AUTONF: AUTOMATED ARCHITECTURE OPTIMIZATION OF NORMALIZING FLOWS USING A MIXTURE DISTRIBUTION FORMULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Although various flow models based on different transformations have been proposed, there still lacks a quantitative analysis of performance-cost trade-offs between different flows as well as a systematic way of constructing the best flow architecture. To tackle this challenge, we present an automated normalizing flow (NF) architecture search method. Our method aims to find the optimal sequence of transformation layers from a given set of unique transformations with three folds. First, a mixed distribution is formulated to enable efficient architecture optimization originally on the discrete space without violating the invertibility of the resulting NF architecture. Second, the mixture NF is optimized with an approximate upper bound which has a more preferable global minimum. Third, a block-wise alternating optimization algorithm is proposed to ensure efficient architecture optimization of deep flow models.

1 INTRODUCTION

Normalizing flow (NF) is a probabilistic modeling tool that has been widely used in density estimation, generative models, and random sampling. Various flow models have been proposed in recent years to improve their expressive power. Discrete flow models are either built based on elemental-wise monotonical functions, named autoregressive flow or coupling layers (Papamakarios et al., 2017), or built with transformations where the determinant of the flow can be easily calculated with matrix determinant lemma (Rezende & Mohamed, 2015). In the continuous flow family, the models are constructed by neural ODE (Grathwohl et al., 2019).

Despite the variety of flow models, there's yet no perfect flow concerning the expressive power and the computation cost. The flow models with higher expressive power usually have higher computational costs in either forward and inverse pass. In contrast, flows which are fast to compute are not able to model rich distributions and are limited to simple applications. For instance, autoregressive flows (Papamakarios et al., 2017) are universal probability approximators but are D times slower to invert than forward calculation, where D is the dimension of the modeled random variable $\boldsymbol{x}$ (Papamakarios et al., 2021). Flows based on coupling layers (Dinh et al., 2015; 2017; Kingma & Dhariwal, 2018) have an analytic one-pass inverse but are less expressive than their autoregressive counterparts. Other highly expressive NF models (Rezende & Mohamed, 2015; Behrmann et al., 2019) cannot provide an analytic inverses and relies on numerical optimizations.

For different applications, the optimal flow model can be drastically different, especially if the computation cost is taken into consideration. For generative models (Dinh et al., 2015; Kingma & Dhariwal, 2018), flows with the fast forward pass are preferable since the forward transformations need to be applied to every sample from the base distribution. For density estimation (Papamakarios et al., 2017; Rippel & Adams, 2013), flows with cheap inverse will prevail. For applications where flow is utilized as a co-trained kernel (Mazoure et al., 2020), the computation cost and performance trade-off are more important, i.e., having a fast model with relatively good performance. However, in the current body of work, the architecture designs of the flow models are all based on manual configuration and tuning. To this date, there is a lack of a systematic way that could automatically construct an optimal flow architecture with a preferred cost.

In this paper, we propose AutoNF, an automated method for normalizing flow architecture optimization. AutoNF has a better performance-cost trade-off than hand-tuned SOTA flow models based on a given set of transformations. Our approach employs a mixture distribution formulation that can search a large design space of different transformations while still satisfying the invertibility requirement of normalizing flow. The proposed mixture NF is optimized via approximate upper bound which provides a better optimization landscape for finding the desired flow architecture. Besides, to deal with exponentially growing optimization complexity, we introduce a block-wise optimization method to enable efficient optimization of deep flow models.

2 Related Work

Normalizing Flows: Various normalizing flow models have been proposed since the first concept in (Tabak & Turner, 2013). Current flow models can be classified into two categories: finite flows based on layer structure, and continuous flow based on neural ODE (Grathwohl et al., 2019). The finite flow family includes flows based on elemental-wise transformation (Papamakarios et al., 2017; Kingma & Dhariwal, 2018) and flows whose transformations are restricted to be contractive (Behrmann et al., 2019). In elemental-wise transformation flows, autoregressive flow and coupling layers are two major flavors and extensive work has been proposed to improve the expressive power of both flow models. In Huang et al. (2018), the dimension-wise scalar transformation is implemented by a sigmoid neural network, which increases the expressive power at the cost of being not analytically invertible. In Durkan et al. (2019), piecewise splines are used as drop-in replacement of affine or additive transformations (Dinh et al., 2015; 2017) and is the current SOTA flow model. Consequently many recent research efforts have been devoted to closing the gap of expressive power, albeit at the cost of more complex and expensive transformations. Moreover, there has been no quantitative trade-off analysis between the performance and cost among different flows.

Neural Architecture Search: Many algorithms have been proposed or applied for neural architecture search. For instance, reinforcement learning (Zoph & Le, 2017), genetic algorithm (Real et al., 2017; Suganuma et al., 2018; Liu et al., 2018), Monte Carlo tree search (Negrinho & Gordon, 2017) or Bayesian optimization (Kandasamy et al., 2018). However, these methods all face the challenge of optimizing on a large discrete space and can take thousand of GPU days to find a good architecture. To address this issue, DARTS (Liu et al., 2019) proposes to relax the search space from discrete to continuous and allows efficient differentiable architecture search with gradient method which could reduce the search time to a single GPU day while still producing the SOTA architecture. However, all current NAS methods focus on optimizing traditional neural network structures (CNN, RNN) and there has yet been any implementation on normalizing flow.

Necessity for the Trade-off Between Performance and Cost: Despite various transformations proposed in the literature, there is no perfect transformation with strong expressive power and low computational cost. Autoregressive flows have better expressive power, but the inverse computation cost grows linearly with data dimension. Coupling layers' inverse calculation is as fast as the forward pass, but their expressive power is generally worse than autoregressive flow with the same element-wise transformation. Even in the same autoregressive flow or coupling layer family, flows with different element-wise transformations have different performance and computation costs. For instance, additive or affine coupling layers (Dinh et al., 2017; 2015) have very fast forward and inverse calculation with limited expressive power while the flow in (Durkan et al., 2019) are highly expressive but are more demanding on computation. In most applications, it is necessary to find the best performance while minimizing at least one specific component of the cost. Unfortunately, the current design of flow models is empirical and therefore cannot ensure the optimal trade-offs.

3 Method

In this work, we aim to tackle the challenge of finding an optimal flow model for a given task via an automated architecture search algorithm.

Assumptions: In the remaining part of this paper, without losing generality, we assume that the transformation is properly modeled such that during the training process, only forward computation is needed. Under this assumption, when the flow model is used for density modeling (Durkan

et al., 2019), the forward calculation is the dominant computation. When the flow model is used for random sampling (Kingma & Dhariwal, 2018), the inverse calculation is computationally intensive. When the flow model is utilized as a module and trained together with other components, e.g., policy network in maximum entropy learning (Mazoure et al., 2020), the training cost of the flow model is an important consideration.

