Graph Transformation Policy Network for Chemical Reaction Prediction

Kien Do
dkdo@deakin.edu.au
Deakin University
Geelong, Victoria, Australia

Truyen Tran
truyen.tran@deakin.edu.au
Deakin University
Geelong, Victoria, Australia

Svetha Venkatesh
svetha.venkatesh@deakin.edu.au
Deakin University
Geelong, Victoria, Australia

ABSTRACT

We address a fundamental problem in chemistry known as chemical reaction product prediction. Our main insight is that the input reactant and reagent molecules can be jointly represented as graphs, and the process of generating product molecules from reactant molecules can be formulated as a set of graph transformations. To this end, we propose Graph Transformation Policy Network (GTPN) – a novel generic method that combines the strengths of graph neural networks and reinforcement learning to learn reactions directly from data with minimal chemical knowledge. Compared to previous methods, GTPN has some appealing properties such as: end-to-end learning, and making no assumption about the length or the order of graph transformations. In order to guide our model search through the complex discrete space of sets of graph transformations effectively, we extend the standard policy gradient loss by adding useful constraints. Evaluation results show that GTPN improves the top-1 accuracy over the current state-of-the-art method by about 3% on the large *USPTO* dataset.

KEYWORDS

Chemical Reaction, Graph Transformation, Reinforcement Learning

ACM Reference Format:
Kien Do, Truyen Tran, and Svetha Venkatesh. 2019. Graph Transformation Policy Network for Chemical Reaction Prediction. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3292500.3330958

1 INTRODUCTION

Chemical reaction product prediction is a fundamental problem in organic chemistry. It paves the way for planning syntheses of new substances [3]. For decades, huge effort has been spent to solve this problem. However, most methods still depend on handcrafted reaction rules [3, 17, 33] or heuristically extracted reaction templates [5, 27], thus are not well generalizable to unseen reactions.

A reaction can be regarded as a set (or an unordered sequence) of graph transformations in which reactants represented as molecular graphs are transformed into products by modifying the bonds between some atom pairs [16] (see Fig. 1 for an illustration). We call an atom pair (u, v) that changes its connectivity during a reaction and its new bond b a *reaction triple* (u, v, b). The reaction product prediction problem now becomes predicting a set of reaction triples given input reactants and reagents. We argue that in order to solve this problem effectively, an intelligent system should have two key capabilities: (a) *Understanding the molecular graph structure* of the input reactants and reagents so that it can identify possible reactivity patterns (i.e., atom pairs with changing connectivity). (b) *Knowing how to choose from these reactivity patterns* a correct set of reaction triples to generate the desired products.

Recent state-of-the-art methods [2, 15] have built the first capability by leveraging graph neural networks [8, 11, 12, 23]. However, these methods are either unaware of the valid sets of reaction triples [15] or limited to sequences of reaction triples with a predefined orders [2]. The main challenge is that the space of all possible configurations of reaction triples is extremely large and non-differentiable. Moreover, a small change in the predicted set of reaction triples can lead to very different reaction products and a little mistake can lead to an invalid prediction.

In this paper, we propose a novel method called Graph Transformation Policy Network (GTPN) that addresses the aforementioned challenges. Our model consists of three main components: a graph neural network (GNN), a node pair prediction network (NPPN) and a policy network (PN). Starting from the initial graph of reactant and reagent molecules, our model iteratively alternates between modeling an input graph using GNN and predicting a reaction triple using NPPN and PN to generate a new intermediate graph as input for the next step until it decides to stop. The final generated graph is considered as the predicted products of the reaction. Importantly, GTPN does not assume any fixed number or any order of bond changes but learn these properties itself. One can view GTPN as a reinforcement learning (RL) agent that operates on a complex and non-differentiable space of sets of reaction triples. To guide our model towards learning a diverse yet robust-to-small-changes policy, we customize the standard policy gradient loss [21] with additional constraints.

To the best of our knowledge, GTPN is the most generic approach for the reaction product prediction problem so far in the sense that: i) It combines graph neural networks and reinforcement learning into a unified framework and trains everything end-to-end; ii) It does not use any handcrafted or heuristically extracted reaction rules/templates to predict products. Instead, it automatically learns various types of reactions from the training data and can generalize to unseen reactions; iii) It can interpret how the products are formed via the sequence of reaction triples it generates.

KDD '19, August 4–8, 2019, Anchorage, AK, USA
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330958

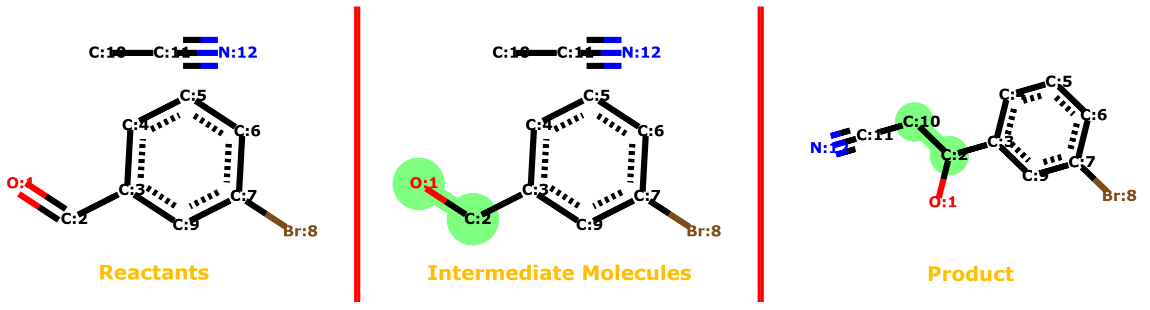


Figure 1: A sample reaction represented as a set of graph transformations from reactants (leftmost) to products (rightmost). Atoms are labeled with their type (Carbon, Oxygen,...) and their index (1, 2,...) in the molecular graph. The atom pairs that change connectivity and their new bonds (if existed) are highlighted in green. There are two bond changes in this case: 1) The double bond between O:1 and C:2 becomes single. 2) A new single bond between C:2 and C:10 is added.

We evaluate GTPN on two large public datasets named *USPTO-15k* and *USPTO*. Our method significantly outperforms all baselines in the top-1 accuracy, achieving new state-of-the-art results of 82.39% and 83.20% on *USPTO-15k* and *USPTO*, respectively. In addition, we also provide comprehensive analyses about the performance of GTPN and about different types of errors our model could make.

2 METHOD

2.1 Chemical Reaction as Markov Decision Process of Graph Transformations

A reaction occurs when reactant molecules interact with each other in the presence (or absence) of reagent molecules to form new product molecules by breaking or adding some of their bonds. Our main insight is that reaction product prediction can be formulated as predicting a set of such bond changes given the reactant and reagent molecules as input. A bond change is characterized by the atom pair that holds the bond (*where* the change happens) and the new bond type (*what* is the change). We call the atom pair a *reaction atom pair* and call the atom pair with the new bond type a *reaction triple*.

