Cost-aware Graph Generation: A Deep Bayesian Optimization Approach

Jiaxu Cui,[1,2] **Bo Yang,**[1,2*] **Bingyi Sun,**[1,2,4] **Jiming Liu** [3]

[1]College of Computer Science and Technology, Jilin University, China
[2]Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, China
[3]Department of Computer Science, Hong Kong Baptist University, Hong Kong
[4]National Laboratory for Parallel and Distributed Processing, National University of Defense Technology, China
jxcui16@mails.jlu.edu.cn, ybo@jlu.edu.cn, bysun15@mails.jlu.edu.cn, jiming@comp.hkbu.edu.hk

Abstract

Graph-structured data is ubiquitous throughout the natural and social sciences, ranging from complex drug molecules to artificial neural networks. Evaluating their functional properties, e.g., drug effectiveness and prediction accuracy, is usually costly in terms of time, money, energy, or environment, becoming a bottleneck for the graph generation task. In this work, from the perspective of saving cost, we propose a novel Cost-Aware Graph Generation (CAGG) framework to generate graphs with optimal properties at as low cost as possible. By introducing a robust Bayesian graph neural network as the surrogate model and a goal-oriented training scheme for the generation model, the CAGG can approach the real expensive evaluation function and generate search space close to the optimal property, to avoid unnecessary evaluations. Intensive experiments conducted on two challenging real-world applications, including molecular discovery and neural architecture search, demonstrate its effectiveness and applicability. The results show that it can generate the optimal graphs and reduce the evaluation costs significantly compared to the state-of-the-art.

Introduction

Deep graph generative models have recently become a popular research topic with their promising applications, including real-world network imitation (Bojchevski et al. 2018), inverse design of materials (Sanchez-Lengeling and Aspuru-Guzik 2018), and chemical molecular generation (Li, Zhang, and Liu 2018; Jin, Barzilay, and Jaakkola 2018; Ma, Chen, and Xiao 2018; Qi et al. 2018; You et al. 2018; Samanta et al. 2019). These deep models produce more fascinating, practical, and complex structures than simple random graph generation methods, such as Erdős-Rényi model (Erdős and Rényi 1959) and Barabási-Albert model (Barabási and Albert 1999). Beyond learning the distribution of an existing dataset, to produce optimal graphs for some certain goals in practical applications is the expectation, such as discovering the molecules with the best drug characteristics (Zhavoronkov et al. 2019) and designing neural architectures with the excellent performance (Ma, Cui, and Yang 2019).

Recently, some efforts have been made to explore the goal-oriented graph generative task, which has been formulated as reinforcement learning (RL) (Guimaraes et al. 2017; Baker et al. 2017; Zoph et al. 2018; Cao and Kipf 2018; Bojchevski et al. 2018; You et al. 2018; Zhavoronkov et al. 2019; Jin, Barzilay, and Jaakkola 2020b), graph-to-graph translation (Jin et al. 2019; Jin, Barzilay, and Jaakkola 2020a), continuous optimization over the latent space learned by variational autoencoders (VAEs) (Gómez-Bombarelli et al. 2018; Kusner, Paige, and Hernández-Lobato 2017; Dai et al. 2018; Qi et al. 2018; Jin, Barzilay, and Jaakkola 2018; Samanta et al. 2019; Zhang et al. 2019; Luo et al. 2018), and optimization directly on graph space (Ramachandram et al. 2018; Kandasamy et al. 2018; Jin, Song, and Hu 2019; Ma, Cui, and Yang 2019). However, these existing approaches usually require a large number of evaluations, such as 34K∼99K (Jin et al. 2019), 5K (Guimaraes et al. 2017; Cao and Kipf 2018), and 3K (Samanta et al. 2019).

Moreover, the evaluation costs of practical graphs are usually expensive in terms of time, money, energy, or environment. When evaluating the classification performance of a single deep VGG network, the training phase on a system equipped with four NVIDIA Titan Black GPUs takes 2∼3 weeks (Simonyan and Zisserman 2015). Based on the carbon emission estimation model (Strubell, Ganesh, and McCallum 2019), the estimated CO_2 emission in this training stage alone has reached 506∼760 lbs, which is roughly equivalent to a round trip by a car from Los Angeles to Las Vegas (CSS 2018). To evaluate the chemical properties of a single 9 heavy atom molecule via an expensive density functional theory (DFT) calculation (Zhang and Musgrave 2007) on a single-core processor takes around an hour (Gilmer et al. 2017). Assuming that the molecular evaluation time is one hour, the aforementioned goal-oriented graph generative methods would take about four months to eleven years, roughly estimated from the number of their assessments. Such a high evaluation cost will become a bottleneck in practical applications.

Motivated by these, from the perspective of saving evaluation cost, we propose to study *how to generate the graphs with optimal properties at as low cost as possible*. In this work, we propose a novel cost-aware framework, named as Cost-Aware Graph Generation (CAGG), to solve this problem. The CAGG is composed of three key components: a surrogate model, a generation model, and an acquisition

*Corresponding author.

function. The surrogate model is to approximate the high-cost evaluation function of graphs and can predict a dummy landscape for real goal. The generation model is to generate graphs as a search space. To generate a desirable search space composed of graphs close to optimal properties, we train the generation model using the predicted dummy landscape to avoid real evaluation. An acquisition function is to choose an optimal candidate graph from the search space. Through the cooperation of three components, our framework can thus generate the graphs with optimal properties under low evaluation costs. Our main contributions are summarized as follows.

- The CAGG is proposed to generate the graphs with optimal properties at as low cost as possible.
- We propose a robust Bayesian graph neural networks as the surrogate model to approach real expensive-to-evaluate objective closely.
- We design a goal-oriented training scheme for the generation model to produce desirable search space close to optimal properties.
- The effectiveness and applicability of the CAGG are evaluated on two challenging real-world applications, including molecular discovery and neural architecture search. The CAGG outperforms the state-of-the-art significantly.