Problem Definition: Given a transformation set with m options $\{T^1, T^2, ...T^m\}$, the goal is to construct an optimal flow model with n layers of transformations from the set. The flow model $p_{NF}(\boldsymbol{x};\boldsymbol{\theta}) = p_{T_1 T_2 ... T_n}(\boldsymbol{x};\boldsymbol{\theta})$ should minimize the KL divergence between the target distribution $p^*(x)$ and itself while minimizing its computational cost C_{NF}. Here, $\boldsymbol{\theta}$ are the parameters of the transformation in the flow model. In this paper, we use the forward KL divergence as our target loss function (Papamakarios et al., 2021):

$$\begin{aligned} \boldsymbol{\theta}^* = &\arg\min_{\boldsymbol{\theta}} \{D_{KL}[p^*(\boldsymbol{x}) \,||\, p_{T_1 T_2 ... T_n}(\boldsymbol{x};\boldsymbol{\theta})] + \lambda \cdot C_{NF}\} \\ &\text{s.t.} \quad T_i \in \{T^1, T^2, ...T^m\} \end{aligned} \tag{1}$$

While λ is a tuning factor capturing the relative importance of the performance-cost trade-off. Finding this optimal flow model is a discrete optimization problem with exponential complexity. To enable efficient architecture optimization, we use proposed method of relaxing the discrete search space to continuous space as suggested in Liu et al. (2019).

3.1 Mixed Flow Ensemble

For the i_{th} transformation layer with m options, we introduce a corresponding weight w_i^j for each option T^j which reflects how likely the transformation will be selected. The weight is parameterized by a vector $\boldsymbol{\alpha}$ and made continuous via softmax:

$$w_i^j = \frac{\exp(\alpha_i^j)}{\sum_{j=1}^m \exp(\alpha_i^j)} \tag{2}$$

By applying this parameterization for each transformation layer, we can construct a ***mixed flow ensemble*** $p_{Mix}(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\alpha})$, where each layer in this mixed model reflects a weighted combination of the effect of all possible transformations. In this case, the architecture optimization problem is reduced to learning the weight vector for each layer and, at the end of the optimization process, weights will be binarized and the transformation with the highest weight in one layer will be selected as the final transformation. The mixed flow ensemble thus degrades to a normal flow model. The whole procedure is illustrated in Fig. 1 (left).

As adopted in (Liu et al., 2019), training of the flow ensemble becomes joint optimization of the architecture parameter $\boldsymbol{\alpha}$ and the model parameter $\boldsymbol{\theta}$ over the training and validation datasets, which could be written as the following bi-level optimization problem:

$$\begin{aligned} \boldsymbol{\alpha}^* = &\arg\min_{\boldsymbol{\alpha}} \; D_{KL}^{val}[p^*(\boldsymbol{x}) \,||\, p_{Mix}(\boldsymbol{x};\boldsymbol{\theta}^*,\boldsymbol{\alpha})] + \lambda \cdot C_{Mix}(\boldsymbol{\alpha}) \\ &\text{s.t.} \quad \boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \; D_{KL}^{train}[p^*(\boldsymbol{x}) \,||\, p_{Mix}(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\alpha})], \\ &\forall\, T \in p_{Mix},\; T \in \{T^1, T^2, ...T^m\}, \end{aligned} \tag{3}$$

While the optimization problem is well defined, the key challenge is to construct the flow ensemble within the normalizing flow framework. This is different from traditional neural architecture search, which can mix various operations with no additional issue. Normalizing flow has its unique requirement for the invertibility of transformations and a preferred simple Jacobian calculation, which requires careful handling.

The mixed flow ensemble $p_{Mix}(\boldsymbol{x};\boldsymbol{\theta}^*,\boldsymbol{\alpha})$ must satisfy two requirements. First, it must be a legal density function such that it can be optimized by the KL divergence formulation. Second, each transformation layer in $p_{Mix}(\boldsymbol{x};\boldsymbol{\theta}^*,\boldsymbol{\alpha})$ should represent a weighted combination of all possible transformations. Consider the i_{th} layer in the mixed flow ensemble with input random variable $\boldsymbol{x}_{in}$ and output random variable $\boldsymbol{x}_{out}$, and $p_{\boldsymbol{x}_{in}}(\boldsymbol{x}_{in})$ and $p_{\boldsymbol{x}_{out}}(\boldsymbol{x}_{out})$ are their corresponding density functions. This layer has m transformation options in $\{T_i^1, T_i^2, ...T_i^m\}$ and w_i^j is the corresponding

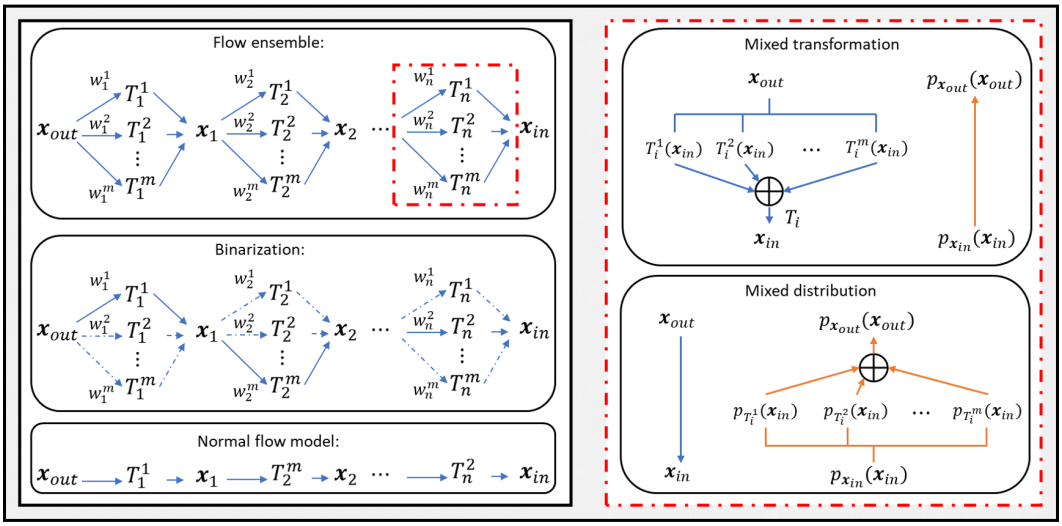


Figure 1: Left-top: the relaxation of search space and the flow ensemble is shown in Fig. 1. Left-middle: binarization of weights. Left-bottom: degradation to normal flow architecture. Right-top: construction flow ensemble by mixed transformations. Right-bottom: construction of flow ensemble by mixing distributions. The blue line in right indicates transformation on random variables and the orange line reflects change in distributions.

weight for each transformation. As discussed in Assumption, we assume all transformations directly model the inverse transformation, i.e. $\boldsymbol{x}_{in} = T_i^j(\boldsymbol{x}_{out})$. Two approaches can be used to construct the mixed flow ensemble.

