**ai**

**python相关库**

* Torch

1. Tensors
2. 和ndarrays相似，但是可以部署到gpu上
3. tensor中可以创建未初始化的、全为零的等固定值的，也可以根据已有tensor创建tensor
4. 可以进行的操作包括转置、索引、切片、数学计算、线性代数、随机数等等
5. 可以和numpy互相转换（共享同个内存空间）
6. 可以部署在gpu/cpu
7. Autograd
8. 进行梯度有关操作
9. with torch.no\_grad()停止梯度计算
10. 神经网络
11. torch.nn
12. 标准训练流程

* 定义一个多层的神经网络
* 对数据集的预处理并准备作为网络的输入
* 将数据输入到网络
* 计算网络的损失
* 反向传播，计算梯度
* 更新网络的梯度，一个简单的更新规则是 weight = weight - learning\_rate \* gradient

1. net.parameters() 可以返回网络的训练参数
2. 损失函数

* 输入(output, target)获得loss

1. 反向传播
2. 清空所有梯度 .zero\_grad()
3. 权重更新

* weight = weight - learning\_rate \* gradient 最简单的方法
* 有很多优化器可以使用
* Matplotlib

1. 包括但不限于axis、tick、grid、legend
2. title中文乱码的问题：

|  |
| --- |
| Plain Text plt.rcParams['font.sans-serif']=['SimHei'] |

1. 可以绘制柱状图、散点图、水平图等等

* Numpy

1. 支持的类型主要包括int8、int16、int32、int64、uint8、uint16、uint32、uint64、float16、float32、float64
2. 可以创建切片、用索引查找内容
3. 可以进行拼接分割维度变换

* Cov2D和Cov1D
* Conv2D 和 Conv1D 是卷积神经网络（CNN）中使用的两种主要卷积操作，它们分别用于处理二维和一维数据。这里是它们的详细说明以及它们的区别：
* Conv2D（二维卷积）:
* Conv2D 主要用于处理图像数据。在卷积操作中，一个二维过滤器滑动在输入数据（通常是图像）上并执行元素乘和求和操作。这个过程对多个通道（例如RGB色彩通道）逐个进行。Conv2D 在图像识别和计算机视觉计算中非常有用。
* Conv1D（一维卷积）:

Conv1D主要用于处理时序数据。与 Conv2D 相似，其在一维输入数据上滑动一个一维过滤器并进行滑动窗口内元素的乘和求和操作。Conv1D 在处理音频数据、文本数据或一般的时序数据上非常有效。

* 主要区别：

1. **输入数据类型**: Conv1D 主要用于处理时间序列数据或者具有单一空间方向的序列数据，例如文本或音频样本。 而 Conv2D 主要用于处理图像数据。
2. **过滤器的维度**： Conv1D 使用一维过滤器，而 Conv2D 在垂直和水平方向上使用二维过滤器。
3. **应用领域**：由于它们处理的数据类型不同, Conv1D 和 Conv2D 在不同的应用领域中使用。例如，Conv1D 在自然语言处理（NLP）、信号处理等领域中常被使用，而 Conv2D 则在计算机视觉任务（如图像处理）中使用。

* BatchNormalization
* 有时激励函数会对一些特定范围的数据不敏感，这个时候Batch Normalization (BN) 就被添加在每一个全连接和激励函数之间会有很好的效果
* LayerNormalization
* 与BN使用场景不同，使用于RNN
* LSTM和GRU
* LSTM(Long Short-Term Memory)
* LSTM是一种特殊的递归神经网络（RNN），能够在长序列数据中有效地捕获时序依赖性。它由Hochreiter & Schmidhuber于1997年提出，解决了传统RNN在处理长距离依赖时出现的梯度消失问题。
* LSTM的主要组成部分包括：

1. **遗忘门**：决定哪些信息从单元状态中忘记或舍弃。
2. **输入门**：决定哪些新进来的信息要更新到单元状态中。
3. **单元状态**：存储单元的中短期记忆。
4. **输出门**：基于当前输入和单元状态，决定最终LSTM单元的输出。

