



基于神经网络图灵机的算法学习的研究

王李荣 高志强

(东南大学计算机科学与工程学院, 南京, 210096)

摘要 如何通过输入输出样例自动学习算法一直是人工智能的一个核心问题。近年来, 机器学习模型被许多研究者应用于该领域, 神经网络图灵机是其中之一, 该模型能学习的复制, 重复复制等简单算法。本文使用序列到序列的框架训练神经网络图灵机, 使该模型学习加法, 乘法等复杂算法。实验结果表明神经网络图灵机能学习到这些算法, 但是需要使用课程学习训练模型, 而且使用传统的课程学习策略效率不高, 因此本文设计了一种新的课程学习策略, 加快了模型的收敛速度。

关键词 神经网络图灵机; 算法学习; 课程学习; LSTM

Learning Algorithms with Neural Turing Machine

Lirong Wang Zhiqiang Gao

(Department of Computer Science, Southeast University, Nanjing, 210096)

Abstract: How to learn algorithm from its input-output examples is a core problem in artificial intelligence. Recently machine learning methods are emerging for learning an algorithm, the most famous being neural turing machine, capable of learning copy and repeat copy. We show that neural turing machine trained with sequence-to-sequence framework can learn algorithms for problems like add, multiplication from pure input-output examples. Notably, it is necessary to use curriculum learning, and while conventional curriculum learning proved ineffective, we design a new variant of curriculum learning that improved our network's performance in all experimental conditions.

Key words: Neural Turing Machine; Algorithms Learning; Curriculum Learning; LSTM

§1 引言

计算机执行一个程序, 需要处理数值计算, *if* 表达式, 赋值等基本操作以及这一系列基本操作的组合。Zareba等人^[1]将这些基本操作称为算法, 例如复制,

加法, 乘法都被称为算法。对于冯诺依曼体系结构的计算机来说, 程序员通过编码操作CPU和寄存器可以很轻松的完成算法任务。但是如何让计算机通过输出输出样例自动学会这些算法, 一直是人工智能的一个核心问题, 该任务被称为算法学习^[1]。



实际上, 算法学习不是一个新任务, 它也被称为程序推理 (Program induction), 可以追溯到1967年^[2, 3], 是一个非常古老但却十分有意义的研究领域。这项任务有着丰富的应用场景, 例如: 帮助没有计算机背景的人编写程序, 帮助程序员发现问题, 发现新算法, 帮助教学。因而在过去几十年, 有许多研究者投入到该任务的研究中^[4, 5, 6, 7], 并做出了卓越的贡献。

在近些年, 有研究者利用机器学习模型, 尤其是循环神经网络 (Recurrent Neural Network, RNN), 研究该问题^[8, 9, 10, 11, 12]。在这些工作之中, 与本文最相关的模型是神经网络图灵机 (Neural Turing Machine, NTM)^[8]。NTM是外部记忆增强的循环神经网络 (Memory augment Recurrent Neural Network)。该网络结构与长短时记忆网路 (Long Short Term Memory, LSTM)^[13]有相似之处。提出这两种结构目的之一是利用其记忆单元解决任务中存在的长距离依赖问题, 不同的是LSTM中, 增加记忆单元的个数会导致模型参数个数的增加, 从而使得模型难以训练。而NTM这类外部记忆增强的循环神经网络为了解决该问题, 将记忆模块从网络中分离出来, 将网络分割为可训练的控制模块和外部记忆模块两个部分, 使得外部记忆模块的大小将不再依赖于网络参数的个数。根据Graves等人^[8]的实验结果, NTM能学习到复制, 重复复制等简单算法。

本文尝试使用NTM学习加法, 乘法等复杂算法时, 发现模型收敛速度慢, 甚至可能会陷入局部极值点, 使得模型在测试集上的准确率无法提升。所以本文尝试使用Bengio^[14]提出的课程学习策略简化问题, 将任务分为多个课程, 逐步增加课程的难度, 让模型从易到难的完成整个任务的学习。然而, 将Bengio提出的课程学习策略应用本文的任务时, 发现该策略未考虑到两点:

1. 利用课程学习训练模型时, 不仅要求模型掌握难度大的课程任务, 而且中间课程的任务也不能遗忘。假设模型的目标是掌握20位数的加法, 那么模型不仅要掌握20位的加法如何运算, 还需要掌握一位数, 两位数的加法如何运算。但是使用普通的课程学习

策略后, 发现模型掌握了复杂的课程任务后 (20位数的加法), 就遗忘了简单的课程任务 (1位数, 两位数的加法)。因此, 在学习新课程时, 需要复习旧课程。

2. 算法任务的复杂度是呈现指数式增长的。一位数的加法总共有将近100种可能性 (第一个加数10种可能, 第二个加数10种可能), 二位数的加法就有10000种可能性, ...。因此, 在学习新课程时, 需要适当的平衡新旧课程的比例, 甚至根据学习速率, 自适应的调整该比例。

基于以上两点, 本文设计了一种基于课程比例的课程学习策略, 提升了模型的性能。本文将在实验部分详细的介绍该策略, 以及其他几种课程学习策略。

综上, 本文的主要工作是:

1. 实现NTM, 并将该模型应用于复杂的算法学习任务, 包括加法, 乘法等任务。
2. 研究课程学习对模型性能的影响。设计了一种新的课程学习的策略, 使用该策略使NTM学到加法, 乘法的算法, 并提升了模型性能。

§2 相关工作

算法学习这个任务最早可以追溯到1967年^[2, 3]。在其漫长的研究过程中, 根据模型不同或者方法不同, 该任务有许多不同的名称, 例如: 合成程序 (program synthesis)、程序归纳 (program induction)、自动编程 (automatic programming) 等。但是其研究目标始终是让计算机有理解程序甚至自动编写程序的能力。Gulwani^[15]和Kitzelmann^[16]从多个角度描述了算法学习的研究现状。

在该任务的诸多模型方法中, 本文重点关注机器学习模型在算法学习任务中的应用, 尤其是Graves^[8]提出的NTM。这是第一次明确提出外部记忆增强的神经网络模型。根据这篇文章的实验结果, NTM能学习到复制,

重复复制等简单算法。

之后陆续提出的模型，都是基于NTM进行改进或者参考了NTM的结构。例如：为了解决NTM不能并行和训练难度大等问题，Kaiser^[10]借鉴了Grid LSTM^[17]的思想，提出了一种高度并行的神经网络结构-神经网络GPU（Neural GPUs, N-GPUs）；Zaremba^[9]结合增强学习和NTM，提出了增强学习神经网络图灵机（Reinforcement Learning neural turing machine, RL-NTM）；Graves^[18]提出的可微神经计算机（Differentiable Neural Computer, DNC），该模型对NTM做了三点改进。其余类似的结构还有记忆神经网络（Memory neural network, MNN）^[19]、指针网络（Pointer Network, Ptr-Nets）^[20]、栈增强的循环神经网络（Stack-Argmented Recurrent Neural Network）^[21]、层次记忆网络（Hierarchical Attentive Memory, HAM）^[12]。

这些模型从并行性、外部记忆结构、寻址模式、寻址效率等方面对NTM做出改进或者提出了类似于NTM的外部记忆增强的循环神经网络结构。虽然这些模型在某些特定方向的性能得到了提升，但是模型的复杂度也随之提升，使得模型的训练难度加大。因此，这些改进是否有意义仍然有待商榷。

最后，本文在训练NTM时使用的关键方法是课程学习，该方法是由Bengio^[14]等人提出的，该策略类似于人类的学习过程，即如果人类学习的知识或者技术是逐步增加难度的，那么他们的学习速度更快。

§3 任务

本文中考虑的三种算法学习任务类似于Zaremba^[22]提出的任务，分别是：复制，加法和乘法。具体的例子可以见图1。需要强调的是：

1. 对于冯诺依曼体系结构下的计算机，程序员通过编码操作CPU和寄存器可以很轻松的完成这些任务。但是对于算法学习来说，不是通过编码解决这类问题，而是使用大量输入输出样例训练模型，使模型