Construction by Mixed Transformations: The straight forward way of building the i_{th} mix flow ensemble layer is to mix all transformations by weighted summation, as shown in Fig. 1 (right-top). The final weighted transformation for this layer can be thus represented as:

$$T_i(\boldsymbol{x}_{in}) = \sum_{j=1}^{m} w_i^j \cdot T_i(\boldsymbol{x}_{out}) \tag{4}$$

There are two drawbacks of this formulation despite its simplicity. First, definition of normalizing flow requires the mixed transformation T_i be invertible and differentiable in order to ensure $p_{\boldsymbol{x}_{out}}(\boldsymbol{x}_{out})$ legal density function. However, this invertibility is not guaranteed even if all candidate transformations are invertible. Second, even if the mixed transformation is invertible, there is no easy way to calculate the Jacobian determinant of this weighted summation of transformations. Meeting the requirement of invertibility and ease of calculating Jacobian determinant brings strict restrictions on the candidate transformations and prevents the optimization of flow architectures on a wider search space. As a result, the construction of the mixed flow ensemble by weighted summation of transformations is not adopted in this paper.

Construction by Mixed Distributions: An alternating way is to build the mixed flow ensemble by mixing distributions. For a given transformation T_i^j in this i_{th} layer, applying the transformation to the input random variable will result in a new distribution:

$$p_{T_i^j}(\boldsymbol{x}_{out}) = p_{\boldsymbol{x}_{in}}(T_i^j(\boldsymbol{x}_{out})) \cdot |\det \boldsymbol{J}_{T_i^j}(\boldsymbol{x}_{out})| \tag{5}$$

By applying this to every transformation option in $\{T_i^1, T_i^2, ...T_i^k\}$, we can obtain k different distributions, and it is possible to mix all the density functions together by their weighted summation, to get a mixture model as shown in eq.(6).

$$p_{T_i(\boldsymbol{x}_{out})} = \sum_{j=1}^{m} w_i^j \cdot p_{T_i^j}(\boldsymbol{x}_{out}) \tag{6}$$

An illustration of this process is shown in Fig. 1 (right-bottom). Different from the previous approach, the mixture model has a legal density function as: $p_{T_i}(\boldsymbol{x}_{out})$. By the definition of normalizing flow, we can assume that there exists an invertible and differentiable transformation T_i, which transforms $\boldsymbol{x}_{in}$ to $\boldsymbol{x}_{out}$, although the transformation itself can not be explicitly written out.

For the next $(i+1)_{th}$ layer, the density of the mixture model will be used as the input density function $p_{\boldsymbol{x}_{in}}(\boldsymbol{x}_{in})$ as in the previous layer. By applying this formulation for n layers, the final mixed flow ensemble can be written as:

$$p_{Mix}(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{a}) = \sum_{k=1}^{m^n} W_k \cdot p_{T_1 T_2 \ldots T_n}(\boldsymbol{x},\boldsymbol{\theta}) = \sum_{k=1}^{m^n} W_k \cdot p_i(\boldsymbol{x};\boldsymbol{\theta}_i)$$
$$\text{where each} \quad W_k = \prod_{i=1}^{n} w_i \quad \text{and} \quad \sum_{k}^{m^n} W_k = 1 \tag{7}$$

Each w_i is defined in eq.(2) and we use $p_k(\boldsymbol{x};\boldsymbol{\theta}_k)$ to represent a "normal flow architecture" with n transformation layers. Clearly, the final mixed flow ensemble is a legal density function which is in fact, a weighted summation of all possible flow models built with n layers of transformations.

3.2 Optimization With Approximated Upper Bound

Optimizing the forward KL divergence between the target distribution and the mixed flow ensemble can be written as:

$$\begin{aligned}\mathcal{L}^{O}_{p_{Mix}} &= D_{KL}\left[p^*(\boldsymbol{x}) \,||\, p_{Mix}(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\alpha})\right] \\ &= -E_{p^*(\boldsymbol{x})}[\log(\sum_{k=1}^{m^n} W_k \cdot p_k(\boldsymbol{x};\boldsymbol{\theta}_k))]\end{aligned} \tag{8}$$

We will demonstrate that direct optimization of this original loss is not always desirable. In the whole search space of the flow ensemble, we are interested only in "normal flow architectures" points, i.e. the points where the weight of one architecture is 1 and others are all 0. However, it can be easily proven that the global minimum of $\mathcal{L}^{O}_{p_{Mix}}$ may not be the desired normal flow architecture (the red points in Fig. 2). Instead, optimization is very likely to end up in a mixture model that is globally optimal with similar weight for each possible flow architecture (the green point in Fig. 2). In this case, we will encounter difficulty when extracting a normal flow architecture with the search result. A heuristic way in (Liu et al., 2019) is binarizing the weights and select corresponding transformations. However, there is no guarantee that the binarized architecture will have a lower loss than other possible normal flow architectures. As a result, optimization with the original loss function is not suitable, and could be risky.

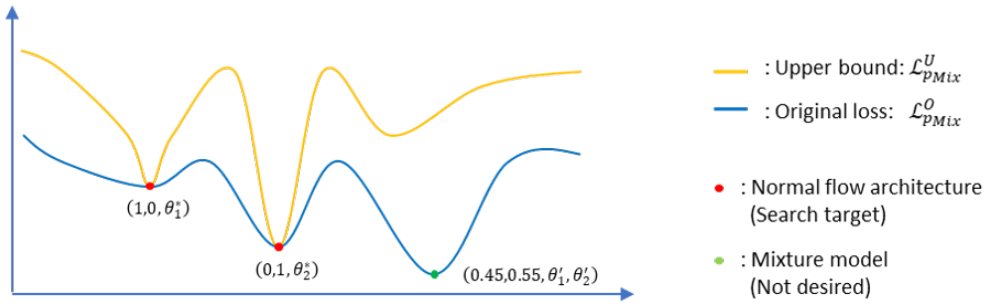


Figure 2: An illustrative example of the original loss and upper bound for a flow ensemble with 2 possible architectures. The red points indicate desired normal flow architectures and the green point indicates the global minimum of $\mathcal{L}^{O}_{p_{Mix}}$, which is a mixture model. The parameters $(a, b, \theta_1, \theta_2)$ refer to the weight of architecture 1, architecture 2 and their corresponding parameters.

