文献综述

——M201873274 赵振凯

# 前言

由深度学习框架(例如，Caffe和TensorFlow)采用的一个主要编程范例是将神经网络表示为静态数据流图，其中，神经网络中的计算函数与图中的节点相关联，并且将计算图的输入和输出与边缘相关联。对于给定的图形和数据样本，软件工具包可以根据反向传播和自区分规则，自动推导出正确的训练或推理算法。通过适当的优化,执行这些静态数据流图可以高效；所有数据的数据流图是固定的，多个样本的评估通过一个图形可以自然地分批处理,利用改进的并行处理能力的现代硬件(例如gpu)。此外，通过分离模型的声明和执行，它可以使图表在声明时被优化一次，在执行时，这些优化可以提高处理任意输入数据的效率。

虽然数据流图具有主要的效率优点，但其适用性高度依赖于固定数据流图(即神经网络体系结构)。然而在动态神经网络中，网络架构随着每个输入样本有条件地发生变化，例如计算可变长度、树和图形序列的神经网络。

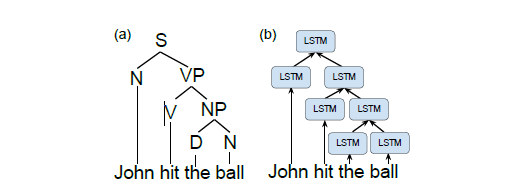
由于人们对这类动态模型越来越感兴趣，近年来基于动态声明的框架越来越流行，每个示例声明不同的数据流图。尽管动态声明对于开发者来说是方便的，因为它消除了在训练开始之前计算被完全指定的限制，但是它显示了一些限制。首先，为每个示例构造一个图会导致大量开销，开销随输入实例的数量线性增长。事实上，我们发现在某些框架中，图形构造所花费的时间要比计算时间长。这也阻碍了复杂静态图优化技术的应用。此外，由于每个示例都拥有一个指定其独特计算模式的数据流图，因此跨实例将类似形状的计算批处理在一起是很困难，因此计算效率较低。尽管最近的研究方面已经取得了一些进展,如何从不同的图形自动批处理计算操作仍然是一个困难的问题。

然而，现有的DL编程模型在处理动态网络体系结构方面效率低下，原因是：(1)由于每个实例的数据流图构造和处理的重复造成了大量的开销；(2)多样本批量执行困难；(3)无法将静态图形中使用的图形优化技术结合起来。

在本文中,我们目前的CAVS,一个运行时系统,克服这些瓶颈,实现高效的动态得到培训和推理。CAVS代表了一种动态神经网络作为静态顶点函数F和特定于一个动态图G .它避免了重复的开销只声明和构造F一次施工的图表,并允许使用静态图优化技术在F预定义操作。

# 研究背景

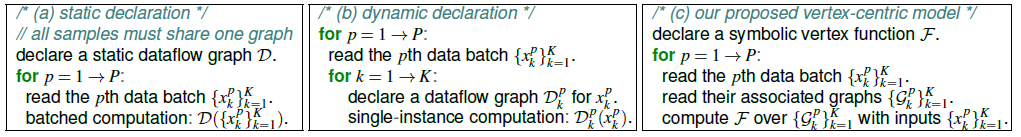
成功的神经网络模型通常表现出合适的架构,捕获输入数据的结构。例如，卷积神经网络的fixed-structured操作适用于固定大小的图像，是非常有效的,正是因为他们捕捉计算机视觉领域中常见的空间不变性。然而，除了图像，许多数据形式在结构上是复杂的，不容易被固定结构的神经网络捕获。在神经网络设计中适当地反映这些结构，在情感分析、句子对之间的语义相似性、图像分割等方面表现出了良好的效果。

我们将选区解析问题作为一个例子。以自然语言编写的句子通常由其选区解析树表示，其结构根据句子的内容(图1(a))而变化。选区分析是自然语言处理的一个重要问题,旨在确定相应的语法解析树的所有内部节点类型的句子。图1(b)显示了一个考虑这种句法结构的网络示例，该网络通过遍历解析树底向上生成句子的表示，并使用称为树结构长短期内存(tree - lstm)的动态NN组合每个子树的表示。特别是，树的每个节点映射到一个LSTM函数。在每个节点上，它接受可变数量的输入，并向父节点返回一个表示解析语义的向量，直到根LSTM节点返回一个表示整个句子语义的向量为止。

