# 一周论文 | 教机器学习编程

2017-07-24 机器之心

作者 | 赵越
学校 | 北卡罗莱纳州立大学博士生
研究方向 | 编译器，程序语言

随着深度学习的发展，人工智能迎来了新一轮的热潮。在图像处理和自然语言处理方面，深度学习已经展现出强大的能力，成为了这些领域的主流方法。与此同时，深度学习也在编程领域开始得到应用。毕竟广义上，编程本身就是人工智能的一部分，机器如果可以学会自动编写程序，那么人工智能岂不是可以实现自举了?

在最近的新闻报道中，大家也会时不时看到一些报道说某个研究可以实现自动编程，甚至任务在不远的将来机器会代替代程序员的工作。实际情况是什么样的呢？本文将通过一些最进的研究探讨一下当前深度学习在编程领域的发展现状，尽可能让读者有一个客观的了解。本文的目的既不是讨论深度学习的技术细节，也不是要做一个全面的综述，而是通过几个典型的应用和相关文章来管中窥豹，看一看当前的现状和挑战。

直观的看，编程语言与自然语言有很多相似的地方，比如都是有一个个符号（token）组成，都可以表达为语法树（parse tree）等等。但是相比在自然领域的应用，深度学习在程序分析或者自动编程领域并没有展现出它独特的优势。这与程序语言和自然语言的区别分不开。我们知道，自然语言具有大量的不确定性和歧义性，所以基于规则的方法很难处理大量的特殊情况。而程序语言则是人为的根据规则严格定义和设计的语言。在语法层面或者低端的语义方面，机器（比如编译器）都可以准确的解析和理解。因此，在这个层面上，基于规则的方法具有天然的优势。

当然，随着程序变得越来越复杂，程序的行为也越来越难靠人工去分析，与此同时，在程序语言的使用上，程序员们需要去记忆或者查找的东西也越来越多。如果人工智能的方法可以让机器理解程序高层的逻辑或者语义，那么程序员就可以从重复低级的任务中解放出来。

实际上，传统的 AI，比如基于逻辑推导的方法，或者基于统计的，贝叶斯推断的机器学习方法在程序生成，分析领域已经有很长的应用历史。深度学习在一些方向可以替代这些方法，达到更好的效果。接下来我们就选择几个典型的程序编程领域来探讨一下他的特点以及深度学习目前的发展。

## 1 程序生成

第一个方向是程序生成，也就是让机器自动编写程序，这是当前 AI 发展的最大的挑战之一。不过这也是深度学习在程序编程领域最热门的方向。相比其他领域，程序生成更需要高级的 “智能”，因此在这个领域，深度学习最可能超越传统方法。

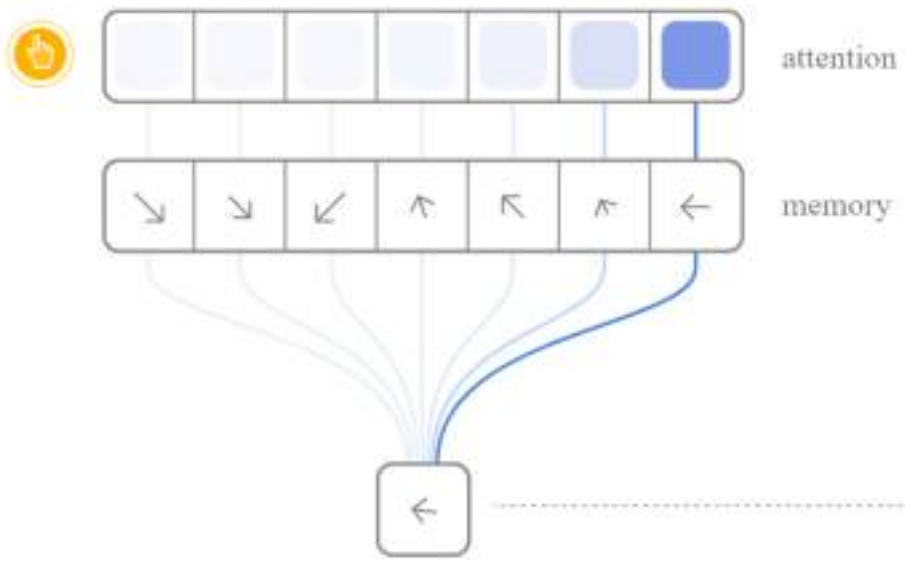
在这个领域下最主要的一类是归纳式程序合成（Inductive Program Synthesis, IPS）。在这种模式下，机器通过观察一系列输入输出的样例来生成一个程序，使得它的行为满足样例中的数据。

