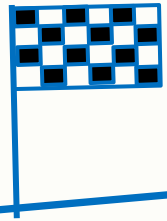
第01讲 深度学习概述

传媒与信息工程学院
欧 新 宇

第01讲 深度学习概述

- 人工智能的基本概念
- 人工智能、机器学习与深度学习之间的关系
- 机器学习（深度学习）发展的时间线
- 深度学习日益流行的关键因素及其未来潜力
- 深度学习的基本原理：基于梯度的优化
- 常用工具软件和开发环境



Part 04

初识神经网络

/ 初识神经网络
/ 什么是导数
/ 张量运算的导数：梯度
/ 随机梯度下降
/ 链式求导：反向传播算法
/ 深度学习的基本原理：基于梯度的优化

4.1 初识神经网络

手写字体识别

MNIST数据集是机器学习领域的一个经典数据集，它包含70000个样本，其中训练集60000个，测试集10000个。每个样本都分为图片和标签，图片是28*28的像素矩阵，标签是0~9的10个数字。

4.1 初识神经网络

手写字体识别的基本代码结构

训练代码基本流程：

Step0：导入全局依赖及全局参数配置

Step1：数据准备：定义数据预处理、数据载入、数据显示和测试

Step2：定义和实例化网络类

Step3：定义过程可视化函数 *（本项目暂不设置）*

Step4：配置训练基本参数：优化器、损失函数和评价指标

Step5：模型训练和在线测试

Step6：离线评估

4.1 初识神经网络

手写字体识别

Step0：导入全局依赖及全局参数配置

```
import numpy as np
import matplotlib.pyplot as plt
import paddle
```

Step1：数据准备：定义数据预处理、数据载入、数据显示和测试

[2]:
```
import paddle.vision.transforms as T
transform = T.Compose([T.Normalize(mean=[127.5], std=[127.5], data_format='CHW')])
```

[3]:
```
# 使用transform对数据集做归一化
train_dataset = paddle.vision.datasets.MNIST(mode='train', transform=transform)
test_dataset = paddle.vision.datasets.MNIST(mode='test', transform=transform)
print('数据载入完毕!')
```

4.1 初识神经网络

手写字体识别

Step2：定义和实例化网络类

[5]:
```
# 1. 初始化模型
LeNet = paddle.vision.models.LeNet(num_classes=10)
```

[6]:
```
# 1. 可视化模型的结构和参数
paddle.summary(LeNet, (1,1,28,28))
```

Layer (type)	Input Shape	Output Shape	Param #
Conv2D-1	[[1, 1, 28, 28]]	[1, 6, 28, 28]	60
ReLU-1	[[1, 6, 28, 28]]	[1, 6, 28, 28]	0
MaxPool2D-1	[[1, 6, 28, 28]]	[1, 6, 14, 14]	0
Conv2D-2	[[1, 6, 14, 14]]	[1, 16, 10, 10]	2,416
ReLU-2	[[1, 16, 10, 10]]	[1, 16, 10, 10]	0
MaxPool2D-2	[[1, 16, 10, 10]]	[1, 16, 5, 5]	0
Linear-1	[[1, 400]]	[1, 120]	48,120
Linear-2	[[1, 120]]	[1, 84]	10,164
Linear-3	[[1, 84]]	[1, 10]	850

```
Total params: 61,610
Trainable params: 61,610
Non-trainable params: 0
---------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.11
Params size (MB): 0.24
Estimated Total Size (MB): 0.35
---------------------------------------
```

4.1 初识神经网络

手写字体识别

Step4：配置训练基本参数：优化器、损失函数和评价指标

```
# 1. 实例化模型
model = paddle.Model(LeNet)

# 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
    )
```

Step5：模型训练和在线测试

```
model.fit(train_dataset,                # 训练集
          # test_dataset,               # 测试集，设置后则在每个周期后进行验证，若不指定则不在验证集上进行验证
          epochs=5,                     # 训练周期数
          batch_size=64,                # 训练的批次大小
          verbose=1)                    # 是否显示训练日志
```

