

# idinn: A Python Package for Inventory-Dynamics Control with Neural Networks

Jiawei Li<sup>1</sup>, Thomas Asikis<sup>2</sup>, Ioannis Fragkos<sup>3</sup>, and Lucas Böttcher<sup>1,4</sup>

<sup>1</sup> Department of Computational Science and Philosophy, Frankfurt School of Finance and Management  
<sup>2</sup> Game Theory, University of Zurich <sup>3</sup> Department of Technology and Operations Management, Rotterdam School of Management, Erasmus University Rotterdam <sup>4</sup> Laboratory for Systems Medicine, Department of Medicine, University of Florida ¶ Corresponding author

DOI: 10.xxxxxx/draft

## Software

- Review
- Repository
- Archive

Editor: Open Journals

## Reviewers:

- @openjournals

Submitted: 01 January 1970

Published: unpublished

## License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0)

## Summary

Identifying optimal policies for replenishing inventory from multiple suppliers is a key problem in inventory management. Solving such optimization problems requires determining the quantities to order from each supplier based on the current inventory and outstanding orders, minimizing the expected ordering, holding, and out-of-stock costs. Despite over 60 years of extensive research on inventory management problems, even fundamental dual-sourcing problems—where orders from an expensive supplier arrive faster than orders from a low-cost supplier—remain analytically intractable (Barankin, 1961; Fukuda, 1964). Additionally, there is a growing interest in optimization algorithms that can handle real-world inventory problems with non-stationary demand (Song et al., 2020).

We provide a Python package, *idinn*, which implements inventory dynamics-informed neural networks designed to control both single-sourcing and dual-sourcing problems. In single-sourcing problems, a single supplier delivers an ordered quantity to the firm within a known lead time (the time it takes for orders to arrive) and at a known unit cost (the cost of ordering a single item). In dual-sourcing problems, which are more complex, the company has two potential suppliers of a product, each with different known lead times and unit costs. The company's decision problem is to determine the quantity to order from each of the two suppliers at the beginning of each period, given the history of past orders and the current inventory level. The objective is to minimize the expected order, inventory, and out-of-stock costs over a finite or infinite horizon. *idinn* implements neural network controllers and inventory dynamics as customizable objects using PyTorch as the backend, allowing users to identify near-optimal ordering policies for their needs with reasonable computational resources.

The methods used in *idinn* take advantage of advances in automatic differentiation (Paszke et al., 2019, 2017) and the growing use of neural networks in dynamical system identification (Chen et al., 2018; Fronk & Petzold, 2023; Wang & Lin, 1998) and control (Asikis et al., 2022; Böttcher et al., 2022, 2024; Böttcher, 2023; Böttcher & Asikis, 2022; Mowlavi & Nabi, 2023).

## Statement of need

Inventory management problems arise in many industries, including manufacturing, retail, hospitality, fast fashion, warehousing, and energy. A fundamental but analytically intractable inventory management problem is dual sourcing (Barankin, 1961; Fukuda, 1964; Xin & Van Mieghem, 2023). *idinn* is a Python package for controlling dual-sourcing inventory dynamics with dynamics-informed neural networks. The classical dual-sourcing problem we consider is usually formulated as an infinite-horizon problem focusing on minimizing average cost while considering stationary stochastic demand. Using neural networks, we minimize costs over

multiple demand trajectories. This approach allows us to address not only non-stationary demand, but also finite-horizon and infinite-horizon discounted problems. Unlike traditional reinforcement-learning approaches, our optimization approach takes into account how the system to be optimized behaves over time, leading to more efficient training and accurate solutions.

Training neural networks for inventory dynamics control presents a unique challenge. The adjustment of neural network weights during training relies on propagating real-valued gradients, while the neural network outputs - representing replenishment orders - must be integers. To address this challenge in optimizing a discrete problem with real-valued gradient descent learning algorithms, we apply a problem-tailored straight-through estimator (Asikis, 2023; Dyer et al., 2023; Yang et al., 2022). This approach enables us to obtain integer-valued neural network outputs while backpropagating real-valued gradients.

idinn has been developed for researchers, industrial practitioners and students working at the intersection of optimization, operations research, and machine learning. It has been made available to students in a machine learning course at the Frankfurt School of Finance & Management, as well as in a tutorial at California State University, Northridge, showcasing the effectiveness of artificial neural networks in solving real-world optimization problems. In a previous publication (Böttcher et al., 2023), a proof-of-concept codebase was used to compute near-optimal solutions of dozens of dual-sourcing instances.

