) = W_L\sigma(W_{L-1}\sigma(...\sigma(W_1x)))$$

where $W_1 \in \mathbb{R}^{m\times d}$, $W_i \in \mathbb{R}^{m\times m}, 2 \leq i \leq L-1$, $W_L \in \mathbb{R}^{m\times 1}$ and $\theta = [\text{vec}(W_1)^\top, ..., \text{vec}(W_L)^\top]^\top \in \mathbb{R}^p$ with $p = md + m^2(L-1) + m$. $\sigma(x) = max\{x, 0\}$ is the rectified linear unit (ReLU) activation function. With $p = 0.5$, dropout was applied to each layer. To simulate click or non-click with Bernoulli distribution for logistic true reward setting, the last layer was created as a sigmoid layer.

5.3 Experiments setting

5.3.1 Setting for Simulation Datasets

We use contextual bandits with context dimension $d = 6$, $K = 20$ actions. The arm set is changed every trial to reflect the real environment in which news article data is continuously updated. The number of rounds $T = 5,000$. The contextual vectors $\{x_{1,1}, ..., x_{T,K}\}$ were randomly chosen from the $\mathcal{N}(0, 1)$. The reward function h is one of the following:

$$h_1(x) = x^\top\theta$$

$$h_2(x) = \frac{1}{1 + e^{-(x^\top\theta)}}$$

$$h_3(x) = (x^\top\theta)^2 - 2(x^\top\theta)$$

$$h_4(x) = (x^\top\theta)^2 - 3(x^\top\theta) + 5$$

$$h_5(x) = \cos(x^\top \theta)$$

$$h_6(x) = \sin(x^\top \theta)$$

The parameter $\theta \in \mathbb{R}^d$ is randomly generated from $\mathcal{N}(0, 1)$. The reward is generated by $r_{t,a} = h_i(x_{t,a}) + \epsilon_t$, where $\epsilon_t \sim \mathcal{N}(0, 1)$.

The algorithm randomly chooses arms for the first phase for Neural SWAG experiments. And it conducts SGD till the SWAG starts epoch. After the SWAG starts the epoch, Neural SWAG executes SWAG, which samples the estimated parameter from the Gaussian distribution until the end of the process. For example, the first 50 trials randomly choose the arm to prevent a case of greedy choice of only one un-optimal arm. In the case of SWAG, scheduling decreased the learning rate from 0.01 to 0.001.

To simulate the situation of online learning as much as possible, the learning proceeds only when the arm displayed to the user is the same as the arm recommended by the algorithm. For a quick experiment, the total trial occurred 3000 times, and the same event occurred about 200 times for the displayed arm and the algorithm chosen arm. For more accurate learning, we plan to gradually increase the total number of trials and proceed with the experiment.

5.3.2 Setting for Real-world Datasets

We used labels 0 and 1 of K labels of real-world datasets to set these two labels as non-click and click feedback of the bandit setting. From train datasets, we filtered out the image features of two labels. We constructed total context vector sets for the bandit algorithm with these image features and labels. At each trial, we sampled 20 context feature vectors and labels from the set and used them as content information.

5.4 Compared Algorithms

In this section, we evaluate NeuralSWAG empirically and compare it with five representative baselines: (1) LinUCB, which is based on UCB but adopts a linear representation. (2) LinTS is based on Thompson Sampling but adopts a linear representation. (3) GLM(Logistic)-UCB (Filippi et al., 2010), which applies a nonlinear link function over a linear function for the UCB method. The logistic function is chosen as a link function for our experiment. (4) GLM(Logistic)-TS applies a nonlinear link function over a linear function for the TS method. The logistic function is chosen as a link function too. (5) Neural ϵ-greedy, which uses neural networks for the epsilon greedy method.

5.5 Experimental Results

5.5.1 Results for Simulation Datasets

Figure 5.1 shows the results of experiments using simulation data. First and foremost, 5.1(a) is the result of experiments using a linear true reward setting. LinUCB and LinTS showed the lowest cumulative regret since they calculate estimated parameters in closed form. Moreover, they showed lower cumulative regrets than the neural ϵ-greedy algorithm. But neural SWAG showed almost the same cumulative regrets as LinUCB and LinTS, although it does not use the exact closed-form of estimated parameters.