Related Work

In recent years, reinforcement learning (RL) has been used to guide the generators, which are usually modeled by a generative adversarial network (GAN) or a variational autoencoder (VAE), to move forward in the desired direction for goal-oriented graph generation (Guimaraes et al. 2017; Baker et al. 2017; Zoph et al. 2018; Cao and Kipf 2018; Bojchevski et al. 2018; You et al. 2018; Zhavoronkov et al. 2019; Jin, Barzilay, and Jaakkola 2020b). The latest RL-based methods introduced domain constraints, including valence rules (You et al. 2018) and molecular substructures (Jin, Barzilay, and Jaakkola 2020b), into the generation policy to ensure the molecular validity by representing molecules as graphs rather than as SMILES strings (Guimaraes et al. 2017). A current generative tensorial RL approach (Gentrl) is developed to accelerate drug development (Zhavoronkov et al. 2019). It pre-trained a VAE with a learned prior as a generator, and then learned the prior via the RL for the desired molecular generation. Although the generative policy in these RL-based methods can move in the desired direction gradually, they usually require a large number of evaluations, e.g., 5K evaluated molecules (Guimaraes et al. 2017; Cao and Kipf 2018) or 12.8K trained nets (Zoph et al. 2018).

Jin et al. (2019) proposed to transform a goal-oriented generation problem into a graph-to-graph (G2G) translation problem. This way maps an input graph into a better graph by performing adversarial regularization on a pre-trained VAE. Jin, Barzilay, and Jaakkola (2020a) introduced motifs as building blocks for generating molecular graphs to deal with larger molecules. However, these methods need to supervised train using large-scale evaluated bad-good pairs, e.g., 34K$\sim$99K evaluated molecular pairs (Jin et al. 2019).

Continuous optimization over the latent space learned by variational autoencoders (VAEs) is another popular way. These methods encoded graphs into continuous representations via a VAE, and then applied the Bayesian optimization (BO) (Shahriari et al. 2016) or gradient-based optimization methods to find the optimal representation over the latent space to decode the found representation into desirable graphs (Gómez-Bombarelli et al. 2018; Kusner, Paige, and Hernández-Lobato 2017; Dai et al. 2018; Qi et al. 2018; Jin, Barzilay, and Jaakkola 2018; Samanta et al. 2019; Zhang et al. 2019; Luo et al. 2018). They, however, usually train the VAE unsupervised; that is, these learned continuous encodings may not always be appropriate for specific tasks. Moreover, they still need many evaluations, e.g., 3K$\sim$90K evaluated molecules (Jin, Barzilay, and Jaakkola 2018; Samanta et al. 2019) or 1K$\sim$5K trained neural architectures (Luo et al. 2018; Zhang et al. 2019), due to the high-dimensional nature of the continuous latent space, e.g.,196 dimensions. To bypass the high-dimensional issue and use graph structures, some other BO-related methods operate directly on the graphs (Ramachandram et al. 2018; Kandasamy et al. 2018; Jin, Song, and Hu 2019). However, their predetermined kernels are explicitly and exclusively designed for neural architecture search (NAS), resulting in limited applications. A more recent approach, the DGBO (Cui, Yang, and Hu 2019), used a Bayesian graph neural network as the surrogate model, where a Bayesian linear regressor is applied to the last layer. However, unlike the above generative methods and our work, it only performs the search in a given fixed search space but does not produce new graphs. Built on the DGBO, the NASGBO (Ma, Cui, and Yang 2019) is to optimize the neural nets by combining an evolutionary algorithm that is just used for acquisition function optimization instead of obtaining the generative mechanism of graphs.

In addition, there are some task-specific approaches, especially for the NAS. These methods usually reduce the extra training costs by joint training and search process (Liu, Simonyan, and Yang 2019; Lu et al. 2020), fine-tuning the nets instead of training from scratch (Cai et al. 2020), or introducing performance predictor (Kokiopoulou et al. 2020; Li et al. 2020), etc. They, however, are difficult to extend to other fields due to the different intrinsic assessments.

In contrast to existing approaches, our method focuses on reducing the evaluation costs in generating the optimal graphs. Moreover, since our method can automatically learn from data without explicitly designing kernels, it can deal with a variety of graphs in many fields.

CAGG: Cost-Aware Graph Generation

In this section, we propose a framework to generate graphs with optimal properties at as low cost as possible.

First, we define a graph G with d_x node types and d_y edge types as consisting of four tuples (V, E, X, Y), where V is a set of nodes, $E \subseteq (V \times V)$ is a set of edges, and $X \subseteq \mathbb{R}^{|V|\times(1+d_x)}$ and $Y \subseteq \mathbb{R}^{|E|\times(1+d_y)}$ are the attribute matrices of all nodes and all edges respectively. Note that the 0-th index in the column of the attribute matrices indicates whether the node/edge exists (0) or not (1). Thus, the

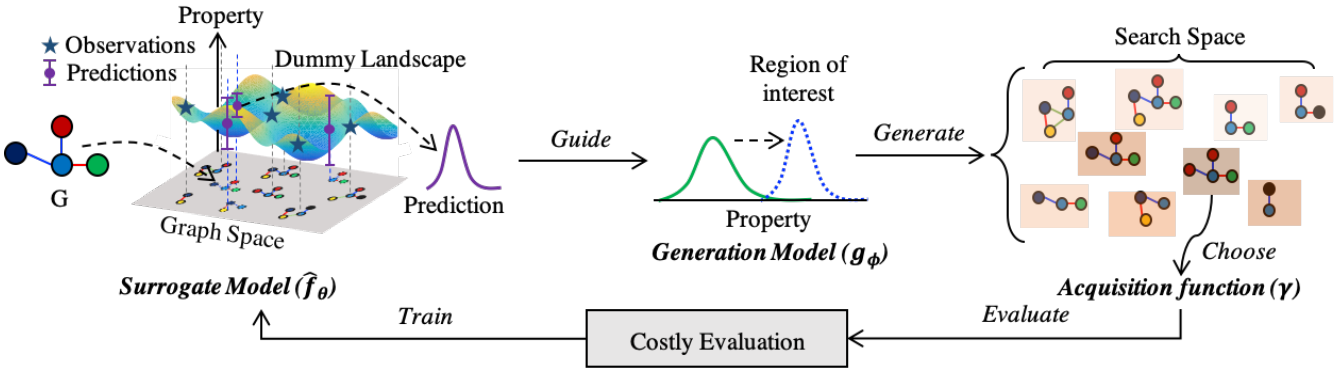

Figure 1: Overview of the CAGG framework. A surrogate model $\hat{f}_{\boldsymbol{\theta}}$ is to approach the expensive-to-evaluate function f, which is trained based on current observed data and can predict a dummy landscape with uncertainty. A generation model $g_{\boldsymbol{\phi}}$ is to produce desirable graphs as a search space, which is guided by the dummy landscape. An acquisition function γ is to choose a graph from a search space. After choosing, the selected graph is evaluated by the costly f to obtain its corresponding property and update current observed data. Through the cooperation of these three components, the CAGG effectively generates graphs with optimal properties under low evaluation costs.

node types range from 1 to d_x and the edge types from 1 to d_y. Given a graph G and the costly evaluation function f, we can evaluate the property of G using $z = f(G)$.

