Report on NeuralUCB: Contextual Bandits with Neural Network-Based Exploration

Anonymous
Department of Computer Science
University of California, San Diego

1 Summary and Review

1.1 Introduction

Contextual bandits are an advanced form of multi-arm bandits which use extra information of the environment to compute the most optimal arm. In this paper the authors present an algorithm named **NeuralUCB** which uses a neural network based feature mapping to construct the UCB. The reward function is learned using a fully connected neural network with depth >= 2. The algorithm leverages the neural network f(**x**,θ) to predict the reward of the context vector **x** and upper confidence bounds. NeuralUCB uses gradient descent to train a deep neural network to learn the reward function based on observed contexts and rewards. Additionally, at each round, NeuralUCB uses the current DNN parameters (θ) to compute an upper confidence bound. NeuralUCB is claimed to achieve $\widetilde{O}(\sqrt{T})$ regret under standard assumptions, where *T* is the number of rounds.

For the analysis of the mentioned regret, the authors assume the reward function *h(x)* to be bounded as ($0 \leq h(x) \leq 1$). Going by this assumption, the worst case regret is not sublinear ($\widetilde{O}(\sqrt{T})$) given that the regret is said to be dependent on an arbitrary large constant S. For this constant *S*, the authors claim only a lower bound on S as mentioned in the remark 4.7 of the paper. There is no upper bound analysis of *S* in the paper. Moreover, in contrast to this assumption on reward function, there is an additional restriction on the reward function to be in the RKHS space to achieve $\widetilde{O}(\sqrt{T})$ regret. This shows that the initial boundedness assumptions on reward function was overstated to achieve the sublinear regret.

The regret analysis is based on Neural Tangent Kernel matrix. This assumes that the width of the neural network is arbitrary large to make a linear approximation in the analysis. However, it is not practical, as neural networks with large number of parameters tend to overfit and perform poorly on real world scenarios. To demonstrate this, I have conducted an experiment to predict CTR on Yahoo news articles.

The experiments used in the paper do not completely demonstrate the capabilities and limitations of NeuralUCB. In this report, I present a set of curated experiments that captures the performance of the algorithm and tests its boundaries in stationary, non-stationary and piece-wise stationary reward functions. As an extension, I have also formulated a new algorithm to handle piece-wise stationary reward functions.

1.2 Related Work

In the Related Work section of the original paper, the authors throws light on some of the relevant work done in the areas of contextual bandits and DNNs. Inspired by the theoretical justification and empirical evidence in these existing literature, the authors presents novel approach to their NN based contextual bandit algorithm.

In the early development of the contextual bandits problems, researchers focused on studying linear bandits[4,6,7,8,23]. Most of methods discussed in this section were based on Upper Confidence

Bound exploration. The original paper discusses some of the popular linear stochastic algorithms introduced by Li et al.[10] and Abbasi-Yadkori et al.[11]. The proposed algorithm NeuralUCB is also based on UCB exploration and it has regret bound that is comparable to that of Abbasi-Yadkori et al[11] in the linear case.

While successful in both theory and practice the linear-reward assumption often fails to hold in practice, which motivates the study of nonlinear[24,25,26] or non-parametric contextual bandits. The authors highlights three approaches to more general non linear bandits. First approach involves family of expert learning algorithms[12, 13] that typically have a time complexity linear in the number of experts. The second approach reduces the bandit problem to supervised learning algorithm like epoch-greedy algorithm[14, 15]. Proposed in 2008 by Langford and Zhang[15], the Epoch Greedy Algorithm is based on balancing the exploration and exploitation to receive a small overall regret in a time horizon T. The algorithm finds a non optimal regret $O(T^{2/3}S^{1/3})$ or better. Here S is the complexity term in a sample complexity bound for standard supervised learning. Later in 2014, Agrarwal et al.[14] develop an algorithm ILOVETOCONBANDITS(Importance weighted LOw-Variance EpochTimed Oracleized CONtextual BANDITS) that enjoys a near-optimal regret, but relies on an oracle, whose implementation still requires proper modeling assumptions. The third approach make use of non-parametric modeling, such as perceptrons[16], random forests [17], Gaussian processes and kernels [18, 19, 27, 28]. Most revelant work being KernelUCB, introduced by Valko et al.[20] which can also be viewd as Kernelized version of LinUCB. It assume the reward function belongs to some Reproducing Kernel Hilbert Space (RKHS) and claims a regret of $\widetilde{O}(\sqrt{\widetilde{d}T})$ where $\widetilde{d}$ is a form of effective dimension similar to NeuralUCB algorithm. Lately, Foster & Rakhlin proposed contextual bandit algorithms[29] which achieve a dimension-independent $O(T^{3/4})$ regret. In the original paper, the author proposes NeuralUCB which achieves a dimension-dependent $\widetilde{O}(\widetilde{d}\sqrt{T})$ regret but with a better dependence on the time horizon.

