Generative Flow Networks for Neural Network Hyperparameter Tuning

Beck LaBash
Northeastern University
labash.b@northeastern.edu

Sebastian Sepulveda
Northeastern University
sepulveda.s@northeastern.edu

Abstract

Hyperparameter selection is a vital component to neural network training due to the sensitivity of model performance to different hyperparameter configurations. Algorithms which select hyperparameters automatically, referred to as hyperparameter tuning algorithms, often suffer from computational complexity or get stuck in local minima. We propose a general framework to apply Generative Flow Networks to hyperparameter selection. We show that our framework can sample diverse, performant batch size, learning rate, and epoch configurations for training a feed-forward neural network on the Fashion-MNIST dataset.

1 Introduction

1.1 Hyperparameter Tuning

Many machine learning algorithms are parametrized by a set of hyperparameters. Unlike trainable parameters learned automatically from a data distribution, hyperparameters dictate *how* models learn from data and are often selected manually by scientists. This is an inefficient, trial-and-error process guided mostly by intuition. Various algorithms have been devised to increase the efficiency of hyperparameter selection. These algorithms seek to solve the hyperparameter optimization problem which can be defined formally as follows:

Given an ML model $\mathcal{M}$, a set of potential hyperparameter configurations Θ, A performance metric $\mathcal{A}$, a training dataset $\mathcal{D}_{\text{train}}$, and a validation dataset $\mathcal{D}_{\text{valid}}$, we seek to find a hyperparameter configuration θ^* such that

$$\theta^* = \arg\max_{\theta \in \Theta} \mathcal{A}(\mathcal{M}(\theta, \mathcal{D}_{train}), \mathcal{D}_{valid}) \tag{1}$$

One of these algorithms is referred to as Grid Search. Grid Search discretizes the hyperparameter domain into a grid, tries each configuration in the grid individually, and selects the best configuration based on a validation metric. The computational cost of trying a single hyperparameter configuration (usually training a model) combined with the exponential growth of the search space with the number of hyperparameters makes this algorithm incredibly costly.

Other methods utilize first order optimization to try to optimize the hyperparameters directly using validation loss as signal Bakhteev and Strijov (2019). These methods often suffer from convergence on local minima rather than finding global optimum.

Neural network performance in particular has been shown to be highly sensitive to different hyperparameter configurations, making the careful choice of hyperparameters an important issue. Additionally, Liao et al. (2022) finds that high performing configurations similar in accuracy may differ significantly according to other metrics such as training or inference latency. Thus, it is necessary to be able to sample a diverse set of optimal configurations.

Preprint. Under review.

1.2 Generative Flow Networks

We propose to apply Generative Flow Networks (GFNs) remedy the issues with the described algorithms and address the problem-specific challenges of hyperparameter tuning. GFNs are stochastic policies that sample objects from an environment. Unlike traditional RL methods, which learn to maximize reward and are often greedy, GFNs sample a diversity of trajectories in proportion to their reward $R(x)$, making them suitable for environments in which multiple candidates are required.

GFNs construct a flow network of states as nodes and actions as edges. States represent intermediate representations of objects and actions represent adding to the object. A trajectory t through the network is represented as a flow from a source node (starting state) to a sink node (terminal state). The inflow into each sink node matches the reward of the corresponding terminal state, and terminal states are thus sampled with probability proportional to their reward. It is important to note that this flow network is implicit; not all states are enumerated. Instead, a flow function F is learned which estimates the flow between a state s and a successive state s'.

To learn F, A neural network is trained according to the TD-like flow-matching objective. That is, we seek to match the flow into a state s' from preceding states s with the flow out of s' into subsequent states s'':

$$\mathcal{L} = (\sum_{s} \hat{F}(s \to s') - R(s') + \sum_{s''} \hat{F}(s' \to s''))^2 \tag{2}$$

From F, a policy π is derived to sample trajectories:

$$\pi(a|s) = \frac{F(s \to s')}{\sum_{s''} F(s \to s'')} \tag{3}$$

2 Methods

We define a GFN with states as partially complete configurations of hyperparameters. Actions add a single hyperparameter and its value to the configuration. Terminal states represent complete hyperparameter configurations.

