Grégoire Montavon
Geneviève B. Orr
Klaus-Robert Müller (Eds.)

Neural Networks: Tricks of the Trade

Second Edition

深度学习框架Caffe学习与应用 第10课

本节课内容

- 准备：
  1. 数据集准备和扩增
  2. 图像预处理
  3. 参数初始化
- 训练：
  4. 卷积层参数tricks
  5. 池化层参数tricks
  6. 学习率
  7.正则化：预防过拟合
- 训练结果图像分析：
  8.观察损失曲线：学习率
  9.放大损失曲线：学习率、batch 大小
  10.准确率曲线
- 实践：
  11.Fine-tuning方法
  12. 模型集成

1. 数据准备与扩增

- 1.1 数据准备：

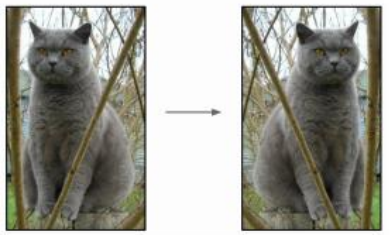
  一般数据集可能不会给出验证集，所以自己会从给的训练集中按照一定比例（9：1）分离出验证集。

- 1.2 数据的扩增

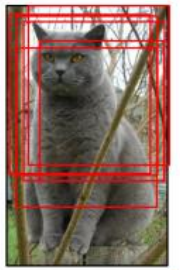
  因为深度网络需要在大规模的训练图像上来满足性能，所以当原始图像中的训练数据集规模不够多时，较好的办法是扩增数据来提升模型性能。换言之，数据扩增对于训练深度网络来说是必须的。

  常用的方法：

  1. 沿着x轴将图片左右翻转
  2. 随机的剪切、缩放、旋转
  3. 颜色抖动



Flip horizontally



Random crops/scales

具体包括反转、平移、缩放、亮度变化、裁剪、光照等外部影响、颜色变换、模糊、灰度等方法，这些是比较常见的图像处理方法。

Color jittering

Random mix/combinations of :
- translation
- rotation
- stretching
- shearing,
- lens distortions, … (go crazy)

- 4. 提高图像中像素的饱和度和值（即 HSV 颜色空间的 S 和 V 成分）到 0.250.25 和 44 之间（在一个样本图像内要保证各个像素该值是一样的），再在图像上加上一个范围在 [−0.1,0.1][−0.1,0.1] 之间的值给 H （ hue ，即 HSV 中的色调）这个成分。

  5. 用pca来改变RGB的强度值，产生分别对应的特征值和特征向量，然后用均值为0方差为0.1的随机数与特征值和特征向量相乘得到新的数据。（《*ImageNet Classification with Deep Convolutional Neural Networks*》）

Fancy PCA:
Consist of altering the intensities of the RGB channels.

1. Compute PCA on all [R,G, B] points values in the training data;

2. Sample some color offset along the principal components at each forward pass;

3. Add the offset to all pixels in a training image.

$$I_{xy} = [I_{xy}^R, I_{xy}^G, I_{xy}^B]^T \qquad [\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T$$

This scheme reduces the top-1 error rate by over 1%.

Fancy PCA 算法流程

2. 图像预处理

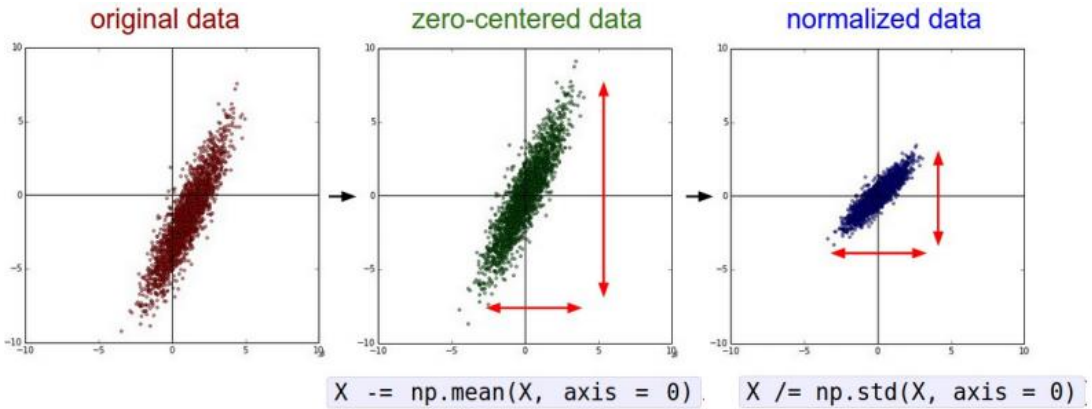
- 常见的是减均值、除方差，还有变化到-1～1，主要针对不同尺度的特征，进行尺度变换normaliz。