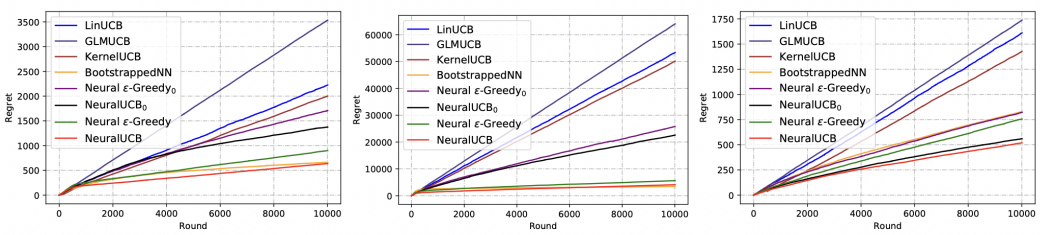
Studies on linear, nonlinear or non-parametric contextual bandits shows that they all require a fairly restrictive assumptions on the reward function. To encouter this shortcomings, researchers have introduced DNNs to learn underlying reward function for contextual bandit problems. Two prominent work being NeuralLinear[21] and NeuralLinearLM[22]. NeuralLinear shows an empirical comparative study on bayesian deep networks for Thompson Sampling. It employs several studies as well as recently developed methods to approximate posterior distributions, combined them with Thompson Sampling and apply them to contextual bandit problems. Instead of computing the exact mean and variance in Thompson sampling, NeuralLinearLM only computes their approximation. Unlike NeuralLinear and NeuralLinearLM that uses all but last layer of DNN, NeuralUCB uses the entire DNN to learn the representation and constructs the upper confidence bound based on the random feature mapping defined by the neural network gradient.

In my search for background literature relevant to this work, I find some existing research papers that are missed in the original paper. One related work being LinRel, introduced in 2002 by Auer [9]. LinRel is a notable algorithm that pioneered the idea of the contextual bandit problem with linear payoffs. It assumes that the rewards are some kind of the linear combination of the previous rewards of the arm. It is the precursor of the LinUCB algorithm mentioned in the original paper.

Other notable work that the authors have missed in their paper is the research authored by Ziwie et al.[30]. In this work they show that neural tanget kernel are universal approximators for continuous function. They also give lower bounds on the number of nodes needed to approximate a function. This result has been directly used by the authors in their regret analysis and it justifies the use of NTK for linear approximation, leading to simplification of the proof.

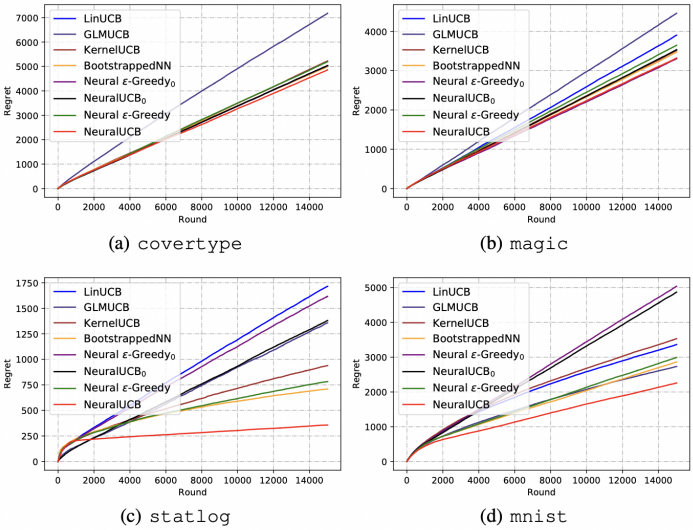
1.3 Technical Results

Neural Contextual Bandits with UCB-based Exploration proposes an algorithm named NeuralUCB that uses a NN based feature mapping to construct the UCB. The reward function is learned using a fully connected neural network with depth L >= 2. The key idea of NeuralUCB is to use a neural network $f(x;\theta)$ to predict the reward of context x and upper confidence bounds. They have experimented on both synthetic and realworld datasets achieving promising empirical results. Results of the paper's experiments can be found in the figures below.