使用深度学习的方法来做 IPS 有两类思路：一是 Google 的 Neural turing machines [1] 采用的思路，直接通过样例去训练一个深度学习网络，利用深度学习网络强大的表达能力去 “模仿” 真实的程序，使其行为与输入输出一致。也就是说最终生成的程序就是网络本身。

正如 Yann LeCun 的观点认为深度学习的关键是可微分编程：

> make traditional computing elements differentiable so they can be integrated in gradient-based learning systems.

这类方法的关键在于将程序的行为可微分化。我们知道程序的行为是离散的，每一次指令都是读出或者写入一个存储（比如寄存器，内存等）。只有让离散的行为可微分化，才能利用 back propagation 去训练模型。NTM 巧妙的利用了 [attention][distll] 机制将 RNN 模型和外部的存储阵列连接起来。每次的输入和输出操作都连接到所有的地址，而每个连接都有一个权重（weight）来决定它的有效性。每次输出的结果可以看作是输入和权重的运算，而权重则可以通过网路训练得到。因此，计算机的离散行为就（通过 attention 机制等）转换成了连续的可微分化操作。沿着这类思路的工作还有 [3] 和 [2] 等。




归纳编程的另一种思路是让神经网络生成程序，也就是说网络本身只是程序生成器，并不是程序本身。事实上，这也是归纳编程原本的定义。在神经网络还没有出现前之前这一领域就已经有很长的研究历史了。一个复杂的程序可以看成是由更基本的编程语言元素构成的，理论上我们可以采用枚举的方式用排列组合的方式将这些基本元素组合成程序，然后从中选择正确的程序。我们把所以程序组成的集合看作是 “程序空间”，归纳编程的问题就是要在这个程序空间中执行搜索任务，找到一个满足约束的实例（约束包括程序本身的正确性以及满足输入输出）。

