NEURAL NETWORK BANDIT LEARNING BY LAST LAYER MARGINALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a new method for training neural networks online in a bandit setting. Similar to prior work, we model the uncertainty only in the last layer of the network, treating the rest of the network as a feature extractor. This allows us to successfully balance between exploration and exploitation due to the efficient, closed-form uncertainty estimates available for linear models. To train the rest of the network, we take advantage of the posterior we have over the last layer, optimizing over all values in the last layer distribution weighted by probability. We derive a closed form, differential approximation to this objective and show empirically over various models and datasets that training the rest of the network in this fashion leads to both better online and offline performance when compared to other methods.

1 INTRODUCTION

Applying machine learning models to real world applications almost always involves deploying systems in dynamic, non-stationary environments. This dilemma requires models to be constantly re-updated with new data in order to maintain a similar model performance across time. Of course, doing this usually requires the new data to be relabeled, which can be expensive or in some cases, impossible. In many situations, this new labeled data can be cheaply acquired by utilizing *feedback* from the user, where the feedback/reward indicates the quality of the action taken by the model for the given input. Since the inputs are assumed independent, this task can be framed in the contextual bandit setting. Learning in this setting requires a balance between *exploring* uncertain actions (where we risk performing sub optimal actions) and *exploiting* actions the model is confident will lead to high reward (where we risk missing out on discovering better actions).

Methods based on Thompson sampling (TS) or Upper Confidence Bounds (UCB) provide theoretically (Auer et al., 2002; Agrawal & Goyal, 2012) and empirically established ways (Li et al., 2010; Chapelle & Li, 2011) for balancing exploration/exploitation in this setting. Unfortunately, both methods require estimation of model uncertainty. While this can be done easily for most linear models, it is a difficult and open problem for large neural network models underlying many modern machine learning systems. An empirical study by Riquelme et al. (2018) shows that having good uncertainty estimates is vital for neural networks learning in a bandit setting. Closed formed uncertainty estimations (and online update formulas) are available for many linear models. Since the last layer of many neural networks are usually (generalized) linear models, a straightforward way for learning neural networks in a bandit setting is to estimate the uncertainty (as a distribution over weights) on the last layer only, holding the previous layers fixed as feature functions which provide inputs to the linear model. This method (and variants thereof) has been proposed in bandit settings (Riquelme et al., 2018; Liu et al., 2018) as well as other related settings (Snoek et al., 2015; O'Donoghue et al., 2018; Lázaro-Gredilla & Figueiras-Vidal, 2010) and has been shown to work surprisingly well considering its simplicity. This style of methods, which we refer to as *Bayesian last layer* or BLL methods, also has the advantage of being both relatively model-agnostic and scalable to large models. Of course, BLL methods come with the tacit assumption that the feature functions defined by the rest of the network output good (linearly separable) representations of our inputs. This means that, unless the input data distribution is relatively static, the rest of the network will need to be updated in regular intervals to maintain low regret.

In order to maintain low regret, the retraining objective must: 1) allow new data to be incorporated quickly into the learned model, and 2) prevent previously learned information from being quickly forgotten. Previous papers retrain BLL methods simply by sampling minibatches from the *entire* pool of previously seen data and maximizing log-likelihood over these minibatches, which fails to meet the first criteria above.

In this paper we present a new retraining objective for BLL methods meeting both requirements. We avoid retraining the last layer with the entire network (throwing out the uncertainty information we learned about the last layer) or retraining with the last layer fixed (fixing the last layer to the mean of its distribution). Instead, we utilize the uncertainty information gathered about the last layer, and optimize the expected log-likelihood of both new and old data, marginalizing[1] over the entire distribution we have on the last layer. This gives a more robust model that performs relatively well over *all* likely values of the last layer. While this objective cannot be exactly computed, we derive a closed form, differentiable, approximation. We show that this approximation meets both criteria above, with a likelihood term to maximize that depends only on the new data (meeting the first point), and a quadratic regularization term that is computed only with previously seen data (meeting the second point). We show empirically that this method improves regret on the most difficult bandit tasks studied in Riquelme et al. (2018). We additionally test the method on a large state-of-the-art recurrent model, creating a bandit task out of a paraphrasing dataset. Finally, we test the method on convolutional models, constructing a bandit task from a benchmark image classification dataset. We show that our method is fast to adapt to new data without quickly forgetting previous information.

2 Problem Setting and Related Work

Contextual bandits are a well researched class of sequential decision problems (Wang et al., 2005; Langford & Zhang, 2007), of which, many variants exist. In this paper, we mainly consider the multiclass contextual bandit problem studied in Kakade et al. (2008). The problem takes place over T online rounds. On round t, the learner receives a context x_t, predicts a class label y_t, and receives binary reward r_t indicating whether the chosen label is correct. No other information is received about the other classes not picked. In our setting, we assume each class c (the arms of the bandit) has associated with it a vector z_c and that the probability of a class label is modeled by: $p(c|x, z_c) = \sigma(z_c^T f_\theta(x))$, where σ is the logistic function and f_θ is a feature function parameterized by θ. In our case, f_θ defines a neural network, while z_c can be seen as the last layer of the network[2].

Our goal is to get low regret, $R = \sum_i^T r_i^* - \sum_i^T r_i$, where r_i^* is the optimal reward at step i. The key to getting low regret is employing a policy for balancing exploration and exploitation. If we capture the uncertainty in each z_c at time t by modeling its posterior distribution over previous data D_{t-1} as a multivariate Gaussian, $z_c \sim p(z_c|D_{t-1})$, then we can easily deploy sound exploration strategies such as Thompson sampling or UCB. If we hold f_θ fixed, then we can easily model this distribution by doing an online Bayesian regression on the outputs of f_θ, which gives us closed form formulas for updating the posterior over the last layer (specifically, its mean and covariance) given a single datapoint.[3] When f_θ is a neural network, then this is an instance of a BLL method.

