深度学习DSL简介

目录

论文基本信息

- 题目：Design and implementation of DeepDSL: A DSL for deep learning
- 作者：Tian Zhao, Xiaobing Huang
- 单位：University of Wisconsin
- 期刊：Computer Languages, Systems & Structures, 2018

- 代码仓库：https://github.com/deepdsl/deepdsl

章节1. Introduction

- 针对目前的工具/框架存在的问题
  - 计算图不利于针对多级抽象做优化，对库进行基于图遍历的启发式优化往往不是最优的
  - 计算图不支持用户级别的访问，用户难以自定义DL应用，难以调试运行时错误
  - 存在很多的软件依赖，限制了DL应用的可移植性
- 开发了一个内嵌在Scala中的深度学习DSL
  - 用表达式来表示DL网络：用tensor函数定义DL层，用函数组合定义DL网络；基于重写规则对表达式进行转换，从而进行梯度求导和优化
  - 可以静态检查错误，例如不正确的网络组合（类型错误），可以做内存使用统计
  - 编译成Java源码，具有可读性、可移植性

章节2. Overview

- 整体架构

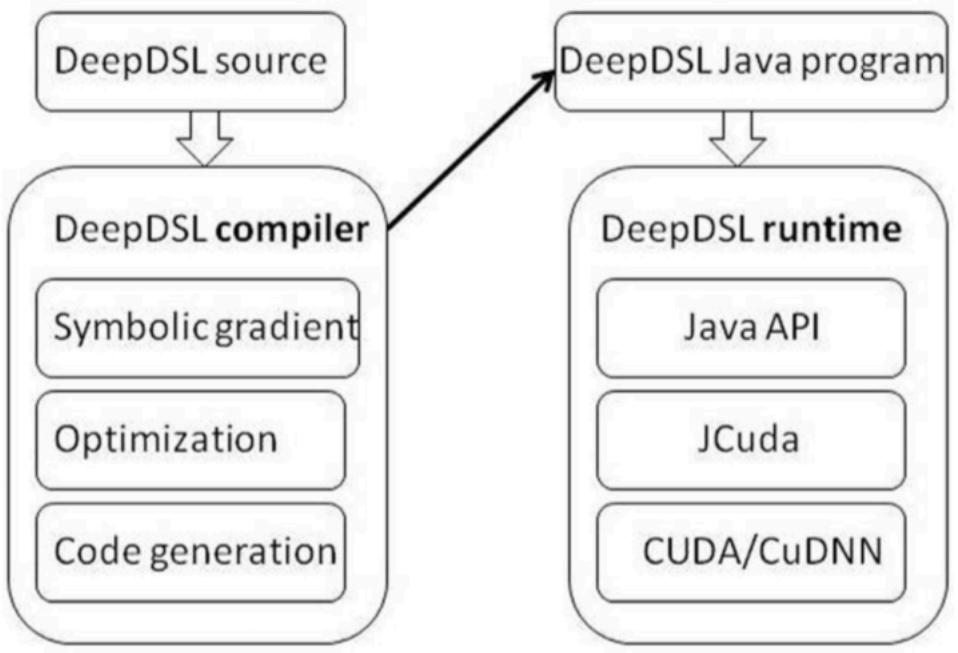


Fig. 1. DeepDSL architecture, where the compiler and runtime are completely separate.

2.2. Tensor

- DeepDSL的核心概念是tensor，用Vec类型来表示，一个Vec对象具有一个维度（Dim类型）数组

```
1 val x = T._new(N, M1)
2 val w = T._new(M2, M1)
3
4 T.sum(M1, k => x(i, k) * w(j, k))
```

Listing 1. Sum of tensor expression.

定义2个2-D tensor：x，w，其中N，M1，M2是维度

在M1维度进行求和，其中k是M1的索引，sum的结果为一个标量

x $\begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$

w $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$

sum 1

假设：N = 1, M1 = 3, M2 = 4

sum = x(i, 0) * w(j, 0) + x(i, 1) * w(j, 1) + x(i, 2) * w(j, 2)

假设：i = 0, j = 0

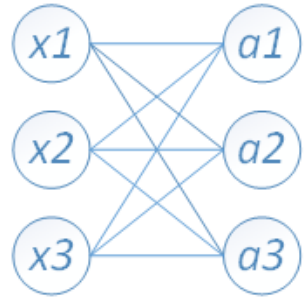
代码实现：deepdsl-java/src/main/scala/deepdsl/ast/Vec.scala

全连接层的tensor

- 返回一个tensor表达式，维度分别为N，M2
- 索引 (i, j) 对应元素的值为一个sum

```
T.vec(N, M2, (i, j) =>
  T.sum(M1, k => x(i, k) * w(j, k)) + b(j)
)
```

Listing 2. Fully connected layer.