In this paper, we propose to optimize an upper bound of the original loss function to provide a better landscape for the search of best normal flow architectures. Our method utilizes Jensen's inequality

$\log(\sum W \cdot x) \geq \sum W \cdot \log(x)$ as follows, since we have $\sum W = 1$ and the log function is concave, we can obtain an upper bound of the KL divergence given as:

$$\mathcal{L}^{O}_{p_{Mix}} = -E_{p^*(\boldsymbol{x})}[\log(\sum_{k}^{m^n} W_k \cdot p_k(\boldsymbol{x};\boldsymbol{\theta}_k)] \leq \mathcal{L}^{U}_{p_{Mix}} = -E_{p^*(\boldsymbol{x})}[\sum_{k}^{m^n} W_k \cdot \log(p_k(\boldsymbol{x};\boldsymbol{\theta}_k))] \quad (9)$$

The benefit of optimizing the upper bound can be summarized as follows:

Proposition 1: *The global minimum point of $\mathcal{L}^{U}_{p_{Mix}}$ is defined by a normal flow architecture.*

Proof: Suppose each flow model $p_k(\boldsymbol{x};\boldsymbol{\theta}_k)$ has an optimal parameter $\boldsymbol{\theta}^*_k$ that minimizes the KL divergence between $p^*(x)$ and it:

$$-E_{p^*(\boldsymbol{x})}[\log(p_k(\boldsymbol{x};\boldsymbol{\theta}^*_k)] \leq -E_{p^*(\boldsymbol{x})}[\log(p_k(\boldsymbol{x};\boldsymbol{\theta}_k)] \quad (10)$$

There also exists a flow architecture ($p_z(\boldsymbol{x};\boldsymbol{\theta}^*_z)$) that has the minimal KL divergence:

$$-E_{p^*(\boldsymbol{x})}[\log(p_z(\boldsymbol{x};\boldsymbol{\theta}^*_z)] \leq -E_{p^*(\boldsymbol{x})}[\log(p_k(\boldsymbol{x};\boldsymbol{\theta}_k)],\ \forall k \in m^n \quad (11)$$

We can then prove the proposition by showing that:

$$\begin{aligned} \mathcal{L}^{U}_{p_{Mix}} &= -E_{p^*(\boldsymbol{x})}[\sum_{k}^{m^n} W_k \cdot \log(p_k(\boldsymbol{x};\boldsymbol{\theta}_k))] \geq -E_{p^*(\boldsymbol{x})}[\sum_{k}^{m^n} W_k \cdot \log(p_k(\boldsymbol{x};\boldsymbol{\theta}^*_k))] \\ &\geq -E_{p^*(\boldsymbol{x})}[\sum_{k}^{m^n} W_k \cdot \log(p_z(\boldsymbol{x};\boldsymbol{\theta}^*_z))] = -E_{p^*(\boldsymbol{x})}[\log(p_z(\boldsymbol{x};\boldsymbol{\theta}^*_z)] \end{aligned} \quad (12)$$

Proposition 2: *At normal architecture points ($W_k = 1, \boldsymbol{W}_{-k} = \boldsymbol{0}$), $\mathcal{L}^{U}_{p_{Mix}} = \mathcal{L}^{O}_{p_{Mix}}$.*

The proof of proposition 2 is apparent and with the above propositions, we can show that the solution set, i.e. all possible normal flow architectures are the same in both $\mathcal{L}^{O}_{p_{Mix}}$ and $\mathcal{L}^{U}_{p_{Mix}}$, and we can do optimization with proposed upper bound without violating the original definition. Furthermore, since the global optimum of the upper bound will always lead to a normal flow architecture, we will not end up in finding a mixture model with the need to do heuristic and risky binarization of weights W.

3.3 Efficient Architecture Optimization for Deep Flow Models

While the flow ensemble by mixed density formulation could reflect the weighted effect of all possible transformation combinations, the architecture optimization complexity grows exponentially with respect to the number of considered transformation types and the number of transformation layers. In this scenario, efficient optimization of the whole flow architecture will not be possible. It is natural to decompose the original problem into sequential optimization of few different blocks, where each block could be optimized in one time with a limited number of layers. We propose two methods to decompose the problem.

Grow Method: The first approach is a straightforward greedy method which we call "Grow". Each time, a block is optimized until convergence, and the weights of the transformation layer are binarized. The searched transformations in this block will be directly added to the searched layer in the previous block. The architecture optimization of later blocks will be based on the existing layers and, the growth of layers stops when reaching the total number of layers constraint. Despite its simplicity, the downside of the "Grow" method is that the optimization is short-sighted. The block being optimized has no information about the architectures which could be added later, and the whole architecture is more likely to be trapped in local minimum.

Block Method: To avoid the issue of getting stuck in a local minimum, we propose another method named "Block" optimization. Blocks $\boldsymbol{B}$ in this approach are optimized alternatively to allow each block to adjust their architectures with respect to other blocks. In fact, the first "Grow" approach is a specific case of the "Block" method, where all the blocks are initialized as identity transformations and optimized only once.

Algorithm 1 Algorithm flow for AutoNF

Require: Transformations: $\{T^1, T^2, ...T^m\}$, Blocks: $\boldsymbol{B} = \{B_1, B_2, ...B_l\}$, Cost: C_{Mix}
Ensure: n-layer flow model:
while not converged **do**
  for each $B_i \in \boldsymbol{B}$ **do**
    while not convergence **do**
      $\boldsymbol{\alpha}_{B_i} = \arg\min_{\boldsymbol{\alpha}_{B_i}} D_{KL}^{val}[p^*(\boldsymbol{x}) \,||\, p_{Mix}(\boldsymbol{x}; \boldsymbol{\theta}_{\boldsymbol{B}}^*, \boldsymbol{\alpha}_{B_i})] + \lambda \cdot C_{Mix}(\boldsymbol{\alpha}_{B_i})$
      $\boldsymbol{\theta}_{\boldsymbol{B}} = \arg\min_{\boldsymbol{\theta}_{\boldsymbol{B}}} D_{KL}^{train}[p^*(\boldsymbol{x}) \,||\, p_{Mix}(\boldsymbol{x}; \boldsymbol{\theta}_{\boldsymbol{B}}, \boldsymbol{\alpha}_{B_i})]$
    end while
    Fix architecture for B_i
  end for
end while

3.4 Cost Model and Algorithm Flow

As discussed in section II, we are interested in modeling the training cost (forward calculation cost) and the inverse calculation cost, since each of them plays a different role based on desired applications. We use an independent experiment to model the cost of different types of flows and summarized in a table which are included in Appendix B. With the cost model, the total cost of the mixed flow ensemble could be extracted based on emphasize on different costs, e.g. if training cost is the major concern, only training cost of different flows will be calculated. This total cost C_{Mix} is then added as an regularization term into the training loss function.

In our paper, gradient based method is used for optimization which is efficient in this very high dimensional search space. The architecture parameter $\boldsymbol{\alpha}$ and the flow model parameter $\boldsymbol{\theta}$ are optimized alternatively with first order approximation in (Liu et al., 2019). The final algorithm flow of our proposed AutoNF method can be summarized in Algorithm 1.

