面经｜经典算法面试题&知识点汇总（附答案）

极市平台 2022-10-25 22:00:52 发表于广东 手机阅读

以下文章来源于WeThinkIn ，作者Rocky Ding

WeThinkIn

我相信人工智能，数据科学，商业逻辑，金融工具，终身成长，以及顺应时代的潮流会...

作者｜Rocky Ding

来源｜WeThinkIn

编辑｜极市平台

极市导读

总结分享了一些CV算法与机器学习相关的经典面试知识点。 >>加入极市CV技术交流群，走在计算机视觉的最前沿

干货篇

----【目录先行】----

深度学习基础：

1. 深度学习中有哪些经典的优化器？

2. 有哪些提高GAN训练稳定性的Tricks？

经典模型&&热门模型：

1. U-Net模型的结构和特点?

2. RepVGG模型的结构和特点?

机器学习基础：

1. Accuracy、Precision、Recall、F1 Scores的相关概念?

2. 梯度爆炸和梯度消失产生的原因及解决方法?

Python/C/C++知识：

1. Python中的None代表什么?

2. Python中和的区别?

3. C/C++中面向对象和面向过程的区别?

模型部署：

1. 主流AI端侧硬件平台有哪些?

2. 主流AI端侧硬件平台一般包含哪些模块?

图像处理基础：

1. 图像噪声的种类?

2. Python中OpenCV和PIL的区别?

计算机基础：

1. Git，GitLab，SVN的相关知识

2. 协程的相关概念

开放性问题：

1. 如何保持数据持续稳定的支持业务?

2. 如何分辨demo业务，一次性业务以及外包业务?

----【深度学习基础】----

【一】深度学习中有哪些经典的优化器?

SGD（随机梯度下降）

随机梯度下降的优化算法在科研和工业界是很常用的。

很多理论和工程问题都能转化成对目标函数进行最小化的数学问题。

举个例子：梯度下降（Gradient Descent）就好比一个人想从高山上奔跑到山谷最低点，用最快的方式奔向最低的位置。

SGD的公式：

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

动量（Momentum）公式：

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

基本的mini-batch SGD优化算法在深度学习取得很多不错的成绩。然而也存在一些问题需解决：

1. 选择恰当的初始学习率很困难。

2. 学习率调整策略受限于预先指定的调整规则。

3. 相同的学习率被应用于各个参数。

4. 高度非凸的误差函数的优化过程，如何避免陷入大量的局部次优解或鞍点。

AdaGrad（自适应梯度）

AdaGrad优化算法（Adaptive Gradient，自适应梯度），它能够对每个不同的参数调整不同的学习率，对频繁变化的参数以更小的步长进行更新，而稀疏的参数以更大的步长进行更新。

AdaGrad公式：

$$g_{t,i} = \nabla_\theta J(\theta_i)$$

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

$g_{t,i}$ 表示时刻的 θ_i 梯度。

$G_{t,ii}$ 表示t时刻参数 θ_i 的梯度平方和。

与SGD的核心区别在于计算更新步长时，增加了分母：**梯度平方累积和的平方根。** 此项能够累积各个参数 θ_i 的历史梯度平方，频繁更新的梯度，则累积的分母逐渐偏大，那么更新的步长相对就会变小，而稀疏的梯度，则导致累积的分母项中对应值比较小，那么更新的步长则相对比较大。

AdaGrad能够自动为不同参数适应不同的学习率（平方根的分母项相当于对学习率 a 进进 行了自动调整，然后再乘以本次梯度），大多数的框架实现采用默认学习率 $\mathrm{a} = 0.01$ 即可 完成比较好的收敛。

优势： 在数据分布稀疏的场景，能更好利用稀疏梯度的信息，比标准的SGD算法更有效地收敛。

缺点： 主要缺陷来自分母项的对梯度平方不断累积，随时间的增加，分母项越来越大，最终导致学习率收缩到太小无法进行有效更新。

RMSProp

RMSProp结合梯度平方的指数移动平均数来调节学习率的变化。能够在不稳定的目标函数情况下进行很好地收敛。

计算t时刻的梯度：

$$g_t = \nabla_\theta J(\theta_{t-1})$$

计算梯度平方的指数移动平均数（Exponential Moving Average），γ是遗忘因子（或称为指数衰减率），依据经验，默认设置为0.9。

$$v_t = \gamma(v_{t-1}) + (1-\gamma)g_t^2$$

梯度更新的时候，与AdaGrad类似，只是更新的梯度平方的期望（指数移动均值），其中 $\varepsilon = 10^{-8}$，避免除数为 0 。默认学习率 $\alpha = 0.001$。

$$\theta = \theta - \alpha * g / (\sqrt{v} + \varepsilon)$$

$$\theta_t = \theta_{t-1} - \alpha \cdot g_t / (\sqrt{v_t} + \epsilon)$$

优势： 能够克服AdaGrad梯度急剧减小的问题，在很多应用中都展示出优秀的学习率自适应能力。尤其在不稳定(Non-Stationary)的目标函数下，比基本的SGD、Momentum、AdaGrad表现更良好。

Adam

Adam优化器结合了AdaGrad和RMSProp两种优化算法的优点。对梯度的一阶矩估计（First Moment Estimation，即梯度的均值）和二阶矩估计（Second Moment Estimation，即梯度的未中心化的方差）进行综合考虑，计算出更新步长。

