第04讲 初识神经网络

欧 新 宇

深度学习概述

什么是深度学习?

深度学习概述

什么是人工智能、机器学习和深度学习?

认知心理学
Psychology

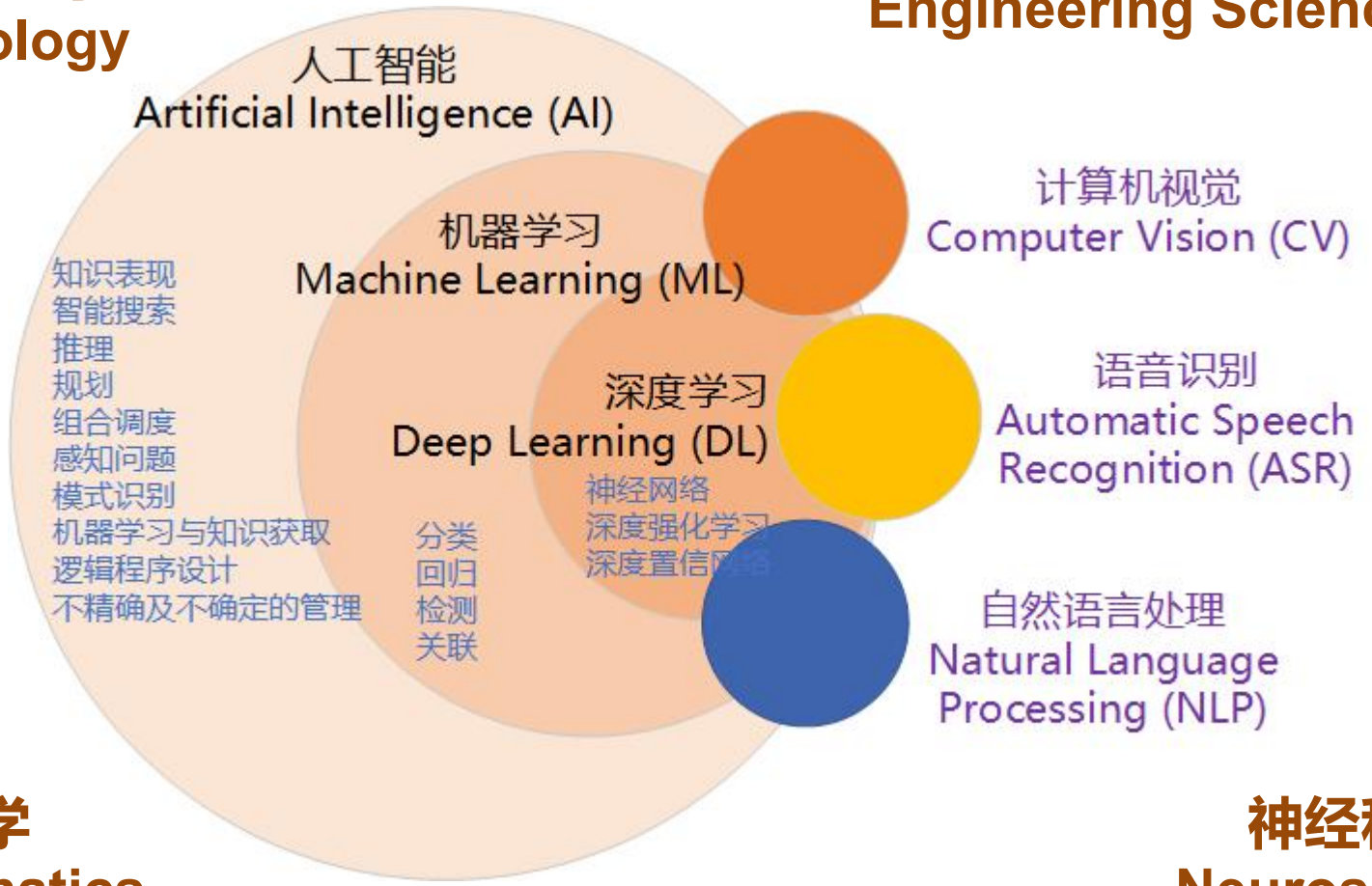
工程学
Engineering Science

计算机科学
Computer Science

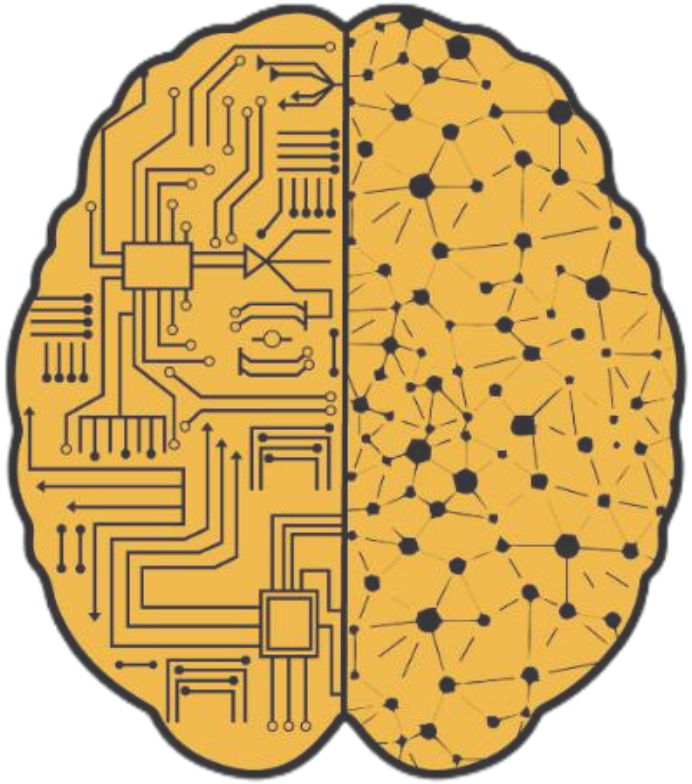
物理学
Physics

数学
Mathematics

神经科学
Neuroscience



深度学习概述



深度学习的深度

深度学习（Deep Learning）是机器学习的一个分支领域，它是从数据中学习表示的一种新方法，强调从连续的**层（layer）**中进行学习，这些层对应于越来越有意义的表示。

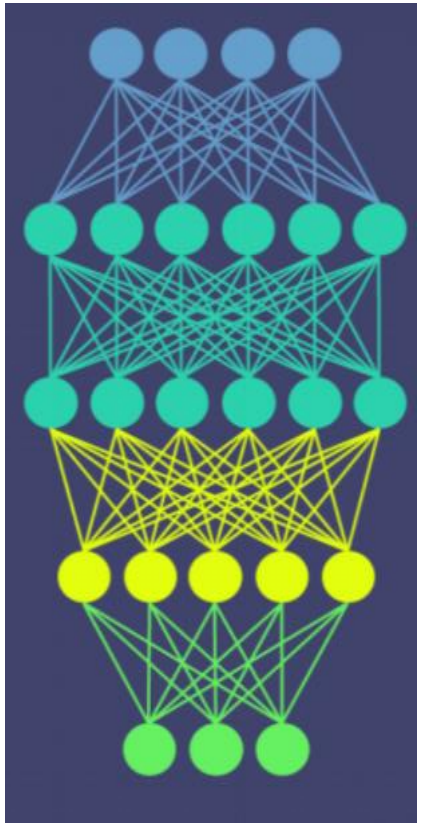
“深度学习”的**“深度”**指的并不是利用这种方法所获取的更深层次的理解，而是指一系列连续的表示层。由于每个层都对应于一种表示，因此模型中包含多少层，其**深度（depth）**就是多少。