自动学会这些算法。

2. 训练模型时采用的是序列到序列（sequence-to-sequence）的框架^[23]，即模型每个时间步读入一个字符，当读到终止符后，模型便不在接收输入，而是每个时间步输出一个字符。

输入: abc123. 输出: abc123	输入: 1234+0009. 输出: 1243	输入: 125*009. 输出: 1125
复制	加法	乘法

图 1 三个任务的例子，从左到右分别是复制、加法和乘法。

复制 这个任务的目的是复制输入字符。输入序列是任意长度的字符串，并以终止符结尾。这个任务需要模型能保存和读取任意长度的信息。

加法 这个任务的输入序列如图1所示，是一个求和的式子，目的是计算该算式。这个任务要求模型：1) 有能力存取任意长的信息。2) 发现加法中进位的概念。3) 理解一位数加法的概念。本文考虑了以下三种表示方式：

1. 1 2 3 4 5 + 5 4
2. 1 2 3 4 5
+ 0 0 0 5 4
3. 1 2 3 4 5 + 0 0 0 5 4

在实际的实验中发现，第一种表示方式处理比较困难，尤其是增加了课程学习的设计难度。第二种情况将两个加数作了对齐，这样处理方式降低了模型学习的难度，但是这种情况不具备通用性，不是所有的算法都可以作对齐的。第三种情况是包含第一种情况的，并且比第二种情况更具有通用性。因此本文的实验采用的是第三种表示方式，即相加的两个数都有相同的长度，并且没有对齐操作。如果两个数的位数不同，则位数少的补零至两个数位数相同。乘法任务也是采用这种方式。

乘法 这个任务的目的是计算乘法算式。和加法任务类似，但是复杂度要高于加法任务。

在模型处理序列之前，必须对输入输出序列编码。以复制任务为例，假设输入序列是“abc123.”，通过ascii码表转换，将每个字符表示成8bit的二进制数，该序列将表示成以下七项有序的二进制序列：

1. 0 1 1 0 0 0 0 1
2. 0 1 1 0 1 1 1 0
3. 0 1 1 0 1 1 1 1
4. 0 0 1 1 0 0 0 1
5. 0 0 1 1 0 0 1 0
6. 0 0 1 1 0 0 1 1
7. 0 0 1 0 1 1 1 0

其中“.”表示终结符，即第七项的字符二进制表示00101110。编码完成后，每个时间步向模型输入一个二进制表示的字符。例如：第一个时间步向模型输入字符“a”的二进制表示01100001，第二个时间步将输入字符“b”的二进制表示01101110，以此类推。除了二进制编码方式外，也可以使用其他编码方式，例如one-hot编码。在本文的实验中，复制算法采用二进制编码，加法和乘法采用one-hot编码。

