南京航空航天大學
NANJING UNIVERSITY OF AERONAUTICS AND ASTRONAUTICS

第 2 章 深度学习基础

魏明强、宫丽娜
计算机科学与技术学院

目录

目录

神经网络基础

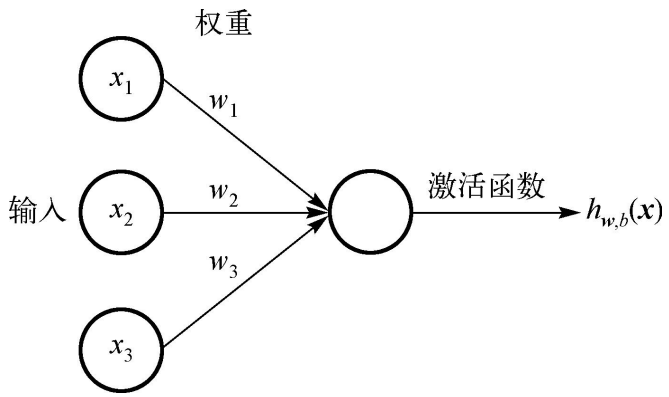
神经网络

□ 人工神经元

人工神经网络（Artificial Neural Network，ANN），简称为神经网络（Neural Network： NN），是指一系列受生物学和神经科学启发的数学模型。

人工神经元，简称为神经元，是构成神经网络的基本单元。

$$h_{w,b}(\boldsymbol{x}) = f(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b)$$

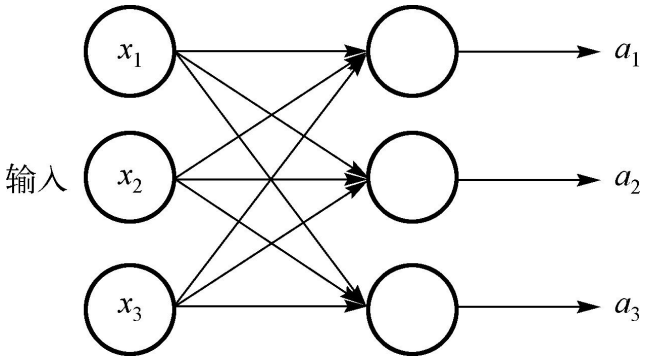


单个神经元计算过程

要想模拟人脑具有的能力，单一神经元是远远不够的，需要众多神经元的协作来完成复杂任务，即神经网络。

在得到单层神经网络的输出之后，可以通过叠加类似的层来构建每层都包含若干神经元的多层神经网络。

$$\boldsymbol{a} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$



单层神经网络计算过程

神经网络基础

神经网络

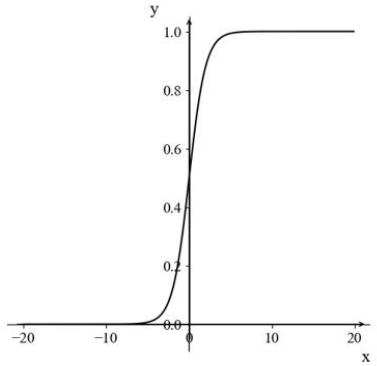
□ 激活函数

激活函数（Activation Function）是神经网络中的一种非线性变换，它赋予神经元更强大的表达能力。如果不使用激活函数，则每层的操作只是对上一层的输出结果进行线性变换，多层神经网络会退化成单层神经网络。

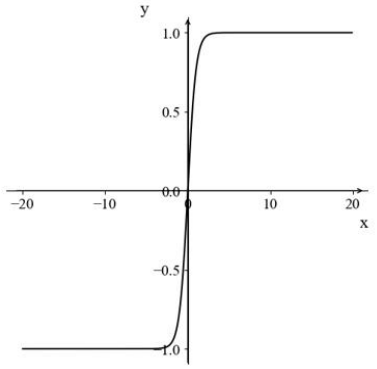
- Sigmoid函数

$$\sigma(x)=\frac{1}{1+e^{-x}}$$

通常用于二分类问题的输出层。



(a) Sigmoid函数



(b) Tanh函数

- Tanh函数

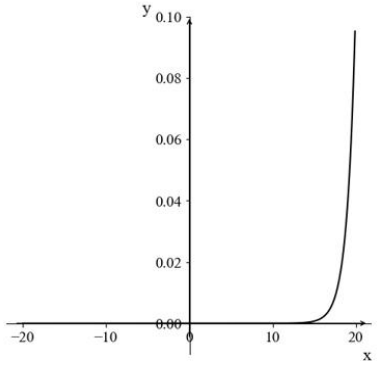
$$\tanh(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

