1. **神经网络权重初始化值不能全相同：**

因为如果神经网络计算出来的输出值都一个样，那么反向传播算法计算出来的梯度值一样，并且参数更新值也一样(w=w−α∗dw)。更一般地说，如果权重初始化为同一个值，网络就不可能不对称(即是对称的)。

太小，梯度消失，太大，梯度爆炸。

如何确定合适的初始化值。

为了防止以上问题的出现，我们可以坚持以下经验原则：

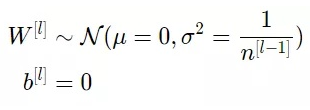
1.激活的平均值应为零。

2.激活的方差应该在每一层保持不变。

合理的权重初始化可以防止梯度爆炸和消失。对于ReLu激活函数，权重可初始化为：

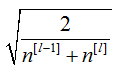


其中n[l−1] 是第l-1层网络中的神经元数量，也叫作“He初始化”。对于tanh激活函数，权重初始化为：





也称为“Xavier初始化”。也可以使用下面这个公式进行初始化：



上述公式中的l指当前处在神经网络的第几层，l-1为上一层。

****2、**如何理解卷积、池化等、全连接层等操作：**

卷积的作用：捕获图像相邻像素的依赖性；起到类似滤波器的作用，得到不同形态的feature map。

激活函数的作用：引入非线性因素。

**3、1x1大小的卷积核的作用**

通过控制卷积核个数实现升维或者降维，从而减少模型参数

对不同特征进行归一化操作

用于不同channel上特征的融合

**4、池化的作用：**

减少特征维度大小，使特征更加可控；

减少参数个数，从而控制过拟合程度；

增加网络对略微变换后的图像的鲁棒性；

达到一种尺度不变性，即无论物体在图像中哪个方位均可以被检测到；

**5、梯度消失梯度爆炸**

预训练加微调

- 梯度剪切、权重正则（针对梯度爆炸）

- 使用不同的激活函数 - 使用batchnorm

- 使用残差结构

- 使用LSTM网络

Relu:

特点：

- 当输入为正数时，输出导数恒为1，缓解了梯度消失的问题。

- 为网络带来稀疏性，当输入值小于0，就会被稀疏掉，人的大脑稀疏性高达95%。

- 不管是正向计算，还是导数计算都非常简单。

缺点：

- 左硬饱和性，当输入小于零时，导数恒为0，会使很多神经元无法得到更新，出现“神经元死亡”。

- relu函数输出无负值。

- 均值漂移，relu函数的输出均值恒大于0（从relu函数的输出范围就能看出来）。

这就导致了tanh特征相差明显时的效果会很好，在循环过程中会不断扩大特征效果显示出来，但有是，在特征相差比较复杂或是相差不是特别大时，需要更细微的分类判断的时候，sigmoid效果就好了。  
还有一个东西要注意，sigmoid 和 tanh作为激活函数的话，一定要注意一定要对 input 进行归一话，否则激活后的值都会进入平坦区，使隐层的输出全部趋同，但是 ReLU 并不需要输入归一化来防止它们达到饱和

**6、BN训练和测试时的参数是一样的嘛？**

对于BN，在训练时，是对每一批的训练数据进行归一化，也即用每一批数据的均值和方差。

而在测试时，比如进行一个样本的预测，就并没有batch的概念，因此，这个时候用的均值和方差是全量训练数据的均值和方差，这个可以通过**移动平均法**求得。

对于BN，当一个模型训练完成之后，它的所有参数都确定了，包括均值和方差，gamma和bata。

BN训练时为什么不用全量训练集的均值和方差呢？

因为用全量训练集的均值和方差容易过拟合，对于BN，其实就是对每一批数据进行归一化到一个相同的分布，而每一批数据的均值和方差会有一定的差别，而不是用固定的值，这个差别实际上能够增加模型的鲁棒性，也会在一定程度上减少过拟合。