(a) $h_1(\mathbf{x}) = 10(\mathbf{x}^\top \mathbf{a})^2$ (b) $h_2(\mathbf{x}) = \mathbf{x}^\top \mathbf{A}^\top \mathbf{A}\mathbf{x}$ (c) $h_3(\mathbf{x}) = \cos(3\mathbf{x}^\top \mathbf{a})$

Figure 1. Comparison of NeuralUCB and baseline algorithms on synthetic datasets.



(a) `covertype` (b) `magic`

(c) `statlog` (d) `mnist`

Figure 2. Comparison of NeuralUCB and baseline algorithms on real-world datasets.

Both Figure 1 and Figure 2 show cumulative regret of NeuralUCB, LinUCB, GLMUCB, Neural ϵ Greedy, Kernel UCB and Bootstrapped NN. As seen NeuralUCB outperforms all the mentioned algorithms.

1.3.1 Problem Setting

The stochastic **K**-armed contextual bandit problem works in the following way. If the total number of trials are **T**, then for each trial **t**:

1. The agent observes the feature vectors, $x_{t,a}$ for all **K** arms. The vector $x_{t,a}$ summarizes information of environment, and is referred to as the *context*.
2. On the basis of context, the agent chooses one arm and receives a reward of r_{t,a_t}.
3. The algorithm then learns from the current output, and refines its decision policy. It is important to note that the agent does not have any information regarding the reward payouts of the rejected arms.
4. The algorithm's goal is to get maximize the rewards reaped at the end of **T** trials, or minimize the pseudo regret $R_T = E[\sum_{t=1}^{T}(r_{t,a_t^*} - r_{t,a_t})]$

Assumptions on reward function At any timestep *t*, reward function is $r_{t,a_t} = h(x_t, a_t) + \xi_t$. Here h is an unknown function satisfying $0 \leq h(x) \leq 1$for any x, and ξ_t is sub-Gaussian noise conditioned on $x_{1,a_1}, ..., x_{t_{n-1},a_{t_{n-1}}}$ satisfying $E[\xi_t] = 0$.

1.3.2 NeuralUCB Algorithm

The major contribution of this paper are two folds. Firstly, the paper uses neural networks to model rewards from context x. Secondly, it uses gradients from the network to compute the exploration term of the upper confidence bound.

1. Calculate upper confidence bound $U_{t,a}$ using context vector $x_{t,a}$ and θ_{t-1} (parameters of neural network). $U_{t,a}$ consists of two terms, first $f(x_{t,a};\theta_{t-1})$ which is the reward estimate from the neural network. Second term is the exploration weight which is calculated via $\sqrt{g(x_{t,a};\theta_{t-1})^T Z_{t-1}^{-1} g(x_{t,a};\theta_{t-1})}$, where $g(x_{t,a})$ is the gradient of the neural network.
2. Choose action a_t with maximum $U_{t,a}$ and receive reward $r_{t,a}$.
3. Update the algorithm's parameters on the basis of gradients computed at each timestep.
4. Train the neural network using stochastic gradient descent using the $r_{t,a}$ and a_t to refine the reward estimates. The loss function $L(\theta)$ of the neural network has been regularized on the basis of l-2 norm of the parameters (θ).

Algorithm 1 NeuralUCB

Input: Number of rounds T, regularization parameter λ, exploration parameter ν, confidence parameter δ, norm parameter S, step size η, number of gradient descent steps J, network width m, network depth L.
Initialization: Randomly initialize $\boldsymbol{\theta}_0$ as described in the text
Initialize $\mathbf{Z}_0 = \lambda \mathbf{I}$
for $t = 1, \ldots, T$ **do**
  Observe $\{\mathbf{x}_{t,a}\}_{a=1}^K$
  for $a = 1, \ldots, K$ **do**
    Compute $U_{t,a} = f(\mathbf{x}_{t,a};\boldsymbol{\theta}_{t-1}) + \gamma_{t-1}\sqrt{\mathbf{g}(\mathbf{x}_{t,a};\boldsymbol{\theta}_{t-1})^\top \mathbf{Z}_{t-1}^{-1}\mathbf{g}(\mathbf{x}_{t,a};\boldsymbol{\theta}_{t-1})/m}$
    Let $a_t = \text{argmax}_{a\in[K]} U_{t,a}$
  end for
  Play a_t and observe reward r_{t,a_t}
  Compute $\mathbf{Z}_t = \mathbf{Z}_{t-1} + \mathbf{g}(\mathbf{x}_{t,a_t};\boldsymbol{\theta}_{t-1})\mathbf{g}(\mathbf{x}_{t,a_t};\boldsymbol{\theta}_{t-1})^\top/m$
  Let $\boldsymbol{\theta}_t = \text{TrainNN}(\lambda, \eta, J, m, \{\mathbf{x}_{i,a_i}\}_{i=1}^t, \{r_{i,a_i}\}_{i=1}^t, \boldsymbol{\theta}_0)$
  Compute