通常用于中间层或输出层。

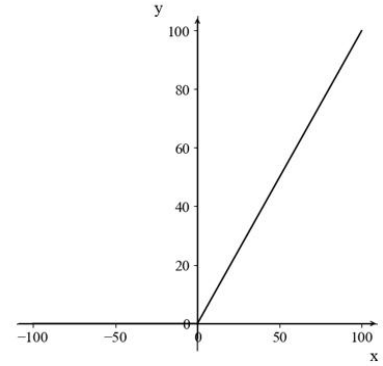
- Softmax函数

$$\mathrm{softmax}(\boldsymbol{x})_i=\frac{\exp(x_i)}{\Sigma_j\exp(x_j)}$$

通常用于多分类问题的输出层。



(c) Softmax函数



(d) ReLU函数

- ReLU函数

$$f_{\mathrm{ReLU}}(x)=\max(0,x)$$

广泛应用于隐藏层，其简单性和非饱和性使其在大多数情况下表现良好。

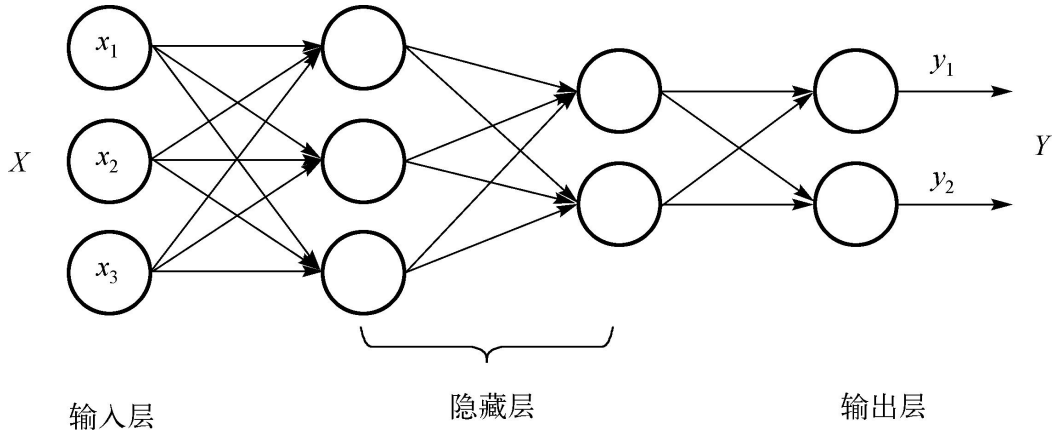
神经网络基础

南京航空航天大学 NANJING UNIVERSITY OF AERONAUTICS AND ASTRONAUTICS

神经网络

□ 全连接神经网络

在全连接神经网络中，每个神经元与前一层的所有神经元相连接，形成一个完全连接的结构。
它的基本组成包括输入层（Input Layer）、若干隐藏层（Hidden Layer）和输出层（Output Layer）。输入层接收原始数据或特征作为网络的输入，每个输入神经元对应于数据或特征的一个维度。隐藏层位于输入层和输出层之间，进行特征的非线性变换和抽象。每个隐藏层包含多个神经元，每个神经元与前一层的所有神经元相连接。多个隐藏层的存在使得网络能够学习更加复杂和抽象的表示。输出层产生网络的最终输出。



全连接神经网络在一些任务上表现良好，但随着问题复杂性的增加，更深层次、更复杂结构的神经网络逐渐取代了全连接神经网络。这是因为全连接神经网络在参数数量和计算复杂度上容易受到限制，而深度学习任务通常需要更强大的神经网络结构。

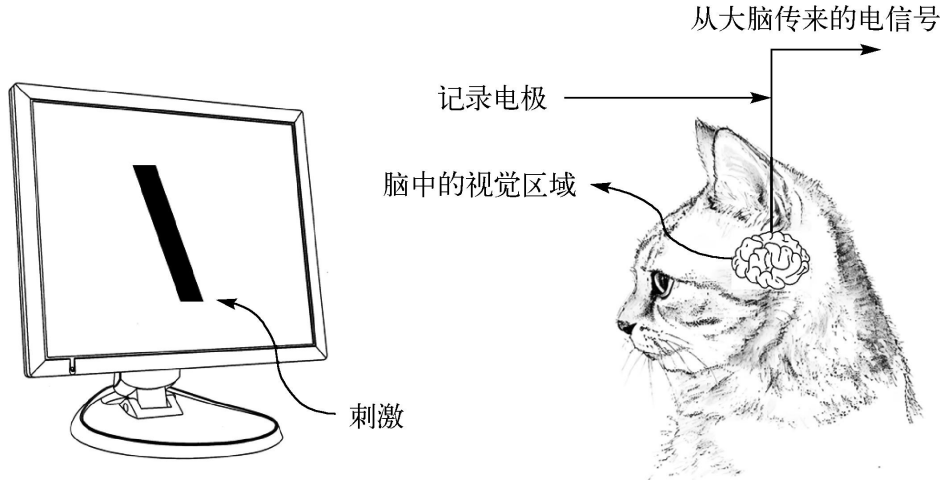
目录

神经网络基础

卷积神经网络

□ 感受野

1962 年，生物学家D.H. Hubel和 T.N. Wiesel对猫的视觉系统进行了研究，猫的视觉系统实验示意图如图2.5所示。他们首次发现了在猫的视觉皮层中存在两种主要类型的神经元，即简单细胞和复杂细胞。这两种类型的细胞对边缘和纹理的敏感性有所不同。神经元对视野中的某一小块区域内的特定边缘或纹理更为敏感，反映了感受野的特性。



感受野（Receptive Field）描述了神经系统中一些神经元对于特定刺激区域的敏感性，这意味着神经元只对其支配区域内的信号做出响应。在视觉神经系统中，视觉皮层中的神经细胞的输出受到视网膜上光感受器的影响，即当视网膜上的光感受器受到刺激并兴奋时，会产生神经冲动信号并传递到视觉皮层。然而，并非所有视觉皮层中的神经元都会接收这些信号。每个神经元都有其特定的感受野，即只有视网膜上特定区域内的刺激才能激活该神经元。

