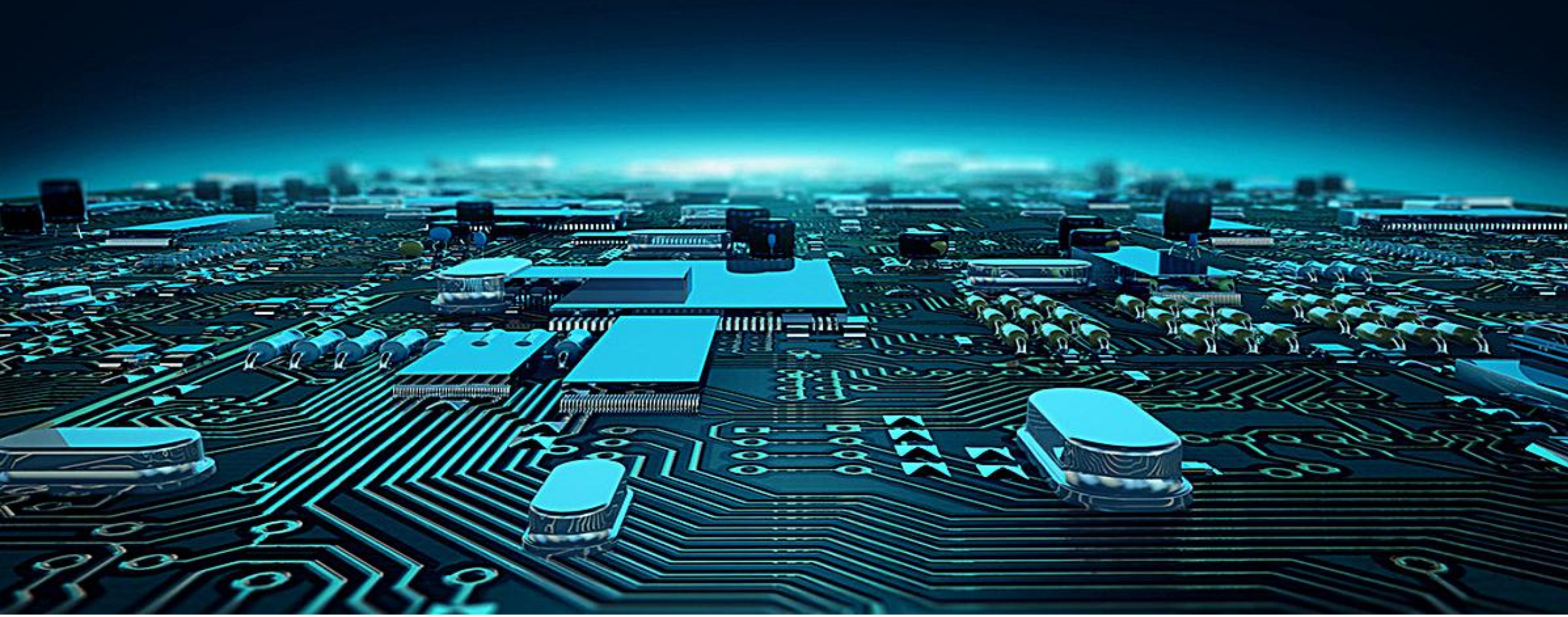
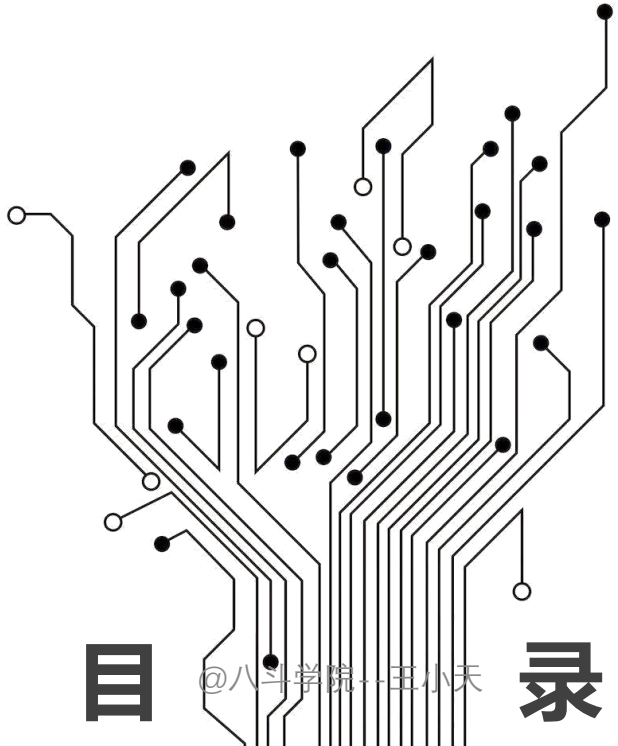


深度学习开源框架

@八斗学院--王小天(Michael)

2023/07/09

1. 深度学习框架
2. 主流深度学习框架
3. Tensorflow
4. Pytorch
5. 相关优化算法：BGD、SGD
6. 拓展：Adam、RMSprop



