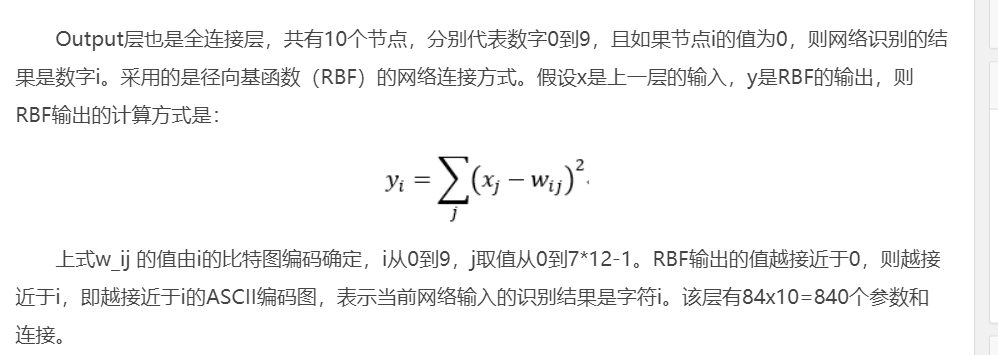
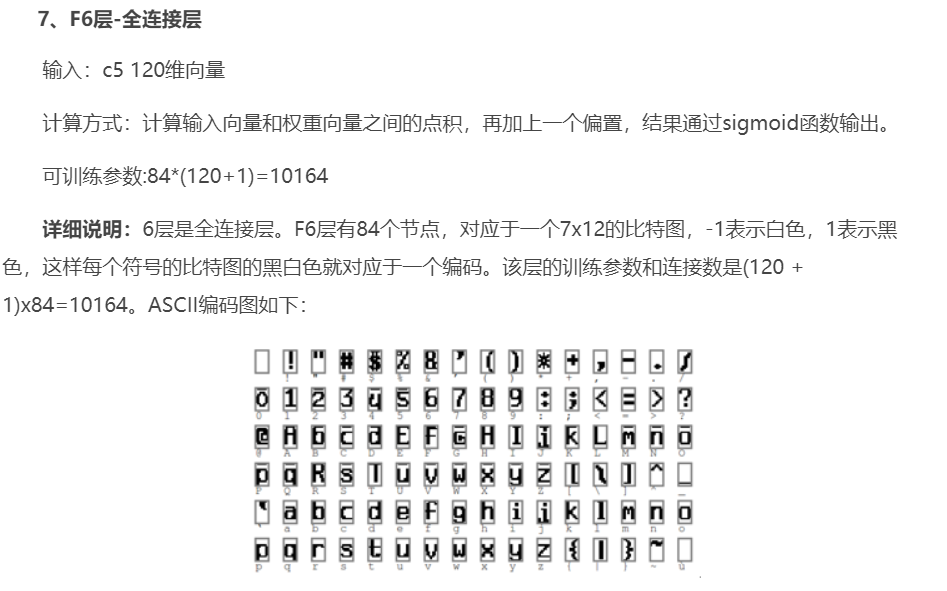
## LeNet 小问题

LeNet的输出层



最后一层wij就代表着真正的某个手写数字的特征向量

上一层是一个84的FC



噪声等级问题：

例如 40的高斯噪声

Mean=0 std=40/255

理解RNN LSTM GRU：

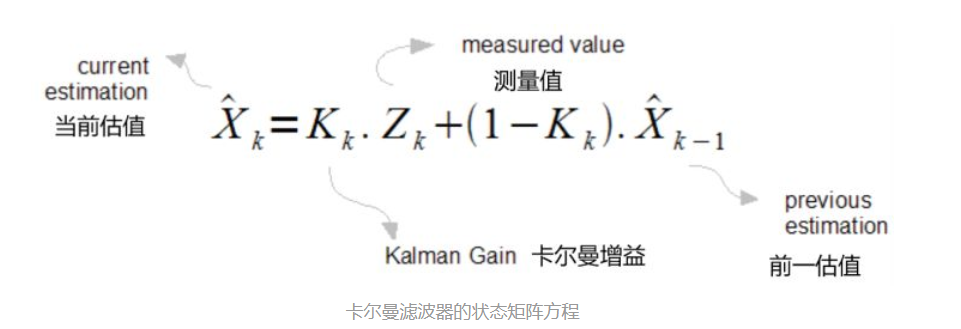
<https://blog.csdn.net/v_JULY_v/article/details/89894058> RNN LSTM

<https://blog.csdn.net/u012223913/article/details/77724621> GRU

### 卡尔曼滤波

结合上一时刻的最优预测值和当前时刻的观测值结合预测和观察噪声来获得当前时刻的最优估计

1. 预测不可靠
2. 观测也不一定可靠



<https://zhuanlan.zhihu.com/p/77327349>

### Batch size 对于训练的影响

（不考虑BN层）batch size 过大花费时间多，同时梯度震荡严重（相邻俩个batch 之间的差距较大）,不利于收敛；batch size 过大，不同batch的梯度方向没有任何变化(相邻batch 之间的差距较小),容易陷入局部极小值

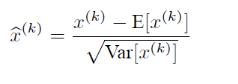
## Batch Normaliation

注： 对小batch 效果不好 因为batch较小时 mean 和σ不具有代表性

我们知道在神经网络训练开始前，都要对输入数据做一个归一化处理，那么具体为什么需要归一化呢？归一化后有什么好处呢？原因在于神经网络学习过程本质就是为了学习数据分布，一旦训练数据与测试数据的分布不同，那么网络的泛化能力也大大降低；另外一方面，一旦每批训练数据的分布各不相同(batch 梯度下降)，那么网络就要在每次迭代都去学习适应不同的分布，这样将会大大降低网络的训练速度，这也正是为什么我们需要对数据都要做一个归一化预处理的原因。