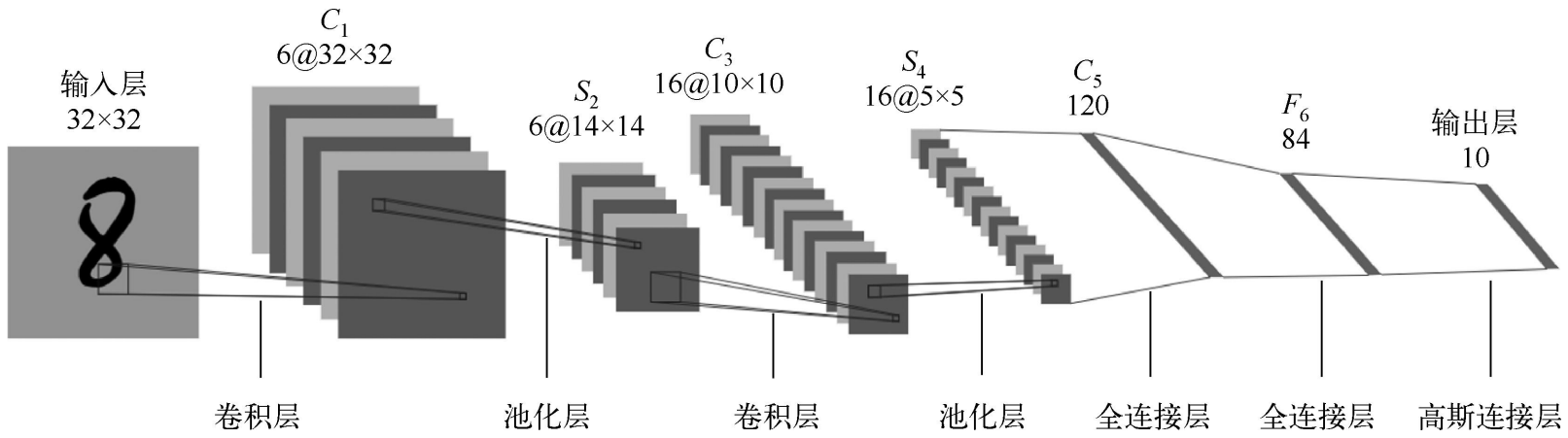
神经网络基础

卷积神经网络

□ 卷积神经网络

卷积神经网络（Convolutional Neural Network，CNN）的设计灵感正是源自生物学中感受野的机制。
卷积神经网络模仿了生物学中神经元对于刺激的局部敏感性。它通过学习局部特征，逐渐建立对整体特征的抽象。它在处理空间结构化数据和视觉数据方面的能力使其在自然语言处理、计算机视觉等领域都发挥着重要作用。

下图展示了第一个诞生的卷积神经网络LeNet-5的网络结构，该网络用于手写数字识别任务。LeNet-5由卷积层、池化层及全连接层组成，它的设计为后续卷积神经网络的发展奠定了基础。

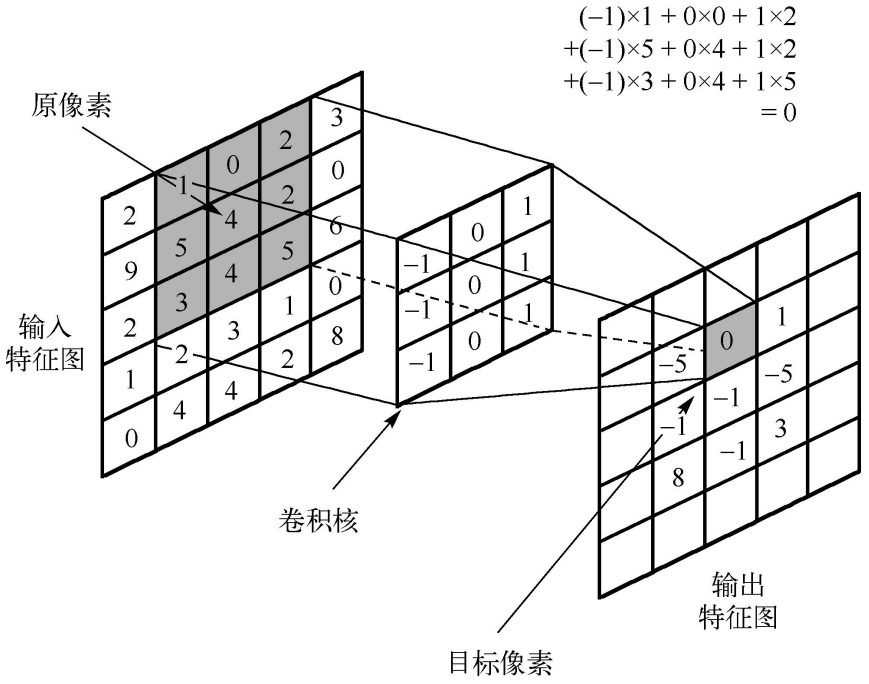


神经网络基础

卷积神经网络

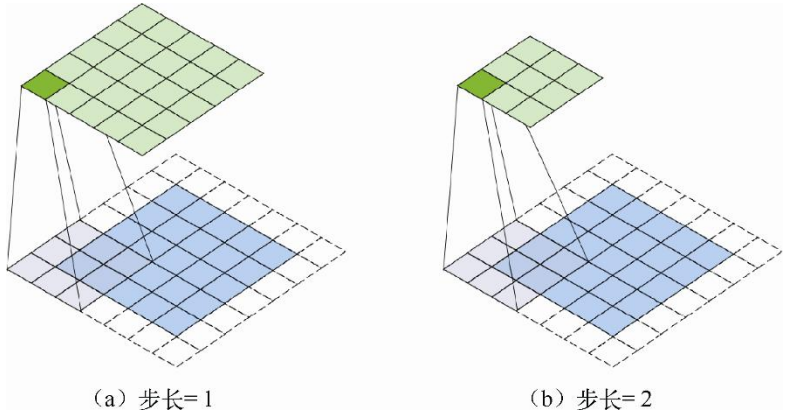
□ 卷积

卷积运算通过滑动一定间隔的卷积核（也称为滤波器）窗口，计算对应位置的元素相乘再求和，得到输出特征图中每个位置的值，当卷积核窗口移动到所示位置时，计算输入特征图与卷积核窗口对应位置的元素乘积，并将其求和，即执行计算：(-1)×1+0×0+1×2+(-1)×5+0×4+1×2+(-1)×3+0×4+1×5=0，从而计算得到输出特征图中相应位置的值为0。之后，卷积核继续向后滑动，重复相同的操作，直到得到完整的输出特征图。



