中国科学: 信息科学 2023 年 第 53 卷 第 12 期: 2458–2471
SCIENTIA SINICA Informationis
论文

$\mathrm{TVM_T}$: 基于 TVM 的高性能神经网络训练编译器

曾军[1], 寇明阳[1], 郑惜元[1], 姚海龙[2*], 孙富春[1,3]
1. 清华大学计算机科学与技术系, 北京 100084
2. 北京科技大学计算机与通信工程学院, 北京 100083
3. 北京信息科学与技术国家研究中心, 北京 100084
* 通信作者. E-mail: hailongyao@ustb.edu.cn
收稿日期: 2022–04–06; 修回日期: 2022–09–26; 接受日期: 2023–05–19; 网络出版日期: 2023–12–12
科技创新 2030—"新一代人工智能" 重大项目 (批准号: 2020AAA0104603) 和北京市重点研发计划 (批准号: Z191100007519015) 资助项目

摘要 随着深度学习应用的快速发展, 神经网络模型的参数量变得越来越大, 这意味着训练一个可用的神经网络模型需要更多的算力和更长的计算时间, 因此如何提升神经网络训练的效率至关重要. 然而训练效率在很大程度上取决于硬件后端和编译器. 为了提升神经网络训练的性能, 编译器的效率亟待提升, 而这主要取决于计算图的优化、算子级别的优化和代码生成. 主流的神经网络训练框架 (如 TensorFlow, PyTorch) 使用了供应商特定的、通过手工设计算子获得的算子库. 然而, 手工设计算子浪费了大量的算子级别的优化空间, 因此研究人员提出了 TVM. 作为一个端到端的编译器, TVM 实现了算子级的自动优化, 比现有框架进一步提高了性能. 此外, TVM 支持从多种神经网络框架中导入神经网络模型, 并在不同主流硬件后端上部署. 然而, TVM 的注意力集中于提升神经网络推理任务的性能, 并不支持神经网络的训练. 本文提出了 $\mathrm{TVM_T}$, 第一个基于 TVM、支持神经网络训练的端到端编译器. 为了支持神经网络训练, 本文提出了以下方法. (1) 合并损失函数到现有的计算图中, 以支持前向和后向传播; (2) 在训练过程中采用设备到主机的机制来更新权重参数; (3) 集成了最先进的张量程序调优器, 以自动优化神经网络训练程序. 实验结果表明, 与 PyTorch 相比, $\mathrm{TVM_T}$ 在 Intel CPU 和 NVIDIA GPU 上的神经网络训练性能达到了最高 4.88 倍的提升; 与 TensorFlow 相比, $\mathrm{TVM_T}$ 在 Intel CPU 和 NVIDIA GPU 上的神经网络训练性能达到了最高 11.5 倍的提升.

关键词 神经网络编译器, 神经网络训练, 算子自动优化, 参数更新, 反向传播

1 引言

最近, 深度学习技术发展迅速, 在各种人工智能应用中显示出巨大的性能提升潜力, 如人脸识别 [1]、自动驾驶 [2]、知识图谱系统 [3] 等. 然而, 从头开始构建和训练一个神经网络给人工智能开

引用格式: 曾军, 寇明阳, 郑惜元, 等. $\mathrm{TVM_T}$: 基于 TVM 的高性能神经网络训练编译器. 中国科学: 信息科学, 2023, 53: 2458–2471, doi: 10.1360/SSI-2022-0140
Zeng J, Kou M Y, Zheng X Y, et al. $\mathrm{TVM_T}$: TVM-based high-performance neural network compiler supporting training (in Chinese). Sci Sin Inform, 2023, 53: 2458–2471, doi: 10.1360/SSI-2022-0140

发者带来了很多挑战. 由于用 Python 和 C++ 等通用编程语言描述基本的神经网络计算十分复杂, 因此人们提出了具有专用编程接口的深度学习框架 (如 TensorFlow [4], PyTorch [5], MXNet [6]), 有效促进了神经网络模型的发展. 然而, 高精度的神经网络模型训练依赖基于大量的训练数据进行的长时间训练和调参, 这一过程是计算密集且耗时巨大的, 这对深度学习框架和编译器的性能提出了要求. 高性能的深度学习框架和编译器可以显著减少训练时间, 加速调参从而可以更快地提升神经网络模型精度, 大大加速深度神经网络模型在人工智能应用中的部署.

在现代深度学习框架中 [4~6], 神经网络模型被表示为计算图, 它是有向无环图 (directed acyclic graph, DAG). 计算图中的节点和边分别代表算子 (例如, 矩阵乘法) 和数据依赖关系. 为了提高训练性能和减少消耗的算力, 研究者们提出了一系列的计算图优化技术来改善计算图的结构, 包括经典的常量折叠、死代码消除优化和后端硬件相关的特定优化技术 [7]. 完成计算图优化之后, 图中的算子被映射到供应商特制的算子库 (如 cuDNN, cuBLAS, MKL-DNN 和 oneDNN) 进行相应的底层运算. 供应商特定的算子库是为特定的硬件后端手动设计和调优的, 因此使用这一算子库使得训练性能得到了进一步的提升. 然而, 开发和维护这种供应商特定的算子库是很复杂且耗时的. 同时, 由于手工设计算子无法考虑所有优化可能, 大量的算子级优化空间被浪费了. 为了解决人工设计算子引起的问题, 研究者们提出了深度学习编译器 [8, 9], 它可以自动生成高性能的算子张量程序. 这种算子的自动代码生成机制不仅将开发者从耗时的手动调优任务中解放出来, 还能有效地探索各种算子的庞大优化空间. 其中, TVM [8] 是最具潜力的深度学习编译器之一.

作为一个端到端的神经网络编译器, TVM 以各种神经网络模型作为输入, 针对各种不同的硬件后端生成高性能的张量程序. 为了支持不同算子的自动代码生成, TVM 采用了 Halide 风格的调度原语 [10], 它很好地分离了调度和底层运算. 以逐元素操作张量算子为例, CPU 的调度原语将张量算子转化为一个嵌套循环, 并对给定的张量元素进行迭代操作. 这种从高层次算子描述到底层运算描述的流程就是调度. 基于 TVM 的调度原语, 研究者们提出了不同的自动调度优化算法 [11, 12] 来加快自动代码生成以及提升生成的张量程序的性能. 然而, 因为过于关注提升神经网络推理的性能, TVM 社区并没有对神经网络训练投入太多的开发精力, 因此目前发布的 TVM 软件并不支持神经网络的训练. 本文首先提出了一个高效的编译器来支持基于 TVM 的高性能训练.

在建立一个支持训练和推理的端到端编译器时, 本文面临以下几个挑战. (1) 为了支持训练, 编译器需要正确构建训练计算图, 构建正确且高效的训练计算图是实现高性能训练的关键因素之一. 与推理计算图不同, 训练计算图中包含训练专用算子, 如 dropout [13] 和 batch normalization [14], 这些算子都需要特别实现. (2) 目前 TVM 的编译流程不支持对输入张量的修改, 这意味着就地更新权重参数是不可行的. 因此, 需要提出一种有效的机制, 在训练过程中更新权重参数. (3) 为了提高训练效率, 需要对训练计算进行优化, 而目前的 TVM 流程中没有这种方法.

