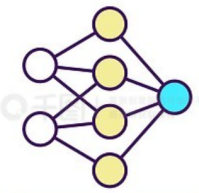


神经网络

王思图[1] Agent body department,SAST,Tsinghua University

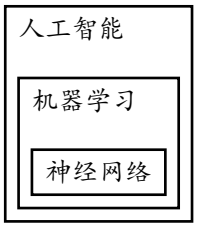
一. 什么是神经网络

一.I. 神经网络的简单定义

神经网络是由包含或不包含参数的简单处理单元相互连接构成的大规模结构。它能够将数据经过由连接和参数决定的流程进行计算，将输入映射到输出。

一.II. 神经网络能够处理的问题

一.II.1. 神经网络方法在人工智能领域的定位:



一.II.2. 神经网络常用来处理什么样的问题:

- 从数据中获取并保存信息（数据驱动任务）
- 拥有结构化的、相对大量的结构化输入-输出对数据
- 有明确、可计算的性能评估指标
- 常见的有分类、预测、生成任务

一.II.3. 神经网络的优点:

- 可以形成拥有大规模参数的结构：信息提取和保存能力强，泛化性能好
- 可扩展性：结构化的输入和输出，便于扩展和复用
- 非线性：从输入特征空间到输出特征空间的映射是非线性的

二. 学习任务的组成

二.I. 学习任务

学习任务是由模型 M、经验 E、任务 T、性能量度 P 组成的。模型的学习是指模型 M 通过得到经验 E，在性能量度 P 意义下相对得到经验 E 并运行学习算法前在任务 T 上有所改进。

对于神经网络而言，模型 M 即为神经网络本身，经验 E 为数据，性能量度 P 为损失函数，而任务 T 即为希望使用神经网络解决的机器学习问题。

例如：神经网络模型的学习任务是指该神经网络在图像分类任务上进行学习，在获得标注后的图片分类数据并进行训练后，在分类准确率标准下相较于训练前有所提升。

Published Jul 10, 2024

在神经网络学习任务中，我们称使用数据作为经验 E，对神经网络使用学习算法进行优化的过程为训练

二.*II.* 数据与经验

对任务 T 进行数学建模：

> 对于某个样本空间已知而概率未知的概率空间$(\Omega, \mathcal{F}, \mathbb{P})$，希望通过独立同分布的采样结果$\{X_i\}$~$\mathbb{P}(x)$ $i.i.d$ 估计分布$\mathbb{P}$。 我们称采样结果$\{X_i\}$为数据

从中可以得到几个数据驱动机器学习任务的经典假设： 1。 真实分布假设：认为数据符合某个确定的真实分布 2。 独立同分布假设：认为数据是从真实分布中独立同分布采样的 3。 低维流形假设：自然的原始数据是低维的流形嵌入于原始数据所在的高维空间（无法通过有限空间和时间估计无穷维样本空间上的概率分布）

二.*III.* 损失函数与性能量度

我们需要一个能够不需要人类参与的性能度量算法，以便神经网络自行迭代而无需人类的干预，从而显著提升效率。

通常来说，性能量度 P 是特定于任务 T 而言的。 而由于人类经验的复杂性，我们通常只能够近似地评估模型的性能。 在神经网络学习任务中，性能量度 P 被称为损失函数：

二.III.1. 损失函数*:*

> 对于给定的模型 M，和给定的经验 E，损失函数定义为函数$\mathcal{L}:(M,E)\to\mathbb{R}$，损失函数值越大，代表人类认为模型 M 在任务 T 上的表现与经验 E 的表现差距越大

比如，在图像分类任务中，$\mathcal{L}(M,E)$可以为在某一组标注好类别图像数据上，模型在预测这组图像类别时的错误率。

二.III.2. 训练集与测试集*:*

在神经网络的评估中，一个重要的指标是其泛化能力，即对于在经验 E 没有涉及的、同时位于任务 T 的样本空间范围内的点的估计效果。 通常我们会从训练数据中单独分出一小部分，在训练时不作为样本对模型进行训练，并评估模型在这些样本上的性能。 我们称 在训练时作为样本对模型进行训练的数据集合为数据集，分出来的那一小部分为测试集。 通常，为了得知模型在训练过程中的性能变化趋势，我们还会从测试集中分出一小部分作为验证集，在训练中途多次在验证集上测试模型的性能表现。