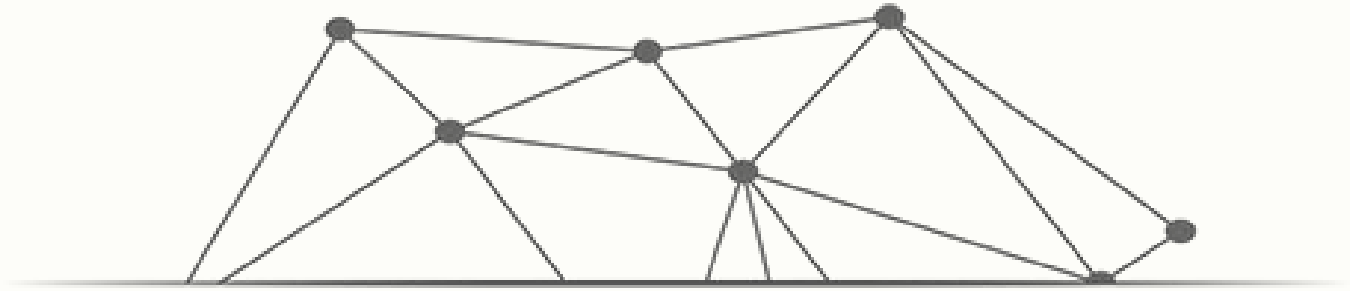
4.1 初识神经网络

手写字体识别

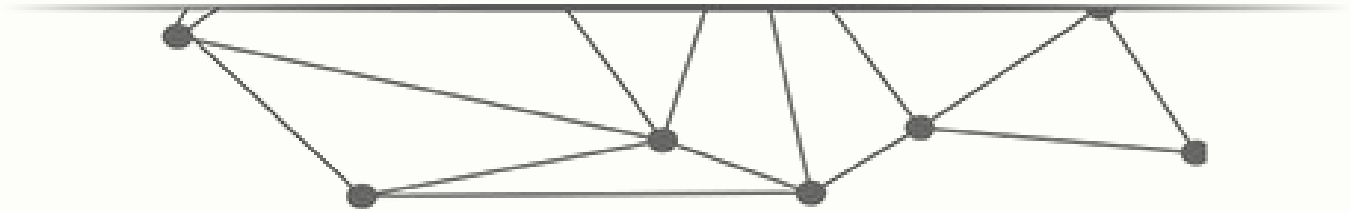
Step6：离线评估

```
# 1. 载入模型
model.load('D:\\WorkSpace\\DeepLearning\\Website\\Data\\Projects\\Project004LeNetMNIST\\Project004LeNetMNIST')

# 2. 运行评估函数在测试集上进行性能评估
model.evaluate(test_dataset, batch_size=64, verbose=1)
```



课堂互动 Link

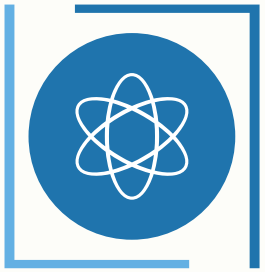


4.1 初识神经网络

神经网络的几个关键点

神经元及网络结构

神经网络的核心

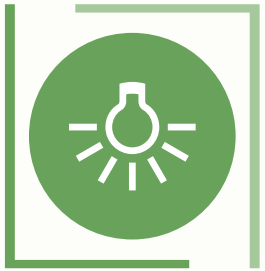


数据

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

4.2 神经网络的核心

神经网络的核心元素 —— 神经元

在上述的神经网络(多层感知机)中，我们使用全连接层来构建网络，即:

hidden=fluid.layers.fc(input=input, size=512, act='relu')

而所谓全连接，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。

4.2 神经网络的核心

神经网络的核心结构 —— 基于神经元的网络

```
# 定义多层感知器
def multilayer_perceptron(input):
    # 第一个全连接层，激活函数为ReLU
    hidden1 = fluid.layers.fc(input=input, size=512, act='relu')
    # 第二个全连接层，激活函数为ReLU
#     hidden2 = fluid.layers.fc(input=hidden1, size=100, act='relu')
    # 以softmax为激活函数的全连接输出层，大小为10
    prediction = fluid.layers.fc(input=hidden1, size=10, act='softmax')
    return prediction
```

以上是构建网络的代码，此处包含两个全连接层(fc_layers)，维度分别是512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。

