BatchGFN: Generative Flow Networks for Batch Active Learning

Shreshth A. Malik[1] **Salem Lahlou**[2 3] **Andrew Jesson**[1] **Moksh Jain**[2 3] **Nikolay Malkin**[2 3] **Tristan Deleu**[2 3] **Yoshua Bengio**[2 3 4] **Yarin Gal**[1]

Abstract

We introduce BatchGFN—a novel approach for pool-based active learning that uses generative flow networks to sample sets of data points proportional to a batch reward. With an appropriate reward function to quantify the utility of acquiring a batch, such as the joint mutual information between the batch and the model parameters, BatchGFN is able to construct highly informative batches for active learning in a principled way. We show our approach enables sampling near-optimal utility batches at inference time with a single forward pass per point in the batch in toy regression problems. This alleviates the computational complexity of batch-aware algorithms and removes the need for greedy approximations to find maximizers for the batch reward. We also present early results for amortizing training across acquisition steps, which will enable scaling to real-world tasks.

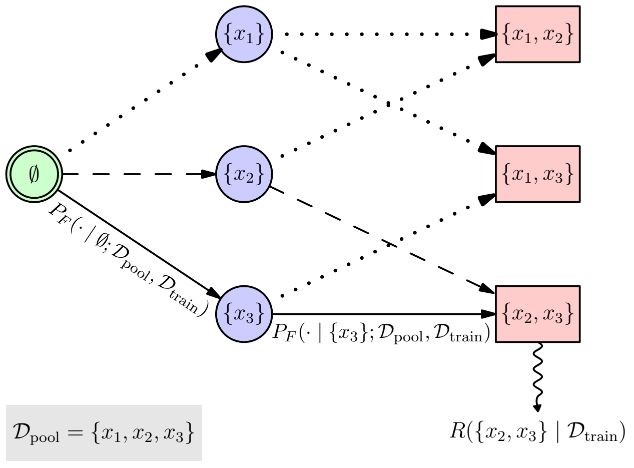
$\emptyset$
$\{x_1\}$
$\{x_2\}$
$\{x_3\}$
$\{x_1, x_2\}$
$\{x_1, x_3\}$
$\{x_2, x_3\}$
$P_F(\cdot \mid \emptyset; \mathcal{D}_{\text{pool}}, \mathcal{D}_{\text{train}})$
$P_F(\cdot \mid \{x_3\}; \mathcal{D}_{\text{pool}}, \mathcal{D}_{\text{train}})$
$R(\{x_2, x_3\} \mid \mathcal{D}_{\text{train}})$
$\mathcal{D}_{\text{pool}} = \{x_1, x_2, x_3\}$

Figure 1. BatchGFN state space for constructing a batch of size 2 from a pool set of 3 points. Starting with an empty set, the policy sequentially chooses points to add to batch until a $\boxed{\text{terminal state}}$, representing a complete batch, is reached. Notice that a state may be reachable by more than one trajectory (e.g., the dashed path).

1. Introduction

Annotating large quantities of data can be prohibitively expensive, particularly in domains which require expertise. Active learning (AL) seeks to overcome this labelling bottleneck by iteratively selecting the most useful points to label to improve model performance (Houlsby et al., 2011; Settles, 2009). In the batch setting, we seek to choose a *set* of points (the query batch) from an unlabelled pool at each acquisition step. Approaches based on single-point scores tend to have redundancy in information in the batch (Gal et al., 2017). Stochastic schemes (Kirsch et al., 2021) can alleviate some of this redundancy but do not have guarantees for optimal batch formation. However, more principled batch-aware methods are in general computationally expensive and often resort to greedy approximations to their true objective to construct the batch (Ash et al., 2021; 2019; Holzmüller et al., 2022; Kirsch et al., 2019; Sener & Savarese, 2018).

In this work, we propose to select batches using a parameterized sampler trained to sample batches proportional to a batch reward function. We use a generative flow network (Bengio et al., 2021a;b) where the state of the network is the query batch under construction, and actions add pool data points to the batch. Once trained, we show that our approach enables sampling highly informative query batches more efficiently than other batch-aware algorithms, without using greedy approximations.

2. Background

2.1. Batch Active Learning

The goal of AL is to train models with as little data as possible. In the pool-based batch AL framework (Lewis, 1995; Settles, 2009), at each data acquisition step, we seek to choose B points from a pool set $\mathcal{D}_{\text{pool}} = \{x_i\}_{i=1}^{N}$ to be labelled by an oracle and added to the training set $\mathcal{D}_{\text{train}} = \{x_i, y_i\}_{i=1}^{M}$, where $x \in \mathcal{X}$ are the data features

[1]OATML, University of Oxford [2]Mila – Québec AI Institute [3]Université de Montréal [4]CIFAR Fellow. Correspondence to: Shreshth A. Malik <shreshth@robots.ox.ac.uk>.

