Example 3.12. Here we provide an example to show that the serial cost sharing mechanism can be far away from optimality. We pick a simple Bernoulli distribution, where an agent's valuation is 0 with 0.5 probability and 1 with 0.5 probability.[5] Under the serial cost sharing mechanism, when there are n agents, only half of the agents are consumers (those who report 1s). So in expectation, the number of consumers is $\frac{n}{2}$. Let us consider another simple mechanism. We assume that there is an ordering of the agents based on their identities (not based on their types). The mechanism asks the first agent to accept a cost share of 1. If this agent disagrees, she is removed from the system. The mechanism then moves on to the next agent and asks the same, until an agent agrees. If an agent agrees, then all future agents can consume the project for free. The number of removed agents follows a geometric distribution with 0.5 success probability. So in expectation, 2 agents are removed. That is, the expected number of consumers is $n - 2$.

4 MECH. DESIGN VS NEURAL NETWORKS

For the rest of this paper, we focus on the excludable public project model and distributions that are not log-concave. Due to the characterization results, we only need to consider the largest unanimous mechanisms. We use neural networks and deep learning to solve for well-performing largest unanimous mechanisms. Our approach involves several technical innovations as discussed in Section 1.

4.1 Mech. Design via Neural Networks

We start with an overview of automated mechanism design (AMD) via neural networks. Previous papers on mechanism design via neural networks [4–6, 13] all fall into this general category.

- Use neural networks to represent the full (or a part of the) mechanism. Like mechanisms, neural networks are essentially functions with multi-dimensional inputs and outputs.
- Training is essentially to adjust the network parameters in order to move toward a better performing network/mechanism. Training is just parameter optimization.
- Training samples are not real data. Instead, the training type profiles are generated based on the known prior distribution. We can generate infinitely many fresh samples. We use these generated samples to build the cost function, which is often a combination of mechanism design objective and constraint penalties. The cost function must be differentiable with respect to the network parameters.
- The testing data are also type profiles generated based on the known prior distribution. Again, we can generate infinitely many fresh samples, so testing is based on completely fresh samples. We average over enough samples to calculate the mechanism's expected performance.

4.2 Network Structure

A largest unanimous mechanism specifies constant cost shares for every coalition of agents. The mechanism can be characterized by a neural network with n binary inputs and n outputs. The n binary inputs present the coalition, and the n outputs represent the constant cost shares. We use $\vec{b}$ to denote the input vector (tensor) and $\vec{c}$ to denote the output vector. We use NN to denote the neural network, so $NN(\vec{b}) = \vec{c}$. There are several network constraints:

- All cost shares are nonnegative: $\vec{c} \geq 0$.
- For input coordinates that are 1s, the output coordinates should sum up to 1. For example, if $n = 3$ and $\vec{b} = (1, 0, 1)$ (the coalition is $\{1, 3\}$), then $\vec{c}_1 + \vec{c}_3 = 1$ (agent 1 and 3 are to share the total cost).
- For input coordinates that are 0s, the output coordinates are irrelevant. We set these output coordinates to 1s, which makes it more convenient for the next constraint.
- Every output coordinate is nondecreasing in every input coordinate. This is to ensure that the agents' cost shares are nondecreasing when some other agents are removed. If an agent is removed, then her cost share offer is kept at 1, which makes it trivially nondecreasing.

All constraints except for the last is easy to achieve. We will simply use $OUT(\vec{b})$ as output instead of directly using $NN(\vec{b})$[6]:

$$OUT(\vec{b}) = \text{softmax}(NN(\vec{b}) - 1000(1 - \vec{b})) + (1 - \vec{b})$$

Here, 1000 is an arbitrary large constant. For example, let $\vec{b} = (1, 0, 1)$ and $\vec{c} = NN(\vec{b}) = (x, y, z)$. We have

$$OUT(\vec{b}) = \text{softmax}((x, y, z) - 1000(0, 1, 0)) + (0, 1, 0)$$

$$= \text{softmax}((x, y - 1000, z)) + (0, 1, 0)$$

$$= (x', 0, z') + (0, 1, 0) = (x', 1, y')$$

In the above, $\text{softmax}((x, y - 1000, z))$ becomes $(x', 0, y')$ with $x', y' \geq 0$ and $x' + y' = 1$ because the second coordinate is very small so it (essentially) vanishes after softmax. Softmax always produces nonnegtive outputs that sum up to 1. Finally, the 0s in the output are flipped to 1s per our third constraint.