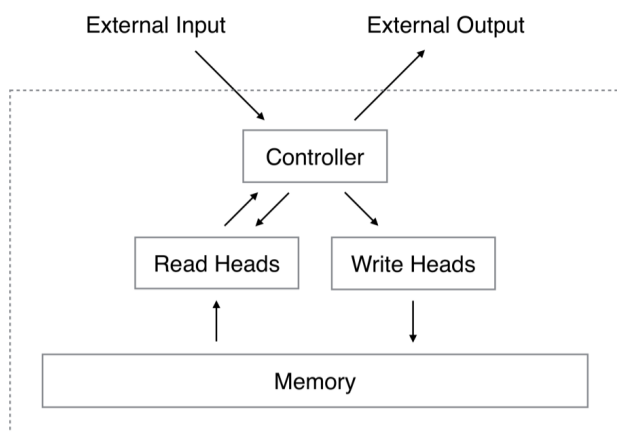


图 2 每一个时间步 t ，控制器从外部获得输入 $input[t]$ ，

并计算得到一个输出 $output[t]$ 。与此同时，控制器通过若干个的读头从记忆模块读取一个向量，并通过写头更新记忆模块。

§4 模型

这部分将简要的介绍NTM。该模型的结构包括两个最基本的部分：控制器（Controller）和外部记忆模块（Memory）。图2展示了一个高度抽象的NTM结构。与神经网络结构类似，控制器在每个时间步（time step）获得外部的一个输入，并计算得到一个输出。不同的是在该时间步控制器还会通过读头（Read Heads）从记忆模块读取一个向量作为输入，并且通过写头（Write Heads）更新记忆模块的内容。模型最关键在于NTM的每个部分都是可微的，所以NTM可以使用梯度下降等优化算法进行训练。

从图2可以知道，NTM有四个部分，分别是控制器，记忆模块，读头和写头。其中，记忆模块是一个给定大小的矩阵；控制器可以选择前向神经网络或者循环神经网络，本文的实验使用的是LSTM^[13]；读头和写头是由控制器的输出确定的。除了这四个部分，模型还需要使用特定的读机制从记忆模块中读取向量，特定的写机制更新记忆模块以及特定的寻址机制确定要读取或者更新的记忆单元。因此，本节的剩余部分将分别介绍读机制，写机制和寻址机制。

§4.1 读机制

M_t 表示在时间 t ，大小为 $N \times M$ 的记忆矩阵，其中 N 表示记忆单元的个数， M 表示每个记忆单元的大小。 w_t 表示在时间 t 通过读头输出的权重向量，其中 w_t 中第 i 维的元素 $\omega_t(i)$ 表示第 i 个记忆单元所占的权重，并且 $\omega_t(i)$ 满足如下约束：

$$\sum_i \omega_t(i) = 1, \quad 0 \leq \omega_t(i) \leq 1, \forall i \quad (1)$$



则按照如下公式计算得到读向量:

$$r_t \leftarrow w_t M_t \quad (2)$$

§4.2 写机制

在时间 t , 写头输出权重向量 w_t , M 维的消除向量 (erase vector) e_t , M 维的加向量 (add vector) a_t , 其中 e_t 的每个元素都属于区间 $[0, 1]$ 。则第 i 个记忆单元的的大小根据如下公式计算得到:

$$\tilde{M}_t(i) \leftarrow M_{t-1}(i)[1 - \omega_t(i)e_t] \quad (3)$$

$$M_t(i) \leftarrow \tilde{M}_t(i) + \omega_t(i)a_t \quad (4)$$

如果存在多个写头, 则将每个写头的消除向量和加向量求和得到最终的消除向量和加向量, 再按照公式3和公式4 进行计算。

§4.3 寻址机制

NTM的寻址机制和现代计算机结构的寻址机制类似, 但是NTM的读写机制是分布式的, 即记忆的读写是按照权重分布到每一个记忆单元的, 因此NTM的取址机制不是定位到某一具体的记忆单元, 而是计算每个记忆单元在读写操作中所占的权重, 也就是之前介绍的读机制和写机制中的权重向量 w_t 。

权重向量的计算结合了内容寻址和位置寻址两种寻址机制, 本文将其总结为以下四个公式:

$$\omega_t^c \leftarrow \frac{\exp(\beta_t K[k_t, M_t(i)])}{\sum_j \exp(\beta_t K[k_t, M_t(j)])} \quad (5)$$

$$w_t^g \leftarrow g_t w_t^c + (1 - g_t) w_{t-1} \quad (6)$$

$$\tilde{\omega}_t(i) \leftarrow \sum_j^{N-1} \omega_t^g(j) s_t(i - j) \quad (7)$$

$$\omega_t(i) \leftarrow \frac{\tilde{\omega}_t(i)^{\gamma_t}}{\sum_j \tilde{\omega}_t(j)^{\gamma_t}} \quad (8)$$