图一 解析树

重要的是，在每个输入样本上，NN结构随底层解析树的变化而变化，但是相同的LSTM单元在形状上是不变的，并且在每个内部节点上重复。类似的例子可以发现图形输入和可变长度的序列。我们将这些对不同输入样本具有不同结构的神经网络称为动态神经网络，而不是对所有样本具有固定网络结构的静态网络。

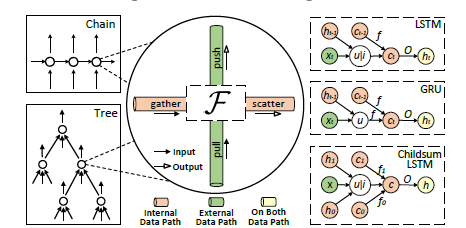
神经网络与有向图之间存在一种自然的联系：我们可以将图节点映射到神经网络中的计算操作或参数，并让边表示节点之间传递数据的方向。在这种情况下，我们可以将训练神经网络的过程表示为通过计算图(即数据流图)流动的数据批。

如前所述,静态声明是一个占主导地位的编程范式编程得到。图2(a)总结了工作流,假设所有样本数据共享一个固定的神经网络结构宣布象征性地在一个数据流图D .静态声明,使用单一数据流图D,不能表达动态NNs与结构变化的数据样本。主要解决这问题是放弃静态数据流图的效率收益,而是使用一个动态的声明框架。图2 (b)说明了动态的工作流声明。通过创建一个独特的数据流图D为每个样本根据其相关结构、动态宣言能够表达sampledependent数据流图。然而造成额外的开销在图施工,并限制运行时优化,这通常会导致效率低下的执行执行。特别是,因为需要构造一个数据流图D /样本,与样品的数量是线性增加的开销,甚至有时收益率降低性能,与图结构优化的实现框架。此外,我们几乎不能受益于任何优化的数据流图。我们将不得不执行图像处理/优化为每个数据流图和每一个样本;但将这种优化本身有一个不小的开销。更重要的是,我们不能批量的计算不同结构图表,我们注意到在图2 (b)单实例计算D会非常低效的没有批处理计算。为了解决批处理问题，最近的一些工作，特别是TensorFlow Fold和DyNet，提出了动态批处理，动态分组来自不同图形的形状类似的操作，并尽可能对它们的执行进行批处理。Fold将动态数据流图转换为静态控制流图以支持批处理执行，但引入了复杂的函数式编程接口和较大的图形预处理开销。正如我们将展示的，图形构造有时会使计算速度减慢4x。DyNet提出了一种自动批处理策略，通过分析每个细粒度操作符来搜索批处理机会，而这一步本身的开销不可忽略。它也不能实现数据流图级优化。

图二 三种声明方式

Cavs的动机来自于动态神经网络的一个关键特性:大多数动态神经网络被设计成显示递归结构;在递归结构中，一个静态计算函数是按照特定于实例的图的拓扑顺序应用的。举例来说,如果我们选区解析树表示为一个图G,在树的每个节点映射到一个顶点在G,我们注意到Tree-LSTM可以解释如下:一个计算细胞功能,提前指定,应用从叶子到根,依赖关系之后G G与输入样本可能会改变,但细胞功能本身总是静态的:当应用于不同G顶点时，它被一组固定的可学习参数参数化，且与它的邻居以相同的方式相互作用。

# CAVS的设计和优化

传统的数据流基于图形的编程模型通常会将F的计算工作流放在G里，并要求用户将其作为完整的数据图表表达出来。相反，Cavs将静态顶点函数F与输入图形G分离(如图3所示)，用户在F中使用相同的符号运算符集合计算工作流，而Cavs提出了4个额外的API，即gather, scatter, pull, push，来指定在G中连接的顶点之间如何传递消息。

图三 Cavs动态输入图G(左)和静态顶点函数F(右)

在静态声明中，用户程序中的一个符号通常对应一个具有批量大小维度的固定大小的张量对象。而在Cavs中，每个批处理任务V都是在运行时确定的。为了提高批处理计算的效率，Cavs必须保证对于每个批处理任务，在一组运行时确定的顶点上，F的每个表达式的输入在内存中合并。

