GPU-accelerated Principal-Agent Game for Scalable Citizen Science

Anmol Kabra
ak2426@cornell.edu
Department of Computer Science,
Cornell University

Yexiang Xue
yexiang@purdue.edu
Department of Computer Science,
Purdue University

Carla P. Gomes
gomes@cs.cornell.edu
Department of Computer Science,
Cornell University

ABSTRACT

Citizen science programs have been instrumental in boosting sustainability projects, large-scale scientific discovery, and crowdsourced experimentation. Nevertheless, these programs witness challenges in submissions' quality, such as sampling bias resulting from citizens' preferences to complete some tasks over others. The sampling bias frequently manifests itself in the program's dataset as spatially clustered submissions, which reduce the efficacy of the dataset for subsequent scientific studies. To address the spatial clustering problem, programs use reward schemes obtained from game-theoretical models to incentivize citizens to perform tasks that are more meaningful from a scientific point of view. Herein we propose a GPU-accelerated approach for the *Avicaching* game, which was recently introduced by the *eBird* citizen science program to incentivize birdwatchers to collect bird data from under-sampled locations. *Avicaching* is a Principal-Agent game, in which the principal corresponds to the citizen science program (*eBird*) and the agents to the birdwatchers or citizen scientists. Previous approaches for solving the *Avicaching* game used approximations based on mixed-integer programming and knapsack algorithms combined with learning algorithms, using standard CPU hardware. Following the recent advances in scalable deep learning and parallel computation on Graphical Processing Units (GPUs), we propose a novel approach to solve the *Avicaching* game, which takes advantage of neural networks and parallelism for large-scale games. We demonstrate that our approach better captures agents' behavior, which allows better learning and more effective incentive distribution in a real-world bird observation dataset. Our approach also allows for massive speedups using GPUs. As *Avicaching* is representative of games that are aimed at reducing spatial clustering in citizen science programs, our scalable reformulation for *Avicaching* enables citizen science programs to tackle sampling bias and improve submission quality on a large scale.

CCS CONCEPTS

• **Computing methodologies → Neural networks**; • **Applied computing → Decision analysis**.

KEYWORDS

Citizen Science, Computational Sustainability, Machine Learning, Mixed-Integer Programming, Parallel Computation

ACM Reference Format:
Anmol Kabra, Yexiang Xue, and Carla P. Gomes. 2019. GPU-accelerated Principal-Agent Game for Scalable Citizen Science. In *ACM SIGCAS Conference on Computing and Sustainable Societies (COMPASS) (COMPASS '19), July 3–5, 2019, Accra, Ghana.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3314344.3332495

1 INTRODUCTION

In the past decade, citizen science programs, such as *eBird* [16, 29], *Zooniverse* [19], and *CoralWatch* [20], have been successful in engaging the general public in collecting meaningful data to answer research questions. These programs allow researchers to not only outsource data collection and observation tasks to the public, but also educate the public about scientific methodologies and concepts [9]. Moreover, the extensive observational data collected in these programs enable researchers to expedite scientific discovery, support environmental conservation, and inform policy decisions on science and sustainability [10, 14, 21].

Although citizens assist in scientific discovery through large-scale data collection, they collect data according to their own motivations, rather than providing systematic observations that satisfy scientific requirements, which the researchers desire [1, 3, 5, 6, 21]. Citizens' motivations are influenced by various "task"-dependent factors, where a "task" can refer to one in many experiments in the program that citizens could complete, or one in many geographical sites that citizens may collect data at. The influence of these factors, which may include tasks' relative appeal compared to other tasks and citizens' personal preferences, introduces sampling bias in the datasets [3]. The bias often manifests itself in the form of undesirable spatio-temporal clustering of submitted observations, hurting the efficacy of dataset-based scientific models and subsequent environmental conservation efforts [12, 30].

This misalignment in the researchers' (principal's) and citizens' (agents') motivations also occurs in settings other than citizen science, such as game-theoretical economics [18], political science [11], and crowdsourcing [27]. Previous studies have compensated for the misalignment in these settings using Principal-agent games [8, 26] (also known as Stackelberg games [23]). In these games, the principal learns incentives, which could motivate agents to complete the most crucial tasks, by factoring in the agents' reasoning process.

As Tiago et al. conclude from their study on spatial bias in different citizen science programs, the programs must either passively compensate for the sampling bias (after data collection) or actively reduce sampling bias (during data collection) [30]. Principal-agent

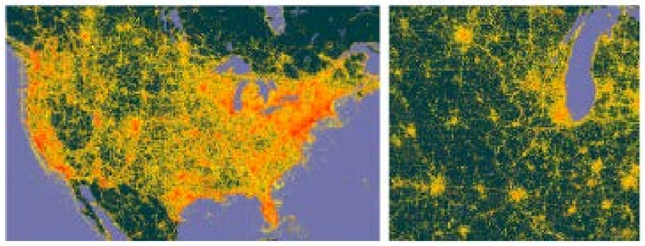


Figure 1: Spatial clustering in *eBird* submissions before 2014 in mainland USA (left) and Midwest USA (right) [31]. While population centers account for a large number of submissions (yellow points), other locations are under-sampled.

games are one of the most promising mechanisms using which citizen science programs can *proactively* reduce spatio-temporal clustering [6, 16, 31]. In addition to accurately modeling the principal and agent's motivations, solutions to these games must also be efficient and scalable to demonstrate their feasibility in large-scale citizen science programs.

For example, consider *eBird*—a bird observation dataset collected by citizen scientists and used for studying seasonal bird migration and changes in habitats [16, 29]. To combat spatial clustering of citizens' visits to birdwatching locations (Figure 1), *eBird* recently introduced the *Avicaching* game to reward citizen scientists for visiting previously under-sampled locations. By motivating citizens with rewards[1] to perform the most crucial tasks, *eBird* organizers witnessed a promising reduction in spatial clustering: $\approx 20\%$ of birdwatching effort shifted from popular to under-sampled locations during a 3-month pilot study in upstate New York, USA [31]. For allocating rewards in *Avicaching*, Xue et al. folded the agents' reasoning process (termed "identification problem") as linear constraints into the reward allocation scheme ("pricing problem"), which they solved using an off-the-shelf Mixed-Integer Programming (MIP) solver [32]. However, this MIP formulation scales poorly as the number of locations in the game increases, hampering the technique's use in large citizen science programs.

In this study, we propose a novel reformulation of the problem to solve large-scale games in citizen science using parallelizable machine learning architectures on GPUs, thus improving scalability and computational costs in these games. Our work thus enables citizen science programs to actively reduce spatial clustering, thus reducing sampling bias in large datasets and improving the efficacy of subsequent scientific modeling.