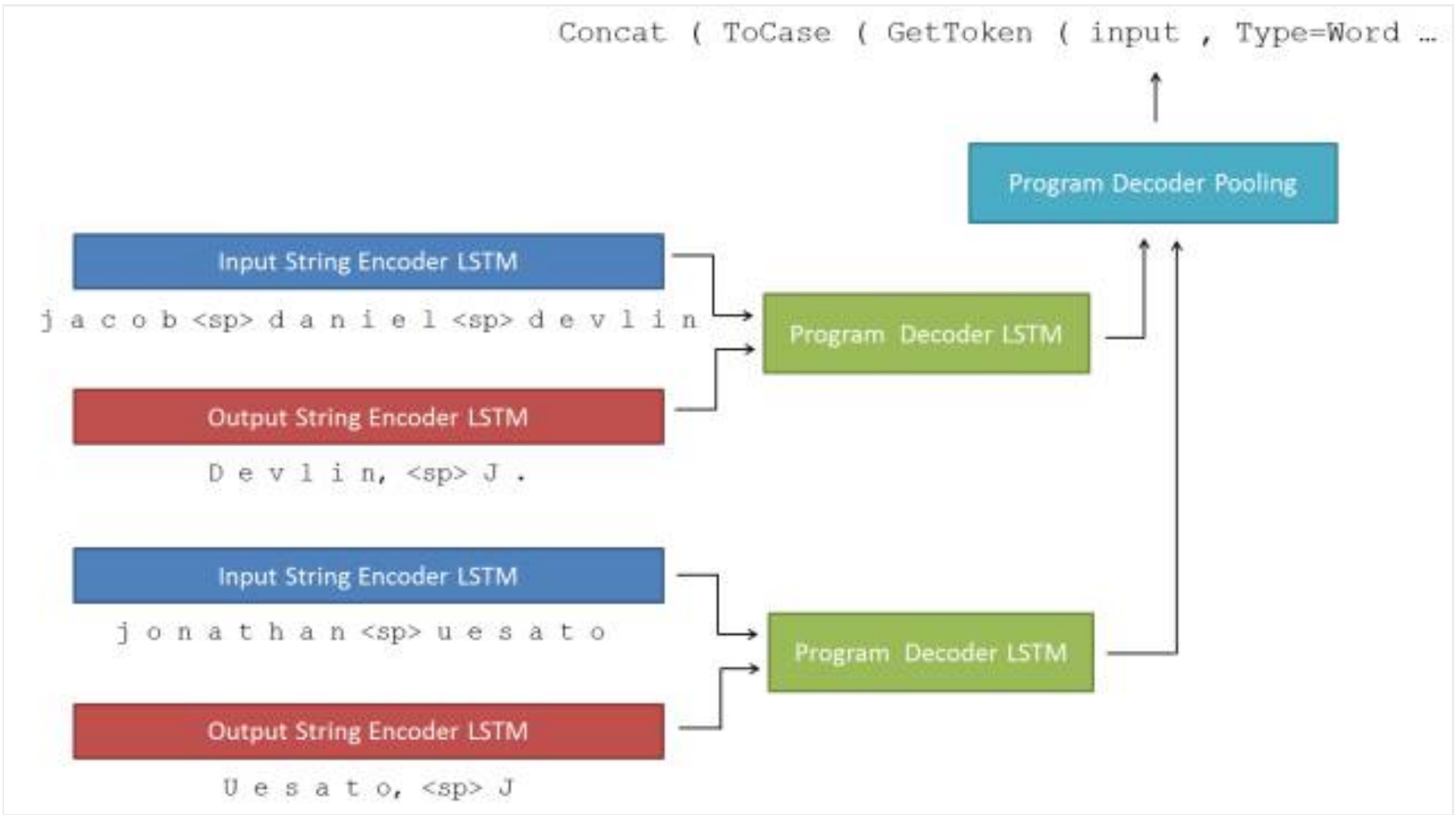
很显然，对于我们常见的程序语言来说，这个空间不仅无限大而且非常稀疏。枚举的方法显然不可行。因此，现在处理归纳编程就要从两方面入手，一是缩小程序空间，二是设计更好的搜索算法。对于前者，我们可以看到当前的研究大多都是在一个自定义的领域语言上（DSL）来生成程序。相比实际的编程语言，这个 DSL 语言包含的元素会少很多，而且往往没有复杂的控制流。其次，在生成过程中，也会限定程序的长度。对于后者，人们常常使用的是基于规则的启发式搜索。这种方法需要大量的专家知识，因此在扩展性上比较差。而且，基于规则的方法无法自我进化。另一种方法就是基于机器学习的算法。本质上就是通过样例数据去学习程序的概率分布。比方说，如果数据样例中，输出都是输入的 2 倍，那么在程序空间中，y=2*x 这样的函数就有很高的概率。对于更复杂的程序，模型需要学习基本元素的出现概率以及他们之间的组合。

相比基于规则的搜索，机器学习的方法具有良好的扩展性和自我进化，适合设计数据驱动的自动化模型。其中，深度神经网络由于具有强大的学习复杂概率分布的能力和良好的生成模式而被越来越多的用于程序生成。其中具有代表性的工作是来自微软的 Deepcoder [4] 以及 RobustFill [5] 。他们也是采用了自己设计的 DSL 语言来降低程序的复杂度，比如图 1，可以看到左边的 DSL 基本数据流是很简单的，而且文中限定了程序长度为 5。

```
a ← [int]                    An input-output example:
b ← FILTER (<0) a            Input:
c ← MAP (*4) b               [-17, -3, 4, 11, 0, -5, -9, 13, 6, 6, -8, 11]
d ← SORT c                   Output:
e ← REVERSE d                [-12, -20, -32, -36, -68]
```

Figure 1: An example program in our DSL that takes a single integer array as its input.