其中, 公式5的 $K[\cdot, \cdot]$ 表示余弦相似度函数:

$$K[u, v] = \frac{u \cdot v}{\|u\| \|v\|} \quad (9)$$

另外, 公式5到公式8 中还涉及了 $k_t, \beta_t, g_t, s_t, \gamma_t$ 这五个参数。这五个参数都依赖于控制器的输出, 而Graves^[8]没有给出明确的定义, 只给定了部分约束条件。在本文的模型实现中, 使用公式10到公式14定义这五个参数。

对于给定的控制器输出 $h_t, k_t, \beta_t, g_t, s_t, \gamma_t$ 满足如下公式:

$$k_t \leftarrow \text{relu}(h_t) \quad (10)$$

$$\beta_t \leftarrow \text{relu}(h_t) \quad (11)$$

$$g_t \leftarrow \text{sigmoid}(h_t) \quad (12)$$

$$s_t \leftarrow \text{softmax}(h_t) \quad (13)$$

$$\gamma_t \leftarrow 1 + \text{relu}(h_t) \quad (14)$$

§5 实验

§5.1 课程学习

课程学习是在训练模型的过程中一项重要的方法, 本文实验中使用的数据集将根据输入序列的长度划分为若干组, 从易到难依次用于训练 (序列的长度越大, 则任务的难度越大)。本节将介绍本文实验中用到的课程学习策略, 在以下的介绍中, MIN 表示所要学习的序列长度的最小值, MAX 表示最大值。

不用课程学习 (No) 该策略作为一个比较基准, 所有的数据都随机生成, 不使用课程学习策略。

普通的课程学习 (Naive) 训练开始时, 课程内容是学习序列长度为 MIN 的算法, 当准确率到达给定的阈值后, 课程难度增加, 开始学习序列长度为 $MIN + 1$ 的算法。重复该过程, 直到序列长度等于 MAX 。该策略是Bengio^[14]提出的, 但是在实验中使用这种策略后模型的性能反而下降了。通过分析实验数据发现模型掌握了新课程的同时就遗忘了以往的课程, 例如, 模型在掌握了两位数的加法, 就遗忘了一位数的加法。这是由于模型在学习新课程的同时, 没有复习以往的课程。

混合的课程学习 (Combining) 在普通的课程学习的基础上, 本文将每个课程的学习内容作了修改, 当



要学习序列长度为 a 的课程的时候,将该课程的学习内容改为序列长度在区间 $[1, a]$ 中随机生成。该策略类似于Zaremba^[1]提出的混合课程学习策略,在本文的实验中,表现出的性能要好于之前的两种策略。但是,随着序列长度增加,新的课程内容所占比例减少,而且课程内容的难度在增加,这使得模型在复杂的课程任务中收敛速度较慢。

基于课程比例的课程学习 (Curriculum-Rate) 在混合的课程学习策略中,新课程的内容在区间 $[1, a]$ 中随机生成,这种方式使得新课程内容的比例随序列长度的增加在不断减少。因此,本文的思路是以一定的比例平衡新旧课程的量,即按照给定的比例生成长度为 a 的序列,剩余的序列从区间 $[1, a - 1]$ 中随机生成。这个策略类似于人类的学习行为,即学习一项新知识的时候,应该以新知识为主,同时还需要复制旧知识。在复杂任务的学习中,该策略的表现要优于不用课程学习和混合的课程学习这两个策略。

本文中实验中使用的课程阈值是0.9, 课程比例是0.5。普通的策略学习性能较差,本文的实验结果部分没有针对该策略进行对比。

表 1 网络超参数

超参数	参数值
学习率	0.0001
momentum	0.9
Mini-batch大小	128
LSTM隐藏层单元的个数	128
读头数量	1
写头数量	1
课程阈值	0.9
课程比例	0.5

