Build Machine Teaching Model in Random Graph

Yuqing Zhu
zhu298@wisc.edu

Abstract

Scratch [3] is a project aims to help children to learn simple programming: children can choose a sequence of *instructions* from a given *instruction set* to build their own programs. Our purpose is to build a random graph-based [4] teaching model as guide to find *optimal teaching strategy* to teach children to build program given a specified *learning target*. In this paper, we model children' learning process as a machine teaching problem [5] with random graph under the assumption that *teachers* know the learning goal and wants to design *optimal* training *programs* for the learners - *children*. Our framework first model children' learning process as a random graph and then apply machine teaching theory to *balance* the effort of the teacher and the loss of learner.

1 Introduction

Scratch provides children with a given set of *movements* (or instructions) which could be used to build simple programs. Considering a simple teaching task the purpose of which is to teach children to build a specified *movement sequences* (essentially a program) from the given set of movements. In order to model the task as a machine teaching problem, we first specifically define our framework. Our framework consists of three entities: the world, the learner, and the teacher. (i) **world** is defined by a target model θ^* and here it is a sequence of movements assigned by the teacher. (ii) **learner** has to learn θ^* from the training programs. And training programs here are provided by teacher. (iii) **teacher** knows the world θ^*, the learner's hypothesis space Θ and how the learner learns given any training programs. The teacher's goal is to design a teaching set $\mathcal{D}$ so that the learner learns θ^* as accurately as possible. Now, we can propose a generic optimization problem for optimal teaching:

$$\min_{\mathcal{D}} \texttt{loss}(\widehat{f_{\mathcal{D}}}, \theta^*) + \texttt{effort}(\mathcal{D})$$

The function `loss`() measures the learner's deviation from the desired θ^*. The quantity $\widehat{f_{\mathcal{D}}}$ represents the state of the learner after seeing the teaching set $\mathcal{D}$

. The function `effort` measures the difficulty the teacher experiences when teaching with $\mathcal{D}$. In order to simplify this combinatorial problem, we add a restriction of $loss()$,And redefine our target:

$$\min_{\mathcal{D}} \texttt{effort}(\mathcal{D})$$

$$\texttt{loss}(\widehat{f_{\mathcal{D}}}) \leq \epsilon$$

In our graph model, learner start at an initial empty vertex(root), we want to help them find a path to the *target vertex* (the sequence specified by us). The reachability without seeing teaching set $\mathcal{D}$ is denoted as $\widehat{f}$. Our main work focus on computing $\widehat{f}$,and we will discuss it later.

2 The Loss and Effort Function with Graph

The choice of `loss`()and `effort`() is problem-specific and depends on the teaching goal θ^*. In this paper , learner has a reachability probability from the initial vertex to our target sequence in the graph model(we will discuss the details of graph model later), so $\texttt{loss}(\widehat{f}, \theta^*) = \theta^* - \widehat{f}$, where $\widehat{f}$ is a function of probability P, P is a $|M|$ size of vector,p_i represents the initial learning ability of i^{th} movement for learner. The `effort`() function reflects resource constraints on the teacher and the learner: how hard is it to create the teaching examples, to deliver them to the learner, and to have learner absorb them. For most of the paper we use the cardinality of the teaching set $\texttt{effort}(\mathcal{D}) = c\mathcal{D}$ [2] where c is a positive per-item cost.In this paper, we focus on how to model the learner's learning process as a graph problem instead of teaching process. And implement a simple method called **Block** to computing the teaching effort.