$$\gamma_t = \sqrt{1 + C_1 m^{-1/6}\sqrt{\log m} L^4 t^{7/6}\lambda^{-7/6}} \cdot \left(\nu\sqrt{\log\frac{\det \mathbf{Z}_t}{\det \lambda\mathbf{I}} + C_2 m^{-1/6}\sqrt{\log m} L^4 t^{5/3}\lambda^{-1/6} - 2\log\delta} + \sqrt{\lambda}S\right)$$
$$+ (\lambda + C_3 tL)\Big[(1-\eta m\lambda)^{J/2}\sqrt{t/\lambda} + m^{-1/6}\sqrt{\log m} L^{7/2} t^{5/3}\lambda^{-5/3}(1+\sqrt{t/\lambda})\Big].$$

end for

Algorithm 2 TrainNN($\lambda, \eta, U, m, \{\mathbf{x}_{i,a_i}\}_{i=1}^t, \{r_{i,a_i}\}_{i=1}^t, \boldsymbol{\theta}^{(0)}$)

Input: Regularization parameter λ, step size η, number of gradient descent steps U, network width m, contexts $\{\mathbf{x}_{i,a_i}\}_{i=1}^t$, rewards $\{r_{i,a_i}\}_{i=1}^t$, initial parameter $\boldsymbol{\theta}^{(0)}$.
Define $\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^t (f(\mathbf{x}_{i,a_i};\boldsymbol{\theta}) - r_{i,a_i})^2/2 + m\lambda\|\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)}\|_2^2/2$.
for $j = 0, \ldots, J-1$ **do**
  $\boldsymbol{\theta}^{(j+1)} = \boldsymbol{\theta}^{(j)} - \eta\nabla\mathcal{L}(\boldsymbol{\theta}^{(j)})$
end for
Return $\boldsymbol{\theta}^{(J)}$.

Note that, although in the original paper, the authors talk about γ_t to be updated via gradient descent. In practice, the computation of γ_t is very expensive and is set as hyper-parameter while conducting the experiment.

1.3.3 Regret Analysis

The analysis uses Neural tangent kernel matrix to come up with the estimate of regret.

Theorem : With probability at least $1 - \delta$, if $m \geq PolyLog(T, L, K, \lambda^{-1}, \lambda_0^{-1}, S^{-1}, log(1/\delta))$ and $S \geq \sqrt{2h^T H^{-1} h}$, regret R_T is defined as

$$R_t \leq \widetilde{O}(\sqrt{T}),$$

given $\widetilde{d}$ as the effective dimension of the neural tangent kernel matrix, defined by,

$$\widetilde{d} = \frac{logdet(I+H/\lambda)}{log(1+TK/\lambda)}$$

Here L is the loss function of the neural network, λ is the regularization parameter, S is a constant which is only lower bounded but no upper bound information is provided, H is the neural tangent kernel matrix $\delta \in (0, 1)$.

Intuitively, the theorem states that for arbitrarily large values of network width, m the regret can be upper bounded via sub-linear function of T.

Understanding the Neural Tangent Kernel Approximation: To analyse the regret associated with the use of neural network, the authors assume the neural network in the infinite-width limit. The neural network in this regime simplify to **linear models** with the neural tangent kernel. This makes gradient descent very easy to analyze. This is because as we increase the width of the network, the loss decreases sharply after few epochs and then it does not change. Consequently, the weights of the neural networks converge and do not change with subsequent epochs. Hence, we can apply first order taylor approximation enabling us to simplify neural networks into linear models.

Once we apply the neural tangent kernel technique for analysis, the math is similar to that in LinUCB and the final upper bound of the regret is also same.

1.4 Review

The experiments and justifications made to support the NeuralUCB in the original paper overlooks some the limitations and restrictions of the algorithm overall. Based on my detailed study of the paper and experimentation of my own, there are some critical observation that I found in the paper which is listed below.