§5.2 网络参数

在正式实验之前,本文首先用一小部分数据集对比了随机梯度下降算法以及其他两种改进的随机梯度下降算法RMSprop和Adam,并针对学习率、Mini-batch大

小、权重初始化策略等超参数进行调优。最终在训练时采用RMSprop算法和表1中展示的各项超参数。

对于每个任务,还需要调整与之相关的超参数,这些参数的取值如表2所示。其中序列长度的区间表示输入序列的长度范围,需要注意的是输入序列不仅包括有效字符,还包括终结符“.”,加号“+”和乘号“ \times ”。因此,模型实际要记忆的字符数量要少于序列长度。复制算法实际要复制的序列长度最小是1,最大是10;加法中,每个加数的长度范围是1到10;乘法中,每个乘数的长度范围是1到10。记忆大小是根据任务的实际需求作出的选择,例如复制长度为10的字符序列,每个字符被编码成8bit的二进制数,理论上需要10个记忆单元,每个记忆单元的大小是8,所以记忆模块是一个 10×8 的矩阵。

表 2 任务相关参数

参数值 \ 任务 \ 超参数	序列长度	记忆大小
复制	[2,11]	10×8
加法	[4,22]	21×10
乘法	[4,22]	30×10

§5.3 实验结果

本文的程序基于深度学习框架tensorflow,代码托管在GitHub。实验中运行程序的机器使用的是美团云服务器,配置是8核的CPU,64G内存,100G固态硬盘和1核的Tesla M60 GPU。实验中的模型是NTM,各项超参数如表1所示。实验任务如图1所示。对每个任务,本文都独立训练了一个模型。复制任务训练了50000个batch,加法任务训练了150000个batch,乘法任务训练了300000个batch。每个任务都分别使用三种课程学习策略训练一次。

为了比较课程学习对模型性能的影响,本文对所有任务都绘制了模型在测试集上的准确率随训练轮数变化情况的曲线图,如图3所示。准确率的定义是模型预测准确的测试样例占全部测试样例中的比例。需要注意的

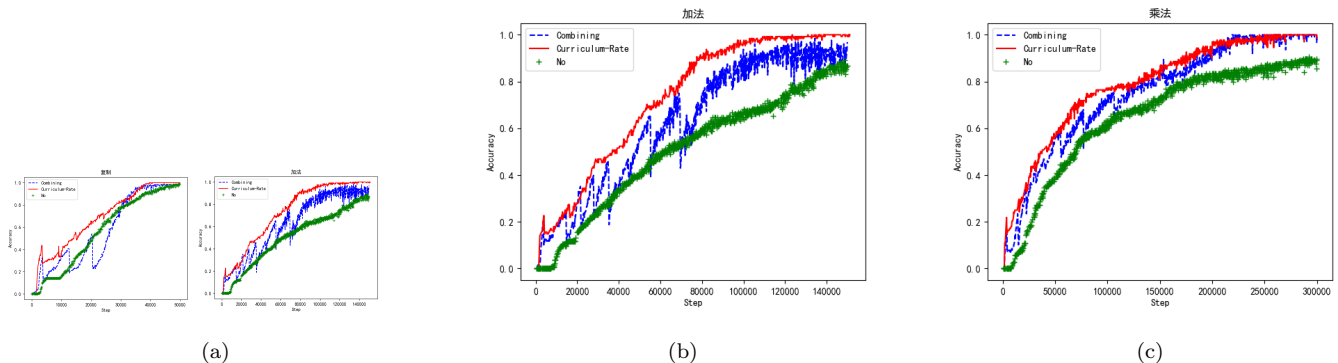


图 3 三种课程学习在测试集上的准确率的对比。纵轴表示的是模型在测试集上的准确率，横轴表示的是训练次数。绿色的“+”连成的曲线表示不用课程学习，蓝色的虚线表示的是使用混合的课程学习策略，红色的实线表示的是使用基于课程比例的课程学习课程。从图中可以看出基于课程比例的课程学习策略训练出来的模型性能最好，而不用课程学习策略训练的模型性能最差。