The last constraint is enforced using a penalty function. For $\vec{b}$ and $\vec{b}'$, where $\vec{b}'$ is obtained from $\vec{b}$ by changing one 1 to 0, we should have that $OUT(\vec{b}) \leq OUT(\vec{b}')$, which leads to this penalty:

$$\text{ReLU}(OUT(\vec{b}) - OUT(\vec{b}'))$$

Another way to enforce the last constraint is to use the *monotonic networks* structure [14]. This idea has been used in [5], where the authors also dealt with networks that take binary inputs and must be monotone. However, we do not use this approach because it is incompatible with our design for achieving the other constraints. There are two other reasons for not using the monotonic network structure. One is that it has only two layers. Some argue that having a *deep* model is important for performance in deep learning [16]. The other is that under our approach, we only need a fully connected network with ReLU penalty, which is highly optimized in state-of-the-art deep learning toolsets (while the monotonic network structure is not efficiently supported by existing toolsets). In our experiments, we use a fully connected network with four layers (100 nodes each layer) to represent our mechanism.

[5] Our paper assumes that the distribution is continuous, so technically we should be considering a smoothed version of the Bernoulli distribution. For the purpose of demonstrating an elegant example, we ignore this technicality.

[6] This is done by appending additional calculation structures to the output layer.

4.3 Cost Function

We focus on maximizing the expected number of consumers. Only slight adjustments are needed for welfare maximization.

Previous approaches of mechanism design via neural networks used *static* networks [4–6, 13]. Our largest unanimous mechanism involves iterative decision making, and the number of rounds is not fixed, as it depends on the users' inputs. To model iterative decision making via a static network, we could adopt the following process. The initial offers are $OUT((1, 1, \ldots, 1))$. The remaining agents after the first round are then $S = \text{sigmoid}(v - OUT((1, 1, \ldots, 1)))$. Here, v is the type profile sample. The next round of offers are then $OUT(S)$. The remaining agents afterwards are then $\text{sigmoid}(v - OUT(S))$. We repeat this n times because the largest unanimous mechanism have at most n rounds. The final coalition is a converged state, so even if the mechanism terminates before the n-th round, having it repeat n times does not change the result (except for additional numerical errors). Once we have the final coalition S^f, we include $\sum_{x \in S^f} x$ (number of consumers) in the cost function. However, this approach performs *abysmally*, due to the *vanishing gradient problem* and numerical errors caused by stacking n sigmoid functions.

We would like to avoid stacking sigmoid to model iterative decision making. Sigmoid is heavily used in existing works on neural network mechanism design, but it is the culprit of significant numerical errors. We propose an alternative approach, where decisions are simulated off the network using a separate program (*e.g.*, any Python function). The advantage of this approach is that it is now trivial to handle complex decision making. However, given a type profile sample v and the current network NN, if we simulate the mechanism off the network to obtain the number of consumers x, and include x in the cost function, then training will fail completely. This is because x is not a differentiable function of network parameters and cannot support backpropagation at all.

One way to resolve this is to interpret the mechanisms as price-oriented rationing-free (PORF) mechanisms [15]. That is, if we single out one agent, then her options (outcomes combined with payments) are completely determined by the other agents and she simply has to choose the utility-maximizing option. Under a largest unanimous mechanism, an agent faces only two results: either she belongs to the largest unanimous coalition or not. If an agent is a consumer, then her payment is a constant due to strategy-proofness, and the constant payment is determined by the other agents. Instead of sampling over complete type profiles, we sample over v_{-i} with a random i. To better convey our idea, we consider a specific example. Let $i = 1$ and $v_{-1} = (\cdot, \frac{1}{2}, \frac{1}{2}, \frac{1}{4}, 0)$. We assume that the current state of the neural network is exactly the serial cost sharing mechanism. Given a sample, we use a separate program to calculate the following entries.