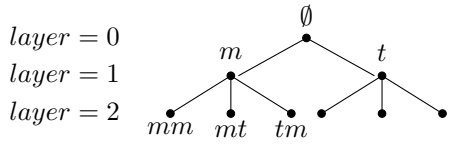
2.1 Notation of graph

Assume that we have a movement set $\mathbf{M}$ and $\mathbf{M}$ contains k movements representing as $m_1, m_2...m_k$. For example $\mathbf{M}$={m,t,...} (m represents *moving one step forward* and $t$represents *making a turn*) and we will need to use these movements to construct our graph model $G(V, E)$ in which V represents the list of vertices, and E represents the list of edges of the graph G. $|V|$ represents the number of vertices and $|E|$ represents the number of edges respectively. Assume there is a *root* vertex r on the graph as the *start point*. Every vertex v is an ordered list which contains the achievable sequence of movements from the root vertex r to v. We use $C(v)$ to denote the size of vertex v which refers to the number of movements in the vertex v (e.g., $v = \{m_1, m_2, ..., m_j\}, C(v) = j$). The root vertex r is an empty list since children have not started to build the sequence yet before making the first movement. When a child *stands at* a vertex, he has k way to choose the next vertex to move to. For example, he can choose to move

one step forward (m) or make a turn (t) (in which case $k = 2$ and $\mathbf{M} = \{m, t\}$). For convenience of graph description, we introduce *layer* notation in the graph. The initial vertex is at the 0_{th} layer. It is intuitive that the 1_{st} layer of graph G contains k vertices while every vertex of the 1_{st} layer is a set containing only one movement. The edge $e(v, \bar{v})$ means $\bar{v}$ is built by adding a movement to v. Since every vertex v is an ordered list, when we want to add a certain movement in this list, there could be $C(v) + 1$ possible different positions of the original ordered list to *insert* the movement . The vertices that have the same *parents* and contain the *same sequence* are merged into one and thus the graph is *not* structured as *tree*. And there is also an alternative graph model that merges all the vertices containing the same sequence at the same layer to a single vertex. We will discuss about this alternative later.

Example 1 (Simple graph model)

We construct a graph $G(V, E)$ with the movement set M containing only two movements m and t ($M = \{m, t\}, k = 2$). Our target is to train children to learn a specific sequence of movements, and every vertex v in G is a *sequence of movements*. When we want to insert $m(m_{new})$ in a single movement $m(m_{old})$, there are two choices: Insert m_{new} in front of the m_{old} movement or insert m_{new} after m_{old}. We merge them as a single node representing the sequence mm instead of showing two mm separately.



Definition 1(Reachability without$\mathcal{D}$)

Reachability $\widehat{f}$ represents the probability from the initial empty vertex to our target sequence in the graph model, and can be computed as a function of learning probability P. As we mention before, there is a *learning probability* for children to learn a specific movement, which means that if children decide to make a movement (t or m), there is a specific probability related to the movement: certain probability to make it, or fail. So the graph shown before is actually under an ideal assumption that every movement has been made correctly (each movement has been assumed to be made with successful probability 1). Thus, for these failed movements , we didn't shown them on the example graph (figure 1). Recall the definition of *learning probability*,for any $m_i \in \mathbf{S}$, $0 \leq p(m_i) \leq 1$. We have a strong assumption here[1]. for any $i \neq j, 1 \leq i, j \leq K$.and $p(m_i)$ is independent of $p(m_j)$ since we assume that children's learning ability of every movement is independent.Also, we assume that for every step learner makes a step independently, which indicates that for every achievable path, the probability of every edge in that path is independent.

[1]This is a strong assumption. It can be relaxed in future work, where we have to estimate the dependence between different movement's learning probability for learner

Example 2(Achievable path) With a given certain sequence (m, t, m), search for achievable path, and calculate the probability that children can build a sequence from the initial vertex to the destination vertices which contain (m, t, m).In figure 2, there are 5 achievable paths which starts from the initial vertex and ends at certain vertices which all contain (m, t, m) sequence. To make the graph looks clear, we only remain the target node with dash in 3^{th}layer. Now, we introduce a method called **Backward computation** to calculate the reachable probability from the initial node to our target node .