Accepted to ICML workshop on Structured Probabilistic Inference & Generative Modeling. Copyright 2023 by the author(s).

and $y \in \mathcal{Y}$ are the corresponding labels. We can formalize batch AL strategies by defining an *acquisition* or *reward* function $R : \mathcal{X}^B \rightarrow \mathbb{R}$ which scores each potential batch. As the number of batches to score grows exponentially with $|\mathcal{D}_{\text{pool}}|$, the maximization problem is intractable, so greedy approximations are often used.

For example, the BALD (Gal et al., 2017; Houlsby et al., 2011) algorithm takes a Bayesian perspective for a model with parameters θ and selects points that maximize the mutual information (MI) between the model predictions and its parameters, $\mathbb{I}[y, \theta \mid x, \mathcal{D}_{\text{train}}] = \mathbb{H}[y \mid x, \mathcal{D}_{\text{train}}] - \mathbb{E}_{\theta \sim p(\theta|\mathcal{D}_{\text{train}})}[\mathbb{H}[y \mid x, \theta]]$, where $\mathbb{H}$ denotes the entropy. Intuitively, labelling points with high mutual information will decrease the uncertainty in the model parameters. The batch reward for BALD is simply the sum of the individual scores for each data point $R_{\text{BALD}} = \sum_{i=1}^{B} \mathbb{I}[y_i, \theta \mid x_i, \mathcal{D}]$, so the top B scoring points are greedily selected. BALD has been shown to be ineffective to acquire batches, given that the maximizers of MI are usually similar. Injecting noise into top-B acquisition scores such as BALD can be used to induce diversity in the batch, and can be viewed as approximating future acquisition scores (Kirsch et al., 2021).

Kirsch et al. (2019) explicitly model the interactions between data points by directly using the *joint* mutual information (JMI),

$$R_{\text{BatchBALD}} = \mathbb{I}[y_{1:B}, \theta \mid x_{1:B}, \mathcal{D}_{\text{train}}] \,. \qquad (1)$$

To deal with the intractability of the maximization problem, the authors propose a greedy approximation that is guaranteed to yield a batch for which the JMI is larger than $(1 - \frac{1}{e})$ times the optimal JMI, because of sub-modularity of the JMI set function. Greedy strategies for maximizing JMI in the regression setting have also been proposed (Holzmüller et al., 2022; Wang et al., 2021). Related to the JMI (Kirsch & Gal, 2022), Fisher Information has also been explored as a batch acquisition objective (Ash et al., 2021). Other AL strategies such as Coresets (Sener & Savarese, 2018) or LCMD (Holzmüller et al., 2022), which use distance-based metrics to ensure diversity in the batch, also use greedy approximations to find a batch with high reward. In addition to the approximation, another major drawback of BatchBALD, and other batch-aware methods such as BADGE (Ash et al., 2019), is their high computational cost.

2.2. Generative Flow Networks

Generative Flow Networks (GFlowNets; GFNs; Bengio et al., 2021a;b) are probabilistic models over discrete sample spaces with a compositional structure. GFNs are stochastic sequential samplers that aim to generate objects from a target distribution, which is given by its unnormalized probability mass function R, also referred to as the *reward* function.

The sample space, denoted $\mathcal{S}^f$, is the subset of terminal nodes (i.e. have no outgoing edges) of the vertices $\mathcal{S}$ of a directed acyclic graph (DAG) $=\{\mathcal{S}, \mathbb{A}\}$ with a special parentless state s_0 called the source state. $\mathcal{S}$ consists of partially constructed objects, s_0 being an empty object, and $\mathbb{A}$ corresponds to actions that can be taken at each of these states. Complete objects $s_f \in \mathcal{S}^f$ are sampled by following a complete trajectory τ starting at s_0 and terminating at s_f.

GFN training objectives allow the learning of a *policy* $P_F(s' \mid s)$ along the edges of the DAG with the goal of making the marginal likelihood of sampling s_f proportional to the reward $R(s_f)$. The parameters of P_F are sequentially updated using the gradient of one of the losses applied to trajectories (or parts of trajectories) sampled from the trajectory distribution induced by P_F (or some exploratory distribution, such as a tempered P_F). The various GFN losses in common use make use of a parametric *backward policy* $P_B(s \mid s')$, specifying distributions over parents of the states in the DAG, and optionally a *state flow* function $F(s)$ (Malkin et al., 2022). In this work, we use the Subtrajectory Balance objective (Madan et al., 2023, to appear.) which provides advantages in training stability. We also leverage the forward-looking parametrization (Pan et al., 2023, to appear.), which uses the stepwise gain in a proxy log-reward computed at intermediate states to improve credit assignment.

The parametric objects learned by GFNs, as well as their rewards, can be conditioned on instance-specific information, in our case a training set and a pool set, enabling generalization to conditioning data not seen in training (Jain et al., 2022b; Zhang et al., 2023).

