Contents

List of Tables

List of Figures

1 Introduction

Optimizing predictive models on datasets that are obtained from citizen-science projects can be computationally expensive as these datasets grow in size with time. Consequently, models based on multiple-layered neural networks, Integer Programming and other optimization routines can prove increasingly difficult as the number of parameters increase, despite using the faster Central Processing Units (CPUs) in the market. Incidentally, it becomes difficult for citizen-science projects to scale if the organizers use CPUs to run optimization models. However, Graphical Processing Units (GPUs), which offer multiple cores to parallelize computation, can outperform CPUs in computing such predictive models if these models heavily rely on large-scale matrix multiplications. By using GPUs over CPUs to accelerate computation on a citizen-science project, the model could achieve better optimization in less time, enabling the project to scale.

Part of the eBird project, which aims to "maximize the utility and accessibility of the vast numbers of bird observations made each year by recreational and professional bird watchers", Avicaching is a incentive-driven game trying to homogenize the spatial distribution of citizens' (agents') observations [cite website]. Since the dataset of agents' observations in eBird is geographically heterogeneous (concentrated in some places like cities and sparse in others), Avicaching homogenizes the observation set by rewarding agents who visit under-sampled locations [1]. To accomplish this task of specifying rewards at different locations based on the historical records of observations, Avicaching would learn the change in agents' behavior when a certain sample of rewards were applied to the set of locations, and then distribute a newer set of rewards across the locations based on those learned parameters [2]. This requirement naturally translates into a predictive optimization problem, which is implemented using multiple-layered neural networks and linear programming.

1.1 Computation Using GPUs

2 Problem Formulation

Since NVIDIA General Purpose GPUs enable faster computation on matrices, accelerated through CUDA and cuDNN, both the Identification (Section 2.1) and the Pricing Problem (Section 2.2) were formulated as 3-layered neural networks using the PyTorch library.

2.1 Identification Problem

As discussed in Section 1, the model should learn parameters that caused the change in agents' behavior when a certain set of rewards was applied to locations in the experiment region. Specifically, given datasets $\mathbf{y_t}$ and $\mathbf{x_t}$ of agents' visit densities with and without the rewards $\mathbf{r_t}$, we want to find weights $\mathbf{w_1}$ and $\mathbf{w_2}$ that caused the change from $\mathbf{x_t}$ to $\mathbf{y_t}$, factoring in possible influence from environmental factors $\mathbf{f}$ and distances between locations $\mathbf{D}$. Although the original model proposed to learn a single set of weights $\mathbf{w}$ [2], this proposed model considers two sets of weights $\mathbf{w_1}$ and $\mathbf{w_2}$ as it may theoretically result into higher accuracy and lower loss. Mathematically, the model can be formulated as:

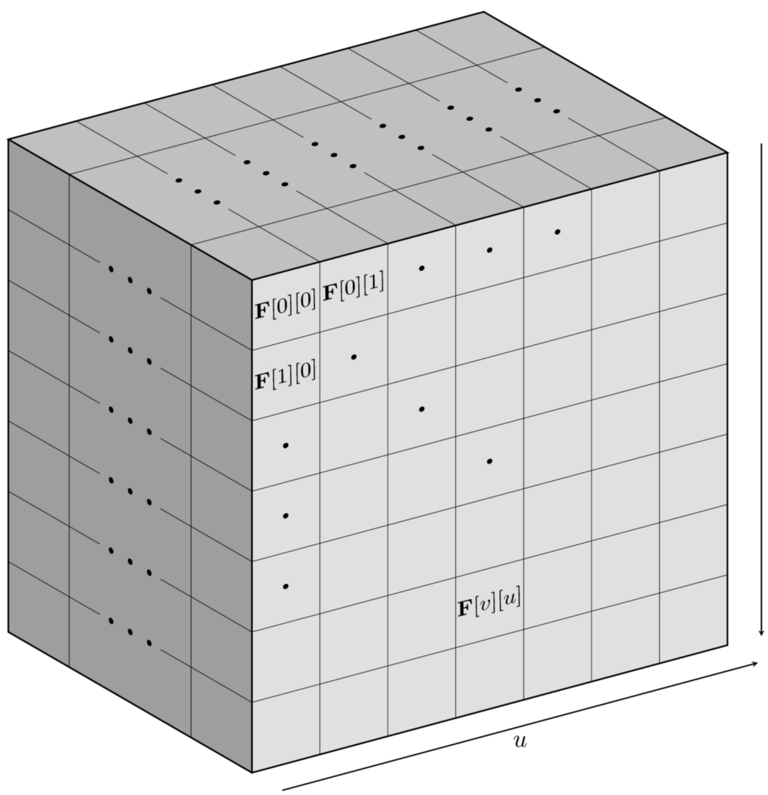
$$\underset{\mathbf{w}}{\text{minimize}} \quad Z_I(\mathbf{w_1}, \mathbf{w_2}) = \sum_t (\omega(\mathbf{y_t} - \mathbf{P}(\mathbf{f}, \mathbf{r_t}; \mathbf{w_1}, \mathbf{w_2})\mathbf{x_t}))^2 \tag{1}$$

where ω is a set of weights at time t capturing penalties relative to the importance of homogenizing at different locations and elements $p_{u,v}$ of $\mathbf{P}$ are given as:

