Predicting Chemical Reaction Type and Reaction Products with Recurrent Neural Networks

Anvita Gupta, Department of Computer Science, Stanford University, `avgupta@cs.stanford.edu`

Abstract

Synthesizing chemical molecules is a necessary and often time-consuming process in fields like drug discovery and materials science. Here, we use recurrent neural networks to automate two tasks in chemical synthesis: 1) classifying the type of reaction two molecules will participate in, and 2) generating the correct products given the reactants and reaction type. We achieve 96% accuracy and 99% AUPRC in the task of reaction type classification, significantly improving on the baseline and previous approaches. In addition, our model for reaction-type generation stabilizes at 87.6% molecular similarity between the predicted and actual products, and reaches 96.6%. We demonstrate considerable improvement over rule-based prediction systems. These two completely data-driven systems can be useful to synthetic chemists seeking to predict the outcome of chemical reactions.

1 Introduction

Constructing the best synthesis pathway for an organic chemical molecule is a crucial problem in drug discovery, environmental science, material science, and many other disciplines requiring organic materials. Synthetic chemists construct synthesis pathways by chaining together several reactions; this process requires a great deal of chemical knowledge, and mistakes are often costly and time consuming.

Here, we use deep recurrent and convolutional neural networks to automate two tasks to simplify chemical synthesis planning for chemists. First, given some number of reactants (a, b), we aim to predict what type of reaction (t) they will participate in. Secondly, given particular reactants (a, b) and reaction type t, we aim to predict the product(s) $(c, d, ...)$ of that particular reaction. Every chemical molecule is represented as a string using the SMILES grammar, as described in [2], which allows us to use NLP techniques.

1.1 Related Work

Schneider *et al* implemented reaction type classification using a predefined reaction fingerprint, where the fingerprint was a bitstring encoding the presence or absence of certain substructures [8]. This method used both the reactant and product fingerprints to achieve a test set accuracy of 0.97%.

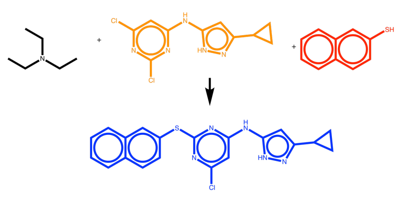
Generating the product of chemical reactions has been largely accomplished by rule-based approaches. Rules are extracted from some large dataset and applied to every new reaction; the outputs from all rules are then prioritized, as done by Segler and Waller [10]. One form of rule-based extraction is template extraction; Wei and Coley extract thousands of reaction templates from US patents, and for any query reaction, they predict a set of possible candidates by applying each template to the query[4]. The candidates are then prioritized. Liu *et al* attempted retrosynthetic prediction – predicting reactants from products, using a Sequence to Sequence architecture [5]. However, they were not able to improve significantly on a baseline of a rule-based expert system, achieving a top 1 accuracy of 34.1% and top 50 accuracy of 71.9%.

Most recently, at NIPS 2017, IBM Zurich published research using Sequence to Sequence architectures to predict outcomes of organic chemistry reactions [9]; their model was trained on more than 1 million reactions and achieved a top-1 accuracy of 65.4% on one dataset and 83.2% on the other. However, they do not take into account reaction type or report molecular similarity of all outputs.

2 Methods

2.1 Dataset and Preprocessing

For both type prediction and product prediction, we use a set of 50,000 Reactions from US Patents, which are represented as SMILES and are a subset of the larger collection from Lowe [1]. This subset has been curated and classified into 50 reaction types, and was additionally used for reaction classification in [8]. As described in [2], in the SMILES grammar each atom is represented by its symbol in the periodic table. The model is traversed methodically starting with some atom; different chemical libraries complete the traversal of the molecule in different ways, and so the same molecule can be mapped to multiple SMILES strings. Consequently, we *canonicalized* the reaction SMILES using the chemical library Rdkit, so each molecule was traversed in the same way.



CCN(CC)CC.Clc1cc(Nc2cc(C3CC3)n[nH]2)nc(Cl)n1.
Sc1ccc2ccccc2c1>>Clc1cc(Nc2cc(C3CC3)n[nH]2)nc(Sc2
ccc3ccccc3c2)n1

Figure 1: Example reaction and its SMILES representation.