目　录

深度学习框架

深度学习框架，比如Caffe、tensorflow这些是深度学习的工具，简单来说就是库，编程时需要import caffe、import tensorflow。

应用优势：

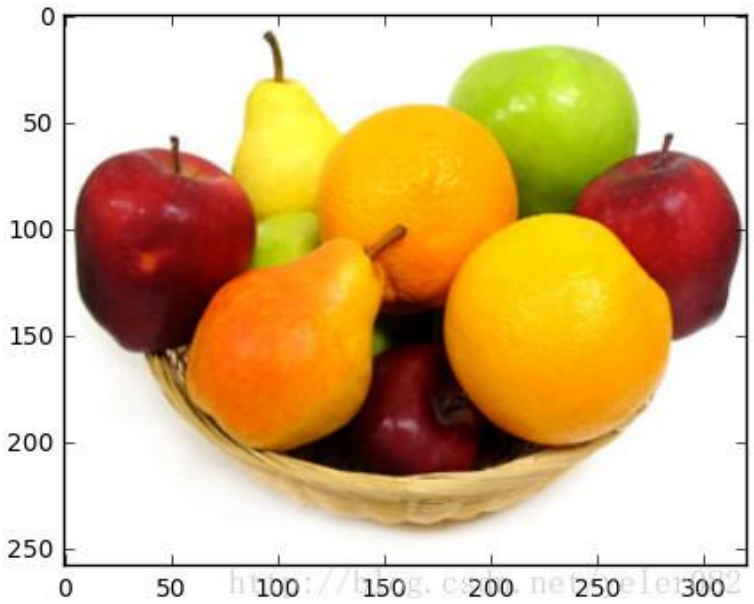
- 深度学习框架的出现降低了入门的门槛，你不需要从复杂的神经网络开始编代码，你可以依据需要，使用已有的模型，模型的参数你自己训练得到，你也可以在已有模型的基础上增加自己的layer，或者是在顶端选择自己需要的分类器和优化算法（比如常用的梯度下降法）。
- 当然也正因如此，没有什么框架是完美的，就像一套积木里可能没有你需要的那一种积木，所以不同的框架适用的领域不完全一致。
- 总的来说深度学习框架提供了一系列的深度学习的组件（对于通用的算法，里面会有实现），当需要使用新的算法的时候就需要用户自己去定义，然后调用深度学习框架的函数接口使用用户自定义的新算法。

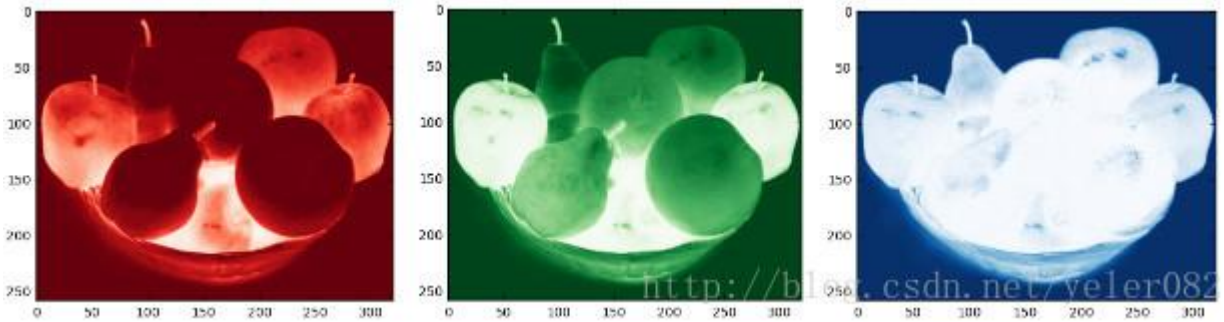
深度学习框架—关于组件

大部分深度学习框架都包含以下五个核心组件:
1. 张量（Tensor）
2. 基于张量的各种操作（Operation）
3. 计算图（Computation Graph）
4. 自动微分（Automatic Differentiation）工具
5. BLAS、cuBLAS、cuDNN等拓展包

深度学习框架—关于组件（张量）

张量是所有深度学习框架中最核心的组件，因为后续的所有运算和优化算法都是基于张量进行的。几何代数中定义的张量是基于向量和矩阵的推广，通俗一点理解的话，我们可以将标量视为零阶张量，矢量视为一阶张量，那么矩阵就是二阶张量。





深度学习框架—关于组件（张量）

前5行、320列的数据为例，每个方格代表一个像素点，其中的数据[1.0, 1.0, 1.0]即为颜色。
假设用[1.0, 0, 0]表示红色，[0, 1.0, 0]表示绿色，[0, 0, 1.0]表示蓝色。那么，前面5行的数据则全是白色。

	0	1	2	3	4	5	6	7	8	9	...	310	311	312	313	314	315	316	317	318	319
0	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	...	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]
1	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	...	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]
2	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	...	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]
3	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	...	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]
4	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	...	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]

深度学习框架—关于组件（张量）

将这一定义进行扩展，我们也可以用四阶张量表示一个包含多张图片的数据集，其中的四个维度分别是：图片在数据集中的编号，图片高度、宽度，以及色彩数据。N,H,W,C

优势：

将各种各样的数据抽象成张量表示，然后再输入神经网络模型进行后续处理是一种非常必要且高效的策略。因为如果没有这一步骤，我们就需要根据各种不同类型的数据组织形式定义各种不同类型的数据操作，这会浪费大量的开发者精力。