To begin with, in order to prove $\widetilde{O}(\sqrt{T})$ regret, the authors uses the Neural Tangent Kernel matrix approximation. NTK assumes that the width of the neural network is arbitrary large to make a linear approximation in the analysis. This assumption is susceptible to NN width. In practise, an arbitrary large value of the width involves large number of parameters in the neural net which tend to over-fit and affect the overall performance of the model. To demonstrate this on the real world scenario, I have conducted an experiment to predict CTR on Yahoo news articles. With a large number of hidden layers(10000), the CTR rates falls significantly which can be attributed to the over-fitting and lack of generalization in the model.

Furthermore, in the original paper, the authors includes a positive scaling factor γ_t in the NeuralUCB algorithm which is to be updated via gradient descent. In practice, the computation of γ_t is computationally expensive and is set as hyper-parameter while conducting the experiment.

There is also minor wording error in the formulation of regret under the problem statement of the original paper. As per section 2, the authors mentions "Our goal is to *maximize* the following pseudo regret (or regret for short)". The word maximize is misplaced here, instead it should be replaced by minimize. The intention is to minimize the regret which is also explained in the subsequent paragraphs. However this seems as an unintentional error which can ignored.

Lastly, the paper fails to explore the case when the number of arms are variable at each timestep. This is a real world situation, where new items are being added and old items are removed from the candidate pool. In this report, I conduct an experiment to analyse this case, details of which can be found in section 2.4.

Additionally, I have designed a set of experiments that captures the performance of the NeuralUCB and tests its boundaries in stationary, non-stationary and piece-wise stationary reward functions. And as an extension, I have also formulated a new algorithm to better handle piece-wise stationary reward functions.

2 Experiments

The experiments are setup to test the claims made by the authors in the original paper. Additional experiments have been designed to test to check if the methodology is applicable to different scenarios. In the experiment conducted the proposed model (NeuralUCB) is evaluated against LinUCB as the base model.

2.0.1 Experimentation setup

In order to verify the claims made in the original paper, we used the same synthetic datasets used in the paper to reproduce similar results. We have compared the performance of the NeuralUCB algorithm with LinUCB. This work can be found in section 2.1.

The authors assume the reward function is stationary in nature, i.e. the reward function h does not depend on the time parameter. As the first extension, I have included an experiment to run this algorithm on a time dependent reward function. This work can be found in section 2.2.

de4Based on the above experiments, I have formulated an additional experiment for piece-wise stationary function details which can be found in section 2.3

Additionally, as the original paper does not conduct any experiment to test the algorithm on any large scale real life scenario, as my next extension, I have analysed the performance of the algorithm for CTR prediction. The data used for this analysis is a subset of **Yahoo!Front Page Today Module User Click Log Dataset** . In this experiment, we also test how does the NeuralUCB algorithm perform under **variable bandit scenario**. This work can be found in section 2.4.

I have also designed an experiment to run over an Atari game on the openai gym called *SpaceInvaders*. Here I wanted to see, how does the algorithm perform in an online scenario.

The original paper lacks enough analysis on the type of reward functions that can be modelled using the NeuralUCB algorithm. I present here a detailed analysis by using stationary, non-stationary and Piecewise stationary reward function.

2.1 Stationary Functions

In these set of experiment, I replicated the results in the paper over the synthetic dataset. I set context dimension d = 4 and K = 4 actions. Number of rounds T are set to 10000 and 10% Gaussian noise was added. I used three reward function:

- $h(x) = 10(x^T a)$
- $h(x) = 100(x^T a)^2$
- $h(x) = 10cos(3x^T a)$

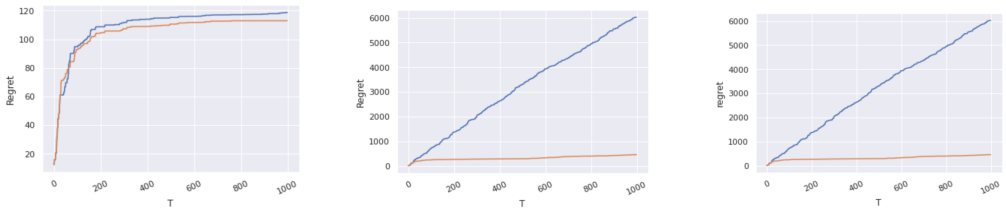


Figure 1: regret on linear, quadratic and cosine **stationary** functions, Blue is LinUCB and orange is NeuralUCB

As we can see in Figure 1, the NeuralUCB is able to learn the the non-linear regret function quite efficiently, whereas LinUCB fails to do so. This is expected as neural networks have the power to learn non-linearity quite well.

