Lesson 4

Interlude: Zero-reversible languages

4.1 What is the problem of learning?

Computational learning theory studies the problem of learning from a computational perspective. In computer science, algorithms are designed to solve problems. For example, there are algorithms that can be used to correctly multiply numbers, to correctly sort lists, to calculate pi to n digits, and to find the sequence of leaves of a tree structure (the 'yield'). Each of these algorithms solves a well-defined problem.

What is the well-defined problem of learning? Of language-learning? Let's begin by thinking about this in the context of Planet Verorez.

4.2 Planet Verorez

There are many nation-states on Planet Verorez and each has its own distinct official language. While people debate whether human languages on Planet Earth exhibit finitely or infinitely much variation, on Planet Verorez, the issue is settled. While there are only finitely many Verorezian languages spoken at any one time, there are infinitely many *possible* languages; the variation is infinite. Nonetheless on Planet Verorez, each typically developing Verorezian child learns the language of their nation-state relatively quickly from examples alone.

Verorezian scientists have studied the different Verorezian languages to understand how their children learn them so quickly from positive evidence. They discovered that all Verorezian languages obey a simple principle. Further, children could use this principle to generalize from example sentences to the language of their nation-state (whichever one it is). To put it another way, the principle lends itself naturally to a learning strategy. They have even discovered how to program a computer which can execute this learning strategy. Let's imagine further that scientists have conducted acquisition studies and experiments that provide additional evidence that children employ this learning strategy, or at least its general contours. What a world is Verorez!

4.3 The Universal Grammar of Verorezians

The fundamental principle of language on Planet Verorez:

If the initial portions of two sentences of a Verorezian language have a common ending then those initial portions the same set of endings!

Formally, consider a language $L \subseteq \Sigma^*$. The "tails" of a string u with respect to L is the set of all strings v that can "continue" u to obtain a string in L.

$$T_L(u) = \{v \mid uv \in L\}$$

What would it mean for two sentences to share a tail? It would mean that there are two strings $u_1v, u_2v \in L$. These two strings share the suffix v. It is a common ending. What the principle says then is that u_1 and u_2 share all tails with respect to L! Formally the fundamental principle can be expressed as follows.

$$\text{if } \exists v,\ u_1v, u_2v \in L \text{ then } T_L(u_1) = T_L(u_2)$$

Example

Suppose that the sentences "John laughed" and "John laughed and laughed" belong to L. Let's consider how to divide these two sentences into prefixes and suffixes so that they share a tail. Here is one way to do this.

prefix	tail
John	laughed
John laughed and	laughed

According to Verorezian UG, it will follow that if "smiled" is a tail of "John" then "smiled" will also be a tail of "John laughed and". In other words, if "John smiled" also belongs to L, then so will "John laughed and smiled".

Another way to divide the sentences is as follows.

prefix	tail
John laughed	λ
John laughed and laughed	λ

Here λ represents the empty string. In other words, these two sentences share the tail λ. It follows that any way we can extend one of these sentences to a string in L then we can extend the other one as well. In fact, we can see that "and laughed" is a tail of "John laughed". Since our two sentences share one tail (the empty string) they have to share all tails! It follows that "and laughed" is also a tail of "John laughed and laughed". In other words, the sentence "John laughed and laughed and laughed" also belongs to L. And since "John laughed and laughed and laughed" also has the tail λ it must also have the tail "and laughed". Therefore, "John laughed and laughed and laughed and laughed" also belongs to L. And so on.

In other words, from the *two* sentences "John laughed" and "John laughed and laughed", Verorezians can infer that infinitely many strings belong to their language!

4.4 An Inductive Principle

As we have seen, Verorezian UG leads directly to an inductive principle, which we can formalize as follows.

$$\text{For all strings } u, v, u', v', \text{ if } uv, u'v, uv' \in L \text{ then } u'v' \in L.$$

4.5 Finite-state Implementation

One of the most remarkable things about Verorezian UG and the inductive principle is that it has a concrete computational implementation.

4.5.1 Representing Verorezian Languages with DFA

First, every Verorezian language can be represented with finite-state acceptors (Angluin, 1982).