## Example usage

### Single-sourcing problems

The overarching goal in single-sourcing and related inventory management problems is for companies to identify the optimal order quantities to minimize inventory-related costs, given stochastic demand. During periods when inventory remains after demand is satisfied, each unit of excess inventory incurs a holding cost  $h$ . If demand exceeds available inventory in one period, the excess demand incurs an out-of-stock cost  $b$ . To solve this problem using idinn, we first initialize the sourcing model and its associated neural network controller. Then, we train the neural network controller using costs generated by the sourcing model. Finally, we can use the trained neural network controller to compute near-optimal order quantities that depend on the state of the system.

#### Initialization

We use the `SingleSourcingModel` class to initialize a single-sourcing model. The single-sourcing model considered in this example has a lead time of 0 (i.e., the order arrives immediately after it is placed) and an initial inventory of 10. The holding cost,  $h$ , and the out-of-stock cost,  $b$ , are 5 and 495, respectively. Demand is drawn from a discrete uniform distribution over the integers  $\{0, 1, \dots, 4\}$ . We use a batch size of 32 to train the neural network, i.e., the sourcing model generates 32 samples simultaneously. The sourcing model is initialized in code as follows.

```
import torch
from idinn.sourcing_model import SingleSourcingModel
from idinn.single_controller import SingleSourcingNeuralController
from idinn.demand import UniformDemand

single_sourcing_model = SingleSourcingModel(
    lead_time=0,
    holding_cost=5,
    shortage_cost=495,
    batch_size=32,
```

```

init_inventory=10,
demand_generator=UniformDemand(low=0, high=4)
)

```

In single-sourcing problems, three events occur within each period. First, the current inventory,  $I_t$ , and the history of past orders that have not yet arrived (i.e., the vector  $(q_{t-1}, q_{t-2}, \dots, q_{t-l})$ ) are used as inputs for the controller to calculate the order quantity,  $q_t$ . Second, the previous order quantity  $q_{t-l}$  arrives. Third, the demand for the current period,  $D_t$ , is realized, resulting in a new inventory level,  $I_t + q_{t-l} - D_t$ . Using the updated inventory, the cost for the individual period,  $c_t$ , is calculated according to

$$c_t = h \max(0, I_t) + b \max(0, -I_t),$$

where  $I_t$  is the inventory level at the end of period  $t$ . The higher the holding cost, the more costly it is to keep inventory positive and high. The higher the out-of-stock cost, the more costly it is to run out of stock when the inventory level is negative. The goal is to identify an ordering policy that minimizes total costs over a given time horizon. The interested reader is referred to Böttcher et al. (2023) for further details.

To control single-sourcing dynamics, we initialize a neural network controller using the `SingleSourcingNeuralController` class. For illustration purposes, we use a simple neural network with 1 hidden layer and 2 neurons. The activation function is `torch.nn.CELU(alpha=1)`.

```

single_controller = SingleSourcingNeuralController(
    hidden_layers=[2],
    activation=torch.nn.CELU(alpha=1)
)

```

#### 94 Training

We train the neural network controller using the `fit()` method, with training data generated from the considered sourcing model. To monitor the training process, we specify the `tensorboard_writer` parameter to log both the training loss and the validation loss. For reproducibility, we also specify the seed of the underlying random number generator using the `seed` parameter.

```

from torch.utils.tensorboard import SummaryWriter

single_controller.fit(
    sourcing_model=single_sourcing_model,
    sourcing_periods=50,
    validation_sourcing_periods=1000,
    epochs=2000,
    tensorboard_writer=SummaryWriter(comment="_single_1"),
    seed=1
)

```

To evaluate the neural network controller, we compute the average cost over a specified number of periods for the previously defined sourcing model. In the following example, the average cost is computed over 1,000 periods.

```
single_controller.get_average_cost(single_sourcing_model, sourcing_periods=1000)
```

For the selected single-sourcing parameters, the optimal average cost is 10.