Each reaction SMILES originally contained a mapping of each atom from reactant to product. This atom mapping was removed during preprocessing. Salts and ions, such as "[Na+][Cl-]" were also removed, as these salts were only added to stabilize the molecule in a solid state and did not participate in the reaction. The reactions were also preprocessed by length such that the center 80% of the data was retained. The data was preprocessed by length in order to remove outliers, such as reactions that contained DNA or other large molecules. An example of a preprocessed SMILES string can be seen in Figure 1.

Finally, every reaction was tokenized, where each token corresponds to a chemical element or number. "G" and "E" were used as sentinels for "go"(start) and "end" respectively; these tokens were used to signal the start and end of the reactants and products. The tokenization procedure yielded 54 total tokens; the data was lastly one-hot-encoded.

2.2 Recurrent Neural Network Overview

For both reaction type classification and product prediction, we implement several different model architectures employing Recurrent Neural Networks (RNNs). RNNs maintain a hidden state which is modified over time as a function of the previous hidden state and the input at that time step. RNNs consist of several recurrent units which modify this hidden state. We will describe two such recurrent units here; the Long Term Short Term (LSTM) recurrent unit and the Gated Recurrent Unit (GRU).

At each time step t, the LSTM processes an input element x_t and maintains both a hidden state h_{t-1} and a cell state C_{t-1}. The hidden and cell states are modified through three gates: the forget gate, the input gate, and the output gate. Each gate depends on the last hidden state h_{t-1} and current input x_t, and outputs a number between 0 and 1. The forget gate f_t learns to what extent the previous cell state should be forgotten; the input gate i_t learns to what extent the new cell state $\hat{C}_t$ should modify the propagated cell state C_t, and the output gate o_t predicts how much the cell state should influence the propagated hidden state h_t (as h_t is passed on to upper layers, unlike the cell state). In addition, the LSTM may produce an output vector y_t at each time step, where $y_t = softmax(W_y h_t + b_y)$. This makes LSTMs ideal for use as generative models, as they are able to model the probability distribution of the input language [3]. Alternatively, an LSTM used for classification rather than generation may only output a vector at the final time step $y_n = softmax(W_y h_n + b_y)$ in order to predict the probability of different output classes.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \qquad \hat{C}_t = W_c[h_{t-1}, x_t] + b_c \tag{1}$$

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \qquad C_t = f_t \otimes C_{t-1} + i_t \otimes \hat{C}_t \tag{2}$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \qquad h_t = o_t \otimes tanh(C_{t-1}) \tag{3}$$

The GRU is a simplified version of the LSTM in that the GRU only maintains the hidden state h rather than an additional hidden cell state. The GRU also has only two gates, as the input and forget gates are combined into an "update" gate. For a more detailed explanation on GRUs as opposed to LSTMs, we refer the reader to Olah [7].

2.3 Sequence to Sequence Overview

Sequence to sequence (Seq2Seq) architectures are often used for machine translation; here, we adapt them to "translate" reactants and type to chemical products. These architectures consist of two components; 1) an encoder which transforms the input sequence into some embedding in some latent space, and 2) a decoder, which takes the embedding from the encoder and attempts to generate the desired output sequence. Both the encoder and decoder in our Seq2Seq architectures are RNNs, specifically with GRUs, where GRUs are described in 2.2.

In standard Seq2Seq models, the last hidden state from the encoder (h_T) is passed on as the initial state for the decoder. However, for long sequences the standard Seq2Seq architecture is challenged, as the dependency of the decoder on the initial hidden state (and thus the encoded inputs) decreases with the length of the decoded sequence.

Consequently, we implement attention in our Seq2Seq model, as described by Luong et al [6]. With the attention model, at each time step t the decoder RNN issues what can be thought of as a "query" (h_t) for relevant information from the encoded outputs. This query is dotted with all of the encoded (source) hidden states h_s to calculate attention weights; these attention weights are normalized to produce a probability distribution over the encoded sequence. The context vector c_t is the weighted average of the attention weights and encoder outputs. This context vector encodes which information from the encoder is most important to the decoder at that time step; we produce an "attentional hidden state", $\bar{h_t}$ that depends on c_t and h_t. $\bar{h_t}$ is passed through another dense layer and softmax to yield $\bar{y_t}$, the predicted token at that time step.

$$a_t(s) = softmax(h_t \cdot h_s) \qquad c_t = \sum_s a_t(s) \cdot h_s \qquad \bar{h_t} = tanh(W_c[h_t; c_t]) \tag{4}$$