常用的预处理方法：

1. 去均值和规范化

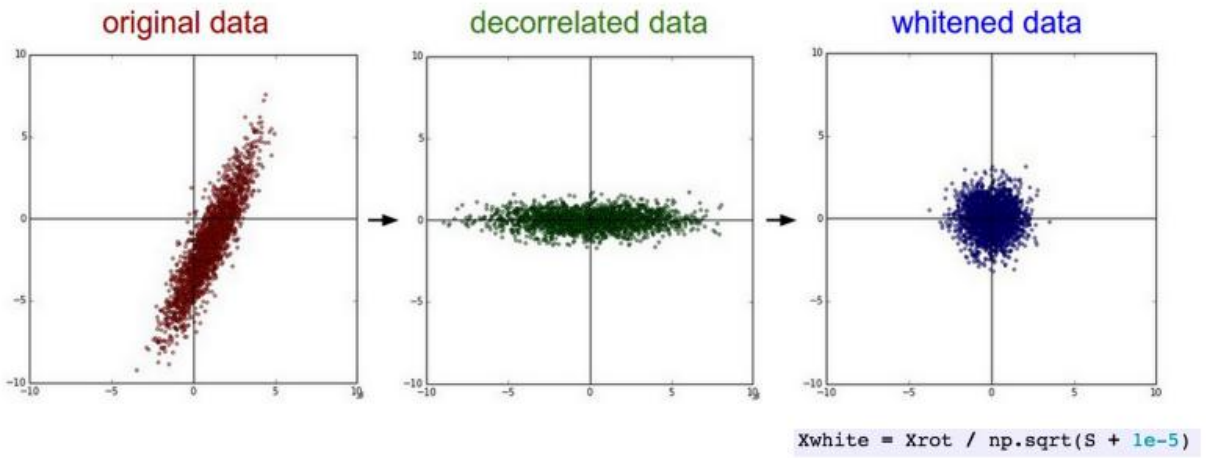
通常作为第一步且较简单的一种方式是去均值（zero-centered ，通俗地说：让每个样本都减去整体样本的均值，使整体样本的新均值为 0 ），并规范化（normalize）它们。

另一种在预处理用于规范化（normalize）数据的方法是将每一个维度的最大最小值分别限定为 1 和 −1 。



2. 主成分分析白化

PCA-Whitening:



Exaggerating noise!