更关键的是，当数据处理完成后，我们还可以方便地将张量再转换回想要的格式。

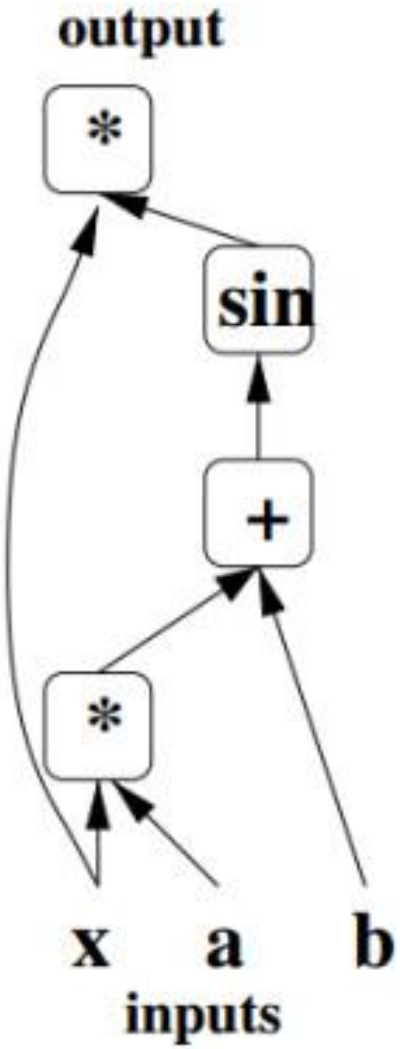
深度学习框架—关于组件（基于张量的各种操作）

- 有了张量对象之后，下面一步就是一系列针对这一对象的数学运算和处理过程。
- **其实，整个神经网络都可以简单视为为了达到某种目的，针对输入张量进行的一系列操作过程。而所谓的“学习”就是不断纠正神经网络的实际输出结果和预期结果之间误差的过程。**
- 这里的一系列操作包含的范围很宽，可以是简单的矩阵乘法，也可以是卷积、池化和LSTM等稍复杂的运算。而且各框架支持的张量操作通常也不尽相同。

深度学习框架—关于组件（计算图）

- 有了张量和基于张量的各种操作之后，下一步就是将各种操作整合起来，输出我们需要的结果。
- 但不幸的是，随着操作种类和数量的增多，管理起来就变得十分困难，各操作之间的关系变得比较难以理清，有可能引发各种意想不到的问题，包括多个操作之间应该并行还是顺次执行，如何协同各种不同的底层设备，以及如何避免各种类型的冗余操作等等。这些问题有可能拉低整个深度学习网络的运行效率或者引入不必要的Bug，而计算图正是为解决这一问题产生的。
- 将计算图作为前后端之间的中间表示（Intermediate Representations）可以带来良好的交互性，开发者可以将Tensor对象作为数据结构，函数/方法作为操作类型，将特定的操作类型应用于特定的数据结构，从而定义出类似MATLAB的强大建模语言。
- 因为计算图的引入，开发者得以从宏观上俯瞰整个神经网络的内部结构，就好像编译器可以从整个代码的角度决定如何分配寄存器那样，计算图也可以从宏观上决定代码运行时的GPU内存分配，以及分布式环境中不同底层设备间的相互协作方式。除此之外，现在也有许多深度学习框架将计算图应用于模型调试，可以实时输出当前某一操作类型的文本描述。

深度学习框架—关于组件（计算图）



深度学习框架—关于组件（自动微分工具）

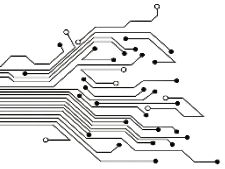
- 计算图带来的另一个好处是让模型训练阶段的梯度计算变得模块化且更为便捷，也就是自动微分法。

- 我们可以将神经网络视为由许多非线性过程组成的一个复杂的函数体，而计算图则以模块化的方式完整表征了这一函数体的内部逻辑关系，因此微分这一复杂函数体，即求取模型梯度的方法就变成了在计算图中简单地从输入到输出进行一次完整遍历的过程。

深度学习框架—关于组件（拓展包）

- 此前的大部分实现都是基于高级语言的（如Java、Python、Lua等），而即使是执行最简单的操作，高级语言也会比低级语言消耗更多的CPU周期，更何况是结构复杂的深度神经网络，因此运算缓慢就成了高级语言的一个天然的缺陷。

- 由于低级语言的最优化编程难度很高，而且大部分的基础操作其实也都有公开的最优解决方案，因此一个显著的加速手段就是利用现成的扩展包。



主流深度学习框架

1. TensorFlow
2. Keras
3. PyTorch
4. MXNet
5. Caffe
6. Caffe2
7. Theano
8. FastAI
9. CNTK
10. Gluon
11. Torch
12. Deeplearning4j
13. Chainer

主流深度学习框架

框　　架	机　　构	支持语言	Stars	Forks	Contributors
TensorFlow	Google	Python/C++/Go/...	41628	19339	568
Caffe	BVLC	C++/Python	14956	9282	221
Keras	fchollet	Python	10727	3575	322
CNTK	Microsoft	C++	9063	2144	100
MXNet	DMLC	Python/C++/R/...	7393	2745	241
Torch7	Facebook	Lua	6111	1784	113
Theano	U. Montreal	Python	5352	1868	271
Deeplearning4J	DeepLearning4J	Java/Scala	5053	1927	101
Leaf	AutumnAI	Rust	4562	216	14
Lasagne	Lasagne	Python	2749	761	55
Neon	NervanaSystems	Python	2633	573	52

