Abstract

Grammar induction is the task of learning a grammar from a set of examples. Recently, neural networks have been shown to be powerful learning machines that can identify patterns in streams of data. In this work we investigate their effectiveness in inducing a regular grammar from data, without any assumptions about the grammar. We train a recurrent neural network to distinguish between strings that are in or outside a regular language, and utilize an algorithm for extracting the learned finite-state automaton. We apply this method to several regular languages and find unexpected results regarding the connections between the network’s states that may be regarded as evidence for generalization.

1 Introduction

Grammar induction is the task of learning a grammar from a set of examples, thus constructing a model that captures the patterns within the observed data. It plays an important role in scientific research of sequential phenomena, such as human language or genetics. We focus on the most basic level of grammars - regular grammars. That is, the set of all languages that can be decided by a Deterministic Finite Automaton (DFA).

Inducing regular grammars is an old and extensively studied problem (De la Higuera, 2010). However, most suggested methods involve prior assumptions about the grammar being learned. In this work, we aim to induce a grammar from examples that are in or outside a language, without any assumption on its structure.

Recently, neural networks were shown to be powerful learning models for identifying patterns in data, including in language-related tasks (Linzen et al., 2016; Kuncoro et al., 2016). This work investigates how good neural networks are at inducing a regular grammar from data. More specifically, we investigate whether RNNs, a neural network that specializes in processing sequential streams, can learn a DFA from data.

RNNs are suitable for this task since they resemble DFAs. At each time step the network has a current state, and given the next input symbol it produces the next state. Formally, let $s_t$ be the current state and $x_{t+1}$ the next input symbol, then the RNN computes the next state $s_{t+1} = \delta(s_t, x_{t+1})$, where $\delta$ is the function learned by the RNN. Consequently, $\delta$ is actually a transition function between states, similar to a DFA.

This analogy between RNNs and DFAs suggests a way to understand RNNs. It enables us to “open the black box” and analyze the network by converting it into the corresponding DFA and examining the learned language.

Inspired by that, we explore a method for grammar induction. Given a labeled dataset of strings that are in and outside a language, we wish to train a network to classify them. If the network succeeds, it must have learned the latent patterns underlying the data. This allows us to extract the states used by the network and reconstruct the grammar it had learned.

There is one major difference between the states of DFAs and RNNs. While the former are discrete and finite, the latter are continuous. In theory, this difference makes RNNs much more powerful than DFAs (Siegelmann and Sontag, 1995). However in practice, simple RNNs[1] are not strong enough to deal with languages beyond the regular domain (Gers and Schmidhuber, 2001).

It should be noted that similar ideas have already been investigated extensively in the 1990s

[1]Without the aid of additional memory such as in LSTMs (Hochreiter and Schmidhuber, 1997)
Given a regular language \( \mathcal{L} \) and a labeled dataset \( \{(X_i, y_i)\} \) of strings \( X_i \) with binary labels \( y_i = 1 \iff X_i \in \mathcal{L} \), the goal is to output a DFA \( A \) such that \( A \) accepts \( \mathcal{L} \), i.e., \( L(A) = \mathcal{L} \).

Since the target language \( \mathcal{L} \) is unknown, we relax this goal to \( A(X_i) = \hat{y}_i \). Namely, \( A \) accepts or rejects correctly on a test set \( \{(\hat{X}_i, \hat{y}_i)\} \) that has not been used for training. Accordingly, the accuracy of \( A \) is defined as the proportion of strings classified correctly by \( A \).

## 3 Method

The proposed method consists of the following three main steps:

1. **Data Generation** - creating a labeled dataset of positive and negative strings.
2. **Learning** - training an RNN to classify the dataset with high accuracy.
3. **DFA Construction** - extracting the states produced by the RNN, quantizing them and constructing a minimized DFA.

### 3.1 Data Generation

The following method was used to create a balanced dataset of positive and negative strings. Given a regular expression we randomly generate sequences out of it. As for the negative strings, two different methods were used. The first method was to generate random sequences of words from the vocabulary, such that their length distribution is identical to the positive ones. The other approach was to apply random transformations on the positive ones, such as word deletions, additions or movements. Both methods did not yield any significant difference in the results, thus we show only results for the first method.

### 3.2 Learning

We used the most basic architecture of RNNs, with one layer and cross entropy loss. In more detail, the RNN’s transition function is a single fully-connected layer given by

\[
\hat{y}_i = \sigma(A\sigma(Bs_n + c) + d).
\]

Prediction is made by another fully-connected layer which gets the RNN’s final state \( s_n \) as an input and returns a prediction \( \hat{y} \in [0, 1] \) by,

\[
\hat{y} = \sigma(A\sigma(Bs_n + c) + d).
\]

where \( \sigma \) stands for the sigmoid function, \( W, U, A, B \) are learned matrices and \( v, c, d \) are learned vectors.