为了应对这些挑战, 本文提出了 TVM_T, 一个支持神经网络训练的端到端编译器. 为了支持训练, TVM_T 提出了以下三点创新. 首先, TVM_T 提出了一种融合损失函数的统一计算图构建方法, 该方法为使用反向传播算法构建训练计算图打下了良好的基础. 其次, TVM_T 提出了一个设备到主机的权重参数更新机制, 该机制的灵活性有效解决了权重参数无法就地更新的问题. 最后, TVM_T 集成了目前最先进的调度算法 [12] 来优化训练计算图的性能. 表 1 给出了 TVM_T, TVM [8] 和 PyTorch [5] 的对比. 从表 1 中可以看出, 不同神经网络编译框架在是否支持静态计算图、是否支持计算图优化、是否支持自动代码生成以及是否支持多种硬件后端等方面有很大的区别. TVM 同时支持计算图优化和自动代码生成, 并且可以在多种硬件后端上部署神经网络模型. 这满足了支持训练神经网络编译器的优化和多后端支持的需求. 因此, TVM 最适合作为支持训练的神经网络编译器开发的基础框架. TVM

表 1 **PyTorch [5], TVM [8] 和 TVM_T 的对比**

Table 1 A comparison between PyTorch [5], TVM [8] and TVM_T

	PyTorch [5]	TVM [8]	TVM_T
Static computational graph	×	✓	✓
Computational graph optimization	✓	✓	✓
Automatic differentiation	✓	✓	✓
Vendor-specific library	✓	✓	✓
Automatic code generation	×	✓	✓
Training support	✓	×	✓
Special training operators [13, 14]	✓	×	✓
Multi-backend support	×	✓	✓

的优点是支持多种硬件后端以及支持计算图优化和自动代码生成. 然而它的缺点是不支持训练, 并且由于 TVM 基于静态计算图, 开发难度大. 因此, 为了扬长避短, 实现高效的端到端编译器, 我们借鉴了 TVM 的计算图优化和自动代码生成功能, 提出了设备 – 主机这一机制来支持训练, 并解决 TVM 静态计算图更新参数困难的问题.

本文使用标准神经网络模型作为基准程序对 TVM_T 在通用处理器 CPU 和 GPU 上的训练性能进行评估, 基准程序包括 MLP-3 [15], LeNet-5 [16], ResNet-18 [17], SqueezeNet [18] 和 LSTM LM [19]. 实验结果表明, 与 PyTorch 相比, TVM_T 在 Intel CPU 和 NVIDIA GPU 上的神经网络训练性能达到了最高 4.88 倍的提升; 与 TensorFlow 相比, TVM_T 在 Intel CPU 和 NVIDIA GPU 上的神经网络训练性能达到了最高 11.5 倍的提升. 本文的主要贡献如下:

- 本文提出了 TVM_T, 第一个基于 TVM 的支持神经网络训练的端到端编译器.
- 本文实现了包括 dropout 和 batch normalization 在内的训练专用算子, 并提出了一个设备到主机的机制, 以支持高效的神经网络训练.
- 本文将最先进的自动调度优化算法运用到神经网络训练优化中, 大大提升了神经网络训练速度.

本文的其余部分组织如下. 第 2 节介绍了 TVM_T 的系统架构, 其中包括控制器、转换器和运行时模块. 第 3 节介绍了将推理模型转换为训练模型的方法. 第 4 节介绍了在训练期间更新权重参数的设备 – 主机机制. 同时, TVM_T 移植了最先进的自动调度优化算法来优化神经网络训练性能. 第 5 节给出了实验结果. 最后, 第 6 节给出了本文的结论.

2 系统概览

TVM_T 是一个基于 TVM 的支持神经网络训练的端到端编译器. 图 1 给出了 TVM_T 系统架构概览. TVM_T 由 3 个主要模块组成. (1) 控制器, 负责整合训练的编译和执行任务; (2) 转换器, 负责将给定的推理模型转换为训练模型; (3) 运行时执行器, 负责编译、优化和执行训练模型.

2.1 控制器

控制器集成了整个系统, 包括图编译器、转换器、运行时执行器和参数更新器. 控制器集成了 Relay [20] 作为图编译器, 首先将各种神经网络模型转换为相应的计算图. 然后, 控制器将 Relay 生成的计算图传递给转换器进行模型转换, 将推理模型转换为训练模型. 转换后, 训练计算图被送到运行时执行器进行编译优化和执行, 并计算损失函数的梯度. 最后, 控制器获得更新的权重参数并将其发

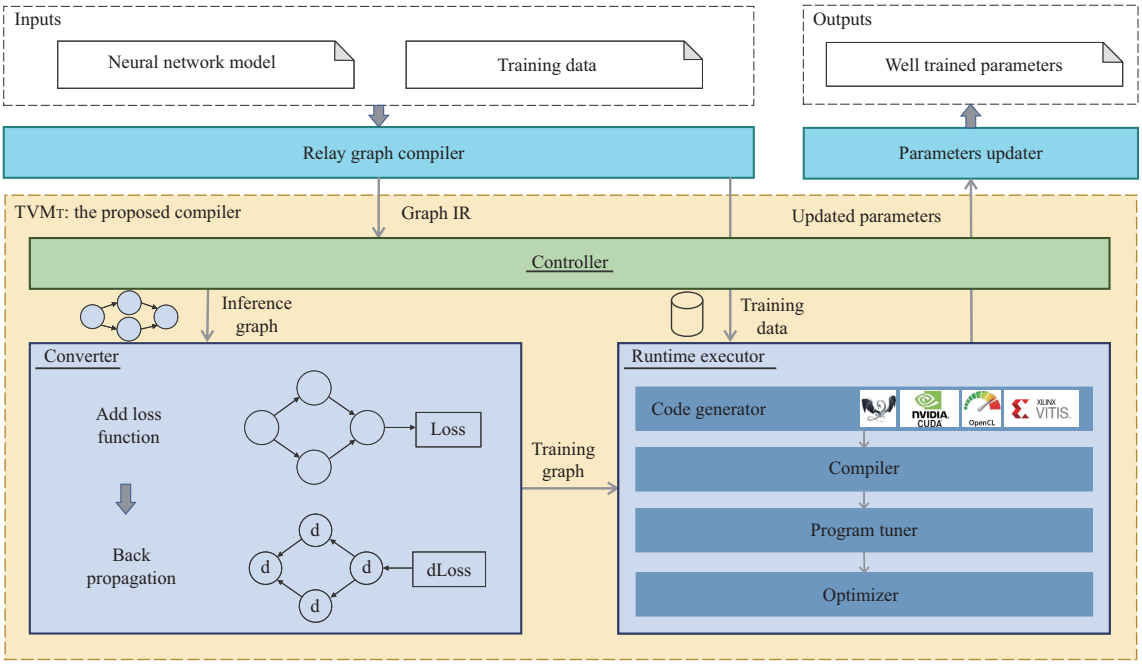

图 1 (网络版彩图) $\mathbf{TVM_T}$ 的系统概览

Figure 1 (Color online) A system overview of $\mathrm{TVM_T}$

送给参数更新器, 参数更新器更新输入神经网络模型的权重参数. 控制器在整合系统中的所有功能模块方面起着关键作用, 正是因为有了控制器, 一个特定的训练任务才可以高效地完成.