Overview of the CAGG

The CAGG is an iterative framework. Its main idea at each iteration is to use a generation model to generate the desirable graphs, whose properties are close to the optimal, and then use a deep Bayesian optimization to select an optimal candidate for real evaluation. Specifically, it is composed of three key components: a surrogate model, a generation model, and an acquisition function (see Fig. 1).

A surrogate model $\hat{f}_{\boldsymbol{\theta}}$ is to approximate the expensive-to-evaluate function f, where $\boldsymbol{\theta}$ is the parameters of this surrogate model. It is trained based on current observed data $\mathcal{D} = \{(G_1, z_1), (G_2, z_2), ...\}$ and can predict a dummy landscape with uncertainty for real f. The generation model is to produce graphs as a search space. We denote $g_{\boldsymbol{\phi}}$ as a generation model, where $\boldsymbol{\phi}$ is the parameters of the generation model. To generate a desirable search space composed of graphs with good properties and reduce the possibility of evaluating undesirable candidates, the generation model is guided by the dummy landscape to generate desirable search space $\mathcal{G}$. The advantage of the dummy landscape is that we avoid real evaluations while guiding the generation model. Since the generation model is trained continuously, we obtain the generative mechanism of the optimal graphs simultaneously. An acquisition function γ is to choose a candidate graph from a search space $\mathcal{G}$. This strategy can balance exploration and exploitation by using the predictive distribution of graph properties, to reduce unnecessary evaluation. After choosing, we evaluate the selected graph G' via the costly f to obtain its corresponding property z' and update current observed data $\mathcal{D}$. Through the cooperation of these three components, the CAGG effectively designs optimal graphs under low costs.

The specific procedure is described in Algorithm 1. To increase the validity of the graphs generated in the early stage, a generation model g_{ori} performs unsupervised training on a certain number of graphs in a VAE framework. We produce several graphs (e.g., 500) using $g_{\boldsymbol{\phi}}$ as the search space (line 7). To increase the diversity of candidates, some graphs (e.g., 500) generated by g_{ori} are also integrated into the search space. When reaching a given budget B, which could be a maximum running time or a maximum number of evaluations, the process terminates and returns the optimum.

We detail each key component in the rest of this section.

Algorithm 1 Procedure of the CAGG

Input: A budget B, the number of graphs for initial evaluations M, and the training interval K;
Output: The optimal graph;

1: Pre-train a generator g_{ori} in an unsupervised way;
2: $g_{\boldsymbol{\phi}} \leftarrow g_{ori}$;
3: Initialize M graphs randomly and evaluate them into $\mathcal{D} \leftarrow \{(G_i, z_i)\}_{i=1}^{M}$;
4: **while** B is not reached **do**
5: Every K steps update $\hat{f}_{\boldsymbol{\theta}}$ based on current $\mathcal{D}$;
6: Every K steps update $g_{\boldsymbol{\phi}}$ using $\hat{f}_{\boldsymbol{\theta}}$;
7: Generate search space $\mathcal{G}$;
8: Choose a G' from $\mathcal{G}$ by maximizing γ;
9: Evaluate G' and augment it into $\mathcal{D} \leftarrow \mathcal{D} \cup \{(G', z')\}$;
10: **return** The optimum in $\mathcal{D}$.

Surrogate Model $\hat{f}_{\boldsymbol{\theta}}$: Robust Bayesian Graph Neural Network

As a core component of reducing the cost, the surrogate model should have the ability to approach f under a small number of evaluations. To achieve this, we thus introduce a robust Bayesian graph neural network, where each layer is discussed as follows.

Embedding layer. Since the original features on nodes

and edges are usually sparse, such as one-hot encodings, we use an embedding layer to convert these sparse features to dense representations. The specific operations to update the edge e and node i are as follows:

$$F_e^{(0)} = (1 - Y_{e,0}) \times \psi_E^{(em)}(Y_{e,1:}), \quad (1)$$

$$H_i^{(0)} = (1 - X_{i,0}) \times \psi_V^{(em)}(X_{i,1:}), \quad (2)$$

where $\psi_E^{(em)}$ and $\psi_V^{(em)}$ denote the trainable networks for edges and nodes respectively. Note that if edge e does not exist, then $(1 - Y_{e,0})$ is equal to 0, the same for node i. We discard the update of non-existent nodes or edges to avoid their effect on learning features by multiplying these weights.

Graph network layer. To learn the features of graphs, we use a simple message-passing strategy to fuse structure and attribute information of graphs. Specifically, we pass the messages on nodes and edges multiple rounds. At t-th round, we update the feature of edge $e : i \to j$ through its neighbor nodes i and j by

$$F_e^{(t)} = (1 - Y_{e,0}) \times \psi_E^{(gn)}([F_e^{(t-1)}, H_i^{(t-1)}, H_j^{(t-1)}]), \quad (3)$$

where $\psi_E^{(gn)}$ is a trainable update neural network for edges.

And then, we update the node features through the aggregation of neighbor edge information by

$$H_i^{(t)} = (1 - X_{i,0}) \times \psi_V^{(gn)}([H_i^{(t-1)}, \sum_{e \in \aleph(i)} F_e^{(t-1)}]), \quad (4)$$

where $\psi_V^{(gn)}$ is a trainable neural network for nodes and $\aleph(i)$ is a set of all edges pointing to node i.

Note that, in these two update operations, we still shut off the message passing process at non-existent nodes and edges to stop their influence by multiplying the weights.

In this way, through multiple rounds (e.g., T) of message passing, we collect the output of each round as the final output of this layer, i.e.,$\{H^{(t)}\}_{t=1}^{T}$.

Global pooling layer. We calculate the entire graph's representation h_G from the output of the GN layer by

$$\boldsymbol{h}_G = \mathit{concat}(\sum_{i \in V} H_i^{(t)} \mid t = 1, 2, ..., T), \quad (5)$$

where *concat*(.) is a concatenation operation.

Readout layer. After obtaining the graph representation $\boldsymbol{h}_G$, we feed it into a Multi-Layer Perceptron (MLP) followed by a linear layer for prediction.