是，对于一个测试样例，模型会输出一个预测序列，只有整个序列都正确，才认为这个测试样例的预测是正确的。只要一个时间步的预测是不正确的，那么这个预测就是错误的。

从图3可以看出，NTM在每个任务上都能收敛，并且达到接近100%的准确率。其中，基于课程比例的课程学习策略使模型收敛的更快。例如，在加法任务中，使用基于课程比例的课程学习策略训练的模型在120000个batch的训练后准确率达到了99%；使用混合的课程学习策略训练的模型在150000batch训练后准确率达到了95%；不使用的课程学习策略训练的模型在150000个batch训练后准确率只达到90%。在乘法任务中，基于课程比例的课程学习策略和混合的课程学习策略大约在230000个batch后准确率达到99%，但是从两条曲线可以看出混合的课程学习策略训练的模型稳定性较差；不使用课程学习策略训练的模型在300000个batch的训练准确率只有90%。而在相对简单的复制算法任务上，课程学习策略的优势并不明显。由此可以看出，对于相对复杂的任务使用课程学习策略能有效的提升模型的性能，加快模型收敛的速度和收敛的稳定性。特别是基于课程比例的课程学习策略。

§6 总结与未来工作

本文研究并实现了NTM，并将NTM应用于算法学习任务。通过实验说明了NTM可以从输入输出样例中学习得到加法，乘法等算法。此外，实验结果表明课程学习策略是模型取得良好性能的关键。但是仍然有许多问题值得更进一步的研究，例如：1) 当测试序列的长度大于训练序列的最大长度时，NTM无法作出正确的预测。2) NTM不能学习到任意的算法，每学习一个新的算法就需要重新训练。3) 还没从理论上证明NTM不存在梯度消失或者梯度爆炸等问题。4) 课程比例自适应的课程学习策略对模型性能的影响。

参考文献:

- [1] Wojciech Zaremba and Ilya Sutskever. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014.
- [2] E Mark Gold. Language identification in the limit. *Information and control*, 10(5):447–474, 1967.
- [3] Dana Angluin and Carl H Smith. Inductive inference: Theory and methods. *ACM Computing Surveys (CSUR)*, 15(3):237–269, 1983.



- [4] Peter Nordin. *Evolutionary program induction of binary machine code and its applications*. Krehl Munster, 1997.
- [5] Percy Liang, Michael I Jordan, and Dan Klein. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446, 2013.
- [6] Mark Wineberg and Franz Oppacher. A representation scheme to perform program induction in a canonical genetic algorithm. In *International Conference on Parallel Problem Solving from Nature*, pages 291–301. Springer, 1994.
- [7] Ray J Solomonoff. A formal theory of inductive inference. part i. *Information and control*, 7(1):1–22, 1964.
- [8] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.
- [9] Wojciech Zaremba and Ilya Sutskever. Reinforcement learning neural turing machines. *arXiv preprint arXiv:1505.00521*, 419, 2015.
- [10] Lukasz Kaiser and Ilya Sutskever. Neural gpu learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.
- [11] Karol Kurach, Marcin Andrychowicz, and Ilya Sutskever. Neural random-access machines. *arXiv preprint arXiv:1511.06392*, 2015.
- [12] Marcin Andrychowicz and Karol Kurach. Learning efficient algorithms with hierarchical attentive memory. *arXiv preprint arXiv:1602.03218*, 2016.
- [13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [14] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.
- [15] Sumit Gulwani. Dimensions in program synthesis. In *Proceedings of the 12th international ACM SIGPLAN symposium on Principles and practice of declarative programming*, pages 13–24. ACM, 2010.
- [16] Emanuel Kitzelmann. Inductive programming: A survey of program synthesis techniques. In *AAIP*, pages 50–73. Springer, 2009.
- [17] Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. Grid long short-term memory. *arXiv preprint arXiv:1507.01526*, 2015.
- [18] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.
- [19] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.
- [20] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.
- [21] Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in neural information processing systems*, pages 190–198, 2015.
- [22] Wojciech Zaremba, Tomas Mikolov, Armand Joulin, and Rob Fergus. Learning simple algorithms from examples. In *International Conference on Machine Learning*, pages 421–429, 2016.
- [23] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.