1.DeepCoder：学习编程（DeepCoder: Learning to Write Programs）

摘要：我们首次提出了使用深度学习通过输入输出样本来解决编程竞赛风格的问题（programming competition-style problems）的方法。这个方法就是通过训练一个神经网络来预测一个通过输入值生成输出值的程序的属性。我们使用神经网络的预测功能来增强那些来自编程语言社区，包括枚举搜索（enumerative search）和基于 SMT 的解算器（SMT-based solver）的搜索技术。经验表明，我们的方法在强大的非增强基线（non-augmented baselines）和循环神经网络方法上也可以大幅度地提升速度，并且我们可以把编程竞赛网站上那些复杂的问题转化为简单的问题进行解决。

2. 具有连续和离散寻址方案的动态神经图灵机（Dynamic Neural Turing Machine with Continuous and Discrete Addressing Schemes）

摘要：在本论文中，我们通过引入一个可训练的存储器寻址方案，将神经图灵机（NTM）扩展成动态神经图灵机（D-NTM）。这个寻址方案可为每个存储器单元维持两个单独的向量——分别是内容向量和地址向量。这允许 D-NTM 学习各种基于位置的寻址策略，其中就包括线性和非线性策略。我们使用了连续可微的和离散不可微分的读/写机制来实现 D-NTM。我们调查了在 Facebook bAbI 任务中同时使用前馈技术和 GRU 控制器学习对内存进行读写实验的机制和效果。我们在 Facebook 的一组 bAbI 任务上对 D-NTM 进行了评估，结果表明，D-NTM 优于 NTM 和 LSTM 基线。同时，我们还提供了在序贯的 MNIST 数据集、联想回忆（associative recall）和复制任务上进行的实验的进一步结果。

3. 通过探索不完美的奖励来改善策略梯度算法（Improving Policy Gradient by Exploring Under-appreciated Rewards）

摘要：本文为带有改进探索性能的无模型强化学习（RL）提出了一种新的策略梯度算法。当前基于策略的方法使用熵正则化（entropy regularization）来促使对奖励景观（reward landscape）进行无向探索，但是这种方法在高维空间中对稀疏奖励是无效的。所以我们提出了一种更有针对性的探索策略，以此来对探索欠佳的奖励区域进行促进。如果动作序列的对数概率是在当前策略下估计的奖励值，则认为动作序列是不令人满意的。我们仅需要对标准 REINFORCE 算法进行一下小修改，那么所提出的探索策略就会易于实现。我们在一组算法任务中对该方法进行了评估，其中这些算法任务对于强化学习而言，是一个长期的挑战。我们发现我们的方法降低了超参数灵敏度，并且相对于基线方法有了明显的改进。值得注意的是，该方法能够解决基准多位数加法任务。据我们所知，这是纯正的强化学习方法第一次仅使用奖励反馈来解决加法问题。

4. 用神经网络拆分和征服（Divide and Conquer with Neural Networks）

我们考虑通过仅观察输入 - 输出对来学习算法任务。而不是把它作为一个在输入 - 输出映射上没有任何假设的黑盒离散回归问题（black-box discrete

regression problem）。我们集中于服从分治算法的原则（principle of divide and conquer）的任务，并研究它在学习方面的影响。这个原则通过学习两个尺度不变的原子运算符：如何将给定的输入拆分（split）为两个不相交的集合和如何将两个部分求解的任务融合（merge）成一个较大的部分解来创建了一个强大的归纳偏差（inductive bias），它是我们利用递归定义的神经结构进行开发的。尺度不变性（scale invariance）创建了可以在该架构的所有阶段共享的参数，并且动态设计创建了其复杂性在可微分的形式下可以进行调谐的架构。

因此，我们的模型通过反向传播进行训练，不仅可以通过执行更浅的计算图（computation graph）来最小化输出处的误差，而且可以尽可能有效地进行。此外，由于尺度不变性，可以仅使用输入/输出对来训练模型，而不需要知道中间分割和合并的方案。事实证明，准确性和复杂性不是独立的性质，同时，当学习的复杂性与底层的复杂性相匹配的时候，排序和查找平面凸包（sorting and finding planar convex hulls）这两个范式问题会得到更高的精度和更好的泛化。