Adam的优势：

1. 实现简单，计算高效，对内存需求少。

2. 参数的更新不受梯度的伸缩变换影响。

3. 超参数具有很好的解释性，且通常无需调整或仅需很少的微调。

4. 更新的步长能够被限制在大致的范围内（初始学习率）。

5. 能自然地实现步长退火过程（自动调整学习率）。

6. 很适合应用于大规模的数据及参数的场景。

7. 适用于不稳定目标函数。

8. 适用于梯度稀疏或梯度存在很大噪声的问题。

Adam的实现原理：

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
$v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1-\beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1-\beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

计算t时刻的梯度：

$$g_t = \nabla_\theta J(\theta_{t-1})$$

然后计算梯度的指数移动平均数，m_0 初始化为 0 。

类似于Momentum算法，综合考虑之前累积的梯度动量。

β_1 系数为指数衰减率，控制动量和当前梯度的权重分配，通常取接近于 1 的值。默认为 0.9 。

$$m_t = \beta_1 m_{t-1} + (1-\beta_{\text{werknith}}\, q_1$$

接着，计算梯度平方的指数移动平均数，v_0 初始化为 0 。

β_2 系数为指数衰减率，控制之前的梯度平方的影响情况。默认为 0.999 。

类似于RMSProp算法，对梯度平方进行加权均值。

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

由于 m_0 初始化为 0，会导致 m_t 偏向于 0，尤其在训练初期阶段。

所以，此处需要对梯度均值 m_t 进行偏差纠正，降低偏差对训练初期的影响。

$$\hat{m}_t = m_t / \left(1-\beta_{1=\text{ hilnikln}}^t\right)$$

同时 v_0 也要进行偏差纠正：

$$\hat{v}_t = v_t / \left(1-\beta_2^t\right)$$

最后总的公式如下所示：

$$\theta_{\text{t}} = \theta_{t-1} - \alpha * \hat{m}_t / \left(\sqrt{\hat{v}} + \varepsilon_t\right)$$

其中默认学习率$\alpha = 0.001,\quad \varepsilon = 10^{-8}$，避免除数变为0。

从表达式中可以看出，对更新的步长计算，能够从梯度均值和梯度平方两个角度进行自适应地调节，而不是直接由当前梯度决定。

Adam的不足：

虽然Adam算法目前成为主流的优化算法，不过在很多领域里（如计算机视觉的图像识别、NLP中的机器翻译）的最佳成果仍然是使用带动量（Momentum）的SGD来获取到的。

【二】有哪些提高GAN训练稳定性的Tricks?

1.输入Normalize

1. 将输入图片Normalize到 $[-1, 1]$之间。
2. 生成器最后一层的输出使用Tanh激活函数。

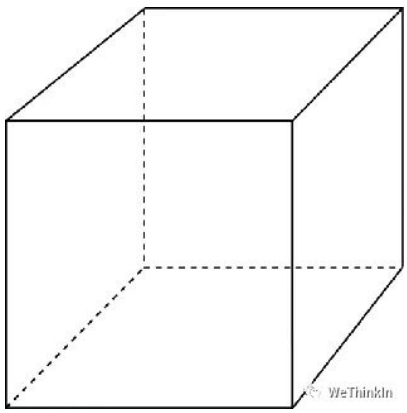
Normalize非常重要，没有处理过的图片是没办法收敛的。图片Normalize一种简单的方法是（images-127.5）/127.5，然后送到判别器去训练。同理生成的图片也要经过判别器，即生成器的输出也是-1到1之间，所以使用Tanh激活函数更加合适。

2.替换原始的GAN损失函数和标签反转

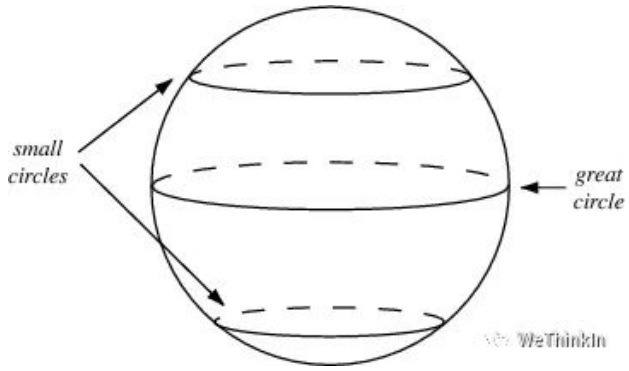
1. 原始GAN损失函数会出现训练早期梯度消失和Mode collapse（模型崩溃）问题。可以使用Earth Mover distance（推土机距离）来优化。
2. 实际工程中用反转标签来训练生成器更加方便，即把生成的图片当成real的标签来训练，把真实的图片当成fake来训练。