相对而言，传统机器学习方法的重点往往是仅学习一两层的数据表示，因此也常被称为**浅层学习（Shallow Learning）**。

深度学习的深度

神经网络和神经生物学

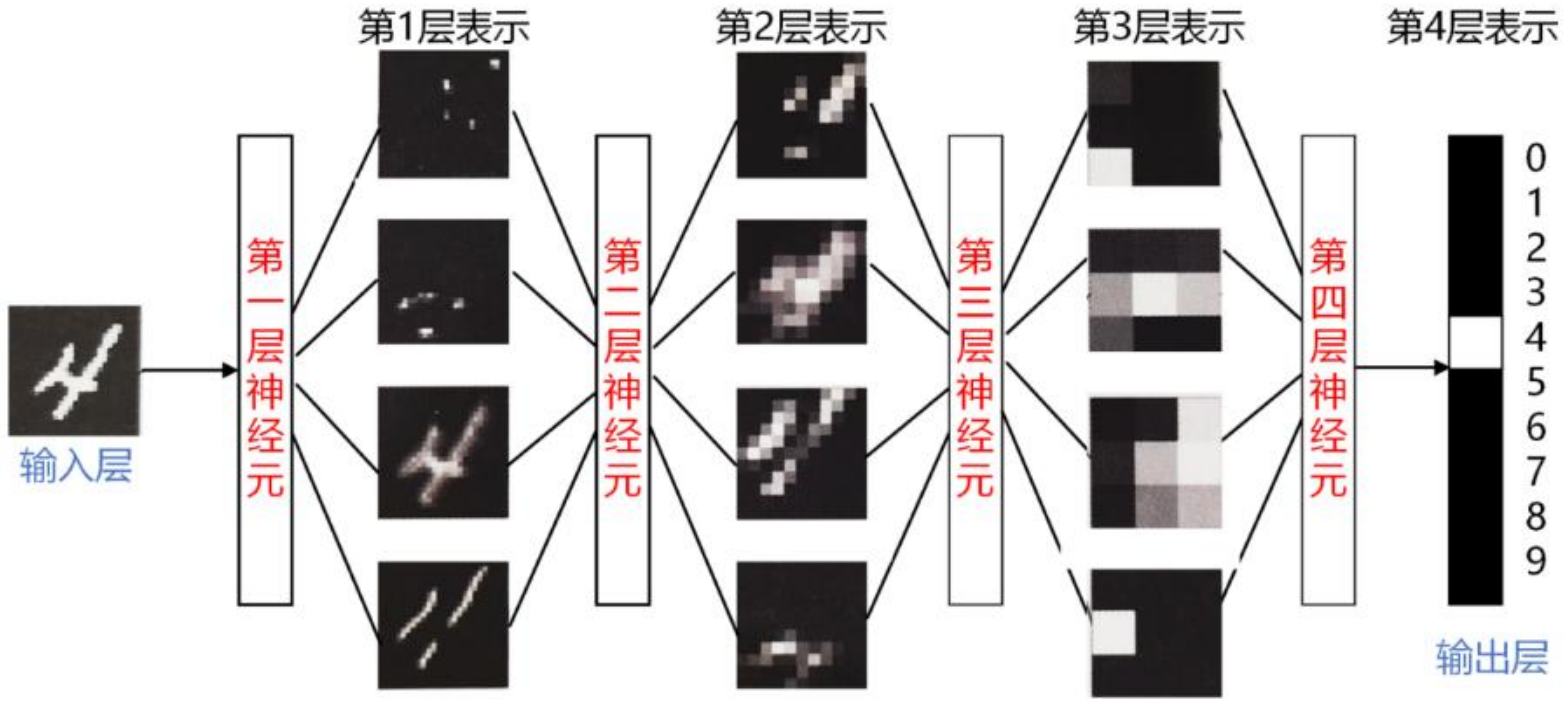
在深度学习中，分层表示学习总是通过**神经网络（Neural Network）**模型来学习，它的结构是逐层堆叠的。



神经网络这一术语来自于神经生物学，因此深度学习的一些概念也来源于人类对大脑理解中汲取的部分灵感，特别是**视觉神经系统**，但深度学习模型并不是大脑模型。至今仍没有证据表明大脑的学习机制与现代深度学习模型是**相同的**。很多流行的科学文章宣称深度学习的工作原理与大脑类似或者是根据大脑的工作原理进行建模，但**事实并非如此**。对于这一领域的新人来说，我们无须那种“就像我们的大脑一样”的神秘包装，也最好忘记读过的深度学习与生物学之间的假想联系。就我们的目的而言，**深度学习是从数据中学习表示的一种数学框架。**

深度学习的深度

从数字图像分类模型学到的深度表示



上述网络将数字图像转换成与原始图像差别越来越大的**表示**，而其中关于最终结果的信息却越来越丰富，即语义信息越来越丰富。深度网络的工作过程可以看作是多级信息蒸馏的操作：**信息穿过连续的过滤器，其纯度越来越高**（对任务的帮助越来越大）。

什么是深度学习?

深度学习的技术定义

学习数据表示的多级方法

非常简单的机制，如果具有**足够大的规模**，将会产生魔法般的效果。

深度学习有何不同?

深度学习有何不同?

特征工程

特征工程是指利用数据所在领域的相关知识来构建特征，使得机器学习算法发挥其最佳的过程。在传统机器学习任务中，特征工程是数据准备的一项核心任务，然而要获取理想的特征是非常困难的。正如Andrew Ng所说“挖掘特征是困难、费时且需要专业知识的事，应用机器学习其实基本上是在做特征工程。”

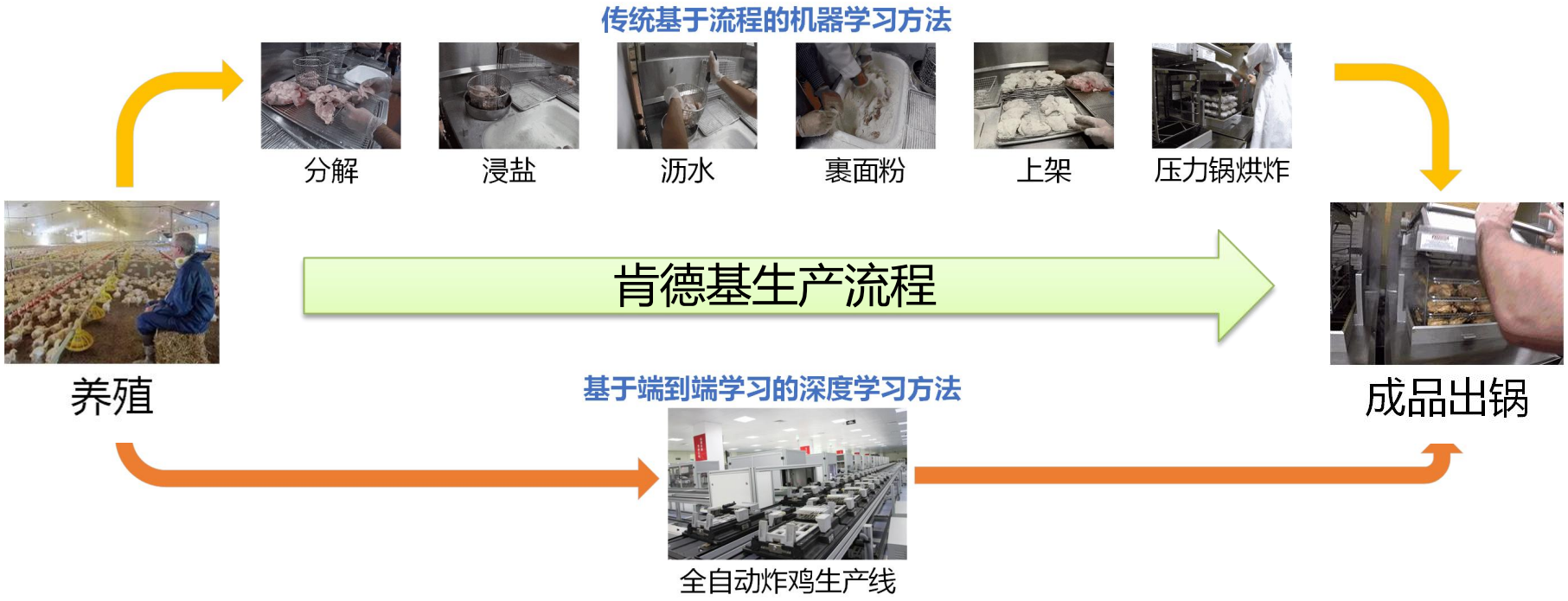
- **抽取特征**：特征工程是关键
- **特征描述子**：SIFT、SURF
- **视觉词袋（聚类）**：Bag of Feature（BoW）
- **分类模型**：支持向量机（SVM）

Picture: David Lowe. Object recognition from local scale-invariant features. *ICCV1999*.

深度学习有何不同?