4 Experiments

4.1 Evaluation of Proposed Upper Bound

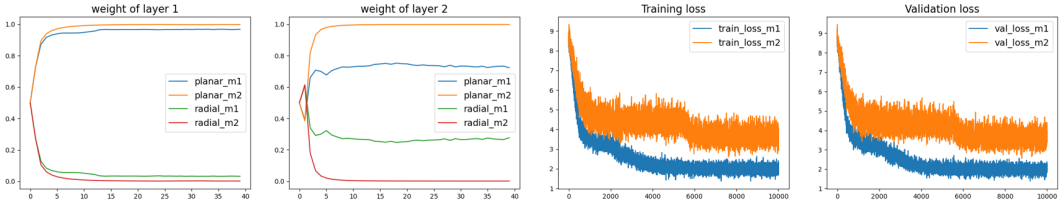
Setup: We use a simple example to demonstrate the necessity of doing optimization with our proposed upper bound. We use AutoNF to build a 4 layer flow model with 2 transformation options including planar flow and radial flow from (Rezende & Mohamed, 2015). We use the POWER dataset as the target and optimize with original loss (name M1) and our proposed upper bound (named M2). We use Adam optimizer for both architecture parameter and model parameter with a learning rate of 0.002. The batch size is 512 and the training iteration is 10000.

The results are shown in Fig.3. For both M1 and M2, we present the weight for planar and radial flow for each layer as well as the training and validation loss during the search process. The final weight for each layer, searched architectures after binarization and the test score are shown in the right-bottom table.

Analysis: Optimization with our proposed upper bound (M2) shows a concrete convergence of weight to 0 or 1 for each layer, which leads to a desired normal flow architecture, while the optimization with the original loss function (M1) ends up in a mixture model instead of a normal flow architecture, as shown in Fig.3(left). This is within in our expectation as shown in Fig.2. Moreover, although the mixture model is mostly likely to be the optimal in the original loss, the normal flow architecture after binarization however, is not an optimal model. As shown in the right-bottom table, the architecture found by M2 has a significantly better test score than M1, and this clearly supports our statement of doing optimization with our proposed upper bound.

4.2 Search for flow Models with Best performance Cost Trade-off

Transformation Options: To evaluate our AutoNF framework, we setup our experiments with four types of non-linear flows and one linear flow. In autoregressive family, we choose affine autore-



	M1		M2	
	planar	radial	planar	radial
Final W (Layer 1)	0.968	0.032	0.998	0.002
Final W (Layer 1)	0.723	0.277	0.999	0.001
Final W (Layer 1)	0.034	0.966	0.998	0.002
Final W (Layer 1)	0.939	0.061	0.999	0.001
Searched Architecture	[planar, planar, radial, planar]		[planar, planar, planar, planar]	
Test Score	4.47±0.034		3.74±0.027	

Figure 3: The result of optimization of a 4-layer flow ensemble with transformation options between planar flow and radial flow with original loss and proposed upper bound. The left four figures are the weight for each layer during the search process. The right-top figures are the training and validation loss during training. The right-bottom table collects final weight for each layer, the searched architecture, and their test score (lower the better).

gressive flow (Papamakarios et al., 2017) and rational quadratic autoregressive flow (Durkan et al., 2019). Affine autoregressive flow has limited expressive power but the computation cost is lower, while the later has the state of art performance in autoregressive family with higher cost. Affine coupling layer (Dinh et al., 2015) and rational quadratic coupling layer (Durkan et al., 2019) are selected from coupling layer family. For linear transformation, we combine a reverse permutation and an LU linear layer together as a single layer. Random permutation (Durkan et al., 2019; Oliva et al., 2018) is not used since it is difficult to reproduce in architecture optimization. Every non-linear transformation layer is paired with a linear transformation layer suggested by Durkan et al. (2019) as a final transformation option, i.e., a layer in our experiment contains a reverse permutation, an LU-linear layer and one of the non-linear transformation layer listed above.

Datasets and Model Configuration: The performance of the flow models are evaluated with density estimation for UCI (Dua & Graff, 2017) and BSDS300 (Martin et al., 2001) datasets. The optimization goal is to search for an eight-layer flow model which minimize both negative log likelihood and the total cost on different datasets with an emphasis on either the training or inverse cost. If the training cost is emphasized, only training cost of the mixed flow ensemble will be included in the regularization term and vice versa. Both "Grow" and "Block" method are used for the optimization and for each block B_i, the number of layers that can be optimized at one time is set to 4, i.e. number of block is 2. The cost regularization weight λ is tuned such that the KL divergence and total cost can be equally minimized.

Manual Flow Setup: Our searched architectures are compared with manually designed flow architectures. In our experiments, we put emphasis on the performance of the manually designed flows, i.e., the manual design will use the transformation with the best performance. For instance, when the training cost is a major concern, we use the rational quadratic autoregressive flow to build the manual design. When the inverse cost is a major concern, we use rational quadratic coupling layers for manual design since we have prior knowledge that the inverse of autoregressive flow is expensive. Detailed experiment setups, as well the hyper parameter settings for each flow, can be found in Appendix C.

Analysis: The architecture search results are reported in Table.1 which includes the negative log likelihood of the test set, and the three different costs. The training cost is consistent with forward cost for different flows. Due to space limitation, we list the searched architectures in Appendix D for reference. Table.1 shows that adding the cost regularization term clearly helps to find architectures that have the lower desired cost. For instance, when the cost emphasize is on inverse calculation, all the searched architectures will not include any autoregressive flow. Consistently, our AutoNF framework can successfully identify architecture with lower preferred cost with only minor degradation on performance compared with manual designs. In some cases it is able to identify architectures

that are better both in terms of performance and cost, such as in the case of [GAS, Training, Grow], [MINOBOONE, Training], [MINOBOONE, Inverse, Block] and [BSDS300, Training].

Comparing the "Grow" method and the "Block" method, we observe that "Block" method can help further optimize the flow architecture to have both better performance and cost compared with "Grow" ([POWER, Inverse], [HEPMASS, Inverse]). While in other cases, it can provides architectures better in at least performance or cost. It is notable that for [HEPMASS, Inverse], even with the same searched architecture types and numbers, the "Block" method can further tune the sequence of transformation layers to further boost the performance.

Table 1: Performance and cost trade-off between searched architectures and human designed architectures on UCI density estimation datasets. Test score is based on negative log likelihood, the lower, the better. Note that the forward cost is consistent to the training cost as expected. The best results for each group of methods are highlited in **bold**.