PCA-Whitening 的缺点：放大噪声

- 在此过程中，数据先经过去均值，然后计算出（能刻画数据内部相关结果的）协方差矩阵：

  >>> *X -= np.mean(X, axis = 0)* # 去均值

  >>> *cov = np.dot(X.T, X) / X.shape[0]* # 计算协方差矩阵

- 之后对数据去相关，方法是将（刚刚去均值后的）原始数据投影到特征基（ eigenbasis ）上：

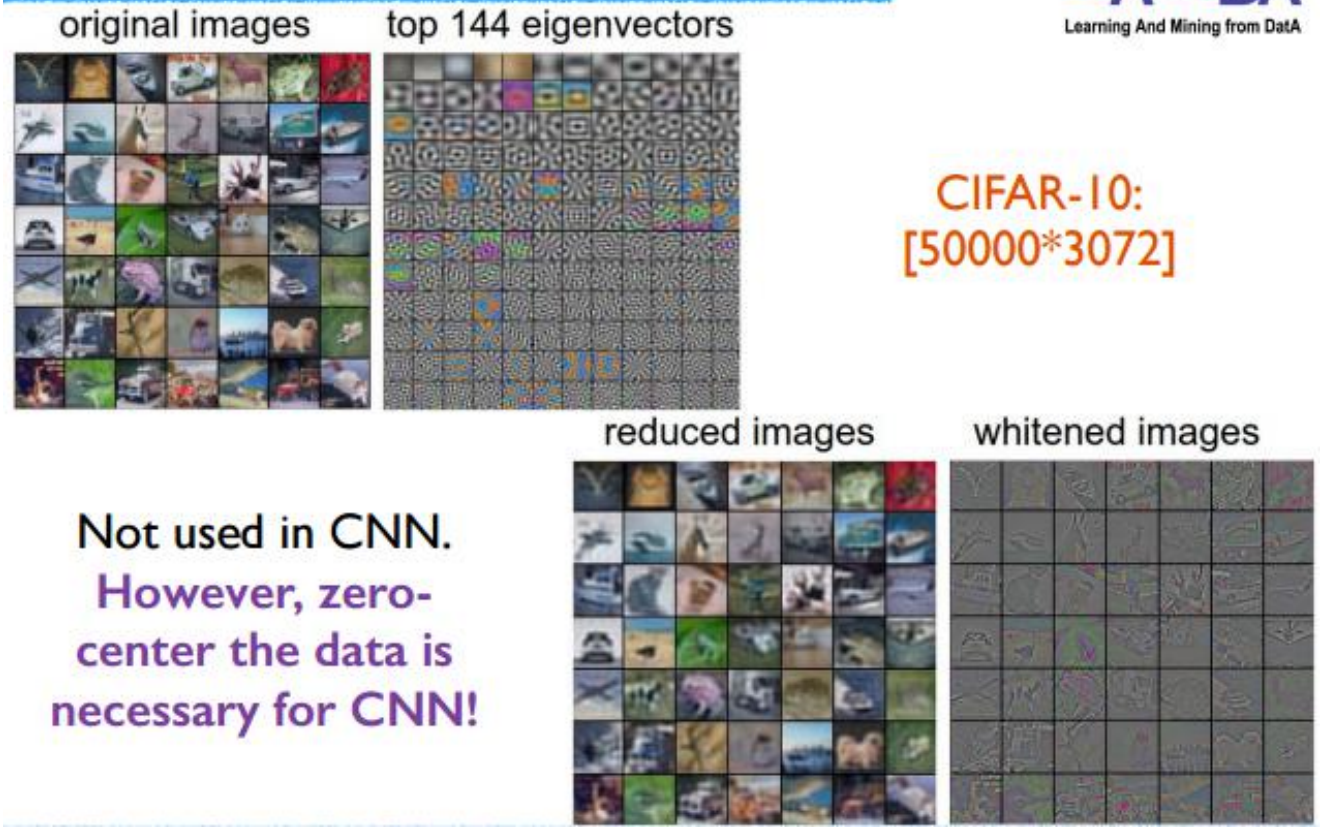
  >>> *U,S,V = np.linalg.svd(cov)* # 对数据的协方差矩阵计算 SVD 分解

  >>> *Xrot = np.dot(X, U)* # 对数据去相关

- 最后一步是白化，它对去相关后的数据在每个维度上的特征值做尺度规范化处理：

  >>> *Xwhite = Xrot / np.sqrt(S + 1e-5)* # 除以特征值（其实是奇异值的开平方根）

PCA-whitening: a demo



去均值对数据来说是很重要的，还有就是对图像像素做规范化（ normalization ）处理也是常见做法。

3.参数初始化

- 训练网络前对参数做初始化。
- 常用的初始化方法：

  1. 全零初始化 —> 错误

  2. 小随机数初始化

  是一种接近 0 但不是 00的权重初始化方法。做法是初始化权重为接近 0 的随机小数，因为很接近 0 但不相等，这也被称为“对称破缺”（ symmetry breaking ）。

We still want the weights to be very close to 0.

$$weights \sim 0.001 \times N(0,1)$$

Uniform distribution is also ok.

