Project Report: BanditNet

Diwen Lu (dl2309), Zian Jiang (zj444), Lizhong Wang (lw2350)

May 2021

1 Introduction

In this project, we plan to reproduce the results from the paper [2] and extend the proposed method with a new experiment suggested from the paper. The paper proposes a technique that fits the contextual bandit problem, which is very common in search engine and recommendation system applications, into a deep learning setting.

Given logged contextual bandit feedback from policy π_0, the paper proposes a counterfactual risk minimization approach for training neural networks using the self-normalized inverse propensity scoring (SNIPS) estimator, which is referred to in lecture as the importance weighted (IW) estimator. According to the paper, the SNIPS estimator does not permit stochastic gradient descent (SGD) optimization in its given form, as we are not able to get an unbiased estimate of the gradient, which is an obstacle in efficiently training a neural network.

In this project, we will introduce two separate approaches to this problem and compare their performance. On one hand, we will follow the proposal from the paper which performs a reformulation of the SNIPS estimator into a series of constrained optimization problems such that it retains both the desired properties of the SNIPS estimator and the ability to use SGD for training. On the other hand, we suspect that even though for the SNIPS estimator an unbiased estimate of the gradient is not available, for a large enough batch sizes the bias may be relatively small and tolerable. In other words, we will experiment performing SGD with the SNIPS objective function using various minibatch sizes and compare its performance with BanditNet.

2 Main Idea of BanditNet and Our Approaches

2.1 Problem Formulation

Consider the contextual bandit setting where a policy π takes an input $x \in \mathcal{X}$ and outputs an action $y \in \mathcal{Y}$. Then we observe risk (or reward) $\delta(x, y) : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$. Note that we do not observe $\delta(x, y'), \forall y' \notin Y$.

To reformulate the above in a deep learning setting, we can view the stochastic policy π as a neural network π_w, where w is the weights of the network and $\pi_w(Y|x)$ is a conditional probability distribution (such as a neural network with a softmax output layer) over all possible actions $y \in \mathcal{Y}$. For example, $\pi_w(y|x)$ is the probability of choosing action y given context x.

Now given the logging policy π_0, we can incorporate the above setup into our setup from the lecture and formulate the logged contextual bandit feedback D, a collection of n tuples of observed context $x_i \sim Pr(X)$, action $y_i \sim \pi_0(Y|x_i)$ chosen by the logging policy π_0, propensity score $p_i \coloneqq \pi_w(y_i|x_i)$, and risk (or reward) $\delta_i(x_i, y_i)$:

$$D = [(x_1, y_1, p_1, \delta_1), ..., (x_n, y_n, p_n, \delta_n)]. \tag{1}$$

Now, we will discuss our batch training objective using our logged contextual bandit dataset D.

2.2 Equivariant Counterfactual Risk Minimization

The paper aims at solving a counterfactual risk minimization (CRM) problem, which can be estimated from our logged contextual bandit feedback D, but this suffers from two problems: 1) we do not know the rewards for actions that were not chosen, 2) D is biased towards π_0. As discussed in lecture, the inverse propensity scoring (IPS) estimator addresses them:

$$\hat{R}_{IPS}(\pi_w) = \frac{1}{n} \sum_{i=1}^{n} \delta(x_i, y_i) \frac{\pi_w(y_i \mid x_i)}{\pi_0(y_i \mid x_i)}. \tag{2}$$

As shown in the lecture, $\hat{R}_{IPS}$ is unbiased and has bounded variance given non-zero conditional probabilities for all actions. But from the lecture we know this estimator is prone to propensity overfitting, which is linked to its lack of equivariance.

Later in the lecture, we introduced the self-normalized IPS (SNIPS) estimator, which is immune to propensity overfitting and has negligible bias and lower variance than the IPS estimator:

$$\hat{R}_{SNIPS}(\pi_w) = \frac{\frac{1}{n} \sum_{i=1}^{n} \delta(x_i, y_i) \frac{\pi_w(y_i|x_i)}{\pi_0(y_i|x_i)}}{\frac{1}{n} \sum_{i=1}^{n} \frac{\pi_w(y_i|x_i)}{\pi_0(y_i|x_i)}} \tag{3}$$

To be consistent with the neural network setting, we will view δ as risk, as opposed to reward, and thus our training objective becomes

$$\hat{w} = \arg\min_{w} \hat{R}_{SNIPS}(\pi_w). \tag{4}$$

2.3 Training Algorithm and an Optimization Obstacle

As we can see in Eq.(3) and Eq.(4), batch training with stochastic gradient descent (SGD) is not permitted in its given form as we are optimizing a ratio

(see Appendix C of [2]). The paper remedies this problem with the following technique.