更好的性能、简单的端到端的学习方法

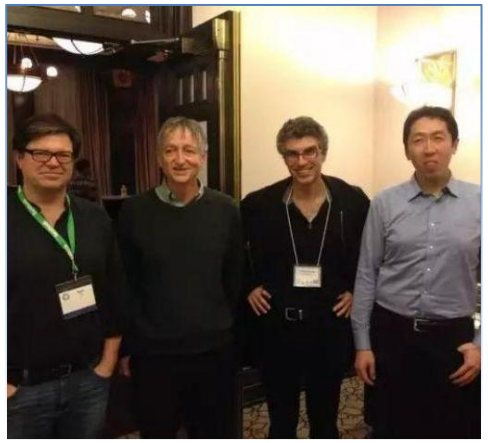
深度学习发展如此迅速，主要原因在于它在很多问题上都表现出更好的性能。但并非唯一原因，深度学习让解决问题变得更简单，因为特征工程完全自动化，不需要手动提取特征，而这一步骤是机器学习是否有效的**关键**。**这种学习方法被称为端到端（End-to-End）学习法。**



深度学习日益流行的关键因素

深度学习日益流行的关键因素及其未来潜力

深度学习用于计算机视觉的两个关键思想，即**卷积神经网络**和**反向传播**，早在1989年就为人所知（LeNet）。长短期记忆（LSTM）算法是深度学习处理时间序列的基础，也于1997年被开发出来，至今仍是自然语言处理核心算法之一。



那为什么深度学习在2012年之后才开始取得成功？这二十年间发生了什么变化？

三种技术推动了机器学习的进步：

- 数据集（互联网高速发展 ---> 众包 ---> 大数据）
- 硬件算力（针对游戏市场的需求开发出了高性能的图形芯片 ---> 大规模并行运算）
- 算法上的改进

硬件算力

从CPU到GPU再到TPU

	1970	1980	1990	2000	2010	2020
Data	10^2 Iris	10x 10^3	10x 10^4 32×32Gray	1000x 10^5 100×100RGB 10^7 web	100x 10^6 500×500RGB 10^9 32×32RGB	1000x 10^{8-9} 1000×1000RGB 10^{12} social nets
RAM	1KB	100KB	10MB	100MB	1GB	100GB
CPU GPU TPU	10KF (8080)	100x 1MF (80186)	10x 10MF (80486)	100x 1GF (Intel Core)	100x 100GF (nVidia GPU) CUDA	10000x >1PF (8xP3 Volta) TPU

神经网络

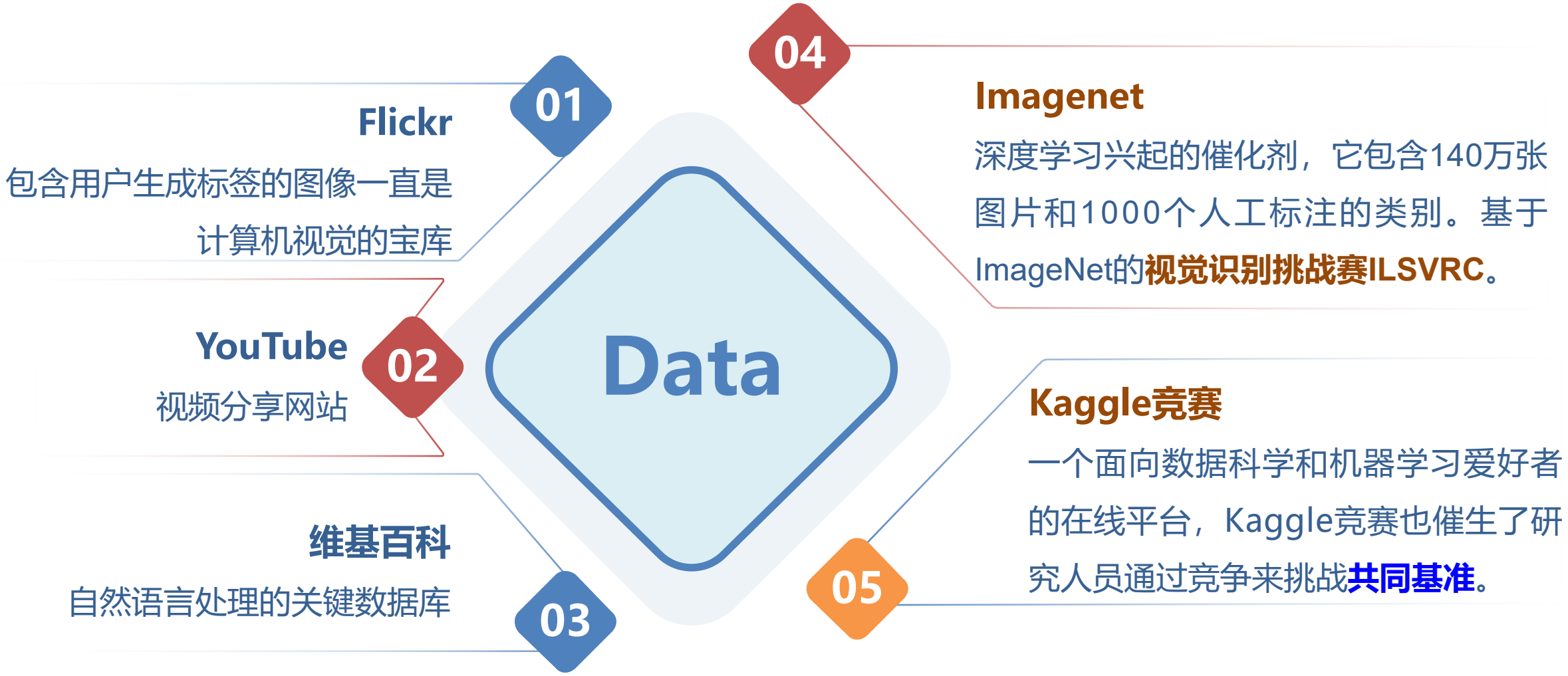
核方法

神经网络

Designed by LiMu-CS329 and edit by ouxinyu.

大数据

互联网 和 Imagenet



算法

针对梯度传播的改进

除了硬件和数据之外，直到21世纪前10年的末期，仍然没有可靠的方法训练非常深的神经网络。因此，神经网络依然很浅，性能无法超越更精确的浅层方法。

这一情况在2009-2010年左右发生了变化：

1. 更好的神经层**激活函数**，Sigmoid，ReLU等
2. 更好的**权重初始化方案**，一开始使用逐层预训练，但很快被放弃。从随机初始化到**Xavier(2010)**。
3. 更好的**优化方案**，例如mini-batch SGD，Adam，RMSProp等

依托以上方法，神经网络可以训练到10层以上，深度学习开始大方异彩；随后的2014-2016年，BatchNorm，残差网络ResNet，深度可分离卷积，使深度增加到1000层以上；而2017年之后，生成对抗网络、Transformer、扩散模型更是让深度学习产生了质的变化。

投资热潮

计算机视觉领域的成功引动了投资

- 2011年前，人工智能风投总额1900万美元，几乎都投给了浅层学习方法的**实际应用**。
- 2014年，针对深度学习的风投涨到惊人的3.94亿美元
- 2011-2014年，成立了数十家创业公司。同时，Google、Facebook（Meta）、百度、微软、亚马逊等大型科技公司成立人工智能研究部门，投资额超过风投现金流
- 2013年，Google收购DeepMind，收购价5亿美元
- 2014年，百度在硅谷启动深度学习研究中心（IDL），投资3亿美元
- 2016年，深度学习硬件创业公司Nervana Systems被Intel收购，收购价4亿美元。

机器学习，特别是深度学习成为科技巨头产品战略的核心。由于这波投资热潮，从业人员暴涨，研究进展也飞速。目前，没有迹象表明这种趋势会在短期内放缓。

深度学习的进展和未来

深度学习的大众化

从C++到Python，从数学原理到框架

在早期，从事深度学习需要精通C++和CUDA，而它们只有少数人能掌握；同时，还需要系统和高深的数学理论支持；此外，还需要一些英文阅读和写作的基本能力。

今天，具有基本的Python脚本技能，就可以从事高级的深度学习研究，得益于Caffe，Theano，MXNet及随后的TensorFlow，Pytorch、Keras、PaddlePaddle等用户友好型框架的兴起，深度学习应用的开发就像操作乐高积木一样简单。

这种趋势的未来?

昙花一现? Or 长盛不衰?

