基于神经网络图灵机的算法学习研究

王李荣[1]，高志强[1]

（1. 东南大学计算机科学与工程学院，南京，210096）

摘　要：通过输入-输出样例自动学习算法一直是人工智能的核心问题之一。近年来，研究者们将机器学习模型应用于该领域，神经网络图灵机是其中之一。神经网络图灵机能学习到复制、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后，该模型能学习加法和乘法等算法。需要注意的是，课程学习是神经网络图灵机能学到加法和乘法的关键因素之一，但是使用传统的课程学习策略效率不高。因此本文设计并实现了两种新的课程学习策略。在使用这两种策略训练后，神经网络图灵机的性能显著提升。

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

Wang Lirong[1], Gao Zhiqiang [1]

(1. School of Computer Science and Engineering, Southeast University, Nanjing, 210096)

Abstract: Learning algorithm from its input-output examples is a core problem in artificial intelligence. Recently, machine learning models are applied to learning an algorithm, the most famous one is neural turing machine, which is capable of learning copy and repeat copy. We show that neural turing machine trained with sequence-to-sequence framework can learn algorithms like addition and multiplication from pure input-output examples. Notably, it is necessary to use curriculum learning. While conventional curriculum learning is proved to be ineffective, we design two new variants of curriculum learning that improve neural turing machine's performance under all experimental circumstances.

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

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、加法和乘法都被称为算法。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。然而如何让计算机通过输入-输出样例自动学会这些算法，一直是人工智能的核心问题之一。通过输入-输出样例学习算法的任务被称为算法学习[1]。

实际上，算法学习不是一个新任务，它可以追溯到 1967 年[2,3]，是一个古老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，协助程序员发现问题，发现新算法，提升教学质量等。因而在过去几十年里，许多研究者投入到该任务的研究中，并做出了卓越的贡献[4,5,6,7]。

近些年，研究者们利用机器学习模型，尤其是循环神经网络（Recurrent Neural Network, RNN），研究该问题[8-12]。其中与本文最相关的模型是神经网络图灵机（Neural Turing Machine, NTM）[8]。NTM 的结构与长短时记忆网络（Long Short Term Memory, LSTM）[13]类似，但是在 LSTM 中，模型参数的数量会随着记忆单元数量的增加而增加，导致 LSTM 难以训练。为了解决该问题，Graves[8]等人将 NTM 的记忆模块从网络分离出来，模型被分割为可训练的控制器模块和外部记忆模块两个部分。外部记忆模块的大小不再依赖于网络参数的数量。因而，NTM 被称为外部记忆增强的循环神经网络。根据 Graves[8]等人的实验结果，NTM 能学习到复制，重复复制等简单算法。

本文在尝试使用 NTM 学习加法，乘法等算法时，发现模型收敛速度慢，甚至可能会陷入局部极值点，导致模型在测试集上的准确率无法提升。所以本文尝试使用课程学习[14]策略简化问题，将任务

基金项目：南京市科技计划项目（2008 软资 03027）

作者简介：王李荣，（1992-），男，硕士研究生，E-mail: 223050513@seu.edu.cn；高志强，（1966-）男，教授，博导，E-mail: zagao@seu.edu.cn.

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。然而在使用课程学习训练模型时，发现课程学习策略有两个问题：

(1) 算法学习任务不仅要求模型掌握难度大的课程任务，而且不能遗忘中间课程。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 位数的加法如何运算，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训练模型，导致模型在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（一位数、两位数的加法）。这是由于模型学习新课程时，没有及时复习旧课程。

(2) 使用课程学习训练模型时，需要逐步提升课程的难度，因此后续课程的难度将高于当前课程的难度，这一点在算法学习中尤为显著。而普通的课程学习任务没有考虑新旧课程的比例，导致模型性能较低。

基于上述的两点问题，本文设计并实现了两种课程学习策略：基于课程比例的课程学习和自适应的课程学习。在使用这两种策略训练 NTM 后，NTM 能学到加法和乘法等算法，并且模型的收敛速度得到显著提升。