More formally, we represent the entire system of input reactant and reagent molecules as a labeled graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with multiple connected components, each of which corresponds to a molecule. Nodes in $\mathcal{V}$ are atoms labeled with their atomic numbers and edges in $\mathcal{E}$ are bonds labeled with their bond types. Given $\mathcal{G}$ as input, we predict a set of reaction triples that transforms $\mathcal{G}$ into a graph of product molecules $\mathcal{G}'$. We model this set as a sequence $(\xi, u, v, b)^0$, $(\xi, u, v, b)^1$,..., $(\xi, u, v, b)^{T-1}$ (or $(\xi, u, v, b)^{0:T}$ for short) where T is the maximum number of steps, (u, v) is a pair of nodes, b is the new edge type of (u, v), and ξ is a binary signal that indicates the end of the sequence. If the sequence ends at $T_{\text{end}} < T$, ξ^0,..., $\xi^{T_{\text{end}}-1}$ will be 1 and $\xi^{T_{\text{end}}}$,..., ξ^{T-1} will be 0. At every step t, if $\xi^t = 1$, we apply the predicted edge change $(u, v, b)^t$ on the current graph $\mathcal{G}^t$ to create a new intermediate graph $\mathcal{G}^{t+1}$ as input for the next step $t + 1$. This iterative process of graph transformation can be formulated as a Markov Decision Process (MDP) characterized by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, in which $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, P is a state transition function, R is a reward function, and γ is a discount factor. Since the process is finite and contains no loop, we set the discount factor γ to be 1. The rest of the MDP tuple is defined as follows:

- **State**: A state $s^t \in \mathcal{S}$ is an intermediate graph $\mathcal{G}^t$ generated at step t $(0 \leq t < T)$. When $t = 0$, we denote $s^0 = \mathcal{G}^0 = \mathcal{G}$.
- **Action**: An action $a^t \in \mathcal{A}$ performed at step t is the tuple $(\xi, u, v, b)^t$. The action is composed of three consecutive sub-actions: ξ^t, $(u, v)^t$, and b^t. If $\xi^t = 0$, our model will ignore the next sub-actions $(u, v)^t$ and b^t, and all the future actions $(\xi, u, v, b)^{t+1:T}$. Note that setting ξ^t to be the first sub-action is useful in case a reaction does not happen, i.e., $\xi^0 = 0$
- **State Transition**: If $\xi^t = 1$, the current graph $\mathcal{G}^t$ is modified based on the reaction triple $(u, v, b)^t$ to generate a new intermediate graph $\mathcal{G}^{t+1}$. We do not incorporate chemical rules such as valency check during state transition because the current bond change may result in invalid intermediate molecules $\mathcal{G}^t$, but later, other bond changes may compensate it to create the valid final products $\mathcal{G}^{T_{\text{end}}}$.
- **Reward**: We use both immediate rewards and delayed rewards to encourage our model to learn the optimal policy faster. At every step t, if the model predicts ξ^t, $(u, v)^t$ or b^t correctly, it will receive a positive reward for each correct sub-action. Otherwise, a negative reward is given. After the prediction process has terminated, if the generated products are exactly the same as the ground-truth products, we give the model a positive reward, otherwise a negative reward. The concrete reward values are provided in Appendix 5.1.

2.2 Graph Transformation Policy Network

In this section, we describe the architecture of our model – a Graph Transformation Policy Network (GTPN). GTPN has three main components namely a Graph Neural Network (GNN), a Node Pair Prediciton Network (NPPN), and a Policy Network (PN). Each component is responsible for one or several key functions shown in Fig. 2: GNN performs functions 1 and 6; NPPN performs function 2; and PN performs functions 3, 4 and 5. Apart from these components, GTPN also has a Recurrent Neural Network (RNN) to keep track of the past transformations. The hidden state of this RNN is used by NPPN and PN to make accurate prediction.

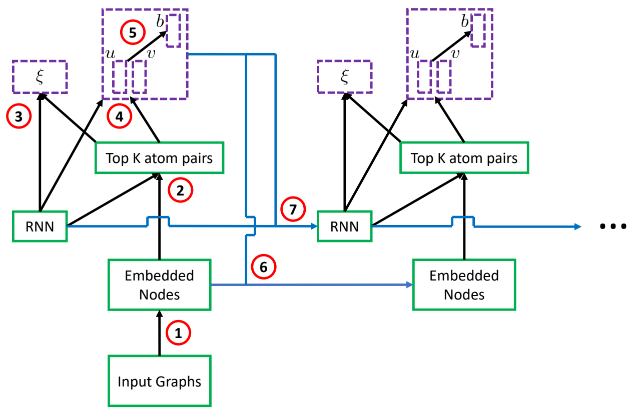


Figure 2: Workflow of a Graph Transformation Policy Network (GTPN). At every step of the forward pass, our model performs 7 major functions: 1) Computing the atom representation vectors, 2) Computing the most possible K reaction atom pairs, 3) Predicting the continuation signal ξ, 4) Predicting the reaction atom pair (u, v), 5) Predicting a new bond b of this atom pair, 6) Updating the atom representation vectors, and 7) Updating the recurrent state.

2.2.1 *Graph Neural Network.* Input to our Graph Neural Network (GNN) is the initial graph $\mathcal{G}^0$ which only contains reactant and reagent molecules. Each atom $i \in \mathcal{V}$ of $\mathcal{G}^0$ is represented by a node feature vector $\boldsymbol{v}_i$ and each bond $(i, j) \in \mathcal{E}$ is represented by an edge feature vector $\boldsymbol{e}_{ij}$. $\boldsymbol{v}_i$ encapsulates important characteristics of the atom i including its type (e.g. carbon, oxygen,...), charge, degree, and valence. Similarly, $\boldsymbol{e}_{ij}$ captures the bond type (e.g. single, double,...) between the two atoms i and j. We denote by $\mathcal{N}(i)$ the set of all neighbor nodes of node i together with their links to node i:

$$\mathcal{N}(i) \equiv \left\{(j, e_{ij}) \mid j \text{ is a neighbor node of } i\right\}$$

If we only consider the neighbor nodes of i not their links, we use the notation $\mathcal{N}_\text{n}(i)$ defined as:

$$\mathcal{N}_\text{n}(i) \equiv \{j \mid j \text{ is a neighbor node of } i\}$$

In addition to $\boldsymbol{v}_i$, each node i also has a state vector $\boldsymbol{x}_i$ to store information about itself and its surrounding chemical structure. The state vector is updated recursively by using a neural message passing method [1, 11, 12, 23, 25] described below.

Computing neighbor messages. At the message passing step $\ell = 0, ..., L-1$, we compute the message $\boldsymbol{m}_{ij}^\ell$ from every neighbor node $j \in \mathcal{N}_n(i)$ to node i as:

$$\begin{aligned}\boldsymbol{m}_{ij}^\ell &= f\left(\boldsymbol{x}_i^\ell, \boldsymbol{x}_j^\ell, \boldsymbol{e}_{ij}\right)\\ &= \sigma\left(W\left[\boldsymbol{x}_i^\ell, \boldsymbol{x}_j^\ell, \boldsymbol{e}_{ij}\right] + b\right) \qquad (1)\end{aligned}$$

where [.] denotes concatenation; and σ is a nonlinear function.

Aggregating neighbor messages. Then, we aggregate all the messages sent to node i into a single message vector by averaging:

$$\boldsymbol{m}_i^\ell = \frac{1}{|\mathcal{N}_\text{n}(i)|}\sum_{j\in\mathcal{N}_\text{n}(i)} \boldsymbol{m}_{ij}^\ell \qquad (2)$$

where $|\mathcal{N}_\text{n}(i)|$ is the number of neighbor nodes of node i.