3. Methods

3.1. BatchGFN: A Sampler for Batches of Data

We propose BatchGFN; a parameterized sampler for batch AL which uses a GFN trained to sample informative query batches of data to label. Prior work on GFNs in the context of active learning does not consider the pool-based setting and instead leverages the GFNs to sample individual candidates that comprise the batch, one at a time (Jain et al., 2022a).

Instead, we use a GFN to construct a batch (a set) of candidates from a pool in a single trajectory. As illustrated in Figure 1, P_F generates a batch of size B through a sequence of steps, each consisting of adding an element from $\mathcal{D}_{\text{pool}}$ to the partially constructed batch. Note that the GFN is conditioned on $\mathcal{D}_{\text{train}}$. Each sampled batch $\{x_1, \ldots, x_B\}$ is scored with a reward function $R(\cdot|\mathcal{D}_{\text{train}})$ which quantifies the utility of acquiring the batch. In this work we use the

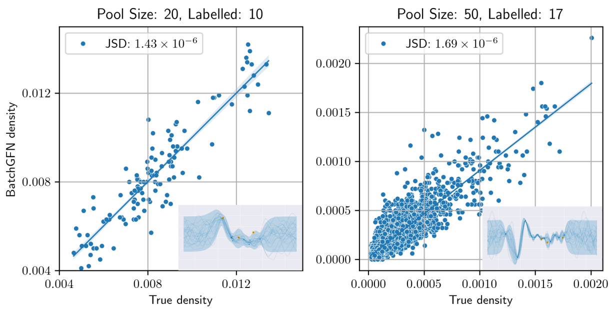

Figure 2. Density parity plots comparing the true reward density for query batches against the empirical distribution of batches sampled from the BatchGFN. Regression lines are also shown. A perfect fit would have all points lying on $y = x$. The inlays shows the current model and labelled/queried data points in blue/orange respectively, and the Jenson-Shannon divergence between the two distributions.

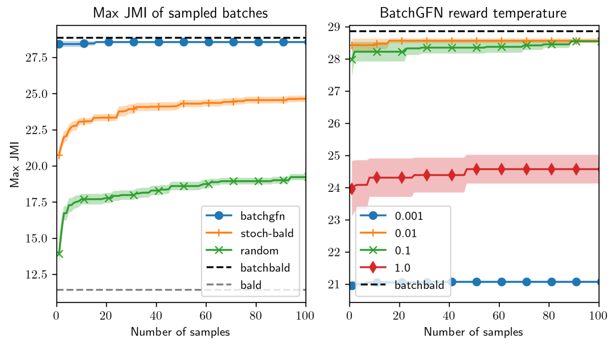

Figure 3. Joint mutual information of sampled query batches from BatchGFN compared to baselines for the 1D regression task with pool size 2000, seed size 10, query size 10, and $T = 0.01$. Uncertainty bars show the standard error over 10 sampled runs. The stochastic-BALD baseline samples from a distribution of single-point BALD scores. The right plot compares BatchGFN sampling at different reward temperatures. We note that very low temperatures cause training instabilities. Conditioning the policy on the temperature may alleviate this behaviour (Zhang et al., 2023).

JMI (1) to account for overlap in information between points in the batch. We can however, in principle, use *any* heuristic that provides a scalar reward to a given batch.

With sufficient capacity, the BatchGFN converges to the true reward distribution in the limit of infinite training trajectories (Bengio et al., 2021a). Therefore, once appropriately trained, BatchGFN can be used to sample batches for AL from the *true batch objective* (1) efficiently, without resorting to greedy, or stochastic top-B approximations. BatchGFN has a time complexity of $\mathcal{O}(B)$ for sampling a batch, requiring only B forward passes of P_F. This is cheap compared to, for example, BatchBALD which requires computing joint entropies for all points in the pool which can be particularly expensive when using Monte-Carlo (MC) samples (Kirsch et al., 2019).

3.2. Amortizing Training Across Acquisition Steps

The reward distribution over possible query batches changes after each acquisition as the model is trained on the newly labelled data. Naïvely, the GFN therefore needs to be retrained to fit the new distribution at each AL step, which can be expensive. In theory, we could train models on samples from a distribution of $\mathcal{D}_{\text{train}}$ and and use these as examples to train an amortized P_F.

Instead, we draw on ideas from GP "fantasization" literature (Hennig & Schuler, 2012; Jiang et al., 2020; Maddox et al., 2021) to "lookahead" to possible future reward distributions after acquisition. We use samples from the current model to hallucinate labels for the next chosen query batch, add these to the training set, retrain the model, and then train the sampler using the new reward function and training set. This greatly restricts the space of conditional reward distributions required to be modelled and thus the computational expense required for training. See Appendix A.3 for further details on lookahead training.