□ 卷积操作的概念

- 偏置（bias）
- 步长（stride）
- 填充（padding）



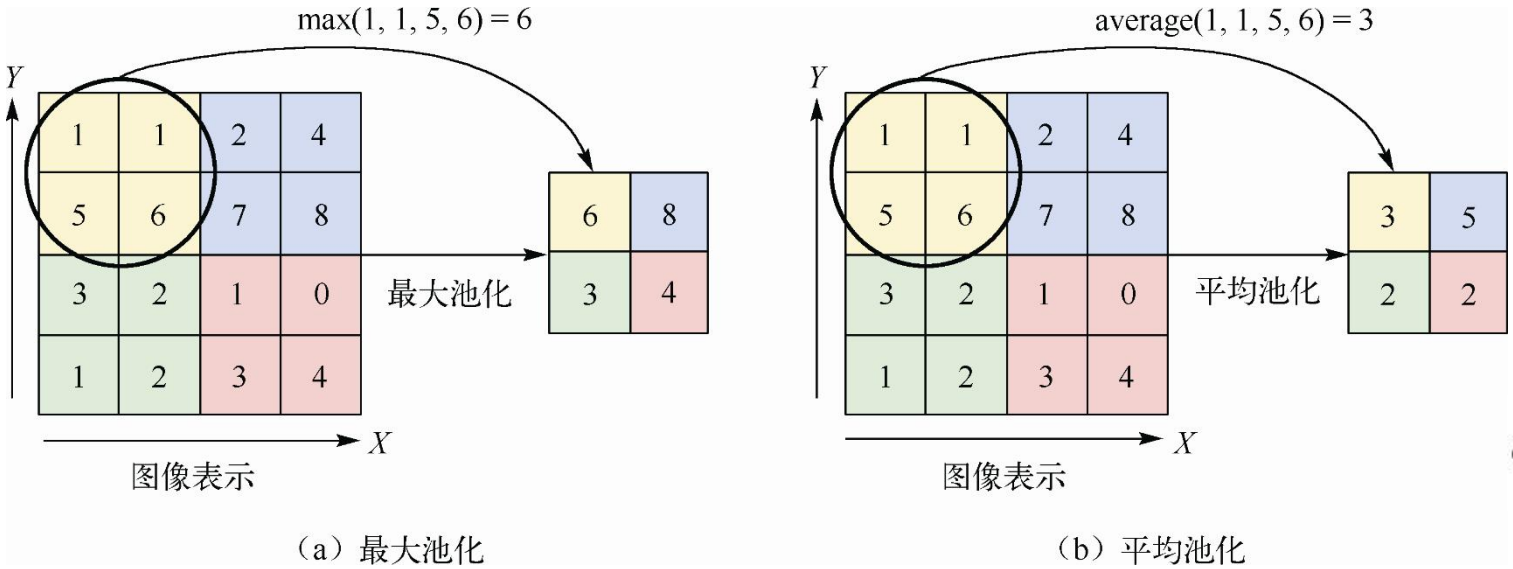
$$\text{output size} = \frac{\text{input size} - \text{kernel size} + 2 \times \text{padding}}{\text{stride}} + 1$$

神经网络基础

卷积神经网络

□ 池化

池化操作通常应用在卷积层之后，通过对特征图的局部区域进行采样，从而获得更小且具有抽象特征的特征图。常见的池化类型有最大池化和平均池化两种。在最大池化中，每个池化窗口选择局部区域的最大值作为采样值。而在平均池化中，每个池化窗口计算局部区域的平均值作为采样值。



（a）最大池化　　（b）平均池化

□ 池化层的特点

- 没有可学习参数
- 不改变通道数
- 平移不变性

$$\text{output size} = \frac{\text{input size} - \text{padding kernel size}}{\text{stride}} + 1$$

神经网络基础

卷积神经网络

□ 批归一化

批归一化的作用是加速神经网络的训练，提高模型的收敛速度，并且有助于避免梯度消失或梯度爆炸问题。批归一化的核心思想是对每层的输入进行归一化，使其均值接近 0，标准差接近1。这样做有助于缓解梯度消失问题，提高网络的稳定性。对于一个批次的输入数据，批归一化首先计算批次的均值和方差，再对输入进行归一化， 即减去均值并除以标准差，然后使用可学习的缩放和平移参数对归一化后的数据进行线性变换。

$$\mathrm{BN}(x)=\gamma\frac{x-\mu}{\sigma}+\beta$$

□ 全连接

全连接层（Fully Connected Layer），也被称为密集连接层，是卷积神经网络中的关键组成部分。在全连接层中，每个神经元都与上一层的所有神经元相连接，形成了一个全连接的结构。对于自然语言处理任务，输入通常是一维向量，如文本数据的词嵌入，以便进行文本分类、情感分析等任务；对于计算机视觉任务，输入通常是多维特征图，这些特征图可能通过卷积层或其他特征提取层从原始图像中提取而来。为了传递给全连接层，这些多维特征图通常需要被展平成一维向量，作为全连接层的输入，以便进行后续处理。

卷积神经网络

Dropout

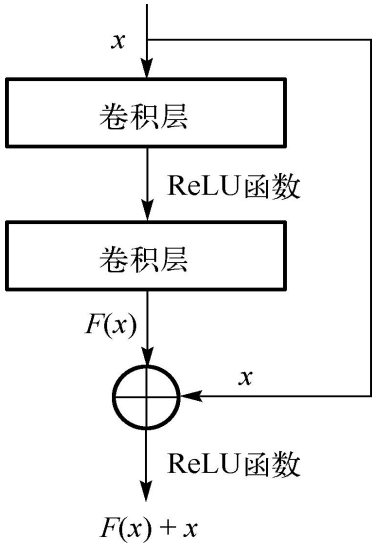
Dropout是一种常用的正则化技术，旨在减少过拟合并提高模型的泛化能力。Dropout的基本思想是在训练过程中以一定概率随机地忽略一部分神经元的输出。具体而言，假设有一个全连接层的输出向量为h，Dropout 的操作如下：

（1）在训练中，以概率（通常为 0.5）随机选择一部分神经元，将它们的输出置为0。

（2）在测试过程中，保持所有神经元的输出，但将它们乘以1 − p以保持期望输出值不变。

残差连接

残差连接将若干卷积层学习到的特征与原始输入相加，从而形成了一种“跳跃连接”的结构，从而使得神经网络更容易进行优化，并且能够构建更深层次的网络结构。残差连接能够在一定程度上缓解深层网络的退化网络问题。并且既不增加额外的参数也不增加计算复杂度，使得网络易于优化，提高了泛化性能。



目录

损失函数和优化算法

南京航空航天大学 NANJING UNIVERSITY OF AERONAUTICS AND ASTRONAUTICS

损失函数

□ 均方误差损失函数

均方误差（Mean Squared Error，MSE）损失函数是一种应用于回归问题的损失函数，用于度量模型预测值与真实值之间的平方差的平均值。

$$\mathrm{MSE}(y,\hat{y})=\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2$$

□ 平方绝对误差损失函数

平均绝对误差（Mean Absolute Error，MAE）损失函数是应用于回归问题的一种损失函数， 用于度量模型预测值与真实值之间的绝对差的平均值。

$$\mathrm{MAE}(y,\hat{y})=\frac{1}{n}\sum_{i=1}^{n}|y_i-\hat{y}_i|$$

□ 交叉熵损失函数

交叉熵损失（Cross-Entropy Loss）函数广泛应用于分类问题。它衡量模型输出的概率分布与真实标签的概率分布之间的差异。

二分类问题：

$$\text{Binary Cross-Entropy}(y,\hat{y})=-(y\log(\hat{y})+(1-y)\log(1-\hat{y}))$$