Our work is motivated by the recent uptick in Deep Learning's scalability and performance for optimization [25], and by principal-agent games' application in reducing sampling bias in citizen science. In addition to building a neural-network-based reformulation of these games to better learn agents' behavior and incentive distribution, we demonstrate how this reformulation yields to parallel computation on Graphical Processing Units (GPUs), which help us dramatically scale both the identification and pricing problems in *Avicaching*. Our reformulation for *Avicaching* also applies to games in other citizen science programs, since *Avicaching* is representative of principal-agent games in citizen science [32]. In particular, our contributions are threefold:

(1) We reformulate the canonical *Avicaching* principal-agent game, previously studied with MIP, with deep neural networks. We develop a novel scheme to learn agents' behavior and allocate rewards on the same neural network architecture, allowing us to solve the identification and pricing problems under the same framework.
(2) Our proposed principal-agent games' reformulation is scalable and practical for large-scale deployment, as we implement our framework using parallelizable tensor operations, enabling deep networks to efficiently train on GPUs.
(3) Finally, we evaluate our solvers on *eBird*, a real-life dataset in citizen science, and find that our framework performs slightly better on accuracy metrics while running several orders of magnitude faster than previous solvers.

Our neural-network-based techniques thus enable citizen science organizers to efficiently limit sampling bias in large datasets, ultimately aiding citizen-science-based scientific discovery.

The code is available at github.com/anmolkabra/avicaching.

2 REDUCING SPATIAL CLUSTERING

In this section we introduce the *Avicaching* game and review Xue et al.'s formulation of the game [31, 32]. We extend Xue et al.'s two-stage principal-agent game's formualation [32] in this study as they demonstrate its efficacy over models that used dynamic programming and knapsack-based approaches [31].

To effectively steer agents towards crucial tasks, the principal must factor in the agents' preferences when allocating rewards. This two-stage process is effectively captured with a bilevel optimization problem [7]. In such problems, the lower-level optimization problem of learning the agents' behavior, with respect to their intrinsic preferences and the principal's rewards, is embedded in the upper-level problem of distributing rewards to maximize the principal's utility, as summarized in Equation (1).

$$\begin{aligned}
\textbf{(Principal)}\ r^* &= \arg\max_r \quad U_p(v^*, r) \\
\text{subject to} &\quad B_p(r) \\
\textbf{(Agents)} &\quad v^* = \arg\max_v \quad U_a(v, r) \\
&\qquad\qquad \text{subject to} \quad B_a(v)
\end{aligned} \tag{1}$$

In this model, the principal wishes to maximize its utility U_p by introducing rewards r, subject to the reward constraints B_p and the agents' behavior v^*, which is the result of the sub-problem: agents' maximizing their utility U_a.

As devised in *Avicaching* for *eBird*, the principal can reward agents to visit under-sampled locations to increase the spatial homogeneity in visits, i.e., reduce spatial clustering. Algorithm 1 describes *eBird* organizers' strategy to achieve spatial homogeneity.

2.1 Identification Problem: Learning Agents' Behavior

We can extract the number of visits to a location u from the *eBird* dataset before and after a reward $r_{t,u}$ was applied during a time period t. Let the visit densities over n locations prior to the reward treatment be $\mathbf{x_t}$ and after the reward treatment be $\mathbf{y_t}$.

[1]The *Avicaching* rewards contributed to a citizen's position on a public leaderboard, which determined the type of award or gift the citizen received.

Algorithm 1 *eBird* organizers' strategy to optimize their utility [32] in Equation (1)

```
1: Deploy rewards and observe agents' behavior
2: while homogeneity not achieved do
3:     Understand agents' behavior via visit patterns (identification problem)
4:     Deploy new, learned rewards (pricing problem) and observe agents' behavior
```

Using Xue et al.'s formulation, learning how the agents' visits changed during period t is equivalent to learning a mapping $\mathbf{P} : \mathbf{x_t} \mapsto \mathbf{y_t}$ [32]. While $\mathbf{P}$ can be any mapping, we use Xue et al.'s assumption [32] that $\mathbf{P}$ is a matrix such that $\mathbf{P}\mathbf{x_t} \approx \mathbf{y_t}$ so as to use their models as our benchmarks. $\mathbf{P}$ is itself a function of location-specific features—rewards $\mathbf{r_t}$, pairwise distances of locations $\mathbf{D}$, and environmental features $\mathbf{f}$ like proximity to wetlands, brackish water bodies etc.—all weighted with learnable parameters $\mathbf{w}$.

Intuitively, the entry $p_{u,v}$ in matrix $\mathbf{P}$ is the probability of the agents' effort shifting from location v to u. This probability can be calculated from the environmental features $\mathbf{f_u}$ of location u, the reward $r_{t,u}$, and the distance of location u from v, all appropriately weighed by a function g with parameters $\mathbf{w}$. Xue et al. [32] formulate $p_{u,v}$ as the following, which we reinterpret as a softmax function [4] applied on probabilities $p_{u,v}$ with v fixed.

$$\begin{aligned} p_{u,v} &= \frac{\exp(g(\mathbf{f_u}, d_{u,v}, r_{t,u}; \mathbf{w}) + \eta \cdot \delta_{uv})}{\sum_{u'} \exp(g(\mathbf{f_{u'}}, d_{u',v}, r_{t,u'}; \mathbf{w}) + \eta \cdot \delta_{u'v})} \\ &= \text{softmax}_v(g + \eta e_v)[u] \end{aligned} \tag{2}$$

In Equation (2), δ_{ij} is the Kronecker delta function (1 if $i = j$ and 0 otherwise), e_v is the v^{th} standard basis vector in $\mathbb{R}^n$, and the hyperparameter η tunes $\mathbf{P}$ to look like the identity matrix when $g = 0$, i.e., when the features are uninformative and $\mathbf{y_t} = \mathbf{x_t}$.

Now, given a dataset $\mathcal{D} = \{(\mathbf{x_t}, \mathbf{y_t}, \mathbf{r_t})_t, \mathbf{f}, \mathbf{D}\}$, the identification problem corresponds to learning $\mathbf{w}^*$ such that $\mathbf{P}\mathbf{x_t} \approx \mathbf{y_t}$, as in the following equation [32]. We add an L2 regularizer to reduce overfitting during training.

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_t \left\| \omega_t (\mathbf{y_t} - \mathbf{P}_{\mathbf{f},\mathbf{D},\mathbf{r_t};\mathbf{w}} \mathbf{x_t}) \right\|_2^2 + \lambda \left\| \mathbf{w} \right\|_2^2 \tag{3}$$

Here, ω_t is the count of visits in period t, i.e., $\omega_t = \|\mathbf{y_t}\|_1$, meant to weigh periods with greater number of visits higher. We also emphasize using subscripts that the entries of $\mathbf{P}$ depend on $\mathbf{f}, \mathbf{D}, \mathbf{r_t}$, and are parameterized by $\mathbf{w}$. We now normalize $\mathbf{x_t}, \mathbf{y_t}$ such that $\forall t, \|\mathbf{x_t}\|_1 = \|\mathbf{y_t}\|_1 = 1$.