2.1 Energy Function

We define the reward of terminal states as the accuracy of a model trained with the hyperparameters of the terminal state. Specifically, we train a simple feed-forward network to classify the Fashion-MNIST dataset Xiao et al. (2017), and parameterize the network on learning rate, batch size, and number of epochs. As an optimization, we performed a hyperparameter sweep over these variables using Ray Tune Liaw et al. (2018) and trained a DNN to predict test accuracy on Fashion-MNIST from the three hyperparameters. We use this DNN as a surrogate reward network which approximates the ground truth, "oracle" reward, vastly speeding up the training of the GFN.

2.2 Environment

Training a flow network requires us to specify an environment for the network to learn. We specify a given state s as a concatenation of a list of hyperparameters and a boolean vector v corresponding to which hyperparameters for the network have been specified. Specifically, $v_i = 1$ where hyperparameter i has been configured, and 0 otherwise. Note this state representation assumes that hyperparameters are chosen in a fixed order. We specify an action a as a one-hot vector, where the index n of the vector which is activated corresponds to the discretized quantity of the hyperparameter chosen. We formulate our action space this way so that we can represent the continuous space of possible hyperparameters with a discrete GFN.

2.3 Forward Policy

We construct a mixture-of-experts model inspired by Mixtral Jiang et al. (2024) to train our forward policy. Different states in the flow network have different lengths: the root node has no set hyperpa-

rameters and thus no information to pass to the network, while sink nodes have three hyperparameters specified and have three pieces of information to pass to the network. Thus, we define four experts for our network, taking in 0-3 hyperparameters respectively. We scale all inputs between 0 and 1.

3 Experiments

To evaluate our method, we attempt to sample optimal hyperparameters (batch size, learning rate, epochs) for the training of a feed-forward neural network on the Fashion-MNIST dataset Xiao et al. (2017). Fashion-MNIST consists of grayscale images of clothing and their corresponding label. The objective of the neural network is to correctly select the label of the clothing based on the image.

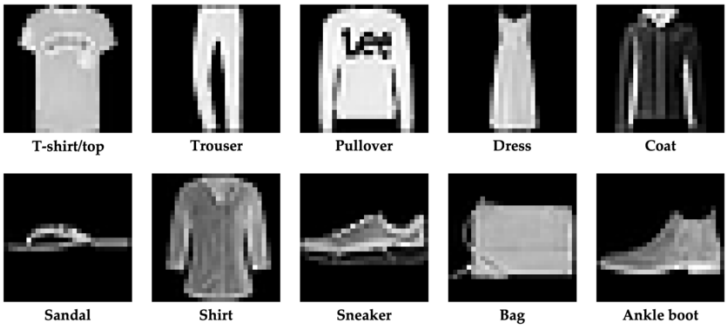

Figure 1: The Fashion-MNIST Dataset

We define the reward of a complete hyperparameter configuration as the accuracy of the trained model on the Fashion-MNIST test set. In order to speed up the training of the GFN, we train a DNN to emulate the ground truth reward function. We train our surrogate model on a dataset of 1400 of hyperparameter configurations, with accuracy scores on Fashion-MNIST as labels. The surrogate model achieves a loss of 0.0006 on its validation set.

We train our forward policy network on 10,000 sample trajectories, with a learning rate of $5 * 10^{-5}$ and a batch size of 32.