其中，x1、x2、x3为全连接层的输入，a1、a2、a3为输出，

$$
\begin{aligned}
a1 &= W_{11} * x1 + W_{12} * x2 + W_{13} * x3 + b_1 \\
a2 &= W_{21} * x1 + W_{22} * x2 + W_{23} * x3 + b_2 \\
a3 &= W_{31} * x1 + W_{32} * x2 + W_{33} * x3 + b_3
\end{aligned}
$$

可以写成如下矩阵形式：

$$
\begin{bmatrix} a1 \\ a2 \\ a3 \end{bmatrix} = \begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \\ W_{31} & W_{32} & W_{33} \end{bmatrix} * \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} + \begin{bmatrix} b1 \\ b2 \\ b3 \end{bmatrix}
$$

https://www.jianshu.com/p/a4b8b0fc7d21

2.3. Tensor function

- 使用tensor函数来表示一个层（layer），从而可以与其它的层组合一起
- VecFun(x, v)表示一个函数，输入为tensor x，输出为tensor v

```
T.vec(N, M2, (i, j) =>
  T.sum(M1, k => x(i, k) * w(j, k)) + b(j)
)
```

Listing 2. Fully connected layer.

```
VecFun(x,
 T.vec(N, M2,
  (i, j) => T.sum(M1, k => x(i, k) * w(j, k)) + b(j)
 )
)
```

Listing 3. Tensor function for fully-connected layer.

tensor

tensor function

代码实现：deepdsl-java/src/main/scala/deepdsl/ast/Fun.scala

2.4. Fixed tensor

- 用来表示库（Cudnn）的实现
- FixVec(layer, param, dim)

这里的FixVec表示1个tensor，用于ReLU类型的层，该层的输入参数列表为List(x)，输出的tensor的维度为x.dim

```
1 def relu(n: Int) = {
2   val x = T._new(n)
3   VecFun(x, FixVec(ReLU(), List(x), x.dim))
4 }
```

Listing 5. Function for ReLU.

代码实现：deepdsl-java/src/main/scala/deepdsl/ast/Vec.scala

2.5. Function application and composition

- 函数调用在内部用VecApp来表示，例如f(x)表示为VecApp(f, x)，activate(f(x))表示为VecApp(activate, VecApp(f, x))

```
// x: N x M1    w: M2 x M1    b: M2
def full(w: VecDec, b: VecDec) = {
 val N = T.dim; val M2 = w.dim(0); val M1 = w.dim(1)
 val x = T._new(N, M1)

 VecFun(x,
  T.vec(N, M2,
   (i, j) => T.sum(M1, k => x(i, k) * w(j, k)) + b(j)
  )
 )
}
```

Listing 4. Method that returns a fully-connected layer.

```
val x = T._new(2)

val M1 = T.dim
val M2 = T.dim(10)  // dimension of size 10

// w is named "W" and initialized as Gaussian variable
val w = T._new(Param.gaussian, "W", M2, M1)
// b is named "B" and initialized as constant 0
val b = T._new(Param.const(0), "B", M2)

val f = full(w, b)
val activate = relu(2)

activate(f(x))
```

Listing 6. Function application.