Updating node state. Finally, we update the state of node i as follows:

$$\begin{aligned}\boldsymbol{x}_i^{\ell+1} &= \text{Highway}\left(\boldsymbol{x}_i^\ell, \boldsymbol{m}_i^\ell, \boldsymbol{v}_i\right) \qquad (3)\\ &= \boldsymbol{\alpha} * \tilde{\boldsymbol{x}}_i^{\ell+1} + (1-\boldsymbol{\alpha}) * \boldsymbol{x}_i^\ell \qquad (4)\end{aligned}$$

where Highway(.) is a Highway Network layer defined in [30]; $\tilde{\boldsymbol{x}}_i^{\ell+1}$ is the nonlinear part which is computed as:

$$\tilde{\boldsymbol{x}}_i^{\ell+1} = \sigma\left(W_1\left[\boldsymbol{x}_i^\ell, \boldsymbol{m}_i^\ell, \boldsymbol{v}_i^\ell\right] + b_1\right)$$

and $\boldsymbol{\alpha}$ is the gate controlling the flow of information:

$$\boldsymbol{\alpha} = \text{sigmoid}(W_2\left[\boldsymbol{x}_i^\ell, \boldsymbol{m}_i^\ell, \boldsymbol{v}_i^\ell\right] + b_2)$$

By combining Eqs. (1,2,3) together, we can write one step of message passing update for node i in a generic way as follows:

$$\boldsymbol{x}_i^{\ell+1} = \text{MessagePassing}\left(\boldsymbol{x}_i^\ell, \boldsymbol{v}_i, \mathcal{N}(i)\right) \qquad (5)$$

where the initial node state vector $\boldsymbol{x}_i^0$ is the nonlinear mapping of $\boldsymbol{v}_i$:

$$\boldsymbol{x}_i^0 = \sigma\left(W\boldsymbol{v}_i + b\right) \qquad (6)$$

After calling the MessagePassing(.) function L times, we obtain the set of all state vectors $\left\{\boldsymbol{x}^L\right\}$ for all nodes of $\mathcal{G}^0$. To avoid confusion with the message passing process, in the next sections, we will use the superscript t instead of ℓ to denote one step of the reaction triple prediction process, and call $\boldsymbol{x}^t$ a *"node representation vector"* instead of a "node state vector" (see the function 1 in Fig. 2). At $t_0 = 0$, the initial representation vector $\boldsymbol{x}_i^{t_0}$ for node i is, indeed, $\boldsymbol{x}_i^L$.

2.2.2 *Node Pair Prediction Network.* Assume that at step $t \geq 0$, we have predicted t reaction triples $(\xi, u, v, b)^0, (\xi, u, v, b)^1, ..., (\xi, u, v, b)^{t-1}$ and have achieved an intermediate molecular graph $\mathcal{G}^t$ represented by $\left\{\boldsymbol{x}^t\right\}$. To predict a new reaction triple $(\xi, u, v, b)^t$, first, we need to find all possible reaction atom pairs at step t. We characterize the reactivity of each atom pair (i, j) by a score $s_{ij}^t \in \mathbb{R}$ satisfied that if s_{ij}^t is high, (i, j) is more likely to have its bond changed. To compute s_{ij}^t, we use a Node Pair Prediction Network (NPPN) similar to the "global network" presented in [15]. Its formulas are provided below:

$$\begin{aligned}\boldsymbol{r}_{ij}^t &= \sigma\left(W_1\left[(\boldsymbol{x}_i^t + \boldsymbol{x}_j^t), \boldsymbol{e}_{ij}\right] + b_1\right) \qquad (7)\\ a_{ij}^t &= \text{softmax}\left(W_2\boldsymbol{r}_{ij}^t + b_2\right) \qquad (8)\\ \boldsymbol{c}_i^t &= \sum_{j\in\mathcal{V}} a_{ij}\boldsymbol{x}_j^t \qquad (9)\\ z_{ij}^t &= \sigma\left(W_3\left[\boldsymbol{h}^{t-1}, (\boldsymbol{x}_i^t + \boldsymbol{x}_j^t), (\boldsymbol{c}_i^t + \boldsymbol{c}_j^t), \boldsymbol{e}_{ij}\right] + b_3\right) \qquad (10)\\ s_{ij}^t &= f^\text{atom pair}\left(z_{ij}^t\right) \qquad (11)\end{aligned}$$

where $W_1, W_2, W_3, b_1, b_2, b_3$ are parameters; $a_{ij}^t \in \mathbb{R}$ is the attention score from node i to any other node j; $\boldsymbol{c}_i^t$ is the summary vector for node i; $\boldsymbol{h}^{t-1}$ is the hidden state of the RNN at the previous step; $f^{\text{atom pair}}$ is a neural network. Note that our NPPN leverages self-attention method [31, 32] (Eqs. 7-9) to model the global interaction between atoms.

Top-K atom pairs. Because the number of atom pairs that actually participate in a reaction is very small (usually smaller than 10) compared to the total number of atom pairs of the input molecules (usually hundreds or thousands), it is much more efficient to identify reaction triples from a small subset of highly probable reaction atom pairs. For that reason, we extract K ($K \ll |\mathcal{V}|^2$) atom pairs with the highest scores. Later, we will predict reaction triples taken from these K atom pairs only. We denote the set of top-K atom pairs as $\{(u_k, v_k)\}$, the set of their corresponding scores as $\left\{s_{u_k v_k}\right\}$ and the set of their representation vectors as $Z_K = \left\{z_{u_k v_k}\right\}$ with $k = 1, ..., K$.

Using reagent information. As can be seen from Table 2, reagent molecules account for about a half of the input molecules on average and 60-80% of all reactions containing reagents. It suggests that the proper use of reagent information will lead to better prediction. In our model, before computing the scores for all atom pairs, we append to the representation vector of every atom a binary scalar indicating whether this atom comes from a reagent molecule or not.

2.2.3 Policy Network.

Predicting continuation signal. To account for varying number of reaction triples for different reactions, at every step t, our Policy Network (PN) generates a continuation signal $\xi^t \in \{0, 1\}$ to indicate whether the prediction process should continue or terminate. ξ^t is drawn from a Bernoulli distribution:

$$p\left(\xi^t = 1\right) \quad = \quad \text{sigmoid}\left(f^{\text{signal}}\left(\left[\boldsymbol{h}^{t-1}, g\left(Z_K^t\right)\right]\right)\right) \qquad (12)$$

where $\boldsymbol{h}^{t-1}$ is the previous RNN state; Z_K^t is the set of representation vectors of the top K atom pairs at the current step; f^{signal} is a neural network; g is a function that maps an unordered set of inputs to an output vector. We simply use the mean function :

$$z_K^{t-1} = g\left(Z_K^t\right) = \frac{1}{K}\sum_{k=1}^{K} W z_{u_k v_k}^{t-1}$$

Predicting atom pair. At the next sub-step, the PN selects a reaction atom pair by sampling from the top-K atom pairs with probability:

$$p\left((u_k, v_k)^t\right) = \text{softmax}_K\left(s_{u_k v_k}^t\right) \qquad (13)$$

where $s_{u_k v_k}^t$ is the score of the atom pair $(u_k, v_k)^t$ computed in Eq. (11). After predicting the atom pair $(u, v)^t$, we will mask it to ensure that it could not be in the top K again at future steps.