2.2 Non Stationary Reward Function

In this paper, the reward function is assumed to be stationary, but this assumption breaks in many real life situations. Since, the neural networks can handle a high degree of non-linearity, I wanted to see how does the neuralUCB algorithm performs when the reward function is made non-stationary. I make it non stationary why adding time as an input metric.

I used three reward function:

- $h(x,t) = 10(x^T a)/t$
- $h(x,t) = 100(x^T a)^2/t$
- $h(x,t) = 10cos(3x^T a)/t$

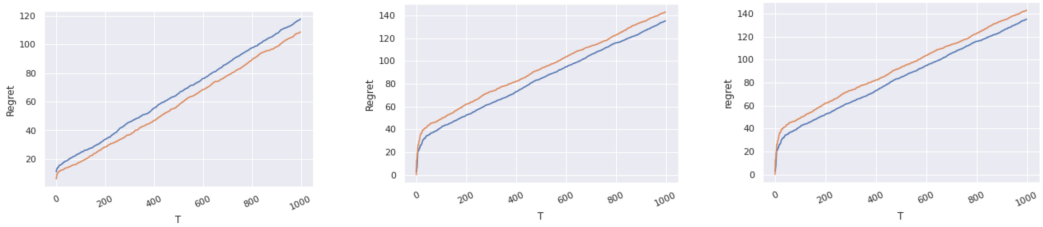


Figure 2: regret on linear, quadratic and cosine **non stationary** functions, Blue is LinUCB and orange is NeuralUCB

It is evident from Figure 2 that both the algorithm fail to efficiently model the reward functions. This is due to the fact that once a set number of trials have been conducted, both the UCB algorithm do not explore as much. To accomodate these kind of reward function, changes will have to be made in the algorithm to forcefully explore after some iterations, when the cumulative regret starts increasing.

2.3 Piece-wise stationary Functions

As we saw in the previous experiment that the NeuralUCB is not able to handle non-stationary reward functions, I decided to experiment with piece-wise stationary functions. To include these in the dataset, I added experimented with the following function:

$$h(x,t) = \left\{ \begin{array}{ll} 10(x^T a), & 0 \leq t \leq 300 \\ 100(x^T a)^2, & 301 \leq t \leq 600 \\ cos(3x^T a), & 601 \leq t \leq 1000 \end{array} \right\}$$

After a few iterations, when the reward have been estimated, the NeuralUCB algorithm never explores anymore. So, when the reward function changes abruptly (Figure 3), the algorithm fails to adapt and the regret increases rapidly. To handle this case, **I have proposed a modification in the NeuralUCB** algorithm, which can be found in section 3.

2.4 CTR prediction on News Articles

In the paper, the authors have not provided any analysis on a large real world dataset. So, in this work I have compared performance of NeuralUCB with LinUCB. The dataset contains featured articles on Yahoo's front page and user responses. The features of users and articles have already been provided in the dataset in the form of a 6-dimensional vector. The original dataset contains of around 43 million rows, in the experiment I only work with a subset of the dataset (100000 rows and 80 news articles). Hence the number of arms K in this case is 80.

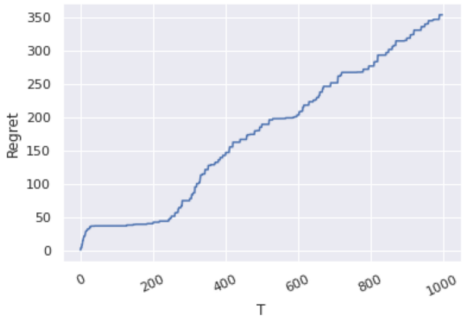


Figure 3: regret piece-wise linear, quadratic and cosine functions for NeuralUCB

One other analysis, that the authors have not provided in their work is the ability to handle variable number of arms at each timestep. If we make few changes in the algorithm the current approach should be able to handle this case as well. Also, no major changes are required to the regret analysis to achieve the proposed bounds. In this experiment I consider variable number of bandits at each timestep. Results of this experiment are mentioned in Table 1.

Table 1: Click Through Rates

Algorithm	CTR
LinUCB	2.41%
NeuralUCB	3.15%
NeuralUCB with limited arms	3.04%
NeuralUCB for NTK approx	2.53%

As evident from the CTR rates the algorithm performs far better than the LinUCB, since it is effectively able to capture the non-linearity. Also, limiting the number of arms at each timestep does not have an adverse effect on the performance. Note that, the dataset was constructed in such a way that the candidate news articles were given with each row. In order to test, if the algorithm can handle variable arms, I had forcefully enabled all 80 arms to be considered to make a comparison.