也正是因此，BN一般要求将训练集完全打乱，并用一个较大的batch值，否则，一个batch的数据无法较好得代表训练集的分布，会影响模型训练的效果。

1. **空洞卷积**

不做pooling损失信息的情况下，加大了感受野，让每个卷积输出都包含较大范围的信息。

1. **Padding**

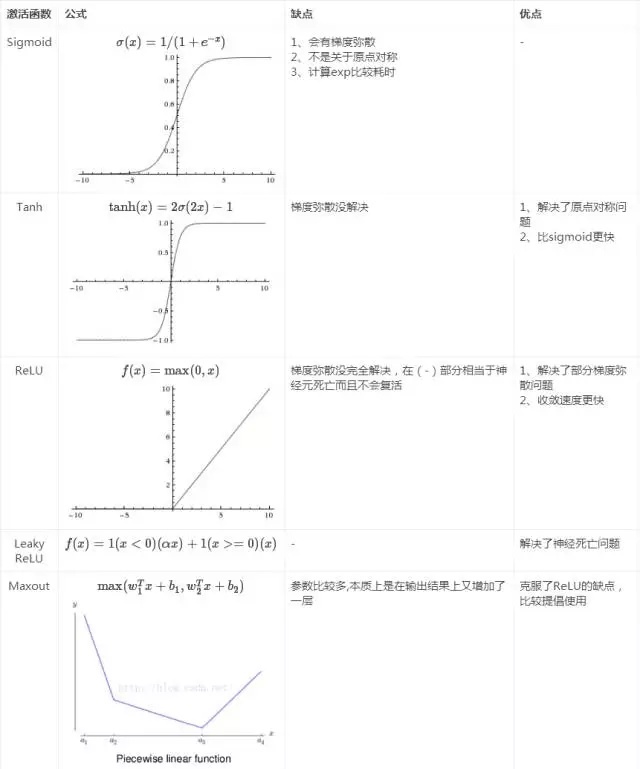
保持边界信息，如果没有加padding的话，输入图片最边缘的像素点信息只会被卷积核操作一次，但是图像中间的像素点会被扫描到很多遍，那么就会在一定程度上降低边界信息的参考程度，但是在加入padding之后，在实际处理过程中就会从新的边界进行操作，就从一定程度上解决了这个问题。

可以利用padding对输入尺寸有差异图片进行补齐，使得输入图片尺寸一致。

（3）在卷积神经网络的卷积层加入Padding，可以使得卷积层的输入维度和输出维度一致。（4）卷积神经网络的池化层加入Padding，一般都是保持边界信息和（1）所述一样。

1. **激活函数，它们的优缺点分别是什么**

激活函数的发展经历了Sigmoid -> Tanh -> ReLU -> Leaky ReLU -> Maxout这样的过程，还有一个特殊的激活函数Softmax，因为它只会被用在网络中的最后一层，用来进行最后的分类和归一化。

****

****Sigmoid:****

Sigmoid函数饱和使梯度消失。当神经元的激活在接近0或1处时会饱和，在这些区域梯度几乎为0，这就会导致梯度消失，几乎就有没有信号通过神经传回上一层。

Sigmoid函数的输出不是零中心的。因为如果输入神经元的数据总是正数，那么关于IMG_256的梯度在反向传播的过程中，将会要么全部是正数，要么全部是负数，这将会导致梯度下降权重更新时出现z字型的下降。

****Tanh:****

Tanh解决了Sigmoid的输出是不是零中心的问题，但仍然存在饱和问题。

为了防止饱和，现在主流的做法会在激活函数前多做一步batch normalization，尽可能保证每一层网络的输入具有均值较小的、零中心的分布。

****Relu:****

ReLU非线性函数图像如下图所示。相较于sigmoid和tanh函数，ReLU对于随机梯度下降的收敛有巨大的加速作用；sigmoid和tanh在求导时含有指数运算，而ReLU求导几乎不存在任何计算量。