3 Results and Discussion

3.1 Reaction Type Classification

We first implemented a logistic regression model as a baseline for reaction type classification. Following the method in [8], the logistic regression took in a bitstring of length 167, where each bit corresponded to the presence of a particular chemical fragment (for example, the presence of a benzene ring). This featurization scheme is known as calculating a molecule's fingerprint. The fingerprints for multiple reactants were summed up to yield one feature vector per reaction. Each data point was weighted inversely to the frequency of its class, thus applying cost sensitive learning to deal with class imbalance. 80% of the data was used for training, and 20% for testing.

Using the Keras and Tensorflow frameworks, we built and trained a Recurrent Neural Network Classifier to learn predict reaction type through features learned directly from the SMILES. The architecture of the RNN consisted of two LSTM layers with hidden state $h = 512 \times 1$. The second LSTM layer's output at the final time step was fed to a dense output layer, with the number of neurons equal to the number of output classes. This dense layer had a softmax activation function, such that each of the outputs i corresponded to the predicted probability of the datapoint belonging to class i.

Using the Adam optimizer with learning rate $lr = 0.001$, we optimized the categorical cross entropy loss of this network. The network was trained using minibatch gradient descent with batch size $B = 100$, and batch normalization was applied between each layer. 60% of the data was retained for training, 20% for validation, and 20% for testing. The model was trained for 40 epochs on a NVIDIA Tesla K80 GPU provided through Azure.

In order to reduce overfitting and improve generalization, we experimented with varying levels of dropout. We tested several combinations of dropout probability p. As shown in Figure 2, increasing dropout improves test accuracy up until $p = 0.1$, but beyond this point, the test accuracy decreases as the network begins to underfit.

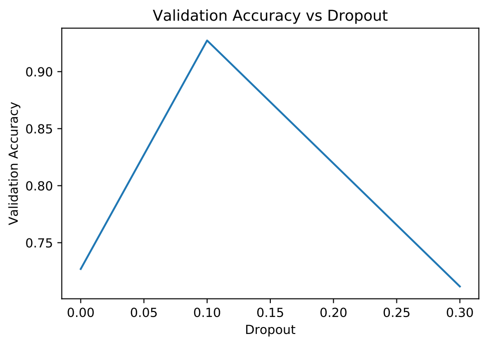


Figure 2: Validation accuracy for varying dropout levels (only performance with same dropout both layers illustrated here).

In addition, the relationship between dropout and generalization is highly nonlinear; increasing dropout in layer 2 improves generalization, as the network with dropout $p_1 = p_2 = 0.3$ performs worse than the network with $p_1 = 0.3, p_2 = 0.5$. The learning curves are shown in Figure 3. Qualitatively, the model with no dropout is clearly overfitting, with validation loss high and fluctuating widely. Adding Dropout of $p = 0.1$ in each layer clearly reduces overfitting and validation loss.

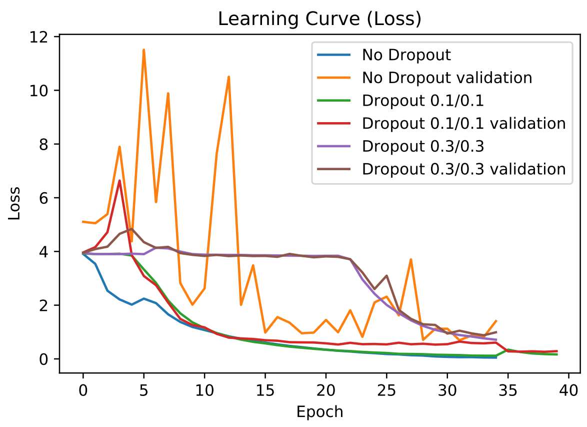
As shown in Table 1, we evaluated several metrics to quantitatively measure the performance of our models. We measured training accuracy, test accuracy, precision, and recall. As this was a multilabel classification problem, we computed a macro-average of precision and recall by averaging them across all different classes. The macro-average is often lower when the data exhibits class imbalance, as classes with few labels (and often worse performance) are given equal weight as classes with many labels. The micro-average is computed by pooling the true positives, false positives, and false negatives for each class, and then calculating the metrics as usual. Table 1 shows the macro-averaged precision, recall, and micro-averaged Area under the Precision Recall Curve (AUPRC). Both micro-averaged and macro-averaged values were within similar ranges. The best performing classifier was the model with Dropout $p_1 = p_2 = 0.1$, which achieved 0.9591 test accuracy with low

