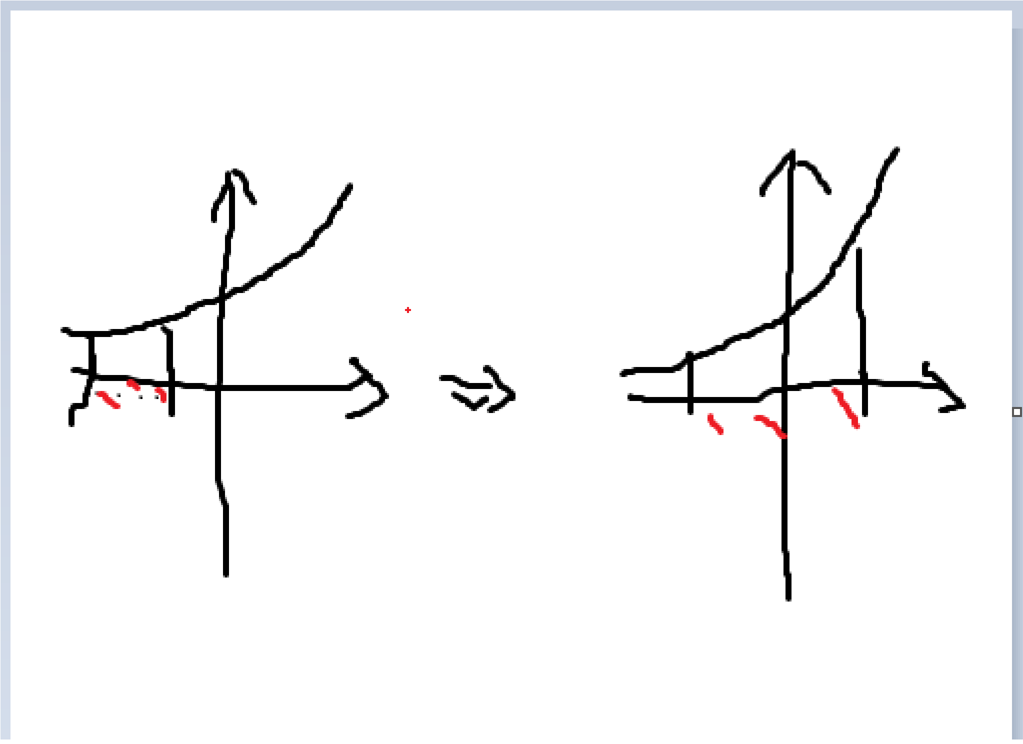
实验2_2BN和正则项

201900161098 马田慧 智能19

Batch Normalization

目的：通过批量归一化将数据分布到一个相对分布比较好的区间，使实验函数发挥表现效果

计算均值和方差通过同时对同一个batch里面进行归一化，方便同一个batch数据分布更有利于函数表现的区间上：



如图红色表示数据分布，左边的分布转化到右边的分布区间，上面表示函数，区间变化之后对应的函数值变化更明显，也就更有利于训练。

其中公式为：

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

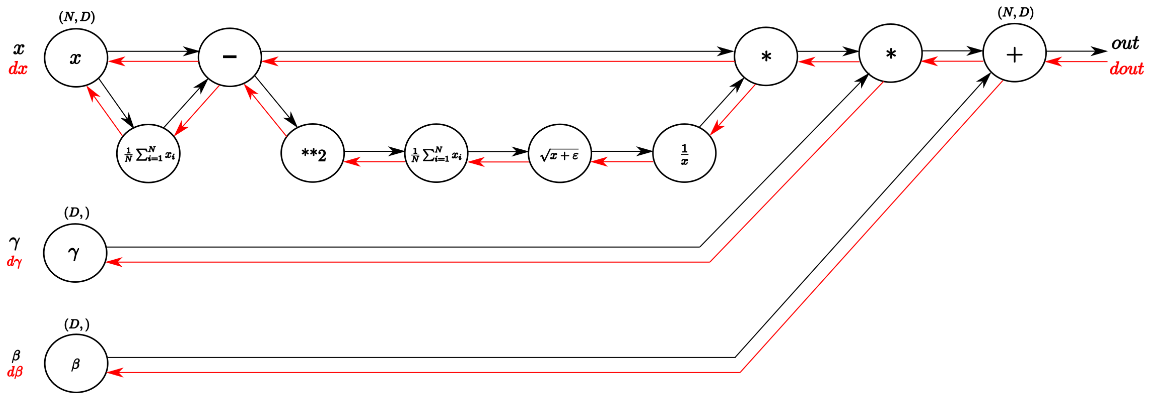
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