2.1 Bayesian Last Layer methods and the Benefits of Marginalization

BLL methods have been shown to be an effective, model agnostic, and scalable way to deal with exploration problems involving neural networks. Previous work has found them to be a pragmatic method for obtaining approximate uncertainty estimates for exploration (Liu et al., 2018; Riquelme et al., 2018; O'Donoghue et al., 2018; Azizzadenesheli et al., 2018) and as proxies for Gaussian processes in both Bayesian Optimization problems (Snoek et al., 2015) and general regression tasks (Lázaro-Gredilla & Figueiras-Vidal, 2010).

If f_θ is fixed, then z_c can be updated efficiently in an online manner. An unanswered question still remains however: how does one actually update and learn f_θ? If you don't care about achieving low regret (ie you only care about *offline* performance), then the answer is easy; just gather your

[1] We do not marginalize in the sense of 'summing out', but rather optimizing over all values of the last layer.

[2] Note that we make no assumption on the form of f_θ, which can be anything differentiable

[3] Either a Bayesian linear or logistic regression will work. However, the Bayesian linear regression is much more efficient in an online setting with similar performance so we use it in all of our implementations.

data, train f_θ offline, possibly with off-policy learning methods (Strehl et al., 2010; Joachims et al., 2018), and learn the Bayesian regression *post-hoc*. Of course, if you are concerned about *online* performance (regret) then this is not a viable option.

A training method for f_θ must take care of two things: *when* do we update the feature functions and *what* do we update them with? A reasonable answer to the former question is to update on a fixed schedule (every T rounds). In this paper, we focus on answering the latter questions of which there are two obvious solutions, each with a corresponding problem:

(1) Sample minibatches only from recent data. *Problem*: We may overfit on this data and forget old information.

(2) Sample minibatches uniformly from the set of all collected data (both recent and old). *Problem*: We have lost the ability to adapt quickly to new data. If the input distribution suddenly shifts, we will likely have to wait many iterations before newer data becomes common enough in our minibatch samples, all while our regret increases.

One thing to consider is that when it comes time to train our feature functions, we have access to a distribution over our last layer. If, for example, our distribution has suddenly shifted, then the last layer distribution should have more variance, ideally placing density on last layers that do well on older data, as well as those that fit well to the new data. If the distribution remains the same, then the variance should be low and density should be placed on relatively few values. Intuitively, we can get the best of both worlds (ability to adapt or retain information when needed) by optimizing over *all values of the last layer* weighted by their probability. In the next section, we derive a local approximation to this objective that shows this indeed is the case.

3 Optimization over the Last Layer Distribution

Let D_t and D_{t-1} be the most recent set of data collected online, and the set of all previously collected data, respectively. Additionally, assume a zero mean Gaussian prior over the last layer, $p(z|\theta)$ that is constant with respect to θ. Recall that during online training we fix the parameters $\theta = \theta_{t-1}$, and model the distribution $Q = p(z_c|D_t, D_{t-1}, \theta_{t-1})$. We want a value of θ such that both our data *and* the last layer values drawn from Q are likely. Thus our objective is to maximize:

$$\mathbb{E}_{z\sim Q}[\log p(D_t, D_{t-1}, z|\theta)] \tag{1}$$

We can write the marginal likelihood as a convolution between a logistic function and a Gaussian (based on our assumption of zero mean Gaussian prior $p(z|\theta)$), which can be approximated in closed form as per MacKay (1992):

$$p(D|\theta) = \int \sigma(z^T f_\theta(D))p(z|\theta)dz \approx \sigma\Big(\frac{\mu f_\theta(x_i)}{\sqrt{1 + \frac{\pi}{8} f_\theta(x_i)^T \Sigma f_\theta(x_i)}}\Big) \tag{2}$$

Where μ is the mean of $p(z|\theta)$. Since we have a zero mean prior, the above term evaluates to $\sigma(0)$ whose value is a constant $\frac{1}{2}$.

Using this result, we can rewrite equation (1) as:

$$\mathbb{E}_{z\sim Q}[\log p(D_t|z, \theta)] - KL[Q, p(z|D_{t-1}, \theta)] + c \tag{3}$$

Where c is a constant entropy term which can be ignored. For the first expectation term, we can use a second order Taylor expansion around $\mathbb{E}_{z\sim Q}[\log p(D_t|z, \theta)]$ to get a closed form approximation[4]:

$$\mathbb{E}_{z\sim Q}[\log p(D_t|z, \theta)] \approx \log \mathbb{E}[p(D_t|z, \theta)] - \frac{\text{Var}[p(D_t|z, \theta)]}{2\mathbb{E}[p(D_t|z, \theta)]^2} \tag{4}$$

This approximation was used in Teh et al. (2007) and shown to work well empirically. The expectations in equation (4) are again logistic/Gaussian convolutions and can be approximated in closed form via equation (2). A closed form approximation (derived in a similar fashion to equation (2)) can be used to evaluate $\text{Var}[p(D_t|z, \theta)]$ (Mahajan et al., 2012).

[4]Since we can easily sample from Q, the term may also be approximated via Monte Carlo. We don't explore this approach here.