4.3 神经网络的输入

输入数据

首先，我们应该看看神经网络的输入到底是什么，下面我们以mnist手写数字识别为例。

```
def load_image(file):
    im = Image.open(file).convert('L')  # 将RGB转化为灰度图像，L代表灰度图像，像素值在0~255之间
    # resize image with high-quality 图像大小为28*28
    im = im.resize((28, 28), Image.ANTIALIAS)
    im = np.array(im).reshape(1, 1, 28, 28).astype(np.float32)  # 返回新形状的数组,把它变成一个 numpy 数组以匹配数据馈送格式。
    # print(im)
    im = im / 255.0 * 2.0 - 1.0  # 归一化到【-1~1】之间
    return im
```

输入图像被保存在形态为4D(n,1,28,28)的float32的Numpy数组（张量）中，其形状为28*28。同时，所有的数据都被归一化到（-1,1)之间。

此外对于整个数据来说，它分为训练集(60000,1,28,28), 测试集(10000,1,28,28)，在输入网络的时候每个样本都会被reshape成（1，28*28）=（1，768）的形态。

4.4 神经网络的起点 -- 权重初始化

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。而在训练开始时，这些权重需要被初始化。**随机初始化**是一种比较好的方法，它使用随机算法将权重矩阵初始化为一些很小的随机值。此时的值不具任何意义，但这是训练的起点。此后，这些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，这个过程就称为**训练**。

当然，还有一些更好的初始化方法，例如：MSRA(He)、**Xavier** 和使用预训练模型（迁移学习）。

4.5 神经网络的产生 -- 迭代训练

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。**

4.5 神经网络的产生 —— 迭代训练

迭代训练

```
BUF_SIZE = 512
BATCH_SIZE = 128
# 用于训练/测试的数据提供器，每次从缓存中随机读取批次大小的数据
train_reader = paddle.batch(paddle.reader.shuffle(paddle.dataset.mnist.train(), buf_size=BUF_SIZE), batch_size=BATCH_SIZE)
test_reader = paddle.batch(paddle.reader.shuffle(paddle.dataset.mnist.test(), buf_size=BUF_SIZE), batch_size=BATCH_SIZE)
```

```
EPOCH_NUM = 5
for pass_id in range(EPOCH_NUM):
    # 进行训练
    for batch_id, data in enumerate(train_reader()):  # 遍历train_reader
        train_cost, train_acc = exe.run(program=fluid.default_main_program(),  # 运行主程序
                                        feed=feeder.feed(data),  # 给模型喂入数据
                                        fetch_list=[avg_cost, acc])  # fetch 误差、准确率
```

现在，我们也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=128个样本，共迭代了epoch_num=5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。5个轮次后，网络进行了2345次梯度更新，每个轮次60000/128=469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写字体进行分类。

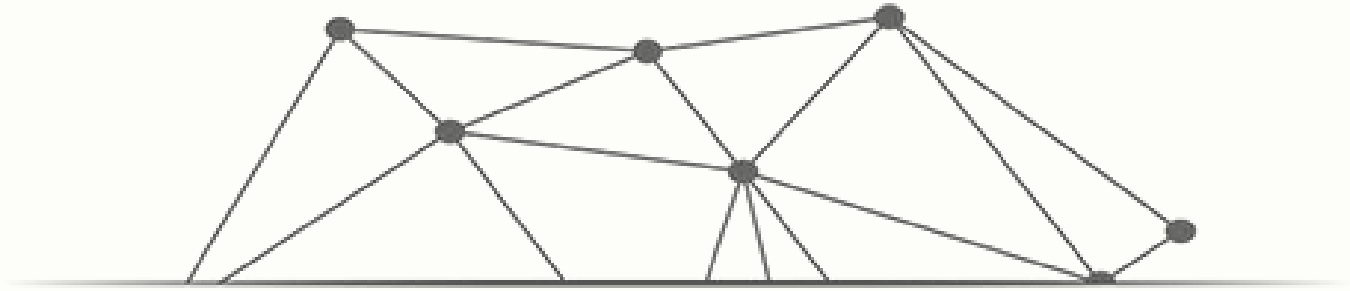
4.6 神经网络的优化--损失函数和优化方法

cross_entropy交叉熵是**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layers.mean()来求所有特征上的平均值来作为最终的loss。