Predicting bond type. Given an atom pair $(u, v)^t$ sampled from the previous sub-step, we predict a new bond type b^t between u and v to get a complete reaction triple $(u, v, b)^t$ using the probability:

$$p\left(b^t\right) = \text{softmax}_B\left(f^{\text{bond}}\left(\left[\boldsymbol{h}^{t-1}, \boldsymbol{z}_{uv}^t, \left(\boldsymbol{e}_b - \boldsymbol{e}_{b^{\text{old}}}\right)\right]\right)\right) \qquad (14)$$

where B is the total number of bond types; $\boldsymbol{z}_{uv}^t$ is the representation vector of $(u, v)^t$ computed in Eq. (10); b^{old} is the old bond of (u, v); $\boldsymbol{e}_{b^{\text{old}}}$ and $\boldsymbol{e}_b$ are the edge feature vectors corresponding to the bond type b^{old} and b, respectively; f^{bond} is a neural network.

2.3 Updating States

After predicting a reaction triple $(u, v, b)^t$, our model will update two things: i) the hidden state $\boldsymbol{h}^t$ of the RNN, and ii) the node representations of the intermediate graph $\mathcal{G}^{t+1}$.

Updating RNN's hidden state. The hidden state of the RNN is updated as follows:

$$\boldsymbol{h}^t = \text{GRU}\left(\boldsymbol{h}^{t-1}, \boldsymbol{z}_{uv}^t\right) \qquad (15)$$

where GRU stands for Gated Recurrent Units [4]; $\boldsymbol{z}_{uv}^t$ is the representation vector of the atom pair $(u, v)^t$ (see Eq. 10). Eq. (15) allows the model to keep track of all the graph transformations so far so it can make more accurate prediction later.

Updating node representations. First, given the predicted reaction triple $(u, v, b)^t$, we update the neighbor set of u and v as follows:

$$\mathcal{N}^t(u) \quad = \quad \left(\mathcal{N}^{t-1}(u) \backslash \left(v, b^{\text{old}}\right)\right) \cup (v, b) \qquad (16)$$

$$\mathcal{N}^t(v) \quad = \quad \left(\mathcal{N}^{t-1}(v) \backslash \left(u, b^{\text{old}}\right)\right) \cup (u, b) \qquad (17)$$

Next, we performs one step of message passing to update the node representation vectors of u and v with the new neighbor sets computed from above:

$$\boldsymbol{x}_u^t \quad = \quad \text{MessagePassing}\left(\boldsymbol{x}_u^{t-1}, \boldsymbol{v}_u, \mathcal{N}^t(u)\right) \qquad (18)$$

$$\boldsymbol{x}_v^t \quad = \quad \text{MessagePassing}\left(\boldsymbol{x}_v^{t-1}, \boldsymbol{v}_v, \mathcal{N}^t(v)\right) \qquad (19)$$

where the MessagePassing(.) function is defined in Eq. (5). For other nodes in the graph to be aware of the new structures of u and v, we need to perform several message passing steps for all nodes in the graph after Eqs. (18, 19). However, it is very costly to run for every prediction step t. Sometimes it is unnecessary since far-away bonds are less likely to be affected by the current bond change. Therefore, in our model, we limit the number of message passing updates for all nodes at step t to be 1.

2.4 Training

Loss function plays a central role in achieving fast training and high performance of our model. We design the following loss:

$$\mathcal{L} \quad = \quad \lambda_1 \mathcal{L}^{\text{A2C}} + \lambda_2 \mathcal{L}^{\text{value}} + \lambda_3 \mathcal{L}^{\text{ap}} + \lambda_4 \mathcal{L}^{\text{over-length}}$$

where $\mathcal{L}^{\text{A2C}}$ is the Advantage Actor-Critic (A2C) loss [21] to account for the correct sequence of reaction triples; $\mathcal{L}^{\text{value}}$ is the loss for estimating the value function used in A2C; $\mathcal{L}^{\text{ap}}$ accounts for binary change in the bond of an atom pair; and $\mathcal{L}^{\text{over-length}}$

penalizes long predicted sequences; and $\lambda_1, ..., \lambda_4 > 0$ are tunable coefficients. The component losses are explained in the following.

2.4.1 Reaction triple loss. The loss follows a policy gradient method known as Advantage Actor-Critic (A2C):

$$\begin{aligned}\mathcal{L}^{\text{A2C}} &= -A^{T_{\text{end}}}_{\text{signal}} \log \pi \left(\xi^{T_{\text{end}}}\right) - \sum_{t=0}^{T_{\text{end}}-1} A^t_{\text{signal}} \log p \left(\xi^t\right) \\ &\quad - \sum_{t=0}^{T_{\text{end}}-1} A^t_{\text{atom pair}} \log p \left((u,v)^t\right) \\ &\quad - \sum_{t=0}^{T_{\text{end}}-1} A^t_{\text{bond}} \log p \left(b^t\right) \qquad (20)\end{aligned}$$

where T_{end} is the first step that $\xi = 0$; A_{signal}, $A_{\text{atom pair}}$ and A_{bond} are called *advantages*. To compute these advantages, we use the unbiased estimations called Temporal Different errors, defined as:

$$A^t_{\text{signal}} = r^t_{\text{signal}} + \gamma V_\phi\left(Z^{t+1}_K\right) - V_\phi\left(Z^t_K\right) \qquad (21)$$

$$A^t_{\text{atom pair}} = r^t_{\text{atom pair}} + \gamma V_\phi\left(Z^{t+1}_K\right) - V_\phi\left(Z^t_K\right) \qquad (22)$$

$$A^t_{\text{bond}} = r^t_{\text{bond}} + \gamma V_\phi\left(Z^{t+1}_K\right) - V_\phi\left(Z^t_K\right) \qquad (23)$$

where r^t_{signal}, $r^t_{\text{atom pair}}$, r^t_{bond} are immediate rewards at step t; at the final step $t = T_{\text{end}}$, the model receives additional delayed rewards; γ is the discount factor; and V_ϕ is the parametric value function. We train V_ϕ using the following mean square error loss:

$$\mathcal{L}^{\text{value}} = \sum_{t=0}^{T_{\text{end}}} \left\| V_\phi\left(Z^t_K\right) - R^t \right\|^2 \qquad (24)$$

R^t is the return at step t computed as:

$$R^t = \sum_{t+1}^{T_{\text{end}}-1} \left(r^t_{\text{signal}} + r^t_{\text{atom pair}} + r^t_{\text{bond}}\right) + r_{\text{product}}$$

where r_{product} is the reward value indicating that whether the predicted product molecules match the true target product molecules or not.

Episode termination during training. Although the loss defined in Eq. (20) is correct, it is not good to use in practice because: i) If our model selects a wrong sub-action (wrong signal, atom pair or bond) at any sub-step of the step T_{wrong} ($T_{\text{wrong}} < T_{\text{end}}$), the whole predicted sequence will be incorrect regardless of what will be predicted from $T_{\text{wrong}} + 1$ to T_{end}. Therefore, computing the loss for actions from $T_{\text{wrong}} + 1$ to T_{end} is redundant. ii) More importantly, the incorrect updates of the graph structure at subsequent steps from $T_{\text{wrong}} + 1$ to T_{end} will lead to cumulative prediction errors which make the training of our model much more difficult.