综上所述，本文的主要工作是：

(1) 研究并实现 NTM。利用序列到序列的框架训练 NTM，使该模型学到加法，乘法等复杂算法。

(2) 研究课程学习对模型性能的影响。设计并实现了两种课程学习策略。使用这两种课程学习策略训练模型后，模型性能显著提升。

1 相关工作

算法学习这个任务最早可以追溯到 1967[2,3]年。根据模型的不同或者方法的不同，该任务有不同的名称和内容，例如:合成程序（Program synthesis）、自动编程（Automatic programming）、程序归纳（Program induction）等。但是这些任务的目标一直是使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。

目前有许多模型和方法被应用于算法学习，其中 NTM 是与本文最相关的模型。该模型最早是 Graves[8]提出的。根据 Graves[8]等人的实验结果，NTM 能学习到复制，重复复制等简单算法。

在 Graves[8]等人提出 NTM 一段时间后，研究者们改进了 NTM 或者提出了与 NTM 结构类似的外部记忆增强的循环神经网络模型。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 的思想，提出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，提出增强学习神经网络图灵机（Reinforcement Learning-Neural Turing Machine, RL-NTM）；Graves[18]提出 NTM 的改良版—可微神经计算机（Differentiable Neural Computer, DNC）。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（Hierarchial 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 做出了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。尽管这些模型在某些特定方向的性能得到了提升，但是这些模型的结构更加复杂，导致训练难度增大。对于算法学习来说，需要模型解决长距离依赖问题和变长序列问题。因此，这些改进对于算法学习任务来说显得过于复杂。

最后，本文在训练 NTM 时使用的关键方法是课程学习，该方法的思想来源于人类的学习过程，即如果人类学习的知识或者技术是从易到难逐步增加难度的，那么他们的学习速度更快。

2 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。具体的例子如图 1 所示。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，本文采用的是序列到序列（sequence-to-sequence）[23]的框架，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。

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

通过实验发现，第一种表示方式最简单，但是增加了课程学习的设计难度。第二种方式将两个加数对齐，这样的处理降低了模型学习的难度，然而不是所有的算法都可以对齐的，因而这种方式不具备通用性。第三种方式是包含第一种方式的，并且不需要预先对齐。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。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。乘法采用相同的方式。

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。

图 1 三个任务的例子，从左到右分别是复制、加法和乘法

在模型处理序列之前，必须先对输入输出序列编码。以复制任务为例，假设输入序列是“abc123.”，首先通过 ascii 码表将每个字符转换成整数，然后该整数表示成 8bit 的二进制数，最后将整个序列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 0 0 1 0
(3) 0 1 1 0 0 0 1 1
(4) 0 1 1 0 0 1 0 0
(5) 0 1 1 0 1 0 0 1
(6) 0 1 1 0 1 1 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。在编码完成后，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。这里采用的是二进制编码方式，除了这种编码方式外，还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。

3 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。与循环神经网络结构类似，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，在每个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。该模型最关键的部分在于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训练。

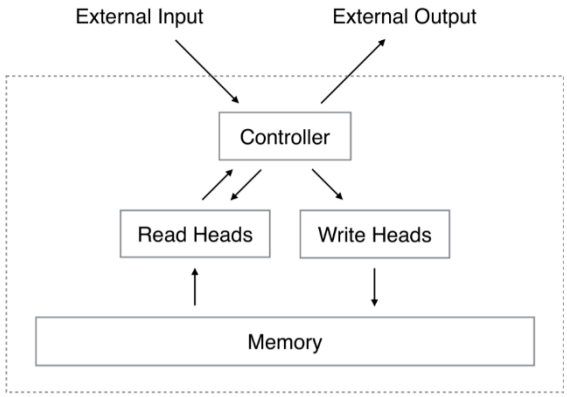

图 2 每一个时间步 *t*，控制器从外部获得输入 *input[t]*，并计算得到一个输出 *output[t]*。与此同时，控制器通过若干个的读头从记忆模块读取一个向量，并通过写头更新记忆模块。