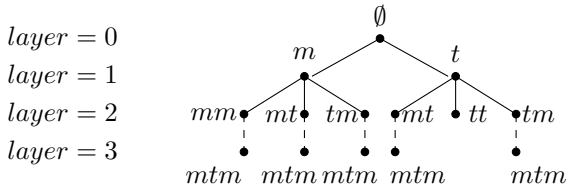


figure 2

(Every dash corresponding to an end of a achievable path)

3 Backward computation

The motivation of Backward computation is that every path starts from the initial node to our target node is overlapped some where(they may share some nodes). Since the $\widehat{f}$ can be computed as the joint probability of all path from initial node to target nodes, and it's difficult to compute the joint probability of those paths. However, if our path starts from the target node to initial node: Every path is independent (no shared nodes) or they will join each other at some nodes and combined as a same path. Also, the achievable probability from the initial node to target is the same with the achievable probability from target node to initial node. That's the basic of our *Backward Computation*.

Backward computation implies that we calculate the probability from the leave of the tree instead of the root. It's decomposed of three main steps *locate target nodes*,*probability production*,*merge probability*.Our path from leave to root is a sequence of steps chosen from those three kind of steps. In order to make it explicit to be understood, we still consider the graph in Example 2.Our target is to compute the probability from initial node to our target nodes which contain the sequence of (mtm).For here, we first consider a simple version $p(m) = p(t) = p$, which indicates the remaining probability for every edge in our graph is p.

Definition 2(Locate target nodes)

We first locate all nodes that satisfying the goal (all nodes representing the same sequence mtm), and given them a p_b probability(p_b denotes the achievable probability from back). It's intuitively that p_b for those node are 1. Different target node represents the different start node of a backward path start from

the leave node and ends at $\emptyset$ node . In figure 3 target nodes have been located and denoted red.

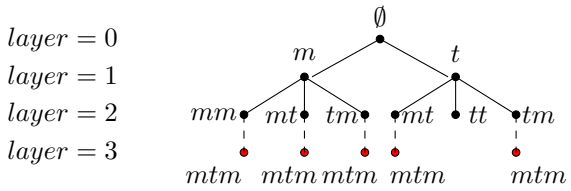


figure 3
(Red nodes are our target nodes)

Definition 3(Probability production)

Probability production and Merge probability are all methods to compute the achievable probability from back for i^{th} node for different situations. Probability production can be applied to a node i when there doesn't exist two or more paths *crash* on node i. We use j to denote i's offspring,(i, j) is an edge in our graph. While we have computed the p_b of j node, we can implement an induction method to compute $p_b(i)$. Since every edge shares the same remaining probability p in every path from leave to initial node. The reachable probability from back of p_i is

$$p_b(i) = p_b(j) \times p$$

Definition 4(Merge probability)

For another situation: once one branch joins another(or more) edge at node i, we need to calculate the joint reachability probability from back of node i. In our figure 4, considering the m node(denoted red),since those three branches *1,2,3* labeled in figure 4 are independent(Because of the backward method), and $p_b(mm) = p_b(mt) = p_b(tm) = p$, so $p_b(m) = 1 - (1 - p)^2$.

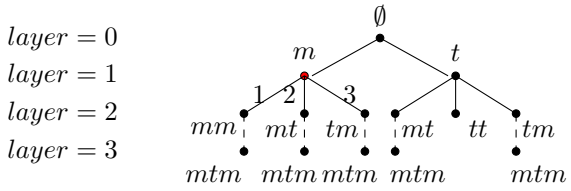


figure 4
(Do merge instruction on red node m)

More generally, suppose there exist r branches join at node i, the j^{th} offspring of i can be denoted as $\texttt{offspring}_i(j), j = 1, ..., r$. the reachable probability from back of i can be calculated as following:

$$p_b(i) = 1 - \prod_{j=1}^{r} (1 - p_b(\texttt{offspring}_i(j)))$$

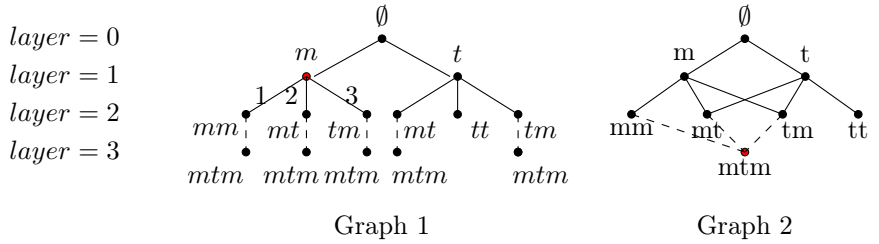
For a simple situation, the remain probability for every edge in the graph is still p.So the graph is under a random process, *the forgetting probability for every edge is* $1-p$.