All parameters of the surrogate model are uncertain. Let $\boldsymbol{\theta}$ be the trainable parameters of the surrogate model. We treat all parameters $\boldsymbol{\theta}$ as random variables and place a prior distribution $p(\boldsymbol{\theta})$ on them. Thus, predictive distribution for a new graph G' can be formed by integrating with respect to the posterior distribution $p(\boldsymbol{\theta} \mid \mathcal{D})$ of $\boldsymbol{\theta}$ as follows:

$$p(z' \mid G', \mathcal{D}) = \int p(z' \mid G', \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid \mathcal{D}) d\boldsymbol{\theta}, \quad (6)$$

where $\mathcal{D} = \{(G_i, z_i)\}_{i=1}^{n}$ denotes a set of n evaluated graphs. This integral, however, is in general intractable. Fortunately, it has been shown that Monte Carlo dropout is equivalent to drawing samples of $\boldsymbol{\theta}$ from the approximate posterior when choosing a suitable variational approximation for the posterior $p(\boldsymbol{\theta} \mid \mathcal{D})$ (Gal and Ghahramani 2016). Thus, in this work, we use a Monte Carlo dropout technology (Gal and Ghahramani 2016) to approximate the integral. Specifically, we sample S $\boldsymbol{\theta}_i$ via dropout and then approximate Eq. 6 by

$$p(z' \mid G', \mathcal{D}) \approx \frac{1}{S} \sum_{i=1}^{S} p(z' \mid G', \boldsymbol{\theta}_i). \quad (7)$$

Moreover, to facilitate the calculation of the acquisition function, we place a Gaussian on the predictive distribution, i.e., $\mathcal{N}(\mu(G'), \sigma^2(G'))$. Therefore, after training the surrogate with dropout using l_2 loss, the predictive mean and variance of a new G' can be calculated practically as follows:

$$\mu(G') \approx \frac{1}{S} \sum_{i=1}^{S} \hat{f}_{\boldsymbol{\theta}_i}(G'), \quad (8)$$

$$\sigma^2(G') \approx \frac{1}{S} \sum_{i=1}^{S} (\hat{f}_{\boldsymbol{\theta}_i}(G') - \mu(G'))^2. \quad (9)$$

Generation Model $g_{\boldsymbol{\phi}}$: Goal-Oriented Search Space Generation

The second key component is a generation model denoted as $g_{\boldsymbol{\phi}} = p_{\boldsymbol{\phi}}(G \mid \boldsymbol{r})$ that can generate graphs by feeding a random vector $\boldsymbol{r}$ sampled from a multivariate standard Gaussian distribution. Following (Ma, Chen, and Xiao 2018), we use a multi-layer deconvolutional neural net as $g_{\boldsymbol{\phi}}$. Given the maximum number of nodes N, we denote the output of $g_{\boldsymbol{\phi}}$ as $P \in \mathbb{R}^{N \times [(1+d_x)+N \times (1+d_y)]}$ that can cover all graphs with nodes no more than N. The generated probability template, thus, can be represented by $\widetilde{G} = (\widetilde{V}, \widetilde{E}, \widetilde{X}, \widetilde{Y})$, where $\widetilde{V}$ and $\widetilde{E}$ denote all nodes and edges of the complete graph with N nodes, $\widetilde{X} \in \mathbb{R}^{N \times (1+d_x)}$ is the first $(1 + d_x)$ columns of P after softmax operation, and $\widetilde{Y} \in \mathbb{R}^{N^2 \times (1+d_y)}$ is the rest of P after reshaping and softmax operation. And then, candidate graphs can be sampled from this template.

Now, the problem we face is *how to train this model to produce the desired graph search space?* In this work, we design a *two-phase* process to train the generation model.

The *first* phase is an unsupervised pre-training. To increase the validity of the graphs generated in the early stage, we train the generation model on a certain number of graphs (i.e., 1,000) in a VAE framework by combining an encoder (see line 1 in Algorithm 1). The encoder has the same architecture as the proposed surrogate model, except for replacing the last linear layer with two separate fully connected layers to generate the mean and variance of the latent vector.

The *second* phase is to learn the pre-trained generation model towards the goals (see line 6 in Algorithm 1). To produce graphs with good properties as search space and reduce unnecessary evaluations, we propose a goal-oriented training scheme. Specifically, the objective is formulated as:

$$\boldsymbol{\phi}^* = \arg\max_{\boldsymbol{\phi}} \mathbb{E}_{G \sim g_{\boldsymbol{\phi}}}[f(G)] + \sum_i \lambda_i c_i(\boldsymbol{\phi}), \quad (10)$$

where c_i denotes the constraint i on the generation model, and λ_i is a coefficient for constraint i.

The first term in this objective can guide the generation model to the desired direction, but it contains the costly evaluation function f. Thus, we replace f with $\hat{f}_{\boldsymbol{\theta}}$ to avoid real evaluation in training. However, we need to sample predictions from the predictive distribution outputted by the surrogate model. Herein, we use a reparameterization technique to deal with the issue of nondifferentiable sampling, i.e., the prediction is $\frac{1}{L}\sum_{l=1}^{L}[\mu(.)+\epsilon_l\sigma(.)]$, where $\epsilon_l \sim \mathcal{N}(0,1)$, L is the number of samples of ϵ in prediction, and μ and σ are the predictive mean (Eq. 8) and standard deviation (Eq. 9), respectively. Moreover, we replace the essentially nondifferentiable graph G with the differentiable probability template $\widetilde{G}$. The objective, thus, can be converted to

$$\boldsymbol{\phi}^* = \arg\max_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{r}\sim\mathcal{N}(0,I)}[\hat{z}] + \sum_i \lambda_i c_i(\boldsymbol{\phi}), \tag{11}$$

where $\hat{z} = \frac{1}{L}\sum_{l=1}^{L}[\mu(\widetilde{G})+\epsilon_l\sigma(\widetilde{G})]$ and $\widetilde{G} = g_{\boldsymbol{\phi}}(\boldsymbol{r})$. We, then, learn the parameters $\boldsymbol{\phi}$ in the generation model by performing a stochastic gradient ascent algorithm on Eq. 11.

The second term in Eq. 11 (c_i) is to constrain the generation model, such as to restrict the generated graphs to be connected. It can be designed according to different tasks and should also be differentiable. Here, we give an example for molecules used in our experiments (please see Section S1 in the supplementary material[1] for the constraint details on neural architectures).