2.2 Pricing Problem: Distributing Rewards

Since the *eBird* organizers aim to reduce spatial clustering, the pricing problem finds rewards that minimize variance in predicted visit densities $\mathbf{y} = \mathbf{P}\mathbf{x}$ where $\mathbf{P}$ is evaluated at $\mathbf{w}^*$ and $\mathbf{x} = \sum_t \mathbf{x_t} / \|\sum_t \mathbf{x_t}\|_1$. For our objective function, we use the 1-norm variance, $\|\mathbf{y} - \overline{\mathbf{y}}\|_1$, rather than the 2-norm version, so as to compare our models' performance directly with that of Xue et al.'s MIP-based models [32]. Here, $\overline{\mathbf{y}} = \text{mean}(y_u)$. Additionally, the rewards must be non-negative, sum to at most budget $\mathcal{R}$, and can only be placed at *Avicaching* locations given by a binary vector $\mathbf{a}$ (rewards at 'non-*Avicaching* locations' must be 0). Since $a_u = 1$ if u is an *Avicaching* location and 0 otherwise, the last constraint is $\forall u, (1 - a_u) r_u = 0$. This optimization problem can be written as Equation (4).

$$\begin{aligned} \mathbf{r}^* = \arg\min_{\mathbf{r}} \quad & \frac{1}{n} \|\mathbf{y} - \overline{\mathbf{y}}\|_1 \\ \text{subject to} \quad & \mathbf{y} = \mathbf{P}_{\mathbf{f},\mathbf{D},\mathbf{w}^*;\mathbf{r}} \mathbf{x} \\ & \forall u, (1 - a_u) r_u = 0 \\ & \forall u, r_u \geq 0 \\ & \|\mathbf{r}\|_1 \leq \mathcal{R} \end{aligned} \tag{4}$$

2.3 Generalizing to Citizen Science Programs

This formulation of principal-agent games generalizes to other citizen science programs as well. In a program with n tasks where the principal has reward data $\mathbf{r_t}$, tasks' characteristics $\mathbf{f}$, and agents' submission data $\mathbf{x_t}, \mathbf{y_t}$, the principal can learn the parameters $\mathbf{w}$ in the linear mapping $\mathbf{P}$ with $p_{u,v}$ defined analogously. The pricing problem's formulation also generalizes as the constraints on rewards frequently appear in general incentive-based motivation schemes: avoid penalizing agents with negative rewards, distribute rewards up to a budget, and be selective of which tasks to reward.

3 SCALABLE PRINCIPAL-AGENT GAMES USING MACHINE LEARNING

Since the previous models for these games do not scale well with increasing number of tasks, we use machine learning and parallel computation to give us approximate models suitable for large-scale games. For instance, the MIP formulation for *Avicaching* is not practical on large-scale games because the MIP takes $\approx 10,000$ seconds for placing rewards on ≈ 30 locations with 33 features per location in *eBird* [32], which has millions of locations. By relaxing the integer constraints on rewards, we are able to develop a novel learning scheme to solve both the identification and pricing problem using the same neural network architecture, which can be efficiently scaled with batch-matrix operations on GPUs. This model thus improves the feasibility of principal-agent games to tackle spatial clustering in large citizen science programs.

3.1 Learning $\mathbf{w}^*$ and $\mathbf{r}^*$

3.1.1 Neural Network Architecture. For calculating $p_{u,v}$ in Equation (2), while Xue et al. choose g to be a linear combination of features $[\mathbf{f_u}, d_{u,v}, r_{t,u}]$ [32], we generalize g to be potentially non-linear. In this study, we use a sequence of linear combinations and activation functions for g, defined as the following:

$$g(\mathbf{f_u}, d_{u,v}, r_{t,u}) = \mathbf{w_{j-1}} \cdot a_{j-1}(\mathbf{w_{j-2}} \cdot \ldots \cdot a_1(\mathbf{w_1} \cdot [\mathbf{f_u}, d_{u,v}, r_{t,u}]))$$

where a_i are activation functions used in neural networks [4], and $\mathbf{w_i}$ are weights. This sequence of $2(j-1)$ operations can also be thought of as $j - 1$ layers in a neural network[2], followed by the softmax layer to get $p_{u,v}$, making a total of j layers. With this formulation, a 2-layer network will have an input layer, $\mathbf{w_1}$, and a softmax layer but no activation functions. A 3-layer network will have $\mathbf{w_2}$

[2]In the context of our neural-network-based models, objective functions of both problems are loss functions, and $\mathbf{w}$ is the set of all $\mathbf{w_i}$.

and an activation function in addition to the 2-layer network's units; the compositions of deeper networks follow similarly.

We build the input as follows: for location v, $\mathbf{F}[v]$ is a $n \times m$ matrix such that $\mathbf{F}[v][u]$ is $[\mathbf{f_u}, d_{u,v}, r_{t,u}]$, a vector of m features that could influence agents to move from location v to u (Algorithm 2). This makes $\mathbf{F}$ a $n \times n \times m$-size tensor.

Algorithm 2 Building F

```
1: function BUILD-DATASET($\mathbf{f}, \mathbf{D}, \mathbf{r_t}$)
2:     for $v = 1 : n$ do
3:         for $u = 1 : n$ do
4:             $\mathbf{F}[v][u] \leftarrow [\mathbf{f}[u], \mathbf{D}[v][u], \mathbf{r_t}[u]]$
5:     return F
```

The network now feeds features in $\mathbf{F}[v]$ forward, as pictured in Figure 2a and defined in Algorithm 3, to obtain $\mathbf{P}[v]$, a probability vector of agents moving from v to other locations. Given this estimate of $\mathbf{P}$, the network calculates the loss value and updates the parameters $\mathbf{w_i}$ using the backpropagation algorithm. As illustrated in Figure 2b, our network thus essentially becomes a sequence of batch-matrix operations that can be parallelized on GPUs [2, 25, 28]. To stabilize training of deeper networks, we batch-normalize tensors, obtained after applying activation functions a_i, by subtracting the tensor's mean and dividing by its standard deviation [15].

Algorithm 3 A forward pass in a 3-layer network

```
1: function FORWARD($\mathbf{f}, \mathbf{D}, \mathbf{r_t}, \mathbf{w_1}, \mathbf{w_2}, \mathbf{x_t}$)
2:     $\mathbf{F} \leftarrow$ BUILD-DATASET($\mathbf{f}, \mathbf{D}, \mathbf{r_t}$)
3:     $\mathbf{H} \leftarrow$ ReLU(BATCH-MULTIPLY($\mathbf{F}, \mathbf{w_1}$))
4:     $\mathbf{P}^T \leftarrow$ softmax(BATCH-MULTIPLY($\mathbf{H}, \mathbf{w_2}$))
5:     return $\mathbf{P}\mathbf{x_t}$
```

The learned weights $\mathbf{w_i^*}$ from the identification problem become inputs to the pricing problem network, which tunes the rewards $\mathbf{r}$ to minimize the objective (Equation (4)). After initializing $\mathbf{r}$ in the feasible region given by the constraints, the network repeatedly feeds $\mathbf{r}$ forward in the same network, calculates loss for the pricing problem, and updates $\mathbf{r}$ using gradient updates (see Figure 2c).

3.1.2 *Enforcing Constraints on Rewards.* Since gradient updates to $\mathbf{r}$ in the pricing problem will potentially violate the constraints in the pricing problem, we explicitly enforce constraints after updating rewards. In the *Avicaching* game, non-negativity and budget constraints are easy and fast to enforce—to efficiently project the updated $\mathbf{r}$ to the feasible space, we clip rewards at minimum 0, and re-normalize to satisfy $\|\mathbf{r}\|_1 = \mathcal{R}$.