The loss used for training is cross entropy,

\[
l = -\frac{1}{n} \sum_{i=1}^{n} y_i \log \hat{y}_i + (1 - y_i) \log (1 - \hat{y}_i),
\]

where \( y_1, \ldots, y_n \) are the true labels and \( \hat{y}_1, \ldots, \hat{y}_n \) are the network’s predictions. To optimize our loss, we employ the Adam algorithm (Kingma and Ba, 2014).

Our goal is to reach perfect accuracy on the validation set, in order to make sure the network succeeded in generalizing and inducing the regular grammar underlying the dataset. This assures that the DFA we extract later is reliable as much as possible.

### 3.3 DFA Construction

When training is finished, we extract the DFA learned by the network. This process consists of the following four steps.

**Collecting the states** First, we collect the RNN’s continuous state vectors by feeding the network with strings from the validation set and collecting the states it outputs while reading them.

**Quantization** After collecting the continuous states, we need to transform them into a discrete set of states. We can achieve this by simply using any conventional clustering method with the Euclidean distance measure. More specifically, we use the \( K \)-means clustering algorithm, where \( K \) is taken to be the minimal value such that the quantized graph’s classifications match the network’s ones with high rate. That is, for each \( K \) we build...
the quantized DFA (as we describe later) and count
the number of matches between the DFA’s classifi-
cations and the network’s over a validation set. We
return the minimal $K$ that exceeds 99% matches.

It should be noted that the initial state is left as is
and is not associated with any of the clusters.

**Building the DFA** Given the RNN’s transition
function $\delta$ and the clustering method $c$, we use the
following algorithm to build the DFA.

**Algorithm 1 DFA Construction**

```plaintext
V, E ← φ, φ
for each sequence $X_i$ do
    $s_{t-1}, v_{t-1} ← s_0, c(s_0)$
    for each symbol $x_{t+1} ∈ X_i$ do
        $s_t ← \delta(s_{t-1}, x_t)$
        $v_t ← c(s_t)$
        Add $v_t$ to $V$
        Add $(v_{t-1}, x_t) → v_t$ to $E$
    end for
    Mark $v_t$ as accept if $\hat{y}_i = 1$
end for
return $V, E$
```

Finally, we use the Myhill-Nerode algorithm
in order to find the minimal equivalent DFA
(Downey and Fellows, 2012).

### 4 Experiments

To demonstrate our method, we applied it on the
following few regular expressions.

**Simple Binary Regexes**

The resulting DFAs for the two binary regexes
$(01)^*$ and $(0|1)^* 100$ are shown in Figure 1.

![Figure 1: Minimized DFAs for binary regexes](image)

It can be observed that the method produced
perfect DFAs that accept exactly the given lan-
guages. The DFAs accuracy was indeed 100%.

A finding worth mentioning is the emergence
of cycles within the continuous states transitions.
That is, the RNN before quantization mapped new
states into the exact same state it has already seen
before. This is surprising because if we think of
the continuous states as random vectors, the prob-
ability to see an exact vector twice is zero. This
finding, which was reproduced for several regexes,
may be an evidence for generalization as we dis-
cuss later.

**Complex Regex**

To test our model on a more complicated grammar,
we created a synthetic regex inspired by natural
language. The regex we used describes a simpli-
ﬁed part-of-speech grammar,

$$\text{D}ET? \ A\text{DJ}∗\text{N}OUN\text{V}ERB\ (\text{D}ET? \ A\text{DJ}∗\text{N}OUN)?$$

The resulting DFA is shown in Figure 2.

![Figure 2: Minimized DFA for synthetic POS regex](image)

The DFA’s accuracy was 99.6%, i.e. the learned
language is not exactly the target one, but is very
close. For example, it accepts sentences like

**The nice boy kissed a beautiful lovely girl**

and rejects sentences like

**The boy nice**

However, by examining the DFA we can find sen-
tences that the network misclassifies. For exam-
ple, it accepts sentences like

**The the boy stands**

Inspecting the DFA’s errors might be meaning-
ful also for training, as a technique for targeted
data augmentation. By the pumping lemma, each
of the states where the network is wrong stands for an infinite class of sequences that end at the same state. In other words, those states are actually a "formula" for generating as much data as we want, such that the network is wrong. This way, the network can be re-trained on its own errors.

5 Discussion

Learning

The network reached 100% accuracy quickly on the synthetic datasets. This may indicate that deciding a regular language is a reasonable task for an RNN, and illustrates the similarity between RNNs and DFAs discussed earlier.