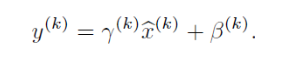
对于深度网络的训练是一个复杂的过程，只要网络的前面几层发生微小的改变，那么后面几层就会被累积放大下去。一旦网络某一层的输入数据的分布发生改变，那么这一层网络就需要去适应学习这个新的数据分布，所以如果训练过程中，训练数据的分布一直在发生变化，那么将会影响网络的训练速度。

我们知道网络一旦train起来，那么参数就要发生更新，除了输入层的数据外(因为输入层数据，我们已经人为的为每个样本归一化)，后面网络每一层的输入数据分布是一直在发生变化的，因为在训练的时候，前面层训练参数的更新将导致后面层输入数据分布的变化。以网络第二层为例：网络的第二层输入，是由第一层的参数和input计算得到的，而第一层的参数在整个训练过程中一直在变化，因此必然会引起后面每一层输入数据分布的改变。我们把网络中间层在训练过程中，数据分布的改变称之为：“Internal  Covariate Shift”。Paper所提出的算法，就是要解决在训练过程中，中间层数据分布发生改变的情况，于是就有了Batch  Normalization，这个牛逼算法的诞生。



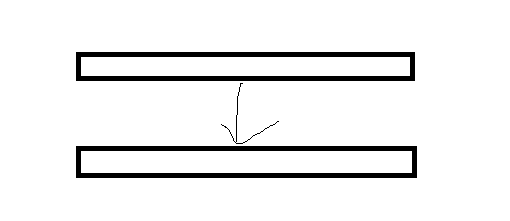
公式简单粗糙，但是依旧很牛逼。因此后面我们也将用这个公式，对某一个层网络的输入数据做一个归一化处理。需要注意的是，我们训练过程中采用batch 随机梯度下降，上面的E(xk)指的是每一批训练数据神经元xk的平均值；然后分母就是每一批数据神经元xk激活度的一个标准差了

好像很简单的样子，不就是在网络中间层数据做一个归一化处理嘛，这么简单的想法，为什么之前没人用呢？然而其实实现起来并不是那么简单的。其实如果是仅仅使用上面的归一化公式，对网络某一层A的输出数据做归一化，然后送入网络下一层B，这样是会影响到本层网络A所学习到的特征的。打个比方，比如我网络中间某一层学习到特征数据本身就分布在S型激活函数的两侧，你强制把它给我归一化处理、标准差也限制在了1，把数据变换成分布于s函数的中间部分，这样就相当于我这一层网络所学习到的特征分布被你搞坏了，这可怎么办？于是文献使出了一招惊天地泣鬼神的招式：变换重构，引入了可学习参数γ、β，这就是算法关键之处：

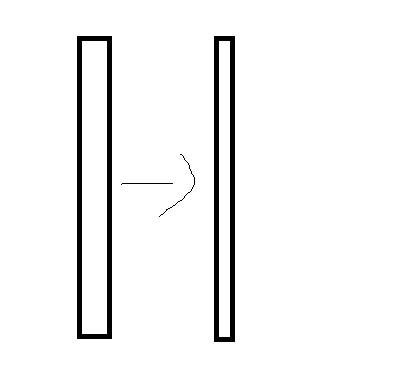


## np.vstack,np.hstack

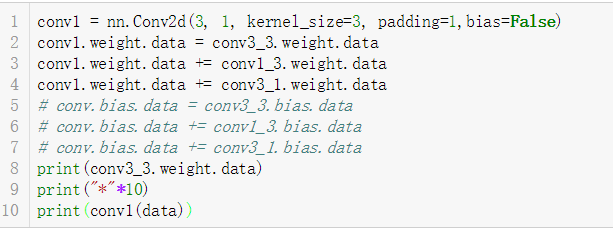
vstack: 一行行竖着垒



hstack 一列列横着垒



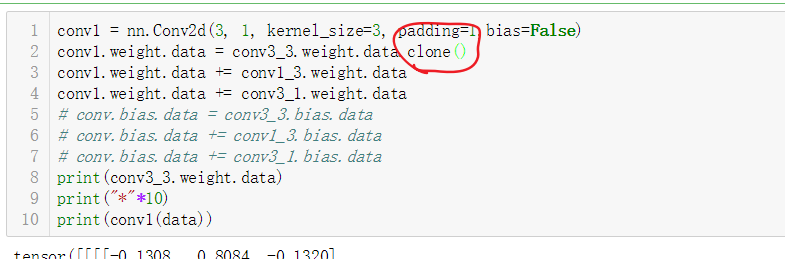
## Pytorch 深坑



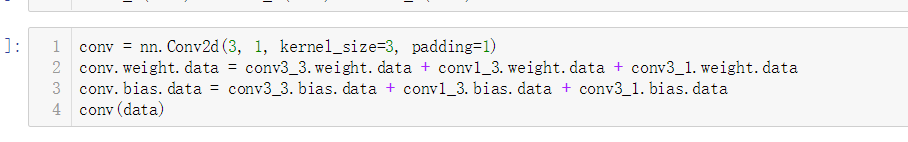
不要这样改weight 有蜜汁问题

使用下列写法

1.



2.



破案了：tensor 是一个引用类型 所以需要使用clone