Furthermore, backpropagation-based gradient updates enable us to reset rewards at non-*Avicaching* locations to 0 efficiently: since $\mathbf{r}$ is updated by gradient steps, we can initialize rewards at such locations to 0 and explicitly set the gradients of these rewards to 0 at each update step. This is possible only since ∇_{r_u}loss, calculated using chain rule, is independent of ∇_{r_v}loss if $u \neq v$. Consequently, rewards at non-*Avicaching* locations remain 0 during training.

Therefore, we can learn $\mathbf{w}^*$ and $\mathbf{r}^*$ using gradient descent updates on the same backpropagation framework and neural network architecture. Algorithm 4 describes the full learning pipeline, highlighting the two-stage process.

Algorithm 4 The complete 3-layer-network-based model

```
   ▷ $\mathbf{f} \in \mathbb{R}^{n \times m}; \mathbf{D} \in \mathbb{R}^{n \times n}; \forall t, \mathbf{x_t}, \mathbf{y_t}, \mathbf{r_t} \in \mathbb{R}^n$
1: ∀ tasks' characteristic $i$, $\mathbf{f}[:, i] \leftarrow$ NORMALIZE($\mathbf{f}[:, i]$)
2: $\mathbf{D} \leftarrow$ NORMALIZE($\mathbf{D}$)
3: $\mathbf{w_1} \leftarrow$ RANDOM( $(n, m, m)$ )          ▷ Identification Problem
4: $\mathbf{w_2} \leftarrow$ RANDOM( $(n, m, 1)$ )
5: for epoch = 1, 2, . . . do
6:     $loss \leftarrow 0$
7:     for $t = 1, 2, \ldots$ do
8:         $\hat{\mathbf{y}}_\mathbf{t} \leftarrow$ FORWARD($\mathbf{f}, \mathbf{D}, \mathbf{r_t}, \mathbf{w_1}, \mathbf{w_1}, \mathbf{x_t}$)
9:         loss ← loss + $(\omega[t](\mathbf{y_t} - \hat{\mathbf{y}}_\mathbf{t}))^2$
10:    GRADIENT-DESCENT(loss, $\mathbf{w_1}, \mathbf{w_2}$)
11:    $\mathbf{w_1}, \mathbf{w_2} \leftarrow$ UPDATE-USING-GRADIENTS($\mathbf{w_1}, \mathbf{w_2}$)
12: $\mathbf{x} \leftarrow$ NORMALIZE($\sum_t \mathbf{x_t}$)          ▷ Pricing Problem
13: $\mathbf{r} \leftarrow$ RANDOM( $(n)$ )
14: $\mathbf{r} \leftarrow$ APPLY-CONSTRAINTS($\mathbf{r}$)
15: $\mathbf{y} \leftarrow$ FORWARD($\mathbf{f}, \mathbf{D}, \mathbf{r_t}, \mathbf{w_1}, \mathbf{w_1}, \mathbf{x}$)
16: loss ← $\|\mathbf{y} - \overline{\mathbf{y}}\|_1 / n$
17: for epoch = 1, 2, . . . do
18:    GRADIENT-DESCENT(loss, $\mathbf{r}$)
19:    $\mathbf{r} \leftarrow$ UPDATE-USING-GRADIENTS($\mathbf{r}$)
20:    $\mathbf{r} \leftarrow$ APPLY-CONSTRAINTS($\mathbf{r}$)
21:    $\mathbf{y} \leftarrow$ FORWARD($\mathbf{f}, \mathbf{D}, \mathbf{r_t}, \mathbf{w_1}, \mathbf{w_1}, \mathbf{x}$)
22:    loss ← $\|\mathbf{y} - \overline{\mathbf{y}}\|_1 / n$
```

3.1.3 *Optimizations in Implementation.* Here, we discuss a couple of optimizations in implementations of the sub-problems that further reduce the computational cost of our methodology.

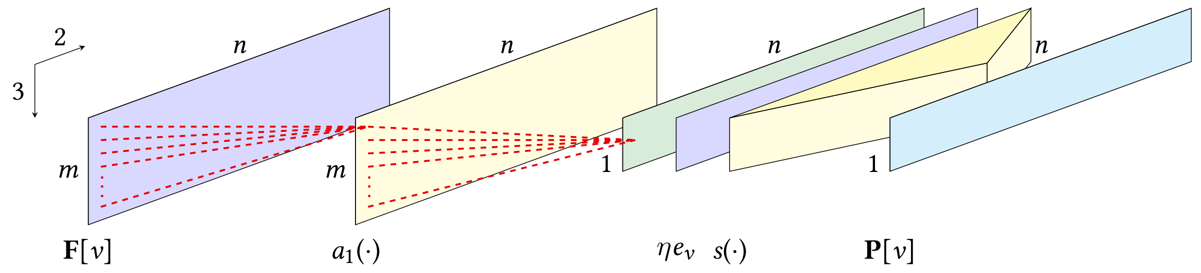
Firstly, in the identification problem, since the environmental features $\mathbf{f}$ and pairwise-distances $\mathbf{D}$, which are used to build the dataset $\mathbf{F}$ in every FORWARD pass in Algorithm 3, do not change, we can avoid the redundant concatenation of $\mathbf{f}[u]$ and $\mathbf{D}[v][u]$ in BUILD-DATASET. We perform this concatenation once before training the identification problem model and simply place the $\mathbf{r_t}$ vector as the last slice of $\mathbf{F}$ in every FORWARD pass for time period t.

Secondly, in the pricing problem, since the entries of $\mathbf{w_1^*}$ that affect the gradient update rule for $\mathbf{r}$ don't undergo arithmetic operations with other entries of $\mathbf{w_1^*}$ or other features in $\mathbf{F}$ due to the backpropagation framework, we can split $\mathbf{w_1^*}$ in the pricing problem into those that interact with $\mathbf{r}$ and those that don't. This splitting reduces redundant computation, as described below.

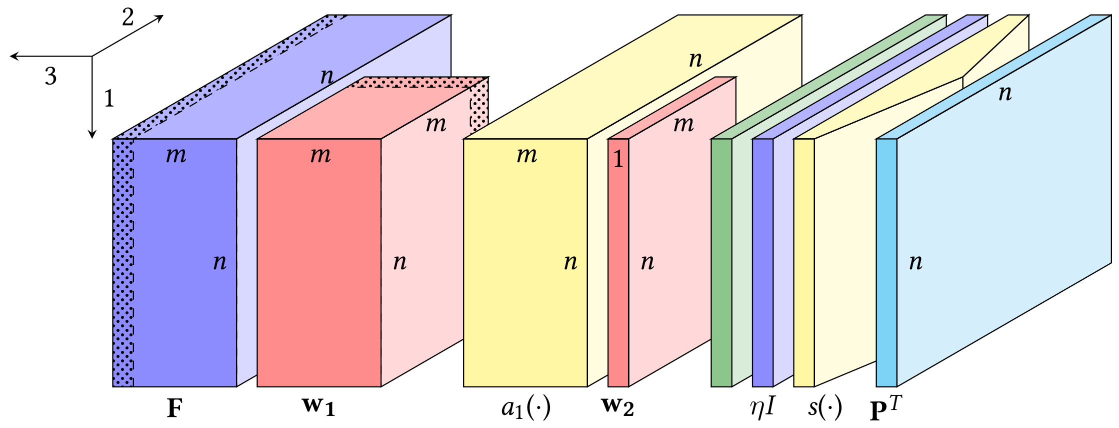
The batch-multiplication of $\mathbf{F}$ and $\mathbf{w_1^*}$ is essentially the product of $\mathbf{F}[v] \in \mathbb{R}^{n \times m}$ and $\mathbf{w_1^*}[v] \in \mathbb{R}^{m \times m}$ for all v, as shown in Figure 2c. Each of these products can be split as the following:

$$\mathbf{F}[v][:, : m] \times \mathbf{w_1^*}[v][: m, :] + \mathbf{F}[v][:, m] \times \mathbf{w_1^*}[v][m, :]$$