2.2 转换器

为了进行有效的训练, 需要正确构建训练计算图. 然而, 构建过程面临着两个主要挑战.

首先, 反向传播过程中需要得到损失函数的梯度并提供给梯度下降算法来进行有效的权重参数更新. 因此, 转换器将损失函数合并到推理计算图中, 并在整合了损失函数的推理计算图上使用反向传播算法构建训练计算图. 图 2 描述了一个训练计算图的构建流程. 左边是原始推理计算图, 右边是相应的训练计算图. 从图 2 中可以看出, 训练计算图是由一个前向过程和一个后向过程组成的. 前向过程与推理计算图相同, 而后向过程是由反向传播算法构建的 [21].

其次, 计算图中算子的正确性对于有效的训练至关重要. 然而, 某些神经网络算子, 如 dropout [13] 和 batch normalization [14] 在推理和训练中表现不同. 如果不能正确处理这些差异, 训练的效果将受到很大影响. 在训练过程中, dropout 算子通过将神经网络权重设置为 0, 随机丢弃一些隐藏层的神经元, 有效地解决了神经网络训练的过拟合问题. 然而, 由于 dropout 算子是被专门设计用来解决训练过拟合问题的, 它在推理过程中没有任何作用, 可以被安全地删除.

至于 batch normalization, 它在训练和推理之间的区别更为复杂. batch normalization 通过

$$y_i = \gamma \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta. \tag{1}$$

对一批数据进行归一化, 其中 γ 和 β 是可训练参数, ϵ 是防止分母为零的超参数, x_i 和 y_i 分别表示原始数据和归一化之后的数据. μ 是该批数据的平均值, σ^2 是该批数据的方差. 在训练过程中, batch normalization 会根据当前输入的一批数据计算平均值和方差. 然后, 计算出的均值和方差被用来对该

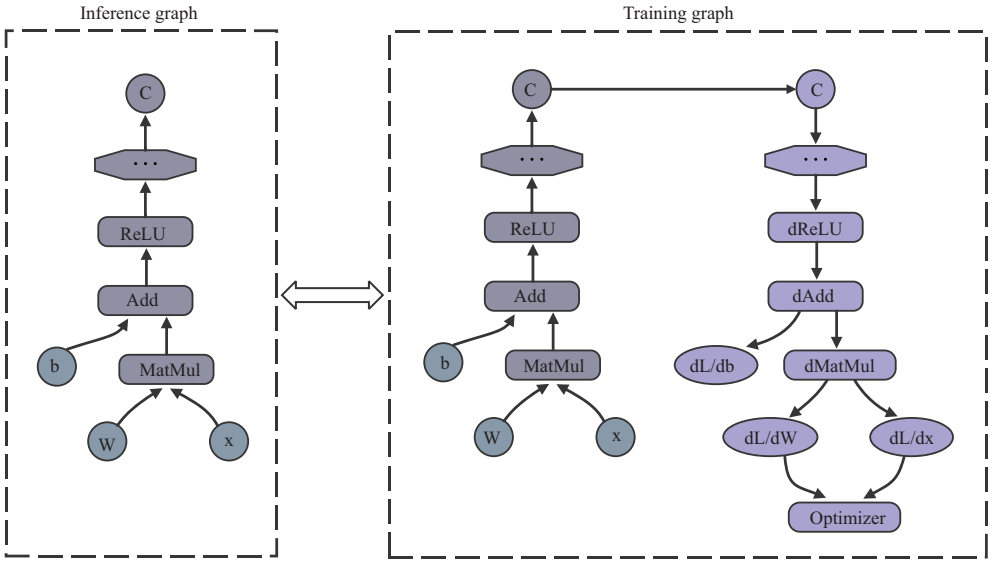

图 2　(网络版彩图) 推理计算图到训练计算图的转换

Figure 2　(Color online) The transformation of an inference graph to a training graph

批数据进行归一化. 相比之下, 与训练过程每次输入一批数据, 一批数据通常包含很多个数据不同, 推理过程通常一批只有一个数据. 在这种情况下, 不存在均值和方差的概念, 因为该批数据只有一个. 为了归一化单个数据, 均值和方差被作为固定参数传入. 这些固定的均值和方差是根据整个训练数据集事先计算出来的. 为了解决上述两个算子在训练和推理过程中表现不同的问题, TVM$_T$ 在转换器中特别设计并实现了 dropout 和 batch normalization 方法.

通过将损失函数合并到推理计算图中并设计特殊的训练算子, 转换器正确地构建了训练计算图, 从而实现了神经网络模型的有效训练. 转换器的实现细节将在第 3 节中讨论.

2.3　运行时执行器

执行训练计算图和更新权重参数的性能对于高效训练都是至关重要的. 为了高效地执行训练计算图, 运行时执行器集成了最先进的算子自动调度优化算法 [12], 该算法为训练计算图中的算子自动生成了高性能的张量程序. 对于权重参数更新, 目前 TVM 的编译流程不支持对输入张量的修改, 因为 TVM 将输入张量表示为常量. 因此, 在 TVM 中就地更新权重参数是不可行的, 这极大地影响了训练效率. 为了解决更新权重参数的问题, 本文提出了一种设备到主机的权重更新机制. 基于所提出的机制, 权重参数在主机端被更新, 这解决了上述 TVM 的缺点. 此外, 灵活的设备到主机的机制很容易整合不同的权重参数更新算法, 这有助于进一步加快训练过程.

总的来说, 本文提出的设备到主机的权重参数更新机制以及针对最先进的算子自动调度优化算法的集成大大提升了神经网络的训练效率. 运行时执行器的细节将在第 4 节中讨论.