- The objective value if i is a consumer (O_s). Under the example, if 1 is a consumer, then the decision must be 4 agents each pays $\frac{1}{4}$. So the objective value is $O_s = 4$.
- The objective value if i is not a consumer (O_f). Under the example, if 1 is not a consumer, then the decision must be 2 agents each pay $\frac{1}{2}$. So the objective value is $O_f = 2$.
- The binary vector that characterizes the coalition that decides i's offer ($\vec{O_b}$). Under the example, $\vec{O_b} = (1, 1, 1, 1, 0)$.

O_s, O_f, and $\vec{O_b}$ are constants without network parameters. We link them together using terms with network parameters, which is then included in the cost function:

$$(1 - F(OUT(\vec{O_b})_i))O_s + F(OUT(\vec{O_b})_i)O_f \quad (1)$$

$1 - F(OUT(\vec{O_b})_i)$ is the probability that agent i accepts her offer. $F(OUT(\vec{O_b})_i)$ is then the probability that agent i rejects her offer. $OUT(\vec{O_b})_i$ carries gradients as it is generated by the network. We use the analytical form of F, so the above term carries gradients.[7]

The above approach essentially feeds the prior distribution into the cost function. We also experimented with two other approaches. One does not use the prior distribution. It uses a full profile sample and uses one layer of sigmoid to select between O_s or O_f:

$$\text{sigmoid}(v_i - OUT(\vec{O_b})_i)O_s + \text{sigmoid}(OUT(\vec{O_b})_i - v_i))O_f \quad (2)$$

The other approach is to feed "even more" distribution information into the cost function. We single out two agents i and j. Now there are 4 options: they both win or both lose, only i wins, and only j wins. We still use F to connect these options together.

In Section 5, in one experiment, we show that singling out one agent works the best. In another experiment, we show that even if we do not have the analytical form of F, using an analytical approximation also enables successful training.

4.4 Supervision as Initialization

We introduce an additional supervision step in the beginning of the training process as a systematic way of initialization. We first train the neural network to mimic an existing manual mechanism, and then leave it to gradient descent. We considered three different manual mechanisms. One is the serial cost sharing mechanism. The other two are based on two different heuristics:

Definition 4.1 (One Directional Dynamic Programming). We make offers to the agents one by one. Every agent faces an offer based on how many agents are left, the objective value cumulated so far by the previous agents, and how much money still needs to be raised. If an agent rejects an offer, then she is removed. At the end of the algorithm, if we collected 1, the project is built and all agents not removed are consumers. This mechanism belongs to the largest unanimous mechanism family. This mechanism is not optimal because we cannot go back and increase an agent's offer.

Definition 4.2 (Myopic Mechanism). For coalition size k, we treat it as a nonexcludable public project problem with k agents. The offers are calculated based on the dynamic program proposed at the end of Subsection 3.3. This mechanism is not necessarily feasible, because the agents' offers are not necessarily nondecreasing when some agents are removed.

5 EXPERIMENTS

The experiments are conducted on a machine with Intel i5-8300H CPU.[8] The largest experiment with 10 agents takes about 3 hours. Smaller scale experiments take only about 15 minutes.

[7] PyTorch has built-in analytical CDFs of many common distributions.
[8] We experimented with both PyTorch and Tensorflow (eager mode). The PyTorch version runs significantly faster, because we are dealing with dynamic graphs.