5. Lie-Access 神经图灵机（Lie-Access Neural Turing Machines）

摘要：最近的工作已经证明了同时使用显式外部存储器结构和用于算法学习的深层神经模型的有效性（Graves et al., 2014; Weston et al., 2014）。这些模型利用传统离散存储器访问结构（随机存取、堆栈、tapes）的可区分版本来提供计算任务所需的可变长度存储。在这项工作中，我们提出了一个专门为神经设置（neural setting）所设计的替代模型——Lie-access 存储器。在这个范式中，使用 key-space manifold 中的连续头来访问存储器。通过由控制器生成的诸如移位或旋转的李群（Lie group）动作来移动磁头，并且根据与每个存储器相关联的键的距离来执行软存储器访问。我们认为李群（Lie group）对离散内存结构进行了自然推广，例如图灵机，因为它们在提供反向和身份运算符的同时也保持了可微分性。为了试验这种方法，我们在几个不同的李群（Lie group）上实施了几个简化的 Lie-access 访问神经图灵机（LANTM）。我们发现这种方法能够在一系列算法任务上表现良好。

6. 通过递归实现神经编程架构通用化（MAKING NEURAL PROGRAMMING ARCHITECTURES GENERALIZE VIA RECURSION）

摘要：从经验上讲，试图使用数据来学习编程的神经网络显示出了较差的通用性。此外，当输入的复杂度超过了阈值，我们就很难去推测这些模型的表现。为了解决这个问题，我们提议增加一个带有关键抽象方法的增强神经架构——递归（recursion）。作为一个应用，我们在神经编程器-解释器框架（Neural Programmer-Interpreter framework）中的两个任务——添加和排序——上实现了递归，实现了在更少量的训练数据下展示出了优越的泛化性和可解释性。通过将问题划分为一个个更小的部分，并且极大地减少每个神经网络组件域，这种方法也让我们可以很容易去证明担保整个系统的行为。根据我们的经验表明，为了让神经架构更稳健地学习程序语义（program semantics），有必要引进一个递归这样的方法。

7. 神经组合优化（Neural Combinatorial Optimization）

摘要：本文提出一个使用神经网络和强化学习来解决组合优化问题的框架。我们专注于旅行商问题（TSP/traveling salesman problem），并且训练了一个循环网络来通过给定一组城市坐标预测不同城市排列的分布。我们使用 负旅程长度（negative tour length）作为奖励信号，使用策略梯度方法优化循环网络的参数。我们在一组训练图和单个测试图上分别让网络参数进行了学习，然后对它们进行了比较。最优的结果是先在训练集上优化网络然后在单个测试图上进行细化。我们的主要的成果是，在具有高达 100 个节点的 2D 欧几里德图上，没有任何监督的情况下，我们的方法会优于 Christofides（1976）算法，并且可以和任何一个最好的开源 TSP 解算器媲美。

8. 神经函数式编程（NEURAL FUNCTIONAL PROGRAMMING）

摘要：我们讨论了在构建一种适合在输入-输出示例中学习程序的端到端可微型编程语言时出现的一系列建模选择。为了从编程语言研究中提取一些隐含的信息，我们研究了内存分配方案、不可变数据、类型系统和内置的控制流结构对学习算法的成功率的影响。我们构建了一系列的模型来生成一个简单的可微函数编程语言。我们的实证评估表明，和现有基线相比，这种语言允许学习更多的程序。

9. 神经程序格（NEURAL PROGRAM LATTICES）

摘要：我们提出神经程序格（NPL），它是一个神经网络，用来学习一个主要基于平面序列基本操作的分层程序结构。与只能使用强监督的现有的方法（例如程序的完整执行轨迹）相比，NPL 学习的监督性弱了很多，并且在仅仅几个完全执行轨迹的帮助下就执行相当好。我们展示了我们的模型学习任务的能力，就如同 ADDITION 和在网格世界中移动块。我们表明，通过训练大多的非结构化操作序列，NPL 能够提取序列的潜在结构，并能学习去表示程序的抽象化。值得注意的是，NPL 可以在比现有方法弱得多的监督下实现更先进的性能。