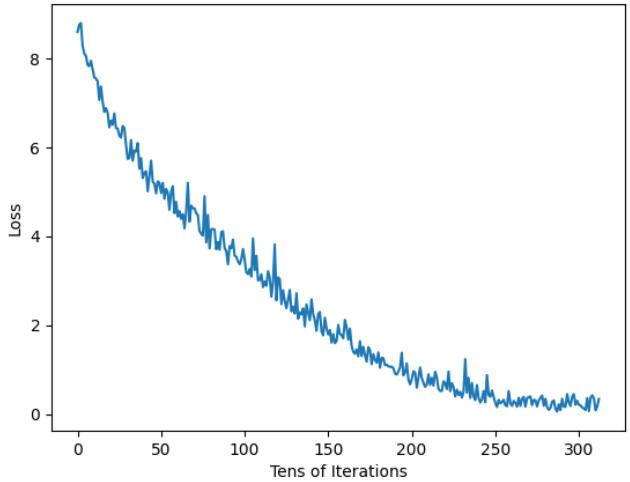

Figure 2: Training losses of forward policy

To evaluate the performance of our policy, we sample 1,000 trajectories from the policy and calculate the energy of each trajectory with our surrogate model. When sampling from a GFN, we can specify a temperature: a scalar to multiply the logits of the policy network by before performing a softmax

over them. We sample with a temperature of 0.1 to generate a greedy distribution of hyperparameter candidates. We sample from the forward policy with trained and untrained weights to show the effectiveness of training, and compare both of these with the distribution of accuracy from the grid search. Note that we compare the energy function computed from the surrogate network for samples to the real test accuracy from grid search data.

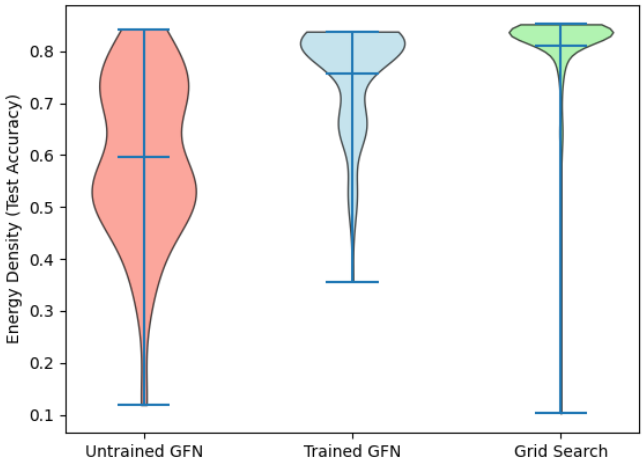

Figure 3: Performance of hyperparameter search strategies

4 Discussion

The violin plot compares the distributions of test accuracy from 10,000 sampled trajectories from a trained and untrained GFN, as well as the distribution of test accuracy from the grid search that was conducted to train the surrogate reward model. We can see that the trained GFN samples higher-reward hyperparameter sets more frequently than the untrained GFN, showing that the GFN does learn a distribution matching the flow from the energy function.

Note that both GFN distributions have energy distributions with lower modes than that of the grid search. Given our use of a surrogate model, this makes sense: If our surrogate was able to perfectly match the reward distribution of the data, we would expect an optimal GFN to sample with the same proportionality. But given the lossy nature of the surrogate and policy networks, we expect some deviation from the original distribution. We also expect that passing the dataset through the surrogate network will smooth out the distribution of rewards from the discrete data points of the grid search, producing some energies of a lower value.

5 Code Release

We publicly release the code for training and sampling here.

References

Bakhteev, O. Y. and Strijov, V. V. (2019). Comprehensive analysis of gradient-based hyperparameter optimization algorithms. *Annals of Operations Research*, 289:51 – 65.

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., de las Casas, D., Hanna, E. B., Bressand, F., Lengyel, G., Bour, G., Lample, G., Lavaud, L. R., Saulnier, L., Lachaux, M.-A., Stock, P., Subramanian, S., Yang, S., Antoniak, S., Scao, T. L., Gervet, T., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. (2024). Mixtral of experts.

Liao, L., Li, H., Shang, W., and Ma, L. (2022). An empirical study of the impact of hyperparameter tuning and model optimization on the performance properties of deep neural networks. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 31:1 – 40.

Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E., and Stoica, I. (2018). Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.