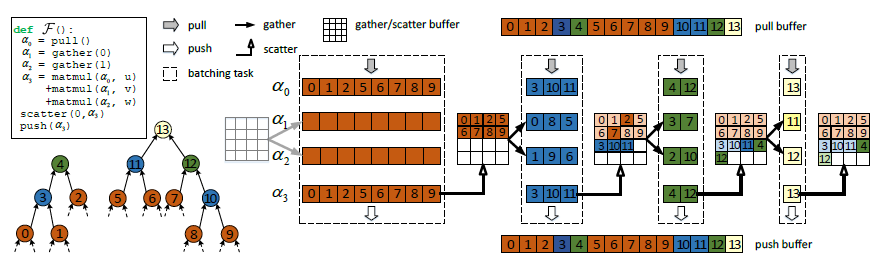
图四 内存管理

图4中，在两个输入树(左下)的F(左上方)前向通过的存储器管理。Cavs首先分析F和输入—它创建四个动态张量，并计算出将有四个批处理任务(带虚线框)。从第一个任务(橙色顶点0;1;2;5;6;7;8;9)开始，Cavs对f中的每个表达式进行批量求值，例如，对于pull表达式a0 = pull()，它使用id对来自pull缓冲区的所有顶点的a0内容进行索引，并将其连续复制到a0；对于分散表达式和推送表达式，它将输出(a3)的一个副本分散到集合缓冲区，并分别将其推送到推送缓冲区。骑士然后继续下一个批处理任务(蓝色顶点)。在这个任务中，Cavs再次计算顶点3;10;11g的F表达式。(例如，对于pull表达式a0 = pull()，它再次从pull buffer中提取a0的内容;收集表达式a2 =聚集在顶点3(1),它集的输出3的第二个孩子,也就是1);它在每个动态张量的末尾连续写入结果。直到所有批处理任务完成为止。

Cavs是使用c++库实现的，可与现有深度学习框架集成，以增强它们对动态神经网络的支持。它由三个主要的层(最流行的框架的理由)：(1)提供设备无关的前端象征性的编程接口；(2)实现核心执行逻辑的中间层；(3)为所有符号操作符提供特定于设备的内核的后端。除了四个api之外，Cavs提供了宏操作符的顶点功能。用户通过编写符号表达式和指定读取输入图形的方法来实例化它。它封装了gather/ scatter语义，因此用户可以继续使用更高级别的api。为了构建更复杂的神经网络架构(例如，编码-解码，LRCN)，用户使用pull和 push来连接多个顶点功能，或外部结构。Cavs在这一层有其核心的运行时逻辑，即批处理调度器、内存管理、执行引擎等。