Constraints on molecules. First, we introduce a constraint on connectivity. A molecular graph is connected if there is a path between every pair of existent nodes. To be differentiable, we use a probabilistic adjacency matrix denoted $\widetilde{A}$, which can be easily constructed by $\widetilde{Y}_{:,0}$, to construct constraints. The path matrix C can be calculated by $C = \textit{sigmoid}(\sum_{i=0}^{N-1}\widetilde{A}_i)$, where $\widetilde{A}_0 = I$, $\widetilde{A}_1 = \widetilde{A}$, $\widetilde{A}_{i+1} = \textit{sigmoid}(\widetilde{A}_i\widetilde{A})$ and $\textit{sigmoid}(x) = \frac{1}{1+e^{-a(x-0.5)}}$. Following (Ma, Chen, and Xiao 2018), we set a to 100 to make the output of $sigmoid$ close to either 0 or 1. Thus, the differentiable constraint is formulated as:

$$c_1 := -\sum_{i,j\in\widetilde{E}} p(i)p(j)\overline{C}_{i,j} + (1-p(i)p(j))C_{i,j}, \tag{12}$$

where $p(i) = 1-\widetilde{X}_{i,0}$ is the probability that node i exists and $\overline{C}_{i,j} = 1-C_{i,j}$.

A molecule is valid iff its configuration of the bonds meets the valence criteria of the atoms. We model this domain knowledge into a constraint to ensure molecular validity. Specifically, the differentiable constraint is formulated as:

$$c_2 := -\sum_{i\in\widetilde{V}} \max(\sum_{e\in\aleph(i)}\sum_j B(j)\widetilde{Y}_{e,j} - \sum_j U(j)\widetilde{X}_{i,j}, 0), \tag{13}$$

where $\aleph(i)$ is a set of edges pointing to node i, $B(j)$ is the capacity function of an edge type j, $U(j)$ is the valence of node type j, and both $B(0)$ and $U(0)$ are set to 0. $\max$ is to filter out the desirable case without penalty.

[1]https://csjtx1021.github.io/files/doc/CAGG-supp.pdf

Acquisition Function γ

The third component is an acquisition function γ. In this work, we use Expected Improvement (EI), which is one of the most commonly used acquisition functions in Bayesian optimization (Shahriari et al. 2016), as a quantitative metric to choose a candidate graph. It quantifies the expectation of the improvement over the current optimal solution (z_+). Moreover, it can balance exploration and exploitation in searching without introducing extra parameters. In our framework, the exploration is to find a graph that makes $\hat{f}_{\boldsymbol{\theta}}$ closer to f and the exploitation is to find a graph with the better property. Specifically, γ is formulated as:

$$\gamma(G) = \int_{z_+}^{+\infty}(z-z_+)p(z \mid \mathcal{D}, G)dz. \tag{14}$$

As the predictive distribution $p(z \mid \mathcal{D}, G)$ is Gaussian, the integral can be solved analytically.

We choose a G' to evaluate from the search space $\mathcal{G}$ by

$$G' = \arg\max_{G\in\mathcal{G}} \gamma(G). \tag{15}$$

Experiments

In this section, we rigorously evaluate the CAGG[2] by applying it to two challenging applications and answering two questions: 1) Can the CAGG produce optimal graphs with few evaluations? 2) Can it be applied to various domains?

Tasks

Molecular discovery is very challenging, especially for new drugs, mainly because the candidate space is quite large (Polishchuk, Madzhidov, and Varnek 2013) and the required resources are intensive, including a development cycle of 10∼20 years and costs of US$0.5∼2.6 billion (Paul et al. 2010; Avorn 2015). We apply the CAGG to accelerate the discovery process and reduce costs. Specifically, we focus on discovering the molecules with optimal properties. The following two common chemical properties (Jin, Barzilay, and Jaakkola 2018; Cui, Yang, and Hu 2019) are used as optimization goals: 1) $z = \textit{logP} - \textit{SA}$, where *logP* is the octanol-water partition coefficient and *SA* is the synthetic accessibility score; 2) $z = 5 \times \textit{QED} - \textit{SA}$, where *QED* is the quantitative estimation of drug-likeness. The molecules with high z are what we expect.

NAS is a key subfield of automatic machine learning (Elsken, Metzen, and Hutter 2019). It aims to automatically design a deep neural architecture with excellent performance for specific tasks. Herein, we perform the following two tasks according to the different types of architectures: 1) cell-based NAS is to design the optimal cell for image classification tasks; 2) multi-branch NAS is to design the optimal whole architecture for regression tasks.

Datasets

QM9 (Ramakrishnan et al. 2014) is a benchmark dataset in quantum chemistry, which contains about 134K molecules with at most 9 heavy atoms.

[2]Code available at: https://github.com/csjtx1021/CAGG

Goals	Methods	# Eval	Algorithm cost (hours)	Evaluation cost (hours)	Total cost			CSP
					Hours	CO_2e (lbs)	Google Cloud Platform	
logP−SA	Gentrl	3,000	4.3	3,000	3,004.3	412.1	US$1,254.6∼US$1616.3	95.70%
	GCPN	3,000	**0.2**	3,000	3,000.2	411.5	US$1,252.9∼US$1614.1	95.69%
	JTVAEBO	3,000	22.5	3,000	3,022.5	414.6	US$1,262.2∼US$1626.1	**95.73%**
	G2G	1,600	2.8	1,600	1602.8	219.8	US$669.3∼US$862.3	91.94%
	DGBO	189	0.3	189	189.3	26.0	US$79.1∼US$101.8	31.75%
	CAGG (ours)	**128**	1.2	**128**	**129.2**	**17.7**	**US$54.0∼US$69.6**	N/A
5×QED−SA	Gentrl	3,000	4.3	3,000	3,004.3	412.1	US$1,254.6∼US$1616.3	**95.16%**
	GCPN	3,000	**0.2**	3,000	3,000.2	411.5	US$1,252.9∼US$1614.1	**95.16%**
	JTVAEBO	1,550	21.5	1,550	1,571.5	215.6	US$656.3∼US$845.5	90.75%
	G2G	1,600	2.8	1,600	1602.8	219.8	US$669.3∼US$862.3	90.93%
	DGBO	448	1.0	448	449.0	61.6	US$187.5∼US$241.6	67.64%
	CAGG (ours)	**144**	1.3	**144**	**145.3**	**19.9**	**US$60.7∼US$78.2**	N/A

Table 1: Comparison of cost with molecular discovery methods. # Eval means the number of evaluations to find the optimal solution (lower is better). We set the maximum # Eval to 3,000. Algorithm cost represents the algorithm execution time, where the Gentrl, JTVAEBO, and CAGG contain the running time in pre-training and designing, the G2G only includes the training time, and both the DGBO and GCPN include only running time in searching or designing, because they do not require pre-training. Evaluation cost represents the cost of evaluating molecules, which is calculated based on # Eval and an hour per molecular evaluation. According to the report in (Gilmer et al. 2017), it is reasonable to estimate the time for calculating the molecular properties to be one hour. CO_2e is the estimated CO_2 emission, which is calculated based on the carbon emission estimation model (Strubell, Ganesh, and McCallum 2019). Google Cloud Platform cost is calculated based on the price of on-demand c2-standard-8 instances. CSP means the Cost Saving Percentage of the CAGG over other baselines.