此外，这类模型都采用了编码器 - 译码器的结构。编码器（某种神经网络，比如 LSTM）负责把输入输出的样例转化为模型内部的隐式表示（latent representation），而译码器（另一个 LSTM 模型）则利用这个内部表示预测和生成代码。




目前这些工作大部分还只是局限于实验室里自创的简单的 DSL 上。不过在某些特定场合下也可以找到应用。比如微软就将可以将 RobustFill 技术用在了 Excel 表格的自动填充上（如下图，表格可以根据几个样本推导出填充公式）。

| Input String | Output String |
| --- | --- |
| jacob daniel devlin | Devlin, J. |
| jonathan uesato | Useato, J |
| Surya Bhupatiraju | Bhupatiraju S. |
| Rishabh q. singh | Singh, R. |
| abdelrahman mohamed | Mohamed, A. |
| pushmeet kohli | Kohli, P. |

当然，除了从样本来学习程序，深度学习网络还可以结合它在其他领域的优势来创建更加复杂和智能的模型。比如最近的 Pix2code，通过将卷积网络和 LSTM 网络结合起来识别 UI 的图片，从而自动生成 DSL 代码，然后再进一步编译成实际的代码。类似的，也可以结合自然语言处理，去通过自然语言的逻辑描述去自动生成代码。

## 2 代码补全及软件修复

由于程序的复杂性，机器自动编程还很难实用化。不过，让电脑辅助程序员完成工作倒是更加实用。比如 IDE 里的自动补全（Code completion）可以大大的提高程序员的效率。

传统的自动补全（也是现在编辑器里用到的）基于文本或者语法。程序员必须知道做什么，而自动补全只是帮助程序员减少了打字或者简单的查找。随着软件的日益复杂，程序员往往要花费大量的时间来学习记忆现有的库函数和 API 接口，以及如果组合这些 API 调用实现某个功能。因此在工业界，人们急切希望能够提高编写程序的效率。比如 DARPA（美国国防部）就以及启动了项目自助在这一领域的研究。很多公司如微软，Google，Facebook 等也越来越重视编程工具的效率。如果能利用更复杂的机器学习模型，比如深度学习网络，机器可以根据上下文更加智能的预测程序员接下来该做什么，帮助程序员更加高效的完成任务。

在这一类任务中，模型需要过学习现有的大量的软件库，比如 Github 上的某个语言的代码库，从而学习在当前上下文环境下的语句概率分布。因此本质上讲，这里的模型也是基于统计的方法。这里的挑战是如何将代码抽象为标准化的表达。尤其是代码中，变量名和函数名的处理。比如在 [7] 和 [8] 中都是利用了神经网络强大的长期记忆能力，使得模型相比之前的方法，能够捕获更长的上下文信息。在

下面的例子中，加粗的部分整体完全是由算法提示的。也就是说，机器可以根据情景主动提示合适的 API 调用，而不像常规的自动补全，需要程序员先输入前面的字符。

```
void exampleMediaRecorder() throws IOException {
    Camera camera = Camera.open();
    camera.setDisplayOrientation(90);
    camera.unlock();
    SurfaceHolder holder = getHolder();
    holder.addCallback(this);
    holder.setType(SurfaceHolder.SURFACE_TYPE_PUS
    rec = new MediaRecorder();
    rec.setCamera(camera);
    rec.setAudioSource(MediaRecorder.AudioSource.
    rec.setVideoSource(MediaRecorder.VideoSource.
    rec.setOutputFormat(MediaRecorder.OutputForma
    rec.setAudioEncoder(1);
    rec.setVideoEncoder(3);
    rec.setOutputFile("file.mp4");
    rec.setPreviewDisplay(holder.getSurface());
    rec.setOrientationHint(90);
    rec.prepare();
    rec.start();
}
```