The KL term in equation (3) can also be approximated locally with a second Taylor expansion around the current value of $\theta = \theta_{t-1}$. Let $K(\theta) = KL(Q||p(z|D_{t-1}, \theta))$. Then, the second order Taylor expansion around $\theta = \theta_{t-1}$ is:

$$K(\theta) \approx K(\theta_{t-1}) + K'(\theta_{t-1})(\theta - \theta_{t-1}) + \frac{1}{2}(\theta - \theta_{t-1})K''(\theta_{t-1})(\theta - \theta_{t-1})^T \tag{5}$$

Utilizing properties of KL divergence, as well as equation (2), it can be derived [5]that $K'(\theta_{t-1})$ will evaluate to 0, and $K''(\theta_{t-1})$ will evaluate to βF_P, where $\beta = \frac{\mathbb{E}_{z\sim P}[p(D_t|z,\theta_{t-1})]}{p(D_t|\theta_t)}$ and F_P is the Fisher Information Matrix of $P = p(z|D_{t-1}, \theta_{t-1})$. Getting rid of constants, we can write the local KL approximation (when θ is close to θ_{t-1}) as:

$$KL[Q, p(z|D_{t-1}, \theta)] \approx \frac{1}{2}(\theta - \theta_{t-1})\beta F_P(\theta - \theta_{t-1})^T \tag{6}$$

The term β defines the ratio between the expected data likelihood given the last layer z distributed as $z \sim p(z|D_{t-1}, \theta_{t-1})$ and the expected data likelihood given the last layer is distributed under the prior $p(z|\theta)$. This indicates that the better our previous values of θ_{t-1} and z explain the data, the more we should regularize when incorporating new data (i.e raising the value of β). In practice, these values may be computed or approximated, however for efficiency, we treat β as a hyperparameter, and linearly increase it throughout the learning process.

Our final objective to optimize is thus:

$$\mathbb{E}_{z\sim Q}[\log p(D_t, D_{t-1}, z|\theta)] \approx \mathbb{E}_{z\sim Q}[\log p(D_t|z, \theta)] - \frac{1}{2}(\theta - \theta_{t-1})\beta F_P(\theta - \theta_{t-1})^T \tag{7}$$

Notice that the old data is now only used to calculate the Fisher information matrix F_P and is not actually involved in the optimization. Thus, the optimization (at least temporarily) over *all* our data can be done by simply drawing minibatches from the new data *only*, while the old data is only used to calculate the regularization term. The practical benefit of this is that the regularization term can be easily computed *in parallel* while doing the online Bayesian regression and collecting new data. The quadratic regularization term shares similarities to objective functions in continual learning which aim to prevent catastrophic forgetting (Kirkpatrick et al., 2016; Ritter et al., 2018).

4 The Full Algorithm

Combining the online learning and the network retraining stages described in the previous section gives us the general form of the iterative algorithm we study in this paper. The algorithm alternates between two stages:

Online Phase: As input, take in a set of data D_{t-1}, the posteriors (one for each arm) of the last layer conditioned on previous data $p(z|D_{t-1})$ as well as a *fixed* value of f_θ. This phase takes place over a series of T online rounds. In every round, the learner receives a context x_i, and uses the posteriors over the last layer to decide which arm to pull (via Thompson sampling or UCB). The learner receives feedback y_i upon pulling the arm c, and updates the posterior over z_c with it. After T rounds, the learner outputs the updated posteriors over z, and the data collected this phase, D_t.

Offline/Retraining Phase: As input, take in D_t, D_{t-1}, and the posteriors over z. Retrain f_θ using method described in Section 3. Set $D_{t-1} = D_t \cup D_{t-1}$. Recompute the posteriors over z conditioned on D_{t-1} using the new value of f_θ. Output D_{t-1}, f_θ, and the updated posteriors $p(z|D_{t-1})$.

The marginalization method described in Section 3 is one type of retraining method. We compare it against two other methods in the next section and present results for various experiments.

5 Evaluation

We evaluate our technique across a diverse set of datasets and underlying models. As an initial sanity check, we first evaluate our method on the three most difficult (but low dimensional) bandit tasks

[5]Detailed derivations are provided in Appendix 7.1.

analyzed in Riquelme et al. (2018). We next look at two higher dimensional problems and models; one being a Natural Language Processing (NLP) task using a state-of-the-art recurrent model and the other being a vision task using a convolutional model. In particular, we look at the degree at which each method can achieve both good online performance (regret) *and* good offline performance (offline test set accuracy), even in the face of large shifts in the data distribution. We provide additional details on hyperparameters, experimental setup, and dataset information in Appendix 7.4. All experiments are run 5 times, and reported with the mean and standard error.

5.1 Baseline Methods

We evaluate all the datasets against the baseline presented in Riquelme et al. (2018), as well as a variant of our proposed method.

Bandit feedback. In this setting the models are trained using bandit feedback as the label:

Marginalize. This is our method of marginalizing over all values of the last layer for the neural network training. Minibatches are sampled from the new data only and the regularization term is computed from the old data.

Sample New. This baseline creates minibatches using only the newly collected data, optimizing the likelihood of the new data only. It is equivalent to our method with a regularization constant of zero. As mentioned in Section 2.1, this method is good at adapting to new data but has a drawback of forgetting the old information.

Sample All. This is the retraining method presented in (Riquelme et al., 2018). In this method, a set number minibatches are created by uniformly sampling from *all* collected data (both old and new). SGD gradient updates are then performed using these batches. This method is slow to adapt but retains older information (refer Section 2.1).