NASBench201 (Dong and Yang 2020) is a unified benchmark for most up-to-date cell-based NAS methods. Due to the different search space designs, hyper-parameter adjustments and data augmentation tricks, it is still an open question to evaluate different NAS methods (Yang, Esperança, and Carlucci 2020). Therefore, we evaluate the methods fairly under the same conditions using the NASBench201 benchmark, which provides the full training and testing records on several common image classification datasets, including CIFAR100 and ImageNet16-120.

Indoor (Torres-Sospedra et al. 2014) collects about 20K localization records of mobile devices. Slice (Graf et al. 2011) contains 53,500 CT images from 74 patients. They are used to evaluate regression performance for neural nets.

Baselines and Setup

We compare our method against the following state-of-the-art methods for molecule discovery: the RL-based approaches, i.e., GCPN (You et al. 2018) and Gentrl (Zhavoronkov et al. 2019); an end-to-end method, i.e., G2G (Jin et al. 2019); a VAE method, i.e., JTVAEBO (Jin, Barzilay, and Jaakkola 2018); and a search algorithm, i.e., DGBO (Cui, Yang, and Hu 2019).

Since we use the NASBench201 benchmark to evaluate the cell-based NAS methods, we compare against the methods that perform top-3 in terms of the accuracy of searched architectures under a given budget, as reported in this benchmark (Dong and Yang 2020): a random search (RS) algorithm (Bergstra and Bengio 2012); an evolution algorithm, i.e., REA (Real et al. 2019); and a commonly used baseline RL method (REINFORCE) (Williams 1992). We also use a classical hand-crafted deep residual architecture (ResNet) (He et al. 2016) as a baseline.

For multi-branch NAS, the representative baselines on regression tasks include: RAND (Kandasamy et al. 2018); the Gaussian process-based BO methods, e.g., TreeBO (Jenatton et al. 2017), NASBOT (Kandasamy et al. 2018), and Auto-Keras (Jin, Song, and Hu 2019); and a method combining evolutionary algorithm with BO, i.e., NASGBO (Ma, Cui, and Yang 2019).

Due to space limitations, please see Section S2 in the supplementary material for the detailed description of the baselines and experimental setup.

Results on Molecular Discovery

From Fig. 2, we see that the CAGG outperforms others and converges quickly to the optimal solution. The JTVAEBO only finds the optimum for the property of $5 \times QED - SA$, whereas both the Gentrl and GCPN cannot converge at a given budget of 3K evaluations. Besides the CAGG, the DGBO shows the fastest convergence, mainly because its predetermined search space contains the optimum. Otherwise, it would perform poorly due to no optimum. Since the G2G is an end-to-end method, which needs to input all the supervision pairs at one time without the process of augmenting data, we thus do not include it here. We also observed that the CAGG can produce better molecules in optimizing than other baselines (see Fig. S3 in the supplementary material for details).

We report the cost of finding the optimal solution for each method in Table 1. For the property of $logP - SA$, the total cost of the Gentrl, GCPN, and JTVAEBO is more than 3,000 hours. This computing consumption will cost US$1,252.9∼US$1,626.1 on the cloud platform and emit 411.5∼414.6 pounds of CO_2e, which is roughly equivalent to a person's two-week carbon emissions (Strubell, Ganesh, and McCallum 2019). However, the CAGG can reduce the costs of these three methods by more than 95% to 129.2

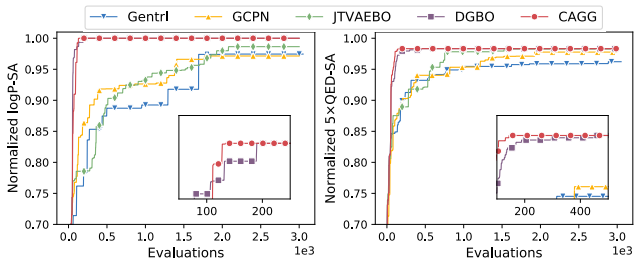

Figure 2: Convergence comparison of methods to optimize a molecule. All methods ran at least three times to eliminate random effects. Solid lines represent mean values. We normalize the z for readability by $\frac{z-z_{min}}{z_{max}-z_{min}}$, where z_{max} and z_{min} are the max and min in the QM9 dataset, respectively.

hours, US$54.0∼US$69.6, and 17.7 pounds of CO_2e, which is only equivalent to a person's carbon emissions in less than one day. We use 16K bad-good pairs constructed by 1,600 evaluations to train the G2G for each property. We see that the cost of the G2G can be reduced by 91.94%. Although the DGBO also converges at a small cost, our CAGG can still further reduce its cost by 31.75%. For the property of $5 \times \mathit{QED} - \mathit{SA}$, the CAGG can reduce the total cost of the Gentrl, GCPN, JTVAEBO, G2G, and DGBO by 95.16%, 95.16%, 90.75%, 90.93%, and 67.64%, respectively. These significant reductions on cost demonstrate its effectiveness.

To find out the specific role of the internal components in the CAGG, we also verify the effectiveness of the proposed surrogate model and generation model in Section S3 in the supplementary material. The observations are: 1) our surrogate model adequately characterizes uncertainty, resulting in better prediction ability than others in the case of scarce data; 2) the generation model produces the desired graphs, which is contributed by the two-phase training strategy.

Results on Neural Architecture Search

Results on cell-based NAS. From Table 2, we clearly see that all methods find a cell that is superior to the hand-crafted one (ResNet). The CAGG can find the comparable cells to baselines, reducing the total cost of 205 hours by 75.46% to 50.3 hours. Moreover, the cells found by the CAGG at the total cost of 140.9 hours (i.e., 31.27% reduction) outperform that found by other methods under 205 hours on both datasets. This demonstrates that the CAGG can find a better cell with less cost. When the total cost reaches similar to other methods, our CAGG finds the optimal cells, which are shown in Fig. S6 in the supplementary material. In the design process, the CAGG still shows the fastest convergence on both datasets and can produce the better cells than others (see Figs. S4 and S5 in the supplementary material).