Model	Train Accuracy	Test Acc	Precision	Recall	AUPRC
No Dropout	0.988	0.9083	0.9131	0.9076	0.96
Dropout 0.1/0.1	0.951	0.9591	0.9596	0.9589	0.99
Dropout 0.3/0.3	0.777	0.7607	0.7903	0.7605	0.85
Dropout 0.3/0.5	0.8393	0.8834	0.8907	0.8777	0.95
Logistic Regression	0.8740	0.8277	0.8231	0.8263	0.65

Table 1: Training accuracy, Test Accuracy, Precision, Recall, and AUPRC. Precision and recall calculated with macro-averages, and AUPRC with micro-average. Macro and micro averages were in similar range.

overfitting, and an $AUPRC = 0.99$, significantly outperforming the logistic regression baseline. The precision recall curve for the best performing classifier is shown in 3b.

a)



b)

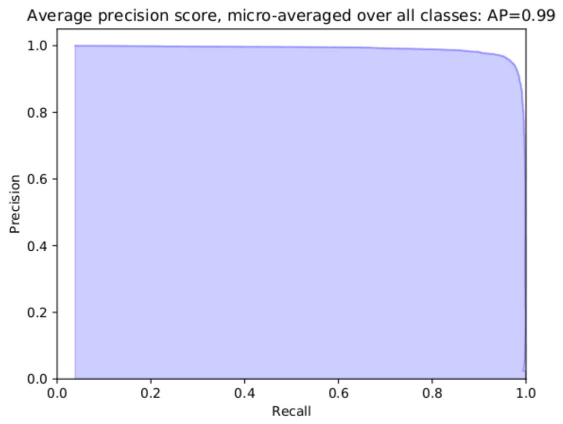


Figure 3: a) Learning curves for classifiers with varying dropout rates. b) Precision Recall Curve (AUPRC = AP = 0.99) for best performing classifier (Dropout = 0.1/0.1).

3.2 Product Prediction

The Seq2Seq architecture for generating reaction products was encoded in Pytorch; we tested both a standard Seq2Seq model, and a Seq2Seq model with a dot product mechanism of attention as described in [6]. On the Seq2Seq model with attention we also included the reaction type as input, where the type was encoded as a special token and prepended to the reactant sequence.

In all models, the decoder consisted of one GRU layer with dropout $p = 0.05$ and hidden state $h = 500$. The Adam optimizer was used, with $lr = 0.0001$, and the model was trained with minibatch gradient descent (batch size was 100). To combat the exploding gradient problem, all gradients were clipped at 50. The encoder was a bidirectional GRU, with one GRU run over the reactants and one run over the reversed reactants, and the outputs from the two GRUs concatenated at each time step. The encoder contained dropout $p = 0.1$. Each model was trained for 20,000 iterations on a Nvidia Tesla K80 GPU.

Training was done by teacher forcing - at each step, the correct next token was input, regardless of the token that predicted by the decoder. Every 500 iterations, 30 products were sampled. 20% of the reactions were reserved beforehand for this sampling procedure, so that the model only predicted outputs of reactions it had not been trained on. Decoding began with the sentinel character "G" and ended when the token "E" was sampled, or after 500 tokens maximally were sampled. As a comparison, a rule-based prediction system was coded in Rdkit following the strategy in [4].

The results from the models are shown in Table 2; the metrics were the percentage of valid SMILES strings and Tanimoto Similarity (molecular similarity) between the predicted and actual products. Molecular similarity provides better resolution of performance than accuracy, as molecular similarity may be improving as accuracy stays constant. The attention model with reactants peaks at a similarity of **0.953**, while the attention model with reactants and type reaches a similarity of **0.9668**, although both models stabilize at around **0.87**. Both models improve upon the rule-based extraction system. Qualitatively, and as shown in Figure 4a, the attention model with reactants and type appears to converge to the optimal Tanimoto coefficient more quickly (around 13,000 iterations) than the model without type (21,500 iterations).