除了自动补全，类似的方法还可以用于软件错误的检测和修复。因为既然可以根据上下文自动补全信息，自然可以让机器根据上下文检测错误。比如 DeepFix [9] 是一个端到端的解决方案。通过训练一个带有 attention 机制的序列到序列（Seq2seq）模型，可以用于预测程序中某个语法错误的位置以及正确的格式。这类工作还可以用于帮助在线编程学习系统检测和修复学生提交的代码中的大量语法错误 [10] 。不过，目前来看，这些修复都还是非常表面的语言使用错误。实际上这些错误都可以被编译器很好的找到。在这点上深度学习模型并没有特别大的优势。不过随着这个方向的发展，希望基于深度学习的模型能够理解程序的更高层面的语义，从而帮助程序员修复逻辑层面的错误。

## 3 结语

编程语言和自然语言的区别决定了深度学习不同的发展状况。不过，深度学习在编程语言上的应用中

可以使用到很多在自然语言处理上的技术，比如对符号的处理，词向量的表征等。在模型上很多都可以使用自然语言处理的模型，比如 RNN 模型和 CNN 模型都会用到，还有一些基于某些结构的模型如基于图或者树的神经网络模型等。

在应用上，深度学习已经被用于简单的程序生成，代码自动补全。但是更多的领域还没有太多深度学习的身影，比如程序性能的分析和优化。在这个方向，深度学习面临的第一个很大的挑战就是数据。我们知道深度学习在图像和自然语言处理领域的快速发展离不开大量高质量标记数据的产生。但是在程序分析领域，一方面是整个社区对深度学习的支持不足，另一方面，程序的很多行为特性与实际的系统紧密相关，因此，我们很少能看到大量高质量的公开数据可供研究。另一个挑战就是这个方向往往需要很多编程语言设计和编译器原理等背景知识，而且对每一种语言都要做专门的语法处理和大量的准备工作，这也极大的提高了研究的门槛。

可以预计，在短期内深度学习在编程领域还会继续发展，但是很难像其他领域一样达到非常突出的效果。让机器替代人类编程，估计还要很久。

[1] Graves, Alex, Greg Wayne, and Ivo Danihelka. “Neural turing machines.” arXiv preprint arXiv:1410.5401 (2014).
[2] Bošnjak, Matko, et al. “Programming With a Differentiable Forth Interpreter.” arXiv preprint arXiv:1605.06640 (2016).
[3] Neelakantan, Arvind, Quoc V. Le, and Ilya Sutskever. “Neural prorammer: Inducing latent programs with gradient descent.” arXiv preprint arXiv:1511.04834 (2015).
[4] Balog, Matej, et al. “Deepcoder: Learning to write programs.” arXiv preprint arXiv:1611.01989 (2016).
[5] Devlin, Jacob, et al. “RobustFill: Neural Program Learning under Noisy I/O.” arXiv preprint arXiv:1703.07469 (2017).
[6] Emilio Parisotto, et al. “Neuro-Symbolic Program Synthesis”, 5th International Conference on Learning Representations (ICLR 2017)
[7] Raychev, Veselin, Martin Vechev, and Eran Yahav. “Code completion with statistical language models.” ACM SIGPLAN Notices. Vol. 49. No. 6. ACM, 2014.
[8] Bhoopchand, Avishkar, et al. “Learning Python Code Suggestion with a Sparse Pointer Network.” arXiv preprint arXiv:1611.08307 (2016).
[9] Gupta, Rahul, et al. “DeepFix: Fixing Common C Language Errors by Deep Learning.” AAAI. 2017.
[10] Bhatia, Sahil, and Rishabh Singh. “Automated correction for syntax errors in programming assignments using recurrent neural networks.” arXiv preprint arXiv:1603.06129 (2016).

## 关于 PaperWeekly

paperweekly

自然语言处理 计算机视觉 深度学习

*Knowledge Sharing Made Easy.*

▲ 长按扫描二维码关注

内容转载自公众号

PaperWeekly 了解更多