多分类问题：

$$\text{Categorical Cross-Entropy}(y,\hat{y})=-\sum_{i=1}^{C}y_i\log(\hat{y}_i)$$

损失函数和优化算法

南京航空航天大学 NANJING UNIVERSITY OF AERONAUTICS AND ASTRONAUTICS

损失函数

□ 序列交叉熵损失函数

序列交叉熵损失（Sequence Cross-Entropy Loss）函数是用于序列到序列（sequence- to-sequence）任务中的一种损失函数，主要应用于自然语言处理领域的机器翻译任务。在这种任务中，模型需要将一个输入序列映射到另一个输出序列，而且输入和输出的序列长度是可变的

$$\text{Sequence Cross-Entropy Loss} = -\frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{N} y_{t,i}\log(\hat{y}_{t,i})$$

□ 焦点损失函数

焦点损失（Focal Loss）函数通过调整难易分类样本的权重，即降低易分类样本的权重，提高难分类样本的权重，使得模型更关注难以分类的样本。

$$\text{Focal Loss} = -(1-\hat{y}_t)^{\gamma}\log(\hat{y}_t)$$

$$\hat{y}_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise} \end{cases}$$

目录

南京航空航天大學
NANJING UNIVERSITY OF AERONAUTICS AND ASTRONAUTICS

损失函数和优化算法

优化算法

☐ 梯度下降法

（1）目标函数设定：设定一个目标函数（损失函数）$J(\theta)$，其中θ表示模型的参数。目标是找到使得目标函数最小化的参数值。

（2）梯度计算：计算目标函数关于参数的梯度，即$\nabla J(\theta)$。梯度表示目标函数在参数空间中的变化率，它指示了函数增长最快的方向。

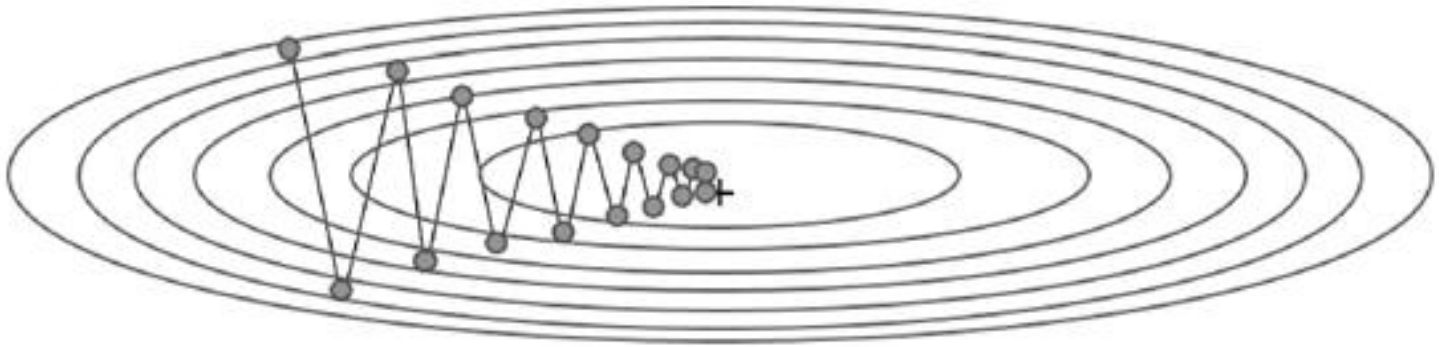
（3）参数更新：根据梯度的反方向调整参数，通过以下规则进行参数更新，即

$$\theta = \theta - \alpha \nabla J(\theta)$$

其中，α是学习率，表示每次更新时沿梯度方向移动的步长。学习率的选择对梯度下降的性能有很大的影响，学习率过大可能导致震荡或发散，学习率过小可能导致收敛缓慢。

☐ 梯度下降法变种

- 批量梯度下降法
- 随机梯度下降法
- 小批量梯度下降法



损失函数和优化算法

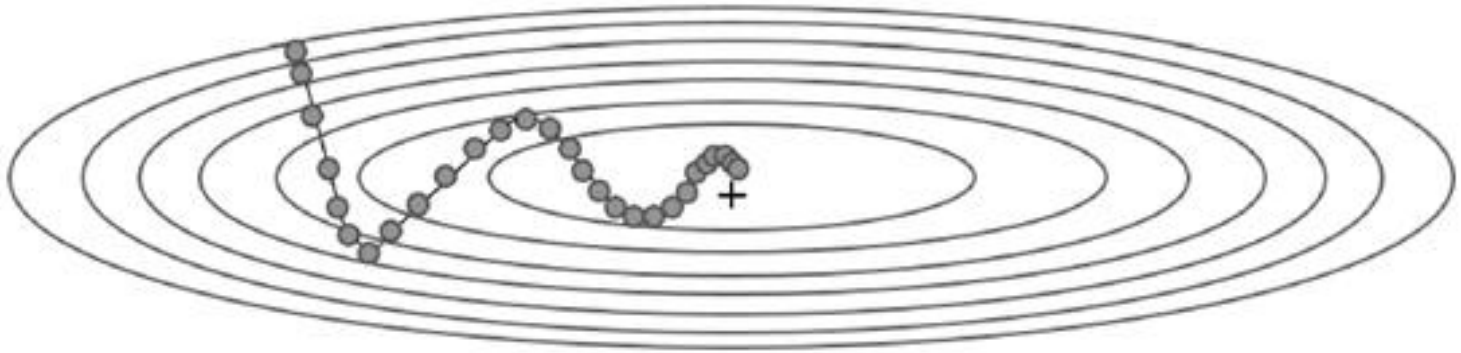
优化算法

□ 动量法

动量法（Momentum）旨在加速收敛并减小震荡。它引入了梯度的“动量”或“速度”概念，类似于物理学中的动量。动量法的核心思想是在更新参数时，不仅考虑当前的梯度，还考虑过去梯度的累积效果，以减小在参数空间的震荡。

$$v = \beta v + (1-\beta)\nabla J(\theta)$$
$$\theta = \theta - \alpha v$$

其中，v是动量（速度）；β是动量系数，控制过去梯度的权重，通常取值为(0, 1)，较大的β表示更多地考虑过去的梯度，从而减小震荡。通常β取值为0.9是一个常见的起点。$\nabla J(\theta)$是目标函数关于参数的梯度。