To resolve this issue, during training, we use a binary vector $\zeta \in \{0,1\}^{3T}$ to keep track of the first wrong sub-action: $\zeta^\tau = \begin{cases} 1 & \text{if } \tau \le \tau_{\text{wrong}} \\ 0 & \text{if } \tau > \tau_{\text{wrong}} \end{cases}$ where τ_{wrong} denotes the first sub-step at which our model chooses a wrong sub-action. The actor-critic loss in Eq. (20) now becomes:

$$\begin{aligned}\mathcal{L}^{\text{A2C}} &= -\sum_{t=0}^{T} \zeta^t A^t_{\text{signal}} \log p \left(\xi^t\right) \\ &\quad - \sum_{t=0}^{T} \zeta^{(t+1)} A^t_{\text{atom pair}} \log p \left((u,v)^t\right) \\ &\quad - \sum_{t=0}^{T} \zeta^{(t+2)} A^t_{\text{bond}} \log p \left(b^t\right) \qquad (25)\end{aligned}$$

where T is the maximum number of steps. Similarly, we change the value loss into:

$$\mathcal{L}^{\text{value}} = \sum_{t=0}^{T} \zeta^t \left\| V_\phi\left(Z^t_K\right) - R^t \right\|^2$$

2.4.2 Reaction atom pair loss. To train our model to assign higher scores to reaction atom pairs and lower to non-reaction atom pairs, we use the following cross-entropy loss function:

$$\mathcal{L}^{\text{ap}} = -\sum_{t=0}^{T_{\text{wrong}}} \sum_{i \ne j} \eta^t_{ij} \left(y_{ij} \log p^t_{ij} + (1 - y_{ij}) \log(1 - p^t_{ij})\right) \qquad (26)$$

where $\eta^t_{ij} \in \{0,1\}$ is a mask of the atom pair (i,j) at step t; $y_{ij} \in \{0,1\}$ is the label indicating whether the atom pair (i,j) is a reaction atom pair or not; $p^t_{ij} = \text{sigmoid}(s^t_{ij})$ (see Eq. (11)).

2.4.3 Constraint on the sequence length. One major difficulty of the chemical reaction prediction problem is to know exactly when to stop prediction so we can make accurate inference. By forcing the model to stop immediately when making wrong prediction, we can prevent cumulative error and significantly reduce variance during training. But it also comes with a cost: The model cannot learn (because it does not have to learn) when to stop. This phenomenon can be visualized easily as the model predicts 1 for the signal at every step t during inference. In order to make the model aware of the correct sequence length during training, we define a loss that punishes the model if it produces a longer sequence than the ground-truth sequence:

$$\mathcal{L}^{\text{over-length}} = -\sum_{T^{\text{gt}}_{\text{end}} \le t < T_{\text{end}}} \log p \left(\xi^t = 0\right) \qquad (27)$$

where $T^{\text{gt}}_{\text{end}}$ is the end step of the ground-truth sequence. Note that the loss in Eq. (27) is not applied when $T_{\text{end}} \le T^{\text{gt}}_{\text{end}}$. The reason is that forcing $\xi^t = 1$ with $T_{\text{end}} \le t < T^{\text{gt}}_{\text{end}}$ is not theoretically correct because all the signals after T_{end} are assumed to be 0. The incentive to force T_{end} close to $T^{\text{gt}}_{\text{end}}$ when it is smaller than $T^{\text{gt}}_{\text{end}}$ has already been included in the advantages in Eq. (25).

3 EXPERIMENTS

3.1 Datasets

We evaluate our model on two standard datasets *USPTO-15k* (15K reactions) and *USPTO* (480K reactions) which have been used in previous works [2, 15, 26]. Details about these datasets are given in Table 1. The *USPTO* dataset contains reactant, reagent and product

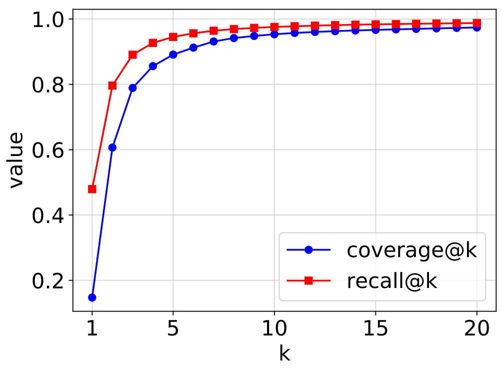


Figure 3: *Coverage@k* and *Recall@k* with respect to k for the *USPTO* dataset.

molecules represented as SMILES strings. Using RDKit[1], we convert the SMILES strings into molecule objects and store them as graphs. All atoms in the reactant and reagent molecules are uniquely identified via "atom map numbers" which remain unchanged during the reaction. Using this knowledge, we compare every atom pair in the input molecules with the counterpart in the product molecules to obtain a ground-truth set of reaction triples for training.

3.2 Reaction Atom Pair Prediction

In this section, we test our model's ability to identify reaction atom pairs by formulating it as a ranking problem with the scores computed in Eq. (11). Similar to [15], we use *Coverage@k* as an evaluation metric, which is the proportion of reactions that have *all* ground-truth reaction atom pairs appearing in the top k predicted atom pairs. We compare our proposed graph neural network (GNN) with Weisfeiler-Lehman Network (WLN) [15] and Column Network (CLN) [23]. From Table 3, we observe that our GNN clearly outperforms WLN and CLN in all cases. We attribute this improvement to two things: i) the use of Highway Networks [30] in updating node states, and ii) the separation of the node state vectors and the node feature vectors. Both help avoid information loss during message passing between nodes. The other two models, however, lack these features.

3.3 Top-K Atom Pair Extraction

The performance of our model depends on the number of selected top atom pairs K. The value of K presents a trade-off between coverage and efficiency. To find a good K, we use the metric *Coverage@k* defined in Sec. 3.2 and the metric *Recall@k* defined as the proportion of correct atom pairs that appear in top k. Fig. 3 shows the values of *Coverage@k* and *Recall@k* for the *USPTO* dataset with respect to k. We see that both curves increase rapidly when $k < 10$ and stablize when $k > 10$. We also ran experiments with $k = 10$, 15, 20 and observed that their prediction results are quite similar. Hence, in what follows, we select $K = 10$ for efficiency.

[1]https://www.rdkit.org/

3.4 Reaction Product Prediction

We validates GTPN on the reaction product prediction task by comparing it with recent state-of-the-art methods [15, 26] using the accuracy metric. Table 4 shows the prediction results. We generate multiple reaction product candidates using beam search with the beam width $N = 20$.

In brief, we compute the normalized-over-length log probabilities of N predicted sequences of reaction triples and sort these values in descending order to get a rank list of N possible reaction outcomes. Given a predicted sequence of reaction triples $(u, v, b)^{0:T}$, we can generate reaction products from input reactants simply by replacing the old bond of $(u, v)^t$ with b^t. However, these products are not guaranteed to be valid (e.g., maximum valence constraint violation or aromatic molecules cannot be kekulized) so we post-process the outputs by removing all invalid products. The removal increases the top-1 accuracy by about 8% and 10% on *USPTO-15k* and *USPTO*, respectively. Due to the permutation invariance of the predicted sequence of reaction triples, some product candidates are duplicate and will also be removed. This does not lead to any change in *P@1* but slightly improves *P@3* and *P@5* by about 0.5-1% on the two datasets.

Overall, GTPN with beam search and post-processing outperforms both WLDN [15] and Seq2Seq [26] in the top-1 accuracy. For the top-3 and top-5, our model's performance is comparable to WLDN's on *USPTO-15k* and is worse than WLDN's on *USPTO*. It is not surprising since our model is trained to accurately predict the top-1 outcomes instead of ranking the candidates directly like WLDN. It is important to emphasize that we did not tune the model hyper-parameters when training on *USPTO* but reused the optimal settings from *USPTO-15k* (which is 25 times smaller than *USPTO*) so the results may not be optimal (see Appendix 5.1 for more training detail).