Intuitively, the denominator of Eq.(3), $S \coloneqq \frac{1}{n}\sum_{i=1}^{n}\frac{\pi_w(y_i|x_i)}{\pi_0(y_i|x_i)}$, is an one-dimensional quantity. We can perform grid search in sensible values $\{S_1, ..., S_k\}$ for the optimal S^* and instead for each S_j, solve

$$\hat{w}_j = \arg\min_w \frac{\frac{1}{n}\sum_{i=1}^{n}\delta(x_i, y_i)\frac{\pi_{w_j}(y_i|x_i)}{\pi_0(y_i|x_i)}}{S_j}. \tag{5}$$

This gives us a list of potential solution $\{\hat{w}_1, ..., \hat{w}_k\}$, and then we simply take the minimum out of them. However, this still leaves the question of how to solve each equality constrained risk minimization problem subject to $S_j = \frac{1}{n}\sum_{i=1}^{n}\frac{\pi_{w_j}(y_i|x_i)}{\pi_0(y_i|x_i)}$ using SGD. Instead, the paper proposes a workaround using its Lagrangian.

Consider the constrained optimization problem

$$\hat{w}_j = \arg\min_w \frac{1}{n}\sum_{i=1}^{n}\delta(x_i, y_i)\frac{\pi_{w_j}(y_i \mid x_i)}{\pi_0(y_i \mid x_i)} \text{ subject to } \frac{1}{n}\sum_{i=1}^{n}\frac{\pi_{w_j}(y_i \mid x_i)}{\pi_0(y_i \mid x_i)} = S_j. \tag{6}$$

Its Lagrangian can be written as

$$L(w, \lambda) = \frac{1}{n}\sum_{i=1}^{n}\frac{(\delta_i - \lambda)\pi_w(y_i \mid x_i)}{\pi_0(y_i \mid x_i)} + \lambda S_j. \tag{7}$$

Since we do not care about the solution to Eq.(6) for any specific S_j, instead merely exploring a sensible range of values for S_j, we can determine the value of S_j in hindsight such that for each $\lambda_j \in \{\lambda_1, ..., \lambda_k\}$, we fix λ_j and solve

$$\hat{w}_j = \arg\min_w L(w, \lambda_j). \tag{8}$$

Using the optimality conditions, we can get the corresponding $\hat{w}_j$ and S_j for a particular λ_j. Then we take the mininum among $\{\hat{w}_1, ..., \hat{w}_j\}$.

2.4 SGD with the SNIPS Objective Function Using Large Batch Sizes

Consider a supervised learning setting where we try to minimize $f(w) = \frac{1}{N}\sum_{i=1}^{N} f_i(w)$. According to SGD we can perform updates given a batch of samples B by $w \longleftarrow w - \alpha\frac{1}{|B|}\sum_{n\in B}\nabla f_n(w)$. The reason this update works is that the stochastic gradient, sampled as a batch, is an unbiased estimator of the full gradient. To see this,

$$E(\frac{1}{|B|}\sum_{n\in B}\nabla f_n(w)) = \frac{1}{N}\sum_{i=1}^{N}\nabla f_i(w) = \nabla f(w). \tag{9}$$

On the other hand, the gradient of the SNIPS estimator in Eq.(3) does not have an unbiased estimator (see Appendix C of [2]). However, it may be interesting to investigate whether the bias in the gradient may be relatively small when we have a large enough batch size. In other words, for each batch of samples B, we will take the gradient of

$$\hat{R}^B_{SNIPS}(\pi_w) = \frac{\frac{1}{|B|}\sum_{i\in B}\delta(x_i,y_i)\frac{\pi_w(y_i|x_i)}{\pi_0(y_i|x_i)}}{\frac{1}{|B|}\sum_{i\in B}\frac{\pi_w(y_i|x_i)}{\pi_0(y_i|x_i)}} \tag{10}$$

Using `PyTorch` and its `autograd` functionality, we are able to perform automatic differentiation of arbitrary scalar valued functions such as Eq.(10). Then we can perform back-propagation and optimize the SNIPS objective function, Eq.(4), directly and skip BanditNet. Even though this approach is largely empirical and not mathematically rigorous, we are interested to see how it performs with respect to various batch sizes and will later compare its performance with BanditNet.

3 Data and Methods

3.1 Dataset and Logged Bandit Feedback Simulation

We use the letter recognition dataset from Homework 3, which has $20,000$ images of 16 dimensions with 26 possible labels corresponding to the alphabet. We split $16,000$ examples for training and $4,000$ examples for testing. In order to evaluate the value of the policy, we keep the full reward information for the test set. Since reward is 1 for the correct label and 0 for the rest, under the full reward information every image's reward vector is essentially an one-hot vector.