深度学习是否只是难以持续的昙花一现？20年后我们是否仍在使用深度神经网络？AIGC到底能发展到什么程度，是否能**达到甚至全面超过人类水平?**

来自现代深度学习的核心概念和重要性质：

- **简单。**深度学习不需要特征工程，它将复杂、不稳定、工程量很大的流程替换为简单的、端到端的可训练的模型，这些模型通常只用到少数几种不同的张量运算。
- **可扩展性。**深度学习非常适合在GPU和TPU上并行运算。此外，深度学习模型可以通过对小批量数据迭代进行训练，因此可以在任意大小的数据集上进行训练。唯一的瓶颈在摩尔定律下，限制也会越来越小。
- **多功能与可复用。**与大多数机器学习方法不同，深度学习无须从头开始就可以在附加数据上进行训练并适配附加数据。此外训练好的模型还可以应用于其他用途。泛化能力远优于传统机器学习。

深度学习已经取得的进展

面向视觉和听觉等感知上的重大突破

高度智能化的数字助理

自然流畅的语音识别

全面成熟的
自动驾驶技术

精准高效的广告投放

智能交互的网络搜索

超越人类水准的
图像分类

实时、高精度的
手写文字转录

无缝衔接的机器翻译

高保真度的文本到语音转换

全面智能的QA系统

人工智能的未来

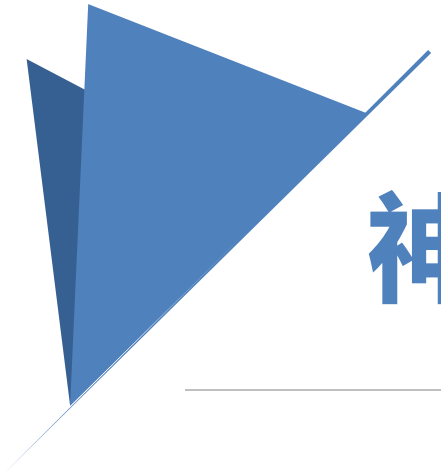
小　结

人工智能的

短期期望不切实际

但长远看前景光明

初识神经网络



神经网络的工作原理

三张图理解深度学习的工作原理

现在我们已经知道：

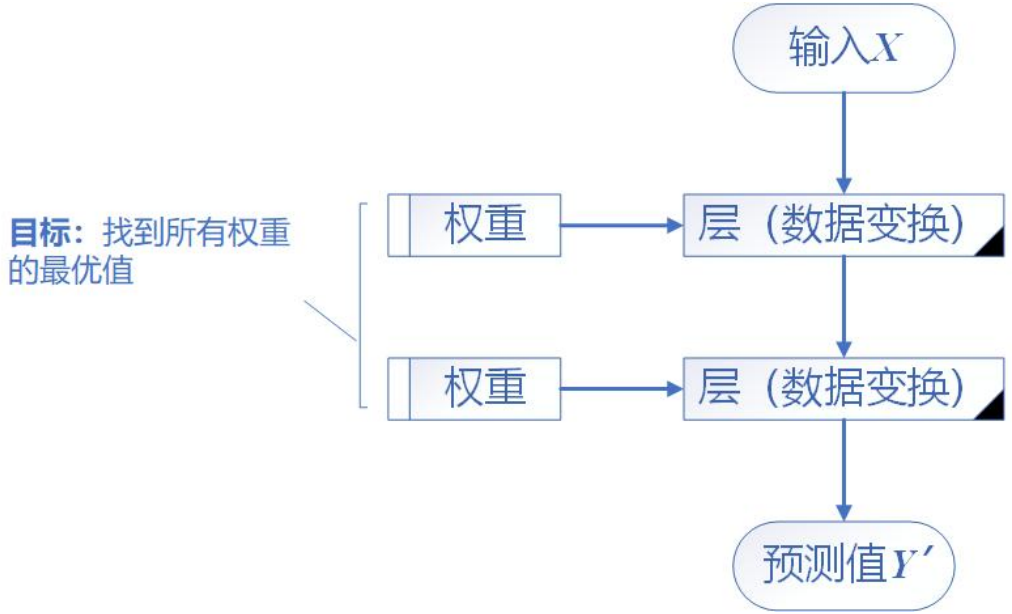
1. **机器学习**是将输入（如图像）映射到目标（比如标签“猫”），这一过程是通过观察许多输入和目标的示例来完成；
2. **深度神经网络**通过一系列简单的数据变换（层）来实现输入到目标的映射，而这些数据变换都是通过观察示例学习到的。

下面来具体看一下这种学习过程是如何发生的。

三张图理解深度学习的工作原理

一、神经网络由其权重来参数化

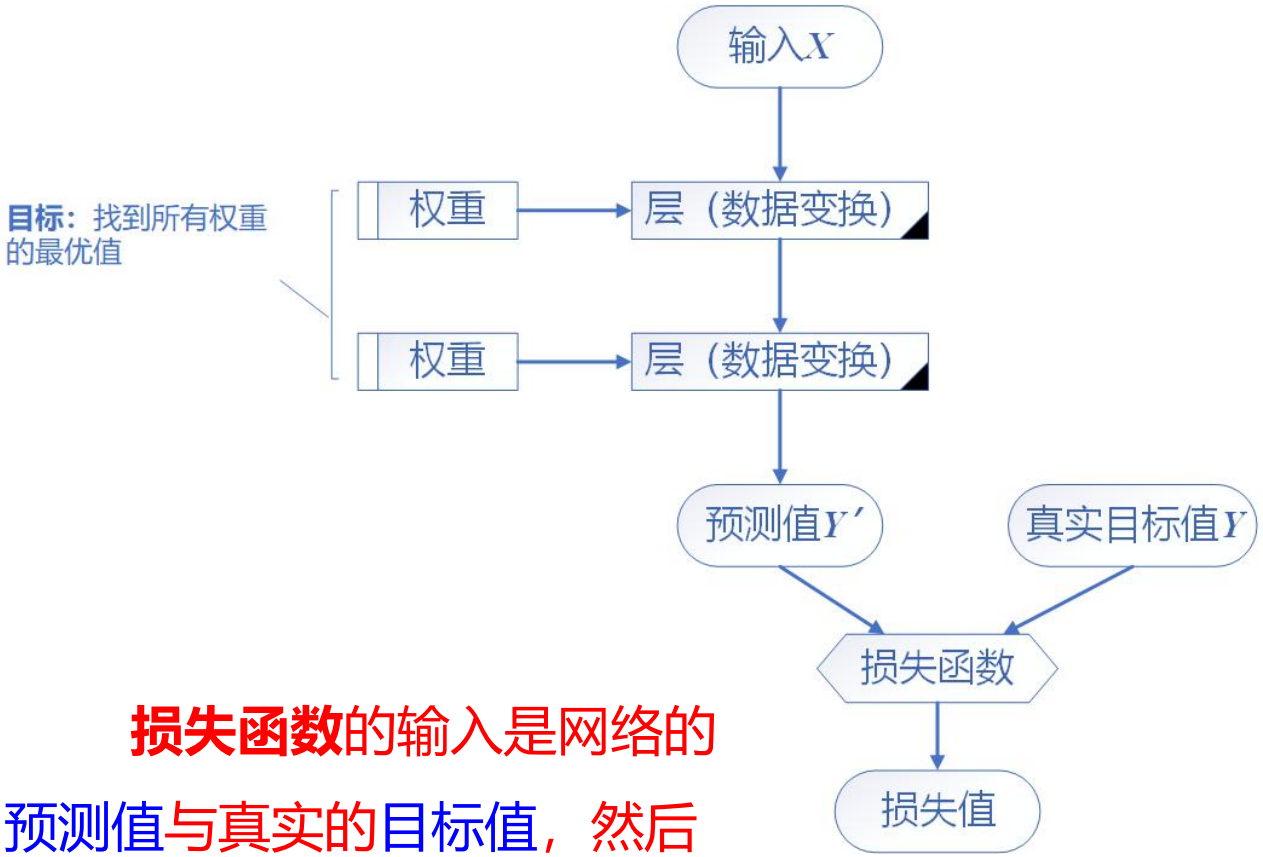
- 神经网络的每一层对输入数据所做的具体操作都保存在该层的**权重(weight)**中，其本质是一串浮点数。权重也被称为该层的**参数**。
- **学习**的本质是为神经网络的所有层找到一组权重值，使得网络能够将输入与目标正确对应。
- 预测值由**输入矩阵X**和**权重矩阵W**进行**逐层矩阵乘法**获得。



