一位算法工程师从30+场秋招面试中总结出的超强面经—语义分割篇（含答案）

一位算法工程师从30+场秋招面试中总结出的超强面经——目标检测篇（含答案）

优化算法

深度学习优化学习方法（一阶、二阶）

一阶方法：随机梯度下降（SGD）、动量（Momentum）、牛顿动量法（Nesterov动量）、AdaGrad（自适应梯度）、RMSProp（均方差传播）、Adam、Nadam。

二阶方法：牛顿法、拟牛顿法、共轭梯度法（CG）、BFGS、L-BFGS。

自适应优化算法有哪些？（Adagrad（累积梯度平方）、RMSProp（累积梯度平方的滑动平均）、Adam（带动量的RMSProp，即同时使用梯度的一、二阶矩））。

梯度下降陷入局部最优有什么解决办法？可以用BGD、SGD、MBGD、momentum，RMSprop，Adam等方法来避免陷入局部最优。

1.梯度下降法原理

梯度下降法又称最速下降法，是求解无约束最优化问题的一种最常用的方法，在对损失函数最小化时经常使用。梯度下降法是一种迭代算法。选取适当的初值x(0)，不断迭代，更新x的值，进行目标函数的极小化，直到收敛。由于负梯度方向时使函数值下降最快的方向，在迭代的每一步，以负梯度方向更新x的值，从而达到减少函数值的目的。

我们首先确定损失函数：

$$J(\Theta)=\frac{1}{2m}\sum_{j=1}^{m}\left[h_{\Theta}\left(x^{i}\right)-y^{i}\right]^{2}$$

其中，J(θ)是损失函数，m代表每次取多少样本进行训练，如果采用SGD进行训练，那每次随机取一组样本，m=1；如果是批处理，则m等于每次抽取作为训练样本的数量。θ是参数，对应（1式）的θ1和θ2。求出了θ1和θ2，h(x)的表达式就出来了：

$$h(\Theta)=\sum\Theta_j x_j=\Theta_1 x_1+\Theta_2 x_2$$

我们的目标是让损失函数J(θ)的值最小，根据梯度下降法，首先要用J(θ)对θ求偏导：

$$\frac{\sigma J(\Theta)}{\sigma\Theta_j}=2\frac{1}{2m}\sum_{i=1}^{m}\left[h_\Theta\left(x^i\right)-y^i\right]x_j^i=\frac{1}{m}\sum_{i=1}^{m}\left[h_\Theta\left(x^i\right)-y^i\right]x_j^i$$

由于是要最小化损失函数，所以参数θ按其负梯度方向来更新：

$$\Theta'=\Theta_j-\frac{\sigma J(\Theta)}{\sigma\Theta_j}=\Theta_j-\alpha\frac{1}{m}\sum_{m}^{i=1}\left(y^i-h_\Theta\left(x^i\right)\right)x_j^i$$

①批量梯度下降（BGD）

批量梯度下降法，是梯度下降法最常用的形式，具体做法也就是在更新参数时使用所有的样本来进行更新

$$\theta_j:=\theta_j+\alpha\sum_{i=1}^{m}\left(y_i-\theta_i^T x_i\right)x_j$$

写成伪代码如下：

```
for i in range(nb_epochs):
    params_grad = evaluate_gradient(loss_function, data, params)
    params = params - learning_rate * params_grad
```

优点：（1）一次迭代是对所有样本进行计算，此时利用矩阵进行操作，实现了并行。（2）由全数据集确定的方向能够更好地代表样本总体，从而更准确地朝向极值所在的方向。当目标函数为凸函数时，BGD一定能够得到全局最优。

缺点：（1）当样本数目 m 很大时，每迭代一步都需要对所有样本计算，训练过程会很慢。（2）不能投入新数据实时更新模型。

②随机梯度下降（SGD）

随机梯度下降法求梯度时选取一个样本j来求梯度。

$$\theta_j := w_j + \alpha\left(y_i - \theta_i^T x_i\right) x_j$$

写成伪代码如下：

```
for i in range(nb_epochs):
    np.random.shuffle(data)
    for example in data:
        params_grad = evaluate_gradient(loss_function , example ,params)
        params = params - learning_rate * params_grad
```