3.使用具有球形结构的随机噪声作为输入

1. 不要使用均匀分布进行采样



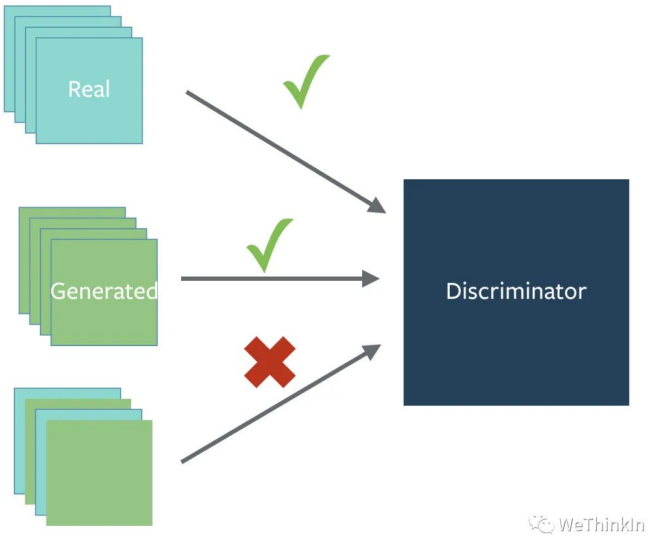
2. 使用高斯分布进行采样



4.使用BatchNorm

1. 一个mini-batch中必须只有real数据或者fake数据，不要把他们混在一起训练。

2. 如果能用BatchNorm就用BatchNorm，如果不能用则用instance normalization。



5.避免使用ReLU，MaxPool等操作引入稀疏梯度

1. GAN的稳定性会因为引入稀疏梯度受到很大影响。

2. 最好使用类LeakyReLU的激活函数。（D和G中都使用）

3. 对于下采样，最好使用：Average Pooling或者卷积+stride。

4. 对于上采样，最好使用：PixelShuffle或者转置卷积+stride。

最好去掉整个Pooling逻辑，因为使用Pooling会损失信息，这对于GAN训练没有益处。

6.使用Soft和Noisy的标签

1. Soft Label，即使用$[0.7-1.2]$ 和 $[0-0.3]$两个区间的随机值来代替正样本和负样本的Hard Label。
2. 可以在训练时对标签加一些噪声，比如随机翻转部分样本的标签。

7.使用Adam优化器

1. Adam优化器对于GAN来说非常有用。
2. 在生成器中使用Adam，在判别器中使用SGD。

8.追踪训练失败的信号

1. 判别器的损失=0说明模型训练失败。
2. 如果生成器的损失稳步下降，说明判别器没有起作用。

9.在输入端适当添加噪声

1. 在判别器的输入中加入一些人工噪声。
2. 在生成器的每层中都加入高斯噪声。

10.生成器和判别器差异化训练

1. 多训练判别器，尤其是加了噪声的时候。

11.Two Timescale Update Rule (TTUR)