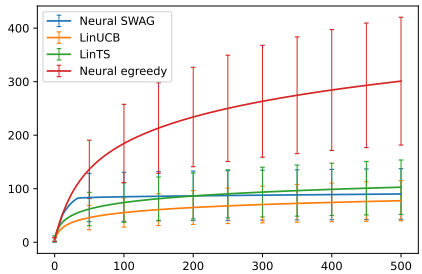
Next, figure 5.1(b) is the result of experiments using a logistic true reward setting that is non-linear. Unlike other regression experiments, classification of click and non-click was conducted in these experiments. GLM(Logistic)-UCB and GLM-(TS) showed the lowest cumulative regrets since they use logistic

regression, which is the true reward setting for the experiments. LinUCB and LinTS showed higher cumulative regrets than GLM(Logistic) bandits. The neural ϵ-greedy offered the highest cumulative regrets. Neural SWAG showed superior performance than GLM(Logistic)-TS even though it does not use the logistic function. It showed higher cumulative regrets than GLM-bandits for the first 200 trials, but it overtook the GLM(Logistic)-TS. It seems that NeuralSWAG could take over GLM(Logistic)-UCB, too, because the overall gradient after 200 trials of NeuralSWAG is lower than GLM(Logistic)-UCB.

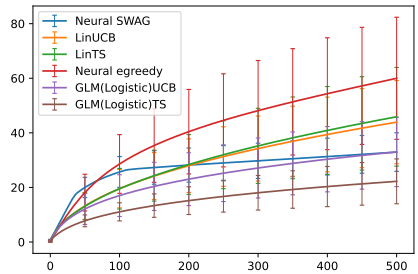
Lastly, figure 5.1(c) (f) shows the result of experiments using other non-linear true reward settings. These experiments contain two quadratic functions, the cosine function, and the sine function. NeuralSWAG offered the lowest cumulative regrets for each result. This suggests that NeuralSWAG can be more competitive than other benchmark algorithms when the true reward function is more complicated. Unlike other linear true reward settings, neural ϵ-greedy showed lower cumulative regrets than LinUCB and LinTS. It suggests that bandit algorithms based on the neural network can be more competitive in a true reward environment with high complexity.

5.5.2 Results for Real-world Datasets

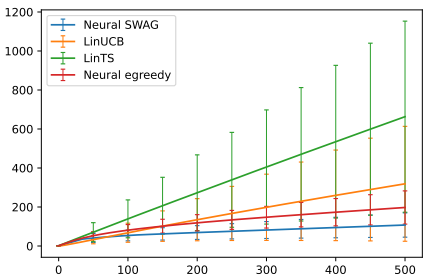
Figure 5.2 is the results of experiments using MNIST and CIFAR10 datasets. NeuralSWAG showed the lowest cumulative regrets for MNIST and CIFAR datasets both. For MNIST datasets, its cumulative regrets increased until less than 100 trials and became almost flat. Neural ϵ-greedy showed the second-lowest cumulative regrets. It offered better results compared to LinUCB and LinTS. It suggests that the neural bandit algorithm has competitiveness on real-world datasets, which has a complex relationship between context features



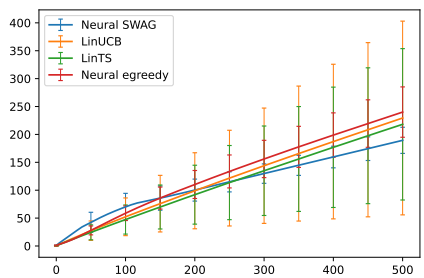
(a) $h_1(x) = x^\top \theta$



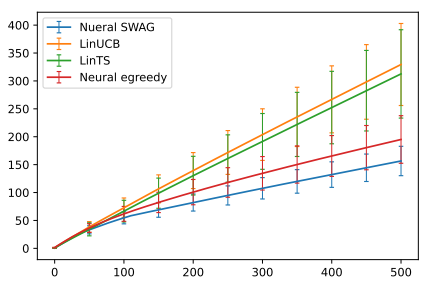
(b) $h_2(x) = \dfrac{1}{1 + e^{-(x^\top \theta)}}$



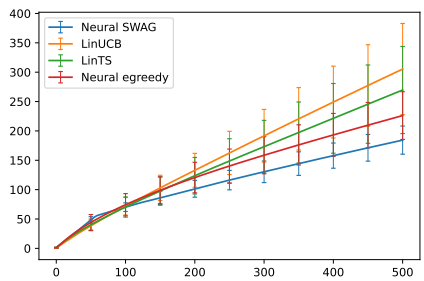
(c) $h_3(x) = (x^\top \theta)^2 - 2(x^\top \theta)$