主流深度学习框架

	模型设计	接　口	部　署	性　能	架构设计	总体评分
TensorFlow	80	80	90	90	100	88
Caffe	60	60	90	80	70	72

	模型设计	接　口	部　署	性　能	架构设计	总体评分
CNTK	50	50	70	100	60	66
Theano	80	70	40	50	50	58
Torch	90	70	60	70	90	76
MXNet	70	100	80	80	90	84
DeepLearning4J	60	70	80	80	70	72

主流深度学习框架

顶级深度学习框架四大阵营：

1. TensorFlow，前端框架Keras，背后巨头Google；
2. PyTorch，前端框架FastAI，背后巨头Facebook；
3. MXNet，前端框架Gluon，背后巨头Amazon；
4. Cognitive Toolkit (CNTK)，前端框架Keras或Gluon，背后巨头Microsoft。

本土深度学习框架

1. 华为MindSpore：支持端、边、云独立的和协同的统一训练和推理框架。2020年3月28日，正式开源

2. 百度PaddlePaddle：PaddlePaddle 100% 都在Github上公开，没有内部版本。PaddlePaddle能够应用于自然语言处理、图像识别、推荐引擎等多个领域，其优势在于开放的多个领先的预训练中文模型。

3. 阿里巴巴XDL (X-Deep Learning)：阿里妈妈将把其应用于自身广告业务的算法框架XDL (X-Deep Learning)进行开源。XDL主要是针对特定应用场景如广告的深度学习问题的解决方案，是上层高级API框架而不是底层框架。XDL需要采用桥接的方式配合使用 TensorFlow 和 MXNet 作为单节点的计算后端，XDL依赖于阿里提供特定的部署环境。

4. 小米MACE：它针对移动芯片特性进行了大量优化，目前在小米手机上已广泛应用，如人像模式、场景识别等。该框架采用与 Caffe2 类似的描述文件定义模型，因此它能非常便捷地部署移动端应用。目前该框架为 TensorFlow 和 Caffe 模型提供转换工具，并且其它框架定义的模型很快也能得到支持。

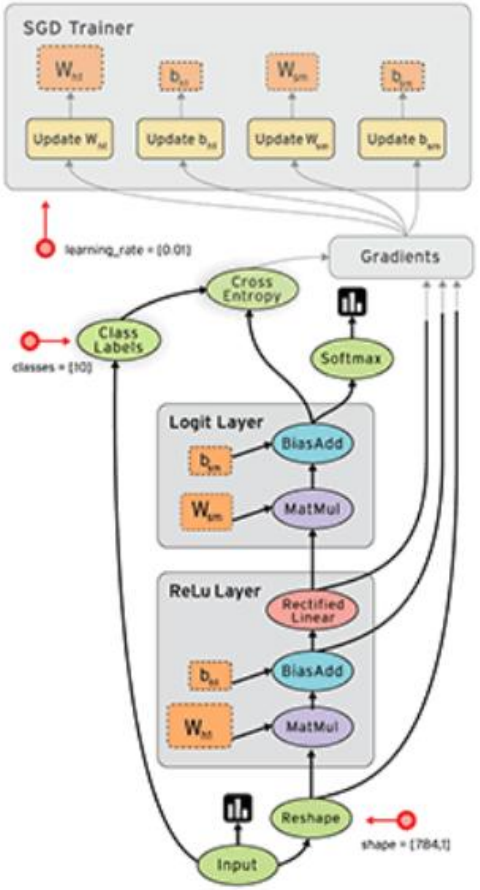
深度学习框架的标准化--ONNX

- 开放神经网络交换（ONNX，“Open Neural Network Exchange”）：
ONNX最初由微软和Facebook联合发布，后来亚马逊也加入进来，并发布了V1版本，宣布支持ONNX的公司还有AMD、ARM、华为、 IBM、英特尔、Qualcomm等。

- ONNX是一个表示深度学习模型的开放格式。它使用户可以更轻松地在不同框架之间转移模型。例如，它允许用户构建一个PyTorch模型，然后使用MXNet运行该模型来进行推理。

Tensorflow

TensorFlow 是一个采用数据流图（data flow graphs），用于数值计算的开源软件库。节点（Nodes）在图中表示数学操作，图中的线（edges）则表示在节点间相互联系的多维数据数组，即张量（tensor）。它灵活的架构可以在多种平台上展开计算，例如台式计算机中的一个或多个CPU（或GPU），服务器，移动设备等等。



Tensorflow—基本用法

1. 使用图 (graph) 来表示计算任务.
2. 在被称之为 会话 (Session) 的上下文 (context) 中执行图.
3. 使用 tensor 表示数据.
4. 通过 变量 (Variable) 维护状态.
5. 使用 feed 和 fetch 可以为任意的操作(arbitrary operation)赋值或者从其中获取数据。

- TensorFlow 是一个编程系统, 使用图来表示计算任务. 图中的节点被称之为 op (operation 的缩写)。一个 op 获得 0 个或多个 Tensor。执行计算, 产生 0 个或多个 Tensor。
- 每个 Tensor 是一个类型化的多维数组。例如,你可以将一小组图像集表示为一个四维浮点数数组, 这四个维度分别是 [batch, height, width, channels].
- 一个 TensorFlow 图描述了计算的过程. 为了进行计算, 图必须在 会话 里被启动.
- 会话 将图的 op 分发到诸如 CPU 或 GPU 之类的 设备 上, 同时提供执行 op 的方法.
- 这些方法执行后, 将产生的 tensor 返回. 在 Python 语言中, 返回的 tensor 是numpy ndarray 对象。