Datasets	Cost Emphasize	Architectures	Test score	Train cost	Forward cost	Inverse cost
POWER	Training	Manual	**-0.46±0.01**	13.31	11.98	74.38
		Grow	-0.44±0.01	12.62	11.50	50.58
		Block	-0.42±0.01	**10.52**	9.89	47.76
	Inverse	Manual	**-0.41±0.01**	11.46	10.70	10.92
		Grow	-0.36±0.01	10.16	9.69	9.82
		Block	-0.37±0.01	9.73	9.35	**9.46**
GAS	Training	Manual	-10.98±0.02	13.31	11.98	99.17
		Grow	**-11.11±0.02**	12.38	11.34	55.04
		Block	-10.46±0.02	**9.98**	9.51	32.90
	Inverse	Manual	**-10.86±0.03**	11.46	10.70	10.92
		Grow	-10.67±0.02	11.02	10.36	**10.56**
		Block	**-10.86±0.03**	11.46	10.70	10.92
HEPMASS	Training	Manual	**16.62±0.02**	13.31	11.98	260.32
		Grow	18.31±0.02	**9.73**	9.45	9.46
		Block	16.87±0.02	10.90	10.16	200.56
	Inverse	Manual	18.40±0.02	11.46	10.70	10.92
		Grow	18.60±0.02	10.60	10.02	**10.19**
		Block	**18.07±0.02**	10.60	10.02	**10.19**
MINIBOONE	Training	Manual	12.20±0.48	13.31	11.98	533.03
		Grow	11.62±0.44	11.48	10.60	428.87
		Block	**11.43±0.44**	**10.29**	9.73	260.18
	Inverse	Manual	13.48±0.53	11.46	10.70	10.92
		Grow	14.58±0.56	8.00	8.00	**8.00**
		Block	**12.75±0.50**	9.30	9.01	9.09
BSDS300	Training	Manual	-153.83±0.28	13.31	11.98	780.95
		Grow	-154.55±0.28	**10.86**	10.17	198.59
		Block	**-154.57±0.28**	11.22	10.45	339.49
	Inverse	Manual	**-154.02±0.28**	11.46	10.70	10.92
		Grow	-152.08±0.28	8.00	8.00	**8.00**
		Block	-153.71±0.28	9.30	9.01	9.09

5 DISCUSSION

Normalizing flow is highly parameterized module and designing a flow model and use it for application requires a lot of hands-on experience and domain knowledge. In this paper, we show that the AutoNF framework is very effective in balancing performance-cost trade-offs when building complex flow models. Moreover, although not demonstrated in this paper, the framework could also be used to help decide hyper parameters in complex flow model, e.g. the hidden features and number of bins in the SOTA coupling layer (Durkan et al., 2019). In additional, the proposed optimization method with upper bound can be easily extended to other suitable probabilistic kernels. one example is to identify the best parameterized distribution(s) within a mixture model. We believe our framework will be very useful in many machine learning applications where normalizing flows are needed.

REFERENCES

Jens Behrmann, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Joern-Henrik Jacobsen. Invertible residual networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 573–582, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/behrmann19a.html.

Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *CoRR*, abs/1410.8516, 2015.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=HkpbnH9lx.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/7ac71d433f282034e088473244df8c02-Paper.pdf.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. FFJORD: free-form continuous dynamics for scalable reversible generative models. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=rJxgknCcK7.

Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. Neural autoregressive flows. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2078–2087. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/huang18d.html.

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/f33ba15effa5c10e873bf3842afb46a6-Paper.pdf.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/d139db6a236200b21cc7f752979132d0-Paper.pdf.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=BJQRKzbA-.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=S1eYHoC5FX.

D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 2, pp. 416–423 vol.2, 2001. doi: 10.1109/ICCV.2001.937655.

Bogdan Mazoure, Thang Doan, Audrey Durand, Joelle Pineau, and R Devon Hjelm. Leveraging exploration in off-policy algorithms via normalizing flows. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura (eds.), *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pp. 430–444. PMLR, 30 Oct–01 Nov 2020. URL https://proceedings.mlr.press/v100/mazoure20a.html.

Renato Negrinho and Geoff Gordon. Deeparchitect: Automatically designing and training deep architectures, 2017.

Junier Oliva, Avinava Dubey, Manzil Zaheer, Barnabas Poczos, Ruslan Salakhutdinov, Eric Xing, and Jeff Schneider. Transformation autoregressive networks. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3898–3907. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/oliva18a.html.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/6c1da886822c67822bcf3679d04369fa-Paper.pdf.

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference, 2021.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, and Alex Kurakin. Large-scale evolution of image classifiers, 2017.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1530–1538, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/rezende15.html.

Oren Rippel and Ryan Prescott Adams. High-dimensional probability estimation with deep density models, 2013.

Masanori Suganuma, Mete Ozay, and Takayuki Okatani. Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search, 2018.

E. G. Tabak and Cristina V. Turner. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, February 2013. ISSN 0010-3640. doi: 10.1002/cpa.21423.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=r1Ue8Hcxg.

A ADDITIONAL BACKGROUND

A.1 BASICS OF NORMALIZING FLOW

Normalizing flow aims to provide a way to do exact density modeling of complex distributions by finding a diffeomorphsim between two manifolds. Suppose there are two random variables $\boldsymbol{x}$ and $\boldsymbol{u}$ from two distributions $p_{\boldsymbol{x}}(\boldsymbol{x})$ and $p_{\boldsymbol{u}}(\boldsymbol{u})$. If there is an invertible and differentiable transformation, which is parameterized by $\boldsymbol{\theta}$, that can transform $\boldsymbol{u}$ to $\boldsymbol{x}$, then the density function of $\boldsymbol{x}$ can be represented in eq.(13).

$$\begin{aligned} \boldsymbol{x} &= T_{\boldsymbol{\theta}}(\boldsymbol{u}) \\ p_{\boldsymbol{x}}(\boldsymbol{x}) &= p_{\boldsymbol{u}}(T_{\boldsymbol{\theta}}^{-1}(\boldsymbol{x})) \cdot |\det \boldsymbol{J}_{T_{\boldsymbol{\theta}}^{-1}}(\boldsymbol{x})| \end{aligned} \tag{13}$$

Where $\det \boldsymbol{J}_{T_{\boldsymbol{\theta}}^{-1}}(\boldsymbol{x})$ is the Jacobian determinant of the inverse transformation $T_{\boldsymbol{\theta}}^{-1}$.Let the distribution $p_{\boldsymbol{x}}(\boldsymbol{x})$ be a target complex distribution which we want to evaluate the density or draw samples from, and $p_{\boldsymbol{u}}(\boldsymbol{u})$ is a simple distribution. Constructing a flow model is to find a transformation that minimizing the KL divergence between the two distributions, as summarized in (Papamakarios et al., 2021). After that, density evaluation and random sampling of $p_{\boldsymbol{x}}(\boldsymbol{x})$ can be easily done with eq.(13). In practice, multiple layers of transformations are stacked together to increase the expressive power of the flow model, as shown in Fig. 4.