Model	Training Loss	% Valid Molecules	Tanimoto Similarity
Vanilla Seq2Seq	1.148	86.00%	0.6355
Attention	0.1182	83.33%	0.8788
Attention (+ Type)	0.1249	86.67%	0.8760
Rule-Based System	N/A	100.00%	0.8079

Table 2: Training loss, percentage valid SMILES strings, and average Tanimoto Similarity (molecular similarity) all averaged across last 5 trials for smoothening. All models were trained for 22,000 iterations (around 10 epochs).

The attention weights were visualized both initially and after the Tanimoto Coefficient was greater than 0.8. As shown in Figure 4(b), the attention weights were initially distributed almost randomly across the input sequence. Figure 4(c) shows the attentions for the trained model on an example reaction in which the output was completely accurate.

The model learns to ignore the initial input molecule, which was actually a reagent (as none of its atoms appeared in the final product) and was formatted by mistake by the authors. The attention is highly local, matching the output to the input sequence almost perfectly for most atoms. The model ignores the Cl which will be removed in this reaction; it also ignores the reaction type (shown in blue). It was surprising that the reaction type was not very predictive of the products, and an area that should be investigated more (see Future Work).

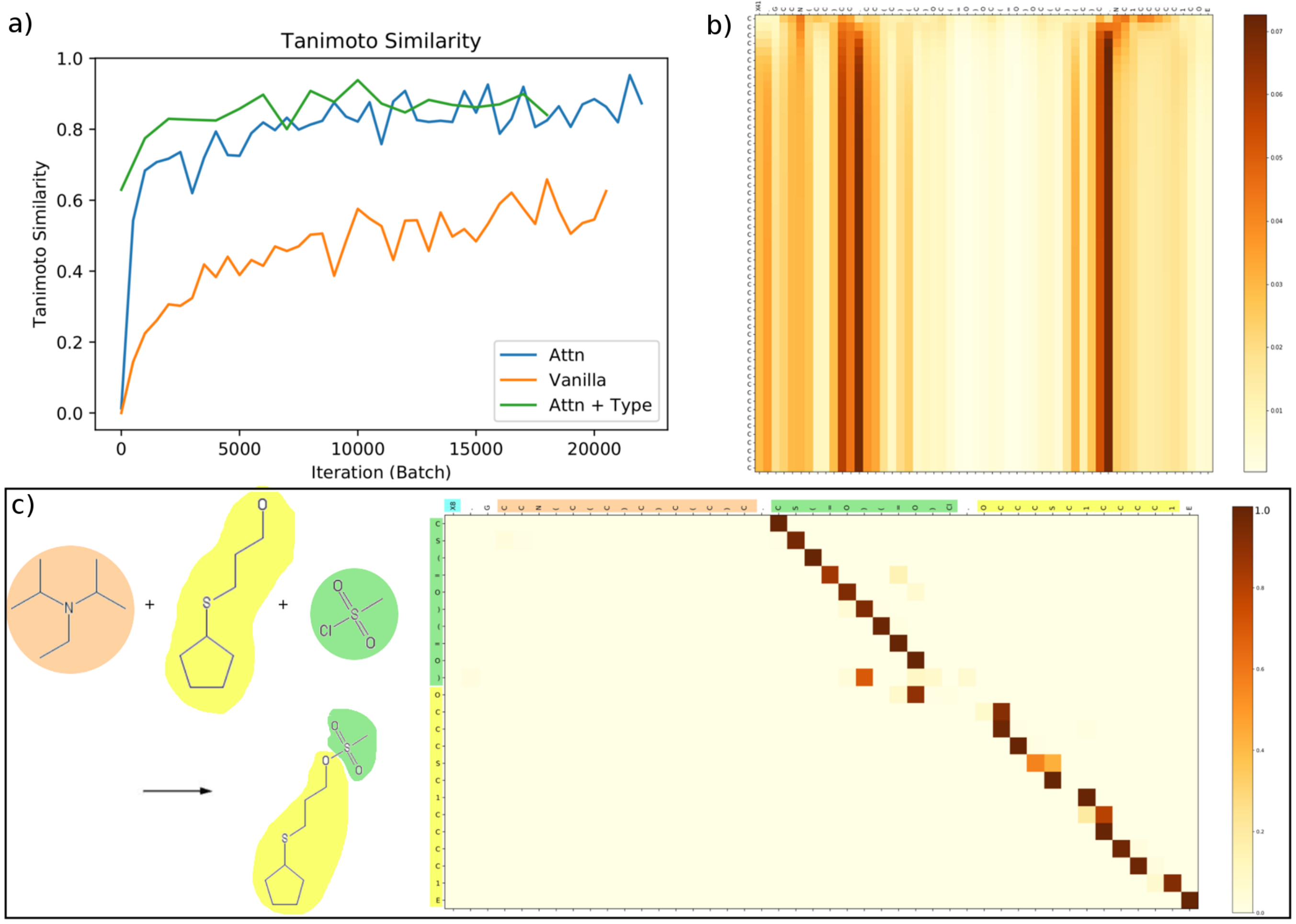


Figure 4: a) Plot of molecular similarity (Tanimoto Coefficient) measured as the models train. b) Initial attention distribution (mostly uniform). c)Reaction being predicted on left. Attention weights, with output tokens as rows and input tokens as columns. Output tokens colored by the input molecule they originate from.

4 Conclusion and Future Work

We have developed a completely data-driven system for predicting reaction type and reaction products, which outperforms the comparable rule-based extraction system. Moving away from rule-based extraction systems is beneficial as hard-coded rules have many exceptions and require a great deal of domain knowledge. In addition, our reaction type prediction significantly improves both upon our baseline and upon existing models that use reaction fingerprints to predict type; we reach this high accuracy using only the reactants, whereas existing systems use both reactants and products [8]. In product generation, the predicted products achieve a molecular similarity to the actual products of 0.96 at the peak. This system can thus be useful to synthetic chemists to accurately plan the synthesis of organic molecules.