3　模型转换

为了支持有效的训练, 训练计算图需要被正确构建. 首先, 转换器将损失函数合并到推理计算图中, 这样就可以在反向传播期间计算损失函数的梯度. 基于计算出的梯度, 梯度下降算法可以有效地更

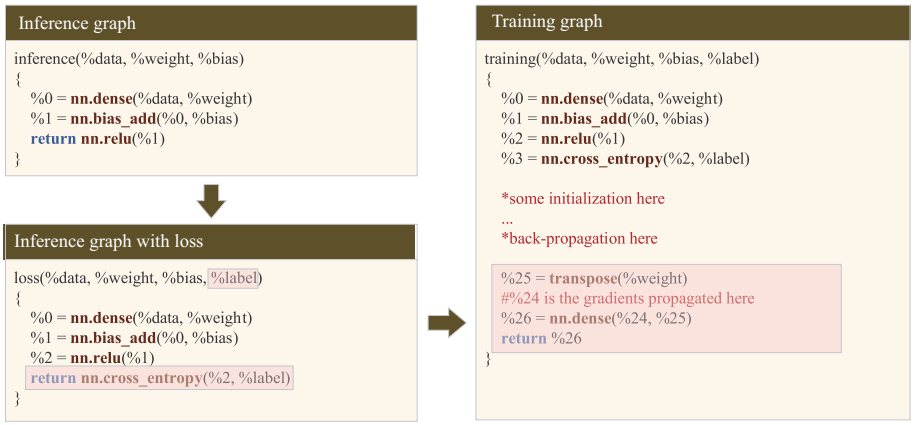

图 3 (网络版彩图) 一个推理计算图转化为训练计算图的例子

Figure 3 (Color online) An example of converting an inference graph to a training graph

新权重参数. 其次, 由于特殊的训练算子及其梯度形式是支持训练的主要障碍之一, 我们精心设计了特殊的训练算子, 如 dropout 和 batch normalization, 以及转换器所使用的这些算子的梯度形式.

3.1 训练计算图构建

图 3 展示了一个训练计算图的构造流程. 由于篇幅有限, 这里简化了训练计算图的代码片段. 在图 3 中, 以 "%" 为前缀的常数代表局部变量, 这些变量只在当前的函数范围内有作用. 构建流程包括 3 个步骤. 首先, 推理图以类似函数的文本格式呈现, 其中输入数据作为函数的参数, 算子被建模为一系列的函数调用. 其次, 损失函数被整合到推理计算图中, 用来衡量预测值和标签值之间的差距. 最后, 反向传播算法被应用到带有损失函数的推理计算图上. 如图 3 中训练计算图的代码片段所示, 训练计算图首先执行前向过程, 然后根据前向过程的输出执行采用反向传播算法生成的后向过程. 在训练计算图执行完成之后, 权重参数的梯度被返回给参数更新模块, 参数更新模块根据梯度下降算法更新权重参数.

3.2 算子设计

在训练和推理过程中, dropout 算子和 batch normalization 算子的行为是不同的. 如果不区分这些算子在训练和推理中的不同行为, 直接将算子的推理形式应用到训练任务中, 整个训练任务会受到极大影响, 甚至可能直接失败. 为了解决这个关键问题, 转换器将这些训练算子设计成训练和推理的多功能形式.

dropout 算子. dropout 算子是为了防止神经网络在训练过程中过拟合而提出的一个算子, 其实现细节在算法 1 和 2 中给出. attribute 参数被用来区分推理和训练, 当属性以 "推理" 的形式传入时, 前向过程直接返回原始张量, 因为 dropout 算子在推理中没有意义. 相反, 当属性是 "训练" 时, 前向过程根据丢弃参数 rate 计算一个掩码, 如 rate = 0.5 表示希望丢弃 50% 的神经网络连接. 然后将该掩码应用于原始张量, 得到一个被掩码掩盖之后的张量, 该张量则是丢弃部分神经网络连接之后的输出. 与前向过程相比, 反向过程要简单得多, 反向过程中采用与前向过程相同的掩码来获得掩码梯度. 接着, 这些掩码梯度被反向传播算法用于计算前部神经网络层的参数梯度.

本文提出的 dropout 算法有两个特点. 首先, 基于 TVM 的编译技术, dropout 算子的随机掩码根据给定的随机种子保持不变. 因此, TVM 中的随机掩码在每次训练迭代中只能丢弃同一组神经网络

算法 1 前向 dropout 算子

Input: original_tensor, random_mask, rate, attribute.
Output: masked_tensor.
1: **if** attribute is inference **then**
2: **return** original_tensor;
3: **else if** attribute is training **then**
4: **for all** element in random_mask **do**
5: **if** element $<$ rate **then**
6: element = 0;
7: **else**
8: element = $\frac{\text{element}}{1-\text{rate}}$;
9: **end if**
10: **end for**
11: **return** original_tensor $\circ$ random_mask.
12: **end if**

算法 2 反向 dropout 算子

Input: random_mask, grad.
Output: new_grad.
1: **return** grad $\circ$ random_mask.

连接. 然而, dropout 算法需要在每次训练迭代中使用随机掩码来停用随机的连接, 而不是同一组连接. 不可改变的随机掩码极大地影响了 dropout 算法的功能, 因此降低了最终的训练精度. 通过将随机掩码建模为可在每次迭代中更新的输入参数, 转换器有效地解决了恒定随机掩码的问题, 从而保证了 dropout 算子的有效性. 其次, 算法 2 重用了算法 1 中生成的随机掩码, 这避免了随机掩码的重复生成, 提升了 dropout 算子的执行性能.

batch normalization 算子. batch normalization 是一个旨在解决内部协变量偏移现象的算子, 它指的是训练过程中参数分布的变化. 同样的, 算法细节在算法 3 和 4 中给出. 与 dropout 算子类似, 算法定义了 attribute 属性来区分推理和训练. 在推理过程中, 算法根据输入的均值和方差对输入的数据进行归一化. 在训练过程中, 算法根据当前输入的数据批计算均值和方差. 至于后向过程, 由于只有 γ 和 β 是可学习的参数, 算法根据文献 [14] 中的数学推导计算 dγ 和 dβ. 值得注意的是, 前向过程中的归一化数据被重复使用, 作为反向过程的输入, 这可以避免重复的归一化计算, 提升 batch normalization 算子的执行性能.

4 模型执行与参数更新

在模型转换之后, 推理计算图被转换为训练计算图. 训练计算图的执行性能对整个训练性能至关重要. 然而, 高效的神经网络训练有两个主要挑战. 首先, 训练计算图中的算子在执行前需要被翻译成张量程序. 算子的不同翻译方式在性能上有很大的差别. 因此, 恰当的算子翻译方式对于整体训练性能至关重要. 运行时执行器将最先进的自动调度算法集成到训练计算图翻译过程中, 用于生成高性能的张量程序. 其次, 权重参数的更新是神经网络训练中的基本步骤, 准确快速地更新算法对于有效和高效的训练至关重要. 然而, 目前 TVM 的编译流程并不支持对输入张量的修改, 这意味着在原地更新权重参数是不可行的, 这使得 TVM 无法完成训练任务. 因此, 本文提出一种设备到主机的机制来有效地

算法 3 前向 batch normalization 算子

Input: $B = x_{1...m}$, γ, β, ϵ, μ, σ, attribute.
Output: $\hat{B} = \hat{x}_{1...m}$
if attribute is inference **then**
  return $\gamma \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$;
else if attribute is training **then**
  $\mu = \text{Mean}(B)$;
  $\sigma^2 = \text{Variance}(B)$;
  return $\gamma \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$;
end if

算法 4 反向 batch normalization 算子

Input: $\hat{B} = \hat{x}_{1...m}$, $\text{grad} = y_{1...m}$.
Output: $\text{d}\gamma, \text{d}\beta$.
$\text{d}\beta = \sum_{i=1}^{m} y_i$;
$\text{d}\gamma = \sum_{i=1}^{m} y_i * \hat{x}_i$;
return $\text{d}\gamma, \text{d}\beta$.