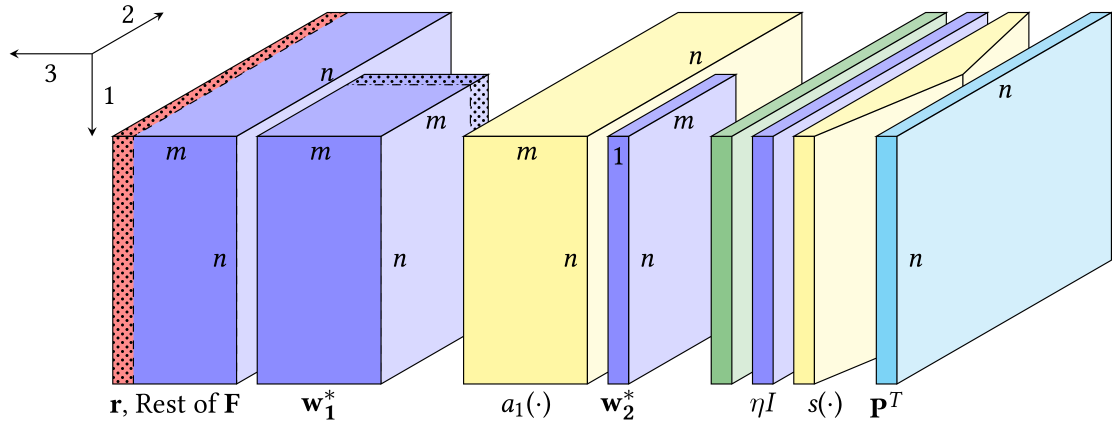
where the first product (involving 'Rest of $\mathbf{F}$' and corresponding entries of $\mathbf{w_1^*}$) remains constant, and the second product (involving $\mathbf{r}$ and corresponding entries of $\mathbf{w_1^*}$) gets updated during optimization. We precompute $\mathbf{F}[v][:, : m] \times \mathbf{w_1^*}[v][: m, :]$ for all v, compute the second product at each iteration, and simply add them to get $\mathbf{F} \times \mathbf{w_1^*}$. Therefore, we are able to perform a tensor addition and matrix multiplication instead of a tensor multiplication at each



(a) A network representation of the identification problem model. The network feeds m features per task-pair (u, v) forward in the network, in a sequence of linear combinations and activations. The regularizer ηe_v is then added ($e_v[u] = 1$ if $u = v$ else 0). After applying softmax, the network outputs $\mathrm{P}[v]$ as in Equation (2).



(b) A tensor representation of Figure 2a. For each t, we place the rewards $\mathrm{r_t}$ on the dotted slice of F, feed horizontal batches of matrices $\mathrm{F}[v]$ and obtain P^T in one pass. This also illustrates how a neural network is a sequence of matrix multiplications, additions, and element-wise function applications. Input $n \times n \times m$-size F is multiplied by $n \times m \times m$-size weights w_1 to get a $n \times n \times m$-size tensor, on which activation function a_1 is applied. We then multiply $n \times m \times 1$-size w_2, add ηI, and row-wise apply softmax to get P^T. The red tensors are then iteratively updated with backpropagation.



(c) Solving the pricing problem. After learning w_i^* in the identification problem, the rewards in F are optimized. Now, only the tasks' rewards (last vertical slice in F) are optimizable. The rewards and their corresponding w_1^* weights are marked with dots.

Figure 2: 3-layer architecture for principal-agent games. Deeper networks simply add $n \times m \times m$-size $\mathrm{w_i}$ and a_i after applying a_1 and before multiplying with the ultimate $n \times m \times 1$-size weights. Static tensors are colored dark blue, optimizable tensors red, intermediate resultant tensors green, and activation functions a_i, applied to tensors resulting from a multiplication with $\mathrm{w_i}$, yellow. $s(\cdot)$ is the softmax function. Directions of dimensions are marked on the top-left.

feed-forward step, thus reducing the computational cost of training the pricing problem model.

4 EXPERIMENTS

In this section, we compare the empirical performance of our neural-network-based models to Xue et al.'s models [32] with the objective functions of identification (Equation (3)) and pricing problem (Equation (4)) as performance metrics. We also compare our models' scalability to that of MIP-based model [32], and discuss differences between CPU-run and GPU-run versions of our models.

We use recorded agents' visit densities from *eBird*, which were also used by Xue et al [32]. This visit density dataset $\mathcal{D}$ has $n = 116$ locations, $T = 182$ time periods, and 33 environmental features.

For the identification problem, we split the dataset 75-5-20 in train-validation-test sets in random order of time periods as the ordering of time periods in $\mathcal{D}$ is not known [13]. After training all neural networks for 1000 epochs with the Adam algorithm for gradient descent [17], we use the loss function's value on the validation set to find hyperparameters λ and η. Thereafter, we split $\mathcal{D}$ into 80-20 train-test sets, train with these regularization hyperparameters for 1000 epochs, and report the test loss, averaged over three random initializations of $\mathbf{w}$.

For each of the three sets of optimized $\mathbf{w}^*$, we optimize pricing problem models using the Adam algorithm with three random reward initializations adding to $\mathcal{R}$. When each of these models is run for 10000 epochs, we record the set of rewards that minimizes the objective function. We set $\mathcal{R} = 365$ as the rewards that were found by the MIP solution in Xue et al.'s work [32] satisfy this constraint. We use Rectified Linear Units (ReLUs) [22] for all activation functions a_i.

To demonstrate our identification problem models' scalability, we run them on random datasets of increasing number of locations n on both GPU and CPU for 50 epochs. We then run the pricing problem models' for 500 epochs similarly. Since our GPU has limited RAM, networks of different depths can train on datasets of different sizes, adjusted by locations n. For both problems, we find the maximum n that a k-layered model can be run on our GPU, and run that model on datasets with n in increments of 30, starting from 0 up to the maximum n. Although this results in different k-layered models getting tested for execution time on different sets of n, the scalability trends remain comparable. Since computational cost per epoch does not vary, the scalability trends for our models for both problems should behave similarly when models are executed for 1000 and 10000 epochs respectively.

We run all our experiments on a Dell Precision Tower 3620 PC with Intel Core i7-7700K (8 cores) with 16GB RAM and Nvidia Quadro P4000 GPU with 8GB RAM, and build our neural networks with CUDA 8.0, cuDNN 5.1.10, PyTorch 0.4.0 [24], NumPy 1.14.5, and MKL 2018.0.3 in Ubuntu 16.04 LTS. We use CPLEX Optimization Studio 12.8 with its Python API to run the MIP benchmark model [32]. The CPLEX solver uses all 8 cores of the CPU.