4 Relaxation

Recall the process of computing an achievable probability for a learner with backward method, we have done a lot of duplicate work.[2] Relaxation here means we apply a *compact* graph to represents the graph defined before. We introduce this notation in order to make graph look compact(which maintain the small number of edges and node, save time and space for double counting).

Example(Relaxation)

Graph 2 is a relax version of Graph 1. The graph under random process(remaining probability for every edge in the graph) is still Graph 1. In other words, Graph 2 is a *compact* graph representation of (Graph 1) which maintains the small number of edges and nodes in the actual graph (Graph 1). We will discuss how it work for reducing duplicate computation later.



Graph 1 Graph 2

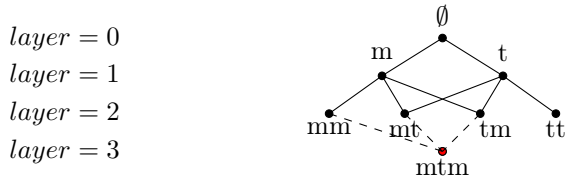
Definition 5 New Backward Computation

To make the new graph model adapted for computing the achievable probability ,we need to adjust our *backward computation* process. For one thing, it's intuitively that there isn't any duplicate sequence in every layer, so the *locate target nodes* step can be moved for we only have one target node.

Definition 6(New Probability Production) The astute reader may notice that duplicate computation occurs on *Probability Production* process. Look at Graph 1 in Example(Relaxation), we have done the same probability production instruction for 3 time in edge 1,2,3. The motivation of *Relaxation* comes from here. We will use an example here to demonstrate how it work.

Example(Relaxation Example)

[2]From figure 4, we have done *probability production* for 3 time in edge 1,2,3. And we can implement *Relaxation* instruction to reduce computation work to one time.



Graph 3

Target *mtm* has three parents, all the edges from *mtm* to its parents share the same probability. Thus the p_b (achievable probability from back)for *mtm*'s parents are same. $p_b(par(mtm) = p_b(mtm) \times p)$ (It only applies for the situation when the remaining probability for every edge in the graph is same. For other case, we will discuss it later).

Theorem 1 Whenever one node x has two or more parents, them all the paths from x to its parents share the same probability. Thus it saves time and space for double counting.

Conclusion *Merge Probability* is same as we defined in *Definition 4*. Relaxation instruction plays an important role in avoiding double counting in probability production step. However, what we must pay attention to is that when we implement relaxation method building the compact graph with less nodes and less edge , it doesn't change the graph itself. (the graph under random process is still the graph without relaxation shown on graph)

5 Teaching effort

In this paper, we focus on how to model learner's learning process as random graph problem, and implement a simple method **Block** to measure the teaching effort.

Definition 7(Block)

Recall that we use a limitation $\texttt{loss}(\widehat{f_{\mathcal{D}}}) \leq \epsilon$ to limit the $\texttt{loss}(\widehat{f_{\mathcal{D}}}, \theta^*)$. So the teaching target here is to improve learner's performance $\widehat{f}$(the achievable probability) in order to reach the requirement for $\widehat{f_{\mathcal{D}}}.\widehat{f}$ can be computed by the *Backward* methods we provide before. And it's easy to find that $\widehat{f}$ is a polynomial function of learner's initial learning probability P. Considering a simple teaching case: Every time teacher show the learner a block of movement $i, i \in \{1, ..., |M|\}$, learner's learning probability of i^{th} movement will be increase l_i percent. l is a $|M|$ dimension vector represents the teaching effect for $|M|$ kind of movements. Readers may point out that it's a very tricky method to teach learners, and may not make scene when applied to real life. Our purpose for applying this *Block* instruction to is to suggest a a feasible method to measure teaching effort $\mathcal{D}$. Since we use the cardinality of the teaching set $\texttt{effort}(\mathcal{D}) = c\mathcal{D}$ to measure the

teaching effort $\mathcal{D}$, and the cardinality here is the number of Block we use. Thus the teaching effort can be computed by the number of Block under our simple assumption of teaching process.The refinement of teaching process will be done in our future work.