Full feedback. In this setting models are trained using all the labels for the datasets:

Batch Train. When evaluating the offline accuracy, we also give the results for a model that has been trained on the shuffled data in batch mode, with all the labels for the dataset (i.e. full feedback). This measures how well we could do given we had access to all the labels (instead of just the bandit feedback), and trained in a normal offline setting. Surprisingly, as we see in some cases, training online with marginalization sometimes performs comparable to training offline.

5.2 Low Dimensional Bandit Tasks

We first confirm that our method gives good online performance on simpler, but previously studied, problems in Riquelme et al. (2018).

We present results on the three hardest bandit tasks analyzed in Riquelme et al. (2018), the Census, Jester, and Adult dataset. The bandit problems for these datasets are defined as in previous work:

Census and Adult. Both these datasets are used for multiclass classification problem. Census dataset has 9 classes whereas Adult dataset consists of 14 classes. For both datasets the bandit problem is created as follows: for each class we assign an arm, and each arm is associated with a logistic regression (parametrized by a vector) that takes a context as input and returns the expected reward (0 or 1) for selecting the arm. In the online round we receive a context (feature vector) and pick an arm according to some policy (like UCB), and receive a reward. Only the picked arm is updated in each round.

Jester (Goldberg et al., 2001) This dataset consists of jokes with their user rating. For the bandit problem, the model receives a context representing a user along with 8 jokes out of which it is required to make recommendation of 1 joke. In this setting each joke is defined as a bandit arm. The problem here is similar to above with the exception that each arm is associated with a linear regression and outputs the predicted user rating for the selected joke. The reward returned is the actual user rating for the selected joke.

Dataset	Method	Average Cum. Regret		Relative Regret (%)	
		UCB	TS	UCB	TS
Jester	Marginalize	2.96 ±0.01	2.95 ±0.03	59.3 ±0.4	58.1±0.5
	Sample New	2.96 ±0.02	2.98 ±0.04	59.1 ±0.4	60.6±0.5
	Sample All	3.58 ±0.01	3.54 ±0.03	72.8 ±0.4	71.1±0.5
Census	Marginalize	0.30 ±0.01	0.29 ±0.03	33.0 ±0.3	33.6±0.5
	Sample New	0.29 ±0.02	0.28 ±0.03	33.3 ±0.4	32.9±0.4
	Sample All	0.33 ±0.01	0.34 ±0.02	37.7 ±0.3	38.5±0.4
Adult	Marginalize	0.69 ±0.01	0.71 ±0.01	75.0 ±0.3	76.0±0.3
	Sample New	0.70 ±0.01	0.69 ±0.02	75.6 ±0.3	75.4±0.4
	Sample All	0.79 ±0.01	0.79 ±0.01	85.9 ±0.2	85.2±0.3

Table 1: Low Dimensional Bandit Task Results for both UCB and Thompson Sampling. The table shows results for average cumulative regret and regret relative to the regret when selecting an arm at random.

5.2.1 Results and discussion

As previously done in Riquelme et al. (2018), we use a two layer MLP as the underlying model, using the same configuration across all methods. For *Marginalize* and *Sample New*, we perform the retraining after 1000 rounds. For *Sample All* we update after 100 rounds just like Riquelme et al. (2018). In Table 1 we report the average cumulative regret as well as the cumulative regret relative to a policy that selects arms uniformly at random. We report the results for both Thompson Sampling (TS) and for UCB policies. Results are similar for either UCB and TS which shows that policies does not influence performance of the training mechanisms.

On most of the tasks both *Marginalize* (our method) and *Sample New* outperforms *Sample All* (method used in Riquelme et al. (2018)) in terms of cumulative regret. Both *Marginalize* and *Sample New* techniques are very similar in performance for the three datasets. All the three datasets used in this experiment are low dimensional, static, and relatively easy to learn, hence there is not much history to retain for *Sample New* technique. In the next section we will present results on larger datasets and also evaluate where we will show that our method performs better than *Sample New*.

5.3 Paraphrasing Bandit Task

Next we evaluate our method with a bigger and more complex underlying model on the NLP domain. We selected Bilateral Multi-Perspective Matching (BiMPM) (Wang et al., 2017), a recurrent model that performs well on several sentence matching tasks, including the paraphrase identification task, to evaluate our method. The goal of the paraphrase identification task is to determine whether a sentence is a paraphrase of another sentence, i.e., whether they convey the same meaning.

5.3.1 Dataset and bandit setting

To evaluate whether our algorithm is robust to shifts in data distribution we combined two different paraphrase identification datasets: i) The Quora Question Pairs dataset (**Quora**),[6] which contains 400,000 question pairs from the QA website Quora.com, and ii) The MSR Paraphrase Corpus (**MSR**) (Dolan et al., 2004), which contains 5,800 pairs of sentences extracted from news articles. To create an online training dataset we concatenate the MSR training set to a sample of 10,000 examples from the Quora training dataset[7].

We run the online algorithms on this dataset to report the regret values, while we report the offline performance on the MSR and Quora test sets. We use UCB as our search strategy, as it performs similarly to Thompson sampling and runs much faster in our implementation. We analyze the following two bandit tasks:

Multiclass. Like the previous datasets, we create a bandit problem by treating each class as an arm parameterized by a vector and the contexts as the individual data instances. A reward 1 is awarded

[6]https://data.quora.com/First-Quora-Dataset-ReleaseQuestion-Pairs

[7]The Quora dataset is subsampled so that the combined dataset is more balanced in terms of the number of examples from each dataset.