4.1 Experiments for the Identification Problem

We find that $\eta = 10$ and $\lambda = 1$ consistently result in low validation loss on any k-layered model. We then train 2-, 4-, and 6-layer models with learning rate 10^{-3} on the 80-20 dataset splits, and run Xue et al.'s BFGS-based 2-layer, Structural SVM, and Random Forest models [32] on the same splits.

Table 1 shows the decrease in test loss as the number of layers in the networks increases, and Figure 3 illustrates the computation speedup on GPUs vs CPUs.

We observe that the 6-layer network optimizes the loss function more than the 2-layer network. However, there is a trade-off between the number of locations a model can optimize for (at once) and the model's capability at optimizing the loss function, specifically due to memory limits and computation costs.

	Loss	Runtime (sec)
Random	1.014	–
Historical	0.554	–
Random Forest	0.491	26.4
SVM	0.978	0.1
BFGS	0.374	507.3
2-layer	0.366	48.0
4-layer	0.368	339.5
6-layer	**0.358**	647.8

Table 1: Comparing average runtimes and test loss, i.e., $\frac{\sum_t \|\omega_t(\mathrm{y_t}-\hat{\mathrm{y}}_\mathrm{t})\|_2^2}{\sum_t \|\omega_t(\mathrm{y_t}-\overline{\mathrm{y}})\|_2^2}$ (lower the better) for predictions $\hat{\mathrm{y}}_\mathrm{t}$, where $\overline{\mathrm{y}} = \mathrm{mean}(y_{t,u})$. The first two are baselines: random $\hat{\mathrm{y}}_\mathrm{t}$ and $\hat{\mathrm{y}}_\mathrm{t} \leftarrow \mathrm{x_t}$, i.e., using historical data as predictions. The next three [32] are CPU-run models, and the last three are our GPU-run neural networks. The last four models predict $\hat{\mathrm{y}}_\mathrm{t} = \mathrm{Px_t}$ by directly learning entries of P. We see that our 6-layer model performs $\approx 1.5\%$ better than the state-of-the-art BFGS-based model, while the 2-layer model delivers $\approx 0.5\%$ lower loss than the BFGS model in one-tenth the time. Moreover, we don't observe overfitting in deeper networks even as the training set remains constant, possibly due to batch-normalization and weight regularization that prevent deeper networks from overfitting during training.

4.2 Experiments for the Pricing Problem

We compare the different reward distributions' efficacy using the pricing problem's objective, and choose the following distributions as baseline metrics: zero, equal, random, and MIP-learned rewards.

We evaluate the zero/equal/random sets of rewards in the following way: using the weights learned by a k-layer model in the identification problem, we plug each reward set into the model and determine the value of the objective function. This process captures the efficacy of these rewards on reducing the predicted visit densities based on the behavior models learned in the identification problem. For each set of rewards, we report the objective value averaged over the three sets of weights that were generated for each k-layer model. For example, we feed zero rewards into each of the three sets of $\mathbf{w}^*$ obtained for 2-layer models, and report the average of the objective values.

Now we explain our setup for the MIP-learned benchmark rewards. We add the budget constraint $\|\mathbf{r}\|_1 \leq \mathcal{R}$ to Xue et al.'s MIP formulation [32] since the formulation didn't have one. Additionally, since the formulation required each reward value to be drawn

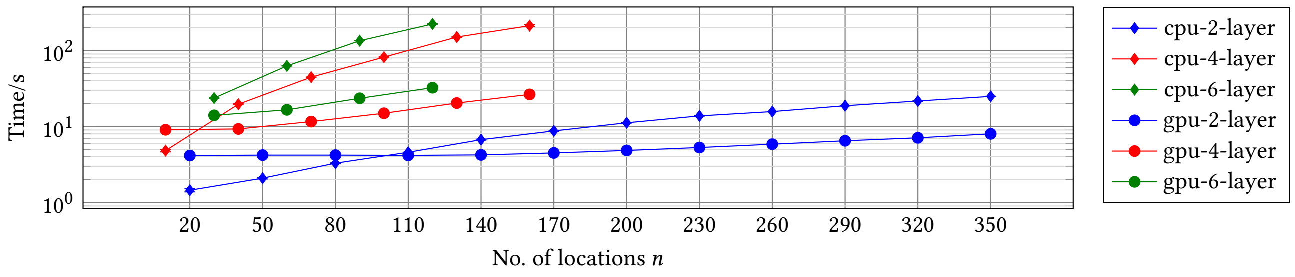


Figure 3: Demonstrating GPU vs CPU scalability in deep networks for identification problem as n increases. Executed for 50 epochs, all models scale better on GPUs, displaying the speedup with GPUs for large-scale games. For the 6-layer network, the GPU speedup is dramatic: upto ≈ 7 times with $n = 120$; the GPU-speedup only increases with deeper networks as parallel computation allows networks to scale. Due to memory restrictions, we could only run 6-layer models for $n \leq 120$ locations on our GPU with 8GB RAM, 4-layer models for $n \leq 160$, and 2-layer models for $n \leq 500$.

from a set of integers $\Re = \{R_1, R_2, \ldots, R_k\}$, we choose this set to be $\Re = \{0, 1, 2, \ldots, R_{max}\}$. Here, we set R_{max} to be a large enough integer based on the total budget $\mathcal{R}$, say $\lfloor \mathcal{R}/4 \rfloor$ (365 / 4 = 91 in our experiments). This is a heuristic based on the reward values we see in Xue et al.'s study and those that our neural networks learn. The MIP then uses the learned weights from our 2-layer network to optimize the objective $\|\mathbf{y} - \bar{\mathbf{y}}\|_1$ in a time limit of 10 hours. We report the objective value averaged over the three sets of weights for the 2-layer model from the identification problem.

Table 2 compares the efficacy of different sets of rewards, Figure 4 illustrates the GPU vs CPU scalability trends, and Table 3 compares the neural network models' runtimes for calculating rewards with the MIP's runtime.

	2-layer	4-layer	6-layer
Learned	1.073	1.236	**1.025**
MIP	1.110	–	–
Zero	1.168	1.505	1.302
Equal	1.169	1.506	1.302
Random	1.169	1.506	1.303

Table 2: Comparing the normalized objective, $\frac{1}{n}\frac{\|\mathbf{y}-\bar{\mathbf{y}}\|_1}{\bar{\mathbf{y}}}$ (lower the better) produced by different sets of rewards, where $\bar{\mathbf{y}} = \text{mean}(y_u)$. We observe that the 6-layer model results in the lowest score compared to 2- and 4-layer models, and our network-learned rewards consistently outperform baseline distributions. The MIP's rewards are not globally optimal as the solver exceeded the time limit of 10 hours in every run; however, the solver was able to find a few integer solutions. This, and the integer constraints on rewards in the MIP, explain why the MIP rewards result in a worse score compared to the 2-layer network optimized using gradient-descent. Since the 2- and 6-layer models did not dramatically differ in performance in the identification problem (Table 1, we contend that the learned rewards by the 6-layer network slightly outperform the MIP's rewards. Since the deeper networks have non-linear activation functions, we could not embed them into the MIP.