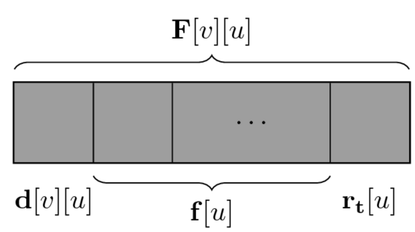
$$p_{u,v} = \frac{\exp(\mathbf{w_2} \cdot \text{reLU}(\mathbf{w_1} \cdot [d_{u,v}, \mathbf{f_u}, r_u]))}{\sum_{u'} \exp(\mathbf{w_2} \cdot \text{reLU}(\mathbf{w_1} \cdot [d_{u',v}, \mathbf{f_{u'}}, r_{u'}]))} = \frac{\exp(\Gamma_{u,v})}{\sum_{u'} \exp(\Gamma_{u',v})} = \text{softmax}(\Gamma_{u,v}) \quad (2)$$

In the expression for $p_{u,v}$ (Equation 2), softmax$(\cdot)$ is the function: $\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_i \exp(z_i)}$ and the reLU$(\cdot)$ function is a "rectified Linear Unit" defined as: $\text{reLU}(z) = \max(0, z)$.
To optimize the loss value $Z_I(\mathbf{w_1}, \mathbf{w_2})$ (Equation 1), the neural network learns the set of weights through multiple iterations of backpropagating the loss using gradient descent. Furthermore, the program processes the dataset before feeding to the network to avoid unnecessary sub-iterations and promote batch operations on matrices.

2.1.1 Structure of Input Dataset



(a) A Tensor representing the Input Dataset $\mathbf{F}$



(b) Contents of $\mathbf{F}[v][u]$

Figure 1: Visual representation of the Input Dataset

Since preprocessing the dataset impacts the efficiency of the network, the input dataset, comprising of distance between locations $\mathbf{D}$, environmental features $\mathbf{f}$ and given rewards $\mathbf{r_t}$

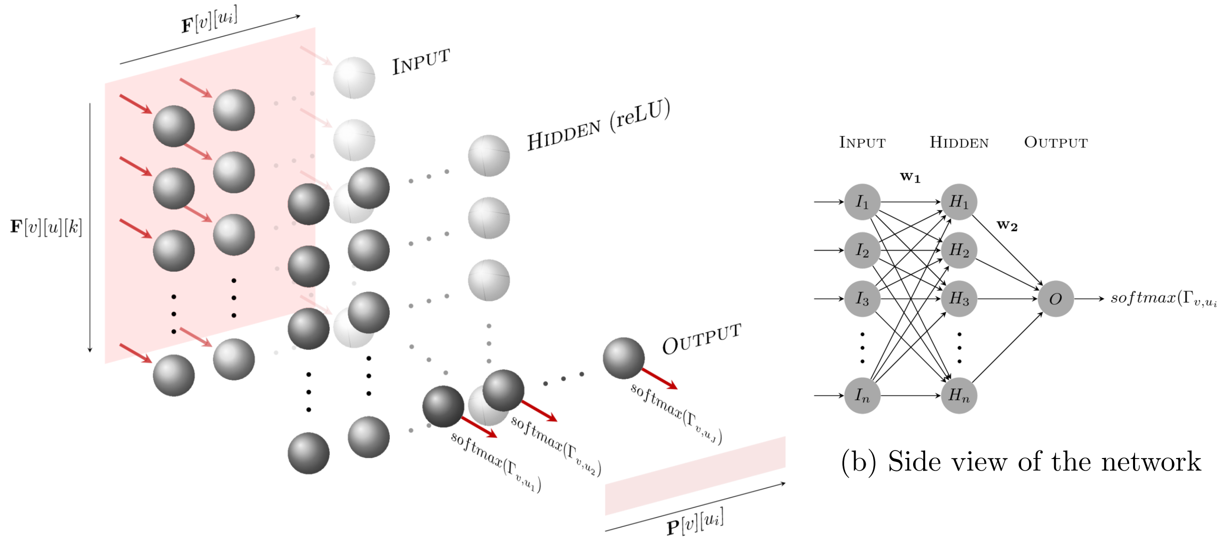
(all normalized) are combined in a specific manner. Since GPUs are efficient in operating on matrices and tensors, the input dataset is built into a tensor (Figure 1a) such that batch operations could be performed on slices $\mathbf{F}[v]$. Another advantage of building the dataset as a tensor comes with the PyTorch library, which provides convenient handling and transfer of tensors residing on CPUs and GPUs. Algorithm 1 describes the steps to construct this dataset.

