今天声学了吗

音频信号处理
及基于音频的深度学习教程

Audio Signal Processing
Audio-based Deep Learning Tutorials

b站：今天声学了吗

公众号：今天声学了吗

邮箱：1319560779@qq.com

神经网络训练

概念：为了得到理想的输出值，需要反复训练模型，使得模型更加通用。

操作：1. 训练的依据——损失函数~ 2. 训练的过程——梯度下降~ 3. 训练的结果——权重更新~

- **什么是损失函数，为什么作为依据?**

衡量输出值与预测值之间的近似程度，不同的损失函数会导致不同的分类，所以选择一个合适的损失函数很重要。

- **训练的过程是什么样的，为什么是梯度下降?**

- 训练的过程是反向传播的过程，利用得到的输出值与预期值之间的偏差（损失函数），更新模型中的参数w和b，使得偏差变得最小，也就是损失函数下降到最小值的过程。
- 梯度：一个曲线/曲面的梯度方向就是指向上升最快的方向（导数值）。
- 因为给定区域的函数的极小值也就是函数最小值，极小值的导函数为0，那么寻找极小值就是沿着梯度下降的方向，所以利用反向链式求导-链式求梯度，最终得到w和b

神经网络训练

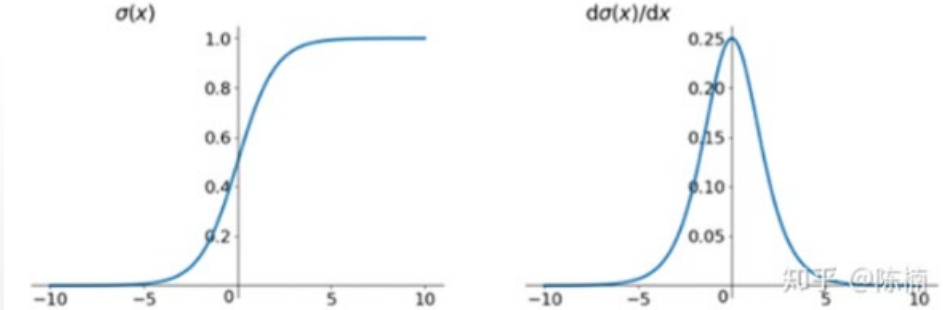
- **反向传播**

举例：对于一个三层，每层只有一个神经元，激活函数是sigmoid()的网络模型

$$z_1 = w_1 \cdot x + b_1, a_1 = \sigma(z_1); z_2 = w_2 \cdot a_1 + b_2, a_2 = \sigma(z_2);$$

$$z_3 = w_3 \cdot a_2 + b_3, a_3 = \sigma(z_3); z_4 = w_4 \cdot a_3 + b_4, a_3 = \sigma(z_4);$$

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



对于第一层偏置值链式求导可以得到的网络：

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) w_2 \sigma'(z_2) w_3 \sigma'(z_3) w_4 \sigma'(z_4) \frac{\partial C}{\partial a_4}$$

神经网络训练

- **反向传播**

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1)w_2\sigma'(z_2)w_3\sigma'(z_3)w_4\sigma'(z_4)\frac{\partial C}{\partial a_4}$$

问题：可能会产生梯度消失。

因为梯度图大于 5或小于 −5部分的梯度接近 0，这会导致在反向传播过程中处于该区域的导数为0，误差将很难甚至根本无法传递至前层，进而导致整个网络无法训练。

在于 $\sigma'(z)$ 同样依赖于w： $\sigma'(z) = \sigma'(w \cdot a + b)$,其中a是输入的激活函数。所以在让w变大时，需要同时不让 $\sigma'(w \cdot a + b)$ 变小，这将是很大限制了，因为w变大，会使得 $w \cdot a + b$ 变大，看 σ' 的图可知，这会让我们走到 σ' 的两翼，会得到很小的值。所以一般情况下会遇到消失的梯度。

并且权重更新过程不是一个收敛的过程，Sigmoid函数的输出值恒大于0，这会导致模型训练的收敛速度变慢。

逼近的思路理解训练

用高阶函数构造sine函数：随机生成一个三阶函数，赋予一组随机参数，得到的输出与sine输出值比较，差值loss最小的那一组参数就为目标函数。

- 流程：根据预测值与标签值得到loss -> loss函数对各个参数反向求偏导 -> 计算每个参数的梯度 -> 更新参数值 -> 梯度置零 -> 再次循环
- 代码：