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；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。本节的剩余部分将分别介绍读机制，写机制和寻址机制。

3.1 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 的第 i 维元素 ω_t 表示第 i 个记忆单元所占的权重，并且 ω_t 满足如下约束：

$$\sum_i \omega_t(i)=1,\ 0\le\omega_t(i)\le 1,\ \forall i \tag{1}$$

则按照（2）计算得到读向量：

$$r_t \leftarrow w_t M_t \tag{2}$$

3.2 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（3）和（4）计算得到：

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

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

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（3）和（4）进行计算。

3.3 寻址机制

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 的寻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：

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

$$\omega_t^g(i)\leftarrow g_t\omega_t^c(i)+(1-g_t)\omega_{t-1}(i) \tag{6}$$

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

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

其中，（5）的 $K[\bullet,\bullet]$ 表示余弦相似度函数：

$$K[u,v]=\frac{u\cdot v}{\|u\|\|v\|} \tag{9}$$

另外，（5）到（8）中还涉及了 k_t，β_t，g_t，s_t，γ_t 五个参数。Graves[8]等人没有给出这五个参数明确的定义，只是给定部分约束条件。在本文的模型实现中，使用（10）到（14）定义这五个参数。

对于给定的控制器输出 h_t，五个参数 k_t，β_t，g_t，s_t，γ_t 满足如下公式：

$$k_t\leftarrow relu(h_t) \tag{10}$$

$$\beta_t\leftarrow relu(h_t) \tag{11}$$

$$g_t\leftarrow sigmoid(h_t) \tag{12}$$

$$s_t\leftarrow softmax(h_t) \tag{13}$$

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

4 实验

4.1 课程学习

本节将介绍在本文实验中使用的课程学习策略。本文实验中使用的数据集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）。训练模型时，用到了三种课程学习策略，下面将一一介绍这些策略。在下面的介绍，*MIN* 表示序列长度的最小值，*MAX* 表示最大值。

不用课程学习 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） 在训练开始时，课程内容是序列长度为 *MIN* 的数据，当模型在验证集上的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* 的课程。重复该过程，

直到序列长度等于 *MAX*。该策略是 Bengio[14]提出的。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，在模型掌握了新课程的同时就遗忘了旧课程。例如，模型在掌握了两位数的加法后，就遗忘了一位数的加法。这是因为该策略在模型学习新课程时，没有让模型复习旧课程。

基于课程比例的课程学习（Curriculum-Rate） 在普通的课程学习的基础上，本文对课程内容作了修改。当要学习序列长度为 *a* 的课程时，先按照预先给定的比例生成长度为 *a* 的序列，然后从区间 *[1,a-1]* 中随机生成剩余的序列。当课程比例取 0.5 时，该策略类似于是 Zaremba[1]提出的混合课程学习策略。在本文的实验中，使用该策略训练模型后，模型收敛速度比之前的两种策略快。但是会出现准确率波动较为严重或者收敛速度缓慢的情况。通过分析实验结果，本文发现这种情况是由于新课程的比例与模型的能力不匹配导致的。即当课程较为简单时，模型有能力学习更多的新课程，新课程的比例应该比较大。而当课程难度加大时，应该减少新课程的比例。具体可以看 4.3 节的实验结果。

自适应的课程学习（Self-Adaptation） 由于基于课程比例的课程学习存在的问题，本文设计了一种根据课程难度自动调整课程比例的课程学习策略。该策略根据课程难度自动调整课程比例，即对于课程难度为 *n* 的课程，课程比例 *cr* 满足：

$$cr = f(n), 0 \le f(n) \le 1 \qquad (15)$$

其中 *f(n)* 是与 *n* 负相关的自适应函数，例如 *f(n)=1/n*，*f(n)=1/(n²)*。该策略在课程难度较低时，新课程的比例较高，而当课程难度较大时，新课程的比例较低。在实验中使用该策略后，模型的性能好于其他课程学习策略。