Algorithm 1 Constructing the Input Dataset

function BUILD-DATASET($\mathbf{D}, \mathbf{f}, \mathbf{r_t}$)
    $\mathbf{D} \leftarrow$ NORMALIZE($\mathbf{D}$) $\triangleright$ $\mathbf{D}[u][v]$ is the distance between locations u and v
    $\mathbf{f} \leftarrow$ NORMALIZE($\mathbf{f}, axis = 0$) $\triangleright$ $\mathbf{f}[u]$ is a vector of env. features at location u
    $\mathbf{r_t} \leftarrow$ NORMALIZE($\mathbf{r_t}, axis = 0$) $\triangleright$ $\mathbf{r_t}[u]$ is the reward at location u
    for $v = 1, 2, \ldots, J$ **do**
        for $u = 1, 2, \ldots, J$ **do**
            $\mathbf{F}[v][u] \leftarrow [\mathbf{D}[v][u], \mathbf{f}[u], \mathbf{r_t}[u]]$ $\triangleright$ As depicted in Figure 1b
    return F

2.1.2 Algorithm for the Program

As shown in Figure 2, the neural network is made of 3 fully connected layers - the input layer, the hidden layer with rectified Linear Units (reLU), and the output layer generating the results using the softmax($\cdot$) function. The network can also be visualized as a collection of 1-dimensional layers (Figure 2b), with the softmax($\cdot$) calculated on the collection's output.



(a) 3-dimensional view of the network slice, taking in $\mathbf{F}[v]$

(b) Side view of the network

Figure 2: Neural network designed for the Identification Problem

It is important to clarify that the network in Figure 2a, which takes in $\mathbf{F}[v]$ as shown, is a slice of the original network, which takes in the complete tensor $\mathbf{F}$ and computes the complete result $\mathbf{P}^T$ per iteration of t. In other words, the input and the hidden layers are 3-dimensional, and the output layer is 2-dimensional. Since it is difficult to visualize the complete network on paper, slices of the network are depicted in Figure 2a. Algorithm 2 details the steps for learning the parameters $\mathbf{w_1}$ and $\mathbf{w_2}$ based on Equations 1 & 2.

Algorithm 2 Algorithm for the Identification Problem per iteration

---

Require: $\mathbf{w_1}, \mathbf{w_2}, T$

function IDENTIFY-WEIGHTS($\mathbf{x}, \mathbf{y}, \mathbf{r}, \mathbf{D}, \mathbf{f}, \omega$)

  for $t = 1, 2, \ldots, T$ **do**

    $\mathbf{F} \leftarrow$ BUILD-DATASET($\mathbf{D}, \mathbf{f}, \mathbf{r}[t]$) ▷ Defined in Algorithm 1

    $\mathbf{\Lambda} \leftarrow$ reLU(BATCH-MULTIPLY($\mathbf{F}, \mathbf{w_1}$)) ▷ **Phase 1: Feed Forward**

    $\mathbf{\Gamma} \leftarrow$ softmax(BATCH-MULTIPLY($\mathbf{\Lambda}, \mathbf{w_2}$))

    $\mathbf{P} \leftarrow \mathbf{\Gamma}^T$

    $loss \leftarrow loss + (\omega(\mathbf{y}[t] - \mathbf{P} \cdot \mathbf{x}[t]))^2$

  GRADIENT-DESCENT($loss, \mathbf{w_1}, \mathbf{w_2}$) ▷ **Phase 2: Backpropagate & Update**

  $\mathbf{w_1}, \mathbf{w_2} \leftarrow$ UPDATE-WEIGHTS($\mathbf{w_1}, \mathbf{w_2}$)

  return $loss$

2.2 Pricing Problem

After learning the set of weights $\mathbf{w_1}$ and $\mathbf{w_2}$ highlighting the change in agents' behavior to collect observations, the Pricing Problem aims to redistribute rewards to the all locations such that the predicted behavior of agents influenced by the new set of rewards is homogeneous. Thus, given a budget of rewards $\mathcal{R}$, this optimization problem can be expressed as:

$$\begin{aligned}
\underset{\mathbf{r}}{\text{minimize}} \quad & Z_P(\mathbf{r}) = \frac{1}{n}\|\mathbf{y} - \overline{\mathbf{y}}\| \\
\text{subject to} \quad & \mathbf{y} = \mathbf{P}(\mathbf{f}, \mathbf{r}; \mathbf{w_1}, \mathbf{w_2})\,\mathbf{x} \\
& \sum_i r_i \leq \mathcal{R} \\
& r_i \geq 0
\end{aligned} \tag{3}$$

where elements of $\mathbf{P}$ are defined as in Equation 2. To allocate the rewards $\mathbf{r}$ optimally, the pricing problem is formulated akin that for the Identification Problem (Section 2.1). In fact, the structure of the network and dataset remain similar to that of the Identification Problem (Sections 2.1.1 & 2.1.2), but the particulars are altered, as this routine would adjust the rewards based on the learned set of weights.

2.3 Input Dataset for Finding Rewards

3 Experiments

4 Results

5 Conclusion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

References

[1] Y. Xue, I. Davies, D. Fink, C. Wood, and C. P. Gomes, "Avicaching: A two stage game for bias reduction in citizen science," in *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, AAMAS '16, (Richland, SC), pp. 776–785, International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[2] Y. Xue, I. Davies, D. Fink, C. Wood, and C. P. Gomes, "Behavior identification in two-stage games for incentivizing citizen science exploration," in *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, pp. 701–717, 2016.