1. A

```
"""用一个三阶函数找到合适的参数 逼近y=sinx
# 1.利用三阶函数y=a+b*x+c*x**2+d*x**3拟合sine,初始参数abcd是任意值
# 2.给定输入，得到sin的输出值为y
# 3.给定输入，得到该函数的输出值,共循环500->2000次
# 3.得到该函数的输出值与y=sinx的输出,偏差loss函数= np.square(y_pre - y).sum()
# 4.为了得到loss最小，求该函数的极小值（导数）grad_y_pre = 2 * (y_pre - y)
# 5.根据梯度值，更新参数 learning_rate=1e-6
"""
```

2. B

```
"""用一个网络模型 逼近y=sinx
# 1.给定输入，得到sin的输出值为y
# 2.给定输入，根据y=a+b*x+c*x**2+d*x**3,计算^1,^2,^3不同幂次下的结果
# 3.构建网络模型，利用线性层将不同幂次下的结果按一定权重相加，包含线性层 Linear(3,1),Flatten()
# 4.将三个结果放入模型得到该函数的输出值,共循环500->2000次
# 4.得到该函数的输出值与y=sinx的输出,偏差loss函数= torch.square(y_pre - y).sum()
# 5.为了得到loss最小，求该函数的极小值（导数）loss.backward()
# 6.根据梯度值，更新参数 param -= learning_rate * param.grad 之后 model.zero_grad()
"""
```

3. C

```
"""用一个三阶函数找到合适的参数 逼近y=sinx auto_grad()
# 1.利用三阶函数y=a+b*x+c*x**2+d*x**3拟合sine,初始参数abcd是任意值
# 2.给定输入，得到sin的输出值为y
# 3.给定输入，得到该函数的输出值,共循环500->2000次
# 3.得到该函数的输出值与y=sinx的输出,偏差loss函数= torch.square(y_pre - y).sum()
# 4.为了得到loss最小，求该函数的极小值（导数）loss.backward()
# 5.根据梯度值，更新参数 a -= learning_rate * a.grad之后a.grad = None
"""
```

4. D

```
"""用一个网络模型 逼近y=sinx
# 1.给定输入，得到sin的输出值为y
# 2.给定输入，根据y=a+b*x+c*x**2+d*x**3,计算^1,^2,^3不同幂次下的结果
# 3.构建网络模型，利用线性层将不同幂次下的结果按一定权重相加，包含线性层 Linear(3,1),Flatten()
# 4.将三个结果放入模型得到该函数的输出值,共循环500->2000次
# 4.得到该函数的输出值与y=sinx的输出,偏差loss函数= torch.square(y_pre - y).sum()
# 5.为了得到loss最小，求该函数的极小值（导数）loss.backward()
# 6.定义优化器torch.optim.RMSprop，根据梯度值，自动更新参数 optimiser.step() 之后 optimizer.zero_grad()
"""
```

CHAPTER 4--神经网络训练

逼近的思路理解训练

用高阶函数构造sine函数

A. 常规方法

- 思路：根据预测值与标签值得到loss -> loss函数对各个参数反向求偏导 -> 计算每个参数的梯度 -> 更新参数值 -> 梯度置零 -> 再次循环

B. loss.backward():

- 思路：根据预测值与标签值得到loss -> loss函数对各个参数反向求偏导 -> loss.backward()自动更新参数的梯度 -> 更新参数值 -> 梯度置零 -> 再次循环
- 区别：每个参数的grad值是自动计算
- 功能：误差张量上调用`.backward()`时，开始反向传播。 然后，Autograd 会为每个模型参数计算梯度并将其存储在参数的`.grad`属性中。

C. optimiser.step()

- 思路：根据预测值与标签值得到loss -> loss函数对各个参数反向求偏导 -> loss.backward()自动更新参数的梯度 -> optimiser.step()更新参数值 -> 梯度置零 -> 再次循环
- 区别：每个参数值是自动更新
- 功能：调用`.step()`启动梯度下降。 优化器通过`.grad`中存储的梯度来调整每个参数