Theorem 1. *Every language on Planet Verorez can be recognized by a finite-state acceptor that is both forward and reverse deterministic.*

In fact, it is also the case that every forward and reverse deterministic finite-state acceptor recognizes a Verorezian language.

Theorem 2. *Every finite-state acceptor that is both forward and reverse deterministic recognizes a Verorezian language.*

Consequently, such automata characterize exactly the Verorezian languages.

4.5.2 Forward and Reverse Determinism

A finite-state acceptor is forward-deterministic if and only if it has only one initial state and for every state q and symbol a, there is at most one state reachable from q by a. A finite-state acceptor is reverse-deterministic if and only if it has only one final state and for every state q and symbol a, q can be reached by at most one transition labeled a.

4.5.3 Implementing the Inductive Principle

Next, the inductive principle can be implemented with operations over these finite state acceptors. This leads to an algorithm which learns Verorezian grammars from positive examples. There are two main operations which this algorithm uses, which I will call "addition" and "merging". The algorithm can be succinctly stated as follows.

1. The initial grammar is an acceptor A with a single non-accepting state with no transitions.
2. When a sentence $s \in L$ is heard, add s to A.
3. Recursively merge states in A until it is both forward and reverse deterministic.
4. Repeat from step 2.

Let's call this VZIA for VeroreZ Inference Algorithm.

4.5.4 Adding Strings to Acceptors

Given a minimal, trim DFA A, a string w, we are interested in constructing a machine that accepts $L(A) \cup \{w\}$. If $w \in L(A)$ then it should just return A. While we could use the classic operations of union (and determinization) to achieve this, there is a more efficient way.

The idea is to submit w to A and trace its path as far as it can go. If A already accepts w, then we will reaches a final state and we are done. No changes are made to A. However, if $w = uv$ and u is the maximal prefix of u that we can trace in A, then we need to construct a new branch recognizing v from the state that u brings us to.

Formally, for any minimal, trim DFA $A = \textsc{add}(A, w)$ equals $A' = \langle \Sigma', Q', q'_0, F', \delta' \rangle$ where

$$\begin{aligned} \Sigma' &= \Sigma \cup \Sigma(w) \\ Q' &= Q \cup V \\ q'_0 &= q_0 \\ F' &= F \cup \{v\} \\ \delta'(q,a) &= \delta(q,a) \qquad (\forall q \in Q, a \in \Sigma) \\ \delta'(q', v_1) &= v_1 \\ \delta'(q,a) &= qa \qquad (\forall q, qa \in V) \end{aligned}$$

where

- $w = uv$,
- for all nonempty v, v_1 is the first symbol of v,
- $\delta^*(q_0, u) = q' \in Q$,
- if v is nonempty then $\delta^*(q_0, uv_1)$ is undefined, and
- $V = \textsc{Prefixes}(v) \backslash \{\lambda\}$.

If we begin with an empty acceptor then applying the above procedure to a finite sample of strings S yields the prefix tree representation of S. Each state in the tree corresponds to a unique prefix in the sample.

Formally, $PT(S)$ is defined to be the NFA $\langle \Sigma, Q, I, F, \delta \rangle$ such that

$$\begin{aligned} \Sigma &= \Sigma(S) \\ Q &= \textsc{Prefixes}(S) \\ I &= \{\lambda\} \\ F &= S \\ \delta(q,a) &= qa \text{ iff } q, qa \in Q \end{aligned}$$

4.5.5 State Merging

As Heinz *et al.* (2015) explain, when distinct states in a finite-state machine are merged, they become a single state. A key concept in state-merging is that transitions are preserved. This is one way in which generalizations may occur—because the post-merged machine accepts everything the pre-merged machine accepts, possibly more.