Tensorflow—构建图

1. 创建源op：源op不需要任何输入。例如常量（constant）。源op的输出被传递给其他op做运算。
2. 在会话（session）中启动图
3. 关闭session以释放资源

Tensorflow—Tensor（张量）

构建图的运算过程输出的结果是一个Tensor，主要由三个属性构成：Name、Shape和Type。

- Name代表的是张量的名字，也是张量的唯一标识符，我们可以在每个op上添加name属性来对节点进行命名，Name的值表示的是该张量来自于第几个输出结果（编号从0开始）。
- Shape代表的是张量的维度。
- Type表示的是张量的类型，每个张量都会有唯一的类型。**我们需要注意的是要保证参与运算的张量类型相一致，否则会出现类型不匹配的错误。**

Tensorflow—Tensor（张量）

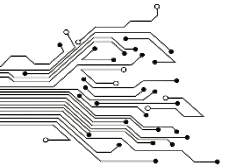
常见的张量类型

Data type	Python type	Description
DT_FLOAT	tf.float32	32 bits floating point.
DT_DOUBLE	tf.float64	64 bits floating point.
DT_INT8	tf.int8	8 bits signed integer.
DT_INT16	tf.int16	16 bits signed integer.
DT_INT32	tf.int32	32 bits signed integer.
DT_INT64	tf.int64	64 bits signed integer.
DT_UINT8	tf.uint8	8 bits unsigned integer.
DT_UINT16	tf.uint16	16 bits unsigned integer.
DT_STRING	tf.string	Variable length byte arrays. Each element of a Tensor is a byte array.
DT_BOOL	tf.bool	Boolean.
DT_COMPLEX64	tf.complex64	Complex number made of two 32 bits floating points: real and imaginary parts.
DT_COMPLEX128	tf.complex128	Complex number made of two 64 bits floating points: real and imaginary parts.
DT_QINT8	tf.qint8	8 bits signed integer used in quantized Ops.
DT_QINT32	tf.qint32	32 bits signed integer used in quantized Ops.
DT_QUINT8	tf.quint8	8 bits unsigned integer used in quantized Ops.

Tensorflow—变量（Variables）

变量Variables维护图执行过程中的状态信息.

通常会将一个统计模型中的参数表示为一组变量.
例如, 你可以将一个神经网络的权重作为某个变量存储在一个 tensor 中. 在训练过程中, 通过重复运行训练图, 更新这个 tensor.



Tensorflow—fetch/feed

Fetch：为了取回操作的输出内容，可以使用session对象的run()调用执行图时，传入一些tensor，这些tensor会帮助你取回结果。

Tensorflow—fetch/feed

Feed：使用一个tensor值临时替换一个操作的输出结果。可以提供feed数据作为run()调用的参数.

feed 只在调用它的方法内有效, 方法结束, feed 就会消失. 最常见的用例是将某些特殊的操作指定为 “feed” 操作, 标记的方法是使用 tf.placeholder() 为这些操作创建占位符。

placeholder是一个数据初始化的容器，它与变量最大的不同在于placeholder定义的是一个模板，这样我们就可以在session运行阶段，利用feed_dict的字典结构给placeholder填充具体的内容，而无需每次都提前定义好变量的值，大大提高了代码的利用率。

拓展-Tensorflow--可视化工具TensorBoard

对大部分人而言，深度神经网络就像一个黑盒子，其内部的组织、结构、以及其训练过程很难理清楚，这给深度神经网络原理的理解和工程化带来了很大的挑战。

为了解决这个问题，tensorboard应运而生。

Tensorboard是tensorflow内置的一个可视化工具，它通过将tensorflow程序输出的日志文件的信息可视化使得tensorflow程序的理解、调试和优化更加简单高效。

Tensorboard的可视化依赖于tensorflow程序运行输出的日志文件，因而tensorboard和tensorflow程序在不同的进程中运行。

拓展-Tensorflow--可视化工具TensorBoard

```
# 生成一个具有写权限的日志文件操作对象，将当前命名空间的计算图写进日志中
writer=tf.summary.FileWriter('logs', tf.get_default_graph())
writer.close()

#启动tensorboard服务（在命令行启动）
tensorboard --logdir logs

#启动tensorboard服务后，复制地址并在本地浏览器中打开，
```

Pytorch

- Pytorch是torch的python版本，是由Facebook开源的神经网络框架，专门针对 GPU 加速的深度神经网络（DNN）编程。Torch 是一个经典的对多维矩阵数据进行操作的张量（tensor ）库，在机器学习和其他数学密集型应用有广泛应用。
- **Pytorch的计算图是动态的，可以根据计算需要实时改变计算图。**
- 由于Torch语言采用 Lua，导致在国内一直很小众，并逐渐被支持 Python 的 Tensorflow 抢走用户。作为经典机器学习库 Torch 的端口，PyTorch 为 Python 语言使用者提供了舒适的写代码选择。

这是一个基于Python的科学计算包，其旨在服务两类场合：

- 替代numpy发挥GPU潜能
- 一个提供了高度灵活性和效率的深度学习实验性平台

Pytorch的优势

1.简洁:

PyTorch的设计追求最少的封装，尽量避免重复造轮子。不像 TensorFlow 中充斥着session、graph、operation、name_scope、variable、tensor、layer等全新的概念，PyTorch 的设计遵循tensor→variable(autograd)→nn.Module 三个由低到高的抽象层次，分别代表高维数组（张量）、自动求导（变量）和神经网络（层/模块），而且这三个抽象之间联系紧密，可以同时进行修改和操作。