更新权重参数.

4.1 神经网络模型编译与执行

恰当的算子翻译方式对于训练计算图的高性能执行至关重要. 运行时执行器将最先进的自动调度算法 Ansor [12] 用于算子的翻译和优化. 然而, Ansor 是为翻译优化推理计算图中的算子而设计的, 并且之前没有工作证明 Ansor 在训练计算图中也能有效工作. 下面本文将对推理程序和训练程序的区别进行详细分析, 并证明在训练计算图中采用 Ansor 进行优化的可行性. 图 3 描述了推理计算图和相应的训练计算图的算子级细节. 从图中可以看出, 推理计算图是由 TVM 的算子库中的若干个算子 (如 nn.dense, nn.bias_add 和 nn.relu) 组成的 [22]. TVM 的算子库 (TVM Operator Inventory, TOPI) 是一个算子定义的集合, 它描述了如何将算子翻译到正确和高效的低层次张量程序. 例如, 如果不对矩阵乘法算子进行任何优化, TOPI 将把该算子翻译成一个三层嵌套循环程序. 同样的, 训练计算图也由 TOPI 中的几个算子组成. 因此, TOPI 也适用于训练计算图, 并将以与推理计算图相同的方式生成低层次张量程序. 通过整合 Ansor 到训练计算图优化程序中, TVM_T 的神经网络训练性能得到了大幅度提升.

4.2 设备到主机的参数更新机制

快速准确的权重更新算法对高效且有效的神经网络训练同样至关重要. 然而, 目前 TVM 的编译流程并不支持对输入张量 (即神经网络权重参数) 的修改. 无法更新参数意味着无法完成神经网络训练任务, 这给基于 TVM 实现神经网络训练带来了巨大挑战. 为了解决这个问题, 本文提出了一种基于设备 – 主机机制的训练计算图, 该计算图通过在设备上运行训练计算图得到权重参数的梯度, 再在主机上运行基于梯度下降方法的优化器更新权重参数. 图 4 描述了本文提出的设备 – 主机机制的概念, 其中标蓝部分在设备端执行, 标绿部分在主机端执行. 如图 4 所示, 控制器是整个基于设备 – 主机机制的训练计算图的中心, 负责控制整个训练流程的执行. 首先, 训练计算图完成前向推理和反向传播, 得到权重参数相对于损失函数的梯度 $\nabla_\theta L(\theta)$ (其中 θ 是权重参数, L 是损失函数) . 接着, 在主机端的控制器接收这些权重梯度, 并传递给同样在主机端的优化器. 优化器基于梯度下降方法, 根据权重梯度更新神经网络参数, 并在更新参数完毕之后将新权重传递给控制器. 至此, 控制器完成了一个训练

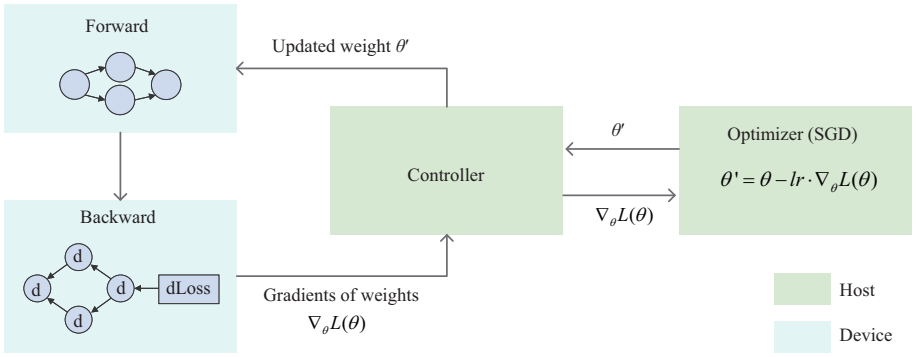

图 4 (网络版彩图) 所提出的基于设备 – 主机机制的训练计算图. 其中标蓝部分在设备端执行, 标绿部分在主机端执行

Figure 4 (Color online) Proposed training computational graph based on the device-host mechanism. The blue part is executed at the device side, and the green part is executed at the host side

迭代周期, 并在下一个训练迭代周期开始时将更新后的权重参数传递给设备上的训练计算图进行训练迭代. 如此往复, 直到满足训练次数或精度要求时停止训练. 所提出的基于设备 – 主机机制的训练计算图将梯度计算与参数更新分离, 有效地解决了 TVM 不能在设备上原地更新权重的问题, 实现了基于 TVM 的神经网络训练.

5 实验结果

TVM_T 是基于 TVM [8] (版本 0.8.dev0), 用 Python 实现前端的端到端编译器. 为了支持训练, 我们用 C++ 实现了 dropout 和 batch normalization 算子, 并集成到 TVM_T 中. 下面本文将从训练的可行性和效率两方面对 TVM_T 进行实验验证. 首先, 本文研究了在 TVM_T 的训练过程中损失值的变化. 接着, 本文将 TVM_T 与流行深度学习框架 PyTorch [5] 和 TensorFlow [4] 进行比较.

实验在两个硬件平台上进行: Intel 服务器级 CPU (Xeon(R) Gold 6271C CPU @ 2.60 GHz) 和 NVIDIA GPU (RTX 2080Ti), 并且所有的实验都以 float32 作为通用数据类型.

5.1 训练可行性

我们使用 MLP-3 [15] 和 LeNet-5 [16] 在 MNIST 数据集 [16] 上评估了 TVM_T 的训练效果. 在本小节中, 训练批次大小被设定为 256, SGD 被用作优化器. 图 5 描述了 TVM_T 进行的训练任务的损失曲线. X 轴表示训练迭代周期, 其中每批数据都被算作一次迭代, 另外 Y 轴表示损失值. 如图 5 所示, 两个训练任务的损失值随着迭代次数的增加而下降, 这意味着 TVM_T 成功地在 MNIST 分类任务中完成了上述两个模型的训练.

5.2 训练效率

本小节用通用的标准神经网络模型作为基准程序对 TVM_T 在通用处理器 CPU 和 GPU 上的端到端训练性能进行评估, 基准程序包括 MLP-3 [15], LeNet-5 [16], ResNet-18 [17], SqueezeNet [18] 以及 LSTM LM [19]. 用于测试的 CPU 和 GPU 型号分别为 Intel (Xeon(R) Gold 6271C CPU @ 2.60 GHz) 和 NVIDIA GPU (RTX 2080Ti).