优点：（1）由于不是在全部训练数据上的损失函数，而是在每轮迭代中，随机优化某一条训练数据上的损失函数，这样每一轮参数的更新速度大大加快。

缺点：（1）准确度下降。由于即使在目标函数为强凸函数的情况下，SGD仍旧无法做到线性收敛。（2）可能会收敛到局部最优，由于单个样本并不能代表全体样本的趋势。（3）不易于并行实现。SGD 因为更新比较频繁，会造成 cost function 有严重的震荡。

③小批量梯度下降算法（mini-batch GD）

小批量梯度下降法是是对于m个样本，我们采用x个样子来迭代，1<x<m。一般可以取x=10，当然根据样本的数据，可以调整这个x的值。

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

伪代码如下：

```
for i in range(nb_epochs):
    np.random.shuffle(data)
    for batch in get_batches(data, batch_size=50):
        params_grad = evaluate_gradient(loss_function, batch, params)
        params = params - learning_rate * params_grad
```

优点：（1）通过矩阵运算，每次在一个batch上优化神经网络参数并不会比单个数据慢太多。（2）每次使用一个batch可以大大减小收敛所需要的迭代次数，同时可以使收敛到的结果更加接近梯度下降的效果。(比如上例中的30W，设置batch_size=100时，需要迭代3000次，远小于SGD的30W次)（3）可实现并行化。

缺点(解释1)：

1.不过 Mini-batch gradient descent 不能保证很好的收敛性，learning rate 如果选择的太小，收敛速度会很慢，如果太大，loss function 就会在极小值处不停地震荡甚至偏离。（有一种措施是先设定大一点的学习率，当两次迭代之间的变化低于某个阈值后，就减小 learning rate，不过这个阈值的设定需要提前写好，这样的话就不能够适应数据集的特点。）对于非凸函数，还要避免陷于局部极小值处，或者鞍点处，因为鞍点周围的error是一样的，所有维度的梯度都接近于0，SGD 很容易被困在这里。（会在鞍点或者局部最小点震荡跳动，因为在此点处，如果是训练集全集带入即BGD，则优化会停止不动，如果是mini-batch或者SGD，每次找到的梯度都是不同的，就会发生震荡，来回跳动。）

2.SGD对所有参数更新时应用同样的 learning rate，如果我们的数据是稀疏的，我们更希望对出现频率低的特征进行大一点的更新。LR会随着更新的次数逐渐变小。

缺点(解释2)：：

（1）batch_size的不当选择可能会带来一些问题。

batcha_size的选择带来的影响：在合理地范围内，增大batch_size的好处：a. 内存利用率提高了，大矩阵乘法的并行化效率提高。b. 跑完一次 epoch（全数据集）所需的迭代次数减少，对于相同数据量的处理速度进一步加快。c. 在一定范围内，一般来说 Batch_Size 越大，其确定的下降方向越准，引起训练震荡越小。

（2）盲目增大batch_size的坏处：

a. 内存利用率提高了，但是内存容量可能撑不住了。b. 跑完一次 epoch（全数据集）所需的迭代次数减少，要想达到相同的精度，其所花费的时间大大增加了，从而对参数的修正也就显得更加缓慢。c. Batch_Size 增大到一定程度，其确定的下降方向已经基本不再变化。

2.梯度下降算法改进

①动量梯度下降法（Momentum）

Momentum 通过加入 γ*vt−1 ，可以加速 SGD， 并且抑制震荡。momentum即动量，它模拟的是物体运动时的惯性，即更新的时候在一定程度上保留之前更新的方向，同时利用当前batch的梯度微调最终的更新方向。这样一来，可以在一定程度上增加稳定性，从而学习地更快，并且还有一定摆脱局部最优的能力。动量法做的很简单，相信之前的梯度。如果梯度方向不变，就越发更新的快，反之减弱当前梯度。r一般为0.9。