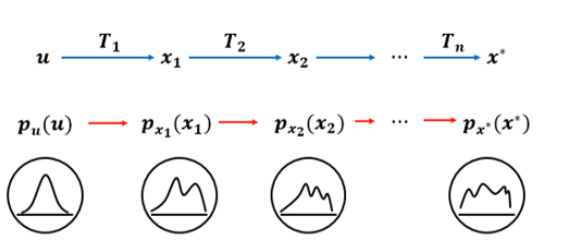


Figure 4: Normalizing flow is constructed by multiple layers of transformations.

A.2 AUTOREGRESSIVE FLOW (AF)

AF transforms a D dimensional input random variable $\boldsymbol{x}$ via dimension-wise scalar functions, termed transformer τ. The parameter $\boldsymbol{h}_i$ of each transformer τ_i should only dependent on inputs whose dimension is smaller than i. Usually, a single masked neural network is implemented to generate the parameters. The transformation of AF is shown in eq. (14).

$$x'_i = \tau(x_i; \boldsymbol{h}_i) \quad \text{where} \quad \boldsymbol{h}_i = \text{NN}(\boldsymbol{x}_{<i}) \tag{14}$$

Autoregressive flow has a triangular Jacobian and the determinant is the product of diagonal elements which can be calculated in $\mathcal{O}(D)$ complexity. Moreover,the forward transformation of autoregressive flow can be calculated within a single neural network and transformer pass. However, since x_i has dependency on $\boldsymbol{x}_{<i}$, the inverse calculation can only be accomplished sequentially, hence is $\mathcal{O}(D)$ times more expensive than the forward pass (Papamakarios et al., 2021).

A.3 COUPLING LAYER

A coupling layer splits the input into two groups. The first group will be passed with no change (or identity transformation). The second group will be transformed by dimension-wise scalar functions whose parameters are dependent on the first group, as shown in eq. (15).

$$\begin{aligned} &\boldsymbol{x}'_{1\sim d} = \boldsymbol{x}_{1\sim d}\ , \quad x'_i = \tau(x_i; \boldsymbol{h}_i) \\ &\text{where} \quad \boldsymbol{h}_i = \text{NN}(\boldsymbol{z}_{<i}) \quad i \in [d+1, D] \end{aligned} \tag{15}$$

Similar to autoregressive flow, the Jacobian matrix of the coupling layer is triangular, which enables one-pass calculation of the forward transformation. Meanwhile, the inverse transformation can also

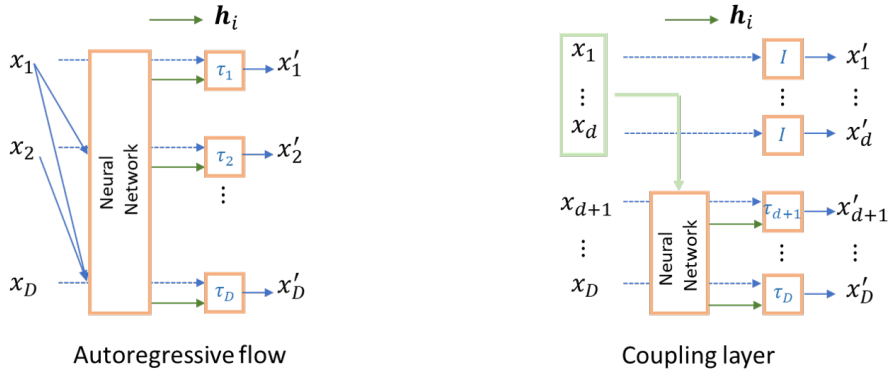


Figure 5: An illustration of autoregressive flow and coupling layer

be calculated with a single neural network and one transformer inverse. However, the computation efficiency of the coupling layer is achieved at the cost of limited expressive power. In most cases, with the same type of transformer, additional layers of transformations are required for coupling layers to achieve the same performance compared to autoregressive flows. A illustration of autoregressive flow and coupling layer is shown in Fig.5

A.4 Invertible Linear Transformation

Permutation between two autoregressive or coupling layers are commonly introduced to ensure good interaction between different dimensions. Random permutation and reverse permutation have been adopted in (Dinh et al., 2015; Kingma & Dhariwal, 2018). Authors in (Rippel & Adams, 2013) proposed a generalization with an LU-decomposed linear transformation: $\boldsymbol{W} = \boldsymbol{PLU}$, where $\boldsymbol{P}$ is a fixed permutation matrix and the diagonal elements of $\boldsymbol{L}$ are all one. The linear transformation is invertible if the diagonal elements of $\boldsymbol{U}$ are all positive. The Jacobian determinant of the linear transformation is the product of all diagonal elements of $\boldsymbol{U}$ and forward transformation is a simple matrix multiplication. Although the inverse transformation has $\mathcal{O}(D^3)$ complexity, as suggested in (Durkan et al., 2019), once the parameters of $\boldsymbol{W}$ are determined, $\boldsymbol{W}^{-1}$ can be computed once and cached for further use.

A.5 Residual flow

Another way of constructing finite flows is to implement the following invertible transformation:

$$\boldsymbol{z}' = \boldsymbol{z} + \boldsymbol{g}_\phi(\boldsymbol{z}) \tag{16}$$

$_\phi(\boldsymbol{z})$ takes input random variable as input and generates a D dimensional translation vector.

The invertibility of residual flow is guaranteed if $\boldsymbol{g}_\phi(\boldsymbol{z})$ can be made contractive to some distance function (Papamakarios et al., 2021). In the later sections for different residual flows, we assume all the transformations are invertible.

Also, different from autoregressive flow, the Jacobian determinant of residual flow is no longer sparse. To reduce the complexity, researchers have proposed several $\boldsymbol{g}_\phi(\boldsymbol{z})$ to complete the determinant calculation in $\mathcal{O}(D)$ time.

A.5.1 Planar Flow

Planar flow (Rezende & Mohamed, 2015) can be described as:

$$\boldsymbol{z}' = \boldsymbol{z} + \boldsymbol{v}\sigma(\boldsymbol{w}^T\boldsymbol{z} + b) \tag{17}$$

In the transformation, $\boldsymbol{v}$ and $\boldsymbol{w}$ are both D dimensional vector and $\sigma(\cdot)$ is the activation function. The residual part $\boldsymbol{g}_\phi(\boldsymbol{z})$ can be interpreted as a neural network which has no hidden layer and has only one neuron.