Task	Method	Average Cum. Regret	Offline Quora Acc.	Offline MSR Acc.
Multiclass	Marginalize	0.304 ±0.003	74.06 ±0.18	71.16 ±0.18
	Sample New	0.322 ±0.002	67.73 ±0.23	70.93 ±0.35
	Sample All	0.329 ±0.003	71.06 ±0.32	68.76 ±0.23
Pool	Marginalize	0.250 ±0.003	72.23 ±0.24	69.16 ±0.07
	Sample New	0.275 ±0.004	66.43 ±0.20	66.27 ±0.15
	Sample All	0.312 ±0.004	72.20 ±0.25	64.79 ±0.14
-	Batch Train	-	74.87 ±0.33	72.63 ±0.19

Table 2: Paraphrase Bandit Task results using the Quora/MSR paraphrase datasets.

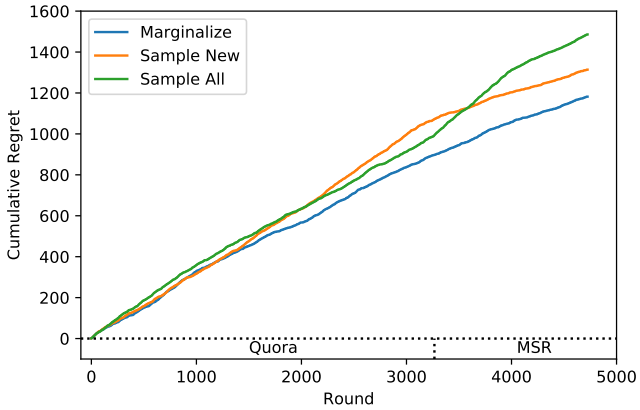

Figure 1: Cumulative regret of the methods on the pool based Paraphrased Bandit Task.

for identifying correctly if the two sentences in the pair are paraphrase. For each method, we perform an offline retraining after 1,000 online rounds.

Pool. Like the multiclass task, the pool based task occurs over a series of rounds. On each round, the model receives a pool of k(=3) instances, and must select one of them for the user. After that the model receives a reward based on its selection. The goal of the model is to learn a scoring function that predicts the expected reward for selecting a certain instance, while at the same time trying to keep regret low. This setting can be seen as an instance of the bandit problem formulation described in (Russo & Van Roy, 2014). In our case, our instances are candidate paraphrase pairs, where the model gets a reward of 1 for returning a valid paraphrase pair, and 0 otherwise. We use the same implementation and hyperparameters for BiMPM as in (Wang et al., 2017). For *Marginalize* and *Sample All*, we perform the retraining every 500 rounds. *Sample New* performed poorly offline on this setting and is thus updated every 1,000 rounds.

5.3.2 Results and Discussion

In Table 2 we show that our method *Marginalize* outperforms both *Sample All* and *Sample New* techniques for both multiclass and pool based tasks. *Sample All* and *Sample New* have comparable cumulative regret. *Sample New* has worse offline accuracy on Quora dataset (because it forgets old information), while it has better offline accuracy on MSR (because it is able to adapt quicker). For *Batch train*, both multiclass and pool based tasks are same—a binary classification problem. *Batch train* performs only slightly better than our method in terms of offline accuracy, where *Batch train* gets full feedback, while our method only gets partial (bandit) feedback. Figure 1 further shows that when the data distribution changes (switching form Quora to MSR in the pool based task) *Marginalize* and *Sample New* are able to adapt much faster than *Sample All*. Overall *Marginalize* achieved a lower regret as well as higher offline accuracy for both the bandit settings.

5.4 Image Classification Bandit Task

We additionally evaluate our method using a convolutional neural network, which is a common network architecture for computer vision applications.

Task	Method	Average Cum. Regret	Offline Accuracy
Multiclass	Marginalize	0.477 $\pm$0.002	59.87 $\pm$0.33
	Sample New	0.487 $\pm$0.003	60.01 $\pm$0.70
	Sample All	0.537 $\pm$0.002	55.21 $\pm$0.41
	Batch Train	-	78.20 $\pm$0.37
Pool	Marginalize	0.205 $\pm$0.001	84.64 $\pm$0.37
	Sample New	0.255 $\pm$0.003	82.55 $\pm$0.19
	Sample All	0.214 $\pm$0.009	85.11 $\pm$0.23
	Batch Train	-	87.97 $\pm$0.30

Table 3: Image classification Bandit Task results on CIFAR-10 dataset.

We use CIFAR-10 dataset (Krizhevsky, 2009) for this experiment. It is a commonly used dataset for image classification task consisting of 60,000 images from 10 different classes. Similar to the Quora/MSR tasks, we simulate a domain shift by concatenating together two datasets. In this case we create two data sets from CIFAR-10 by partitioning the dataset into images depicting animals (6 classes) and images depicting transportation (4 labels). As above, we analyze two bandit tasks:

Multiclass. We define the multiclass bandit task similarly as above, for each of the 10 classes in CIFAR-10, we assign one arm. At each round, the model receives an image, guesses a class, and receives feedback (0 or 1) for this class only. The task is considerably more difficult than the multiclass paraphrase bandit due to the number of classes. We use 1,000 rounds for the retraining frequency for all methods.

Pool. We also define a pool based bandit, similar to the pool based paraphrase bandit, with pool size $k = 5$. In this case we turn CIFAR-10 info a binary classification task. We select the two most difficult classes to classify (airplanes and birds, according to the confusion matrix of our base CNN model) in CIFAR-10, and denote these as the *positive* class. Like the previous pool task, the learner receives a pool of images and must select one. A reward of 1 is given for selecting an image belonging to the positive class, 0 otherwise. As done in the previous pool task, the data is sorted as to simulate a change in domain. We use a standard convolutional neural network architecture for this task, detailed in Appendix 7.5. We use 500 rounds for the retraining frequency for all methods.