Warning! Small numbers will diminish the "gradient signal" flowing backward through a network.

也可使用均匀分步
在网络的回传过程中，小值权重会减弱“梯度信号”

- 3. 方差校准

使用前面讲到方法对权重随机初始化得到的神经元都存在一个问题，网络输出单元的值的分布的方差（ variance ）会随着输入单元的增多而变大。但可以让每个随机得到的权重向量通过除以输入单元个数的平方根 sqrt(n) 来规范化（ normalize ），代码如下：

```
>>> w = np.random.randn(n) / sqrt(n) # 使用输入单元个数 n 的平方根来校正最终输出导致的高方差
```

其中函数 randn 表示生成的结果服从标准正态分布（即高斯分布），变量 n 表示输入单元的个数。这样确保了网络中神经元在最初时有着大致相同的输出分布，以及收敛速度的提升。

the raw activation before non-linear filters

$$\mathrm{Var}(s) = \mathrm{Var}\left(\sum_i^n w_i x_i\right)$$

i.i.d.

$$= \sum_i^n \mathrm{Var}(w_i x_i)$$

$$= \sum_i^n [E(w_i)]^2 \mathrm{Var}(x_i) + E[(x_i)]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

assumptions: $E(w_i)=E(x_i)=0$

$$= \sum_i^n \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

$$= (n\mathrm{Var}(w))\,\mathrm{Var}(x)$$

equals one

$$nVar(w) = Var(w/\sqrt{n})$$

```
w = np.random.randn(n) / sqrt(n)
```

注：i.i.d是独立同分布（independent identically distributed ， IID ）的缩写。

- 4. 推荐方法

  先前通过校准神经元上的方差来初始化参数并未考虑使用 ReLUs 这样的激活函数。最近一篇论文《*Surpassing Human-Level Performance on ImageNet Classification*》讨论了如何为 ReLUs 这样的激活函数做参数初始化，从而使网络中神经元的方差为 2.0/n，初始化方式如下：

  ```
  >>> w = np.random.randn(n) * sqrt(2.0/n) # 目前推荐做法
  ```

They reached the conclusion that the variance of neurons in the network should be **2.0/n**.

```
w = np.random.randn(n) * sqrt(2.0/n)
```