一个深度神经网络可能包含数千万个参数。找到所有参数的正确取值是一项非常艰巨的任务，特别是考虑到参数间可能存在**相互制约**的问题。

三张图理解深度学习的工作原理

二、损失函数用来衡量网络输出结果的质量

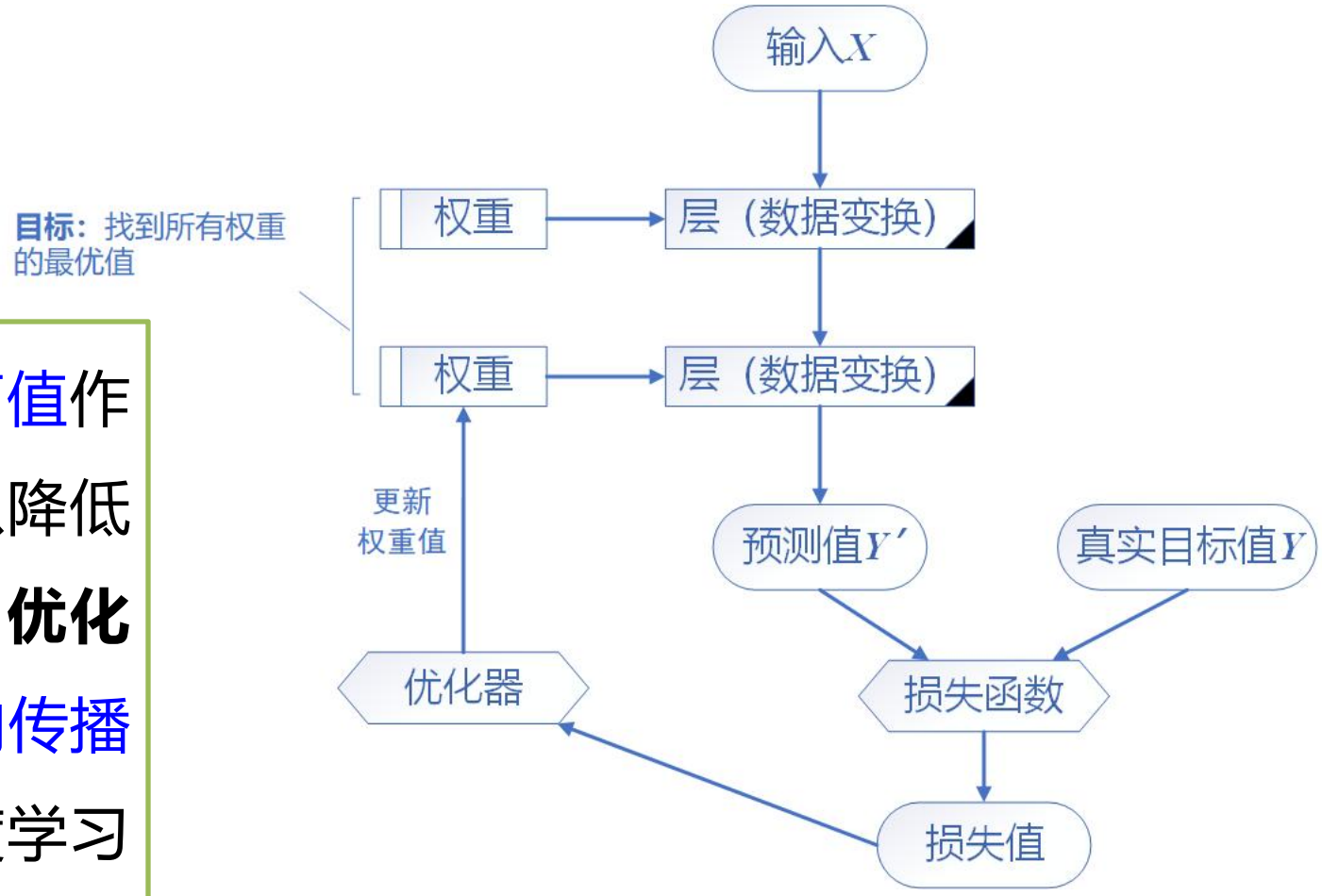


损失函数的输入是网络的预测值与真实的目标值，然后计算一个距离，用于衡量该网络在这个示例上的效果好坏。

想要能控制神经网络的输出（预测值），就需要能够衡量预测值与期望值之间的距离。这就是**损失函数(loss function)**的任务，该函数也称为**目标函数(objective function)**。

三张图理解深度学习的工作原理

三、将损失值作为反馈信号来调节权重



深度学习的基本技巧是利用距离值作为**反馈信号**来对权重值进行**微调**，以降低当前示例对应的损失值。这种调节由**优化器**(optimizer)来完成，它实现了反向传播算法(backpropagation)，这是深度学习的**核心算法**。

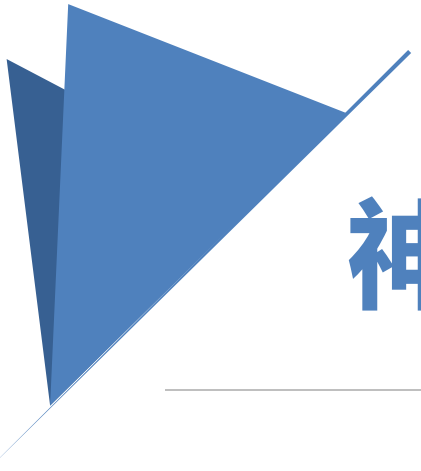
三张图理解深度学习的工作原理

工作原理小结

1. 训练开始时，神经网络中对权重进行随机初始化，因此网络刚开始的时候实现的是一系列的随机变换，其输出结果和预期值相去甚远，相应地，损失值也越高。
2. 随着网络对样本处理的数量越来越多，权重值也向正确方向进行逐步微调，损失值逐步降低。

步骤2是一个反复迭代的过程，这就是**训练循环**，这种循环重复足够多的次数，得到的权重值可以使损失函数达到最小（最优）。具有最小损失的网络，其输出值与目标值会尽可能的接近，这就是**训练好的网络**。

这是一个简单的机制，但对于足够大规模的网络和数据，将会产生魔法般的效果。

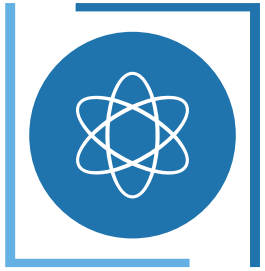


神经网络的关键点

神经网络的关键点

神经元及网络结构

神经网络的核心



数据

神经网络的输入

权重初始化

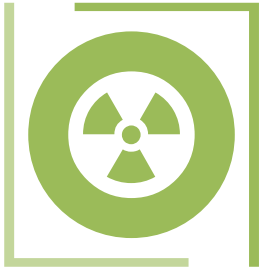
神经网络的起点

基于梯度的优化

神经网络的基本原理

迭代训练

神经网络的产生



损失函数和优化方法

神经网络的优化

神经网络的核心

神经网络的核心元素 —— 神经元

在传统神经网络(多层感知机)中，我们使用全连接层来构建网络，即:

```
paddle.nn.Linear(in_features=100, out_features=256)
paddle.nn.ReLU()
```

所谓全连接，其本质含义是指两个层之间的所有神经元都需要相互连接，我们可以使用 output = relu(dot(W, input)+ b) 来表示这种关系。

其中，W **和** b 都是张量，分别对应该层的kernel和bias属性，它们可以被理解为该层的**权重**或**可训练参数**，这些权重包含网络从训练数据中学到的信息。relu()是**激活函数**，用于实现非线性。

神经网络的核心

神经网络的核心结构 —— 基于神经元的网络

```
# 1. 定义MLP网络
class myMLP(nn.Layer):
    def __init__(self, num_classes=10):
        super(myMLP,self).__init__()
        self.num_classes = num_classes
        self.features = nn.Sequential(
            nn.Linear(in_features=3*100*100, out_features=100),
            nn.ReLU(),
            nn.Linear(in_features=100, out_features=256),
            nn.ReLU(),
            nn.Linear(in_features=256, out_features=256),
            nn.ReLU(),
            nn.Linear(in_features=256, out_features=num_classes)
        )

    def forward(self,input):
        output = self.features(input)
        return output
```

```
---------------------------------------------------------------------------
 Layer (type)       Input Shape          Output Shape         Param #
===========================================================================
   Linear-1       [[10, 30000]]          [10, 100]         3,000,100
    ReLU-1          [[10, 100]]          [10, 100]               0
   Linear-2         [[10, 100]]          [10, 256]          25,856
    ReLU-2          [[10, 256]]          [10, 256]               0
   Linear-3         [[10, 256]]          [10, 256]          65,792
    ReLU-3          [[10, 256]]          [10, 256]               0
   Linear-4         [[10, 256]]           [10, 10]           2,570
===========================================================================
```

以上是构建多层感知机的代码，此处包含四个全连接层(fc_layers)，维度分别是100,256,256和10，它们在前向传播时都会进行一些简单的张量运算，这些运算都包含权重张量。权重张量是这些层的**属性**，里面保存了从数据中学到的**知识（knowledge）**。

最后一层是一个包含10个神经元的softmax分类器。这个部件，我们将在后面进行介绍。

神经网络的输入

输入数据

以**CIFAR10**数据集为例。输入图像被保存在形态为**4D(n,3,32,32)**的float32的Numpy数组（张量）中，其形状为3*32*32它分为训练集(**60000**,3,32,32), 测试集(**10000**,3,32,32)。