3.3. Implementation Details

Batch Reward Function and Active Learning Model In the experiments shown here we chose to use the JMI (1) as the measure of utility for the constructed batches. We use exact inference Gaussian processes (GPs) as the model for AL. This allows us to use a closed form solution for the JMI between batch labels $y_{1:B}$ and the model f,

$$\mathbb{I}\left(y_{1:B}; f\right) = \mathbb{I}\left(y_{1:B}; f_{1:B}\right) = \frac{1}{2}\log\left|\boldsymbol{I} + \sigma^{-2}\boldsymbol{K}_{1:B}\right| \quad (2)$$

where $\boldsymbol{K}_{1:B} = \left[k\left(x, x'\right)\right] x, x' \in x_{1:B}$ is the covariance matrix for the batch, σ^2 is the GP observation noise variance, and $\boldsymbol{I}$ is the identity matrix (Holzmüller et al., 2022; Srinivas et al., 2010). For AL we would like to preferentially sample batches with high JMI. Thus in practice, we modify the reward function to $R = \exp(\mathbb{I}\left(y_{1:B}; f\right)/T)$, which includes a temperature parameter T that enables sampling from a more peaky reward distribution to focus on the modes.

Policy Network Architecture and Training Note that our framework is agnostic to policy parametrization. For experiments presented here we use a simple set-invariant architecture which is conditioned on the current batch and the training data (Appendix A.2). We use the Subtrajectory Balance objective with forward-looking parametrization which exploits the submodularity of the JMI objective for intermediate rewards. During training, we encourage state exploration by sampling from an ϵ-random policy (Bengio et al.,

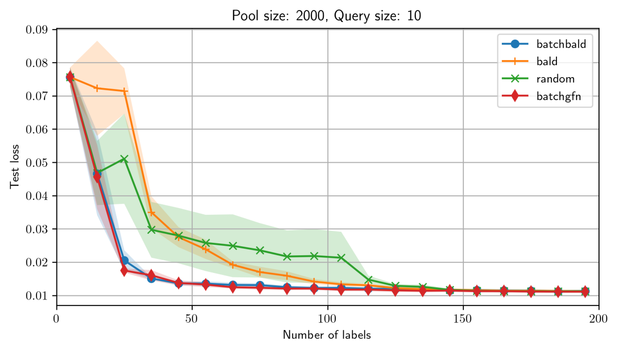

Figure 4. Test loss as function of number of labelled examples for different active learning strategies. Active learning with BatchGFN is on par with BatchBALD. Uncertainty bars show the standard error over 5 runs with different seed sets.

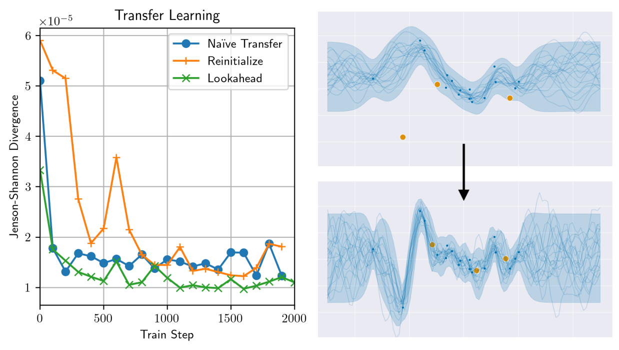

Figure 5. Training curve plot showing the Jenson-Shannon divergence between the true reward distribution and the empirical BatchGFN distribution when transferring from one acquisition step to the next (shown on the right). We train on 10 hallucinated samples of the query batch to be labelled.

2021a)[1].

4. Experiments

First we show through a toy example that the BatchGFN converges to a policy that samples batches of points proportional to their reward as expected. Then we show that by tuning the reward temperature, we can efficiently sample highly informative batches for AL. Finally, we present early results into amortizing training across acquisition steps.

4.1. Sampling Proportional to the Batch Reward

We consider a toy 1D regression task to verify behaviour. See Appendix A.1 for details on the dataset. For small pool and query sizes, it is possible to exhaustively evaluate the reward for every possible query batch. We compare the true reward distribution to the empirical distribution of batches generated by the trained BatchGFN (with $T = 1$) in Figure 2. We see that the BatchGFN samples batches approximately proportional to their batch reward as expected.

4.2. Sampling High Joint Mutual Information Batches

In practice, we would like to sample high JMI batches for applications such as AL. Figure 3 shows the JMI of sampled batches from the BatchGFN at different reward temperatures for the 1D regression task. We find that by decreasing the temperature we are able to sample higher JMI batches with greater sample efficiency compared to other stochastic approaches. The batches sampled are on par with BatchBALD while being less computationally expensive[2].

4.3. Active Learning with BatchGFN

We have shown that BatchGFN can sample high JMI batches effectively. Now we assess how effective these are for AL. Figure 4 shows the test loss of the model at the toy regression task using queries sampled from BatchGFN. We sample 20 trajectories from the BatchGFN at inference and choose the batch with the highest reward. We find that BatchGFN is on par with BatchBALD and significantly outperforms BALD and random acquisition. Appendix B.1 contains example acquisition plots.