10. 神经-符号程序合成（Neuro-Symbolic Program Synthesis）

摘要：近年来，很多人建议使用神经结构来解决程序归纳（Program Induction）问题。这些神经结构可以通过一些给定的输入- 输出样本进行学习，然后在输入输出之间建立一种映射关系，最后可以推广到一些新的输入。虽然这种方法的结果让人印象深刻，但是它们也有很多比较重要的限制：（a）它们难以训练并且会花费很昂贵的计算资源，（b）必须为每个任务（程序）单独地训练模型，（c）难以解释或验证所学习的映射的正确性（因为它是由神经网络定义的）。在本文中，我们提出了一种新技术：神经-符号程序合成（Neuro-Symbolic Program Synthesis），可以克服上述问题。经过训练，我们的方法可以使用特定领域的语言自动的构建计算机程序，这种语言与测试时所提供的一组输入输出所用的示例语言一致。我们的方法是基于两个新的神经模块。第一个模块称为互相关 I/O 网络（the cross correlation I/O network），它可以通过一组给定的输入输出示例，生成 I / O 示例集合的连

续表示。第二个模块，被称为递归-反递归神经网络（R3NN：Recursive-Reverse-Recursive Neural Network），它接收第一个模块生成的样本的连续表示，通过递增地扩展部分程序来合成程序。我们通过将其应用于一种丰富、复杂并且基于正则表达式的字符串变换域来证实我们的方法的有效性。实验表明，R3NN 模型不仅能够从新的输入-输出样本构建程序，而且还能够为在训练期间从未观察到的任务构建新的程序。

11. 使用一个可微分的 FORTH 解释器编程（PROGRAMMING WITH A DIFFERENTIABLE FORTH INTERPRETER）

摘要：通过提供足够的训练数据，很多神将网络都可以通过学习去计算任意一种函数。然而，在现实生活中，除了一小问题具有较多的训练数据外其他问题的训练数据都很少，现在有一个核心问题是如何将先验知识纳入模型。在这里我们考虑到程序过程先验知识的情况，比如知道序列转换程序的所有递归结构或者知道程序可能对实数使用算术运算来解决任务。为此，我们为编程语言 Forth 提出了一个可微分的解释器。通过基于 Forth 的双栈机器的神经实现，程序员可以编写具有空缺的程序草图，该空缺会被填充一些从程序输入 - 输出数据训练所得的行为。由于程序解释器是端到端可微分的，我们可以通过在用户指定的目标上使用梯度下降技术直接优化这种行为，并且也可以将程序集成到任何更大的神经计算图中。经验表明，我们的解释器能够有效地利用不同水平的先前的程序结构，在大幅度减少数据量的情况下学习复杂的转导任务，如序列排序或加法运算，并且对程序的规模进行了更好的概括。此外，我们引入了基于符号计算和并行分支的神经程序优化设计，这让运行速度得到了明显的改进。

12. 使用分层生成卷积神经网络的无监督程序推导（Unsupervised Program Induction with Hierarchical Generative Convolutional Neural Networks）

摘要：自动推断可以将一组输入映射到输出的计算机程序仍然是一个开放和极具挑战性的问题。由于可能的情况十分多，因此存在巨大的搜索空间，并且还需要处理高阶逻辑（例如 for-loops 或递归），所以程序导入任务是十分困难的。在本文中，我们使用分层生成卷积神经网络（Hierarchical Generative Convolutional Neural Networks（HGCNN））从输入/输出对自动生成程序。HGCNN 可以从使用标准的搜索技术构建的程序中以分层的方式预测候选的代码行。值得一提的是，该程序仅使用随机生成的程序进行构建，因此可以被视为无监督学习方法。我们提出的方法不仅可以生成过程简单的程序（比如交换变量值），也可以生成一些复杂的循环和分支（例如，找到一个数组的最大元素）。我们还展示了一些带有循环嵌套的程序（比如冒泡排序）。与 LSTM 这样的基准相比，我们的方法拥有更好的预测精度。