Emergence of cycles

The emergence of cycles within the continuous states, mentioned in experiment 4.1, might be understood as an evidence for generalization. Having learned those cycles means that for an infinite set of sequences the network will always traverse the same path and predict the same label. In other words, the model has generalized for sequences of arbitrary length before quantization.

It should be noted that such cycles have also been noticed in (Tino et al., 1998). Nevertheless, in their work the learning objective was predicting the next state rather than classifying the whole sentence. As a result, the emergence of cycles is expected, since the network was forced to learn them by the supervision. This differs from the case of classification, as learning the states and the cycles is not supervised.

Quantization

The process used for constructing the quantized DFA may introduce conflicts, if two different states in the same cluster lead to two different clusters for the same input. However, all of our experiments did not yield any conflict. This means that the clusters do reflect well the RNN’s different states.

This is reasonable due to the continuity of the RNN’s transition function, which maps "close" states into "close" states in terms of Euclidean distance. Thus, states within the same cluster will have similar transitions and therefore be mapped into the same cluster.

Another way to confirm the clusters validity is to check their compatibility with the network’s decisions, i.e., whether accepting or rejecting states are clustered together. Figure 3 presents the continuous vectors for the binary regex (0|1)*100.

Clearly, the states are divided into five distinct clusters and only one of them is accepting.

6 Conclusions

We investigated a method for grammar induction using recurrent neural networks. This method gives some insights about RNNs as a learning model, and raises several questions, for example regarding the cycles within the continuous states.

Quantization via clustering proved itself to reflect well the true states of the network. Identifying an infinite set of vectors as one state may reduce the network’s sensitivity to noise. Thus, states quantization during or after training may be considered as a technique for injecting some robustness to the model and reducing overfitting.

Finally, this method may serve as a tool for scientific research, by finding regular patterns within a real-world sequence. For example, it would be interesting to use this method for natural language, more specifically to induce grammar rules of phonology, which is claimed to be regular (Kaplan and Kay, 1994). Another example is to use it to find regularity within the structure of DNA, which is also regarded as regular (Gusfield, 1997).

References

[Cleeremans et al.1989] Axel Cleeremans, David Servan-Schreiber, and James L McClelland. 1989. Finite state automata and simple recurrent networks. Neural computation, 1(3):372–381.

[De la Higuera2010] Colin De la Higuera. 2010. Grammatical inference: learning automata and grammars. Cambridge University Press.

[Downey and Fellows2012] Rodney G Downey and Michael Ralph Fellows. 2012. Parameterized complexity. Springer Science & Business Media.

4To reduce the vectors dimensions we used PCA.
[Elman1991] Jeffrey L Elman. 1991. Distributed representations, simple recurrent networks, and grammatical structure. *Machine learning*, 7(2-3):195–225.

[Gers and Schmidhuber2001] Felix A Gers and Jürgen Schmidhuber. 2001. Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340.

[Giles et al.1990] C Lee Giles, Guo-Zheng Sun, Hsing-Hen Chen, Yee-Chun Lee, and Dong Chen. 1990. Higher order recurrent networks and grammatical inference. In *Advances in neural information processing systems*, pages 380–387.

[Gusfield1997] Dan Gusfield. 1997. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge university press.

[Hochreiter and Schmidhuber1997] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Kaplan and Kay1994] Ronald M Kaplan and Martin Kay. 1994. Regular models of phonological rule systems. *Computational linguistics*, 20(3):331–378.

[Kingma and Ba2014] D. Kingma and J. Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kuncoro et al.2016] Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A Smith. 2016. What do recurrent neural network grammars learn about syntax? *arXiv preprint arXiv:1611.05774*.

[Linzen et al.2016] Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of lstms to learn syntax-sensitive dependencies. *arXiv preprint arXiv:1611.01368*.

[Morris et al.1998] William C Morris, Garrison W Cottrell, and Jeffrey Elman. 1998. A connectionist simulation of the empirical acquisition of grammatical relations. In *International Workshop on Hybrid Neural Systems*, pages 175–193. Springer.

[Omlin and Giles1996] Christian W Omlin and C Lee Giles. 1996. Extraction of rules from discrete-time recurrent neural networks. *Neural networks*, 9(1):41–52.

[Schmidhuber2015] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117.

[Siegelmann and Sontag1995] Hava T Siegelmann and Eduardo D Sontag. 1995. On the computational power of neural nets. *Journal of computer and system sciences*, 50(1):132–150.

[Tino et al.1998] Peter Tino, Bill G Horne, and C Lee Giles. 1998. Finite state machines and recurrent neural networks–automata and dynamical systems approaches. Technical report.