	2-layer	4-layer	6-layer	MIP
Runtime (sec)	9.65	26.80	44.45	$\geq 36,000$

Table 3: Comparing runtimes of pricing problem models, all run on original *eBird* data with 116 locations. Since the MIPs exceeded the 10-hour time limit, there runtimes are at least 10 hours. Whereas the MIPs take $\geq 36,000$ seconds to terminate, our GPU-run networks finish 10000 epochs in ≤ 50 seconds. The 6-layer model thus runs $\geq 800x$ faster than the MIP model, demonstrating that our models provide several orders of magnitude speedup.

As noted in Table 2, the MIP could not find optimal rewards in the time limit of 10 hours, which may be a reason why rewards learned by our networks outperform MIP rewards. On a subset of the *eBird* dataset with only $n = 10$ locations, we repeated the pricing problem experiments for 2-layer networks to test if the optimal MIP strategy performs worse than the neural-network-based approach even on a very small dataset. After calculating 3 sets of $\mathbf{w}^*$ from the identification problem, the neural-network-based approach achieved 0.946 on the metric, averaged over 3 random initializations of rewards and different sets of $\mathbf{w}^*$. The MIP solver convincingly outperformed after running to completion in ≈ 1 minute, scoring 0.573 on the metric, averaged over different $\mathbf{w}^*$. Therefore, although our neural-network-based approach cannot match the MIP's performance when the dataset is small, our models are competitive in performance when the dataset grows in size. This behavior highlights the importance of scalability in obtaining influential rewards for the *Avicaching* game, as the speedup of our approach aids the performance to slightly outperform the previous optimal MIP approach.

5 CONCLUSION

In this study, we tackle spatial clustering in large-scale citizen science programs with a machine-learning-based formulation of principal-agent games. We build scalable architectures for *Avicaching*, and use the same architecture to solve both the identification and pricing problems using backpropagation for optimization. Our formulation puts parallelization on the center stage to solve

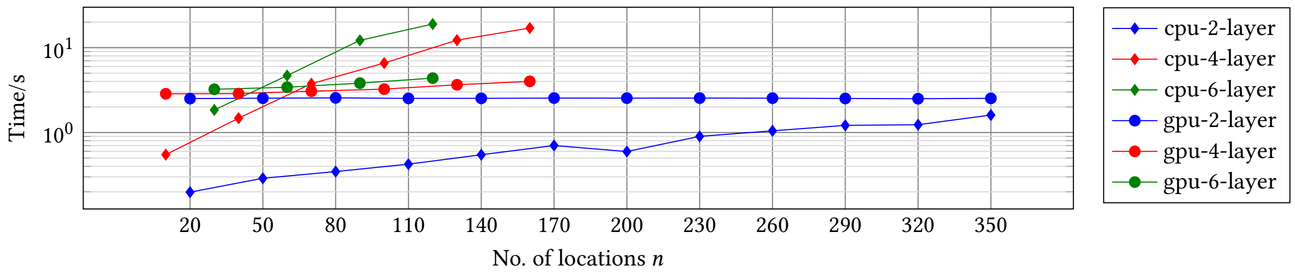


Figure 4: Demonstrating GPU vs CPU scalability in deep networks for pricing problem as n increases. Run for 500 epochs each, we see that computation cost is not a limiting factor in the pricing problem. This is because there are only n parameters in the neural network model, namely the rewards, making backpropagation cheaper than in the identification problem. While GPU runs are costlier for small datasets ($n \leq 480$ for 2-, $n \leq 60$ for 4-, and $n \leq 40$ for 6-layer model) as CPU-GPU memory transfer overhead dominates, GPU and CPU runs follow a similar trend vs n as in the identification problem.

large-scale games, and we demonstrate our models' viability compared to the previous state of the art, which used MIP. We evaluate our models on real-life bird observational data from *eBird*, and demonstrate that our models capture agents' behavior and incentive distribution more effectively than previous studies. Moreover, the structure of our models allows us to tap parallel computation to speed up runtime by several orders of magnitude.

With scalable models to learn citizens' behavior and incentives that motivate citizens to complete unappealing tasks, citizen science programs can reduce sampling bias on an unprecedented scale, and improve the datasets' use in rapid scientific discovery and modeling.

As for the future work, it will be interesting to implement our techniques for other citizen science programs with esoteric constraints on reward allocation. While the *Avicaching* game is representative of the general game in most programs, games with unconventional constraints will challenge our methods. There is also immense potential in considering memory-efficient models to enable deeper networks, and larger networks in terms of the number of locations. Confronting the memory-performance trade-off in the identification problem is a compelling research project. We also see merit in end-to-end principal-agent game solvers, which do not take the two-stage strategy we propose. End-to-end solvers will be easier and more practical to deploy, and might circumvent ad-hoc hyperparameter optimization. Future research may also further compare the neural networks with mixed-integer programming. Our study benefits from relaxing the integer constraints on rewards that Xue et al. enforced [31, 32], and studies on these types of relaxations may lead to interesting juxtapositions between the two optimization frameworks.

6 ACKNOWLEDGEMENTS

We are grateful to the anonymous reviewers for their suggested clarifications, and comments that helped us improve our experiments. We also thank the thousands of *eBird* participants, and the Cornell Lab of Ornithology for managing *eBird* and deploying *Avicaching*. This research study was supported by the National Science Foundation (NSF) awards CCF-1522054 and CNS-0832782 (Expeditions), and the computing infrastructure was supported by the Army Research Office (ARO) grant W911NF-17-1-0187 (DURIP).