Formally, let $A = \langle \Sigma, Q, I, F, \delta \rangle$ be any NFA. Consider any partition π of Q, and let $B(q, \pi)$ refer to the set of states in the same block of the partition as state q. Then merging states in the same blocks of A according to π yields another acceptor $A/\pi = \langle \Sigma', Q', I', F', \delta' \rangle$ defined as follows:

$$\begin{aligned} \Sigma' &= \Sigma \\ Q' &= \{B : B(q,\pi) \text{ such that } q \in Q\} \\ I' &= \{B : B(q,\pi) \text{ such that } q \in I\} \\ F' &= \{B : B(q,\pi) \text{ such that } q \in F\} \\ \delta'(B_0(q_0,\pi), a) &= \{B_1(q_1,\pi) : q_1 \in \delta(q_0, a)\} \end{aligned}$$

A/π is sometimes called the *quotient* of A and π. Notice that any block containing at least one final (initial) state is itself a final (initial) state in the new machine. Similarly, if there is at least one transition labeled a from any state in block B_0 to another state in block B_1 then in the new machine, there is a transition from B_0 to B_1 labeled a.

4.5.6 Theoretical Results

There are several interesting things that can be shown about VZIA (Angluin, 1982). First, we can see from its definition that VZIA is incremental. This means it can update each grammar with each observed sentence.

Second, VZIA is efficient in the size of the sample.

Theorem 3. *VZIA can be implemented to run in time $O(n\alpha(n))$, where $n = \|S\| + 1$ and α is a very slowly growing function (Tarjan (1975) defines α.)*

Third, *every* finite sample corresponds to a particular Verorezian language.

Theorem 4. *For any finite set of strings S, VZIA(S) returns a DFA which represents the smallest Verorezian language containing S.*

Finally, the behavior of the algorithm "in the limit" guarantees that any Verozian language L can be learned from any sequence of examples – which meet particular conditions – drawn from L.

Theorem 5. *VZIA identifies every Verorezian language in the limit from positive data.*

The last means VZIA solves a particular kind of learning problem. We will study this in subsequent days.

4.6 Some things to think about

4.6.1 Is this really induction?

It could be said that once the basic principle is known, the learning problem is no longer a problem of induction. It has become a problem of deduction. We will study the concepts induction, deduction, and abduction.

4.6.2 How is Planet Verorez like Planet Earth?

Verorezian UG is not like human UG. It is not too difficult to think of examples in human languages where proper application of the inductive principle leads to ungrammatical sentences.

But the above is instructive because it is a concrete example of what a learning algorithm can look like that can be said to successfully learn a nontrivial class of languages.

I am particularly taken by the fact that given any lump of clay (any set of strings), VZIA transforms it into a grammar that obeys a basic principle. I think human children are like this. Whatever their linguistic environment they construct something out of it that obeys principles of human language. It is like a flower that grows and takes shape as it is fed water and sunlight.

4.7 Updates since 1982

Clark and Eyraud (2007) study the substitutable languages. Here they are

$$\text{if } xuy, xvy \in L \text{ then } C_L(u) = C_L(v)$$

where the $C_L(u)$ is defined as the *contexts* that u can occur in.

$$C_L(u) = \{(x, y) : xuy \in L\}$$

The fundamental principle of language on Planet Substitutable:

If two strings share one context then they share all contexts!

This paper launched a series of papers on "distributional learning" where the distributions of strings (as determined by their contexts) is used to make syntactic generalizations. Overviews are provided in Clark and Yoshinaka (2016) and Chater *et al.* (2015, chap. 4).

Bibliography

Angluin, Dana. 1982. Inference of reversible languages. *Journal for the Association of Computing Machinery* 29:741–765.

Chater, Nick, Alexander Clark, John A. Goldsmith, and Amy Perfors. 2015. *Empiricism and Language Learnability*. Oxford University Press.

Clark, Alexander, and Rémi Eyraud. 2007. Polynomial identification in the limit of substitutable context-free languages. *Journal of Machine Learning Research* 8:1725–1745.

Clark, Alexander, and Ryo Yoshinaka. 2016. *Distributional Learning of Context-Free and Multiple Context-Free Grammars*, 143–172. Berlin, Heidelberg: Springer Berlin Heidelberg.

Heinz, Jeffrey, Colin de la Higuera, and Menno van Zaanen. 2015. *Grammatical Inference for Computational Linguistics*. Synthesis Lectures on Human Language Technologies. Morgan and Claypool.

Tarjan, Robert Endre. 1975. Efficiency of a good but not linear set union algorithm. *Journal of the ACM* 22:215–225. Https://doi.org/10.1145/321879.321884.