PyTorch [5] 和 TensorFlow [4] 被选为基准框架, 其中 TensorFlow 的版本为 1.7.0+CUDA9.0, PyTorch

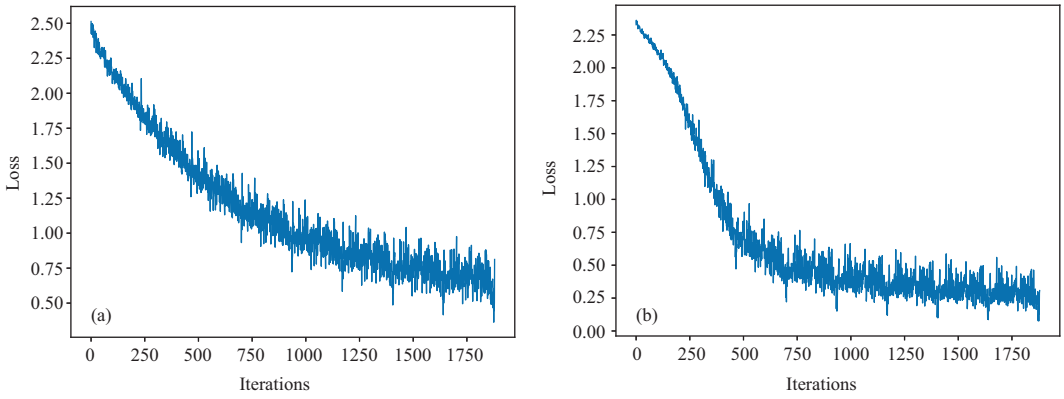

图 5 (网络版彩图) 损失函数变化曲线. (a) 用 MNIST 数据集训练的 MLP-3 和 (b) 用 MNIST 数据集训练的 LeNet-5

Figure 5 (Color online) Loss curve. (a) MLP-3 trained on MNIST dataset and (b) LeNet-5 trained on MNIST dataset

的版本为 1.10.2+CUDA10.1. 实验中, 在 3 个神经网络训练平台 (PyTorch, TensorFlow, TVM_T) 上, 我们将每个基准神经网络模型训练 10 个 epoch, 并取后 9 个 epoch 的平均训练时间作为最终的训练时间, 其中第一个 epoch 的时间则视作启动 epoch 而丢弃. 对于层数较深的神经网络模型 (如 ResNet-18 和 SqueezeNet), 我们报告了批处理量为 16 的结果, 而对于相对层数较浅的网络 (如 MLP-3, LeNet-5 和 LSTM LM), 我们则报告了批处理量为 256 的结果.

图 6 显示了在 Intel CPU 和 NVIDIA GPU 上的实验结果. 如图 6(a) 所示, 针对 Intel CPU, TVM_T 在所有的基准神经网络模型上的训练性能都优于 PyTorch 和 TensorFlow, 并且达到了最高 11.5 倍的性能提升. 如图 6(b) 所示, 针对 NVIDIA GPU, TVM_T 在基准神经网络模型上的训练性能都优于 PyTorch 和 TensorFlow, 并且达到了最高 4.88 倍的性能提升.

此外, XLA [23] 和 TorchScript [24] 分别是 TensorFlow 和 PyTorch 可以在训练过程中使用的神经网络编译器. 针对 XLA 和 TorchScript, 我们在与 TensorFlow 和 PyTorch 相同的实验设置下测试了这两者的端到端训练性能.

图 7 显示了在 Intel CPU 和 NVIDIA GPU 上的实验结果. 如图 7(a) 所示, 针对 Intel CPU, TVM_T 在所有的基准神经网络模型上的训练性能都优于 XLA 和 TorchScript, 并且达到了最高 6.7 倍的性能提升. 如图 7(b) 所示, 针对 NVIDIA GPU, TVM_T 在所有的基准神经网络模型上的训练性能都优于 XLA 和 TorchScript, 并且达到了最高 5.25 倍的性能提升.

5.3 参数更新时间

在所提出的设备到主机的参数更新机制之下, 参数更新在主机端完成, 这会带来一定的通信和计算开销. 本小节用通用的标准神经网络模型对参数更新时间在 TVM_T 的总体训练时间中的所占比例进行测试. 测试的硬件平台选择了通用处理器 CPU 和 GPU, 优化器选择了 SGD, 所有的训练参数设置均与 5.2 小节中相同.

图 8 给出了在总体训练时间中, 参数更新时间与模型时间所占的比例. 如图 8(a) 所示, 针对 Intel CPU, 参数更新时间平均占总体训练时间的 18.2%. 并且, 对于较大的模型 (如 ResNet-18), 模型执行时间占总体训练时间的绝大部分. 而对于较小的模型 (如 MLP-3), 参数更新时间在总体训练时间中占有一定比例, 但是要少于模型执行时间. 如图 8(b) 所示, 针对 NVIDIA GPU, 参数更新时间平均占总

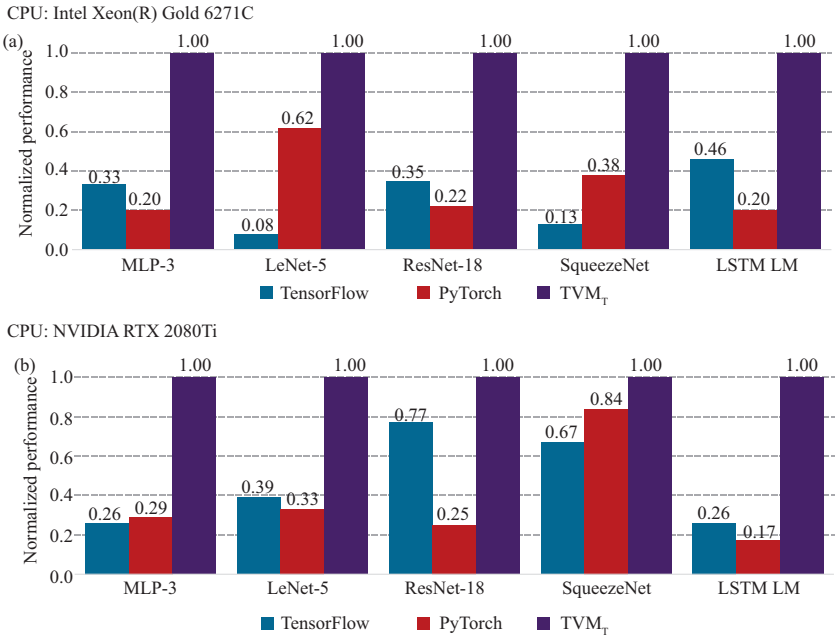

图 6 (网络版彩图) 针对 **PyTorch** 与 **TensorFlow** 的神经网络模型训练效率对比. Y 轴是根据较快的训练速度归一化的结果

Figure 6 (Color online) Comparison of training efficiency on network models over PyTorch and TensorFlow. The Y-axis is the normalized execution time relative to the fastest one for each network. (a) Intel CPU; (b) NVIDIA GPU