3.5 Comparison with ELECTRO

ELECTRO [2] is a recently proposed method that also formulate the reaction product prediction problem as a graph transformation problem like ours. However, ELECTRO regards a reaction as an ordered sequence of transformations that alternates between removing and adding a single bond while our model assumes no specific order of bond changes as well as the number of valences that a bond can change. Thus, our model is more generic than ELECTRO in term of model design and can cover a much larger set of reactions.

Because ELECTRO was evaluated on a subset of *USPTO* that contains only reactions with linear chain topology and single-valence bond changes, we prepare a similar dataset by following the procedure described in [2] to make a fair comparison. After processing, our test set contains 29,808 reactions, close to the their reported number of 29,360 reactions. We use the same model as in Section 3.4 with settings are provided in Appendix 5.1. Similar to [2], we also use beam search and post-processing to compute results. As shown in Table 5, GTPN achieves the highest top-1 accuracy of 87.35%, outperforming ELECTRO and WLDN by 0.35% and 3%, respectively. For the top-3 and top-5 accuracies, our model, however, does worse than the other two. Especially, while both ELECTRO and WLDN have big jumps from *P@1* to *P@3* with about 7% improvement,

Dataset		*#reactions*	*#changes*	*#molecules*	*#atoms*	*#bonds*
USPTO-15k	train	10,500	1 | 11 | 2.3	1 | 20 | 3.6	4 | 100 | 34.9	3 | 110 | 34.7
	valid	1,500	1 | 11 | 2.3	1 | 20 | 3.6	7 | 94 | 34.5	5 | 99 | 34.2
	test	3,000	1 | 11 | 2.3	1 | 16 | 3.6	7 | 98 | 34.9	5 | 102 | 34.7
USPTO	train	409,035	1 | 6 | 2.2	2 | 29 | 4.8	9 | 150 | 39.7	6 | 165 | 38.6
	valid	30,000	1 | 6 | 2.2	2 | 25 | 4.8	9 | 150 | 39.6	7 | 158 | 38.5
	test	40,000	1 | 6 | 2.2	2 | 22 | 4.8	9 | 150 | 39.8	7 | 162 | 38.7

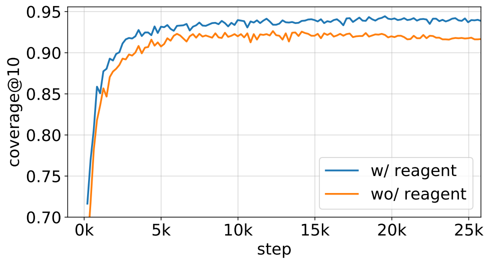
Table 1: Statistics of *USPTO-15k* and *USPTO* datasets. *“changes”* means bond changes, *“molecules”* means reactants and reagents in a reaction; *“atoms”* and *“bonds”* are defined for a molecule. Apart from *“#reactions”*, other columns are presented in the format “min | max | mean”.

Dataset		%reactions containing reagents	%reagents over input molecules
USPTO-15k	train	63.1%	41.3%
	valid	65.3%	42.3%
	test	63.6%	40.9%
USPTO	train	79.7%	54.0%
	valid	80.0%	54.4%
	test	79.9%	54.2%

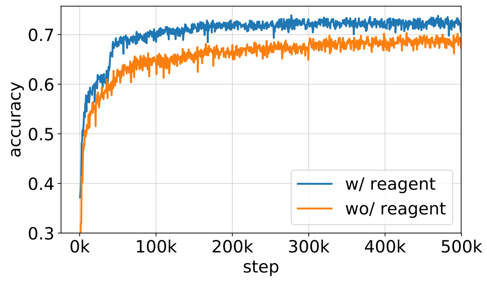
Table 2: Proportion of reactions containing reagents and proportion of reagents over input molecules on *USPTO-15k* and *USPTO*.

Model	*USPTO-15k*		*USPTO*	
	C@6	*C@10*	*C@6*	*C@10*
WLN [15]	88.45	93.34	90.97	95.26
CLN [23]	88.68	93.07	90.72	94.80
Our GNN	**88.92**	**93.57**	**91.24**	**95.33**

Table 3: Results for reaction atom pair prediction. *C@k* is coverage at k. Best results are highlighted in bold.



(a) Atom pair prediction



(b) Reaction triple prediction

Figure 4: Learning curves of our model with and without using reagent information on *USPTO-15k*.

GTPN only has 3% increase. We think the main reason for this is that GTPN is designed to be a generic model instead of focusing on this specific kind of reactions.

3.6 Using Reagent Information

From Figs. 4a and 4b, we see that using reagent information improves the performance of our model by about 2% on the atom pair prediction task and about 4% on the reaction triple prediction task.

4 RELATED WORK

4.1 Learning to Predict Chemical Reaction

In chemical reaction prediction, machine learning has replaced rule-based methods [3] for better generalizability and scalability. Existing machine learning-based techniques are either template-free [9, 15, 17] or template-based [5, 27, 33]. Both groups share the same mechanism: running multiple stages with the aid of reaction templates or rules. In [33], the authors proposed a two-stage model that first classifies reactions into different types based on the neural fingerprint vectors [8] of reactant and reagent molecules. Then, it applies pre-designed SMARTS transformation on the reactants with respect to the most suitable predicted reaction type to generate the reaction products.

The method of [15] also solve the reaction product prediction in two steps. First, they predict which atom pairs are likely to be reactive using a variant of graph neural networks called Weisfeiler-Lehman Networks (WLNs). In the next step, they do almost the same as [5] by modifying the bond type between the selected atom pairs (with chemical rules satisfied) to create product candidates and rank them (with reactant molecules as addition input) using another kind of WLNs called Weifeiler-Lehman Different Networks (WLDNs).

To the best of our knowledge, [15] is the first work that achieves remarkable results (with the top-1 accuracy is about 79.6%) on the

Model	*USPTO-15k*			*USPTO*		
	P@1	*P@3*	*P@5*	*P@1*	*P@3*	*P@5*
WLDN [15]	*76.7*	*85.6*	***86.8***	*79.6*	***87.7***	***89.2***
Seq2Seq [26]	-	-	-	80.3*	86.2*	87.5*
GTPN	**82.39**	**85.73**	**86.78**	**83.20**	86.03	86.48

Table 4: Results for reaction prediction. *P@k* is precision at k. State-of-the-art results from [15] are written in italic. Results from [26] are marked with * and they are computed on a slightly different version of *USPTO* that contains only single-product reactions. Best results are highlighted in bold.

Model	*Processed USPTO*		
	P@1	*P@3*	*P@5*
WLDN [15]	84.0	91.1	92.3
ELECTRO [2]	87.0	**94.5**	**95.9**
GTPN	**87.35**	90.22	90.68

Table 5: Results for the reaction prediction task. *P@k* is the precision at k. Best results are highlighted in bold.

large *USPTO* dataset containing more than 480 thousands reactions. Works of [22] and [26] avoid multi-stage prediction by building a seq2seq model that generates the (canonical) SMILES string of the single product from the concatenated SMILES strings of the reactants and reagents in an end-to-end manner. However, their methods cannot deal with sets of reactants/reagents/products properly as well as cannot demonstrate reaction steps concretely.

The most recent work on this topic is [2] which solves the reaction prediction problem by predicting a sequence of bond changes given input reactants and reagents represented as graphs. To handle ordering, they only select reactions with predefined topology. Our method, by contrast, is order-free and can be applied to almost any kind of reactions.