4.4. Amortizing Across Acquisition Steps

Retraining the BatchGFN between acquisition steps is impractical for real-world usage. As discussed in Section 3.2, we can amortize training over acquisition steps by conditioning the policy on the training data. In Figure 5 we compare how quickly the BatchGFN adapts to the new conditional reward distribution after an acquisition on a small pool/query size such that we can compute the true distribution (Section 4.1). We find that lookahead training significantly decreases the number of training steps required to fit to the new distribution compared to reinitializing the policy network. Naïvely continuing training from the previous reward distribution also works well but starts at higher divergence as it is overfit to the previous distribution, making it unsuitable for zero-shot transfer, and potentially unstable for few-shot transfer. This shows promise for amortizing BatchGFN training over multiple acquisition steps and for larger tasks.

[1]Our code is available at `https://github.com/s-a-malik/batchgfn`

[2]We note that the speed-up in run-time compared to BatchBALD using the exact GP JMI (2) in the current setting is modest ($\sim 10\%$ faster for a single sample from BatchGFN). However, BatchBALD is orders of magnitude more expensive when using MC sampling, whereas BatchGFN inference will be unaffected.

5. Discussion

Contributions In this work we developed BatchGFN—a novel method based on GFNs to sample batches of data points proportional to an arbitrary batch reward function. We have shown that this method can be used to efficiently sample high JMI batches for AL tasks in toy regression problems and has performance on par with more expensive batch construction methods, without resorting to greedy approximations.

Further Work In Section 4.4 we showed that it is possible to amortize training. However for practical usage, we need to show that this is possible over longer acquisition step horizons, larger pool sets, and higher dimensional data. To do this, we can investigate the following: 1) Architecturally, we can improve the representational ability of the policy network by considering attention between data points (Kossen et al., 2021). 2) To scale to larger pool sizes, we could incorporate perceiver-like bottlenecks (Jaegle et al., 2021). 3) To enable training on tasks where a cheap JMI estimate is not available, we should investigate alternative batch heuristics to train the GFN efficiently.

Acknowledgements

We would like to thank Panagiotis Tigas and Andreas Kirsch for helpful discussions about the project. SM acknowledges funding from the EPSRC Centre for Doctoral Training in Autonomous Intelligent Machines and Systems (Grant No: EP/S024050/1). YG acknowledges funding from the Turing Fellowship (Grant No. EP/V030302/1). MJ, NM, TD, and YB acknowledge financial support from IBM, Samsung, Microsoft, and Google.

References

Ash, J., Goel, S., Krishnamurthy, A., and Kakade, S. Gone fishing: Neural active learning with fisher embeddings. *Advances in Neural Information Processing Systems*, 34: 8927–8939, 2021.

Ash, J. T., Zhang, C., Krishnamurthy, A., Langford, J., and Agarwal, A. Deep batch active learning by diverse, uncertain gradient lower bounds. *arXiv preprint arXiv:1906.03671*, 2019.

Bengio, E., Jain, M., Korablyov, M., Precup, D., and Bengio, Y. Flow network based generative models for non-iterative diverse candidate generation. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021a. URL https://openreview.net/forum?id=Arn2E4IRjEB.

Bengio, Y., Deleu, T., Hu, E. J., Lahlou, S., Tiwari, M., and Bengio, E. Gflownet foundations, 2021b.

Gal, Y., Islam, R., and Ghahramani, Z. Deep bayesian active learning with image data. In *International Conference on Machine Learning*, pp. 1183–1192. PMLR, 2017.

Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., and Wilson, A. G. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. *Advances in neural information processing systems*, 31, 2018.

Hennig, P. and Schuler, C. J. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(6), 2012.

Holzmüller, D., Zaverkin, V., Kästner, J., and Steinwart, I. A framework and benchmark for deep batch active learning for regression. *arXiv preprint arXiv:2203.09410*, 2022.

Houlsby, N., Huszár, F., Ghahramani, Z., and Lengyel, M. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv: Arxiv-1112.5745*, 2011.

Jaegle, A., Gimeno, F., Brock, A., Vinyals, O., Zisserman, A., and Carreira, J. Perceiver: General perception with iterative attention. In *International conference on machine learning*, pp. 4651–4664. PMLR, 2021.

Jain, M., Bengio, E., Hernandez-Garcia, A., Rector-Brooks, J., Dossou, B. F., Ekbote, C. A., Fu, J., Zhang, T., Kilgour, M., Zhang, D., et al. Biological sequence design with gflownets. In *International Conference on Machine Learning*, pp. 9786–9801. PMLR, 2022a.

Jain, M., Raparthy, S. C., Hernandez-Garcia, A., Rector-Brooks, J., Bengio, Y., Miret, S., and Bengio, E. Multi-objective gflownets. *arXiv preprint arXiv:2210.12765*, 2022b.

Jiang, S., Jiang, D., Balandat, M., Karrer, B., Gardner, J., and Garnett, R. Efficient nonmyopic bayesian optimization via one-shot multi-step trees. *Advances in Neural Information Processing Systems*, 33:18039–18049, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kirsch, A. and Gal, Y. Unifying approaches in active learning and active sampling via fisher information and information-theoretic quantities. *Transactions on Machine Learning Research*, 2022.