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

缺点：这种情况相当于小球从山上滚下来时是在盲目地沿着坡滚，如果它能具备一些先知，例如快要上坡时，就知道需要减速了的话，适应性会更好。

②Nesterov accelerated gradient法（NAG）

用 θ−γv_t−1 来近似当做参数下一步会变成的值，则在计算梯度时，不是在当前位置，而是未来的位置上。仍然是动量法，只是它要求这个下降更加智能。这个算法就可以对低频的参数做较大的更新，对高频的做较小的更新，也因此，对于稀疏的数据它的表现很好，很好地提高了 SGD 的鲁棒性。

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

esterov 的好处就是，当梯度方向快要改变的时候，它提前获得了该信息，从而减弱了这个过程，再次减少了无用的迭代。超参数设定值: 一般 γ 仍取值 0.9 左右。

③Adagrad

这个算法就可以对低频的参数做较大的更新，对高频的做较小的更新，也因此，对于稀疏的数据它的表现很好，很好地提高了 SGD 的鲁棒性，例如识别 Youtube 视频里面的猫，训练 GloVe word embeddings，因为它们都是需要在低频的特征上有更大的更新。

梯度更新规则:

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

其中g为：t时刻参数θ_i的梯度

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

如果是普通的 SGD， 那么 θ_i 在每一时刻的梯度更新公式为：

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$

但这里的learning rate η也随t和i而变：

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

其中 Gt 是个对角矩阵， (i,i) 元素就是 t 时刻参数 θi 的梯度平方和。

Adagrad 的**优点**是减少了学习率的手动调节。超参数设定值：一般η选取0.01。

缺点：它的缺点是分母会不断积累，这样学习率就会收缩并最终会变得非常小。

④Adadelta

这个算法是对 Adagrad 的改进，和Adagrad相比，就是分母的 G 换成了过去的梯度平方的衰减平均值，指数衰减平均值

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E\left[g^2\right]_t + \epsilon}} g_t$$

这个分母相当于梯度的均方根 root mean squared (RMS)，在数据统计分析中，将所有值平方求和，求其均值，再开平方，就得到均方根值，所以可以用 RMS 简写：

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t$$

其中 E 的计算公式如下，t 时刻的依赖于前一时刻的平均和当前的梯度：

$$E\left[g^2\right]_t = \gamma E\left[g^2\right]_{t-1} + (1-\gamma) g_t^2$$

梯度更新规则:此外，还将学习率η换成了RMS[Δθ]，这样的话，我们甚至都不需要提前设定学习率了：

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$
$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

超参数设定值: γ 一般设定为 0.9

⑤RMSprop

RMSprop 是 Geoff Hinton 提出的一种自适应学习率方法。RMSprop 和 Adadelta 都是为了解决 Adagrad 学习率急剧下降问题的。

梯度更新规则:RMSprop 与 Adadelta 的第一种形式相同：（使用的是指数加权平均，旨在消除梯度下降中的摆动，与Momentum的效果一样，某一维度的导数比较大，则指数加权平均就大，某一维度的导数比较小，则其指数加权平均就小，这样就保证了各维度导数都在一个量级，进而减少了摆动，允许使用一个更大的学习率η）。

$$E\left[g^2\right]_t = 0.9E\left[g^2\right]_{t-1} + 0.1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E\left[g^2\right]_t + \epsilon}} g_t$$

超参数设定值:Hinton 建议设定 γ 为 0.9, 学习率 η 为 0.001。

⑥Adam：Adaptive Moment Estimation

Adam 算法和传统的随机梯度下降不同。随机梯度下降保持单一的学习率（即 alpha）更新所有的权重，学习率在训练过程中并不会改变。而 Adam 通过计算梯度的一阶矩估计和二阶矩估计而为不同的参数设计独立的自适应性学习率。这个算法是另一种计算每个参数的自适应学习率的方法，相当于 RMSprop + Momentum。

除了像 Adadelta 和 RMSprop 一样存储了过去梯度的平方 vt 的指数衰减平均值 ，也像 momentum 一样保持了过去梯度 mt 的指数衰减平均值：