2.速度:

PyTorch 的灵活性不以速度为代价，在许多评测中，PyTorch 的速度表现胜过 TensorFlow和Keras 等框架。

3.易用:

PyTorch 是所有的框架中面向对象设计的最优雅的一个。PyTorch的面向对象的接口设计来源于Torch，而Torch的接口设计以灵活易用而著称，Keras作者最初就是受Torch的启发才开发了Keras。

4.活跃的社区:

PyTorch 提供了完整的文档，循序渐进的指南，作者亲自维护的论坛，供用户交流和求教问题。Facebook 人工智能研究院对 PyTorch 提供了强力支持。

Pytorch-常用工具包

1. torch： 类似 NumPy 的张量库，支持GPU;
2. torch.autograd： 基于 type 的自动区别库，支持 torch 之中的所有可区分张量运行;
3. torch.nn： 为最大化灵活性而设计，与 autograd 深度整合的神经网络库;
4. torch.optim：与 torch.nn 一起使用的优化包，包含 SGD、RMSProp、LBFGS、Adam 等标准优化方式;
5. torch.multiprocessing： python 多进程并发，进程之间 torch Tensors 的内存共享;
6. torch.utils：数据载入器。具有训练器和其他便利功能;
7. torch.legacy(.nn/.optim)： 出于向后兼容性考虑，从 Torch 移植来的 legacy 代码;

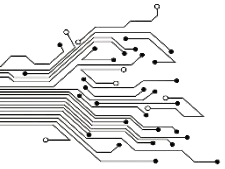
特别注意：

通道问题：不同的视觉库对于图像读取的方式不一样，图像的通道也不一样：

- **opencv的默认imread就是H*W*C，Pytorch的Tensor为C*H*W，TensorFlow两者都支持。**

理解pytorch的基础主要从以下三个方面

1. Numpy风格的Tensor操作。pytorch中tensor提供的API参考了Numpy的设计。
2. 变量自动求导。在一序列计算过程形成的计算图中，参与的变量可以方便的计算自己对目标函数的梯度。
3. 神经网络层与损失函数优化等高层封装。网络层的封装存在于torch.nn模块，损失函数由torch.nn.functional模块提供，优化函数由torch.optim模块提供。



Pytorch-Tensor

Torch 定义了七种 CPU tensor 类型和八种 GPU tensor 类型:

Data type	CPU tensor	GPU tensor
32-bit floating point	torch.FloatTensor	torch.cuda.FloatTensor
64-bit floating point	torch.DoubleTensor	torch.cuda.DoubleTensor
16-bit floating point		torch.cuda.HalfTensor
8-bit integer (unsigned)	torch.ByteTensor	torch.cuda.ByteTensor
8-bit integer (signed)	torch.CharTensor	torch.cuda.CharTensor
16-bit integer (signed)	torch.ShortTensor	torch.cuda.ShortTensor
32-bit integer (signed)	torch.IntTensor	torch.cuda.IntTensor
64-bit integer (signed)	torch.LongTensor	torch.cuda.LongTensor

创建Tensor的常见接口：

方法名	说明
Tensor()	直接从参数构造一个的张量，参数支持list,numpy数组
eye(row, column)	创建指定行数，列数的二维单位tensor
linspace(start,end,count)	在区间[s,e]上创建c个tensor
logspace(s,e,c)	在区间[10^s, 10^e]上创建c个tensor
ones(*size)	返回指定shape的张量，元素初始为1
zeros(*size)	返回指定shape的张量，元素初始为0
ones_like(t)	返回与t的shape相同的张量，且元素初始为1
zeros_like(t)	返回与t的shape相同的张量，且元素初始为0
arange(s,e,sep)	在区间[s,e)上以间隔sep生成一个序列张量

Tensor对象的方法：

方法名	作用
size()	返回张量的shape属性值
numel(input)	计算tensor的元素个数
view(*shape)	修改tensor的shape，与np.reshape类似，view返回的对象共享内存
resize	类似于view，但在size超出时会重新分配内存空间
item	若为单元素tensor，则返回pyton的scalar
from_numpy	从numpy数据填充
numpy	返回ndarray类型