对比sigmoid类函数主要变化是：

1）单侧抑制；

2）相对宽阔的兴奋边界；

3）稀疏激活性。

存在问题：

ReLU单元比较脆弱并且可能“死掉”，而且是不可逆的，因此导致了数据多样化的丢失。通过合理设置学习率，会降低神经元“死掉”的概率。

Relu的变种：

1. **各个评价指标**
2. **各个优化函数**

**https://zhuanlan.zhihu.com/p/32230623**

1. **各个损失函数**

对回归问题，常用的有：平方损失(for linear regression)，绝对值损失；

对分类问题，常用的有：hinge loss(for soft margin SVM)，log loss(for logistic regression)。

1. **如何知道网络中是否有梯度爆炸问题？**

在网络训练过程中，如果发生梯度爆炸，那么会有一些明显的迹象表明这一点，例如：

模型无法在训练数据上收敛（比如，损失函数值非常差）；

模型不稳定，在更新的时候损失有较大的变化；

模型的损失函数值在训练过程中变成NaN值；

如果你遇到上述问题，我们就可以深入分析网络是否存在梯度爆炸问题。还有一些不太为明显的迹象可以用来确认网络中是否存在梯度爆炸问题：

模型在训练过程中，权重变化非常大；

模型在训练过程中，权重变成NaN值；

每层的每个节点在训练时，其误差梯度值一直是大于1.0；

1. **如何解决梯度爆炸问题？**

1.重新设计网络模型

在深层神经网络中，梯度爆炸问题可以通过将网络模型的层数变少来解决。此外，在训练网络时，使用较小批量也有一些好处。在循环神经网络中，训练时使用较小时间步长更新（也被称作[截断反向传播](https://link.zhihu.com/?target=https://machinelearningmastery.com/gentle-introduction-backpropagation-time/" \t "https://zhuanlan.zhihu.com/p/_blank)）可能会降低梯度爆炸发生的概率。

2.使用修正线性激活函数

在深度多层感知机中，当激活函数选择为一些之前常用的Sigmoid或Tanh时，网络模型会发生梯度爆炸问题。而使用修正线性激活函数（ReLU）能够减少梯度爆炸发生的概率。

3.使用长短周期记忆网络

由于循环神经网络中存在的固有不稳定性，梯度爆炸可能会发生。比如，通过时间反向传播，其本质是将循环网络转变为深度多层感知神经网络。通过使用长短期记忆单元（LSTM）或相关的门控神经结构能够减少梯度爆炸发生的概率。

4.使用梯度裁剪

在深度多层感知网络中，当有大批量数据以及LSTM是用于很长时间序列时，梯度爆炸仍然会发生。当梯度爆炸发生时，可以在网络训练时检查并限制梯度的大小，如果梯度值大于某个阈值，我们就进行梯度裁剪。

具体而言，检查误差梯度值就是与一个阈值进行比较，若误差梯度值超过设定的阈值，则截断或设置为阈值。

在Keras深度学习库中，在训练网络之前，可以对优化器的clipnorm和 clipvalue参数进行设置来使用梯度裁剪，一般而言，默认将clipnorm和 clipvalue分别设置为1和0.5.