三. 神经网络的构成

我们在第一节强调了神经网络构成的两个部分：带参数或不带参数的简单结构，以及它们的连接。 在本节中将详细说明这两部分。

三.*I.* 感知机

线性感知机算法（PLA）1957 年由 Frank Rosenblatt 提出。 感知机是二分类的线性分类模型，其输入为实例的特征向量，输出为实例的类别，取值为+1 和-1。

perceptron:

$$f(\vec{x}) = sign(\vec{w}^T\vec{x} + b) \tag{1}$$

，where $w, x \in \mathbb{R}^n, b \in \mathbb{R}$

显然，这是一个含参数w的，且能够将输入根据参数w唯一映射到输出的简单结构。

感知机是神经网络中最常使用、最简单的含参结构。虽然感知机只能进行简单的线性可分分类问题，但通过连接大量感知机，以及更换符号函数为其他非线性函数，我们可以构造出复杂的结构。

三.II. 多层感知机 MLP

多层感知机是通过感知机的拼接组成的结构。它是最简单的神经网络结构。这个结构在几乎所有的现代深度神经网络中都存在。

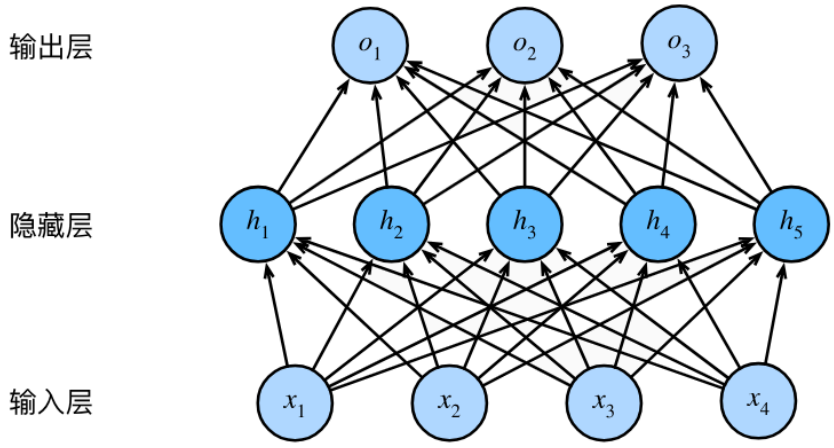
multiple layer perceptron:

$$f(\vec{x}) = f_N \circ \cdots \circ f_0 \tag{2}$$

$$f_{i(\vec{x})} = \sigma_{i(W\vec{x}+\vec{b})} \tag{3}$$

，where $W \in \mathbb{R}^{n_i \times n_{i+1}}, b \in \mathbb{R}^n_{i+1}, i \in \{0, \ldots, N\}$

其结构可以表示为：



其中的h_i、o_i代表一个将符号函数替换为非线性激活函数的感知机。（尝试根据图中给出的结构，推导上面给出的公式。进一步的，写出W_1, W_2与各感知机参数之间的关系）

我们可以发现，仅通过连接感知机，我们就获得了一个可以任意增大规模而基础结构相同的，从任意维欧式空间向任意维欧式空间映射的结构。注意到，由于引入了非线性激活函数（试想如果使用线性激活函数会怎样?），这个结构还可以表示非线性的映射。于是我们获得了一个典型的神经网络模型。

四. 神经网络的学习算法

四.I. 神经网络的优化

在第二节我们探讨了学习任务，并定义了神经网络中的性能量度——损失函数。在上一节，我们获得了一个典型的神经网络模型。在这一章，我们自然需要讨论如何通过使用经验 E，即数据，使得神经网络在损失函数意义下获得性能提升。

四.II. 最小化损失函数

注意到损失函数的定义，我们假定了模型性能越差，损失函数越大。也就是说，我们只需要反过来通过调整 M，在经验 E 上最小化损失函数，就能达到提高模型 M 在任务 T 上表现的目的。于是，神经网络的优化任务可以如下形式化定义：

$$\theta^* = \arg\min_{\theta} \mathcal{L}(M_\theta, E) \tag{4}$$

其中θ是模型中所有的可调整参数。通过损失函数，我们成功将主观的性能评估和改进问题，转化为了计算和优化损失函数的问题，进而可以通过数值方法解决。可以认为，通过设计损失函数，我们将主观性转移至了损失函数中。

五. 基于梯度的优化方法

在数值计算领域，优化问题的最常用、效果最好的算法绝大部分是基于梯度的算法。由于神经网络的简单基本结构的可导性以及参数空间的连续性，我们可以方便地使用梯度法作为优化方法。