Methods	Total cost	CIFAR100	ImageNet16-120
ResNet	N/A	70.86	43.63
RS	205 hours	72.48±1.04	46.04±0.46
REINFORCE	205 hours	72.48±0.31	45.85±0.51
REA	205 hours	73.09±0.25	46.12±0.67
CAGG (ours)	50.3 hours	72.87±0.27	46.13±0.46
	140.9 hours	73.25±0.42	46.34±0.27
	201.5 hours	**73.38±0.16**	**46.37±0.24**

Table 2: Comparison of cost and classification accuracy with baselines for cell-based NAS. The total cost includes the algorithm execution time and evaluation costs. The evaluation cost per architecture is assigned to half an hour, which is estimated based on the running time on a personal computer. The last two columns show the test classification accuracy (%). All methods ran five times to eliminate random effects. We set the budget to 205 hours for all baselines and report the results found by the CAGG under various total costs.

Results on multi-branch NAS. From Table 3, we see that under the same 12-hour budget, the CAGG can find better architectures than others for both datasets (the found optimal whole architectures are shown in Fig. S6 in the supplementary material). For the Indoor dataset, the architecture found by the CAGG under a 4-hour budget outperforms that found by others; that is, our CAGG reduces the cost by 66.67% to find the optimum. For the Slice dataset, the architectures found by the CAGG under a 4/8-hour budget are comparable to others. Therefore, the CAGG can find optimal or comparable architectures under less cost in this task.

Methods	Total cost	Indoor	Slice
RAND	12 hours	0.156±0.023	0.932±0.044
TreeBO	12 hours	0.168±0.023	0.759±0.079
NASBOT	12 hours	0.114±0.009	0.615±0.044
Auto-Keras	12 hours	0.112±0.010	0.870±0.054
NASGBO	12 hours	0.090±0.012	0.560±0.046
CAGG (ours)	4 hours	0.072±0.003	0.788±0.003
	8 hours	0.066±0.002	0.625±0.001
	12 hours	**0.063±0.001**	**0.433±0.010**

Table 3: Comparison of cost and the test regression mean squared error (lower is better) with baselines for multi-branch NAS. We set the budget to 12 hours and report the results found by the CAGG under 4, 8, and 12 hours.

Conclusion

In this work, we propose a novel cost-aware framework to design optimal graphs at as low cost as possible. We apply it to two challenging real-world problems, i.e., molecular discovery and NAS, to rigorously evaluate its effectiveness and applicability. The results show that the CAGG can quickly converge to the optimal solution and significantly reduce the cost compared with the state-of-the-art.

Several challenges and further studies to make the CAGG more powerful are worth mentioning. First, how to relax the limit of the maximum number of nodes fixed in advance when generating graphs will be considered in future work. Second, we assume that graphs have the same evaluation cost, so we mainly focus on reducing the number of evaluations. To be more in line with the actual situation that different graphs usually have different evaluation costs, how to introduce these differences between costs is also an interesting research direction. Handling other complex generative tasks and multi-objective situation are promising extensions.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant No. 61876069; Jilin Province Key Scientific and Technological Research and Development Project under Grant Nos. 20180201067GX and 20180201044GX; Jilin Province Natural Science Foundation under Grant No. 20200201036JC; and the Hong Kong Research Grants Council (RGC) under project RGC/HKBU12201318.

References

Avorn, J. 2015. The $2.6 Billion Pill — Methodologic and Policy Considerations. *New England Journal of Medicine* 372(20): 1877–1879.

Baker, B.; Gupta, O.; Naik, N.; and Raskar, R. 2017. Designing Neural Network Architectures using Reinforcement Learning. In *ICLR*.

Barabási, A.; and Albert, R. 1999. Emergence of Scaling in Random Networks. *Science* 286(5439): 509–512.

Bergstra, J.; and Bengio, Y. 2012. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research* 13: 281–305.

Bojchevski, A.; Shchur, O.; Zügner, D.; and Günnemann, S. 2018. NetGAN: Generating Graphs via Random Walks. In *ICML*.

Cai, H.; Gan, C.; Wang, T.; Zhang, Z.; and Han, S. 2020. Once for All: Train One Network and Specialize it for Efficient Deployment. In *ICLR*.

Cao, N. D.; and Kipf, T. 2018. MolGAN: An implicit generative model for small molecular graphs. In *ICML Workshops Track*.

CSS. 2018. Carbon Footprint Factsheet. Technical report, Center for Sustainable Systems, University of Michigan.

Cui, J.; Yang, B.; and Hu, X. 2019. Deep Bayesian optimization on attributed graphs. In *AAAI*.

Dai, H.; Tian, Y.; Bo, D.; Skiena, S.; and Le, S. 2018. Syntax-Directed Variational Autoencoder for Structured Data. In *ICLR*.

Dong, X.; and Yang, Y. 2020. NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. In *ICLR*.

Elsken, T.; Metzen, J. H.; and Hutter, F. 2019. Neural Architecture Search: A Survey. *Journal of Machine Learning Research* 20: 55:1–55:21.

Erdős, P.; and Rényi, A. 1959. On Random Graphs I. *Publicationes Mathematicae* 4: 3286–3291.

Gal, Y.; and Ghahramani, Z. 2016. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *ICML*.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural Message Passing for Quantum Chemistry. In *ICML*.

Gómez-Bombarelli, R.; Wei, J. N.; Duvenaud, D.; Hernández-Lobato, J. M.; Sánchez-Lengeling, B.; Sheberla, D.; Aguilera-Iparraguirre, J.; Hirzel, T. D.; Adams, R. P.; and Aspuru-Guzik, A. 2018. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science* 4(2): 268–276.

Graf, F.; Kriegel, H.; Schubert, M.; Pölsterl, S.; and Cavallaro, A. 2011. 2D Image Registration in CT Images Using Radial Image Descriptors. In *MICCAI*.

Guimaraes, G. L.; Sanchez-Lengeling, B.; Farias, P. L. C.; and Aspuru-Guzik, A. 2017. Objective-Reinforced Generative Adversarial Networks (ORGAN) for Sequence Generation Models. *arXiv:1705.10843* .

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *CVPR*, 770–778.

Jenatton, R.; Archambeau, C.; Gonzalez, J.; and Seeger, M. 2017. Bayesian Optimization with Tree-Structured Dependencies. In *ICML*.