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

如果mt和vt被初始化为0向量，那它们就会向0偏置，所以做了偏差校正，通过计算偏差校正后的mt和vt来抵消这些偏差：

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

梯度更新规则:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

超参数设定值:建议 β1 ＝ 0.9，β2 ＝ 0.999，ϵ ＝ 10e−8。

实践表明，Adam 比其他适应性学习方法效果要好。

Adam 和 SGD 区别：Adam = Adaptive + Momentum，顾名思义Adam集成了SGD的一阶动量和RMSProp的二阶动量。

3.牛顿法

利用二阶导数，收敛速度快；但对目标函数有严格要求，必须有连续的一、二阶偏导数，计算量大。利用牛顿法求解目标函数的最小值其实是转化成求使目标函数的一阶导为0的参数值。这一转换的理论依据是，函数的极值点处的一阶导数为0.其迭代过程是在当前位置x0求该函数的切线，该切线和x轴的交点x1，作为新的x0,重复这个过程，直到交点和函数的零点重合。此时的参数值就是使得目标函数取得极值的参数值。

简述梯度下降法和牛顿法的优缺点？梯度下降法和牛顿法区别

1.牛顿法：是通过求解目标函数的一阶导数为0时的参数，进而求出目标函数最小值时的参数。①收敛速度很快。②海森矩阵的逆在迭代过程中不断减小，可以起到逐步减小步长的效果。③缺点：海森矩阵的逆计算复杂，代价比较大，因此有了拟牛顿法。

2.梯度下降法：是通过梯度方向和步长，直接求解目标函数的最小值时的参数。越接近最优值时，步长应该不断减小，否则会在最优值附近来回震荡。

Batch Normalization

1.批标准化(Batch Normalization)

可以理解为是一种数据预处理技术，使得每层网络的输入都服从（0，1）0均值，1方差分布，如果不进行BN，那么每次输入的数据分布不一致，网络训练精度自然也受影响。前向公式：

前向传播代码

```
def batchnorm_forward(x, gamma, beta, eps):

  N, D = x.shape
  #为了后向传播求导方便，这里都是分步进行的
  #step1: 计算均值
  mu = 1./N * np.sum(x, axis = 0)

  #step2: 减均值
  xmu = x - mu

```

```
    #step3: 计算方差
    sq = xmu ** 2
    var = 1./N * np.sum(sq, axis = 0)

    #step4: 计算x^的分母项
    sqrtvar = np.sqrt(var + eps)
    ivar = 1./sqrtvar

    #step5: normalization->x^
    xhat = xmu * ivar

    #step6: scale and shift
    gammax = gamma * xhat
    out = gammax + beta

    #存储中间变量
    cache =  (xhat,gamma,xmu,ivar,sqrtvar,var,eps)

    return out, cache
```

后向传播代码

```
def batchnorm_backward(dout, cache):

```

```
    #解压中间变量
    xhat,gamma,xmu,ivar,sqrtvar,var,eps = cache

    N,D = dout.shape

    #step6
    dbeta = np.sum(dout, axis=0)
    dgammax = dout
    dgamma = np.sum(dgammax*xhat, axis=0)
    dxhat = dgammax * gamma

    #step5
    divar = np.sum(dxhat*xmu, axis=0)
    dxmu1 = dxhat * ivar #注意这是xmu的一个支路

    #step4
    dsqrtvar = -1. /(sqrtvar**2) * divar
    dvar = 0.5 * 1. /np.sqrt(var+eps) * dsqrtvar

    #step3
    dsq = 1. /N * np.ones((N,D)) * dvar
    dxmu2 = 2 * xmu * dsq #注意这是xmu的第二个支路

    #step2
    dx1 = (dxmu1 + dxmu2) 注意这是x的一个支路

    #step1
```

```
    dmu = -1 * np.sum(dxmu1+dxmu2, axis=0)
    dx2 = 1. /N * np.ones((N,D)) * dmu 注意这是x的第二个支路

    #step0 done!
    dx = dx1 + dx2

    return dx, dgamma, dbeta
```