Figure 1: Effect of Distribution Info on Training

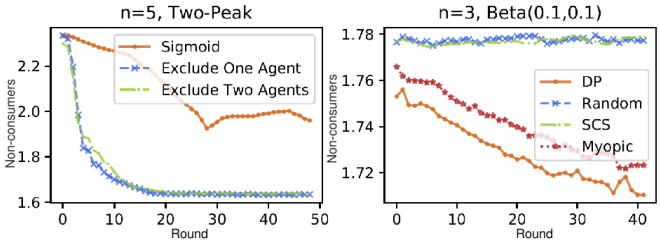
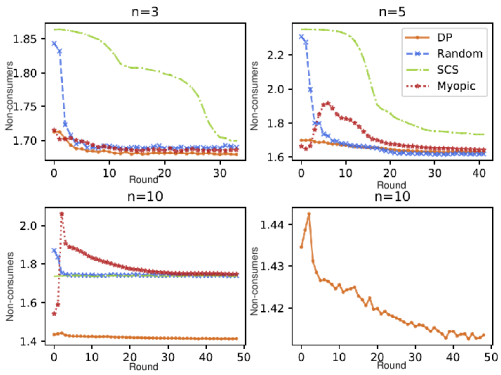


Figure 2: Supervision to Different Manual Mechanisms



In our experiments, unless otherwise specified, the distribution considered is two-peak $(0.15, 0.1, 0.85, 0.1, 0.5)$. The x-axis shows the number of training rounds. Each round involves 5 batches of 128 samples (640 samples each round). Unless otherwise specified, the y-axis shows the expected number of **non**consumers (so lower values represent better performances). Random initializations are based on Xavier normal with bias 0.1.

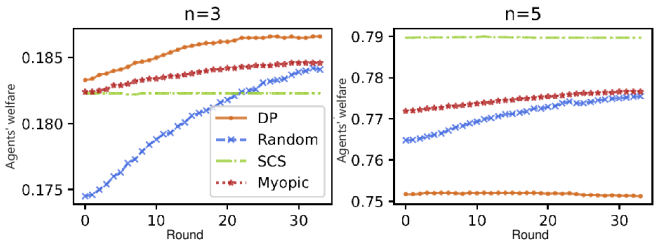
Figure 1 (Left) shows the performance comparison of three different ways for constructing the cost function: using one layer of sigmoid (without using distribution) based on (2), singling out one agent based on (1), and singling out two agents. All trials start from random initializations. In this experiment, singling out one agent works the best. The sigmoid-based approach is capable of moving the parameters, but its result is noticeably worse. Singling out two agents has almost identical performance to singling out one agent, but it is slower in terms of time per training step.

Figure 1 (Right) considers the Beta $(0.1, 0.1)$ distribution. We use Kumaraswamy $(0.1, 0.354)$'s analytical CDF to approximate the CDF of Beta $(0.1, 0.1)$. The experiments show that if we start from random initializations (Random) or start by supervision to serial cost sharing (SCS), then the cost function gets stuck. Supervision to one directional dynamic programming (DP) and Myoptic mechanism (Myopic) leads to better mechanisms. So in this example scenario, approximating CDF is useful when analytical CDF is not available. It also shows that supervision to manual mechanisms works better than random initializations in this case.

Figure 2 (Top-Left $n = 3$, Top-Right $n = 5$, Bottom-Left $n = 10$) shows the performance comparison of supervision to different manual mechanisms. For $n = 3$, supervision to DP performs the best. Random initializations is able to catch up but not completely close the gap. For $n = 5$, random initializations caught up and actually became the best performing one. The Myopic curve first increases and then decreases because it needs to first fix the constraint violations. For $n = 10$, supervision to DP significantly outperforms the others. Random initializations closes the gap with regard to serial cost sharing, but it then gets stuck. Even though it looks like the DP curve is flat, it is actually improving, albeit very slowly. A magnified version is shown in Figure 2 (Bottom-Right).

Figure 3 shows two experiments on maximizing expected agents' welfare (y-axis) under two-peak $(0.2, 0.1, 0.6, 0.1, 0.5)$. For $n = 3$,

Figure 3: Maximizing Agents' Welfare



supervision to DP leads to the best result. For $n = 5$, SCS is actually the best mechanism we can find (the cost function barely moves). It should be noted that all manual mechanisms *before training* have very similar welfares: 0.7517 (DP), 0.7897 (SCS), 0.7719 (Myopic). Even random initialization before training has a welfare of 0.7648. In this case, there is just little room for improvement.