4.2 Graph Neural Networks for Modeling Molecules

In recent years, there has been a fast development of graph neural networks (GNNs) for modeling molecules. These models are proposed to solve different problems in chemistry including toxicity prediction [8], drug activity classification [6, 24, 28], protein interface prediction [10] and drug generation [14, 29]. Most of them can be regarded as variants of message-passing graph neural networks (MPGNNs) [11].

4.3 Reinforcement Learning for Structural Reasoning

Reinforcement learning (RL) has become a standard approach to many structural reasoning problems[2] because it allows agents to perform discrete actions. A typical example of using RL for structural reasoning is drug generation [20, 34]. Both [20] and [34] learn the same generative policy whose action set including: i) adding a new atom or a molecular scaffold to the intermediate graph, ii) connecting existing pair of atoms with bonds, and iii) terminating generation. However, [34] uses an adversarial loss to enforce global chemical constraints on the generated molecules as a whole instead of using the common reconstruction loss as in [20]. Other examples are path-based relational reasoning in knowledge graphs [7] and learning combinatorial optimization over graphs [18].

[2]Structural reasoning is a problem of inferring or generating new structure (e.g. objects with relations)

5 DISCUSSION

We have introduced a novel method named Graph Transformation Policy Network (GTPN) for predicting products of a chemical reaction. GTPN uses graph neural networks to represent input reactant and reagent molecules, and uses reinforcement learning to find an optimal sequence of bond changes that transforms the reactants into products. We train GTPN using the Advantage Actor-Critic (A2C) method with appropriate constraints to account for notable aspects of chemical reaction. Experiments on real datasets have demonstrated the competitiveness of our model.

Although the GTPN was proposed to solve the chemical reaction problem, it is indeed generic to solve the graph transformation problem, which can be useful in reasoning about relations (e.g., see [35]) and changes in relation. Open rooms include addressing dynamic graphs over time, extending toward full chemical planning and structural reasoning using RL.

REFERENCES

[1] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. 2016. Interaction Networks for Learning about Objects, Relations and Physics. In *Advances in neural information processing systems*. 4502–4510.

[2] John Bradshaw, Matt J Kusner, Brooks Paige, Marwin HS Segler, and José Miguel Hernández-Lobato. 2018. Predicting Electron Paths. *arXiv preprint arXiv:1805.10970* (2018).

[3] Jonathan H Chen and Pierre Baldi. 2009. No electron left behind: a rule-based expert system to predict chemical reactions and reaction mechanisms. *Journal of chemical information and modeling* 49, 9 (2009), 2034–2043.

[4] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *EMNLP* (2014).

[5] Connor W Coley, Regina Barzilay, Tommi S Jaakkola, William H Green, and Klavs F Jensen. 2017. Prediction of organic reaction outcomes using machine learning. *ACS central science* 3, 5 (2017), 434–443.

[6] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative Embeddings of Latent Variable Models for Structured Data. In *International Conference on Machine Learning*. 2702–2711.

[7] Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. 2018. Go for a Walk and Arrive at the Answer: Reasoning over Paths in Knowledge Bases using Reinforcement Learning. *ICLR* (2018).

[8] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *Advances in Neural Information Processing Systems*. 2224–2232.

[9] David Fooshee, Aaron Mood, Eugene Gutman, Mohammadamin Tavakoli, Gregor Urban, Frances Liu, Nancy Huynh, David Van Vranken, and Pierre Baldi. 2018.

Deep learning for chemical reaction prediction. *Molecular Systems Design & Engineering* (2018).
[10] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. 2017. Protein Interface Prediction using Graph Convolutional Networks. In *Advances in Neural Information Processing Systems*. 6530–6539.
[11] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *Proceedings of the International Conference on Machine Learning*.
[12] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of Advances in Neural Information Processing Systems*. 1025–1035.
[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[14] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. 2018. Junction Tree Variational Autoencoder for Molecular Graph Generation. *International Conference on Machine Learning (ICML)* (2018).
[15] Wengong Jin, Connor Coley, Regina Barzilay, and Tommi Jaakkola. 2017. Predicting Organic Reaction Outcomes with Weisfeiler-Lehman Network. In *Advances in Neural Information Processing Systems*. 2604–2613.
[16] Clemens Jochum, Johann Gasteiger, and Ivar Ugi. 1980. The principle of minimum chemical distance (PMCD). *Angewandte Chemie International Edition in English* 19, 7 (1980), 495–505.
[17] Matthew A Kayala and Pierre F Baldi. 2011. A Machine Learning Approach to Predict Chemical Reactions. In *Advances in Neural Information Processing Systems*. 747–755.
[18] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. 2017. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*. 6348–6358.
[19] Diederik P Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations (ICLR)* (2015).
[20] Yibo Li, Liangren Zhang, and Zhenming Liu. 2018. Multi-Objective De Novo Drug Design with Conditional Graph Generative Model. *Journal of Cheminformatics* 10 (2018).
[21] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.
[22] Juno Nam and Jurae Kim. 2016. Linking the Neural Machine Translation and the Prediction of Organic Chemistry Reactions. *arXiv preprint arXiv:1612.09529* (2016).
[23] Trang Pham, Truyen Tran, Dinh Phung, and Svetha Venkatesh. 2017. Column Networks for Collective Classification. In *Proceedings of AAAI Conference on Artificial Intelligence*.
[24] Trang Pham, Truyen Tran, and Svetha Venkatesh. 2018. Graph Memory Networks for Molecular Activity Prediction. *ICPR* (2018).
[25] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling Relational Data with Graph Convolutional Networks. *15th European Semantic Web Conference (ESWC-18)* (2018).
[26] Philippe Schwaller, Theophile Gaudin, David Lanyi, Costas Bekas, and Teodoro Laino. 2018. "Found in Translation": Predicting Outcome of Complex Organic Chemistry Reactions using Neural Sequence-to-Sequence Models. *Chemical Science* 9 (2018), 6091–6098.
[27] Marwin HS Segler and Mark P Waller. 2017. Neural-Symbolic Machine Learning for Retrosynthesis and Reaction Prediction. *Chemistry–A European Journal* 23, 25 (2017), 5966–5971.
[28] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. 2011. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research* 12, Sep (2011), 2539–2561.
[29] Martin Simonovsky and Nikos Komodakis. 2018. GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders. *arXiv preprint arXiv:1802.03480* (2018).
[30] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Training very deep networks. In *Advances in neural information processing systems*. 2377–2385.
[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*. 5998–6008.
[32] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. 2018. Non-local neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
[33] Jennifer N Wei, David Duvenaud, and Alán Aspuru-Guzik. 2016. Neural Networks for the Prediction of Organic Chemistry Reactions. *ACS Central Science* 2, 10 (2016), 725–732.
[34] Jiaxuan You, Bowen Liu, Rex Ying, Vijay Pande, and Jure Leskovec. 2018. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. *NIPS* (2018).
[35] Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, et al. 2018. Relational Deep Reinforcement Learning. *arXiv preprint arXiv:1806.01830* (2018).

APPENDIX

5.1 Model Configurations

We optimize our model's hyper-parameters in two stages: First, we tune the hyper-parameters of the GNN and the NPPN for the reaction atom pair prediction task. Then, we fix the optimal settings of the first two components and optimize the hyper-parameters of the PN for the reaction product prediction task.

We provide details about the settings that give good results on the *USPTO-15k* dataset below. With these settings, we trained another model on the *USPTO* dataset from scratch. Because training on the large dataset such as the *USPTO* takes time, we did not tune hyper-parameters on the *USPTO*, even though it is possible to increase model sizes for better performance. Unless explicitly stated, all neural networks in our model have 2 layers with the same number of hidden units, Leaky ReLU activation and residual connections [13].