2.5 Impact of Neural Tangent Kernel Approximation on performance

Since in the literature while deriving the upper bound of total regret, neural kernel approximation has been used, I have conducted an experiment where I initialize the neural network with very large the number of hidden layers (10000). I ran this scenario on the dataset used in section 2.4. The CTR rates for this experiment (Table 1) is very less. This was due to the fact the neural network was over-fitted and failed to generalize over new scenarios. Also, the training time had increased two folds.

We can conclude that though the Neural tangent kernel matrix was crucial to prove the bounds, it has adverse impact on the performance of the algorithm in real life scenarios.

2.6 SpaceInvader Atari game

In this experiment, I have run the neuralucb algorithm on SpaceInvader using the openai's gym. Here I have included the the observation provided at the each timestep in the feature vector. This provides us with the state information and helps the neural network to take the correct decision. Reward function has been plotted below. In the visualization, we can see that the UCB algorithm has learned to hide, to get saved from the bullet. Also, it has learned that it has to shoot in clear air in order to gain the rewards.

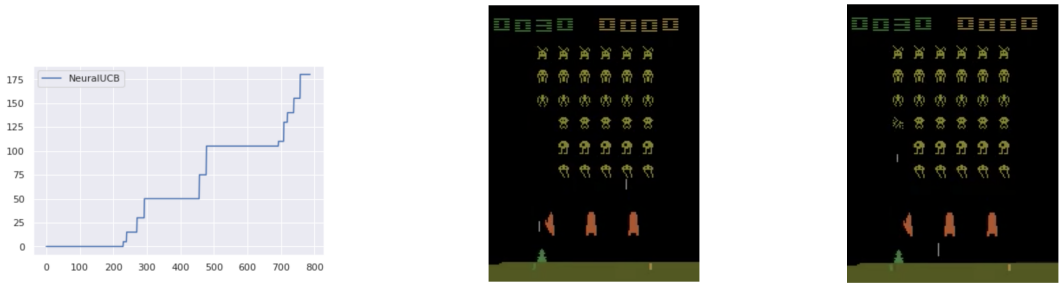
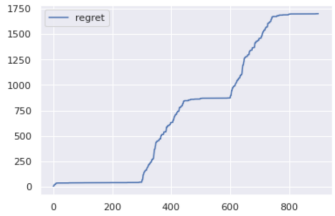


Figure 4: Visualization showing cumulative reward, hiding and shooting

3 Extension

In addition to the extensions captured in the above experiments, I have also extended the algorithm to handle piece-wise stationary reward functions. In real world cases, we seldom encounter stationary functions. Almost always the reward function changes according the situation. For example, on social media, there is always a seasonality associated with content and after some time the current viral content fades out and new content comes up. The main reason neuralUCB cannot handle this case is because it stops doing exploration after few timesteps. In order to accommodate these scenarios, I have proposed a new algorithm which is based on the neuralUCB algorithm where I introduce a new parameter called **exploration enabler**. This term keeps track of the average rewards earned for the past k iterations and when the average reward dips by a factor of f, it resets the parameter of the neural network forcing it to relearn the distribution of reward functions. To demonstrate the algorithm I ran it on the piece-wise stationary function mentioned in section 2.3. Regret graph can be found the the Figure 5.



The approach works on both piecewise stationary and stationary reward functions. It will still not handle cases when the piece-wise function changes very rapidly. If this is the case, then the neural network does not get enough timesteps to train.

References

[1] Alexander, J.A. & Mozer, M.C. (1995) Template-based algorithms for connectionist rule extraction. In G. Tesauro, D.S. Touretzky and T.K. Leen (eds.), *Advances in Neural Information Processing Systems 7*, pp. 609–616. Cambridge, MA: MIT Press.

[2] Bower, J.M. & Beeman, D. (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System.* New York: TELOS/Springer–Verlag.

[3] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *Journal of Neuroscience* **15**(7):5249-5262.

[4] Abe, N., Biermann, A. W., and Long, P. M. Reinforcement learning with immediate rewards and linear hypotheses. Algorithmica, 37(4):263–293, 2003.

[5] Dani, V., Hayes, T. P., and Kakade, S. M. Stochastic linear optimization under bandit feedback. 2008.

[6] Rusmevichientong, P. and Tsitsiklis, J. N. Linearly parame- terized bandits. Mathematics of Operations Research, 35 (2):395–411, 2010

[7] Chu, W., Li, L., Reyzin, L., and Schapire, R. Contextual bandits with linear payoff functions. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pp. 208–214, 2011.