Kirsch, A., Van Amersfoort, J., and Gal, Y. Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning. *Advances in neural information processing systems*, 32, 2019.

Kirsch, A., Farquhar, S., Atighehchian, P., Jesson, A., Branchaud-Charron, F., and Gal, Y. Stochastic batch acquisition for deep active learning. *arXiv preprint arXiv:2106.12059*, 2021.

Kossen, J., Band, N., Lyle, C., Gomez, A. N., Rainforth, T., and Gal, Y. Self-attention between datapoints: Going beyond individual input-output pairs in deep learning. *Advances in Neural Information Processing Systems*, 34: 28742–28756, 2021.

Lahlou, S., Viviano, J. D., and Schmidt, V. torchgfn: A pytorch gflownet library. *arXiv preprint arXiv: 2305.14594*, 2023.

Lewis, D. D. A sequential algorithm for training text classifiers: Corrigendum and additional data. In *Acm Sigir Forum*, volume 29, pp. 13–19. ACM New York, NY, USA, 1995.

Madan, K., Rector-Brooks, J., Korablyov, M., Bengio, E., Jain, M., Nica, A., Bosc, T., Bengio, Y., and Malkin, N. Learning gflownets from partial episodes for improved convergence and stability. *International Conference on Machine Learning (ICML)*, 2023, to appear.

Maddox, W. J., Stanton, S., and Wilson, A. G. Conditioning sparse variational gaussian processes for online decision-making. *Advances in Neural Information Processing Systems*, 34:6365–6379, 2021.

Malkin, N., Jain, M., Bengio, E., Sun, C., and Bengio, Y. Trajectory balance: Improved credit assignment in gflownets. *Neural Information Processing Systems (NeurIPS)*, 2022.

Pan, L., Malkin, N., Zhang, D., and Bengio, Y. Better training of gflownets with local credit and incomplete trajectories. *International Conference on Machine Learning (ICML)*, 2023, to appear.

Sener, O. and Savarese, S. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*, 2018.

Settles, B. Active learning literature survey. 2009.

Srinivas, N., Krause, A., Kakade, S., and Seeger, M. Gaussian process optimization in the bandit setting: no regret and experimental design. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pp. 1015–1022, 2010.

Wang, C., Sun, S., and Grosse, R. Beyond marginal uncertainty: How accurately can bayesian regression models estimate posterior predictive correlations? In *International Conference on Artificial Intelligence and Statistics*, pp. 2476–2484. PMLR, 2021.

Zhang, D. W., Rainone, C., Peschl, M., and Bondesan, R. Robust scheduling with gflownets. *arXiv preprint arXiv:2302.05446*, 2023.

A. Further Experimental Details

A.1. Data

Toy Regression Task The toy regression task used for experiments in Section 4 is shown in Figure 6. The pool and test data was generated from the function $f(x) = (-0.6667 - 0.6012x - 1.0172x^2 - 0.7687x^3 + 1.4680x^5 - 0.1678x^6)\sin(\pi x)\exp(-0.5x^2) + N(0, 0.1)$, and $x \sim N(0, 1)$ where N is the normal distribution. The test set was fixed at 1000 points and generated from a different random seed. Seed datasets for active learning experiments were chosen randomly from the pool set.

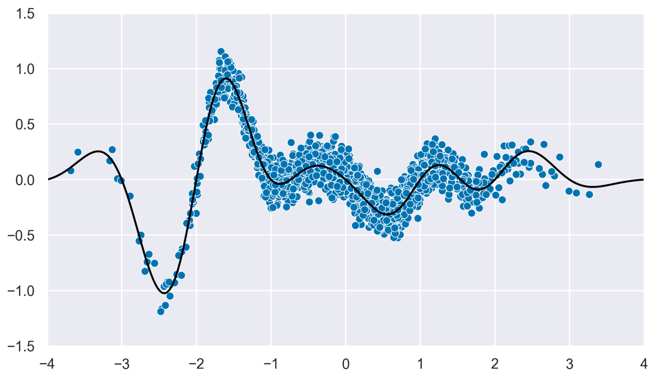

Figure 6. Toy example 1D regression data.

A.2. Models

The policy network architecture is shown in Figure 7. We use feed-forward neural network (FFN) encoders to encode each pool point and points in the current state. The embeddings of the points in the current state are summed (for set-invariance), and concatenated with each embedded pool point as a context vector. The combined embedding is fed through an additional FFN to output the probability of selecting each pool point. We share all parameters apart from output layers across P_F, P_B and the log-state-flow network (Madan et al., 2023, to appear.). For amortization experiments, we concatenate an additional context vector of summed embeddings of the training set points and their labels.

Hyperparameters for the policy network and the GP used for active learning are given in Table 1. We use the default GPyTorch (Gardner et al., 2018) settings for implementing the GPs[3].