#### 104 Order calculation

For a given inventory level and trained controller, we use the `predict` function to compute the corresponding orders. In the following example, we set the current inventory level to 10.

```
single_controller.predict(current_inventory=10)
```

107 This function returns the order quantity under the `single_controller` policy, given a current  
108 inventory level of 10.

#### 109 Base-stock controller

110 In addition to the neural network control method, single-sourcing dynamics can also be  
111 managed using a traditional base-stock controller (Arrow et al., 1951; Scarf & Karlin, 1958).  
112 This approach provides a useful baseline for comparison and is often employed in inventory  
113 management due to its simplicity and interpretability.

114 The example below demonstrates how to initialize, train, and evaluate a base-stock controller  
115 using the same single-sourcing model.

```
from idinn.single_controller import BaseStockController

single_controller_base = BaseStockController()
single_controller_base.fit(single_sourcing_model)
single_controller_base.get_average_cost(single_sourcing_model, sourcing_periods=1000)
```

116 As with the neural network controller, order decisions can be made using the `predict` function.

```
single_controller_base.predict(current_inventory=10)
```

117 This function returns the optimal order quantity under the base-stock policy, given a current  
118 inventory level of 10.

#### 119 Dual-sourcing problems

120 Solving dual-sourcing problems with `idinn` follows a similar workflow to that of single-sourcing  
121 problems, as described in the previous section. The key difference is that the cost calculation  
122 accounts for the ordering costs from two distinct suppliers.

#### 123 Initialization

124 To solve dual-sourcing problems, we use the `DualSourcingModel` and `DualSourcingNeuralController`  
125 classes, which define the sourcing model and its associated controller. In the example below,  
126 we examine a dual-sourcing model characterized by the following parameters: the regular order  
127 lead time is 2; the expedited order lead time is 0; the regular order cost,  $c_r$ , is 0; the expedited  
128 order cost,  $c_e$ , is 20; and the initial inventory is 6. Additionally, the holding cost,  $h$ , and the  
129 out-of-stock cost,  $b$ , are 5 and 495, respectively. Demand is drawn from a discrete uniform  
130 distribution over the integers  $\{0, 1, \dots, 4\}$  and the batch size is 256.

```
import torch
from idinn.sourcing_model import DualSourcingModel
from idinn.dual_controller import DualSourcingNeuralController
from idinn.demand import UniformDemand

dual_sourcing_model = DualSourcingModel(
    regular_lead_time=2,
    expedited_lead_time=0,
    regular_order_cost=0,
    expedited_order_cost=20,
    holding_cost=5,
    shortage_cost=495,
    batch_size=256,
    init_inventory=6,
```

```
demand_generator=UniformDemand(low=0, high=4)
)
```

131 In dual-sourcing dynamics, the cost in period  $t$ ,  $c_t$ , is

$$c_t = c_r q_t^r + c_e q_t^e + h \max(0, I_t) + b \max(0, -I_t),$$

132 where  $I_t$  is the inventory level at the end of period  $t$ ,  $q_t^r$  is the regular order placed in period  
 133  $t$ , and  $q_t^e$  is the expedited order placed in period  $t$ . The higher the holding cost, the more  
 134 expensive it is to keep inventory positive and high. The higher the out-of-stock cost, the more  
 135 expensive it is to run out of stock when inventory is negative. The higher the regular and  
 136 expedited order costs, the more expensive it is to place those orders.

137 To control dual-sourcing dynamics, we initialize a neural network controller using the  
 138 DualSourcingNeuralController class. We use a simple neural network with six hidden layers.  
 139 The number of neurons in each layer is 128, 64, 32, 16, 8, and 4, respectively. The activation  
 140 function is torch.nn.CELU(alpha=1).

```
dual_controller = DualSourcingNeuralController(
    hidden_layers=[128, 64, 32, 16, 8, 4],
    activation=torch.nn.CELU(alpha=1)
)
```

141 The inputs to the controller are the inventory level,  $I_t$ , and the history of past orders. Since  
 142 there are now two suppliers in the system, we need to include the order history of both suppliers.  
 143 Therefore, the inputs associated with the past orders are  $(q_{t-1}^r, \dots, q_{t-l_r}^r, q_{t-1}^e, \dots, q_{t-l_e}^e)$ . The  
 144 cost for each period is calculated similarly to the single-sourcing model: past orders arrive,  
 145 new orders are placed, and demand is realized. The objective remains to identify an ordering  
 146 policy that minimizes total costs over a given time horizon. The interested reader is referred  
 147 to Böttcher et al. (2023) for more details.