五.I. 梯度下降算法

基于梯度的优化方法中最易理解和实现的是梯度下降算法：

令

$$\theta_{n+1} = \theta_n - \eta \frac{\partial \mathcal{L}\left(M_{\theta_n}, E\right)}{\partial \theta_n} \tag{5}$$

则对于一类性质较好的$\mathcal{L}$和M_θ，我们有

$$\lim_{\eta \to 0, n \to \infty} \theta_n = \theta^* \tag{6}$$

我们知道梯度的反向是该点邻域中函数值下降最快的方向，因此当函数不太差时，沿梯度的反向进行参数的更新都有机会达到函数的最小值。

然而梯度下降算法有两个缺点。首先，该方法对于函数的性质有要求，否则对于参数的初值较为敏感，容易陷入局部极小值。(考虑有两个谷的函数)。其次，该方法需要在全部样本上计算出梯度，并求平均。这极大的降低了梯度下降的效率。

五.II. 随机梯度下降算法(SGD)

随机梯度下降算法是目前几乎所有神经网络的学习算法。目前已知的绝大多数算法都是基于这个方法的优化或微调。或者说，今天我们所讲的神经网络就是指能够使用随机梯度下降算法进行训练的神经网络。

令

$$\theta_{n+1} = \theta_n - \eta \frac{\partial \mathcal{L}\left(M_{\theta_n}, \{X_i\} \subset E\right)}{\partial \theta_n} \tag{7}$$

则对于大多数$\mathcal{L}$和M_θ，我们有

$$\lim_{\eta \to 0, n \to \infty} \theta_n = \theta^* \tag{8}$$

这个算法的证明设计到一些统计知识，在此暂且不证明。(如有兴趣，可以考虑: 1。 梯度在样本分布下的无偏估计是什么? 2。 考虑使用部分样本进行梯度计算时引入的噪声ε。这对于处于局部极小值的参数的梯度计算有什么影响?)

随机梯度下降法使得神经网络能够通过每次选取部分数据计算梯度，根据学习率η更新

五.III. 梯度的计算方法

理论上，给定任何确定的网络结构，我们都可以写出梯度的解析表达。 然而当网络规模增大时，这显然是不可实现的。 因此我们需要寻找数值方法计算参数的梯度。

1。 数值微分法

$$\widehat{\theta_{n+1,i}} = \widehat{\theta_{n,i}} - \eta \frac{\mathcal{L}\left(M_{\widehat{\theta_{n,i}}+\varepsilon}, \{X_i\}\right) - \mathcal{L}\left(M_{\widehat{\theta_{n,i}}-\varepsilon}, \{X_i\}\right)}{\varepsilon} \tag{9}$$

该方法的时间复杂度是$\mathcal{O}(N) \times \mathcal{O}(\text{forward})$，$N$是参数规模。 通常来说，$\mathcal{O}(\text{forward})$ 与参数量成近似线性关系，因此总的复杂度为$\mathcal{O}(N^2)$。 这个复杂度在参数量很大(现代深度神经网络的参数量通常在千万到百亿级别)时是不可接受的。

我们注意到，在进行网络中靠近输出部分的参数的梯度计算时，两次正向传播中有大量的重复计算。 因此我们希望寻找一种能够复用计算结果的方法来降低重复计算的开销。

2。 反向传播 反向传播法是现有的神经网络计算梯度的最优方法。 现代深度神经网络训练框架全部使用反向传播进行梯度计算。

- 链式法则

回顾微积分中的链式法则。

令 $f: \mathbb{R}^a \to \mathbb{R}^b, g: \mathbb{R}^b \to \mathbb{R}^c, c \in \mathbb{R}^a$，则有

$$\frac{\partial\left(g^{(i)}\left(\vec{f}(\vec{x})\right)\right)}{\partial x^{(j)}} = \sum_{t=1}^{b} \frac{\partial g^{(i)}\left(f^{(1)}(\vec{x}), \ldots, f^{(b)}(\vec{x})\right)}{\partial f^{(t)}(\vec{x})} \cdot \frac{\partial f^{(t)}(\vec{x})}{\partial x^{(j)}} \tag{10}$$