To simulate logged bandit feedback under a static policy π_0, we reuse the policy code from Homework 3, which includes the uniform policy, Vlassis policy [4], and the scikit-learn policy. For example for the scikit-learn policy, we first pick a model from the scikit-learn package to use, such as Logistic Regression. Then we fit the model to the training set. Now for every image x_i with label y_i, we need to get the conditional probability distribution $\pi_0(A|x_i)$, which is a vector of dimension 26 that sums up to 1. In scikit-learn this can be calculated using the attribute `predict_proba()`. Then we sample action $a_i \sim \pi_0(A|x_i)$ with probability $p_i := \pi_0(a_i|x_i)$ and also set reward $\delta_i = 1$ if $a_i = y_i$, or 0 otherwise. Saving the above information as tuple, we arrive at a collection of n tuples of observed context $x_i \sim Pr(X)$, action $a_i \sim \pi_0(A|x_i)$, propensity score p_i, and reward $\delta_i(a_i, y_i)$:

$$D = [(x_1, a_1, p_1, \delta_1), ..., (x_n, a_n, p_n, \delta_n)]. \tag{11}$$

3.2 Training and Evaluation

For modeling, we use a simple neural network with one hidden layer with 8 neurons and rectified linear unit (ReLU) activation. For BanditNet, we largely

follow the implementation by [3]. We use $\lambda = 0.9$ for BanditNet.For SGD with the SNIPS objective training, since we want to experiment with various batch sizes, we create PyTorch data loaders with various batch sizes and train them separately and compare results. In particular, since PyTorch minimizes a cost function by default, we regard δ as risk instead of reward, whose connection is risk $= 1 -$ reward. Thus, in Section 4, the lower the training loss the better the result is.

For both approaches, we use the SGD optimizer with learning rate $\alpha = 0.1$ and train for 50 epochs. At the end of each epoch we calculate the value of policy π_w (expected average reward received for playing policy π_w) using the direct method with the full-bandit feedback from test set. Finally we save the model weights w whose associated policy π_w has the highest value estimate. Eq.(12) is the direct method to calculate the value of policy π_w given full-bandit feedback.

$$V(\pi_w) = \frac{1}{n} \sum_{i=1}^{n} \sum_{a=1}^{k} \delta(a, y_i) \pi_w(a|x_i). \tag{12}$$

All results are averaged over three runs.

4 Results

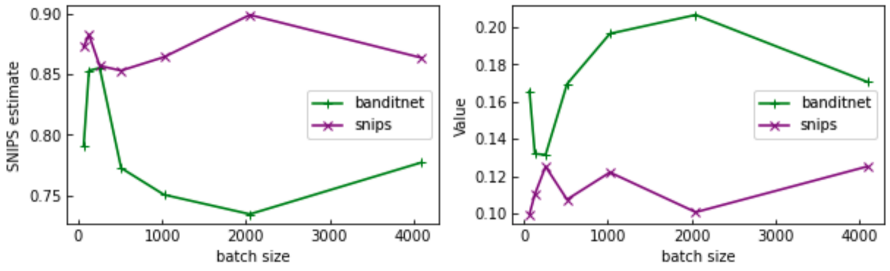

Figure 1: Left: SNIPS objective estimates using different batch sizes. Right: value estimate of policy π_w using direct method using different batch sizes.

Our hypothesis is that, if we train the SNIPS objective directly, with a large enough batch size, the bias in the gradient will be relatively small and training will potentially be stable enough so that BanditNet may not be necessary in practice. However, as we can see from Figure 1, BanditNet's value estimate is still better than that of direct SNIPS objective.

There are few other insights we can get from Figure 1. Firstly, we know that in training neural networks, batch size selection has an influence on speed and stability of the learning process. As we can see, it has a bigger influence on BanditNet than on direct SNIPS objective optimization. Also, recall that we choose the uniform policy as π_0, whose value estimate is only $\frac{1}{26} \approx 0.04$.

The best BanditNet model we have trained, with a batch size of 2048, is able to achieve 0.2 in value. This corresponds to a classification model with an accuracy of 20.6%, which is impressive considering that our training process involves no ground-truth labels at all. On the other hand, our best model trained under the SNIPS objective corresponds to a classification model with an accuracy of 17.0%. In the original paper [2], Bandit-ResNet is able to converge to the fully-supervised performance given enough bandit feedback. We believe the fact that pre-trained ResNet [1] is an expressive feature extractor explains why.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[2] Thorsten Joachims, Adith Swaminathan, and Maarten de Rijke. Deep learning with logged bandit feedback. In *International Conference on Learning Representations*, 2018.

[3] Noveen Sachdeva. Banditnet. `https://github.com/noveens/banditnet`, commit = 7f2c408b7e31943a3a59f1e5e44be1d8c5d365a7, 2019.

[4] Nikos Vlassis, Aurelien Bibaut, Maria Dimakopoulou, and Tony Jebara. On the design of estimators for bandit off-policy evaluation. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6468–6476. PMLR, 09–15 Jun 2019.