对判别器和生成器使用不同的学习速度。使用较低的学习率更新生成器，判别器使用较高的学习率进行更新。

12. Gradient Penalty （梯度惩罚）

使用梯度惩罚机制可以极大增强 GAN 的稳定性，尽可能减少mode collapse问题的产生。

13. Spectral Normalization（谱归一化）

Spectral normalization可以用在判别器的weight normalization技术，可以确保判别器是K-Lipschitz连续的。

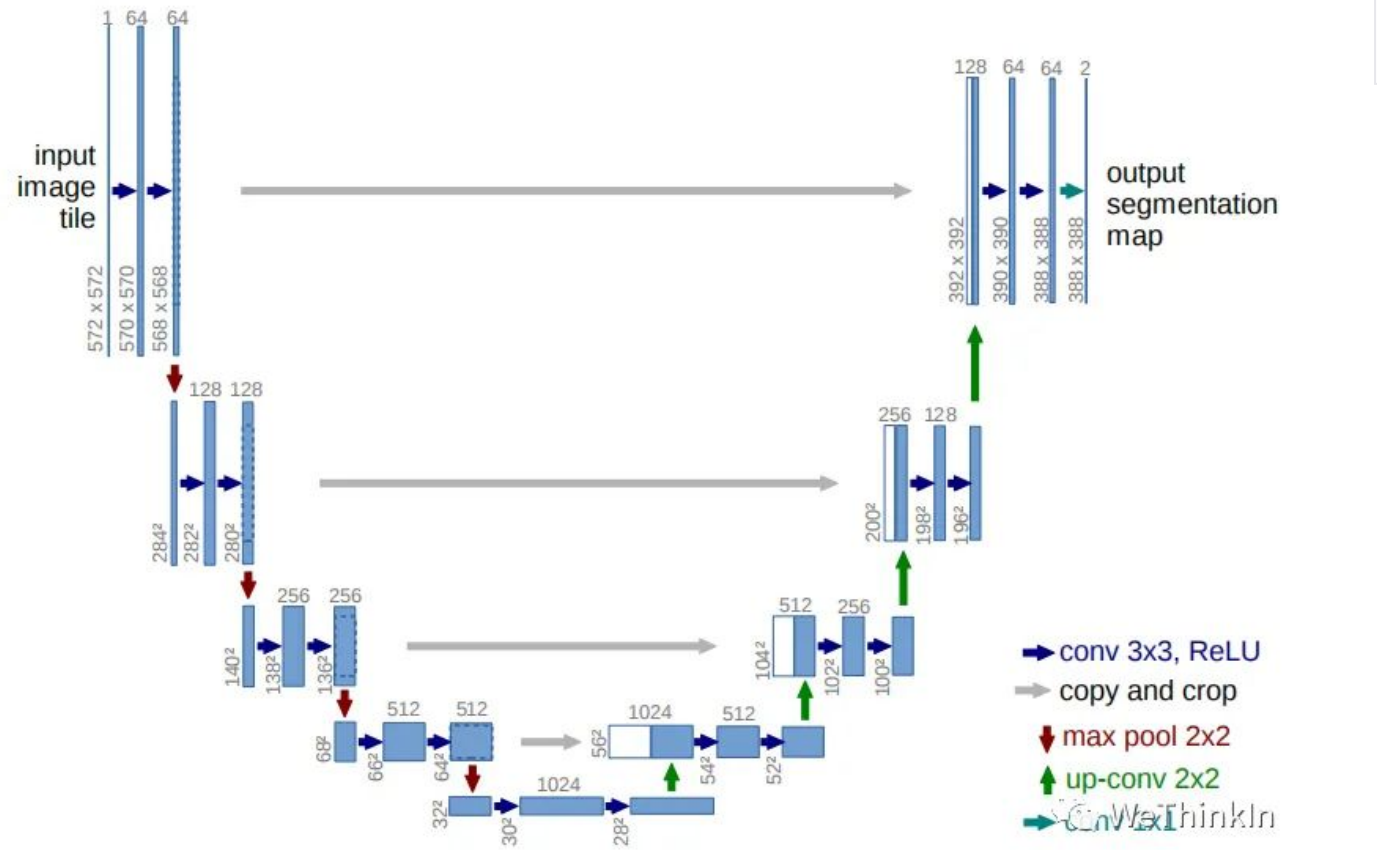
14. 使用多个GAN结构

可以使用多个GAN/多生成器/多判别器结构来让GAN训练更稳定，提升整体效果，解决更难的问题。

【经典模型&&热门模型】

【一】U-Net模型的结构和特点？

U-Net网络结构如下所示：



U-Net网络结构

U-Net网络的特点：

1. 全卷积神经网络：使用1×1卷积完全取代了全连接层，使得模型的输入尺寸不受限制。
2. 左半部分网络是收缩路径（contracting path）：使用卷积和max pooling层，对feature map进行下采样。

3. 右半部分网络是扩张路径（expansive path）：使用转置卷积对feature map进行上采样，并将其与收缩路径对应层产生的特征图进行concat操作。上采样可以补充特征信息，加上与左半部分网络收缩路径的特征图进行concat（通过crop操作使得两个特征图尺寸一致），这就相当于在高分辨率和高维特征当中做一个融合折中。
4. U-Net提出了让人耳目一新的编码器-解码器整体结构，让U-Net充满了生命力与强适应性。

U-Net在医疗图像，缺陷检测以及交通场景中有非常丰富的应用，可以说图像分割实际场景，U-Net是当仁不让的通用Baseline。

U-Net的论文地址：https://arxiv.org/pdf/1505.04597.pdf

【二】RepVGG模型的结构和特点?

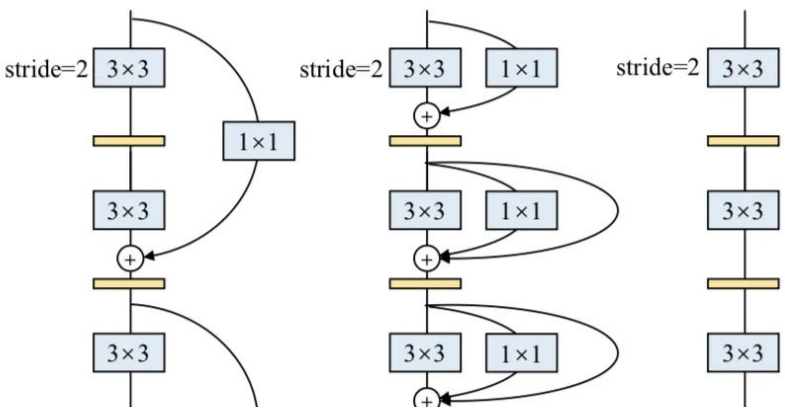
RepVGG模型的基本架构由20多层3×3卷积组成，分成5个stage，每个stage的第一层是stride=2的降采样，每个卷积层用ReLU作为激活函数。

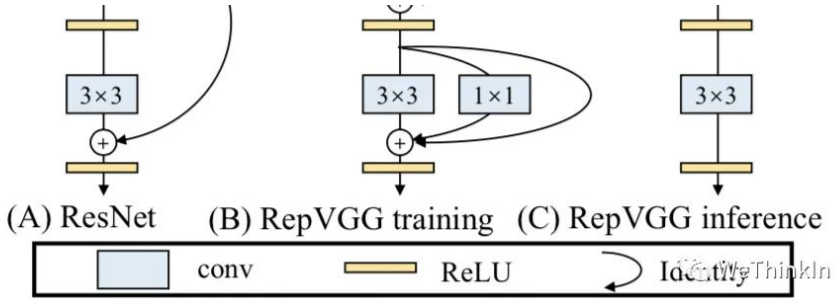
RepVGG的主要特点：

1. 3×3卷积在GPU上的计算密度（理论运算量除以所用时间）可达1x1和5x5卷积的四倍.
2. 直筒型单路结构的计算效率比多路结构高。
3. 直筒型单路结构比起多路结构内存占用少。
4. 单路架构灵活性更好，容易进一步进行模型压缩等操作。
5. RepVGG中只含有一种算子，方便芯片厂商设计专用芯片来提高端侧AI效率。

那么是什么让RepVGG能在上述情形下达到SOTA效果呢?

答案就是**结构重参数化（structural re-parameterization）**。





结构重参数化逻辑

在训练阶段，训练一个多分支模型，并将多分支模型等价转换为单路模型。在部署阶段，部署单路模型即可。这样就可以同时利用多分支模型训练时的优势（性能高）和单路模型推理时的好处（速度快、省内存）。

更多结构重参数化细节知识将在后续的篇章中展开介绍，大家尽情期待！

【机器学习基础】

【一】Accuracy、Precision、Recall、F1 Scores的相关概念?