6 CONCLUSION

In this paper, we studied optimal-in-expectation mechanism design for the public project model. We are the first to use neural networks to design iterative mechanisms. To avoid modeling iterative decision making via the sigmoid function, we simulate the different options an agent faces under an iterative mechanism and combine the options using distribution information to build the cost function for our optimal-in-expectation objective. We showed that under various conditions, existing mechanisms or mechanisms derived via classic mechanism design approaches are optimal. When classic mechanism design approaches do not suffice, we derived better mechanisms via neural networks by first training the neural networks to mimic manual mechanisms and then improving by gradient descent.

REFERENCES

[1] Mark Bagnoli and Ted Bergstrom. 2005. Log-concave probability and its applications. *Economic Theory* 26, 2 (01 Aug 2005), 445–469. https://doi.org/10.1007/s00199-004-0514-4

[2] Vincent Conitzer and Tuomas Sandholm. 2002. Complexity of Mechanism Design. In *UAI '02, Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence, University of Alberta, Edmonton, Alberta, Canada, August 1-4, 2002*, Adnan Darwiche and Nir Friedman (Eds.). Morgan Kaufmann, 103–110.

[3] Rajat Deb and Laura Razzolini. 1999. Voluntary cost sharing for an excludable public project. *Mathematical Social Sciences* 37, 2 (1999), 123 – 138.

[4] Paul Duetting, Zhe Feng, Harikrishna Narasimhan, David Parkes, and Sai Srivatsa Ravindranath. 2019. Optimal Auctions through Deep Learning. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, Long Beach, California, USA, 1706–1715.

[5] Noah Golowich, Harikrishna Narasimhan, and David C. Parkes. 2018. Deep Learning for Multi-Facility Location Mechanism Design. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 261–267.

[6] Padala Manisha, C. V. Jawahar, and Sujit Gujar. 2018. Learning Optimal Redistribution Mechanisms Through Neural Networks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, Elisabeth André, Sven Koenig, Mehdi Dastani, and Gita Sukthankar (Eds.). International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 345–353.

[7] Andreu Mas-Colell, Michael Whinston, and Jerry R. Green. 1995. *Microeconomic Theory*. Oxford University Press.

[8] J. Moore. 2006. *General Equilibrium and Welfare Economics: An Introduction*. Springer.

[9] H. Moulin. 1988. *Axioms of Cooperative Decision Making*. Cambridge University Press.

[10] Hervé Moulin. 1994. Serial Cost-Sharing of Excludable Public Goods. *The Review of Economic Studies* 61, 2 (1994), 305–325.

[11] Shinji Ohseto. 2000. Characterizations of Strategy-Proof Mechanisms for Excludable versus Nonexcludable Public Projects. *Games and Economic Behavior* 32, 1 (2000), 51 – 66.

[12] Ran Shao and Lin Zhou. 2016. Optimal allocation of an indivisible good. *Games and Economic Behavior* 100 (2016), 95 – 112.

[13] Weiran Shen, Pingzhong Tang, and Song Zuo. 2019. Automated Mechanism Design via Neural Networks. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems* (Montreal QC, Canada) *(AAMAS '19)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 215–223.

[14] Joseph Sill. 1998. Monotonic Networks. In *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10* (Denver, Colorado, USA) *(NIPS '97)*. MIT Press, Cambridge, MA, USA, 661–667.

[15] Makoto Yokoo. 2003. Characterization of Strategy/False-name Proof Combinatorial Auction Protocols: Price-oriented, Rationing-free Protocol. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence* (Acapulco, Mexico) *(IJCAI'03)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 733–739.

[16] Zhi-Hua Zhou and Ji Feng. 2017. Deep Forest: Towards an Alternative to Deep Neural Networks. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence* (Melbourne, Australia) *(IJCAI'17)*. AAAI Press, 3553–3559.