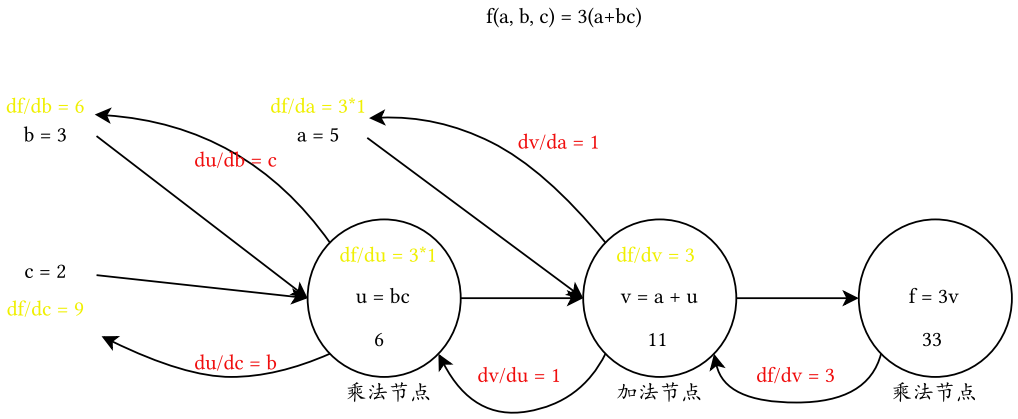
可以发现，对于任何完全由复合函数构成的函数，我们都可以将其分为两部分进行计算，这两部分都仅与构成复合函数的部分自身相关。 回顾神经网络的定义，我们发现神经网络确实由这样可导的简单部分相互连接(即复合)构成的。 这表明我们对于神经网络的任意分割，都可以分别计算它们各自的参数的梯度，再将其按照网络结构的连接关系进行组合得到整个网络的参数的梯度。

- 计算图

计算图是一种有向无环图，其中节点表示计算操作，边表示数据流。计算图提供了一种清晰的方式来表示复杂的计算过程，并允许方便的使用使用链式法则来计算梯度

计算图由两部分组成: 节点上的运算(通常用圈表示)，以及复合关系(通常用箭头表示)。一个神经网络唯一对应了一个计算图。

下面是一个典型的计算图:



计算图最重要的一点是实现了局部计算: 对于每个节点，我们只需要保存正向过程的输入，并且获得下一级节点的梯度，就可以函数关于本节点的梯度。

- 反向传播算法

反向传播算法使用计算图来组织和管理计算过程，从而有效的计算梯度。反向传播通过从计算图的输出节点开始计算出误差的梯度，并沿着计算图将其逐层传递到输入节点。这个过程被称为梯度的反向传播。通过一次从输出到输入的完整反向传播过程，我们就可以获得所有参数关于损失函数的梯度。

- 局部计算与实现的简便性

局部计算除了通过信息的复用显著降低了梯度求解的时间复杂度以外，也方便了使用现代编程语言实现神经网络。我们可以简单的将每个计算节点实例化为一个对象，这些相互不耦合的对象可以通过数据的传递完成一个完整神经网络的功能。下面给出上方计算图的实现，以及其对应的神经网络在某个训练集上的训练过程。我们假设这个神经网络的输入是 a，输出为乘法节点的输出，即 ab。该神经网络只含有一个参数 b。我们选用的损失函数为$L(\text{out}, \text{label}) = 3 \times (\text{label} + \text{out})$。(请自行验证该神经网络的计算过程符合上方计算图)。我们给出以下实现:

```
class MulNode:
  def __init__(self):
    self.x = None
    self.y = None

  def forward(self, x, y):
    self.x = x
    self.y = y
    return x * y

  def backward(self, d_next):
    return self.y, self.x
```

```
class AddNode:
  def forward(x, y):
    return x + y

  def backward(d_next):
    return d_next, d_next

class MyNetwork:
  def __init__(self):
    self.node1 = MulNode()
    self.node2 = AddNode()
    self.node3 = MulNode()

  def forward(self, a, b, c):
    h1 = self.node1.forward(b, c)
    h2 = self.node2.forward(a, h1)
    return self.node3.forward(3, h2)

  def backward(self, d_out):
    _, d_h2 = self.node3.backward(d_out)
    d_a, d_h1 = self.node2.backward(d_h2)
    d_b, d_c = self.node1.backward(d_h1)

model = MyNetwork()
lr = 1e-4
for data, label in dataset:
  parameter = b
  loss = model.forward(label, parameter, data)
  d_parameter = model.backward(1)
  parameter -= lr * d_parameter
```

六. 机器学习任务中的