损失函数和优化算法

优化算法

□ AdaGrad算法

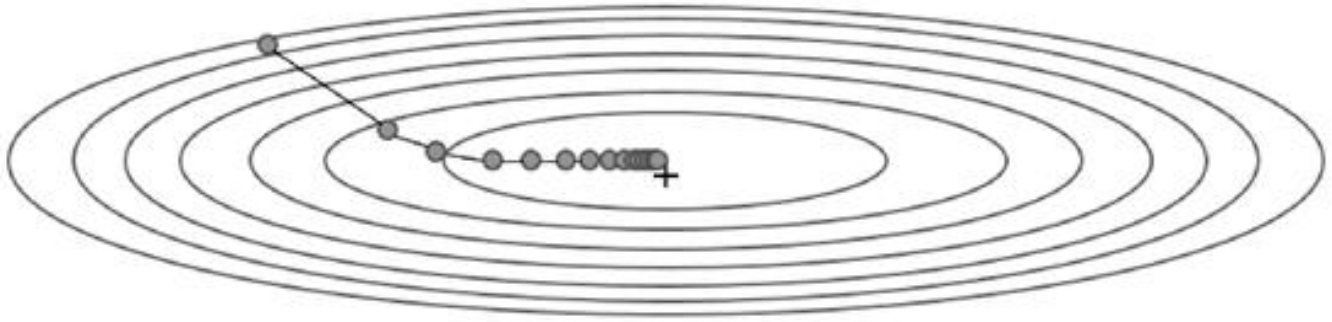
AdaGrad 算法（Adaptive Gradient Algorithm）是一种自适应学习率的优化算法，其主要目标是处理稀疏数据和非平稳目标函数。AdaGrad算法的主要特点在于对每个参数使用不同的学习率，使其在训练过程中能够适应不同参数的更新速度。

$$g = \nabla J(\theta)$$

$$s = s + g^2$$

$$\theta = \theta - \frac{\alpha}{\sqrt{s+\varepsilon}} \cdot g$$

其中， g是当前迭代的梯度； s是一个累积的平方梯度的历史和；$\sqrt{s+\varepsilon}$是梯度的平方根； ε是一个很小的数，用于避免分母为零。AdaGrad算法根据每个参数的历史梯度平方和自适应地调整学习率。



损失函数和优化算法

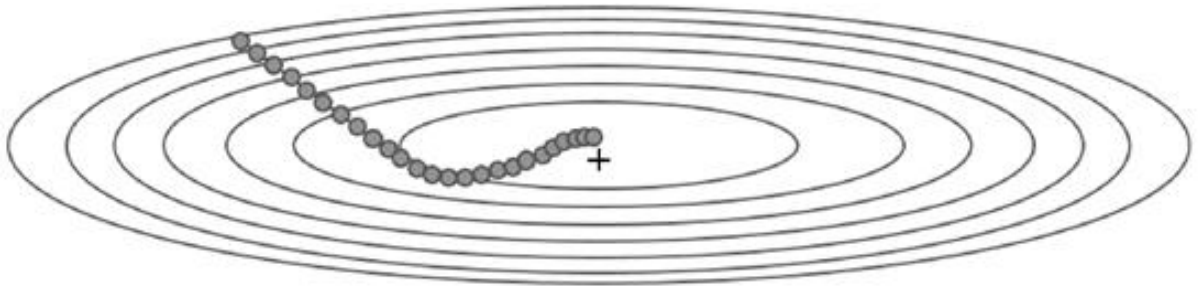
优化算法

□ Adam算法

Adam算法是一种自适应学习率的优化算法，结合了动量法和AdaGrad算法思想，在深度学习中得到了广泛应用，对于不同类型的神经网络和任务都有较好的适应性。其核心思想是为每个参数维护两个移动平均量，一个是梯度的一阶矩估计（动量项），另一个是梯度的二阶矩估计（AdaGrad项），然后使用这两个估计来调整学习率。

$$m = \beta_1 m + (1-\beta_1) g$$
$$v = \beta_2 v + (1-\beta_2) g^2$$
$$\hat{m} = \frac{m}{1-\beta_1^t}$$
$$\hat{v} = \frac{v}{1-\beta_2^t}$$
$$\theta = \theta - \alpha \frac{\hat{m}}{\sqrt{\hat{v}} + \varepsilon}$$

其中， g是当前迭代的梯度； m是一阶矩估计，表示梯度的移动平均； v是二阶矩估计，表示梯度平方的移动平均； t表示当前迭代次数；β_1和β_2是衰减系数，通常取值分别为0.9和0.999；$\hat{m}$和$\hat{v}$分别是对m和v进行修正后的估计，避免在训练初期m和v过小的问题。Adam算法使用梯度的一阶矩估计和二阶矩估计来自适应地调整学习率，使得在训练过程中不同参数可以有不同的学习率。



目录

神经网络训练

南京航空航天大学 NANJING UNIVERSITY OF AERONAUTICS AND ASTRONAUTICS

梯度和链式法则

□ 梯度的计算

给定一个具有n个输入 和1个标量输出的函数：$F(x)=F(x_1,x_2,\cdots,x_n)$

其对输入计算梯度，得到一个与输入具有相同维度的向量，向量的每个维度是输出对于输入中相应维度的偏导数：

$$\frac{\partial F}{\partial x}=\left[\frac{\partial F}{\partial x_1},\frac{\partial F}{\partial x_2},\ldots,\frac{\partial F}{\partial x_n}\right]$$

给定一个有n个输入和m个输出的函数：

$$F(x)=[F_1(x_1,x_2\cdots,x_n),F_2(x_1,x_2\cdots,x_n),\cdots,F_m(x_1,x_2\cdots,x_n)]$$

