深度学习快速入门

11 线性层及其他层

POET

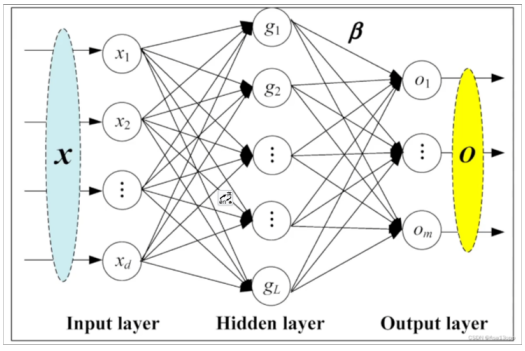
2024 年 2 月 16 日

1 神经网络

神经网络是一种受到人脑神经元结构启发的计算模型，用于机器学习和人工智能任务。它由大量相互连接的人工神经元（也称为节点或神经元）组成，这些神经元按层次排列，分为输入层、隐藏层（可以有多个）和输出层。

主要组成部分

1. **输入层（Input Layer）：**接受外部输入数据的层次。每个输入节点代表输入数据的一个特征。
2. **隐藏层（Hidden Layer）：**位于输入层和输出层之间的一层或多层神经元。每个神经元接收来自上一层神经元的输入，通过权重和偏置进行加权求和，然后输入到激活函数中。
3. **输出层（Output Layer）：**提供网络的输出，通常代表模型的预测或分类结果。



工作原理

神经网络的基本工作原理是通过学习数据中的模式和规律来进行预测或分类。在训练阶段，网络接收输入数据，通过前向传播计算输出，然后与真实结果比较，利用反向传播算法调整权重和偏置，减小预测误差。这个过程重复进行，直到网络能够准确地预测或分类新的未见过的数据。

主要概念

1. 权重（Weights）：用于调整输入信号的影响力。较大的权重表示该输入特征对输出的影响较大。

2. 偏置（Bias）：为每个神经元提供一个偏移量，使得即使所有输入特征为 0，神经元仍然能激活

3. 激活函数（Activation Function）：将神经元的输入转换为输出的函数。常用的激活函数包括 Sigmoid、ReLU（Rectified Linear Unit）和 Softmax 等。

4. 损失函数（Loss Function）：用于度量预测结果与实际结果之间的差异。训练过程中的目标是最小化损失函数的值。

5. 优化算法（Optimization Algorithm）：用于调整网络的参数（权重和偏置）以最小化损失函数。常用的优化算法包括梯度下降法和其变种（如随机梯度下降）。

2 线性层

线性层（Linear Layer）是神经网络中的一种基本层次，也被称为全连接层（Fully Connected Layer）或稠密层（Dense Layer）。这一层的每个节点与前一层的每个节点都有连接，每个连接都有一个权重，用来调整输入数据的影响力，同时还有一个偏置（bias），用于增加网络的灵活性。线性层通常位于神经网络的隐藏层或输出层。

工作原理

在线性层中，输入数据被展平成一个向量，每个输入特征与相应的权重相乘，然后将所有结果相加，并加上偏置。这个操作可以用数学表达式表示为：

$$\text{输出} = \sum_i (\text{输入}_i \times \text{权重}_i) + \text{偏置}$$

这一操作实际上是一个线性变换，将输入数据映射到输出空间。线性层的输出可以作为下一层神经网络的输入，也可以是最终的输出结果，具体取决于网络的结构和任务需求。

作用

1. 特征提取：线性层通过学习合适的权重和偏置，可以自动地学习输入数据中的特征，这些特征可以用于后续的任务。

2. 非线性建模：单独的线性变换无法捕捉复杂的非线性关系，但是通过堆叠多个线性层，并在它们之间加入非线性激活函数，神经网络可以学习到更为复杂的非线性关系。

用途

线性层通常用于回归任务，其中网络的最后一层是一个线性层，直接输出数值。它也常用于分类任务的中间层，通过学习特征表示，帮助网络更好地理解输入数据。在深度学习中，线性层通常和非线性激活函数（如ReLU）交替使用，构成神经网络的基本组成部分，被广泛应用于各种任务中。

线性拉平

```
import torch
import torchvision
from torch import nn
from torch.nn import Linear
```

```
5     from torch.utils.data import DataLoader
6     from torch.utils.tensorboard import SummaryWriter
7
8     dataset = torchvision.datasets.CIFAR10("./dataset
          ",train=False,transform=torchvision.transforms.
          ToTensor(),download=True)
9     dataloader = DataLoader(dataset, batch_size=64,
          drop_last=True)
10
11    class Tudui(nn.Module):
12        def __init__(self):
13            super(Tudui, self).__init__()
14            self.linear1 = Linear(196608,10)
15
16        def forward(self, input):
17            output = self.linear1(input)
18            return output
19
20    tudui = Tudui()
21    writer = SummaryWriter("logs")
22    step = 0
23
24    for data in dataloader:
25        imgs, targets = data
26        print(imgs.shape)
27        writer.add_images("input", imgs, step)
28        output = torch.reshape(imgs,(1,1,1,-1)) # 方法
              一：拉平
29        print(output.shape)
30        output = tudui(output)
31        print(output.shape)
```

```
32            writer.add_images("output", output, step)
33            step = step + 1
```