The Jacobian determinant of the planar flow can be analytically written as:

$$\det \boldsymbol{J}_T(\boldsymbol{z}) = 1 + \sigma'(\boldsymbol{w}^T\boldsymbol{z} + b)\boldsymbol{w}^T\boldsymbol{v} \tag{18}$$

A.5.2 RADIAL FLOW

Radial flowRezende & Mohamed (2015) can be written as:

$$\boldsymbol{z}' = \boldsymbol{z} + \frac{\beta}{\alpha + r(\boldsymbol{z})}(\boldsymbol{z} - \boldsymbol{z}_0), \quad \text{where} \quad r(\boldsymbol{z}) = ||\boldsymbol{z} - \boldsymbol{z}_0|| \tag{19}$$

α is restricted to be a positive value, and a sufficient condition for existing inversion is $\beta > -\alpha$. The Jacobian determinant of radial flow can also be calculated in $\mathcal{O}(D)$ complexity as:

$$\det \boldsymbol{J}_T(\boldsymbol{z}) = (1 = \frac{\alpha\beta}{(\alpha + r(\boldsymbol{z}))^2})(1 = \frac{\beta}{\alpha + r(\boldsymbol{z})})^{D-1} \tag{20}$$

B COST MODELING OF FLOWS

The cost of different flows models are extracted by stacking different number of layers (1, 2, 4, 8) to do training for 3000 iterations, run forward transformation for 10000 data point and draw 10000 random samples. For fair comparison and reproducity, each flow model is run with Pytorch on a AMD Ryzen 5800X CPU for 3 times to take average and linear regression is applied to extract the exact cost. The final cost is normalized based on affine coupling layer. The experiment on POWER dataset is shown in Table.2.

Table 2: Cost extraction of training cost, forward cost and inverse cost with POWER dataset.

		Affine MAF			RQ MAF			Affine Coupling			RQ Coupling		
		train	forward	inverse	train	forward	inverse	train	forward	inverse	train	forward	inverse
Number of Layers	1	13.501	0.083	0.278	20.565	0.121	0.436	13.279	0.111	0.08	18.164	0.107	0.067
	2	26.098	0.145	0.573	39.354	0.202	0.844	24.512	0.135	0.099	35.082	0.176	0.121
	4	53.642	0.26	1.164	79.774	0.326	1.726	47.757	0.255	0.191	68.742	0.286	0.242
	8	106.459	0.422	2.381	163.375	0.607	3.389	99.146	0.419	0.389	141.38	0.535	0.499
Cost		13.324	0.0479	0.3007	20.482	0.068	0.423	12.309	0.0454	0.0455	17.625	0.0607	0.0621
Cost(Normalized)		1.0825	1.0551	1.1015·D	1.664	1.4978	1.5495·D	1	1	1	1.4319	1.337	1.3648

C DETAILED EXPERIMENTAL SETUPS

The training data of each dataset is split into two equal part as the "training data" and "validation data" for architecture optimization. For all data set, we set the training and validation batch size to 512 and the optimization of architecture parameter and model parameters are done with Adam with a learning rate of are set to be 0.005. For the "Grow" method, the training iterations for each block is 5000 and for "Block" method, the training iteration for each block is 3000 and the whole flow will be trained with 2 rounds, i.e. each block will be trained twice. The cost regularization weights for each dataset (from POWER to BSDS300) are: 0.1, 0.1, 1, 1, 5. The hyper parameter settings of different flow models are summarized in Table.3.

Table 3: Hyper parameter setting for all autoregressive flow and coupling layers.

	Affine MAF	RQ-MAF	Affine Coupling	RQ coupling
Hidden Feature	256	256	256	256
Residual Blocks	2	2	2	2
Bins	/	8	/	8
Tail bound	/	3	/	3
Dropout	0	0	0	0

For evaluation of searched flow models, we use Adam optimizer to optimize the flow parameter with a learning rate of 0.0005 and the gradient norm is clipped into $[-5, 5]$ as adopted in (Durkan et al., 2019). The batch size is set to be 512 and trained for 30000 iterations. For MINIBOONE dataset, overfitting is a problem and we only train for 5000 iterations. The models with best validation negative log likelyhood are stored and tested with test data. The mean value and standard deviation is reported as the final test score.

D SEARCHED ARCHITECTURES

The full table of searched architectures in the experiments are listed as in Table.4. The number is used as a code name for one type of the flow: where 1, 2, 3, 4, 5 stands for affine MAF, rational quadratic MAF, affine coupling layer, rational quadratic coupling layer and a combined layer of a reverse permutation and an LU linear layer.

Table 4: Searched architectures by AutoNF framework with emphasize on different cost.

Dataset	Cost Emphasize	Design method	Searched architectures
POWER	Training	Manual	[52, 52, 52, 52, 52, 52, 52, 52]
		Grow	[54, 52, 52, 52, 54, 54, 52, 52]
		Block	[54, 52, 54, 52, 51, 51, 51, 51]
	Inverse	Manual	[54, 54, 54, 54, 54, 54, 54, 54]
		Grow	[54, 54, 53, 54, 53, 54, 53, 54]
		Block	[54, 54, 54, 54, 53, 53, 53, 53]
GAS	Training	Manual	[52, 52, 52, 52, 52, 52, 52, 52]
		Grow	[54, 54, 52, 52, 54, 54, 52, 52]
		Block	[54, 54, 54, 54, 53, 51, 51, 51]
	Inverse	Manual	[54, 54, 54, 54, 54, 54, 54, 54]
		Grow	[54, 54, 54, 53, 54, 54, 54, 54]
		Block	[54, 54, 54, 54, 54, 54, 54, 54]
HEPMASS	Training	Manual	[52, 52, 52, 52, 52, 52, 52, 52]
		Grow	[54, 54, 53, 53, 54, 54, 53, 53]
		Block	[52, 52, 52, 52, 53, 51, 51, 51]
	Inverse	Manual	[54, 54, 54, 54, 54, 54, 54, 54]
		Grow	[53, 54, 54, 54, 54, 53, 54, 54]
		Block	[54, 54, 53, 53, 54, 54, 54, 54]
MINIBOONE	Training	Manual	[52, 52, 52, 52, 52, 52, 52, 52]
		Grow	[52, 52, 51, 51, 52, 52, 52, 53]
		Block	[54, 52, 54, 54, 51, 51, 51, 51]
	Inverse	Manual	[54, 54, 54, 54, 54, 54, 54, 54]
		Grow	[53, 53, 53, 53, 53, 53, 53, 53]
		Block	[54, 53, 54, 54, 53, 53, 53, 53]
BSDS300	Training	Manual	[52, 52, 52, 52, 52, 52, 52, 52]
		Grow	[54, 52, 52, 53, 54, 52, 53, 53]
		Block	[54, 54, 52, 52, 51, 54, 51, 54]
	Inverse	Manual	[54, 54, 54, 54, 54, 54, 54, 54]
		Grow	[53, 53, 53, 53, 53, 53, 53, 53]
		Block	[54, 54, 53, 53, 54, 53, 53, 53]