所有数据都被归一化到[0,1]之间。并且在输入网络的时，每个样本都会被*reshape*成（1,3*32*32）=（1,3072）的形态。

```
# 根据索引获取单个样本
def __getitem__(self, index):
    # 步骤三：实现 __getitem__ 函数，定义指定 index 时如何获取数据，并返回单条数据（样本数据、对应的标签）
    image_path, label = self.data[index]    # 根据索引，从列表中取出一个图像，并将数据拆分成路径和列表
    img = cv2.imread(image_path, 1)         # 使用cv2进行数据读取可以强制将的图像转化为彩色模式，其中0为灰度模式，1为彩色模式
    img = cv2.resize(img, (100, 100))       # 将图像尺度resize为指定尺寸
    img = np.array(img).astype('float32')   # 将图像数据类型转化为float32（Paddle默认的内部数据格式）
    img = img/255.0                         # 将像素值归一化到[0, 1]之间，仅在MLP中使用

    img = self.transforms(img)              # 调整数据形状paddle默认张量格式
    img = paddle.reshape(img, [3*32*32])    # 将图像拉成一维向量

    label = np.array(label, dtype='int64')  # CrossEntropyLoss要求label格式为int，将Label格式转换为 int

    return img, label
```

神经网络的起点

神经网络的起点 —— 权重初始化

- 神经网络的**权重**被称为可训练参数，这意味着训练的目的是获得合适的权重。
- 在训练开始时，权重需要被初始化。
- **随机初始化**是一种比较好的初始化方法，它使用随机算法将权重矩阵初始化为一些很小的随机值。
- **初始权重值**一般不具任何意义，但这是训练的起点。初始权重会根据反馈信号逐渐进行调节，使其能够更好地反映训练数据的内在特征，这个过程称为**训练**。
- 更好的初始化方法，例如：MSRA(He)、**Xavier** 和使用预训练模型（迁移学习）。

神经网络的产生

神经网络的产生 —— 迭代训练

Step1：抽取训练样本*X*和对应目标*y*组成的数据**批量**；

Step2：在*X*上运行网络 [这一步叫做**前向传播**]，得到预测值*y*_pred；

Step3：计算网络在这批数据上的损失*loss*，用于衡量*y*_pred和*y*之间的距离；

Step4：更新网络的所有权重(*W*和*b*)，使网络在这批数据上的损失*loss*略微降低；

Step5：反复**迭代**以上过程，最终得到的网络(权重)在训练数据上具有非常小的损失*argmin(loss)*，即预测值*y*_pred和预期目标*y*之间的距离非常小。**此时训练结束。**

神经网络的产生

神经网络的产生 —— 迭代训练的技术挑战

在迭代训练的过程中，

- **Step1**，不仅仅是获取输入数据，还包括数据预处理和数据组织
- **Step2**、**Step3**，是一些张量的基本计算
- **Step4**，更新权重是一个难点，其关键是如何判断权重应该增加还是减少？
  - 一种简单的解决方案是，考虑某个标量系数，并固定其他参数，通过求预测值 y_pred和期望值 y 之间的损失loss，来判断权重的调节方向。——但这种方法并不可行，因为一个网络通常有成千上万，甚至数百万千万的参数。
  - 一种更好的方法，是利用网络中所有运算都可微的事实，计算损失相对网络系数的梯度，然后向梯度的反方向改变系数，从而使损失降低。

神经网络的产生

神经网络的产生 —— 迭代训练

```
total_epoch       = 30
# 3. 创建迭代读取器
train_reader = DataLoader(dataset_train, batch_size=64, shuffle=True, drop_last=True)
val_reader = DataLoader(dataset_val, batch_size=64, shuffle=False, drop_last=False)
test_reader = DataLoader(dataset_test, batch_size=64, shuffle=False, drop_last=False)
def train(model):

    for epoch in range(1, total_epoch+1):
        for batch_id, (image, label) in enumerate(train_reader()):
```

现在，我们知道了训练中发生了什么：网络开始在训练数据上进行迭代，每个小批量包含batch_size=64个样本，共迭代了total_epoch=30次 [在所有训练数据上迭代一次叫作一个轮次(epoch)]。在每次迭代过程中，网络会计算批量损失相对于权重的**梯度**，并相应地更新**权重**。30个轮次后，网络进行了28140次梯度更新，每个轮次60000/64=938次，网络损失值将变得足够小，使得网络能够以很高的精度对手彩色图像进行分类。

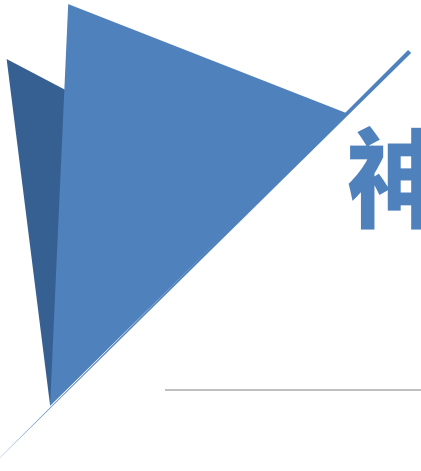
神经网络的优化

损失函数和优化方法

CrossEntropyLoss是交叉熵**损失函数**，又称为**代价函数**，用于衡量预测值和真实值之间的差距。同时我们可以通过metric.Accuracy()来计算平均精度。

```
# 2. 配置模型训练参数
model.prepare(
    paddle.optimizer.Adam(learning_rate=0.001, parameters=model.parameters()),     # 优化函数Adam
    paddle.nn.CrossEntropyLoss(),                                                  # 交叉熵损失函数
    paddle.metric.Accuracy()                                                       # 精度评价指标
    )
```

优化方法在训练过程中它用来学习权重张量的反馈信号，训练的目标就是使它最小化。在训练过程中，减少损失（最小化损失函数）是通过带动量的小批量随机梯度下降来实现的，具体的方法由**优化器**(optimizer.Adam)来决定。并且梯度下降的步长step用**学习率** learning_rate来进行控制。

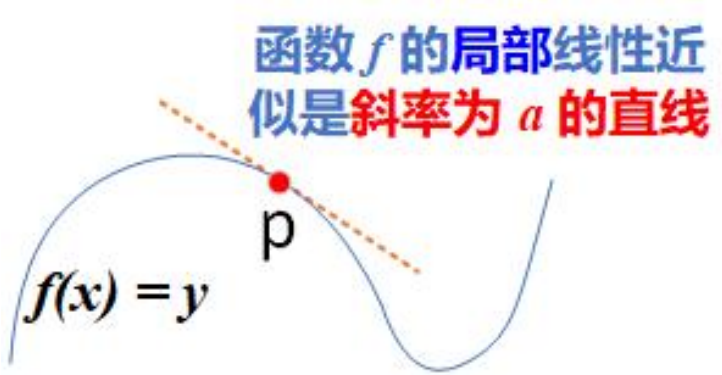


神经网络的基本原理——导数和梯度

什么是导数

函数的连续和光滑

假设有一个连续的光滑函数 $f(x)=y$，将实数 x 映射为另一个实数 y。由于函数是**连续的**，x 的微小变化只能**导致** y 的微小变化——这就是函数连续性的直观解释。



假设 x 增大了一个很小的因子 $\varepsilon_x\,(epsilon\_x)$，这会导致 y 也发生了很小的变化，即 $\varepsilon_y\,(epsilon\_y)$：

$$f(x+\varepsilon_x)=y+\varepsilon_y$$

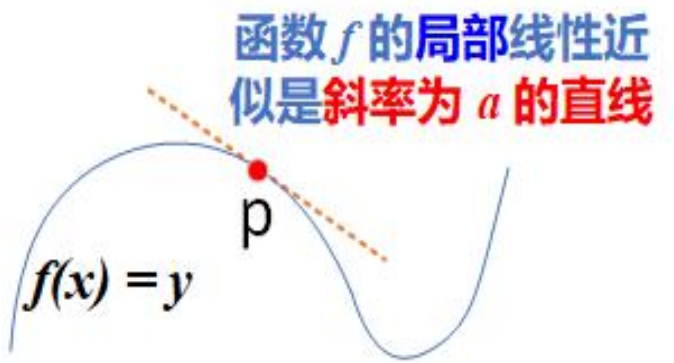
此外，由于函数是**光滑的**（函数曲线没有突变角度），在某个点 p 附近，如果 $epsilon\_x$ 足够小，则可以将 f 近似为斜率为 a 的线性函数，这样 $epsilon\_y$ 就变成了 $a*epsilon\_x$，则有：

$$f(x+\varepsilon_x)=y+\mathrm{a}*\varepsilon_x$$

什么是导数

函数的连续和光滑

显然，只有在x足够接近 p 时，这个线性近似才有意义。斜率*a* 被称为 *f* 在 p 点**导数**(*derivative*)。



- 若 ***a*** 是负值，则 *f(x)* 是减函数， ***x*** 在 p 点附近的变化将导致 ***f(x)*** 减小；
- 若 ***a*** 是正值，则 *f(x)* 是增函数， ***x*** 在 p 点附近的变化将导致 ***f(x)*** 增大。
- ***a*** 的绝对值的大小（导数大小），表示增大或减小的速度快慢。