REFERENCES

[1] Abdulmonem Alabri and Jane Hunter. 2010. Enhancing the Quality and Trust of Citizen Science Data. In *2010 IEEE Sixth International Conference on e-Science*. 81–88. https://doi.org/10.1109/eScience.2010.33
[2] Ron Bekkerman, Mikhail Bilenko, and John Langford. 2011. *Scaling Up Machine Learning: Parallel and Distributed Approaches*. Cambridge University Press, New York, NY, USA.
[3] Tomas J. Bird, Amanda E. Bates, Jonathan S. Lefcheck, Nicole A. Hill, Russell J. Thomson, Graham J. Edgar, Rick D. Stuart-Smith, Simon Wotherspoon, Martin Krkosek, Jemina F. Stuart-Smith, Gretta T. Pecl, Neville Barrett, and Stewart Frusher. 2014. Statistical solutions for error and bias in global citizen science datasets. *Biological Conservation* 173 (2014), 144 – 154. https://doi.org/10.1016/j.biocon.2013.07.037
[4] Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.
[5] Rick Bonney, Caren B. Cooper, Janis Dickinson, Steve Kelling, Tina Phillips, Kenneth V. Rosenberg, and Jennifer Shirk. 2009. Citizen Science: A Developing Tool for Expanding Science Knowledge and Scientific Literacy. *BioScience* 59, 11 (2009), 977–984. https://doi.org/10.1525/bio.2009.59.11.9
[6] Matthew Brown, Sandhya Saisubramanian, Pradeep Varakantham, and Milind Tambe. 2014. STREETS: Game-theoretic Traffic Patrolling with Exploration and Exploitation. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI'14)*. AAAI Press, 2966–2971. http://dl.acm.org/citation.cfm?id=2892753.2892962
[7] Benoît Colson, Patrice Marcotte, and Gilles Savard. 2007. An overview of bilevel optimization. *Annals of Operations Research* 153, 1 (01 Sep 2007), 235–256. https://doi.org/10.1007/s10479-007-0176-2
[8] Vincent Conitzer and Nikesh Garera. 2006. Learning Algorithms for Online Principal-agent Problems (and Selling Goods Online). In *Proceedings of the 23rd International Conference on Machine Learning (ICML '06)*. ACM, New York, NY, USA, 209–216. https://doi.org/10.1145/1143844.1143871
[9] Angela J. Dean, Emma K. Church, Jenn Loder, Kelly S. Fielding, and Kerrie A. Wilson. 2018. How do marine and coastal citizen science experiences foster environmental engagement? *Journal of Environmental Management* 213 (2018), 409 – 416. https://doi.org/10.1016/j.jenvman.2018.02.080
[10] Daniel Fink, Theodoros Damoulas, and Jaimin Dave. 2013. Adaptive Spatio-Temporal Exploratory Models: Hemisphere-wide species distributions from massively crowdsourced eBird data. https://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6417/6852
[11] Sean Gailmard, Mark Bovens, Robert E. Goodin, and Thomas Schillemans. 2014. Accountability and Principal-Agent Theory. http://www.oxfordhandbooks.com/view/10.1093/oxfordhb/9780199641253.001.0001/oxfordhb-9780199641253-e-016
[12] Jonas Geldmann, Jacob Heilmann-Clausen, Thomas E. Holm, Irina Levinsky, Bo Markussen, Kent Olsen, Carsten Rahbek, and Anders P. Tøttrup. 2016. What determines spatial bias in citizen science? Exploring four recording schemes with different proficiency requirements. *Diversity and Distributions* 22, 11 (11 2016), 1139–1149. https://doi.org/10.1111/ddi.12477
[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. The MIT Press.
[14] Allen H. Hurlbert and Zhongfei Liang. 2012. Spatiotemporal Variation in Avian Migration Phenology: Citizen Science Reveals Effects of Climate Change. *PLOS ONE* 7, 2 (02 2012), 1–11. https://doi.org/10.1371/journal.pone.0031662

[15] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37 (ICML'15)*. JMLR.org, 448–456. http://dl.acm.org/citation.cfm?id=3045118.3045167

[16] Steve Kelling, Jeff Gerbracht, Daniel Fink, Carl Lagoze, Weng-Keen Wong, Jun Yu, Theodoros Damoulas, and Carla Gomes. 2012. eBird: A Human/Computer Learning Network for Biodiversity Conservation and Research. https://www.aaai.org/ocs/index.php/IAAI/IAAI-12/paper/view/4880

[17] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). arXiv:1412.6980

[18] Jean-Jacques Laffont and David Martimort. 2002. *The Theory of Incentives: The Principal-Agent Model.* Princeton University Press. http://www.jstor.org/stable/j.ctv7h0rwr

[19] Chris J. Lintott, Kate Land, Kevin Schawinski, Anže Slosar, Daniel Thomas, Robert C. Nichol, Steven Bamford, Alex Szalay, Jan Vandenberg, M. Jordan Raddick, Dan Andreescu, and Phil Murray. 2008. Galaxy Zoo: morphologies derived from visual inspection of galaxies from the Sloan Digital Sky Survey*. *Monthly Notices of the Royal Astronomical Society* 389, 3 (09 2008), 1179–1189. https://doi.org/10.1111/j.1365-2966.2008.13689.x

[20] N Justin Marshall, Diana A Kleine, and Angela J Dean. 2012. CoralWatch: education, monitoring, and sustainability through citizen science. *Frontiers in Ecology and the Environment* 10, 6 (2012), 332–334. http://www.jstor.org/stable/41811402

[21] Duncan C. McKinley, Abe J. Miller-Rushing, Heidi L. Ballard, Rick Bonney, Hutch Brown, Susan C. Cook-Patton, Daniel M. Evans, Rebecca A. French, Julia K. Parrish, Tina B. Phillips, Sean F. Ryan, Lea A. Shanley, Jennifer L. Shirk, Kristine F. Stepenuck, Jake F. Weltzin, Andrea Wiggins, Owen D. Boyle, Russell D. Briggs, Stuart F. Chapin, David A. Hewitt, Peter W. Preuss, and Michael A. Soukup. 2017. Citizen science can improve conservation science, natural resource management, and environmental protection. *Biological Conservation* 208 (2017), 15 – 28. https://doi.org/10.1016/j.biocon.2016.05.015

[22] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning (ICML'10)*. Omnipress, USA, 807–814. http://dl.acm.org/citation.cfm?id=3104322.3104425

[23] Praveen Paruchuri, Jonathan P. Pearce, Janusz Marecki, Milind Tambe, Fernando Ordonez, and Sarit Kraus. 2008. Playing Games for Security: An Efficient Exact Algorithm for Solving Bayesian Stackelberg Games. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2 (AAMAS '08)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 895–902. http://dl.acm.org/citation.cfm?id=1402298.1402348

[24] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.

[25] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural Networks* 61 (2015), 85 – 117. https://doi.org/10.1016/j.neunet.2014.09.003

[26] Steven Shavell. 1979. Risk Sharing and Incentives in the Principal and Agent Relationship. *The Bell Journal of Economics* 10, 1 (1979), 55–73. http://www.jstor.org/stable/3003319

[27] Yaron Singer and Manas Mittal. 2013. Pricing Mechanisms for Crowdsourcing Markets. In *Proceedings of the 22Nd International Conference on World Wide Web (WWW '13)*. ACM, New York, NY, USA, 1157–1166. https://doi.org/10.1145/2488388.2488489

[28] D. Steinkraus, I. Buck, and P. Y. Simard. 2005. Using GPUs for machine learning algorithms. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*. 1115–1120 Vol. 2. https://doi.org/10.1109/ICDAR.2005.251

[29] Brian L. Sullivan, Christopher L. Wood, Marshall J. Iliff, Rick E. Bonney, Daniel Fink, and Steve Kelling. 2009. eBird: A citizen-based bird observation network in the biological sciences. *Biological Conservation* 142, 10 (2009), 2282 – 2292. https://doi.org/10.1016/j.biocon.2009.05.006

[30] Patrícia Tiago, Ana Ceia-Hasse, Tiago A. Marques, César Capinha, and Henrique M. Pereira. 2017. Spatial distribution of citizen science casuistic observations for different taxonomic groups. *Scientific Reports* 7, 1 (2017), 12832. https://doi.org/10.1038/s41598-017-13130-8

[31] Yexiang Xue, Ian Davies, Daniel Fink, Christopher Wood, and Carla P. Gomes. 2016. Avicaching: A Two Stage Game for Bias Reduction in Citizen Science. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 776–785. https://dl.acm.org/citation.cfm?id=2936924.2937038

[32] Yexiang Xue, Ian Davies, Daniel Fink, Christopher Wood, and Carla P. Gomes. 2016. Behavior Identification in Two-Stage Games for Incentivizing Citizen Science Exploration. In *Principles and Practice of Constraint Programming*, Michel Rueher (Ed.). Springer International Publishing, Cham, 701–717. https://doi.org/10.1007/978-3-319-44953-1_44