#### 148 Training

149 As in the previous section, we train the neural network controller using the fit() method.

```
from torch.utils.tensorboard import SummaryWriter

dual_controller.fit(
    sourcing_model=dual_sourcing_model,
    sourcing_periods=100,
    validation_sourcing_periods=1000,
    epochs=2000,
    tensorboard_writer=SummaryWriter(comment="dual"),
    seed=123
)
```

150 To evaluate the neural network controller, we compute the average cost over a specified number  
 151 of periods using the previously defined sourcing model. In the example below, the average cost  
 152 is computed over 1,000 periods.

```
dual_controller.get_average_cost(dual_sourcing_model, sourcing_periods=1000)
```

153 For the selected dual-sourcing parameters, the optimal average cost is 23.07.

#### 154 Order calculation

155 For a given dual-sourcing controller, orders can be computed as follows.

```
dual_controller.predict(current_inventory=10, past_regular_orders=[1, 1], past_expedited
```

156 If the regular and expedited lead-time values are greater than 0, one has to specify the  
157 corresponding `past_regular_orders` and `past_expedited_orders`.

#### 158 Other dual-sourcing controllers

159 In addition to the neural network control method, dual-sourcing dynamics can also be managed  
160 using capped dual index (Sun & Van Mieghem, 2019) and dynamic programming controllers.  
161 These methods offer valuable baselines for comparison. The example below illustrates how to  
162 initialize and train these controllers using the same dual-sourcing model.

```
from idinn.dual_controller import CappedDualIndexController, DynamicProgrammingController

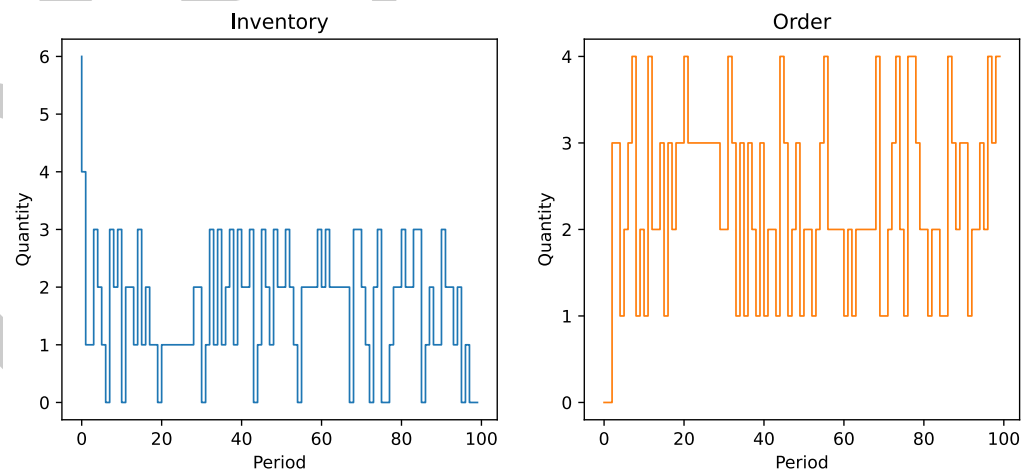
dual_controller_cdi = CappedDualIndexController()
dual_controller_cdi.fit(
    dual_sourcing_model,
    sourcing_periods=100
)

dual_controller_dp = DynamicProgrammingController()
dual_controller_dp.fit(
    dual_sourcing_model,
    max_iterations=10000,
    tolerance=1e-6
)
```

163 As with the dual-sourcing neural network controller, average costs and order quantities can be  
164 computed using the `.get_average_cost()` and `.predict()` methods, respectively.