Batch Norm即批规范化，目的是为了解决每批数据训练时的不规则分布给训练造成的困难，对批数据进行规范化，还可以在梯度反传时，解决梯度消失的问题。

Batchnorm也是一种正则的方式，可以代替其他正则方式如dropout，但通过这样的正则化，也消融了数据之间的许多差异信息。

2. batchnorm的几个参数，可学习的参数有哪些？

第四步加了两个参数γ和β，分别叫做缩放参数和平移参数，通过选择不同的γ和β可以让隐藏单元有不同的分布。这里面的γ和β可以从你的模型中学习，可以用梯度下降，Adam等算法进行更新。

3. Batch Normalization的作用

神经网络在训练的时候随着网络层数的加深,激活函数的输入值的整体分布逐渐往激活函数的取值区间上下限靠近,从而导致在反向传播时低层的神经网络的梯度消失。而BatchNormalization的作用是**通过规范化的手段,将越来越偏的分布拉回到标准化的分布,使得激活函数的输入值落在激活函数对输入比较敏感的区域,从而使梯度变大,加快学习收敛速度,避免梯度消失的问题**。

①不仅仅极大提升了训练速度，收敛过程大大加快；②还能增加分类效果，一种解释是这是类似于Dropout的一种防止过拟合的正则化表达方式，所以不用Dropout也能达到相当的效果；③另外调参过程也简单多了，对于初始化要求没那么高，而且可以使用大的学习率等。

4. BN层怎么实现

1.计算样本均值。2.计算样本方差。3.样本数据标准化处理。4.进行平移和缩放处理。引入了γ和β两个参数。来训练γ和β两个参数。引入了这个可学习重构参数γ、β，让我们的网络可以学习恢复出原始网络所要学习的特征分布。

5.BN一般用在网络的哪个部分啊？

先卷积再BN

Batch normalization 的 batch 是批数据, 把数据分成小批小批进行 stochastic gradient descent. 而且在每批数据进行前向传递 forward propagation 的时候, 对每一层都进行 normalization 的处理

6.BN为什么要重构

恢复出原始的某一层所学到的特征的。因此我们引入了这个可学习重构参数γ、β，让我们的网络可以学习恢复出原始网络所要学习的特征分布。

7.BN层反向传播，怎么求导

反向传播：

反向传播需要计算三个梯度值，分别是

$$\frac{\partial \ell}{\partial x_i}, \frac{\partial \ell}{\partial y}, \frac{\partial \ell}{\partial \beta}。$$

定义

$$\frac{\partial \ell}{\partial y_i}$$

为从上一层传递过来的残差。

$$\text{计算 } \frac{\partial \ell}{\partial Y} \text{ : } \quad \frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \overline{x_i}.$$

$$\text{计算 } \frac{\partial \ell}{\partial \beta} \text{ : } \quad \frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

$$\text{下面计算 } \frac{\partial \ell}{\partial x_i} .$$

观察缩放和移位与归一化公式，可以看到从xi到yi的链式计算过程：

$$\text{有} \quad x_i \gg \mu\mathcal{B} \gg \sigma_{\mathcal{B}}^2 \gg \overline{x_i} \gg y_i$$

$$\text{同时 } x_i \gg \mu\mathcal{B} \gg \overline{x_i} \gg y_i \text{ .}$$

$$\text{同时 } x_i \gg \overline{x_i} \gg y_i \text{ .}$$

$$\text{则} \quad \frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial y_i} \cdot \frac{\partial y_i}{\partial \overline{x_i}} \left(\frac{\partial \overline{x_i}}{\partial x_i} + \frac{\partial \overline{x_i}}{\partial \mu\mathcal{B}} \cdot \frac{\partial \mu\mathcal{B}}{\partial x_i} + \frac{\partial \overline{x_i}}{\partial \sigma_B^2} \cdot \frac{\partial \sigma_B^2}{\partial \mu\mathcal{B}} \cdot \frac{\partial \mu\mathcal{B}}{\partial x_i} \right) .$$