可以将m个输出拆分成m个具有n个输入的单输出函数。相当于由m个神经元构成了一层神经网络。m个输出分别对n个输入求微分，得到m×n大小的雅可比矩阵（Jacobian Matrix）。该矩阵的第i行第j列元素是第i个输出对于第j个输入的偏导数。

神经网络训练

南京航空航天大学 NANJING UNIVERSITY OF AERONAUTICS AND ASTRONAUTICS

梯度和链式法则

□ 链式法则

链式法则是复合函数求导数的性质，其定义如下：如果某个函数由复合函数表示，则该复合函数的导数可以用构成复合函数的各个函数的导数的乘积表示。

以一元函数为例，为了求z对x的导数，使用链式法则，先求z对y的导数，再求y对x的导数，再将两个导数相乘，即为z对x的导数：

$$z = 3y$$
$$y = x^2$$
$$\frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}z}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}x} = 3 \times 2x = 6x$$

推广到多输入多输出的函数：

$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \cdots$$

要求h对x的偏导，同样地运用链式法则，先求h对z的偏导以及z对x的偏导，两者都可以表示成雅可比矩阵，再将矩阵相乘，得到最终的结果。

目录

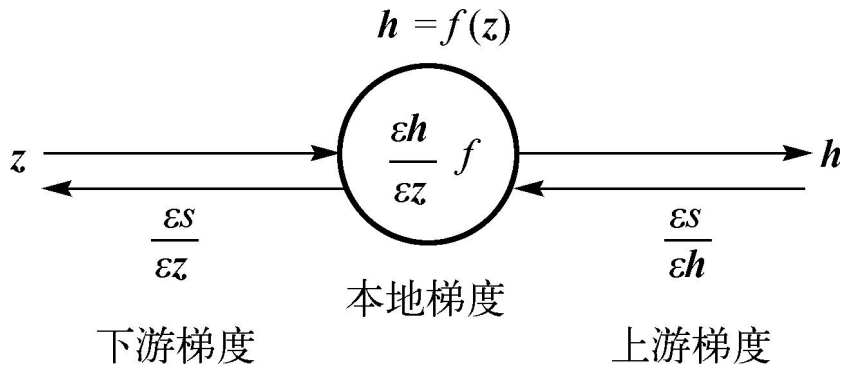
神经网络训练

前向传播和反向传播

□ 计算图

计算图能够将神经网络的计算过程以图形化的方式呈现。在这个图中，源节点表示网络的输入，内部节点表示各种计算操作，有向边用于传递各节点计算出的值，同时存储当前计算操作得到的值。按照有向边的方向进行顺序计算，就能得到神经网络的输出值，这个过程称为**前向传播**。

反向传播的过程则是沿着计算图相反的方向进行计算，计算每个参数的梯度，从而在优化过程中更新这些参数。通过反向传播，神经网络能够学习调整权重和偏置，使得模型的预测与实际结果更加接近，从而提高整体性能。



单个节点的反向传播：下游梯度 = 上游梯度×本地梯度

神经网络训练

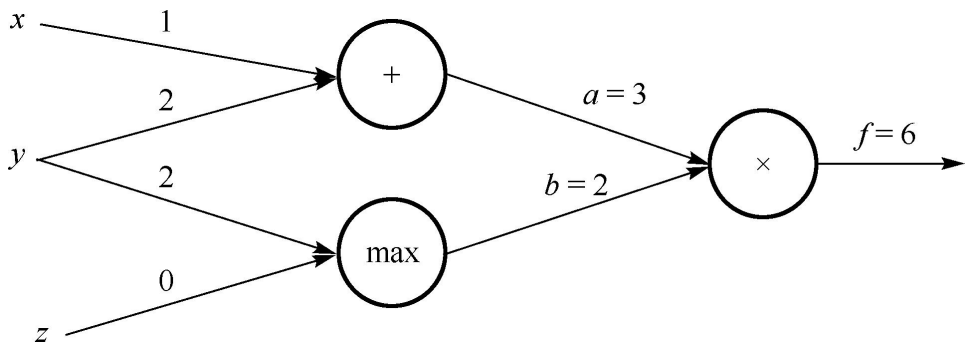
前向传播和反向传播

- **计算图计算过程示例**

$$f(x,y,z)=(x+y)\max(y,z)$$
$$x=1,\ \ y=2,\ \ z=0$$

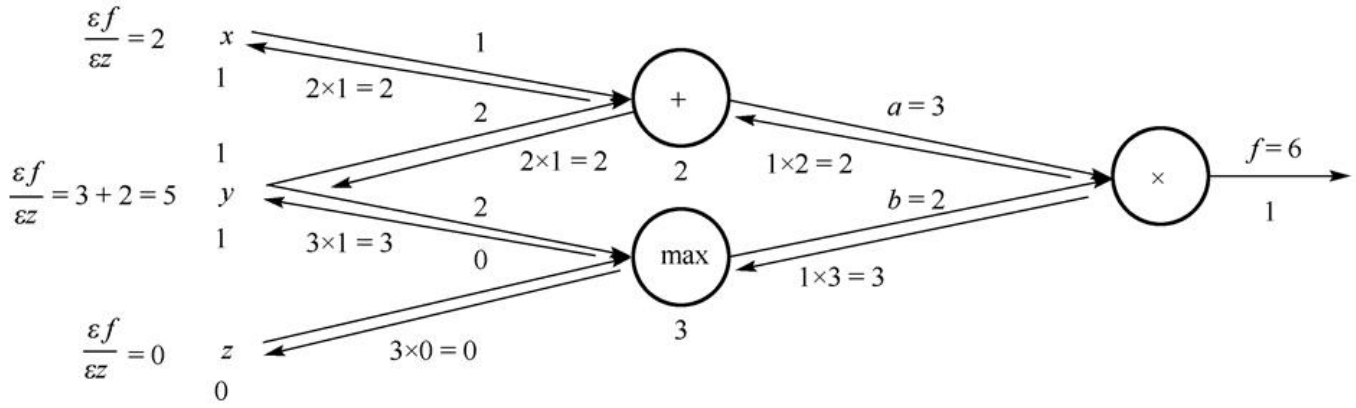
前向传播

$$a=x+y=3$$
$$b=\max(y,z)=2$$
$$f=ab=6$$



反向传播

$$\frac{\partial a}{\partial x}=1,\ \ \frac{\partial a}{\partial y}=1$$
$$\frac{\partial b}{\partial y}=1(y>z)=1,\ \ \frac{\partial b}{\partial z}=1(z>y)=0$$
$$\frac{\partial f}{\partial a}=b=2,\ \ \frac{\partial f}{\partial b}=a=3$$