等价表示形式

activate o f <=> activate.o(f)

```
activate o f
```

2.6. Network as function composition

- CudaLayer和Layer为Scala object，封装了一些用来创建层的帮助函数

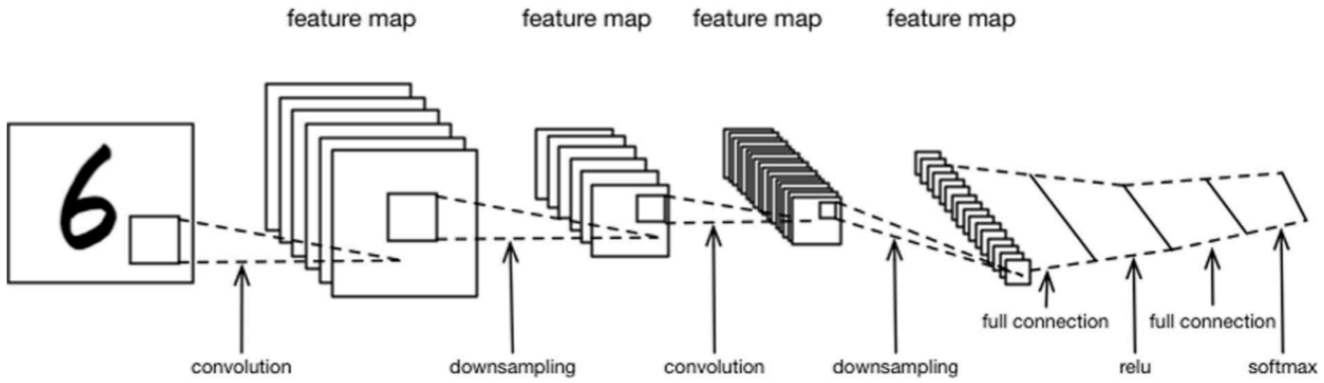


Fig. 2. A version of the LeNet-5 network structure [30].

```
1  val cv1 = CudaLayer.convolv("cv1", 5, 20)
2  val cv2 = CudaLayer.convolv("cv2", 5, 50)
3  val mp = CudaLayer.max_pool(2)
4  val relu = CudaLayer.relu(2)
5  val f = Layer.full("fc1", 500)
6  val f2 = Layer.full("fc2", 10)
7  val flat = Layer.flatten(4, 1)
8
9  val network = f2 o relu o f o flat o
10                mp o cv2 o mp o cv1
```

Listing 7. Network definition of Lenet.

代码实现：deepdsl-java/src/main/scala/deepdsl/layer/CudaLayer.scala
deepdsl-java/src/main/scala/deepdsl/layer/Layer.scala

2.6. Network as function composition

- CudaLayer和Layer为Scala object，封装了一些用来创建层的帮助函数

创建1个 5 x 5 的kernel，输出的channel大小为20

创建1个max pooling层，将输入缩放2倍

创建1个全连接层，输出大小为500

将4-D tensor展平为2-D tensor，4表示维度个数，1表示折叠的起始索引

```
1  val cv1 = CudaLayer.convolv("cv1", 5, 20)
2  val cv2 = CudaLayer.convolv("cv2", 5, 50)
3  val mp = CudaLayer.max_pool(2)
4  val relu = CudaLayer.relu(2)
5  val f = Layer.full("fc1", 500)
6  val f2 = Layer.full("fc2", 10)
7  val flat = Layer.flatten(4, 1)
8
9  val network = f2 o relu o f o flat o
10                mp o cv2 o mp o cv1
```

Listing 7. Network definition of Lenet.

2.7. Training

- LeNet

```
// batch size, channel, width, and height
val N = 500; val C = 1; val N1 = 28; val N2 = 28
val dim = List(N,C,N1,N2)

val y = T._new("Y", List(N)) // image class labels
val x = T._new("X", dim)     // training images

val y1 = y.asIndicator(10).asCuda
val x1 = x.asCuda            // load to GPU memory

val softmax = CudaLayer.log_softmax
val loss = Layer.loss(y1)

val p = network(x1)  // p is the prediction
                     // c is loss of training
val c = (loss o softmax o network) (x1)
```

Listing 8. Loss expression of Lenet.

```
val param = c.freeParam

// name, train/test iterations, learn rate, momentum
// weight decay, gradient clipping bound (0 means none)
val solver =
    Train("lenet", 100, 10, 0.01f, 0.9f, 0.0005f, 0)

val mnist = Mnist(dim)        // training dataSet
val loop = Loop(c, p, mnist, (x, y), param, solver)

// generate training and testing file
CudaCompile("path").print(loop)
```

Listing 9. Compilation of Lenet.

讨论

- DeepDSL与TensorFlow Python接口有什么区别？
- CudaLayer、Mnist这些出现在语言层面是否合适？
- Train、Loop、CudaCompile这些跟框架有什么区别？

- Java在AI领域是否应用广泛？