5.4.1 Results and discussion

In Table 3 we present results for the image classification bandit task, using average cumulative regret and offline accuracy as evaluation metrics. Again, *Sample New* performs better than *Sample All* for cumulative regret but under-performs in the offline setting. As expected, our method performs well for both cumulative regret and offline setting. For the multiclass task, our method performs significantly lower than *batch train*. This is not too surprising, for two reasons: i) Training a CNN architecture takes many more epochs over the data to converge (~ 20 in our case) which is not achieved in a bandit setting; ii) CIFAR-10 has 10 classes, each defining an arm and in our setting; the bandit algorithms only gets feedback for one class in each round, compared to the full feedback received in *batch train*. Effectively, the number of labels per class in cut by a factor of 10. This is not as much an issue in the pool task, where we can see the results between *batch train* and the bandit algorithms are comparable.

6 Conclusion

In this paper we proposed a new method for training neural networks in a bandit setting. We tackle the problem of exploration-exploitation by estimating uncertainty only in the last layer, allowing the method to scale to large state-of-the-art models. We take advantage of having a posterior over the last layer weights by optimizing the rest of the network over *all values of the last layer*. We show that method outperforms other methods across a diverse set of underlying models, especially in online tasks where the distribution shifts rapidly. We leave it as future work to investigate more sophisticated methods for determining when to retrain the network, how to set the weight (β) of the regularization term in a more automatic way, and its possible connections to methods used for continual learning.

REFERENCES

Shipra Agrawal and Navin Goyal. Analysis of thompson sampling for the multi-armed bandit problem. In *COLT 2012 - The 25th Annual Conference on Learning Theory*, 2012.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 2002.

Kamyar Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. Efficient exploration through bayesian deep q-networks. 2018. URL http://arxiv.org/abs/1802.04412v1.

Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In *Advances in Neural Information Processing Systems*, 2011.

Bill Dolan, Chris Quirk, and Chris Brockett. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *Proceedings of the 20th international conference on Computational Linguistics*. Association for Computational Linguistics, 2004.

Kenneth Y. Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 2001.

Thorsten Joachims, Adith Swaminathan, and Maarten de Rijke. Deep learning with logged bandit feedback. *ICLR*, 2018.

Sham M. Kakade, Shai Shalev-Shwartz, and Ambuj Tewari. Efficient bandit algorithms for online multiclass prediction. In *Proceedings of the Twenty-Fifth International Conference*. ACM, 2008.

James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *CoRR*, 2016. URL http://arxiv.org/abs/1612.00796.

Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Master's thesis, 2009. URL http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

John Langford and Tong Zhang. The epoch-greedy algorithm for multi-armed bandits with side information. In *Advances in Neural Information Processing Systems*, 2007.

Miguel Lázaro-Gredilla and Aníbal R Figueiras-Vidal. Marginalized neural network mixtures for large-scale regression. *IEEE transactions on neural networks*, 2010.

Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, 2010.

Bing Liu, Tong Yu, Ian Lane, and Ole J. Mengshoel. Customized nonlinear bandits for online response selection in neural conversation models. *AAAI*, 2018. URL http://arxiv.org/abs/1711.08493v1.

David JC MacKay. The evidence framework applied to classification networks. *Neural computation*, 1992.

Dhruv Kumar Mahajan, Rajeev Rastogi, Charu Tiwari, and Adway Mitra. Logucb: an explore-exploit algorithm for comments recommendation. In *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 2012.

Brendan O'Donoghue, Ian Osband, Rémi Munos, and Volodymyr Mnih. The uncertainty bellman equation and exploration. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Carlos Riquelme, George Tucker, and Jasper Snoek. Deep bayesian bandits showdown: An empirical comparison of bayesian deep networks for thompson sampling. *ICLR*, 2018.

Hippolyt Ritter, Aleksandar Botev, and David Barber. Online structured laplace approximations for overcoming catastrophic forgetting. *NIPS*, 2018. URL http://arxiv.org/abs/1805.07810v1.

Daniel Russo and Benjamin Van Roy. Learning to optimize via posterior sampling. *Mathematics of Operations Research*, 2014.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Prabhat, and Ryan P. Adams. Scalable bayesian optimization using deep neural networks. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

Alex Strehl, John Langford, Lihong Li, and Sham M Kakade. Learning from logged implicit exploration data. In *Advances in Neural Information Processing Systems*, pp. 2217–2225, 2010.

Yee W Teh, David Newman, and Max Welling. A collapsed variational bayesian inference algorithm for latent dirichlet allocation. In *Advances in neural information processing systems*, 2007.

Chih-Chun Wang, Sanjeev R. Kulkarni, and H. Vincent Poor. Bandit problems with side observations. *IEEE Transactions on Automatic Control*, 2005.

Zhiguo Wang, Wael Hamza, and Radu Florian. Bilateral multi-perspective matching for natural language sentences. 2017. URL http://arxiv.org/abs/1702.03814v3.