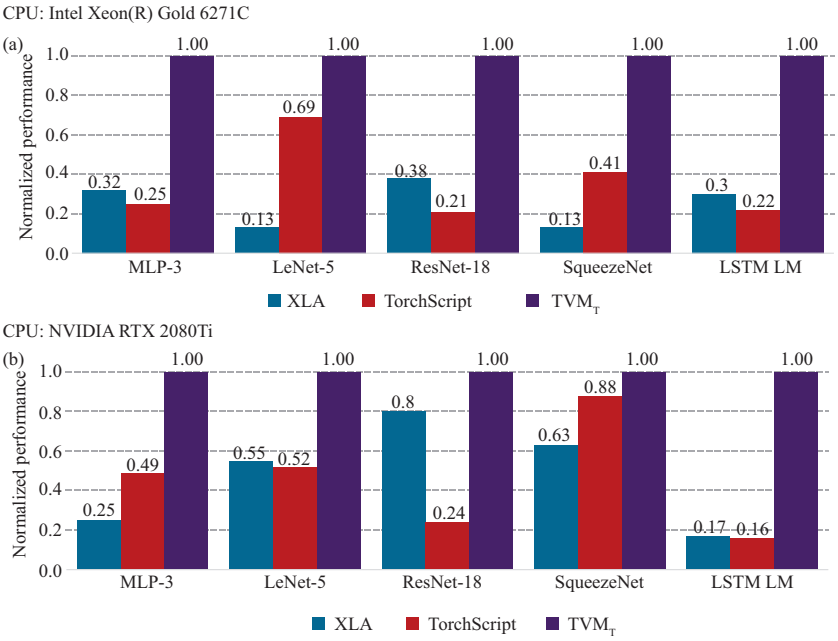

图 7 (网络版彩图) 针对 **XLA** 与 **TorchScript** 的神经网络模型训练效率对比. Y 轴是根据较快的训练速度归一化的结果

Figure 7 (Color online) Comparison of training efficiency on network models over XLA and TorchScript. The Y-axis is the normalized execution time relative to the fastest one for each network. (a) Intel CPU; (b) NVIDIA GPU

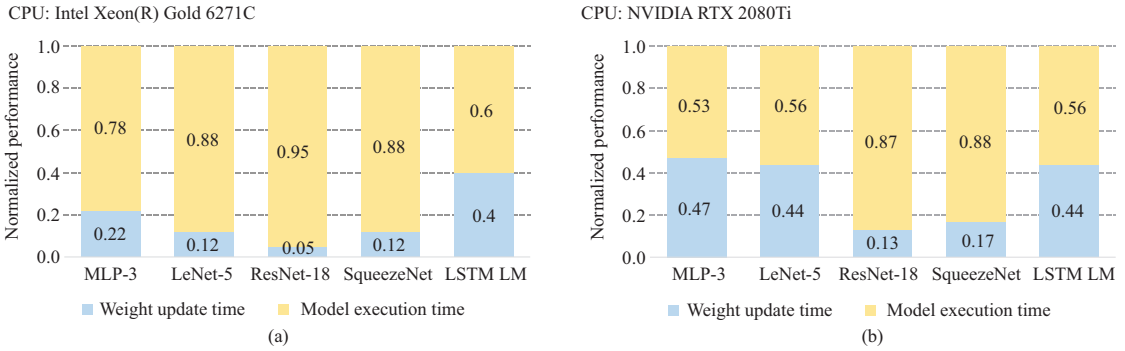

图 8 (网络版彩图) 参数更新时间与模型执行时间对比. Y 轴是根据较快的训练速度归一化的结果

Figure 8 (Color online) The comparison between weight update time and model execution time. The Y-axis is the normalized execution time relative to the fastest one for each network. (a) Intel CPU; (b) NVIDIA GPU

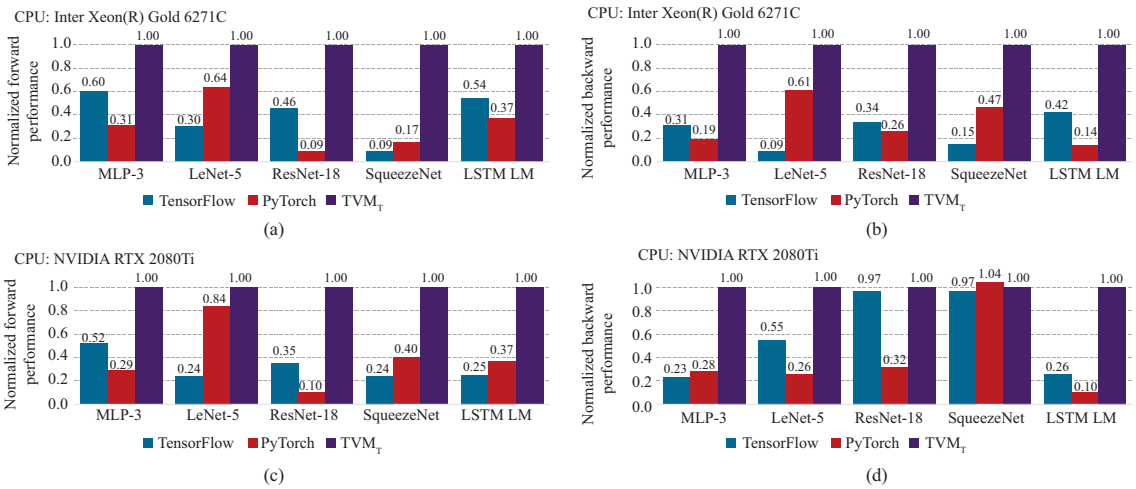

图 9 (网络版彩图) 在两个平台上, 单独比较 TVM$_T$, TensorFlow 与 PyTorch 前向推理与反向传播的速度. Y 轴是根据较快的训练速度归一化的结果

Figure 9 (Color online) The comparison for the speed of TVM$_T$, TensorFlow, and PyTorch on forward and backward processes, respectively. The Y-axis is the normalized execution time relative to the fastest one for each network. (a) Intel CPU forward process; (b) Intel CPU backward process; (c) NVIDIA GPU forward process; (d) NVIDIA GPU backward process

体训练时间的 33%. 参数更新时间与模型执行时间的比例关系与 Intel CPU 类似, 大模型中模型执行时间占主导, 小模型中参数更新时间与模型执行时间均占有一定比例.

5.4 前向推理与反向传播时间

训练计算图整体执行流程由前向推理和反向传播构成. 为了更好地理解 TVM$_T$ 相对于 TensorFlow 与 PyTorch 的优化, 本小节针对这 3 个神经网络训练工具做了拆解测试, 分别比较 TVM$_T$ 在前向推理和反向传播阶段比 TensorFlow 和 PyTorch 的性能优势. 由于 TVM$_T$ 生成的是静态训练计算图, 前向推理和反向传播的执行流程是一体的, 因此通过测试整体训练时间和前向推理时间, 再作差得到反向传播时间.