5.使用权重正则化

如果梯度爆炸问题仍然发生，另外一个方法是对网络权重的大小进行校验，并对大权重的损失函数增添一项惩罚项。

在Keras深度学习库中，可以在每层上使用L1或L2正则器设置kernel\_regularizer参数来完成权重的正则化操作。

1. **如何加速网络收敛**

优化初始权值

Batchnomal

Adam

Dropout

换激活函数

每一个epoch就shuffle训练数据

1. **网络出现的问题**

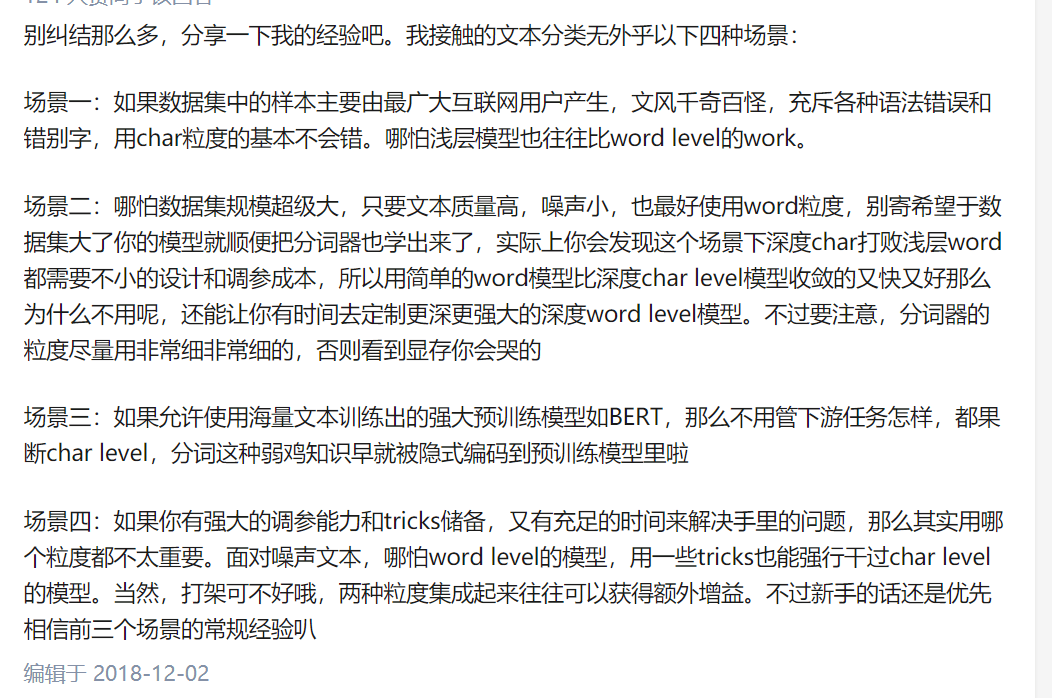
LOSS不下降，准确率升高

<https://www.zhihu.com/question/291369144>

1. **梯度消失产生的原因，尽量多说**
2. **Padding在训练网络之前和之后有什么不同**
3. **Beamsearch连乘太小的时候怎么办**

Beam search算法也有许多改进的地方，根据最后的概率公式可知，该算法倾向于选择最短的句子，因为在这个连乘操作中，每个因子都是小于1的数，因子越多，最后的概率就越小。解决这个问题的方式，最后的概率值除以这个生成序列的单词数（记生成序列的单词数为N），这样比较的就是每个单词的平均概率大小。  
此外，连乘因子较多时，可能会超过浮点数的最小值，可以考虑取对数来缓解这个问题。

1. **char level word level怎么抉择**



1. **怎么评价词向量的好坏：**

对于一些传统方法做的任务，可以直接当作特征加进去，看看提升的效果。  
对于用神经网络做的，可以用词向量作为词那一层的初始值，初始值选得好，就当做词向量好。

比如常见的用来评价近义词（353），实际上就是实现标记了若干近义词对。把word2vec的词向量当做输入，直接用马氏距离或者cos计算个分类器，看对近义词的分类效果。

可以尝试A-B=C-D这种方式，word analogy。

目前常用的衡量word embedding质量好坏的task主要分为两类：

word analogy & document classification

1.word analogy task

假设给了一对单词 (a , b) 和一个单独的单词c, task会找到一个单词d，使得c与d之间的关系相似于a与b之间的关系，举个简单的例子：(中国，北京)和 日本，应该找到的单词应该是东京，因为北京是中国的首都，而东京也是日本的首都。 在给定word embedding的前提下，task一般是通过在词向量空间寻找离(b-a+c)最近的词向量来找到d。

2.document classification task

这是一种通过使用词向量来计算文本向量（可以被用来进行文本分类的工作）的方法，为了得到文本向量，task通常采用了一种很简单的方法:取文本中所有词的词向量的平均值来代表该文本的文本向量，进而通过文本分类的准确度来反向评价训练出的词向量的质量。

对于给定的word embedding，以上的方法确实可以大致地评价词向量的质量好坏，但我们不应该局限于它们，就像楼上所言，我们应该以word embedding对于实际任务的收益为评价标准，如果脱离了实际任务，很难讲不同的word embedding谁比谁更“好”，因为word embedding更像是一个解决问题所使用的工具而不是一个问题的终点。

1. **Adam容易出现什么问题**

<https://zhuanlan.zhihu.com/p/32338983>

但AdaDelta和Adam则不然。二阶动量是固定时间窗口内的累积，随着时间窗口的变化，遇到的数据可能发生巨变，使得 IMG_256可能会时大时小，不是单调变化。这就可能在训练后期引起学习率的震荡，导致模型无法收敛。