首先Rocky介绍一下相关名词：

1. TP（True Positive）：预测为正，实际为正

2. FP（False Positive）：预测为正，实际为负

3. TN（True Negative）：预测为负，实际为负

4. FN（false negative）：预测为负，实际为正

Accuracy、Precision、Recall、F1 Scores的公式如下所示：

$$\text{Accuracy} = (\text{true positives} + \text{true negatives}) / (\text{total examples})$$

$$\text{Precision} = (\text{true positives}) / (\text{true positives} + \text{false positives})$$

$$\text{Recall} = (\text{true positives}) / (\text{true positives} + \text{false negatives})$$

$$F_1 \text{ score} = (2 * \text{precision} * \text{recall}) / (\text{precision} + \text{recall})$$

Accuracy（准确率）：分类正确的样本数占样本总数的比例。

Precision（精准度/查准率）：当前预测为正样本类别中被正确分类的样本比例。

Recall（召回率/查全率）：预测出来的正样本占正样本总数的比例。

F1-score是Precision和Recall的综合。F1-score越高，说明分类模型越稳健。

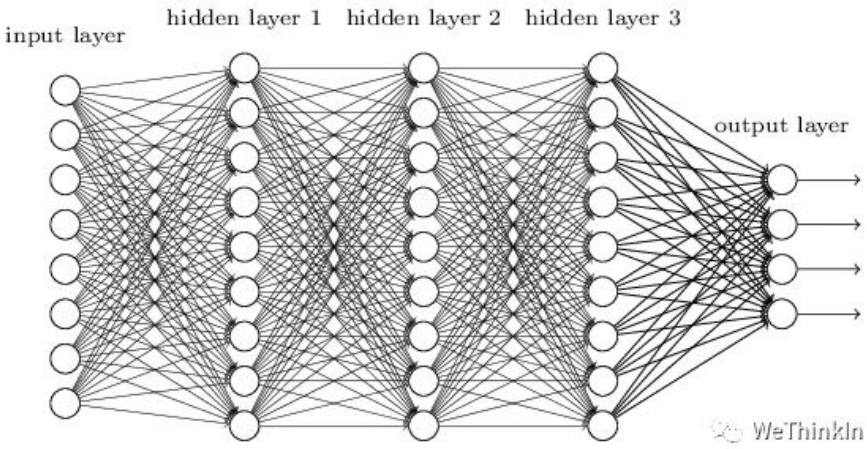
【二】梯度爆炸和梯度消失产生的原因及解决方法?

梯度爆炸和梯度消失问题

一般在深层神经网络中，我们需要预防梯度爆炸和梯度消失的情况。

梯度消失（gradient vanishing problem）和梯度爆炸（gradient exploding problem）一般随着网络层数的增加会变得越来越明显。

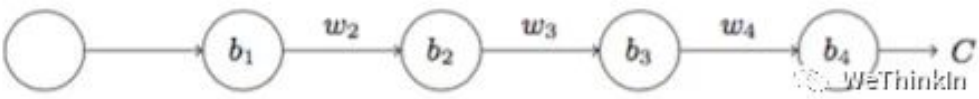
例如下面所示的含有三个隐藏层的神经网络，梯度消失问题发生时，接近输出层的hiden layer 3的权重更新比较正常，但是前面的hidden layer1的权重更新会变得很慢，导致前面的权重几乎不变，仍然接近初始化的权重，**这相当于hidden layer1没有学到任何东西，此时深层网络只有后面的几层网络在学习，而且网络在实际上也等价变成了浅层网络。**



产生梯度爆炸和梯度消失问题的原因

我们来看看看反向传播的过程：

（假设网络每一层只有一个神经元，并且对于每一层$y_i = \sigma(z_i) = \sigma(w_i x_i + b_i)$）



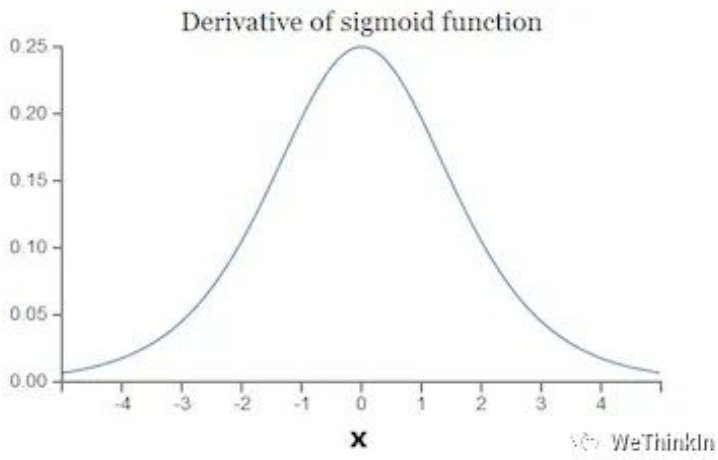
可以推导出：

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial y_4}\frac{\partial y_4}{\partial z_4}\frac{\partial z_4}{\partial x_4}\frac{\partial x_4}{\partial z_3}\frac{\partial z_3}{\partial x_3}\frac{\partial x_3}{\partial z_2}\frac{\partial z_2}{\partial x_2}\frac{\partial x_2}{\partial z_1}\frac{\partial z_1}{\partial b_1}$$

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

$$\frac{\partial y_4}{\partial} \quad (z_4)\, w_4 \quad (z_3)\, w_3 \quad (z_2)\, w_2$$

