<https://blog.csdn.net/mounty_fsc/article/details/51085654>

**基本类Blob，Layer，Net**

# [caffe中权值初始化方法](https://www.cnblogs.com/tianshifu/p/6165809.html)

首先说明：在caffe/include/caffe中的 filer.hpp文件中有它的源文件，如果想看，可以看看哦，反正我是不想看，代码细节吧，现在不想知道太多，有个宏观的idea就可以啦，如果想看代码的具体的话，可以看：<http://blog.csdn.net/xizero00/article/details/50921692>，写的还是很不错的（不过有的地方的备注不对，不知道改过来了没）。

文件 filler.hpp提供了7种权值初始化的方法，分别为：常量初始化（constant）、高斯分布初始化（gaussian）、positive\_unitball初始化、均匀分布初始化（uniform）、xavier初始化、msra初始化、双线性初始化（bilinear）。

275 Filler<Dtype>\* GetFiller(const FillerParameter& param) {

276 const std::string& type = param.type();

277 if (type == "constant") {

278 return new ConstantFiller<Dtype>(param);

279 } else if (type == "gaussian") {

280 return new GaussianFiller<Dtype>(param);

281 } else if (type == "positive\_unitball") {

282 return new PositiveUnitballFiller<Dtype>(param);

283 } else if (type == "uniform") {

284 return new UniformFiller<Dtype>(param);

285 } else if (type == "xavier") {

286 return new XavierFiller<Dtype>(param);

287 } else if (type == "msra") {

288 return new MSRAFiller<Dtype>(param);

289 } else if (type == "bilinear") {

290 return new BilinearFiller<Dtype>(param);

291 } else {

292 CHECK(false) << "Unknown filler name: " << param.type();

293 }

294 return (Filler<Dtype>\*)(NULL);

295 }

并且结合 .prototxt 文件中的 FillerParameter来看看怎么用：

43 message FillerParameter {

44 // The filler type.

45 optional string type = 1 [default = 'constant'];

46 optional float value = http://www.cnblogs.com/yinheyi/p/2 [default = 0]; // the value in constant filler

47 optional float min = 3 [default = 0]; // the min value in uniform filler

48 optional float max = 4 [default = 1]; // the max value in uniform filler

49 optional float mean = 5 [default = 0]; // the mean value in Gaussian filler

50 optional float std = 6 [default = 1]; // the std value in Gaussian filler

51 // The expected number of non-zero output weights for a given input in

52 // Gaussian filler -- the default -1 means don't perform sparsification.

53 optional int32 sparse = 7 [default = -1];

54 // Normalize the filler variance by fan\_in, fan\_out, or their average.

55 // Applies to 'xavier' and 'msra' fillers.

56 enum VarianceNorm {

57 FAN\_IN = 0;

58 FAN\_OUT = 1;

59 AVERAGE = 2;

60 }

61 optional VarianceNorm variance\_norm = 8 [default = FAN\_IN];

62 }

### constant初始化方法：

它就是把权值或着偏置初始化为一个常数，具体是什么常数，自己可以定义啦。它的值等于上面的.prototxt文件中的 value 的值，默认为0

下面是是与之相关的.proto文件里的定义，在定义网络时，可能分用到这些参数。

45 optional string type = 1 [default = 'constant'];

46 optional float value = http://www.cnblogs.com/yinheyi/p/2 [default = 0]; // the value in constant filler

### uniform初始化方法

它的作用就是把权值与偏置进行 均匀分布的初始化。用min 与 max 来控制它们的的上下限，默认为（0，1）.

下面是是与之相关的.proto文件里的定义，在定义网络时，可能分用到这些参数。

45 optional string type = 1 [default = 'constant'];

47 optional float min = 3 [default = 0]; // the min value in uniform filler

48 optional float max = 4 [default = 1]; // the max value in uniform filler

### Gaussian 初始化

给定高斯函数的均值与标准差,然后呢？生成高斯分布就可以了。

不过要说明一点的就是， gaussina初始化可以进行 sparse，意思就是可以把一些权值设为0. 控制它的用参数 sparse. sparse表示相对于 num\_output来说非0的个数，在代码实现中，会把 sparse/num\_output 作为 bernoulli分布的概率，明白？？ 生成的bernoulli分布的数字（为0或1）与原来的权值相乖，就可以实现一部分权值为0了。即然这样，我有一点不明白，为什么不直接把sparsr定义成概率呢？？这样多么简单啦，并且好明白啊。。对于 num\_output是什么，你在定义你的网络的.prototxt里，一定分有的啦，不信你去看看；

下面是是与之相关的.proto文件里的定义，在定义网络时，可能分用到这些参数。

45 optional string type = 1 [default = 'constant'];

49 optional float mean = 5 [default = 0]; // the mean value in Gaussian filler

50 optional float std = 6 [default = 1]; // the std value in Gaussian filler

51 // The expected number of non-zero output weights for a given input in

52 // Gaussian filler -- the default -1 means don't perform sparsification.