7 APPENDIX

7.1 OBJECTIVE FUNCTION DERIVATION

We utilize the same notation here as in Section 3. First, we show how to arrive at equation (3) from the main objective, equation (1).

$$\begin{aligned}\mathbb{E}_{z\sim Q}[\log p(D_t, D_{t-1}, z|\theta)] &= \mathbb{E}_{z\sim Q}[\log p(D_t|z,\theta)] \\ &+ \mathbb{E}_{z\sim Q}[\log p(z|D_{t-1},\theta)] \\ &+ \mathbb{E}_{z\sim Q}[\log p(D_{t-1}|\theta)]\end{aligned}$$

By marginalizing over our zero mean Gaussian prior and using equation (2), it follows that the last expectation is approximately a constant, and can be removed. The second expectation is equal to the negative cross entropy between $Q = p(z|D_t, D_{t-1}, \theta_{t-1})$ and $P_\theta = p(z|D_{t-1}, \theta)$. Using the fact that $KL(Q||P_\theta) = CE(Q||P_\theta) - H(Q)$, we can replace the negative cross entropy with $-KL(Q||P_\theta) - H(Q)$, where $H(Q)$ is the entropy of Q which is constant and can be ignored, yielding equation (3).

We next derive the KL term. As mentioned, we approximate the KL term locally around θ_{t-1} with a second order Taylor expansion. Again, let $K(\theta) = KL(p(z|D_t, D_{t-1}, \theta_{t-1})||p(z|D_{t-1}, \theta))$. The second order Taylor expansion around $\theta = \theta_{t-1}$ is:

$$K(\theta) \approx K(\theta_{t-1}) + K'(\theta_{t-1})(\theta - \theta_{t-1}) + \frac{1}{2}(\theta - \theta_{t-1})K''(\theta_{t-1})(\theta - \theta_{t-1})^T$$

We can rewrite $K'(\theta)$ with respect to θ as:

$$\nabla KL(Q||P_\theta) = \nabla \int (Q\log Q - Q\log P_\theta)dz = \int \nabla Q \log P_\theta dz = \mathbb{E}_{z\sim Q}[\nabla \log p(z|D_{t-1},\theta)]$$

This can also be done for K''. Let ∇^i indicate either the gradient ($i = 1$) or the Hessian ($i = 2$). Then we can rewrite the above expectation with respect to the distribution $P = p(z|D_{t-1}, \theta_{t-1})$ instead:

$$\mathbb{E}_{z\sim Q}[\nabla^i \log P_\theta] = \mathbb{E}_{z\sim P}\Big[\frac{p(z|D_t, D_{t-1}, \theta_{t-1})}{p(z|D_{t-1}, \theta_{t-1})}\nabla^i \log P_\theta\Big]$$

Using the fact that $p(z|D_t, D_{t-1}, \theta_{t-1}) = p(D_t|\theta_{t-1})^{-1}p(D_t|z, \theta_{t-1})p(z|D_{t-1}, \theta_{t-1})$, we can rewrite the above as:

$$\mathbb{E}_{z\sim Q}[\nabla^i \log P_\theta] = p(D_{t-1}|\theta_t)^{-1}\mathbb{E}_{z\sim P}[p(D_t|z, \theta_{t-1})\nabla^i \log P_\theta]$$

Since $p(z|\theta)$ is assumed constant with respect to θ, and $p(D_{t-1}|\theta)$ is approximately constant (using the same trick with equation (2) previously used), we can rewrite $\nabla^i \log P_\theta$ as:

$$\nabla^i \log P_\theta = \nabla^i \log p(D_{t-1}|z,\theta) + \nabla^i \log p(z|\theta) - \nabla^i \log p(D_{t-1}|\theta) \approx \nabla^i \log p(D_{t-1}|z,\theta)$$

With this is mind it follows from the conditional independence of the data that:

$$\begin{aligned}\mathbb{E}_{z\sim Q}[\nabla^i \log P_\theta] &= p(D_t|\theta_{t-1})^{-1}\mathbb{E}_{z\sim P}[p(D_t|z,\theta_{t-1})\nabla^i \log p(D_{t-1}|z,\theta)] \\ &= p(D_t|\theta_{t-1})^{-1}\mathbb{E}_{z\sim P}[p(D_t|z,\theta_{t-1})]\mathbb{E}_{z\sim P}[\nabla^i \log p(D_{t-1}|z,\theta)] \\ &= p(D_t|\theta_{t-1})^{-1}\mathbb{E}_{z\sim P}[p(D_t|z,\theta_{t-1})]\mathbb{E}_{z\sim P}[\nabla^i \log P_\theta]\end{aligned}$$

If we plug in θ_{t-1} into $K'(\theta)$, the rightmost expectation in the expression above will be:

$$\mathbb{E}_{z\sim P}[\nabla \log p(z|D_{t-1},\theta_{t-1})] = \mathbb{E}_{z\sim P}[\nabla \log P] = 0$$

Thus, $K'(\theta_{t-1}) = 0$. If we plug in θ_{t-1} into $K''(\theta)$, the rightmost expectation in the expression above will be:

$$\mathbb{E}_{z\sim P}[\nabla\nabla \log p(z|D_{t-1},\theta_{t-1})] = \mathbb{E}_{z\sim P}[\nabla\nabla \log P] = F_P$$

Where F_P is the Fisher Information Matrix. Getting rid of constants, we can write the local KL approximation (when θ is close to θ_{t-1}) as equation (6):

$$KL[Q, p(z|D_{t-1},\theta)] \approx \frac{1}{2}(\theta - \theta_{t-1})\beta F_P(\theta - \theta_{t-1})^T$$

where $\beta = \frac{\mathbb{E}_{z\sim P}[p(D_t|z,\theta_{t-1})]}{p(D_t|\theta_t)}$. Replacing above approximation in equation (3) yields the following equation (7), which is our final objective for optimization:

$$\mathbb{E}_{z\sim Q}[\log p(D_t, D_{t-1}, z|\theta)] \approx \mathbb{E}_{z\sim Q}[\log p(D_t|z,\theta)] - \frac{1}{2}(\theta - \theta_{t-1})\beta F_P(\theta - \theta_{t-1})^T$$

7.2 Practical Considerations

Maintaining the last layer distribution. In the ideal situation we would like the posterior of the last layer to be conditioned on *all* currently seen data. However, for any BLL method, whenever the parameters θ change we must recompute the last layer distribution before going online again if we wish to keep it conditioned on previous data. This is done by doing a forward pass through the collected data and computing a regression on the output. For practical reasons, we keep a finite buffer of the past n instances and only recompute the distribution conditioned on these points. For all experiments and methods, we set $n = 20000$.