而sigmoid的导数$\sigma'(x)$如下图所示：



可以知道, $\sigma'(x)$ 的最大值是 $\frac{1}{4}$, 而我们初始化的权重 $|w|$ 通常都小于 1 , 因此 $\sigma'(x)|w| <= \frac{1}{4}$, 而且链式求导层数非常多, 不断相乘的话, 最后的结果越来越小, 趋向 于0, 就会出现梯度消失的情况。

梯度爆炸则相反, $\sigma'(x)|w| > 1$ 时, 不断相乘结果变得很大。

梯度爆炸和梯度消失问题都是因为网络太深，网络权重更新不稳定造成的，本质上是梯度方向传播的连乘效应。

梯度爆炸和梯度消失的解决方法

1. 使用预训练加微调策略。

2. 进行梯度截断。

3. 使用ReLU、LeakyReLU等激活函数。

4. 引入BN层。

5. 使用残差结构。

6. 使用LSTM思想。

【Python/C/C++知识】

【一】Python中的None代表什么？

None是一个特殊的常量，表示空值，其和False，0以及空字符串不同，它是一个特殊Python对象, None的类型是NoneType。

None和任何其他的数据类型比较返回False。

```
>>> None == 0
False

>>> None == ' '
>>> False
>>> None == None
>>> True
>>> None == False
>>> False
>>>
```

我们可以将None复制给任何变量，也可以给None赋值。

【二】Python中和的区别?

$*args$ 和 $**kwargs$主要用于函数定义。我们可以将不定数量的参数传递给一个函数。这里的不定的意思是：预先并不知道函数使用者会传递多少个参数, 所以在这个场景下使用这两个关键字。

$*args$

$*args$是用来发送一个非键值对的可变数量的参数列表给一个函数。

我们直接看一个例子：

```
def test_var_args(f_arg, *argv):
    print("first normal arg:", f_arg)
    for arg in argv:
        print("another arg through *argv:", arg)

test_var_args('hello', 'python', 'ddd', 'test')

-----------------结果如下-----------------------
first normal arg: hello
another arg through *argv: python
another arg through *argv: ddd
another arg through *argv: test
```

$**kwargs$

$**kwargs$允许我们将不定长度的键值对, 作为参数传递给一个函数。如果我们想要在一个函数里处理带名字的参数, 我们可以使用$**kwargs$。

我们同样举一个例子：

```
def greet_me(**kwargs):
    for key, value in kwargs.items():
        print("{0} == {1}".format(key, value))

greet_me(name="yasoob")

------------结果如下-------------
name == yasoob
```

【三】C/C++中面向对象和面向过程的区别?

面向对象（Object Oriented Programming，OOP）编程模型首先抽象出各种对象（各种类），并专注于对象与对象之间的交互，对象涉及的方法和属性都封装在对象内部。

面向对象的编程思想是一种依赖于类和对象概念的编程方式，一个形象的例子是将大象装进冰箱：

1. 冰箱是一个对象，大象也是一个对象。

2. 冰箱有自己的方法，打开、存储、关闭等；大象也有自己的方法，吃、走路等。

3. 冰箱有自己的属性：长、宽、高等；大象也有自己的属性：体重、高度、体积等。

面向过程（Procedure Oriented Programming，POP）编程模型是将问题分解成若干步骤（动作），每个步骤（动作）用一个函数来实现，在使用的时候，将数据传递给这些函数。

面向过程的编程思想通常采用自上而下、顺序执行的方式进行，一个形象的例子依旧是将大象装进冰箱：

1. 打开冰箱。

2. 把大象装进冰箱。

3. 关闭冰箱。

面向对象和面向过程的区别：

1. **安全性角度。** 面向对象比面向过程安全性更高，面向对象将数据访问隐藏在了类的成员函数中，而且类的成员变量和成员函数都有不同的访问属性；而面向过程并没有办法来隐藏程序数据。
2. **程序设计角度。** 面向过程通常将程序分为一个个的函数；而面向对象编程中通常使用一个个对象，函数通常是对象的一个方法。
3. **逻辑过程角度。** 面向过程通常采用自上而下的方法；而面向对象通常采用自下而上的方法。
4. **程序扩展性角度。** 面向对象编程更容易修改程序，更容易添加新功能。

【模型部署】

【一】主流AI端侧硬件平台有哪些?

1. 英伟达
2. 海思
3. 寒武纪
4. 比特大陆
5. 昇腾
6. 登临
7. 联咏
8. 安霸
9. 耐能
10. 爱芯
11. 瑞芯

【二】主流AI端侧硬件平台一般包含哪些模块?

1. 视频编解码模块

2. CPU核心处理模块

3. AI协处理器模块

4. GPU模块

5. DSP模块

6. DDR内存模块

7. 数字图像处理模块

【图像处理基础】

【一】图像噪声的种类?

常规噪声

1. 高斯噪声

2. 脉冲噪声

3. 泊松噪声

4. 乘性噪声

5. 瑞利噪声

6. 伽马噪声

7. 指数噪声

8. 均匀噪声

9. 椒盐噪声

10. 散粒噪声

11. 泊松噪声

对抗噪声

1. 白盒对抗噪声

2. 黑盒查询对抗噪声

3. 黑盒迁移噪声

4. 物理对抗噪声

【二】Python中OpenCV和PIL的区别?