(d) $h_4(x) = (x^\top \theta)^2 - 3(x^\top \theta) + 5$

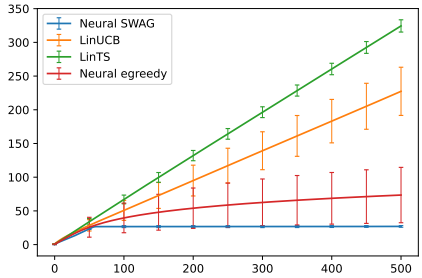


(e) $h_5(x) = \cos(x^\top \theta)$

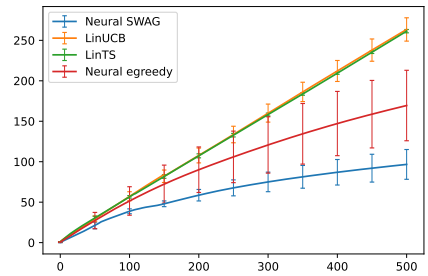


(f) $h_6(x) = \sin(x^\top \theta)$

Figure 5.1 Comparison of NeuralSWAG and baseline algorithms on simulation datasets.



(a) MNIST



(b) CIFAR10

Figure 5.2 Comparison of NeuralSWAG and baseline algorithms on real-world datasets.

and rewards. Since LinUCB and LinTS hypothesize the true reward function as linear, it is natural that these two methods are inferior to the neural network-based model.

For CIFAR10 datasets, it showed similar results to MNIST datasets. NeuralSWAG showed the most competitive outcome. It showed a sub-linear plot correctly, and its gradient fell quicker than other benchmarks. Neural ϵ-greedy also showed the sub-linear plot butt was inferior to NeuralSWAG. In contrast, LinUCB and LinTS showed linear plots similar to the experiment results of MNIST. It suggests again that the neural bandit algorithm has competitiveness on CIFAR10 and MNIST datasets both.

6 CONCLUSIONS

In this paper, we proposed NeuralSWAG, a new algorithm for contextual bandits based on neural networks and Stochastic Weight Averaging Gaussian. Using the bayesian SWAG method, we showed that our algorithm shows superior cumulative regrets results than other state-of-the-art benchmarks, including the linear bandits, the logistic bandits, and ϵ-greedy, especially for the complex true reward function settings. Since it is possible to sample all parameters of neural networks from the gaussian distribution, calculation costs can be saved than general neural networks with lower cumulative regrets. NeuralSWAG showed superior performance not only on simulation datasets but also on real-world MNIST and CIFAR10 datasets.

In the future, we plan to find other optimal hyper-parameters for improving cumulative regrets performance and conduct experiments on other real-world datasets, not only image-classification datasets but also news datasets, customer datasets, etc. Finally, since the contextual bandit problem is highly related to a real-world situation, it is interesting to make a recommendation service with our algorithm and improve the business performance of companies.

Bibliography

Agrawal, Shipra, and Navin Goyal, 2013: Thompson sampling for contextual bandits with linear payoffs. *International conference on machine learning*. PMLR, 127–135.

Kveton, Branislav, Manzil Zaheer, Csaba Szepesvari, Lihong Li, Mohammad Ghavamzadeh, and Craig Boutilier, 2020: Randomized exploration in generalized linear bandits. *International Conference on Artificial Intelligence and Statistics*. PMLR, 2066–2076.

Li, Lihong, Wei Chu, John Langford, and Robert E Schapire, 2010: A contextual-bandit approach to personalized news article recommendation. *Proceedings of the 19th international conference on World wide web*, 661–670.

Li, Lihong, Yu Lu, and Dengyong Zhou, 2017: Provably optimal algorithms for generalized linear contextual bandits. *International Conference on Machine Learning*. PMLR, 2071–2080.

Maddox, Wesley J, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson, 2019: A simple baseline for bayesian uncertainty in deep learning. *Advances in Neural Information Processing Systems*, **32**.

Wilson, Andrew G, and Pavel Izmailov, 2020: Bayesian deep learning and a probabilistic perspective of generalization. *Advances in neural information processing systems*, **33**, 4697–4708.

Zhang, Weitong, Dongruo Zhou, Lihong Li, and Quanquan Gu, 2020: Neural thompson sampling. *arXiv preprint arXiv:2010.00827*.

Zhou, Dongruo, Lihong Li, and Quanquan Gu, 2020: Neural contextual bandits with ucb-based exploration. *International Conference on Machine Learning*. PMLR, 11492–11502.