Pytorch-自动求导

tensor对象通过一系列的运算可以组成动态图，对于每个tensor对象，有下面几个变量控制求导的属性。

变量	作用
requirs_grad	默认为False，表示变量是否需要计算导数
grad_fn	变量的梯度函数
grad	变量对应的梯度

Pytorch-神经网络

torch.nn模块提供了创建神经网络的基础构件，这些层都继承自Module类。
下面我们简单看下如何实现一个线性层（linear layer）。

Pytorch-神经网络

下面表格中列出了比较重要的神经网络层组件。
对应的在nn.functional模块中，提供这些层对应的函数实现。

通常对于可训练参数的层使用module，而对于不需要训练参数的层如softmax这些，可以使用functional中的函数。

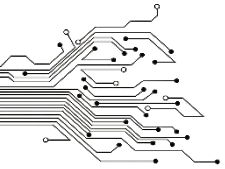
Layer对应的类	功能说明
Linear(in_dim, out_dim, bias=True)	提供了进行线性变换操作的功能
Dropout(p)	Dropout层，有2D,3D的类型
Conv2d(in_c, out_c, filter_size, stride, padding)	二维卷积层，类似的有Conv1d，Conv3d
ConvTranspose2d()	
MaxPool2d(filter_size, stride, padding)	二维最大池化层
MaxUnpool2d(filter, stride, padding)	逆过程
AvgPool2d(filter_size, stride, padding)	二维平均池化层
FractionalMaxPool2d	分数最大池化
AdaptiveMaxPool2d([h,w])	自适应最大池化
AdaptiveAvgPool2d([h,w])	自自应平均池化
ZeroPad2d(padding_size)	零填充边界
ConstantPad2d(padding_size,const)	常量填充边界
ReplicationPad2d(ps)	复制填充边界
BatchNorm1d()	对2维或3维小批量数据进行标准化操作
RNN(in_dim, hidden_dim, num_layers, activation, dropout, bidi, bias)	构建RNN层
RNNCell(in_dim, hidden_dim, bias, activation)	RNN单元
LSTM(in_dim, hidden_dim, num_layers, activation, dropout, bidi, bias)	构建LSTM层
LSTMCell(in_dim, hidden_dim, bias, activation)	LSTM单元
GRU(in_dim, hidden_dim, num_layers, activation, dropout, bidi, bias)	构建GRU层
GRUCell(in_dim, hidden_dim, bias, activation)	GRU单元

容器类型	功能
Module	神经网络模块的基类
Sequential	序列模型，类似于keras，用于构建序列型神经网络
ModuleList	用于存储层，不接受输入
Parameters(t)	模块的属性，用于保存其训练参数
ParameterList	参数列表

Pytorch-神经网络

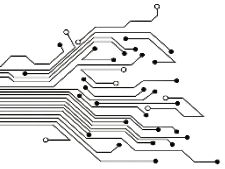
- torch.nn.Module提供了神经网络的基类，当实现神经网络时需要继承自此模块，并在初始化函数中创建网络需要包含的层，并实现forward函数完成前向计算，网络的反向计算会由自动求导机制处理。

- 通常将需要训练的层写在init函数中，将参数不需要训练的层在forward方法里调用对应的函数来实现相应的层。



在pytorch中就只需要分三步：

1. 写好网络；
2. 编写数据的标签和路径索引；
3. 把数据送到网络。



Pytorch实现MNIST分类

优化算法

深度学习的优化算法主要有GD，SGD，Momentum，RMSProp和Adam算法，还有诸如Adagrad算法，不过大同小异，理解了前面几个，后面的也就引刃而解了。

名词解释

名词	定义
original-loss	整个训练集上的loss
minibatch-loss	在一个mini batch上的loss
BGD	最原始的梯度下降算法，为了计算original-loss上的梯度，需要使用训练集全部数据
SGD	（近似）计算original-loss梯度时，只使用一个mini batch，相当于用minibatch-loss上的梯度去近似original-loss梯度

SGD（随机梯度下降）

SGD 又称 online 的梯度下降， 每次估计梯度的时候， 只选用一个或几个batch训练样本。

当训练数据过大时，用BGD可能造成内存不够用，那么就可以用SGD了。深度学习使用的训练集一般都比较大（几十万～几十亿）。而BGD算法，每走一步（更新模型参数），为了计算original-loss上的梯度，就需要遍历整个数据集，这显然效率是很低的。而SGD算法，每次随机选择一个mini-batch去计算梯度，每走一步只需要遍历一个minibatch（一～几百）的数据。

Momentum

通常情况我们在训练深度神经网络的时候把数据拆解成一小批一小批地进行训练，这就是我们常用的mini-batch SGD训练算法，然而虽然这种算法能够带来很好的训练速度，但是在到达最优点的时候并不能够总是真正到达最优点，而是在最优点附近徘徊。

另一个缺点就是这种算法需要我们挑选一个合适的学习率，当我们采用小的学习率的时候，会导致网络在训练的时候收敛太慢；当我们采用大的学习率的时候，会导致在训练过程中优化的幅度跳过函数的范围，也就是可能跳过最优点。

我们所希望的仅仅是网络在优化的时候网络的损失函数有一个很好的收敛速度，同时又不至于摆动幅度太大。

Momentum（拓展）

$$v_{dw} = \beta v_{dw} + (1-\beta)dW \ (1)$$
$$v_{db} = \beta v_{db} + (1-\beta)db \ (2)$$
$$W = W - \alpha v_{dw} \ (3)$$
$$b = b - \alpha v_{db} \ (4)$$

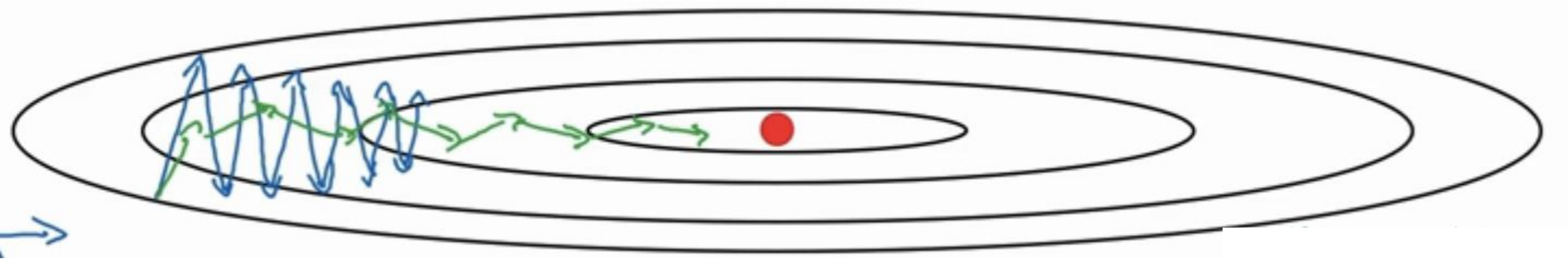
其中，在上面的公式中v_{dw}和v_{db}分别是损失函数在前$t-1$轮迭代过程中累积的梯度梯度动量，β是梯度累积的一个指数，这里我们一般设置值为0.9。所以Momentum优化器的主要思想就是利用了类似与移动指数加权平均的方法来对网络的参数进行平滑处理的，让梯度的摆动幅度变得更小。