[8] Auer, P. Using confidence bounds for exploitation- exploration trade-offs. Journal of Machine Learning Research, 3(Nov):397–422, 2002

[9] P. Auer and N. CesaBianchi, "Finitetime Analysis of the Multiarmed Bandit Problem," Machine Learning 47, p. 22, 2002.

[10] Li, L., Chu, W., Langford, J., and Schapire, R. E. A contextual-bandit approach to personalized news article recommendation. In Proceedings of the 19th international conference on World wide web, pp. 661–670. ACM, 2010.

[11] Abbasi-Yadkori, Y., Pa l, D., and Szepesva ri, C. Improved algorithms for linear stochastic bandits. In Advances in Neural Information Processing Systems, pp. 2312–2320, 2011

[12] Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. E. The nonstochastic multiarmed bandit problem. SIAM Journal on Computing, 32(1):48–77, 2002.

[13] Beygelzimer, A., Langford, J., Li, L., Reyzin, L., and Schapire, R. E. Contextual bandit algorithms with supervised learning guarantees. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pp. 19–26, 2011.

[14] A. Agarwal, D. Hsu, S. Kale, J. Langford, L. Li, and R. E. Schapire, "Taming the Monster: A Fast and Simple Algorithm for Contextual Bandits," in Proceedings of the 31 st International Conference on Machine Learning, Beijing, China, 2014.

[15] J. Langford and T. Zhang, "The epochgreedy algorithm for multiarmed bandits with side information," in Advances in Neural Information Processing Systems 20, J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, Eds. Curran Associates, Inc., 2008, pp. 817–824.

[16] Kakade, S. M., Shalev-Shwartz, S., and Tewari, A Efficient bandit algorithms for online multiclass prediction. In Proceedings of the 25th international conference on Machine learning, pp. 440–447, 2008.

[17] Feraud, R., Allesiardo, R., Urvoy, T., and Cl ´ erot, F. Random forest for the contextual bandit problem. In Artificial Intelligence and Statistics, pp. 93–101, 2016.

[18] Kleinberg, R., Slivkins, A., and Upfal, E. Multi-armed bandits in metric spaces. In Proceedings of the fortieth annual ACM symposium on Theory of computing, pp. 681–690. ACM, 2008.

[19] Srinivas, N., Krause, A., Kakade, S., and Seeger, M. Gaussian process optimization in the bandit setting: no regret and experimental design. In Proceedings of the 27th International Conference on International Conference on Machine Learning, pp. 1015–1022. Omnipress, 2010.

[20] Valko, M., Korda, N., Munos, R., Flaounas, I., and Cristianini, N. Finite-time analysis of kernelised contextual bandits. arXiv preprint arXiv:1309.6869, 2013.

[21] Riquelme, C., Tucker, G., and Snoek, J. Deep Bayesian bandits showdown in International Conference on Learning Representations, 2018.

[22] Zahavy, T. and Mannor, S. Deep neural linear bandits: Overcoming catastrophic forgetting through likelihood matching. arXiv preprint arXiv:1901.08612, 2019.

[23] Chu, W., Li, L., Reyzin, L., and Schapire, R. Contextual bandits with linear payoff functions. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pp. 208–214, 2011.

[24] Filippi, S., Cappe, O., Garivier, A., and Szepesvari, C. Parametric bandits: The generalized linear case. In Advances in Neural Information Processing Systems, pp. 586–594, 2010.

[25] Li, L., Lu, Y., and Zhou, D. Provably optimal algorithms for generalized linear contextual bandits. In Proceedings of the 34th International Conference on Machine LearningVolume 70, pp. 2071–2080. JMLR. org, 2017.

[26] Jun, K.-S., Bhargava, A., Nowak, R. D., and Willett, R. Scalable generalized linear bandits: Online computation and hashing. In Advances in Neural Information Processing Systems 30 (NIPS), pp. 99–109, 2017.

[27] Krause, A. and Ong, C. S. Contextual Gaussian process bandit optimization. In Advances in neural information processing systems, pp. 2447–2455, 2011.

[28] Bubeck, S., Munos, R., Stoltz, G., and Szepesvari, C. X-armed bandits. Journal of Machine Learning Research, 12(May):1655–1695, 2011.

[29] Foster, D. J. and Rakhlin, A., Beyond ucb: Optimal and efficient contextual bandits with regression oracles. arXiv preprint arXiv:2002.04926, 2020.

[30] Ziwei Ji, Matus Telgarsky, Ruicheng Xian, Neural tangent kernels, transportation mappings, and universal approximation in International Conference on Learning Representations, 2019.