Jin, H.; Song, Q.; and Hu, X. 2019. Auto-Keras: An Efficient Neural Architecture Search System. In *KDD*.

Jin, W.; Barzilay, R.; and Jaakkola, T. 2018. Junction Tree Variational Autoencoder for Molecular Graph Generation. In *ICML*.

Jin, W.; Barzilay, R.; and Jaakkola, T. 2020a. Hierarchical Generation of Molecular Graphs using Structural Motifs. In *ICML*.

Jin, W.; Barzilay, R.; and Jaakkola, T. 2020b. Multi-Objective Molecule Generation using Interpretable Substructures. In *ICML*.

Jin, W.; Yang, K.; Barzilay, R.; and Jaakkola, T. 2019. Learning Multimodal Graph-to-Graph Translation for Molecular Optimization. In *ICLR*.

Kandasamy, K.; Neiswanger, W.; Schneider, J.; Poczos, B.; and Xing, E. P. 2018. Neural Architecture Search with Bayesian Optimisation and Optimal Transport. In *NeurIPS*.

Kokiopoulou, E.; Hauth, A.; Sbaiz, L.; Gesmundo, A.; Bartók, G.; and Berent, J. 2020. Task-aware Performance Prediction for Efficient Architecture Search. In *ECAI*.

Kusner, M. J.; Paige, B.; and Hernández-Lobato, J. M. 2017. Grammar Variational Autoencoder. In *ICML*.

Li, Y.; Dong, M.; Wang, Y.; and Xu, C. 2020. Neural Architecture Search in a Proxy Validation Loss Landscape. In *ICML*.

Li, Y.; Zhang, L.; and Liu, Z. 2018. Multi-objective de novo drug design with conditional graph generative model. *Journal of Cheminformatics* .

Liu, H.; Simonyan, K.; and Yang, Y. 2019. DARTS: Differentiable Architecture Search. In *ICLR*.

Lu, X.; Huang, H.; Dong, W.; Li, X.; and Shi, G. 2020. Beyond Network Pruning: a Joint Search-and-Training Approach. In *IJCAI*, 2583–2590.

Luo, R.; Tian, F.; Qin, T.; Chen, E.-H.; and Liu, T.-Y. 2018. Neural Architecture Optimization. In *NeurIPS*.

Ma, L.; Cui, J.; and Yang, B. 2019. Deep Neural Architecture Search with Deep Graph Bayesian Optimization. In *WI*.

Ma, T.; Chen, J.; and Xiao, C. 2018. Constrained Generation of Semantically Valid Graphs via Regularizing Variational Autoencoders. In *NeurIPS*.

Paul, S. M.; Mytelka, D. S.; Dunwiddie, C. T.; Persinger, C. C.; Munos, B. H.; Lindborg, S. R.; and Schacht, A. L. 2010. How to improve R&D productivity: the pharmaceutical industry's grand challenge. *Nature Reviews Drug Discovery* 9(3): 203–214.

Polishchuk, P. G.; Madzhidov, T. I.; and Varnek, A. 2013. Estimation of the size of drug-like chemical space based on GDB-17 data. *Journal of Computer-Aided Molecular Design* 27(8): 675—679.

Qi, L.; Allamanis, M.; Brockschmidt, M.; and Gaunt, A. L. 2018. Constrained Graph Variational Autoencoders for Molecule Design. In *NeurIPS*.

Ramachandram, D.; Lisicki, M.; Shields, T. J.; Amer, M. R.; and Taylor, G. W. 2018. Bayesian optimization on graph-structured search spaces: Optimizing deep multimodal fusion architectures. *Neurocomputing* 298: 80–89.

Ramakrishnan, R.; Dral, P. O.; Rupp, M.; and von Lilienfeld, O. A. 2014. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data* 1: 140022.

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2019. Regularized Evolution for Image Classifier Architecture Search. In *AAAI*, 4780–4789.

Samanta, B.; De, A.; Ganguly, N.; and Gomez-Rodriguez, M. 2019. NeVAE: A Deep Generative Model for Molecular Graphs. In *AAAI*.

Sanchez-Lengeling, B.; and Aspuru-Guzik, A. 2018. Inverse molecular design using machine learning: Generative models for matter engineering. *Science* 361(6400): 360–365.

Shahriari, B.; Swersky, K.; Wang, Z.; Adams, R. P.; and De Freitas, N. 2016. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE* 104(1): 148–175.

Simonyan, K.; and Zisserman, A. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *ICLR*.

Strubell, E.; Ganesh, A.; and McCallum, A. 2019. Energy and Policy Considerations for Deep Learning in NLP. In *ACL*, 3645–3650.

Torres-Sospedra, J.; Montoliu, R.; Martínez-Usó, A.; Avariento, J. P.; Arnau, T. J.; Benedito-Bordonau, M.; and Huerta, J. 2014. UJIIndoorLoc: A new multi-building and multi-floor database for WLAN fingerprint-based indoor localization problems. In *IPIN*.

Williams, R. J. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning* 8: 229–256.

Yang, A.; Esperança, P. M.; and Carlucci, F. M. 2020. NAS evaluation is frustratingly hard. In *ICLR*.

You, J.; Liu, B.; Ying, R.; Pande, V.; and Leskovec, J. 2018. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. In *NeurIPS*.

Zhang, G.; and Musgrave, C. B. 2007. Comparison of DFT Methods for Molecular Orbital Eigenvalue Calculations. *The Journal of Physical Chemistry A* 111(8): 1554–1561.

Zhang, M.; Jiang, S.; Cui, Z.; Roman, G.; and Chen, Y. 2019. D-VAE: A Variational Autoencoder for Directed Acyclic Graphs. In *NeurIPS*.

Zhavoronkov, A.; Ivanenkov, Y. A.; Aliper, A.; Veselov, M. S.; Aladinskiy, V. A.; Aladinskaya, A. V.; Terentiev, V. A.; Polykovskiy, D. A.; Kuznetsov, M. D.; Asadulaev, A.; Volkov, Y.; Zholus, A.; Shayakhmetov, R. R.; Zhebrak, A.; Minaeva, L. I.; Zagribelnyy, B. A.; Lee, L. H.; Soll, R.; Madge, D.; Xing, L.; Guo, T.; and Aspuru-Guzik, A. 2019. Deep learning enables rapid identification of potent DDR1 kinase inhibitors. *Nature Biotechnology* 37(9): 1038–1040.

Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2018. Learning Transferable Architectures for Scalable Image Recognition. In *CVPR*, 8697–8710.