In future work, we plan to further interpret the model to investigate whether the learned features align with human intuition. We have taken a step in that direction by visualizing the attention weights for accurate predictions; it was surprising that the reaction type was ignored by the decoder, as human intuition would take the reaction type highly into account. We plan to investigate further how best to include the reaction type, and other approaches to interpret what the model has learned.

In the reaction type prediction, it is worth noting that these 50 types of reactions were imposed by chemists somewhat arbitrarily; there are a number of exceptions. It would be interesting to employ autoencoders and clustering to see whether the different categories of reactions can be learned in an unsupervised manner, rather than imposing 50 human-created classes.

References

[1] Lowe, daniel. reactions from patents. https://bitbucket.org/dan2097/patent-reaction-extraction/downloads/. Accessed: 2017-10-18.

[2] Reaction smiles and smirks. http://www.daylight.com/meetings/summerschool01/course/basics/smirks.html. Accessed: 2017-10-18.

[3] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[4] Wengong Jin, Connor W. Coley, Regina Barzilay, and Tommi S. Jaakkola. Predicting organic reaction outcomes with weisfeiler-lehman network. *CoRR*, abs/1709.04555, 2017. URL http://arxiv.org/abs/1709.04555.

[5] Bowen Liu, Bharath Ramsundar, Prasad Kawthekar, Jade Shi, Joseph Gomes, Quang Luu Nguyen, Stephen Ho, Jack Sloane, Paul Wender, and Vijay S. Pande. Retrosynthetic reaction prediction using neural sequence-to-sequence models. *CoRR*, abs/1706.01643, 2017. URL http://arxiv.org/abs/1706.01643.

[6] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015. URL http://arxiv.org/abs/1508.04025.

[7] Christopher Olah. Understanding lstms.

[8] Nadine Schneider, Daniel M. Lowe, Roger A. Sayle, and Gregory A. Landrum. Development of a novel fingerprint for chemical reactions and its application to large-scale reaction classification and similarity. *Journal of Chemical Information and Modeling*, 55(1):39–53, 2015. doi: 10.1021/ci5006614. URL http://dx.doi.org/10.1021/ci5006614. PMID: 25541888.

[9] Philippe Schwaller, Theophile Gaudin, David Lanyi, Costas Bekas, and Teodoro Laino. "found in translation": Predicting outcomes of complex organic chemistry reactions using neural sequence-to-sequence models. *CoRR*, abs/1711.04810, 2017. URL http://arxiv.org/abs/1711.04810.

[10] Marwin H. S. Segler and Mark P. Waller. Neural-symbolic machine learning for retrosynthesis and reaction prediction. *Chemistry – A European Journal*, 23(25):5966–5971, 2017. ISSN 1521-3765. doi: 10.1002/chem.201605499. URL http://dx.doi.org/10.1002/chem.201605499.