对于每个可微函数 *f(x)*（**可微**的意思是“可以被求导”。例如，光滑的连续函数可以被求导），都存在一个导函数 *f'(x)*, 能够将 ***x*** 映射为 ***f*** 在该点的局部线性近似的斜率。

如果想要将 ***x*** 改变一个小因子 ***epsilon_x***，目的是将 ***f(x)*** 最小化，那么只要知道 ***f*** 的导数，那么问题就解决了。由于导数完全描述了改变 ***x*** 后 ***f(x)*** 会如何变化，因此，如果希望减小 ***f(x)*** 的值，只需将 ***x*** 沿着导数的反方向移动一小步。

张量运算的导数——梯度

从导数到梯度

- **梯度（gradient）**是张量运算的导数。它是导数这一概念从一元函数向多元函数导数的推广。多元函数是**以张量作为输入**的函数。

假设有一个输入向量 x，一个矩阵 W、一个目标 y 和一个损失函数 $loss$。则，可以通过 W 来计算预测值 y_pred，然后计算损失，即预测值 y_pred 和目标 y 之间的距离。

$$y\_pred = \mathrm{dot}(W, x)$$

$$loss\_value = loss(y\_pred, y)$$

如果输入数据 x 和 y 保持不变，那么可以将$loss$看作是将W映射到损失值的函数。

$$loss\_value = f(W)$$

张量运算的导数——梯度

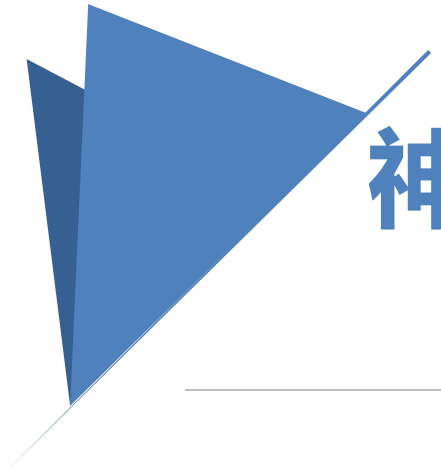
从导数到梯度

假设 W 的当前值为W_0，则 f 在 W_0点的导数是一个张量 $gradient(f)(W_0)$，可表示为$\nabla_W f(W_0)$，其形状与 W_0 相同，它每个元素的值 $\nabla_W f(W_0)$[i, j,..., n] 表示改变 W_0[i, j,..., n] 时，损失值 $loss\_value$ 变化的方向和大小。

函数 $f(w)$ 的曲线在 p 点的斜率 ➡ **单变量函数 $f(w)$ 的导数**

$f(W)$ 在 W_0 附近曲率的张量 ➡ **多元函数 $f(W)$ 的梯度**

与函数$f(w)$的导数类似，对于张量W的函数$f(W)$，可以通过将W向梯度的反方向移动来减小$f(W)$，例如：$W_1 = W_0 - step * \nabla_W f(W_0)$。其中，$step$是一个很小的比例因子。也就是说，沿着曲率的反方向移动，直观上看在曲线上的位置会更低。其中，比例因子$step$是必需的，与 p 点导数相似，$\nabla_W f(W_0)$只是在W_0附近曲率的近似值，不能离W_0太远。



神经网络的基本原理二
——随机梯度下降

随机梯度下降

解析法失效了

给定一个可微函数，理论上可以使用解析法求其最小值，即导数为0的点所对应的函数位置的值。

对于一个神经元来说，理论上，可以通过其梯度方程 $\nabla_{\boldsymbol{W}} f(\boldsymbol{W_0}) = \mathbf{0}$ 求解权重$\boldsymbol{W}$。

然而，事实上这并不现实。对于任何一个神经网络来说，权重参数的个数至少都有数千，甚至上百万，**解析法**无法进行求解。

事实上，我们可以通过前面介绍的**五步循环训练方法**来实现参数的优化：基于当前在随机数据批量上的损失，一点一点地对参数进行调节，沿着多元函数梯度的反方向进行权重更新，损失每次都会变小一点。

随机梯度下降

五步循环训练方法

Step1：抽取一定数量的训练样本X和对应目标y组成的数据**批量**；

Step2：在X上运行网络，得到预测值y_pred；

Step3：计算网络在这批数据上的损失loss，用于衡量y_pred和y之间的距离；

Step4：计算网络相对于网络参数的梯度[一次**反向传播(backward pass)**]

Step5：将参数沿着梯度的反方向移动一点，比如$W = W - step * gradient$，从而使这批数据上的损失减小一点。

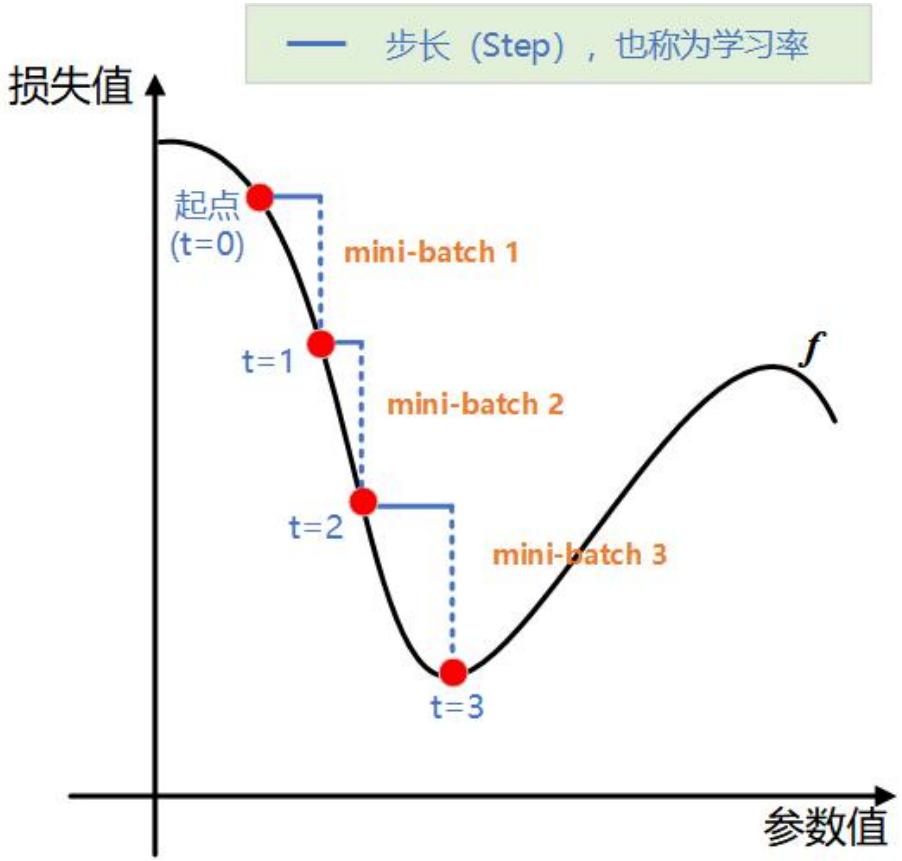
反复**迭代**以上过程，最终得到在训练数据上具有非常小损失时的网络权重。**此时训练结束。**