#### 165 Other utility functions

166 The `idinn` package includes several utility functions for both the `SingleSourcingModel` and  
167 `DualSourcingModel` classes.



**Figure 1:** Evolution of inventory and orders for a neural network controller applied to single-sourcing dynamics.

168 To further evaluate a controller's performance in a given sourcing environment, users can  
169 visualize the inventory and order histories (see Figure 1).

```
single_controller.plot(sourcing_model=single_sourcing_model, sourcing_periods=100)
```

In addition to the discrete uniform demand distributions used in the previous sections, custom demand distributions can also be defined using the CustomDemand class together with a demand\_generator. The CustomDemand class accepts a dictionary specifying possible demand values and their associated probabilities. We show an example of a dual-sourcing model with custom demand distribution below.

```
from idinn.demand import CustomDemand

dual_sourcing_model = DualSourcingModel(
    regular_lead_time=2,
    expedited_lead_time=0,
    regular_order_cost=0,
    expedited_order_cost=20,
    holding_cost=5,
    shortage_cost=495,
    init_inventory=0,
    demand_generator=CustomDemand({5: 0.02, 6: 0.9, 7: 0.02, 8: 0.02, 9: 0.02, 10: 0.02})
)
```

In this dual-sourcing model, there is a 90% probability that the demand will be 6, and a 2% probability that the demand will be either 5, 7, 8, 9, or 10, respectively. The CustomDemand generator allows users to input demands customized to their specific requirements.

The idinn package also provides functions for saving and loading model checkpoints. To save and load a given model, one can use the save() and load() methods, respectively.

```
# Save the model
dual_controller.save("optimal_dual_sourcing_controller.pt")
# Load the model
dual_controller_loaded = DualSourcingNeuralController(
    hidden_layers=[128, 64, 32, 16, 8, 4],
    activation=torch.nn.CELU(alpha=1)
)
dual_controller_loaded.init_layers(
    regular_lead_time=2,
    expedited_lead_time=0
)
dual_controller_loaded.load("optimal_dual_sourcing_controller.pt")
```

For controllers that are not based on neural networks, Python's pickle module can be used for saving and loading. Further details are provided in the official idinn documentation.

## Acknowledgements

LB acknowledges financial support from hessian.AI and the Army Research Office (grant W911NF-23-1-0129). TA acknowledges financial support from the Schweizerischer Nationalfonds zur Förderung der Wissenschaftlichen Forschung through NCCR Automation (grant P2EZP2 191888).

## References

- Arrow, K. J., Harris, T., & Marschak, J. (1951). Optimal inventory policy. *Econometrica*, 19(3), 250–272.
- Asikis, T. (2023). Towards recommendations for value sensitive sustainable consumption. *NeurIPS 2023 Workshop on Tackling Climate Change with Machine Learning: Blending*