本文实验中使用的课程阈值是 0.9，课程比例是 0.5，自适应函数为 *f(n)=1/n*。由于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做对比。

4.2 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。

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
自适应函数	*f(n)=1/n*

对于每个任务，还需要调整与任务相关的超参数，这些参数的取值如表 2 所示。其中序列长度表示输入序列的长度范围，例如在复制任务中，序列长度的范围是 2 到 11。需要注意的是，输入序列包括终结符“.”，加号“+”和乘号“×”。因此，模型实际要记忆的字符数量要小于序列长度。即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。

4.3 实验结果

本文实验中的程序代码是基于深度学习框架 tensorflow 编写的，托管在 GitHub 平台。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 和表 2 所示。实验任务如图 1 所示。每个任务都独立训练一个模型。在复制任务中，模型训练了 50000 个 batch；在加法任务中，模型训练了 150000 个 batch；在乘法任务中，模型训练了 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，本文的所有任务都绘制了模型在测试集上的准确率随训练次数变化情况的曲线图，如图 3 所示。其中准确率

表 1 任务相关参数

任务	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

是指模型预测准确的测试样例占全部测试样例的比例。需要注意的是，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。只要有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 在测试集上的准确率都能收敛，并且达到接近 100%的准确率。其中使用自适应的课程学习策略训练的模型性能最好。具体体现在：在加法任务中，模型在使用自适应的课程学习策略训练了 120000 个 batch 后，准确率达到了 99%；模型在使用基于课程比例的课程学习策略训练了 150000 个 batch 后，准确率达到了 95%；而不使用课程学习策略，模型在训练了 150000 个 batch 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自适应的课程学习策略大约在 230000 个 batch 的训练后，模型在测试集上的准确率达到 99%，但是从两条曲线可以看出基于课程比例的课程学习策略训练的模型在收敛过程，震荡比较严重；而不使用课程学习策略的话，模型在 300000 个 batch 的训练后，准确率只有 90%。而在相对简单的复制算法任务上，课程学习的优势并不明显。由此可以看出，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稳定性。特别是自适应的课程学习策略。

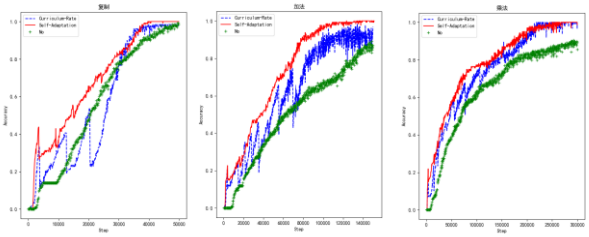

图 3 三种课程学习在测试集上的准确率的对比。从左到右分别是复制，加法和乘法。纵轴表示的是模型在测试集上的准确率，横轴表示的是训练次数。绿色的“+”连成的曲线表示不用课程学习，蓝色的虚线表示的是基于课程比例的课程学习策略，红色的实线表示的是自适应的课程学习策略。从图中可以看出基于课程比例的课程学习策略和自适应的课程学习策略训练出来的模型性能好于不用课程学习策略训练的模型性能。

5 总结与未来工作

本文研究并实现了 NTM，将该模型应用于算法学习任务。实验结果表明：在使用课程学习训练后，NTM 可以从输入-输出样例中学习到加法，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因素之一。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模型。3)没有从理论上证明 NTM 不存在梯度消失或者梯度爆炸等问题。

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, et al. 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.

[10] Lukasz Kaiser and Ilya Sutskever. Neural gpus 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, Jérˆome Louradour, Ronan Collobert, andJason Weston. Curriculum learning. In Proceedings of the 26th annual international conference on machin e 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, 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.