目录

1.全连接层的作用是什么?

1. 全连接层（fully connected layers，FC）在整个卷积神经网络中起到分类器模块的作用。在深度学习中，卷积层、池化层和激活函数层等操作将原始数据映射到隐层高维特征空间，全连接层则将这些学到的隐层高维特征映射到样本标签（label）空间中。在实际使用中，全连接层可由卷积操作实现：对前层是全连接的全连接层可以转化为卷积核为 1×1 的卷积；而前层是卷积层的全连接层可以转化为卷积核为 $h \times w$ 的全局卷积， h 和 w 分别为前层卷积结果的高和宽。

以VGG-16为例，对于224x224x3的输入，经过最后一层卷积的输出为7x7x512，如果后层是一个含4096个神经元的FC层，那么可用卷积核为7x7x512x4096的全局卷积来实现这一全连接运算过程，其中该卷积核参数如下：

“filter size = 7, padding = 0, stride = 1, D_in = 512, D_out = 4096”

经过此卷积操作后可得输出为1x1x4096的特征矩阵。如需再次叠加一个2048的FC层，则可设定参数如下的卷积层操作：

“filter size = 1, padding = 0, stride = 1, D_in = 4096, D_out = 2048”

2. 由于全连接层参数存在冗余（仅全连接层参数就可占整个传统深度学习网络参数的80%左右）的情况，一些经典的传统深度学习模型（ResNet和GoogLeNet等）均用全局平均池化（global average pooling，GAP）层来取代FC层去融合模型学到的深度特征，后续再接softmax等损失函数作为传统深度学习模型的目标函数来指导训练过程。同时研究发现，用GAP层替代FC层的模型通常有较好的预测性能。

3. 学术界研究发现FC层可在模型表示能力的迁移过程中充当“防火墙”的作用。具体来讲，假设在ImageNet上预训练得到的模型为 $\mathcal{M}$ ，则ImageNet可视为源域（迁移学习中的source domain）。微调（fine tuning）是深度学习领域最常用的迁移学习技术。针对模型的微调过程，如果目标域（target domain）中的图像与源域中图像差异巨大（假设相比ImageNet，目标域图像不是物体为中心的图像，而是智慧城市场景数据），不含FC层的网络微调后的结果要差于含FC层的网络。因此FC层可视作模型表达能力的“防火墙”，特别是在源域与目标域差异较大的情况下，FC层可保持较大的模型capacity从而保证模型表达能力的迁移，这是FC层冗余参数带来的优势。

4. 在Transformers中，FC层重新繁荣，成为了Transformers架构模型的标配，成为AI领域中不可获取的关键部分。正是因为FC层能够保持大模型的capacity能力，如果没有全连接层，Self-Attention层输出的只有一些线性表达特征，表达能力有限，而全连接层可以自己学习复杂的特征表达。

2.介绍一下MLP网络

多层感知器（Multilayer Perceptron, MLP）可以说是最基本的神经网络，由Frank Rosenblatt于1957提出，一直广泛应用于各种机器学习和深度学习任务中。

在传统深度学习时代，由于卷积神经网络的出现，一定程度上MLP网络的使用频率有所减少，但是在Transformer发布后，MLP结构重新站上AI行业的舞台。**在现在的AIGC时代中，MLP结构已经成为AIGC模型的重要组成部分。**