损失函数与优化器

- **常用的损失函数：**

- 平方损失， sum(输出值-预期值)的平方，找出最小的二乘得到的值。
- 最大似然处理，将输出的结果（似然值）视为概率，再去求得到该结果概率值最大的权重系数w。已知事情发生的结果，反推发生该结果概率最大的参数w P(x|w,b)。
- 交叉熵损失：交叉熵越小，两个模型最相似

- 代码：

```
# 1.定义两个变量
# 2.损失函数选择L1Loss()，参量选择 均值与取和——(P1-E1)+(P2-E2)+...(PN-EN)/N
# 3.损失函数选择MSELoss()——(P1-E1)^2+(P2-E2)^2+...(PN-EN)^2/N
```

Autograd全过程

- 在正向传播中，通过模型的每一层运行输入数据以进行预测prediction。

`prediction = model(data) # forward pass`，Autograd 同时执行两项操作

- 运行请求的操作requires_grad = True
- 在 DAG 中维护操作的梯度函数。

- 反向传递，使用模型的预测和相应的标签来计算误差（`loss`），然后通过网络反向传播此误差，在这个过程中 Autograd 会为每个模型参数计算梯度并将其存储在参数的`.grad`属性中。最后利用误差导数并使用梯度下降来优化参数来实现。 - `loss = (prediction - labels).sum() loss.backward() # backward pass`

- 对每个参量计算梯度`.grad_fn`，并保存在各自的张量的`.grad`属性中，
- 使用链式规则，一直反向传播到最开始的输入张量。

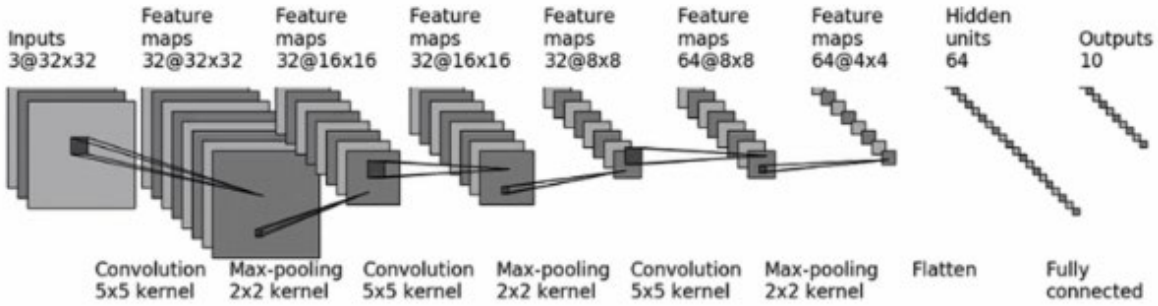
构建神经网络全过程

- 搭建网络的流程
  - 定义具有一些可学习参数（或权重）的神经网络
  - 遍历输入数据集
  - 通过网络处理输入
  - 计算损失（输出正确的距离有多远）
  - 将梯度传播回网络参数
  - 通常使用简单的更新规则来更新网络的权重：`weight = weight - learning_rate * gradient`
- 下载数据->加载数据->准备模型->设置损失函数->设置优化器->开始训练->最后验证->结果聚合展示

构建神经网络训练全过程

- 代码：

```
"""网络结构训练全过程
# 1.加载网络模型
# 2.损失函数和优化器的选择
# 计算均方差函数
# 设置优化器
# 3.网络模型训练
训练10次，一共有5个batch
"""
```



- Mnist训练代码：

```
"""网络的构建+训练+测试
# 1.构建网络，网络将(batch_size,1,28,28)->(batch_size,10)包含
flatten,liner1(28*28,256),relu(),liner1(256,10),softmax()
# 2.下载训练数据集mnist
# 2.定义训练的函数
# 3.调用网路
# 3.调用下载数据集函数，
# 3.利用数据集，选择合适的loss_func,optimiser,训练10次,学习率为1e-6
nn.CrossEntropyLoss() torch.optim.Adam(feed_forward_net.parameters(),lr=Learing_rate)
# 4.保存网络
"""
```

```
"""网络测试
# 1.加载上节课的网络
# 2.调用下载数据集函数，测试数据
# 3.将测试数据输入到模型得到输出，
# 4.输出结果，根据其中的比重分布，占比最大的位置就为索引值
# 5.根据索引图将输出结果转换为label
# 6.查看测试结果
"""
```