图 9 给出了 TVM$_T$ 在前向推理阶段和反向传播阶段与 TensorFlow 和 PyTorch 的性能比较. 如

图 9 所示, 在前向推理阶段, TVM_T 在所有模型上都优于 TensorFlow 和 PyTorch, 并在 Intel CPU 上达到了最高 10.1 倍的性能提升, 以及在 NVIDIA GPU 上最高 9 倍的性能提升. 在反向传播阶段, TVM_T 在除 SqueezeNet 的所有模型上都优于 TensorFlow 和 PyTorch, 但优化效果没有前向推理阶段明显.

6 总结

本文提出了 TVM_T, 一个用于神经网络模型训练和推理的端到端编译器. TVM_T 分别提出了一种有效的训练计算图构造方法和一种高效准确更新权重参数的设备 – 主机机制, 以及整合了最先进的张量程序调优算法以有效执行训练计算图, 最终实现了不同神经网络模型的高性能训练. 实验结果表明, 与 PyTorch 相比, TVM_T 在 Intel CPU 和 NVIDIA GPU 上的神经网络训练性能达到了最高 4.88 倍的提升; 与 TensorFlow 相比, TVM_T 在 Intel CPU 和 NVIDIA GPU 上的神经网络训练性能达到了最高 11.5 倍的提升. 未来的工作包括进一步提高 TVM_T 在不同批处理量下的训练性能, 基于就地权重更新的方法进一步提高权重参数更新的效率, 实现设备上的信息存取以支持复杂优化器, 以及扩展 TVM_T 以实现在不同硬件后端上的高性能神经网络训练.

参考文献

1 Parkhi O M, Vedaldi A, Zisserman A. Deep face recognition. In: Proceedings of the British Machine Vision Conference, Swansea, 2015

2 Cordts M, Omran M, Ramos S, et al. The Cityscapes dataset for semantic urban scene understanding. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, 2016. 3213–3223

3 Ji S, Pan S, Cambria E, et al. A survey on knowledge graphs: representation, acquisition, and applications. IEEE Trans Neural Netw Learn Syst, 2022, 33: 494–514

4 Abadi M, Barham P, Chen J, et al. TensorFlow: a system for large-scale machine learning. In: Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, Savannah, 2016. 265–283

5 Paszke A, Gross S, Massa F, et al. PyTorch: an imperative style, high-performance deep learning library. In: Proceedings of Annual Conference on Neural Information Processing Systems, Vancouver, 2019. 8024–8035

6 Chen T, L M, Li Y, et al. MXNet: a flexible and efficient machine learning library for heterogeneous distributed systems. 2015. ArXiv:1512.01274

7 Chen J X, Lin G Q, Chen J X, et al. Towards efficient allocation of graph convolutional networks on hybrid computation-in-memory architecture. Sci China Inf Sci, 2021, 64: 160409

8 Chen T, Moreau T, Jiang Z, et al. TVM: an automated end-to-end optimizing compiler for deep learning. In: Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation, Carlsbad, 2018. 578–594

9 Lattner C, Pienaar J, Amini M, et al. MLIR: a compiler infrastructure for the end of Moore's law. 2020. ArXiv:2002.11054

10 Ragan-Kelley J, Barnes C, Adams A, et al. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In: Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation, Seattle, 2013. 519–530

11 Chen T, Zheng L, Yan E, et al. Learning to optimize tensor programs. In: Proceedings of Annual Conference on Neural Information Processing Systems, Montréal, 2018. 3393–3404

12 Zheng L, Jia C, Sun M, et al. Ansor: generating high-performance tensor programs for deep learning. In: Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation, 2020. 863–879

13 Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res, 2014, 15: 1929–1958

14 Ioffe S, Szegedy C. Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd International Conference on Machine Learning, Lille, 2015. 448–456

15 Pal S K, Mitra S. Multilayer perceptron, fuzzy sets, and classification. IEEE Trans Neural Netw, 1992, 3: 683–697

16 Lecun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition. Proc IEEE, 1998, 86: 2278–2324
17 He K, Zhang X, Ren S, et al. Deep residual learning for image recognition. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, 2016. 770–778
18 Iandola F N, Han S, Moskewicz M W, et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size. 2016. ArXiv:1602.07360
19 Zaremba W, Sutskever I, Vinyals O. Recurrent neural network regularization. 2014. ArXiv:1409.2329
20 Roesch J, Lyubomirsky S, Weber L, et al. Relay: a new IR for machine learning frameworks. In: Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, Philadelphia, 2018. 58–68
21 Rumelhart D E, Hinton G E, Williams R J. Learning representations by back-propagating errors. Nature, 1986, 323: 533–536
22 Ehsan M K. Introduction to TOPI. [2021-04-23]. https://tvm.apache.org/docs/tutorials/topi/intro_topi.html
23 Sabne A. XLA: compiling machine learning for peak performance. 2020. https://research.google/pubs/pub50530
24 DeVito Z, Ansel J, Constable W, et al. Using Python for model inference in deep learning. 2021. ArXiv:2104.00254

$\mathbf{TVM_T}$: TVM-based high-performance neural network compiler supporting training

Jun ZENG[1], Mingyang KOU[1], Xiyuan ZHENG[1], Hailong YAO[2*] & Fuchun SUN[1,3]

1. *Department of Computer Science and Technology, Tsinghua University, Beijing* 100084*, China*;
2. *School of Computer & Communication Engineering, University of Science and Technology Beijing, Beijing* 100083*, China*;
3. *Beijing National Research Center for Information Science and Technology, Beijing* 100084*, China*
* Corresponding author. E-mail: hailongyao@ustb.edu.cn

Abstract With the rapid development of deep learning applications, the number of parameters for neural network models grows dramatically, meaning that more computing power and longer computation time are required to train a usable neural network model. Therefore, it is crucial to accelerate the training process of neural networks. To enhance the performance of neural network training, the compiler's effectiveness, which mainly depends on computational graph optimization, operator-level optimization, and code generation, must be improved. Mainstream training frameworks (e.g., TensorFlow and PyTorch) provide vendor-specific operator libraries obtained from manual operator design. However, a large operator-level optimization space is wasted because of the manual operator design. Therefore, TVM has been proposed. As an end-to-end compiler, TVM enables automatic operator-level optimization and thus further enhances the performance compared to the existing frameworks. Moreover, TVM supports deploying different neural network models on a wide range of hardware backends. Unfortunately, TVM only focuses on the inference performance and does not support the training of neural networks. This paper presents $\mathrm{TVM_T}$, the first end-to-end compiler based on TVM for neural network training. To support neural network training, the paper presents the following methods: (1) The loss function is merged into an existing computational graph to support forward and backward propagations; (2) the device-to-host mechanism is adopted to update the weight parameters during the training process; (3) and a state-of-the-art tensor program tuner is integrated to automatically generate and optimize the program for the training process. Experimental results show that compared with PyTorch and TensorFlow, $\mathrm{TVM_T}$ improves the training performance of popular deep neural networks on the Intel CPU and NVIDIA GPU by up to 4.88-fold and 11.5-fold, respectively.

Keywords neural network compiler, neural network training, automatic optimization for operators, parameters update, back propagation