这篇文章也给出了一个修正的方法。由于Adam中的学习率主要是由二阶动量控制的，为了保证算法的收敛，可以对二阶动量的变化进行控制，避免上下波动。

IMG_256

通过这样修改，就保证了IMG_256 ，从而使得学习率单调递减。

Adam的收敛速度比SGD要快，但最终收敛的结果并没有SGD好。他们进一步实验发现，主要是后期Adam的学习率太低，影响了有效的收敛。他们试着对Adam的学习率的下界进行控制，发现效果好了很多。

于是他们提出了一个用来改进Adam的方法：前期用Adam，享受Adam快速收敛的优势；后期切换到SGD，慢慢寻找最优解。这一方法以前也被研究者们用到，不过主要是根据经验来选择切换的时机和切换后的学习率。这篇文章把这一切换过程傻瓜化，给出了切换SGD的时机选择方法，以及学习率的计算方法，效果看起来也不错。

不同优化算法的优劣依然是未有定论的争议话题。据我在paper和各类社区看到的反馈，主流的观点认为：Adam等自适应学习率算法对于稀疏数据具有优势，且收敛速度很快；但精调参数的SGD（+Momentum）往往能够取得更好的最终结果。

1. **各种注意力**

***信息的加权平均。***

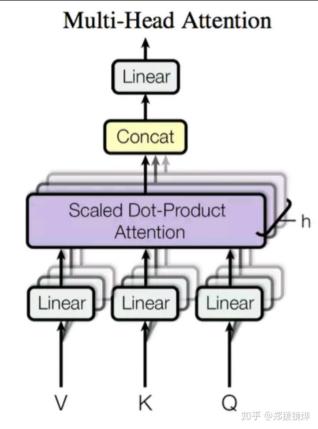
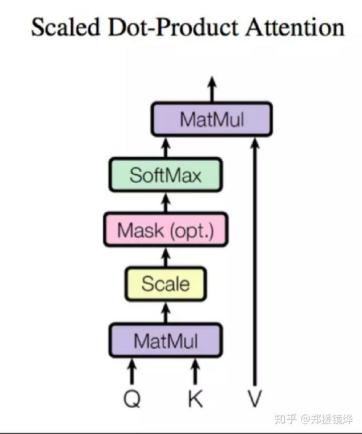
<https://www.zhihu.com/question/68482809/answer/597944559>

1. **Mutihead作用**

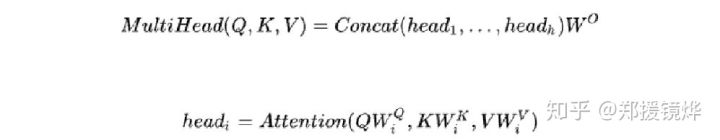
原文中使用了 8 个 scaled dot-product attention ，在同一 multi-head attention 层中，输入均为 KQV ，同时进行注意力的计算，彼此之前参数不共享，最终将结果拼接起来，这样可以允许模型在不同的表示子空间里学习到相关的信息。

简而言之，就是希望每个注意力头，只关注最终输出序列中一个子空间，互相独立。其核心思想在于，抽取到更加丰富的特征信息。

1. **Multihead 很多不同维度做非线性操作再concat,具体？**



直白的解释就是将上面的Scaled Dot-Product Attention步骤重复执行，然后将每次执行的结果拼接起来，需要注意的是每次重复执行Scaled Dot-Product Attention步骤的参数并不共享。



1. **Seq2seq decoder输入几个**
2. **Attention有什么缺陷**

优点：

一步到位获取全局与局部的联系，不会像RNN网络那样对长期依赖的捕捉会收到序列长度的限制。

每步的结果不依赖于上一步，可以做成并行的模式

相比CNN与RNN，参数少，模型复杂度低。(根据attention实现方式不同，复杂度不一）

缺点：

没法捕捉位置信息，即没法学习序列中的顺序关系。这点可以通过加入位置信息，如通过位置向量来改善，具体可以参考最近大火的BERT模型。

1. 网络训练过程中，节约内存的方法
2. 网络训练过程中，节约显存的方法
3. Autograd的底层实现
4. Dataloder效率低，有什么高效的读取方式