Graph Neural Network (GNN). There are 72 different types of atom depending on their atomic numbers and 5 different types of bond including NONE, SINGLE, DOUBLE, TRIPLE and AROMATIC. The size of embedding vectors for atom and bond are 51 and 21, respectively. Apart from atom type, each atom has 5 more attributes including: degree, explicit valence, explicit number of Hs, charge, part of a ring or not. These attributes are normalized to the range of [0, 1] and are concatenated to the atom embedding vector to form a final atom feature vector of size 56. The state vector and the neighbor message vector for an atom both have the size of 99. The number of message passing steps is 6.

Node Pair Prediction Network (NPPN). This component consists of two parts. The first part computes the representation vector z_{ij} of an atom pair (i, j) using a neural network with hidden size of 71. The second part maps z_{ij} to an unnormalized score s_{ij} using the function $f^{\text{atom pair}}$ (see Eqs. (11)). This function is also a neural network with hidden size of 51.

Policy Network (PN). The recurrent network is a GRU [4] with 101 hidden units. The value function V_ϕ is a neural network with 99 hidden units. The two functions f^{signal} for computing signal scores (see Eq. (12)) and f^{bond} for computing scores over bond types (see Eq. (14)) are neural networks with 81 hidden units.

Training. At each step, we set the reward to be 1.0 for correct prediction of signal/atom pair/bond type and -1.0 for incorrect prediction. After the prediction sequence is terminated (zero signal was emitted), we check whether the entire set of predicted reaction triples is correct or not. If it is correct, we give the model a reward value of 2.0, otherwise -2.0. From the rewards and estimated values for signal, atom pair and bond type, we define the Advantage Actor Critic loss (A2C) as in Eq. (25). The coefficients of components in the final loss $\mathcal{L}$ are set empirically as follows:

$$\mathcal{L} = \mathcal{L}^{\text{A2C}} + 0.5 \times \mathcal{L}^{\text{value}} + \mathcal{L}^{\text{ap}} + 0.2 \times \mathcal{L}^{\text{over length}}$$

We trained our model using Adam [19] with the initial learning rate of 0.001 for both *USPTO-15k* and *USPTO*. For *USPTO-15k*, the learning rate will decrease by half if the *Precision@1* does not improve on the validation set after 1,000 steps until it reaches the minimum value of 5×10^{-5}. For *USPTO*, the decay rate is 0.8 after every 500 steps of no improvement until reaching the minimum learning rate is 2×10^{-5}. The maximum number of training iterations is 10^6 and the batch size is 20.

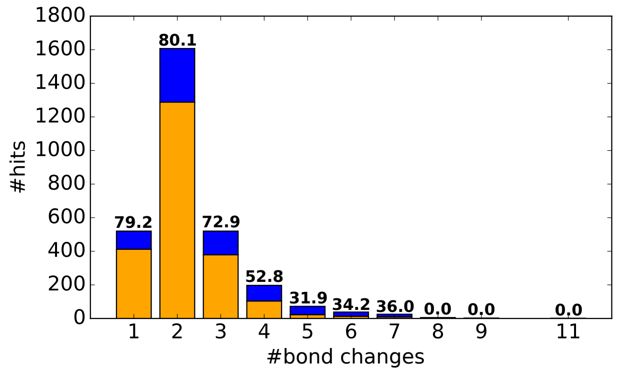
5.2 Error Analysis

In this section, we analyze several kinds of errors that our model makes during prediction. All the results below are computed on the *USPTO-15k* dataset by using beam search decoding with the beam width $N = 20$ and no post-processing.

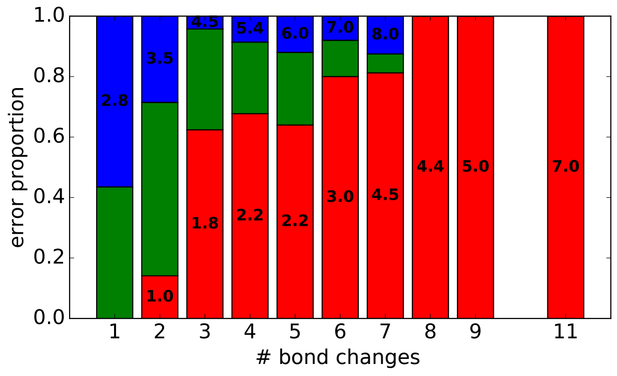
Errors grouped by numbers of transformation steps. Fig. 5a shows the top-1 accuracies for reactions with different numbers of transformation steps. Our model performs poorly on reactions with many steps. However, these kinds of reactions only account for a small proportion in the dataset. From Fig. 5b, we see that the lengths of the error sequences tend to be shorter than the lengths of the ground-truth sequences.

Errors caused by signal/atom pair/bond type predictions. We define an "erroneous sub-action" as the first sub-action that our model makes a wrong decision. In Fig. 6a, we plot the proportion of errors with respect to three different types of sub-actions: signal prediction, atom pair prediction, bond type prediction. Clearly, atom pair prediction causes the most errors (nearly two third). This makes sense because this sub-action is harder than signal prediction and bond type prediction. Therefore, more effort should be put on improving the atom pair prediction.

Errors caused by structural symmetry. There exists cases in which different sequences of transformations can result in the same products due to structural similarity. This kind of errors accounts for 5.7% of the top-1 errors on the *USPTO-15k* dataset as shown in Fig. 6b.

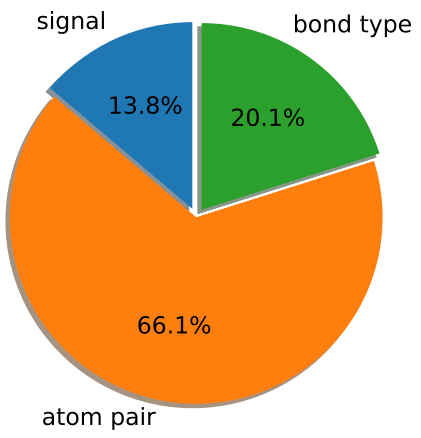


(a) Top-1 accuracies grouped by number of transformation steps.

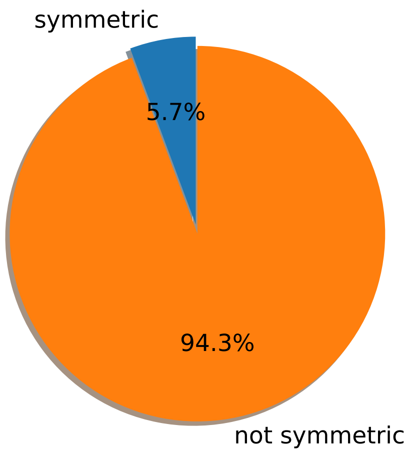


(b) Proportions of top-1 errors grouped by lengths.

Figure 5: Performance with respect to different numbers of transformation steps. In (a), blue: all reactions having that sequence length; orange: correct predicted reactions. In (b), red: the predicted sequence is shorter (than the ground-truth sequence); green: the predicted and the ground-truth have the same length; blue: the predicted sequence is longer; number indicate the average length.



(a) Proportions of top-1 errors caused by types of sub-actions.



(b) Proportions of top-1 errors caused by structural symmetry.

Figure 6: Errors caused by types of sub-actions (a) and by structural symmetry (b)