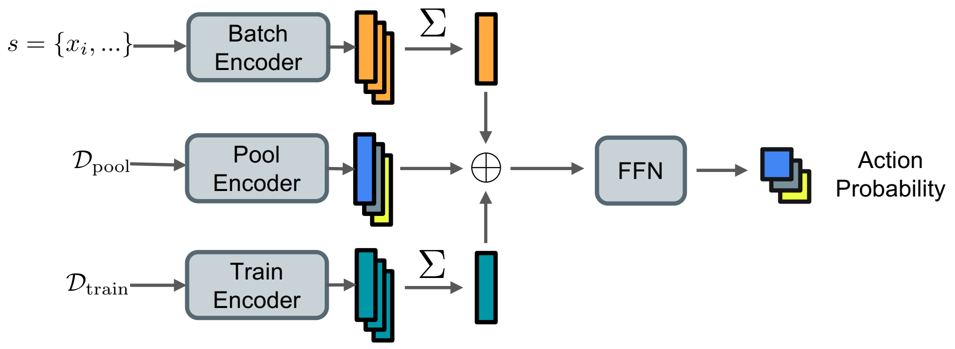

Figure 7. Policy network architecture. For non-transfer experiments we do not include the train encoder. The outputs of the policy network are the probabilities for adding each pool point to the current state, i.e. the query batch being built.

[3]Our code is available at `https://github.com/s-a-malik/batchgfn`. It relies on torchgfn (Lahlou et al., 2023) for implementing the GFlowNets.

Table 1. Hyperparameter configuration for active learning regression experiments.

Hyperparameter	**Value**
BatchGFN Optimization	
Loss	SubTB (forward-looking) (Madan et al., 2023, to appear.; Malkin et al., 2022)
SubTB λ	0.9
Training exploration ϵ	0.1
Reward Temperature T	0.1
Optimizer	Adam (Kingma & Ba, 2014)
Learning Rate	0.001
Batch Size	8
Training Iterations	5000
BatchGFN Architecture	
Hidden Layer Dimension	256
Number of Encoder Hidden Layers	2
Number of Batch Samples for Inference	20
Lookahead Experiments	
Lookahead Samples	10
Reward Temperature	0.1
Seed Size	17
Query Size	3
Pool Size	50
Pool sampling for lookahead	BatchGFN with training exploration
Active Learning Model	
Model	Exact Gaussian Process
Hyperparameter Training Epochs	1000
Optimizer	Adam (Kingma & Ba, 2014)
Learning Rate	0.1
Kernel	Matérn

A.3. Active Learning with BatchGFN

Algorithm 1 shows pseudocode for active learning with BatchGFN, including optional lookahead training where we hallucinate labels and train on possible future reward distributions. We are able to transfer to the new conditional reward distribution with fewer BatchGFN training iterations by using lookahead training (Section 4.4).

B. Further Results

B.1. Acquisition Plots

Figure 8 shows example acquisition plots comparing BatchGFN (gfn) acquisition to other baselines. BatchGFN acquires points that are diverse and uncertain like BatchBALD, whereas BALD acquires similar points.

B.2. Further Amortization Plots

We provide an additional example of transfer between AL steps on a smaller pool set of size 20 in Figure 9. Lookahead training again allows for faster convergence to the true distribution.

Algorithm 1 BatchGFN active learning with optional lookahead training.

```
Require: Query batch size: $B$, Seed set size: $B_0$, Pool dataset: $\mathcal{D}_{\text{pool}}$: $\{x_0, \dots, x_N\}$
Require: Active learning model: $f$, BatchGFN: $g$, Reward Function: $R$, Lookahead samples: $L$
  Randomly sample $\{x_0^*, \dots, x_{B_0}^*\}$ from $\mathcal{D}_{pool}$
  Label seed batch, $y_x^* \leftarrow \text{Oracle}(x)\ \forall x \in \{x_0^*, \dots, x_{B_0}^*\}$
  $\mathcal{D}_{\text{train}} \leftarrow \{(x_0^*, y_0^*), \dots, (x_{B_0}^*, y_{B_0}^*)\}$
  $\mathcal{D}_{\text{pool}} \leftarrow \mathcal{D}_{\text{pool}} \setminus \{x_0^*, \dots, x_{B_0}^*\}$
  while labelling budget not exhausted do
    Train $f$ on $\mathcal{D}_{\text{train}}$
    Train $g$ using $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{pool}}$, $f$, and $R$
    Sample batches from $g$
    batch $\leftarrow \max_R(\text{batches})$
    if lookahead then
      for $i = 1$ to $L$ do
        Sample lookahead_batch from $g$
        Hallucinate labels $y'_x \leftarrow f.\text{sample}(x')\forall x' \in$ lookahead_batch
        $\mathcal{D}'_{\text{train}} \leftarrow \mathcal{D}_{\text{train}} \cup \{(x'_0, y'_0), \dots, (x'_B, y'_B)\}$
        $\mathcal{D}'_{\text{pool}} \leftarrow \mathcal{D}_{\text{pool}} \setminus \{x'_0, \dots, x'_B\}$
        Train $f$ on $\mathcal{D}'_{\text{train}}$
        Train $g$ using $\mathcal{D}'_{\text{train}}$, $\mathcal{D}'_{\text{pool}}$, $f$, and $R$
      end for
    end if
    Get true labels $y_x^* \leftarrow \text{Oracle}(x)\ \forall x \in$ batch
    $\mathcal{D}_{\text{train}} \leftarrow \mathcal{D}_{\text{train}} \cup \{(x_0^*, y_0^*), \dots, (x_B^*, y_B^*)\}$
    $\mathcal{D}_{\text{pool}} \leftarrow \mathcal{D}_{\text{pool}} \setminus \{x_0^*, \dots, x_B^*\}$
  end while
```