Simple Case

For a simple case where the learner's initial learning probability for every movement in M is same. So $\widehat{f}$ is a monotone increasing function with variable p. Suppose p_D represents the learner's learning probability after the training data provided by teacher. The requirement for loss is

$$\widehat{f_{\mathcal{D}}}(p_{\mathcal{D}}) \geq \theta^* - \epsilon$$

$$p_{\mathcal{D}} = (1 + l_p)^{|\mathcal{D}|}$$

Now we want to first compute the critical value of $p_{\mathcal{D}}$, and then use $p_{\mathcal{D}}$ to compute the size of our training data $\mathcal{D}$. Since $p_{\mathcal{D}\leq 1}$, suppose the critical precision between $p_{\mathcal{D}}$ and $\widehat{p_{\mathcal{D}}}$ is δ, $\widehat{p_{\mathcal{D}}}$ represents the probability we are experimenting in our experiment. With the help of **Bisection method** , we can solve this optimization method in $\log(\frac{1}{\delta})$ times experiment.

Muli-p Case We will discuss the *Multi-p* part (Where $p_i \in P$ is different from each other) as our future work.

6 Different graph model

After reviewing the definition of our graph model A in *Definition 1*, you will find that we have define another kind of graph model B*merge all the vertice containing the same sequence in the same layer to a single vertex*. The c[onstruction of those two graph are totally different, though the relaxation version of graph A may look the same as graph B,but the graph under random process is different. As we mention before, graph model represents the children' making decision process in *Scratch* project. Different graph model represents the different probability assumption of how do children make a movement. In graph model B, we assume there aren't any duplicate sequences in every layer. Because of the limitation of domain knowledge related to psychology currently , we have no idea to decide which graph is more fit for children' decision process. We want to refine it later. However, what I want to point out is that , even with the same movement set S and the same probability assumption *the remaining probability for every edge is same*, The reachable probability from the root to our target sequence is still different.

Principle of Inclusion-Exclusion

We can apply *Principle of inclusion-Exclusion*[1] to compute the reachability of graph model B. (We can also apply *Principle of inclusion-Exclusion* to compute the achievable probability in Graph A. However the time complexity

is much higher compared with *Backward method*. That's because *Principle of inclusion-Exclusion* need to traversal all the achievable path from initial node to targets. The time complexity could be exponential.

Compare two methods

As we mention before, the time complexity of *Principle of inclusion-Exclusion* method could be exponential ,and the time complexity of *Backward Computation* can be reduced to $O(G(|V|, |E|)$. Unfortunately, the *Backward Computation* method can not be applied for Graph B, because the path started from the target node to the initial node is still dependent, we may implement *junction tree* [**?**]method to solve this problem.

Future work

For future work, we will discuss the optimization process for multi-probability, and try to find a efficient way to compute the achievable probability in Graph Model B (defined in *different model*) part.

References

[1] Y. Chen. *Introduction to probability theory*. The lecture notes on information theory. Duisburg-Essen University, 2010.

[2] S. A. Goldman and M. J. Kearns. On the complexity of teaching. *Journal of Computer and System Sciences*, 50(1):20–31, 1995.

[3] D. Kahle, T. Savitsky, S. Schnelle, and V. Cevher. Junction tree algorithm. *STAT*, 631, 2008.

[4] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.

[5] D. B. West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

[6] X. Zhu. Machine teaching for bayesian learners in the exponential family. In *Advances in Neural Information Processing Systems*, pages 1905–1913, 2013.