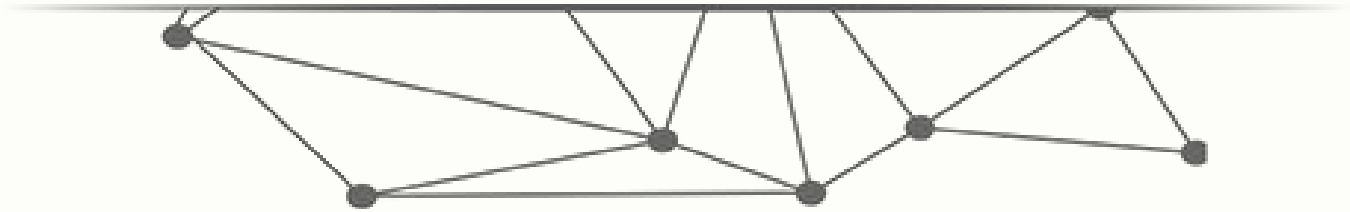
```
# 获取分类器
predict = multilayer_perceptron(image)
# 获取损失函数和准确率函数
cost = fluid.layers.cross_entropy(input=predict, label=label)  # 使用交叉熵损失函数,描述真实样本标签和预测概率之间的差值
avg_cost = fluid.layers.mean(cost)
acc = fluid.layers.accuracy(input=predict, label=label)
```

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**(fluid.optimizer)来决定。并且梯度下降的步长step用**学习率**learning_rate来进行控制。

```
# 定义优化方法
optimizer = fluid.optimizer.MomentumOptimizer(learning_rate=0.001, momentum=0.9)
opts = optimizer.minimize(avg_cost)
```



课堂互动 Link



4. 初识神经网络

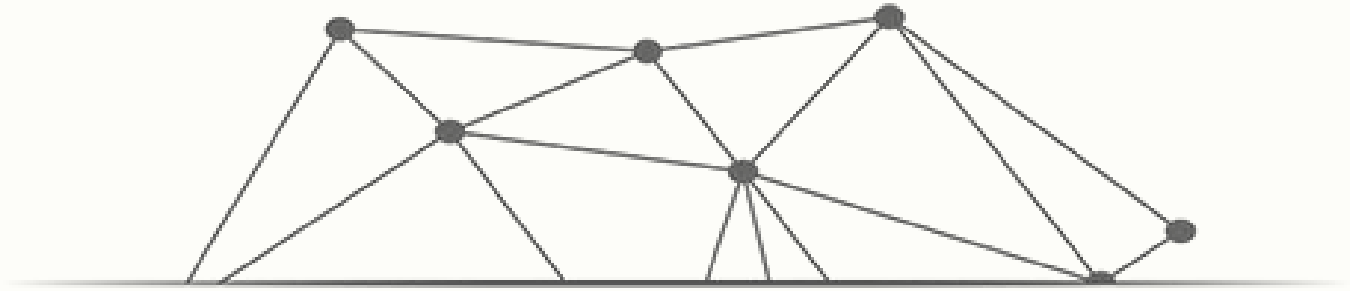
到目前为止

我们已经了解神经网络的大部分知识

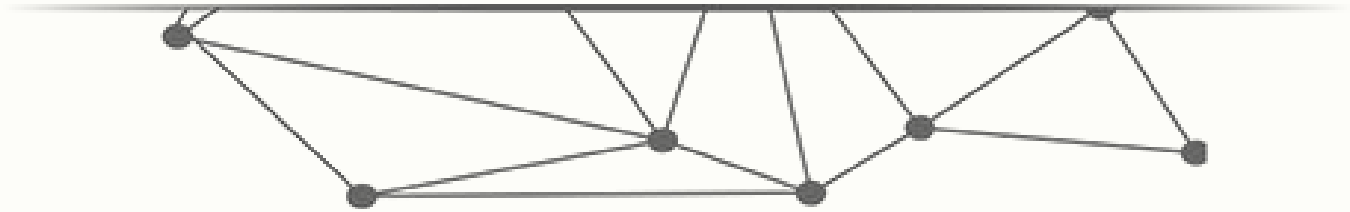
4. 初识神经网络

本节小结

- **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批量，并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（距离由学习率指定）
- 整个学习过程之所以能够实现，是因为神经网络是一系列可微分的张量运算，因此可以利用求导的链式法则来得到梯度函数，这个函数将当前参数和当前数据批量映射为一个梯度值
- **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，例如RMSProp、带动量的随机梯度下降（SGD）等



课堂互动 Link



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