目录

神经网络训练

训练神经网络示例

PyTorch训练神经网络示例

训练神经网络需要先将训练数据输入模型中，通过前向传播计算预测值，然后计算损失函数，并通过反向传播调整模型参数，以最小化损失。这一过程使用合适的优化算法来更新模型的权重和偏置。

以卷积神经网络为例，使用MNIST数据集完成手写数字识别任务，使用PyTorch 框架来演示训练神经网络的具体流程。

（1）导入必要的库

```
# 导入PyTorch深度学习框架
import torch
# 导入PyTorch中构建神经网络所需的模块和类
import torch.nn as nn
# 导入PyTorch中用于优化算法的模块
import torch.optim as optim
# 导入PyTorch计算机视觉扩展库，用于处理图像数据集
from torchvision import datasets, transforms
```

神经网络训练

南京航空航天大学 NANJING UNIVERSITY OF AERONAUTICS AND ASTRONAUTICS

训练神经网络示例

PyTorch训练神经网络示例

（2）定义一个简单的卷积神经网络模型，包括卷积层、激活函数、池化层和全连接层。

```
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        # 第一个卷积层，输入通道为1（灰度图像），输出通道为32，卷积核大小为3x3
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        # 第二个卷积层，输入通道为32，输出通道为64，卷积核大小为3x3
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        # 全连接层，输入特征为64*7*7，输出特征为128
        self.fc1 = nn.Linear(64 * 7 * 7, 128)
        # 输出层，输出特征为10（对应10个类别）
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        # 第一层卷积，激活函数，池化
        x = self.conv1(x)
        x = self.relu(x)
        x = self.pool(x)
        # 第二层卷积，激活函数，池化
        x = self.conv2(x)
        x = self.relu(x)
        x = self.pool(x)
        # 将特征图展平成一维向量
        x = x.view(-1, 64 * 7 * 7)
        # 全连接层，激活函数
        x = self.fc1(x)
        x = self.relu(x)
        # 输出层
        x = self.fc2(x)
        return x
```

神经网络训练

训练神经网络示例

PyTorch训练神经网络示例

（3）加载数据集并进行数据预处理，将图像转换为Tensor格式并进行归一化。

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

train_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
test_dataset = datasets.MNIST(root='./data', train=False, download=True, transform=transform)

train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=64, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=64, shuffle=False)
```

（4）定义损失函数和优化器，损失函数使用交叉熵损失函数，优化器使用 Adam优化器，学习率设置为0.001。

```
model = SimpleCNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

神经网络训练

南京航空航天大学 NANJING UNIVERSITY OF AERONAUTICS AND ASTRONAUTICS

训练神经网络示例

PyTorch训练神经网络示例

（5）进行模型训练，迭代数据集，计算损失，反向传播更新模型参数。

```
# 设置训练过程迭代100轮次
num_epochs = 100

for epoch in range(num_epochs):
    # 将模型切换到训练模式，启用一些特殊的层（例如Dropout）
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        # 梯度清零，避免梯度的累积对后续迭代的计算产生影响
        optimizer.zero_grad()
        # 前向传播，得到模型的输出
        output = model(data)
        # 计算损失，度量模型输出与真实标签之间的差异
        loss = criterion(output, target)
        # 反向传播，计算损失对模型参数的梯度
        loss.backward()
        # 更新模型参数，以最小化损失
        optimizer.step()

        # 打印训练信息，便于实时监控训练过程
        if batch_idx % 100 == 0:
            print(f'Epoch {epoch + 1}/{num_epochs}, Batch {batch_idx}/{len(train_loader)}, Loss: {loss.item()}')
```

目录

深度学习框架

南京航空航天大學 NANJING UNIVERSITY OF AERONAUTICS AND ASTRONAUTICS

主流深度学习框架

Jittor 计图

https://cg.cs.tsinghua.edu.cn/jittor/

Keras

https://keras.io/

Caffe2

http://caffe.berkeleyvision.org/

TensorFlow

https://www.tensorflow.org/

mxnet

https://mxnet.apache.org/

[M]S MindSpore

https://www.mindspore.cn/

PyTorch

https://pytorch.org/

PaddlePaddle 飞桨

https://www.paddlepaddle.org.cn/

目录

深度学习框架

南京航空航天大学 NANJING UNIVERSITY OF AERONAUTICS AND ASTRONAUTICS

框架选择优缺点比较

框架	优点	缺点	适用场景
Jittor	动态图计算；自动微分；异步计算	相对较新；文档和生态系统可能有限	灵活模型需求；动态图计算场景
Tensorflow	广泛应用；高性能；丰富生态系统	相对复杂；开发迭代速度相对较慢	大规模部署；复杂模型需求
PyTorch	动态图模型；易用性；研究支持	部署相对复杂；稳定性较差	研究领域；快速试验与原型开发
Keras	简单易用；轻量级	灵活性相对较低	初学者；快速搭建简单模型
Caffe	高效；简单明了	功能有限；缺乏动态图支持	嵌入式设备；实时应用
MXNet	多语言支持；可扩展性	文档相对不足；相对小众	多语言项目；可扩展性需求
PaddlePaddle	面向产业；动静结合	生态系统相对较小；学习难度较大	工业应用；动静结合需求
MindSpore	全场景支持；动静结合	生态相对较新；资源相对有限	多场景支持；新兴项目

目录

思考

- **损失函数的选择**
  - 多任务问题→联合损失函数/各损失函数独立优化
  - 类别不平衡问题→加权损失函数
  - 特定问题→结合业务领域知识自定义损失函数
- **优化算法的选择**
  - 自适应学习率算法
  - 学习率衰减策略
  - 正则化控制项
- **模型架构的选择**
  - 利用预训练模型的优势
  - 引入注意力机制
  - 增加网络的深度

谢谢！
Thanks!