1. 在读取图片时，OpenCV按照BGR的色彩模式渲染通道，而PIL按照RGB的色彩模式渲染通道。
2. OpenCV性能较优，可以作为算法与工程的必备模块。

OpenCV的一些常用操作：

```
import cv2 # 导入OpenCV库
import numpy as np
img = cv2.imread('xxx.jpg', 0) # 读取图片：灰度模式
img = cv2.imread('xxx.jpg', -1) # 读取图片：BGRA模式(BGR+Alpha通道)
img = cv2.imread('xxx.jpg', 1) # 读取图片：BGR模式
img = cv2.imread('xxx.jpg') # 读取图片：第二参数默认为1，BGR模式
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # 将颜色通道从BGR转为RGB
if img == None: # 读取图片失败
  print('image failed to load')
cv2.imshow('src', img) # 图片源src为img
print(img.shape) # 输出图片(高度h,宽度w,通道c)
print(img.size) # 像素总数目
print(img.dtype) # 输出图片类型,uint8为[0-255]
print(img) # 输出所有像素的RGB值
cv2.waitKey()  # 按键关闭窗口
```

```
# waitKey(delay)函数的功能是不断刷新图像，频率时间为delay，单位为ms，返回值为当前键盘按键值

# waitKey() 是在一个给定的时间内(单位ms)等待用户按键触发；如果用户没有按下键,则接续等待(循环)

imgL = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY) # 读取img灰度图

cv2.imshow('gray',imgL) # 图片源gray为imgL

cv2.imwrite('imgL.jpg',imgL) # 将imgL储存名为imgL.jpg的图片

img = img.transpose(2,0,1) # 图片矩阵变换为(通道c,高度h,宽度w)

img = np.expand_dims(img, axis=0) # 图片矩阵扩展维度添加在第一维

print(img.shape) # (1,通道c,高度h,宽度w)

print(img[10,10]) # 访问图片img像素[10,10],输出 [0-255 0-255 0-255]

print(imgL[10,10]) # 访问灰色图片img像素[10,10],输出 0-255

img[10,10] = [255,255,255]  # 修改图片img像素点[10,10]为[255,255,255]

imgL[10,10] = 255 # 修改灰色图片img像素点[10,10]为255

img[:,:,2] = 0 # 将R通道全部修改为0

roi = img[200:550,100:450,:]  # ROI操作,坐标(高度范围,宽度范围,通道范围)

cv2.imshow('roi',roi) # 图片源roi为roi
```

PIL的一些常用操作：

```
from PIL import Image #导入PIL库
import numpy as np

img = Image.open('../xx.jpg') # 读取图片

imgL = Image.open('../xx.jpg').convert('L') # 读取图片灰度图

imgL.show() # 展示灰度图

img1 = img.copy() # 复制图片

print(img.format) # 输出图片格式
```

```
print(img.size) # 输出图片(宽度w,高度h)

print(img.mode) # 输出图片类型,L为灰度图,RGB为真彩色,RGBA为RGB+Alpha透明度

im.show()  # 展示画布

imgData = np.array(img) # 将对象img转化为RGB像素值矩阵

print(imgData.shape) # 输出图片(宽度w,高度h,通道c)

print(imgData.dtype) # 输出图片类型,uint8为[0-255]

print(imgData) # 输出所有像素的RGB值

imgN = Image.fromarray(imgData) # 将RGB像素值矩阵转化为对象imgN

imgN.save('xxx.jpg') # 储存为文件xxx.jpg

r ,g ,b = img.split() # 分离通道

img = Image.merge("RGB", (b, g, r)) # 合并通道

# ROI(region of interest),只对ROI区域操作

roi = img.crop((0, 0, 300, 300)) # (左上x，左上y，右下x，右下y)坐标

roi.show() # 展示ROI区域

#捕捉异IOError,为读取图片失败

try:
  img = Image.open('xxx.jpg')
except IOError:
 print('image failed to read')
```

【计算机基础】

【一】Git，GitLab，SVN的相关知识

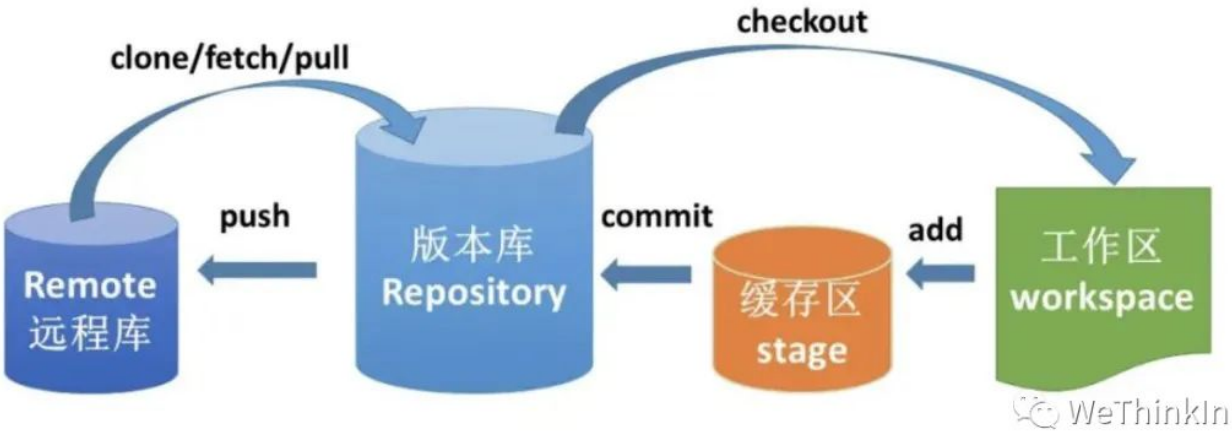
Git

Git是当前主流的一种**开源分布式版本控制系统，可以有效、快速的进行项目版本管理。**

Git没有中央服务器，不同于SVN这种需要中央服务器的集中式版本控制系统。

Git的功能：版本控制（版本管理，远程仓库，分支协作）

Git的工作流程：



Git工作流程

Git的常用命令：

```
git init 创建仓库

git clone 克隆github上的项目到本地

git add  添加文件到缓存区

git commit 将缓存区内容添加到仓库中
```