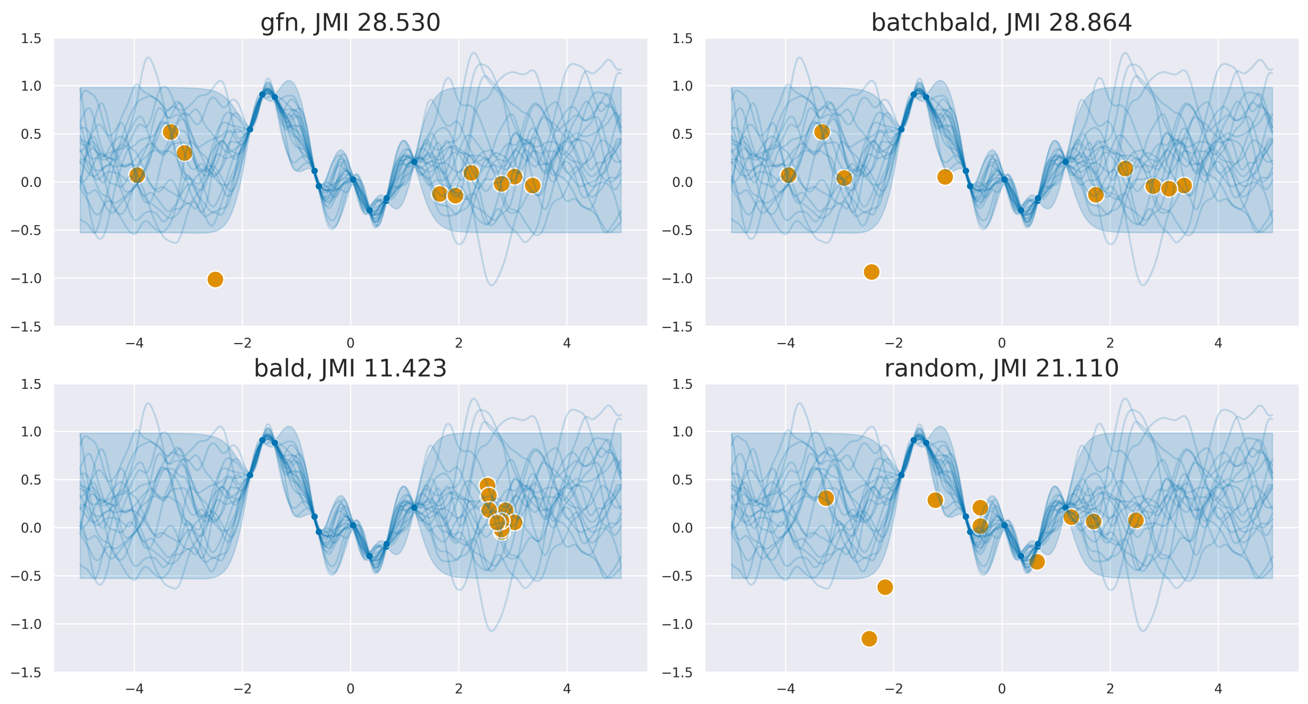

Figure 8. Acquisition plots for different active learning strategies. Pool size 2000, query size 10, training set size 10. Labelled/queried data points are shown in blue/orange respectively.

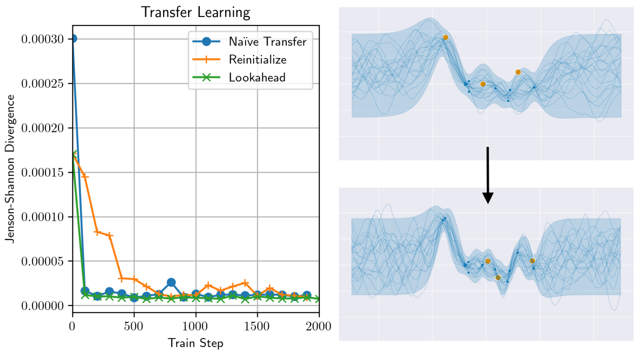

Figure 9. Training curve plot showing the Jenson-Shannon divergence between the true reward distribution and the empirical BatchGFN distribution when transferring from one acquisition step to the next (shown on the right). We use 10 lookahead samples.