* GRU(Gated Recurrent Unit)
* GRU是Cho等人在2014年提出的RNN支线，是LSTM的变体。GRU试图解决和LSTM相同的问题—— 梯度消失，但构造更为简单。
* GRU的主要组成部分：

1. **更新门**：决定如何将新进来的信息同旧的记忆结合。
2. **重置门**：决定在整合新输入信息时，过去的信息有多少保留。
3. **隐藏状态**：存储模型的中短期记忆。

* LSTM 与 GRU 的区别
* 主要的区别在于内部架构和算法复杂度：

1. **门的数量和内部结构**：LSTM有三个门(输入门，遗忘门，输出门)和一个单元状态，而GRU有两个门(更新门和重置门)和一个隐藏状态。这使得GRU相对简单，计算上更有效。
2. **记忆机制**：GRU结合了单元状态和隐藏状态，而LSTM则将它们分别管理。有些研究结果表明，LSTM的这种特性使其在一些复杂任务中更占优势。
3. **计算资源**：因为GRU的结构更简单，它通常需要更少的计算资源。反之，LSTM则更复杂，需要更多的计算资源。
4. **性能**：在不同的任务和数据集上，LSTM和GRU可能有不同的性能表现。尽管没有明确的规则可以决定应该在何时使用哪种类型，但一般建议在可用资源有限的时候试试GRU，如果需要处理更复杂的数据结构或模式时，可以尝试LSTM。

* 总的来说，这两种网络架构都广泛地使用在序列预测任务中（如文字生成，机器翻译，语音识别等）并且都表现出色。选择哪种取决于具体的应用，数据复杂性和可用资源。
* Dropout
* 可以在一定程度上解决过拟合的问题
* 通过忽略一半的特征检测器
* 优化器

1. **1. SGD（Stochastic Gradient Descent）**

* SGD是最基础的优化算法，它每次使用一个训练样本来进行更新。这样可以减少计算开销和内存用量。

1. **2. Adagrad**

* Adagrad是一种自适应学习率的优化方法，对每个参数具有不同的学习率。对于出现频率较高的特征，它作用的更新速率较慢；对于较低频率的特征，其更新速率较快。然而，Adagrad的一个主要问题是它的学习率始终在下降，可能会在训练中期或后期时降得过低。

1. **3. RMSprop**

* 为了解决Adagrad学习率持续下降的问题，RMSProp应运而生。RMSProp使用"权重衰减"方法（即只考虑一定窗口的梯度平方均值），这使其在非凸情况下的性能更好。

1. **4. Adadelta**

* Adadelta是一种扩展版的RMSprop，其主要改进在于避免手动设置全局学习率。

1. **5. Adam（Adaptive Moment Estimation）**

* Adam结合了Momentum和RMSprop的思想，一方面使用过去梯度的指数加权平均来获取带有动量的效果；另一方面，使用过去梯度平方的指数加权平均来获得自适应学习率的特性。

1. **6. Nadam**

* Nadam是Adam的一种变体，結合了Adam和Nesterov Momentum（NAG）。NAG的一大特点是，参数更新有预测性质。Nadam的效果通常优于Adam。

1. 彼此之间的区别
2. **计算复杂性和内存需求**：SGD最简单，其余的包含更多复杂的操作和更高的内存需求。
3. **收敛速度**：Adagrad, RMSprop, Adadelta, Adam, Nadam通常比SGD收敛快，因为它们考虑到了过去梯度信息，并且有自适应的学习率。
4. **性能**：在多数情况下，Adam和Nadam表现优于其它优化器。
5. **特定问题**：在所有的优化器中，没有哪个优化器在所有任务中表现最好的。因此，适合的优化器取决于具体的问题。

* 激活函数

1. **1. Softmax函数**

* Softmax函数常用于多类别分类问题，它将一组实数转换为概率分布。Softmax函数通过对每个类别的得分进行指数化，并归一化得到概率。这样可以使得各个概率之和等于1，从而表示每个类别的概率大小。