分母的ϵ 防止除以0;

具体求解时使用计算图：黑色表示正向，红色表示反向，反向时，每一个单独圆圈表示一个函数计算来看:



详细过程:

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

  N, D = x.shape

  #step1: calculate mean
  mu = 1./N * np.sum(x, axis = 0)

  #step2: subtract mean vector of every trainings example
  xmu = x - mu

  #step3: following the lower branch - calculation denominator
  sq = xmu ** 2

  #step4: calculate variance
```

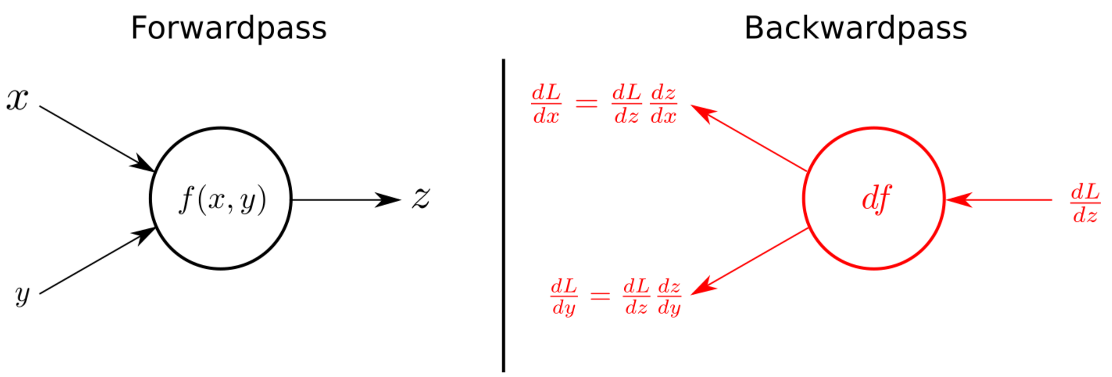
```
    var = 1./N * np.sum(sq, axis = 0)

    #step5: add eps for numerical stability, then sqrt
    sqrtvar = np.sqrt(var + eps)

    #step6: invert sqrtwar
    ivar = 1./sqrtvar

    #step7: execute normalization
    xhat = xmu * ivar

    #step8: Nor the two transformation steps
    gammax = gamma * xhat

    #step9
    out = gammax + beta

    #store intermediate
    cache = (xhat,gamma,xmu,ivar,sqrtvar,var,eps)

    return out, cache
test:
        x_hat=(x-running_mean)/np.sqrt(running_var+eps)
```

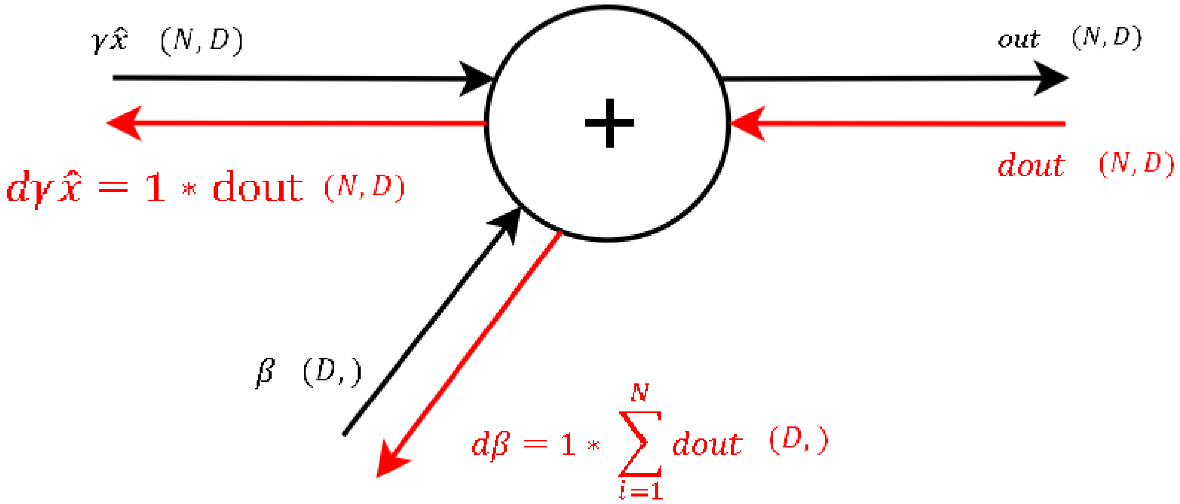
精简表示:

```
mean=np.mean(x)
var=np.var(x)
xmu=x-mean
std=np.sqrt(var+eps)
istd=1./std
x_hat=xmu/std
y=gamma*x_hat+beta
running_mean = momentum * running_mean + (1 - momentum) * mean#用于预测时候使用
running_var = momentum * running_var + (1 - momentum) * var
out=y
cache=(xmu,std,x_hat,gamma,istd)
```

γ和 β的作用：是新的学习参数，原本的是表示下一层的结构，而这里是可以学习的参数，均值仅由beta来决定;

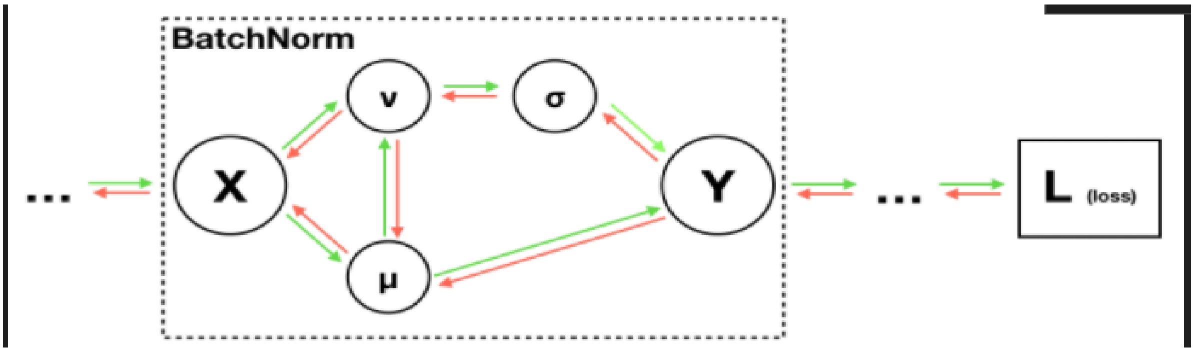
理解反向传播:





画出计算图每一步用正向和反向的箭头标好对应变量名称，然后再写出对应的维度，实际计算用到矩阵时，基本根据维度来计算，比如这里的beta维度是D，而后面的out维度是NxD，就对于dout的D维度求和就得到beta的梯度；

```
xmu,std,x_hat,gamma,istd=cache#x,mean,var,std,eps,x_hat,gamma,beta

N,D=xmu.shape
dgamma=np.sum(dout*x_hat,0)
dbeta=np.sum(dout,0)

dx_hat=dout*gamma

# std=np.sqrt(var+eps)
# istd=1./std
distd=np.sum(dx_hat*xmu,0)
dstd=-1/std**2*distd
dvar=1/(2*std)*dstd
dxmusqu=1/N*dvar*np.ones_like(xmu)
dmu1=2*xmu*dxmusqu

dmu2=istd*dx_hat

dmu=dmu1+dmu2
du=-1*np.sum(dmu,0)
dx1=1/N*du*np.ones_like(xmu)

dx2=dmu
dx=dx1+dx2
```



将内部整合成一个函数：**注意沿着某个维度求和或扩大到某个矩阵不能直接简单相乘，否则得到的数值不一样**

```
xmu,std,x_hat,gamma,istd=cache#x,mean,var,std,eps,x_hat,gamma,beta
dgamma=np.sum(x_hat*dout,0)
dbeta=np.sum(dout,0)
N,D=xmu.shape
dxmu=gamma*dout*istd+np.sum(xmu*gamma*dout,0)*(-1/std**3)*np.ones_like(xmu)/N*xmu
dx1=np.sum(-dxmu,0)/N*np.ones_like(xmu)
dx2=dxmu
dx=dx1+dx2
```

ref:https://kratzert.github.io/2016/02/12/understanding-the-gradient-flow-through-the-batch-normalization-layer.html

Regulazation

Regularization will help you reduce overfitting.

- Regularization will drive your weights to lower values.

- L2 regularization and Dropout are two very effective regularization techniques.

```
L2_regularization_cost =
(np.sum(np.square(W1))+np.sum(np.square(W2))+np.sum(np.square(W3)))*lambd/(2*m)
cost = cross_entropy_cost + L2_regularization_cost
```

使用l2正则化对于每一个参数都求解loss；

使用dropout随机使某些X变成0，同时在反向传播时同样的使用mask；

总结

通过本次实验认识到ML实验时的技巧，方便模型的训练，同时这两方面也会不断有新的研究出来，可能会出现优的策略。