dW和db分别是损失函数反向传播时候所求得的梯度，下面两个公式是网络权重向量和偏置向量的更新公式，α是网络的学习率。当我们使用Momentum优化算法的时候，可以解决mini-batch SGD优化算法更新幅度摆动大的问题，同时可以使得网络的收敛速度更快。

拓展--RMSprop（Root Mean Square Prop）

在Momentum优化算法中，虽然初步解决了优化中摆动幅度大的问题。所谓的摆动幅度就是在优化中经过更新之后参数的变化范围。

如下图所示，蓝色的为Momentum优化算法所走的路线，绿色的为RMSProp优化算法所走的路线。



拓展--RMSprop（Root Mean Square Prop）

为了进一步优化损失函数在更新中存在摆动幅度过大的问题，并且进一步加快函数的收敛速度，RMSProp算法对权重 W 和偏置 b 的梯度使用了微分平方加权平均数。

其中，假设在第 t 轮迭代过程中，各个公式如下所示：

$$
\begin{aligned}
s_{dw} &= \beta s_{dw} + (1-\beta) dW^2 \\
s_{db} &= \beta s_{db} + (1-\beta) db^2 \\
W &= W - \alpha \frac{dW}{\sqrt{s_{dw}} + \varepsilon} \\
b &= b - \alpha \frac{db}{\sqrt{s_{db}} + \varepsilon}
\end{aligned}
$$

算法的主要思想就用上面的公式表达完毕了。在上面的公式中 s_{dw}和 s_{db}分别是损失函数在前 $t-1$轮迭代过程中累积的梯度梯度动量，β 是梯度累积的一个指数。所不同的是，RMSProp算法对梯度计算了 **微分平方加权平均数**。这种做法有利于消除了摆动幅度大的方向，用来修正摆动幅度，使得各个维度的摆动幅度都较小。另一方面也使得网络函数收敛更快。（比如当 dW 或者 db 中有一个值比较大的时候，那么我们在更新权重或者偏置的时候除以它之前累积的梯度的平方根，这样就可以使得更新幅度变小）。为了防止分母为零，使用了一个很小的数值 ϵ 来进行平滑，一般取值为 10^{-8}。

拓展--Adam

有了上面两种优化算法，一种可以使用类似于物理中的动量来累积梯度，另一种可以使得收敛速度更快同时使得波动的幅度更小。

那么将两种算法结合起来所取得的表现一定会更好。Adam（Adaptive Moment Estimation）算法是将Momentum算法和RMSProp算法结合起来使用的一种算法，我们所使用的参数基本和上面的一致，在训练的最开始我们需要初始化梯度的累积量和平方累积量。

拓展--Adam

$$v_{dw}=0, v_{db}=0; s_{dw}=0, s_{db}=0$$

假设在训练的第 t 轮训练中，我们首先可以计算得到Momentum和RMSProp的参数更新：

$$\begin{aligned} v_{dw} &= \beta_1 v_{dw} + (1-\beta_1) dW \\ v_{db} &= \beta_1 v_{db} + (1-\beta_1) db \\ s_{dw} &= \beta_2 s_{dw} + (1-\beta_2) dW^2 \\ s_{db} &= \beta_2 s_{db} + (1-\beta_2) db^2 \end{aligned}$$

由于移动指数平均在迭代开始的初期会导致和开始的值有较大的差异，所以我们需要对上面求得的几个值做偏差修正。

$$\begin{aligned} v_{dw}^c &= \frac{v_{dw}}{1-\beta_1^t} \\ v_{db}^c &= \frac{v_{db}}{1-\beta_1^t} \\ s_{dw}^c &= \frac{s_{dw}}{1-\beta_2^t} \\ s_{db}^c &= \frac{s_{db}}{1-\beta_2^t} \end{aligned}$$

通过上面的公式，我们就可以求得在第 t 轮迭代过程中，参数梯度累积量的修正值，从而接下来就可以根据Momentum和RMSProp算法的结合来对权重和偏置进行更新。

$$\begin{aligned} W &= W - \alpha \frac{v_{dw}^c}{\sqrt{s_{dw}^c}+\varepsilon} \\ b &= b - \alpha \frac{v_{db}^c}{\sqrt{s_{db}^c}+\varepsilon} \end{aligned}$$

拓展--Adam

上面的所有步骤就是Momentum算法和RMSProp算法结合起来从而形成Adam算法。在Adam算法中，参数 β_1 所对应的就是Momentum算法中的 β 值，一般取0.9，参数 β_2 所对应的就是RMSProp算法中的 β 值，一般我们取0.999，而 ϵ 是一个平滑项，我们一般取值为 10^{-8}，而学习率 α 则需要我们在训练的时候进行微调。