# 参考文献

1. ABADI, M., BARHAM, P., CHEN, J., CHEN, Z., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., IRVING, G., ISARD, M., ET AL. Tensorflow: A system for large-scale machine learning. arXiv preprint arXiv:1605.08695 (2016).
2. BAHDANAU, D., CHO, K., AND BENGIO, Y. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473 (2014).
3. BERGSTRA, J., BASTIEN, F., BREULEUX, O., LAMBLIN, P., PASCANU, R., DELALLEAU, O., DESJARDINS, G., WARDE-FARLEY, D., GOODFELLOW, I. J., BERGERON, A., AND BENGIO, Y. Theano: Deep Learning on GPUs with Python. In NIPSW (2011).
4. CHEN, R., SHI, J., CHEN, Y., AND CHEN, H. Powerlyra: Differentiated graph computation and
5. CHEN, T., LI, M., LI, Y., LIN, M., WANG, N., WANG, M., XIAO, T., XU, B., ZHANG, C., AND ZHANG, Z. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. arXiv preprint arXiv:1512.01274 (2015).
6. CHETLUR, S., WOOLLEY, C., VANDERMERSCH, P., COHEN, J., TRAN, J., CATANZARO, B., AND HELHAMER, E. cudnn: Efficient primitives for deep learning. arXiv preprint arXiv:1410.0759 (2014).
7. CUI, H., ZHANG, H., GANGER, G. R., GIBBONS, P. B., AND XING, E. P. Geeps: Scalable deep learning on distributed gpus with a gpuspecialized parameter server. In Proceedings of the Eleventh European Conference on Computer Systems (2016), ACM, p. 4.
8. DAVE, C., BAE, H., MIN, S.-J., LEE, S., EIGENMANN, R., AND MIDKIFF, S. Cetus: A source-tosource compiler infrastructure for multicores. Computer 42, 12 (2009).
9. DONAHUE, J., ANNE HENDRICKS, L., GUADARRAMA, S., ROHRBACH, M., VENUGOPALAN, S., SAENKO, K., AND DARRELL, T. Long-term recurrent convolutional networks for visual recognition nd description. In Proceedings of the IEEE conference on computer vision and pattern recognition (2015), pp. 2625–2634.
10. DYNET VARIABLE LENGTH LSTM. https://github.com/neulab/dynet-benchmark.
11. FACEBOOK. http://pytorch.org/.
12. FACEBOOK OPEN SOURCE. Caffe2 is a lightweight, modular, and scalable deep learning framework. https://github.com/caffe2/ caffe2, 2017.
13. GAUNT, A., JOHNSON, M., RIECHERT, M., TARLOW, D., TOMIOKA, R., VYTINIOTIS, D., AND WEBSTER, S. Ampnet: Asynchronous modelparallel training for dynamic neural networks. arXiv preprint arXiv:1705.09786 (2017).
14. GONZALEZ, J. E., LOW, Y., GU, H., BICKSON, D., AND GUESTRIN, C. Powergraph: Distributed graph-parallel computation on natural graphs.
15. GOOGLE. Tensorflow bucketing. https://www.tensorflow.org/versions/r0.12/api\_docs/python/contrib.training/bucketing.
16. GOOGLE. Tensorflow dynamic rnn. https://www.tensorflow.org/api\_docs/python/tf/nn/dynamic\_rnn.
17. GOOGLE. Tensorflow static rnn. https://www.tensorflow.org/api\_docs/python/tf/nn/static\_rnn.
18. GOOGLE TENSORFLOW XLA. https://www. tensorflow.org/performance/xla/.
19. GUSTAFSON, J. L. Reevaluating amdahl’s law. Communications of the ACM 31, 5 (1988), 532–533.
20. GYSI, T., OSUNA, C., FUHRER, O., BIANCO, M., AND SCHULTHESS, T. C. Stella: A domainspecific tool for structured grid methods in weather and climate models. In High Performance Computing, Networking, Storage and Analysis, 2015 SCInternational Conference for (2015), IEEE, pp. 1–12.
21. HINTON, G., DENG, L., YU, D., DAHL, G. E., MOHAMED, A.-R., JAITLY, N., SENIOR, A.,VANHOUCKE, V., NGUYEN, P., SAINATH, T. N., ET AL. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine 29, 6 (2012), 82–97.
22. HOCHREITER, S., AND SCHMIDHUBER, J. Long short-term memory. Neural computation 9, 8 (1997), 1735–1780.partitioning on skewed graphs. In Proceedings of the Tenth European Conference on Computer Systems(2015), ACM, p. 1.
23. Fitzpatrick B. Distributed caching with memcached[J]. Linux Journal, 2004, 2004(124):págs. 72-76.
24. Ongaro D, Rumble S M, Stutsman R, et al. Fast crash recovery in RAMCloud[C]// ACM Symposium on Operating Systems Principles. ACM, 2011:29-41.
25. Bjornsson H, Chockler G, Saemundsson T, et al. Dynamic performance profiling of cloud caches[C]// Symposium on Cloud Computing. ACM, 2013:59.
26. Kasture H, Sanchez D. Ubik: efficient cache sharing with strict qos for latency-critical workloads[J]. Acm Sigplan Notices, 2014, 49(4):729-742.
27. Cidon A, Eisenman A, Alizadeh M, et al. Dynacache: dynamic cloud caching[C]// Usenix Conference on Hot Topics in Cloud Computing. USENIX Association, 2015:19-19.
28. [28] A. Cidon, A. Eisenman, M. Alizadeh, and S. Katti. Cliffhanger: Scaling performance cliffs in web memory caches. In 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI16), pages 379–392, Santa Clara, CA, Mar. 2016.
29. Aghayev A, Desnoyers P. Log-structured cache: trading hit-rate for storage performance (and winning) in mobile devices[C]// The Workshop on Interactions of Nvm/flash with Operating Systems and Workloads. ACM, 2013:7.
30. Rumble S M, Kejriwal A, Ousterhout J. Log-structured memory for DRAM-based storage[C]// Usenix Conference on File and Storage Technologies. USENIX Association, 2014:1-16.
31. Li C, Cox A L. GD-Wheel: a cost-aware replacement policy for key-value stores[C]// Tenth European Conference on Computer Systems. ACM, 2015:5.
32. Beckmann N, Sanchez D. Bridging Theory and Practice in Cache Replacement[J]. 2015.
33. Stefanovici I, Thereska E, O'Shea G, et al. Software-defined caching: managing caches in multi-tenant data centers[C]// ACM Symposium on Cloud Computing. ACM, 2015:174-181.
34. Cidon A, Rushton D, Rumble S M, et al. Memshare: a Dynamic Multi-tenant Key-value Cache[C]//2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17). USENIX} Association}, 2017: 321-334.