上式三个加号对应三条链式计算。

8. batchnorm训练时和测试时的区别

训练阶段：首先计算均值和方差（每次训练给一个批量，计算批量的均值方差），然后归一化，然后缩放和平移。

测试阶段：每次只输入一张图片，这怎么计算批量的均值和方差，于是，就有了代码中下面两行，在训练的时候实现计算好mean、 var，测试的时候直接拿来用就可以了，不用计算均值和方差。

9.先加BN还是激活，有什么区别（先激活）

目前在实践上，倾向于把BN放在ReLU后面。也有评测表明BN放ReLU后面效果更好。

二、基础卷积神经网络

1.CNN的经典模型

LeNet，AlexNet，VGG，GoogLeNet，ResNet，DenseNet

2.对CNN的理解

CNN＝ 数据输入层 (Input Layer)＋ {[卷积计算层（CONV Layer)＊ a ＋ReLU激励层 (ReLU Layer)]＊ b ＋ 池化层 (Pooling Layer) } * c ＋全连接层 (FC Layer) ＊ d 。

3.CNN和传统的全连接神经网络有什么区别？

在全连接神经网络中，每相邻两层之间的节点都有边相连，于是会将每一层的全连接层中的节点组织成一列，这样方便显示连接结构。而对于卷积神经网络，相邻两层之间只有部分节点相连，为了展示每一层神经元的维度，一般会将每一层卷积层的节点组织成一个三维矩阵。全连接神经网络和卷积神经网络的唯一区别就是神经网络相邻两层的连接方式。

4.讲一下CNN，每个层及作用

卷积层：用它来进行特征提取

池化层：对输入的特征图进行压缩，一方面使特征图变小，简化网络计算复杂度；一方面进行特征压缩，提取主要特征，

激活函数：是用来加入非线性因素的，因为线性模型的表达能力不够。

全连接层（fully connected layers，FC）在整个卷积神经网络中起到“分类器”的作用。全连接层则起到将学到的“分布式特征表示”映射到样本标记空间的作用。

5.为什么神经网络使用卷积层？-共享参数，局部连接；

使用卷积层的前提条件是什么？-数据分布一致

6.resnet相比于之前的卷积神经网络模型中，最大的改进点是什么？，解决了什么问题

跳跃连接(residual block)和瓶颈层。resnet本身是一种拟合残差的结果，让网络学习任务更简单，可以有效地解决梯度弥散的问题。

Resnet为啥能解决梯度消失，怎么做的，能推导吗？

由于每做一次卷积（包括对应的激活操作）都会浪费掉一些信息：比如卷积核参数的随机性（盲目性）、激活函数的抑制作用等等。这时，ResNet中的shortcut相当于把以前处理过的信息直接再拿到现在一并处理，起到了减损的效果。

7.resnet第二个版本做了哪些改进，Resnet性能最好的变体是哪个，结构是怎么样的，原理是什么？

Resnetv2：1、相比于原始的网络结构，先激活的网络中f是恒等变换，这使得模型优化更加容易。2、使用了先激活输入的网络，能够减少网络过拟合。

Resnet性能最好的变体是Resnext。

ResNeXt可以说是基于Resnet与Inception 'Split + Transfrom + Concat'而搞出的产物，结构简单、易懂又足够强大。（Inception网络使用了一种split-transform-merge思想，即先将输入切分到不同的低维度中，然后做一个特征映射，最后将结果融合到一起。但模型的泛化性不好，针对不同的任务需要设计的东西太多。）

ResNeXt提出了一个基数（cardinatity）的概念，用于作为模型复杂度的另外一个度量。基数（cardinatity）指的是一个block中所具有的相同分支的数目。

与 ResNet 相比，相同的参数个数，结果更好：一个 101 层的 ResNeXt 网络，和 200 层的 ResNet 准确度差不多，但是计算量只有后者的一半。