- 192 New and Existing Knowledge Systems. <https://nips.cc/virtual/2023/76939>
- 193 Asikis, T., Böttcher, L., & Antulov-Fantulin, N. (2022). Neural ordinary differential equation  
194 control of dynamics on graphs. *Physical Review Research*, 4(1), 013221.
- 195 Barankin, E. (1961). A delivery-lag inventory model with an emergency provision. *Naval*  
196 *Research Logistics Quarterly*, 8, 285–311.
- 197 Böttcher, L. (2023). Gradient-free training of neural ODEs for system identification and control  
198 using ensemble Kalman inversion. *ICML Workshop on New Frontiers in Learning, Control,*  
199 *and Dynamical Systems, Honolulu, HI, USA, 2023.*
- 200 Böttcher, L., Antulov-Fantulin, N., & Asikis, T. (2022). AI Pontryagin or how artificial neural  
201 networks learn to control dynamical systems. *Nature Communications*, 13(1), 1–9.
- 202 Böttcher, L., & Asikis, T. (2022). Near-optimal control of dynamical systems with neural  
203 ordinary differential equations. *Machine Learning: Science and Technology*, 3(4), 045004.
- 204 Böttcher, L., Asikis, T., & Fragkos, I. (2023). Control of dual-sourcing inventory systems  
205 using recurrent neural networks. *INFORMS Journal on Computing*, 35(6), 1308–1328.
- 206 Böttcher, L., Fonseca, L. L., & Laubenbacher, R. C. (2024). Control of medical digital twins  
207 with artificial neural networks. *arXiv Preprint arXiv:2403.13851.*
- 208 Chen, T. Q., Rubanova, Y., Bettencourt, J., & Duvenaud, D. (2018). Neural ordinary  
209 differential equations. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N.  
210 Cesa-Bianchi, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems*  
211 *31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018,*  
212 *December 3-8, 2018, Montréal, Canada* (pp. 6572–6583). [https://proceedings.neurips.cc/  
213 paper/2018/hash/69386f6bb1dfed68692a24c8686939b9-Abstract.html](https://proceedings.neurips.cc/paper/2018/hash/69386f6bb1dfed68692a24c8686939b9-Abstract.html)
- 214 Dyer, J., Quera-Bofarull, A., Chopra, A., Farmer, J. D., Calinescu, A., & Wooldridge, M.  
215 J. (2023). Gradient-assisted calibration for financial agent-based models. *4th ACM*  
216 *International Conference on AI in Finance, ICAIF 2023, Brooklyn, NY, USA, November*  
217 *27-29, 2023*, 288–296.
- 218 Fronk, C., & Petzold, L. (2023). Interpretable polynomial neural ordinary differential equations.  
219 *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 33(4).
- 220 Fukuda, Y. (1964). Optimal policies for the inventory problem with negotiable leadtime.  
221 *Management Science*, 10(4), 690–708.
- 222 Mowlavi, S., & Nabi, S. (2023). Optimal control of PDEs using physics-informed neural  
223 networks. *Journal of Computational Physics*, 473, 111731.
- 224 Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A.,  
225 Antiga, L., & Lerer, A. (2017). Automatic differentiation in PyTorch. *NIPS 2017 Autodiff*  
226 *Workshop.*
- 227 Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin,  
228 Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E. Z., DeVito, Z., Rai-  
229 son, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019).  
230 PyTorch: An imperative style, high-performance deep learning library. In H. M. Wal-  
231 lach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, & R. Garnett (Eds.),  
232 *Advances in Neural Information Processing Systems 32: Annual Conference on Neural*  
233 *Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancou-*  
234 *ver, BC, Canada* (pp. 8024–8035). [https://proceedings.neurips.cc/paper/2019/hash/  
235 bdbca288fee7f92f2bfa9f7012727740-Abstract.html](https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html)
- 236 Scarf, H., & Karlin, S. (1958). Inventory models of the arrow-harris-marschak type with time  
237 lag. In K. J. Arrow, S. Karlin, & H. E. Scarf (Eds.), *Studies in the mathematical theory of*  
238 *inventory and production.* Stanford University Press.



- 239 Song, J.-S., Van Houtum, G.-J., & Van Mieghem, J. A. (2020). Capacity and inventory  
240 management: Review, trends, and projections. *Manufacturing & Service Operations*  
241 *Management*, 22(1), 36–46.
- 242 Sun, J., & Van Mieghem, J. A. (2019). Robust dual sourcing inventory management: Optimality  
243 of capped dual index policies and smoothing. *Manufacturing & Service Operations*  
244 *Management*, 21(4), 912–931.
- 245 Wang, Y.-J., & Lin, C.-T. (1998). Runge–Kutta neural network for identification of dynamical  
246 systems in high accuracy. *IEEE Transactions on Neural Networks*, 9(2), 294–307.
- 247 Xin, L., & Van Mieghem, J. A. (2023). Dual-sourcing, dual-mode dynamic stochastic inventory  
248 models. In *Research Handbook on Inventory Management* (pp. 165–190). Edward Elgar  
249 Publishing.
- 250 Yang, Z., Lee, J., & Park, C. (2022). Injecting logical constraints into neural networks via  
251 straight-through estimators. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G.  
252 Niu, & S. Sabato (Eds.), *International Conference on Machine Learning, ICML 2022,*  
253 *17-23 July 2022, Baltimore, Maryland, USA* (Vol. 162, pp. 25096–25122). PMLR.  
254 <https://proceedings.mlr.press/v162/yang22h.html>

DRAFT