下面是Rocky对MLP网络的详细讲解：

1. MLP的基本结构

MLP由一个输入层、一个或多个隐藏层和一个输出层组成。每一层都包含多个神经元（或节点），这些神经元之间是全连接的，即每个神经元的输出连接到下一层的每个神经元。

1.1 输入层

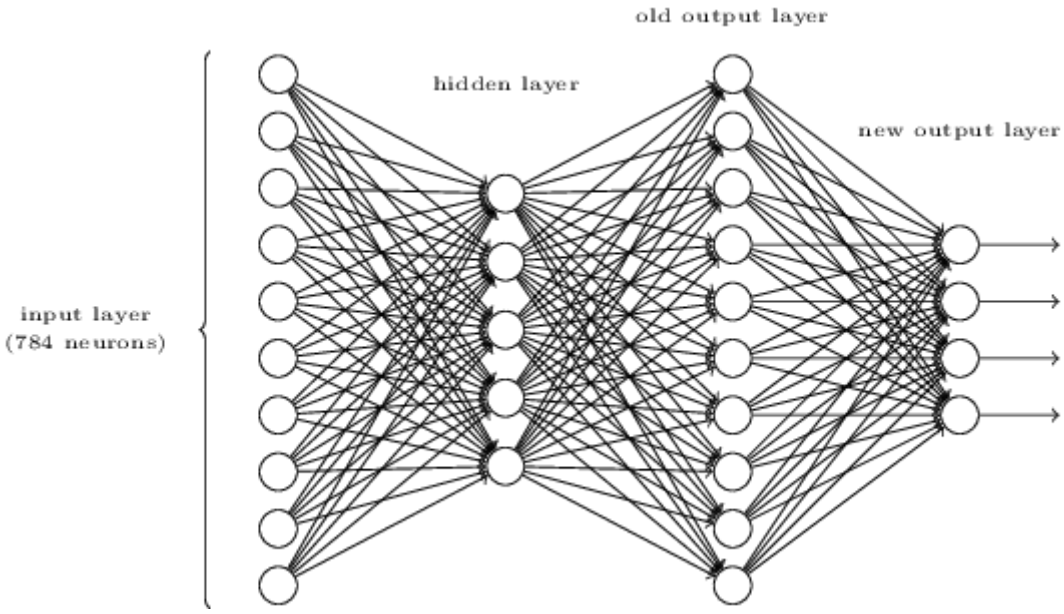
输入层的神经元数等于输入数据的特征数。如果输入是一个包含28x28像素的图像，则输入层的神经元数为784。

1.2 隐藏层

隐藏层由一个或多个层组成，每层包含若干个神经元。隐藏层的数量和每层的神经元数是超参数，我们可以根据不同的场景进行对应的设置。隐藏层通过激活函数（如ReLU、Sigmoid、Tanh等）引入非线性，使得MLP能够学习复杂的分布和特征。

1.3 输出层

输出层的神经元数取决于具体的AI任务。例如，对于二分类任务，输出层通常包含一个神经元；对于多分类任务，输出层的神经元数等于类别数。



2. MLP的工作原理

MLP的工作原理基于前向传播和反向传播两个过程。

2.1 前向传播

在前向传播过程中，输入数据通过网络层层传递，经过每一层的加权和激活函数计算，最终得到输出结果。具体步骤如下：

1. **加权求和**：每个神经元接收前一层的输出，通过权重进行加权求和，并加上一个偏置项。

$$z_j = \sum_i w_{ij} \cdot x_i + b_j$$

其中， w_{ij} 是权重， x_i 是输入， b_j 是偏置项。

3. **激活函数**：对加权求和的结果应用激活函数，引入非线性。

$$a_j = \sigma(z_j)$$

其中， σ 是激活函数，常见的激活函数包括ReLU、Sigmoid和Tanh等。

4. **输出**：将激活函数的输出传递给下一层，直到最后一层得到最终输出。

2.2 反向传播

在反向传播过程中，网络通过计算损失函数的梯度来更新权重和偏置项，以最小化预测误差。具体步骤如下：

1. **损失函数**：计算网络的输出与实际标签之间的损失。常见的损失函数包括均方误差（MSE）和交叉熵损失。

$$L = \frac{1}{2} \sum_i (y_i - \hat{y}_i)^2$$

其中， y_i 是实际标签， $\hat{y}_i$ 是网络输出。

2. **梯度计算**：通过链式法则计算每个权重和偏置项的梯度。

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial a_j} \cdot \frac{\partial a_j}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ij}}$$

4. **权重更新**：使用梯度下降算法更新权重和偏置项。

$$w_{ij} = w_{ij} - \eta \cdot \frac{\partial L}{\partial w_{ij}}$$

其中， η 是学习率。