Computing the fisher information. Computing the Fisher information matrix in equation (7) requires computing the log likelihood gradient for previously collected data. As above, we only compute the gradients for the past n instances (where n is the same number as above). For efficiency we only compute the diagonal elements of the matrix. As we already performed a forward pass to recompute the last layer distribution, this calculation only requires a backwards pass. Fortunately, this backwards pass can be done in parallel with the online learning stage and thus does not bottleneck the process.

Bayesian linear versus Bayesian logistic regression. Either a Bayesian online logistic or linear regression may be used to model the posterior over the last layers. While a logistic regression is a more natural choice for classification tasks, the online Bayesian linear regression is much more efficient (as it essentially only involves rank one matrix updates). We found that the performance between the two is similar, and hence in our implementation, we use a linear regression. Rather than regressing on 0-1 labels, we regress on logit values of 3 (for positive classes, corresponding to a target probability of 0.95) and -3 (for negative classes, corresponding to a target probability of 0.05).

7.3 Calculating the UCB

Our approach to UCB is essentially the same as that presented by Li et al. (2010), that is, we essentially run the algorithm LinUCB on top of the last layer of the neural network. Calculating

the UCB is straightforward; if Σ is the covariance of our posterior over z, and x is the context, then with probability at least $1-\delta$, the term $(1+\sqrt{ln(2/\delta)/2})\sqrt{f_\theta(x)^T \Sigma f_\theta(x)}$ is an upper confidence bound. The term $\alpha = (1+\sqrt{ln(2/\delta)/2})$ is treated as a hyperparameter. The arm c chosen is the one whose parameter vector z_c maximizes the following:

$$f_\theta(x)^T z_c + \alpha\sqrt{f_\theta(x)^T \Sigma_c f_\theta(x)}$$

7.4 HYPERPARAMETERS

The hyperparameter values that are used for all methods across all tasks (the global hyperparameters) are presented in Table 4. The hyper parameters for the low dimensional bandit tasks (we uses the same values for each low dimensional dataset), the paraphrase bandit (multiclass and pool), and the image classification bandit (multiclass and pool) are presented in Table 5. The meanings of the non obvious hyperparameter values are described below:

Retrain Epochs: For Sample New and Marginalize; how many times to pass over the new data D_t.

Update Frequency: How many online rounds to do before updating the rest of the network offline.

Num Updates: For Sample All; the number of batches to uniformly sample (and update with) from the entire distribution.

Hyperparameter	Value	Description
Optimizer	RMSProp (lr=0.0001, decay=0.95)	Optimizer and learning rate
UCB α	2.07 (80% confidence interval)	Constant used in LinUCB
Round Robin Init	50 per arm	Number of times each arm was pulled for init

Table 4: Global (used for all methods) experiment details for low dimensional bandit task.

Task	Method	Batch Size	Retrain Epochs	Update Frequency	Num Updates
Low Dimensional	Marginalize	5	5	1000	-
Low Dimensional	Sample New	32	4	1000	-
Low Dimensional	Sample All	128	-	100	100
Paraphrase Multiclass	Marginalize	5	5	1000	-
Paraphrase Multiclass	Sample New	8	3	1000	-
Paraphrase Multiclass	Sample All	64	-	1000	1000
Paraphrase Pool	Marginalize	5	5	500	-
Paraphrase Pool	Sample New	8	3	1000	-
Paraphrase Pool	Sample All	64	-	500	500
Images Multiclass	Marginalize	5	5	1000	-
Images Multiclass	Sample New	16	4	1000	-
Images Multiclass	Sample All	64	-	1000	1000
Images Pool	Marginalize	5	5	500	-
Images Pool	Sample New	16	4	500	-
Images Pool	Sample All	64	-	500	500

Table 5: Method specific experiment details for all tasks.

7.5 MODEL ARCHITECTURES

In this section we detail the architectures used for each task. We use the same underlying model for each method.

Low dimensional task models. For the Low dimensional tasks, we utilize the same exact Multi Layer Perceptron (MLP) models as used in Riquelme et al. (2018). The model is a 2 layer MLP with a hidden layer dimension of 100, with ReLu activation functions.

Paraphrase task. The paraphrase task uses the same BiMPM architecture and parameters used in Wang et al. (2017). We refer readers to the corresponding paper for details.

Image classification task. The convolutions architecture we use is defined as: (i) Two 3×3 convolutional layers with 64 filters and ReLu activations; (ii) A 2×2 max pooling layer with a

stride of 2; (iii) Dropout layer with drop probability 0.25; (iv) A 3×3 and 2×2 convolutional layer both with 128 filters and ReLu activations; (v) A 2×2 max pooling layer with a stride of 2; (vi) Dropout layer with drop probability 0.25; (vii) A fully connected layer with 1024 units, followed by a tanh activation, followed by another fully connected layer with 100 units and a tanh activation.

We utilize tanh activations at the end as per Snoek et al. (2015), who note that ReLu activations lead to difficulties in estimating the uncertainty.