GitLab

GitLab是一个基于Git实现的在线代码仓库软件，可以基于GitLab搭建一个类似于GitHub的仓库，**但是GitLab有完善的管理界面和权限控制，有较高的安全性，可用于企业和学校等场景。**

SVN

SVN全名Subversion，是一个开源的版本控制系统。**不同于Git，SVN是集中式版本控制系统。**

SVN只有一个集中管理的服务器，保存所有文件的修订版本，而协同工作的人们都通过客户端连到这台服务器，取出最新的文件或者提交更新。

SVN的特点是**安全，效率，资源共享。**

SVN的常用操作：

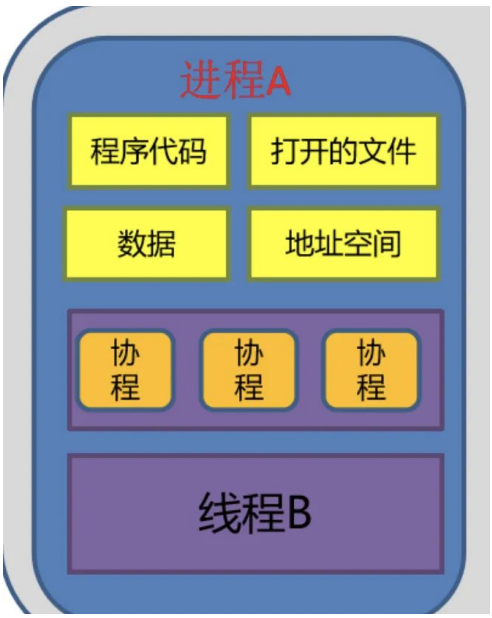
```
Checkout 检出代码

Update 更新代码

Commit 提交代码

Add 提交新增文件

Revert to this version + commit 撤销已经提交的代码
```

【二】协程的相关概念

协程（Coroutine，又称微线程）**运行在线程之上，更加轻量级，协程并没有增加线程总数，只是在线程的基础之上通过分时复用的方式运行多个协程，大大提高工程效率。**

协程的特点：

1. 协程类似于子程序，但执行过程中，协程内部可中断，然后转而执行其他的协程，在适当的时候再返回来接着执行。协程之间的切换不需要涉及任何系统调用或任何阻塞调用。
2. 协程只在一个线程中执行，发生在用户态上的一个逻辑。并且是协程之间的切换并不是线程切换，而是由程序自身控制，协程相比线程节省线程创建和切换的开销。
3. 协程中不需要多线程的锁机制，因为只有一个线程，也不存在同时写变量冲突，在协程中控制共享资源不加锁，只需要判断状态就好了，所以执行效率比多线程高很多。



协程适用于有大量I/O操作业务的场景，可以到达很好的效果，一是降低了系统内存，二是减少了系统切换开销，因此系统的性能也会提升。

在协程中尽量不要调用阻塞I/O的方法，比如打印，读取文件等，除非改为异步调用的方式，并且协程只有在I/O密集型的任务中才会发挥作用。

【开放性问题】

这些问题基于我的思考提出，希望除了能给大家带来面试的思考，也能给大家带来面试以外的思考。这些问题没有标准答案，我相信每个人心中都有自己灵光一现的创造，你的呢?

【一】如何保持数据持续稳定的支持业务?

在“CV兵器”基本上都是开源的情况下，数据成为了支持业务迭代最重要的一部分，如何建立数据护城河，形成业务与数据双向正反馈，是AI行业从业者必须要面对的课题。

【二】如何分辨demo业务，一次性业务以及外包业务?

这个问题不仅可以考察面试者，面试者也可以用来反向判断面试官及其背后公司的运行逻辑。陷入demo业务，一次性业务以及外包业务的循环中是无法成长的，也不利于建立业务/产品的护城河。知道了这一点，那么如何去选择部门，如何去选择公司，如何去看需求，就变成了非常值得研究的事情。

765篇原创内容

公众号

△点击卡片关注极市平台，获取最新CV干货

极市干货

算法竞赛：往届获奖方案总结以及经验详解｜ACCV2022国际细粒度图像分析挑战赛

技术综述：BEV 学术界和工业界方案、优化方法与tricks综述｜PyTorch下的可视化工具（网络结构/训练过程可视化）

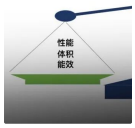
极视角动态：极视角与华为联合发布基于昇腾AI的「AICE赋能行业解决方案」｜算法误报怎么办？自训练工具使得算法迭代效率提升50%！

阅读原文

喜欢此内容的人还喜欢

YOLOv7部署加速比5.89，BERT部署加速比6.37，自动化压缩工具实战30+热门AI模型

飞桨PaddlePaddle



机器学习模型迭代方法(Python)

算法进阶

『拼多多』数据分析岗面试真题（含答案）

数据攻略