53 optional int32 sparse = 7 [default = -1];

### positive\_unitball 初始化

通俗一点，它干了点什么呢？即让每一个单元的输入的权值的和为 1. 例如吧，一个神经元有100个输入，这样的话，让这100个输入的权值的和为1. 源码中怎么实现的呢？ 首先给这100个权值赋值为在（0，1）之间的均匀分布，然后，每一个权值再除以它们的和就可以啦。

感觉这么做，可以有助于防止权值初始化过大，使激活函数（sigmoid函数）进入饱和区。所以呢，它应该比适合simgmoid形的激活函数。

它不需要参数去 控制。

### XavierFiller初始化：

对于这个初始化的方法，是有理论的。它来自这篇论文[《Understanding the difficulty of training deep feedforward neural networks》](http://machinelearning.wustl.edu/mlpapers/paper_files/AISTATS2010_GlorotB10.pdf)。在推导过程中，我们认为处于 tanh激活函数的线性区，所以呢，对于ReLU激活函数来说，XavierFiller初始化也是很适合啦。

如果不想看论文的话，可以看看 <https://zhuanlan.zhihu.com/p/22028079>，我觉得写的很棒，另外，<http://blog.csdn.net/shuzfan/article/details/51338178>可以作为补充。

它的思想就是让一个神经元的输入权重的（当反向传播时，就变为输出了）的方差等于：1 / 输入的个数；这样做的目的就是可以让信息可以在网络中均匀的分布一下。

对于权值的分布：是一个让均值为0，方差为1 / 输入的个数 的 均匀分布。

如果我们更注重前向传播的话，我们可以选择 fan\_in，即正向传播的输入个数；如果更注重后向传播的话，我们选择 fan\_out, 因为吧，等着反向传播的时候，fan\_out就是神经元的输入个数；如果两者都考虑的话，那就选  average = (fan\_in + fan\_out) /2

下面是是与之相关的.proto文件里的定义，在定义网络时，可能分用到这些参数。

45 optional string type = 1 [default = 'constant'];

54 // Normalize the filler variance by fan\_in, fan\_out, or their average.

55 //[www.90168.org](http://www.90168.org/) Applies to 'xavier' and 'msra' fillers.

56 enum VarianceNorm {

57 FAN\_IN = 0;

58 FAN\_OUT = 1;

59 AVERAGE = 2;

60 }

61 optional VarianceNorm variance\_norm = 8 [default = FAN\_IN];

### MSRAFiller初始化方式

它与上面基本类似，它是基于[《Delving Deep into Rectifiers:Surpassing Human-Level Performance on ImageNet Classification》](http://arxiv.org/pdf/1502.01852.pdf)来推导的，并且呢，它是基于激活函数为 ReLU函数哦，

对于权值的分布，是基于均值为0，方差为 2 /输入的个数 的高斯分布，这也是和上面的Xavier Filler不同的地方；它特别适合激活函数为 ReLU函数的啦。

下面是是与之相关的.proto文件里的定义，在定义网络时，可能分用到这些参数。

45 optional string type = 1 [default = 'constant'];

54 // Normalize the filler variance by fan\_in, fan\_out, or their average.

55 // Applies to 'xavier' and 'msra' fillers.

56 enum VarianceNorm {

57 FAN\_IN = 0;

58 FAN\_OUT = 1;

59 AVERAGE = 2;

60 }

61 optional VarianceNorm variance\_norm = 8 [default = FAN\_IN];

### BilinearFiller初始化

对于它，要还没有怎么用到过，它常用在反卷积神经网络里的权值初始化；

直接上源码，大家看看吧；

213 /\*!

214 @brief Fills a Blob with coefficients for bilinear interpolation.

215

216 A common use case is with the DeconvolutionLayer acting as upsampling.

217 You can upsample a feature map with shape of (B, C, H, W) by any integer factor

218 using the following proto.

219 \code

220 layer {

221 name: "upsample", type: "Deconvolution"

222 bottom: "{{bottom\_name}}" top: "{{top\_name}}"

223 convolution\_param {

224 kernel\_size: {{2 \* factor - factor % 2}} stride: {{factor}}

225 num\_output: {{C}} group: {{C}}

226 pad: {{ceil((factor - 1)[www.90168.org](http://www.90168.org/) / 2.)}}

227 weight\_filler: { type: "bilinear" } bias\_term: false

228 }

229 param { lr\_mult: 0 decay\_mult: 0 }

230 }

231 \endcode

232 Please use this by replacing `{{}}` with your values. By specifying

233 `num\_output: {{C}} group: {{C}}`, it behaves as

234 channel-wise convolution. The filter shape of this deconvolution layer will be

235 (C, 1, K, K) where K is `kernel\_size`, and this filler will set a (K, K)

236 interpolation kernel for every channel of the filter identically. The resulting

237 shape of the top feature map will be (B, C, factor \* H, factor \* W).

238 Note that the learning rate and the

239 weight decay are set to 0 in order to keep coefficient values of bilinear

240 interpolation unchanged during training. If you apply this to an image, this

241 operation is equivalent to the following call in Python with Scikit.Image.

242 \code{.py}

243 out = skimage.transform.rescale(img, factor, mode='constant', cval=0)

244 \endcode

245 \*/

246 template <typename Dtype>

247 class BilinearFiller : public Filler<Dtype> {

248 public:

249 explicit BilinearFiller(const FillerParameter& param)

250 : Filler<Dtype>(param) {}

251 virtual void Fill(Blob<Dtype>\* blob) {

252 CHECK\_EQ(blob->num\_axes(), 4) << "Blob must be 4 dim.";

253 CHECK\_EQ(blob->width(), blob->height()) << "Filter must be square";

254 Dtype\* data = http://www.cnblogs.com/yinheyi/p/blob->mutable\_cpu\_data();

255 int f = ceil(blob->width() / 2.);

256 float c = (2 \* f - 1 - f % 2) / (2. \* f);

257 for (int i = 0; i < blob->count(); ++i) {

258 float x = i % blob->width();

259 float y = (i / blob->width()) % blob->height();

260 data[i] = (1 - fabs(x / f - c)) \* (1 - fabs(y / f - c));

261 }

262 CHECK\_EQ(this->filler\_param\_.sparse(), -1)

263 <<"Sparsity not supported by this Filler.";

264 }

265 };