随机梯度下降

随机梯度下降

以上的过程被称为**小批量随机梯度下降**(mini-batch SGD)。右图给出了样本随机梯度下降的示例。直观上看出，比例因子step的取值非常重要。

- ✓ 取值太小，则沿曲线下降需要很多次迭代，且可能会陷入局部极小点；
- ✓ 取值太大，则更新权重后，可能会出现在曲线上完全随机的位置。

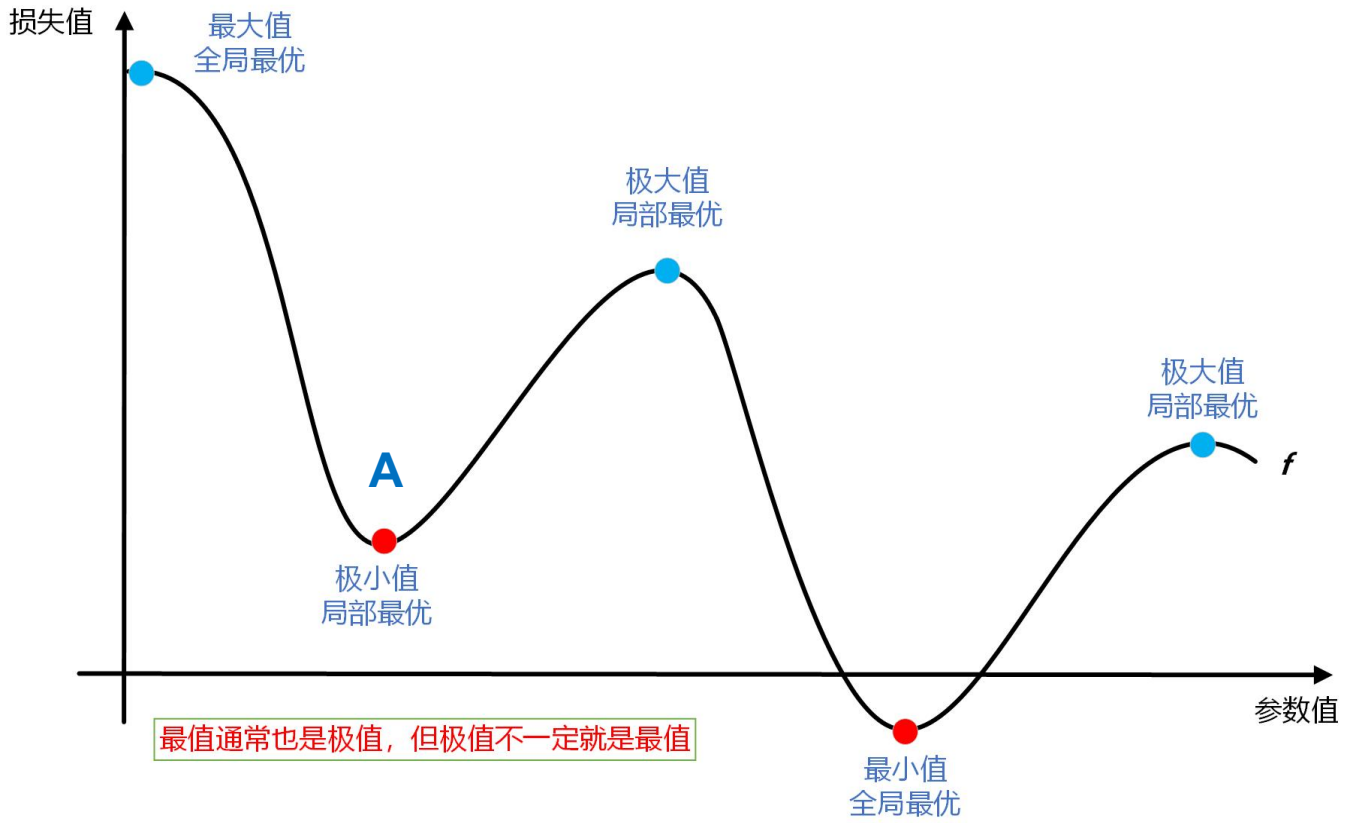


沿着一维损失函数曲线的随机梯度下降（仅一个需要学习的参数）

随机梯度下降

局部最优问题

局部极小点和全局最小点

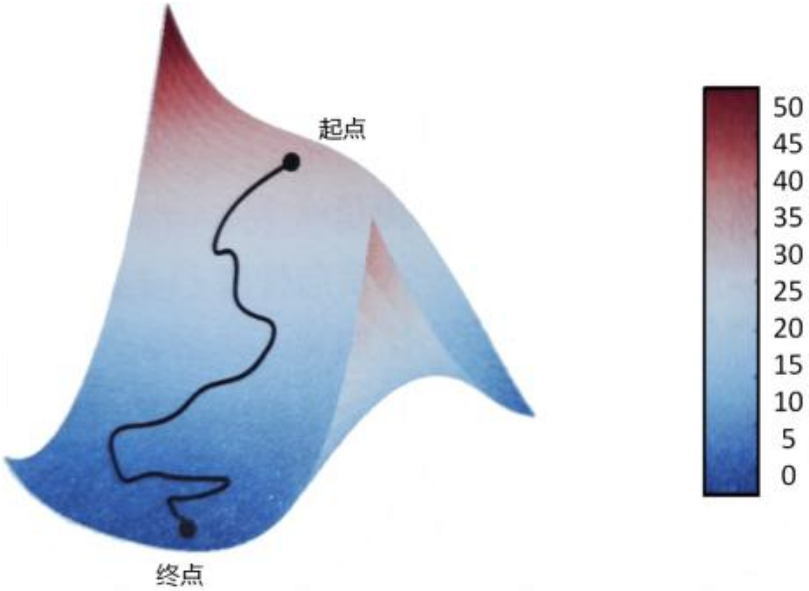


如图所示，在某个参数附近存在一个**局部极小点**(local minimum)，在这个点附近，无论向左还是向右都会导致损失值增大。如果使用小学习率(step)的SGD进行优化，那么优化过程就会陷入局部极小点，导致无法找到全局最小点。

随机梯度下降

优化方法

值得注意的是，上面的图例仅表示出了一维参数空间的梯度下降。但实际的网络可能有数百万的维度。下图给出了一个二维曲面内的梯度下降示例。



沿着二维损失曲面的梯度下降

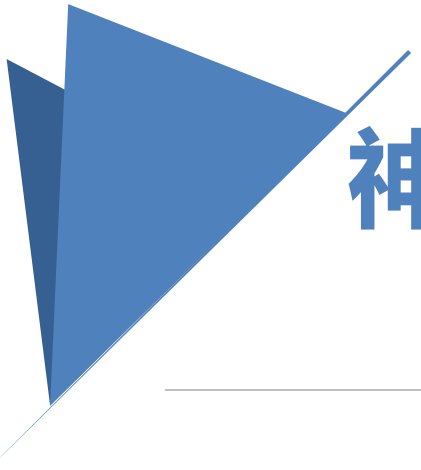
- 为了解决SGD的弊端，很多变种被提出，其特点是计算下一次权重的时候还考虑了上一次权重的更新。例如，带动量的SGD、Adagrad、RMSProp、Adam等。

- **动量（Momentum）**是一个重要的概念，它被应用到很多变体中，**动量**解决了两个问题：**收敛速度**和**局部最优点**。

随机梯度下降

带动量的SGD代码示例

```
1    past_velocity = 0          // 过去的速度
2    momentum = 0.9             <------ 不变的动量因子
3    while loss > 0.01          <------ 优化循环，当loss小于0.01时停止训练
4        w, loss, gradient = get_current_parameters()      // 后去当前参数和损失
5        velocity = past_velocity * momentum – learning_rate * gradient
6        w = w + momentum * velocity – learning_rate * gradient
7        past_velocity = velocity      // 将当前更新量赋值给past用于下一轮更新
8        update_parameter(w)           // 更新参数
```



神经网络的基本原理三
——链式法则和反向传播算法

链式求导：反向传播算法

链式法则和反向传播算法

神经网络包含许多连接在一起的张量运算，每个运算都有简单的、已知的导数。例如：某个网络f包含3个张量运算a, b和c，还有3个权重矩阵$\boldsymbol{W_1}, \boldsymbol{W_2}$ 和$\boldsymbol{W_3}$，则网络可以表示为：

$$\boldsymbol{f(W_1, W_2, W_3) = a(W_1, b(W_2, c(W_3)))}$$

根据微积分的知识，这种函数可以利用下面的恒等式进行求导，它称为**链式法则**(chain rule)：

$$\boldsymbol{m(x) = f(g(x))}, \text{则 } \boldsymbol{m'(x) = f'\big(g(x)\big) * g'(x)}$$

将链式法则应用于神经网络梯度值的优化计算，得到的算法叫**反向传播算法（Back-propagation, BP）**（或反式微分）。反向传播从最终损失值开始，从最顶层反向作用至最底层，利用链式法则逐层计算每个参数对损失值的贡献大小。

由于很多深度学习框架都支持**符号微分**，因此我们无需手动实现反向传播，只需要理解基于梯度的优化方法的工作原理即可。

初识神经网络

到目前为止

我们已经了解神经网络的大部分知识

初识神经网络

小　结

- **学习**是指找到一组模型参数，使得在给定的训练数据样本和对应目标值上的损失函数最小化。
- **学习的过程**：随机选取包含数据样本及其目标值的小批量，计算批量相对于网络参数的梯度，随后将网络参数沿着梯度的反方向稍稍移动。利用反向传播算法（基于张量导数和链式法则）可以实现整个网络参数的更新。
- **损失**是在训练过程中需要最小化的量，它可以衡量当前任务是否已经成功解决。
- **优化器**是使用损失梯度更新参数的具体方式。
- **DeepLearning 中如何选择（现阶段）？**
  1. 首选mini-batch SGD or Adam
  2. 根据GPU最大承受能力（略有冗余）设置批大小。

欧老师的联系方式

读万卷书 行万里路 只为最好的修炼

QQ：14777591 (宇宙骑士)

Email：ouxinyu@alumni.hust.edu.cn

Website: http://ouxinyu.cn

Tel：18687840023