ResNet的特点 引入跳跃连接，有效地解决了网络过深时候梯度消失的问题，使得设计更深层次的网络变得可行。

8.简述InceptionV1到V4的网络、区别、改进

Inceptionv1的核心就是把googlenet的某一些大的卷积层换成1*1*, *3*3, 5*5的小卷积，这样能够大大的减小权值参数数量。

inception V2在输入的时候增加了batch_normal，所以他的论文名字也是叫batch_normal，加了这个以后训练起来收敛更快，学习起来自然更高效，可以减少dropout的使用。

inception V3把googlenet里一些*7*7的卷积变成了1*7*和7*1的两层串联，*3*3的也一样，变成了1*3和*31，这样加速了计算，还增加了网络的非线性，减小过拟合的概率。另外，网络的输入从224改成了299.

inception v4实际上是把原来的inception加上了resnet的方法，从一个节点能够跳过一些节点直接连入之后的一些节点，并且残差也跟着过去一个。另外就是V4把一个先1*1再*3*3*那步换成了先3*3再*1*1*.

论文说引入resnet不是用来提高深度，进而提高准确度的，只是用来提高速度的。

9. DenseNet为什么比ResNet有更强的表达能力？

DenseNet在增加深度的同时，加宽每一个DenseBlock的网络宽度，能够增加网络识别特征的能力，而且由于DenseBlock的横向结构类似 Inception block的结构，使得需要计算的参数量大大降低。

三、损失函数

1.说一下smooth L1 Loss,并阐述使用smooth L1 Loss的优点

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

Smooth L1的优点：①相比于L1损失函数，可以收敛得更快。②相比于L2损失函数，对离群点、异常值不敏感，梯度变化相对更小，训练时不容易跑飞。

2. L1_loss和L2_loss的区别

平均绝对误差(L1 Loss): 平均绝对误差（Mean Absolute Error,MAE) 是指模型预测值f(x)和真实值y之间距离的平均值，其公式如下：

$$MAE = \frac{\sum_{n=1}^{n} |f(x_i) - y_i|}{n}$$

均方误差MSE (L2 Loss):均方误差（Mean Square Error,MSE）是模型预测值f(x) 与真实样本值y 之间差值平方的平均值，其公式如下

$$MSE = \frac{\sum_{i=1}^{n} (f_{x_i} - y_i)^2}{n}$$

3.为何分类问题用交叉熵而不用平方损失？啥是交叉熵

$$L = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$$

1.用平方误差损失函数，误差增大参数的梯度会增大，但是当误差很大时，参数的梯度就会又减小了。2.使用交叉熵损失是函数，误差越大参数的梯度也越大，能够快速收敛。

分类中为什么交叉熵损失函数比均方误差损失函数更常用？

交叉熵损失函数关于输入权重的梯度表达式与预测值与真实值的误差成正比且不含激活函数的梯度，而均方误差损失函数关于输入权重的梯度表达式中则含有，由于常用的sigmoid/tanh等激活函数存在梯度饱和区，使得MSE对权重的梯度会很小，参数w调整的慢，训练也慢，而交叉熵损失函数则不会出现此问题，其参数w会根据误差调整，训练更快，效果更好。

4.一张图片多个类别怎么设计损失函数，多标签分类问题

多标签分类怎么解决，从损失函数角度考虑

分类问题名称　输出层使用激活函数 对应的损失函数

二分类 sigmoid函数 二分类交叉熵损失函数（binary_crossentropy）

多分类 Softmax函数 多类别交叉熵损失函数（categorical_crossentropy）

多标签分类 sigmoid函数 二分类交叉熵损失函数（binary_crossentropy）

(多标签问题与二分类问题关系在上文已经讨论过了，方法是计算一个样本各个标签的损失（输出层采用sigmoid函数），然后取平均值。把一个多标签问题，转化为了在每个标签上的二分类问题。)

5. LR的损失函数？它的导数是啥？加了正则化之后它的导数又是啥？

Logistic regression （逻辑回归）是当前业界比较常用的机器学习方法。

Logistic回归虽然名字里带“回归”，但是它实际上是一种分类方法，主要用于两分类问题，利用Logistic函数（或称为Sigmoid函数），自变量取值范围为(-INF, INF)，自变量的取值范围为(0,1)，函数形式为：

$$g(z) = \frac{1}{1 + e^{-z}}$$

LR的损失函数为交叉熵损失函数。

参考文献