1. **2. ReLU函数（Rectified Linear Unit）**

* ReLU函数是一种常用的激活函数，它在输入大于零时返回该输入，并在输入小于等于零时返回零。ReLU函数具有线性增长的性质，且计算简单，不会引起梯度消失的问题。因此，ReLU函数在深度学习中被广泛使用。

1. **3. Sigmoid函数**

* Sigmoid函数是一种常用的激活函数，它将输入映射到一个介于0和1之间的输出。Sigmoid函数具有平滑的s型曲线，将输入值压缩到概率范围内。由于其输出在0和1之间，Sigmoid函数常用于二分类问题和神经网络的输出层。

1. **4. Tanh函数（双曲正切函数）**

* Tanh函数是一种s型曲线的激活函数，将输入值映射到介于-1和1之间。Tanh函数在零点是对称的，主要用于将输入归一化到负一和正一之间，有助于网络的收敛和稳定性。

1. **5. LeakyReLU函数**

* LeakyReLU函数是ReLU函数的一种扩展形式，它在输入小于零时，不再返回零，而是返回一个较小的负数。LeakyReLU函数的主要目的是解决ReLU函数在输入小于零时可能导致的"神经元死亡"问题，即某些神经元永远不会被激活。通过保留一部分负值，LeakyReLU函数能够缓解这个问题。

1. **直接的区别**

* Softmax函数输出的是概率分布，适用于多类别问题的分类；
* ReLU函数在输入大于零时返回该输入，适用于解决梯度消失问题；
* Sigmoid函数将输入映射到介于0和1之间的输出，适用于二分类问题；
* Tanh函数将输入映射到介于-1和1之间的输出，适用于归一化和稳定性；
* LeakyReLU函数在输入小于零时返回一个较小的负数，与ReLU函数相比，更不容易出现"神经元死亡"问题。
* **1. VGG (Visual Geometry Group)**
* VGG是由牛津大学视觉几何组（Visual Geometry Group）在2014年提出的深度卷积神经网络架构。VGG网络的主要特点是使用重复的简单卷积层来构建网络。它通过使用较小的卷积核和多层堆叠的策略，构建了16层或19层的深度网络结构。VGG网络在ImageNet图像分类竞赛中取得了优异的性能，其简洁和易于理解的架构也成为后续卷积神经网络的基础。
* **2. AlexNet**
* AlexNet是由Alex Krizhevsky等人在2012年提出的卷积神经网络架构。它是首个在ImageNet图像分类竞赛中击败传统方法的深度卷积神经网络。AlexNet采用了8个卷积层和3个全连接层的深层结构，并利用Dropout和ReLU等技术来提高训练效果和减轻过拟合。AlexNet的成功极大地推动了深度学习在计算机视觉领域的发展。
* **3. ResNet (Residual Network)**
* ResNet是由微软研究院在2015年提出的深度卷积神经网络架构。ResNet的核心思想是通过引入跳跃连接（skip connection）来解决深度网络中的梯度消失和网络退化问题。跳跃连接允许信息在网络中以残差的形式传递，使得网络能够更轻松地学习到残差映射，从而构建更深的网络。ResNet的堆叠结构让深度可以达到数十层甚至上百层，取得了在ImageNet图像分类任务上的优异表现。
* **4. ViT (Vision Transformer)**
* ViT是由Google在2020年提出的一种基于Transformer的视觉模型架构。Transformer最初是应用于自然语言处理领域的模型，ViT则将其成功应用到计算机视觉任务中。ViT将输入图像分解为一系列图像块，并使用Transformer编码器来捕捉块之间的上下文关系。通过以块作为输入，而不是像早期的卷积神经网络那样以像素为输入，ViT在许多计算机视觉任务上取得了竞赛级别的性能。
* **5. YOLO (You Only Look Once)**
* YOLO是由Joseph Redmon等人在2015年提出的一种实时目标检测算法。YOLO的特点是将目标检测任务视为回归问题，通过在单个网络中同时预测物体的边界框和类别。与传统的目标检测算法相比，YOLO具有较快的检测速度，能够在实时性要求较高的场景下应用。YOLO算法的最新版本YOLOv4在准确度和速度方面取得了显著的提升，成为目标检测领域的重要算法之一。