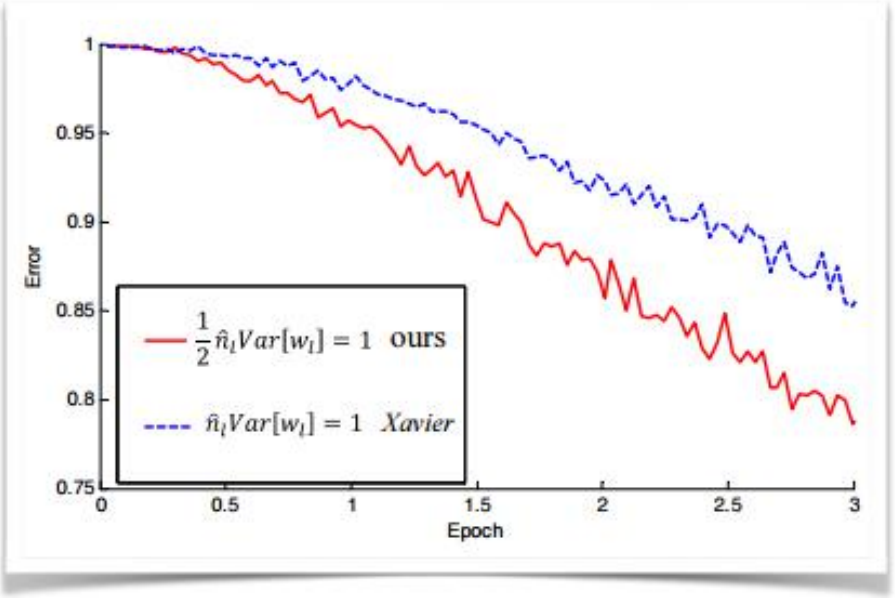


Figure 2. The convergence of a **22-layer** large model (B in Table 3). The x-axis is the number of training epochs. The y-axis is the top-1 error of 3,000 random val samples, evaluated on the center crop. We use ReLU as the activation for both cases. Both our initialization (red) and "*Xavier*" (blue) [7] lead to convergence, but ours starts reducing error earlier.

参数初始化在Caffe上的设置

```
layer {
  name: "conv1"                    # 名称：conv1
  type: "Convolution"              # 类型：卷积层
  bottom: "data"                   # 输入层：数据层
  top: "conv1"                     # 输出层：卷积层1
  # 滤波器（filters）的学习速率因子和衰减因子
  param { lr_mult: 1 decay_mult: 1 }
  # 偏置项（biases）的学习速率因子和衰减因子
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 96                 # 96个滤波器（filters）
    kernel_size: 11                # 每个滤波器（filters）大小为11*11
    stride: 4                      # 每次滤波间隔为4个像素
    weight_filler {
      type: "gaussian"             # 初始化高斯滤波器（Gaussian）
      std: 0.01                    # 标准差为0.01，  均值默认为0
    }
    bias_filler {
      type: "constant"             # 初始化偏置项（bias）为零
      value: 0
    }
  }
}
```

```
optional FillerParameter weight_filler = 7; // The filler for the weight
```

```
message FillerParameter {
  // The filler type.
  optional string type = 1 [default = 'constant'];
  optional float value = 2 [default = 0]; // the value in constant filler
  optional float min = 3 [default = 0]; // the min value in uniform filler
  optional float max = 4 [default = 1]; // the max value in uniform filler
  optional float mean = 5 [default = 0]; // the mean value in Gaussian filler
  optional float std = 6 [default = 1]; // the std value in Gaussian filler
  // The expected number of non-zero output weights for a given input in
  // Gaussian filler -- the default -1 means don't perform sparsification.
  optional int32 sparse = 7 [default = -1];
  // Normalize the filler variance by fan_in, fan_out, or their average.
  // Applies to 'xavier' and 'msra' fillers.
  enum VarianceNorm {
    FAN_IN = 0;
    FAN_OUT = 1;
    AVERAGE = 2;
  }
  optional VarianceNorm variance_norm = 8 [default = FAN_IN];
}
```

4. 卷积层参数tricks

- 1. 图片输入是2的幂次方，例如32、64、96、224等。
- 2. 卷积核大小是3*3或者5*5。
- 3. 输入图片上下左右需要用0补充，即padding，且假如卷积核大小是5那么padding就是2（图片左右上下都补充2），卷积核大小是3padding大小就是1。

5.池化层参数tricks

- 1. pooling层也能防止过拟合，使用overlapped pooling，即用来池化的数据有重叠，但是pooling的大小不要超过3，常用的池化是2X2。
- 2. max pooling比avg pooling效果会好一些。

6.学习率

- 1. 0.1是学习率的常用值
- 2. 在实际中，如果在验证集上看不到性能的提升（如损失函数值下降或者准确率上升），那就可以对当前的学习率除以 2（或 5 ）看看效果并循环这一过程，或许能给你一个惊喜。

7.正则化：防止过拟合

- 过拟合，就是拟合函数需要顾忌每一个点，最终形成的拟合函数波动很大。在某些很小的区间里，函数值的变化很剧烈。这就意味着函数在某些小区间里的导数值（绝对值）非常大，由于自变量值可大可小，所以只有系数足够大，才能保证导数值很大。而正则化是通过约束参数的范数使其不要太大，所以可以在一定程度上减少过拟合情况。
- 常用防止过拟合方式：

1.数据集扩增

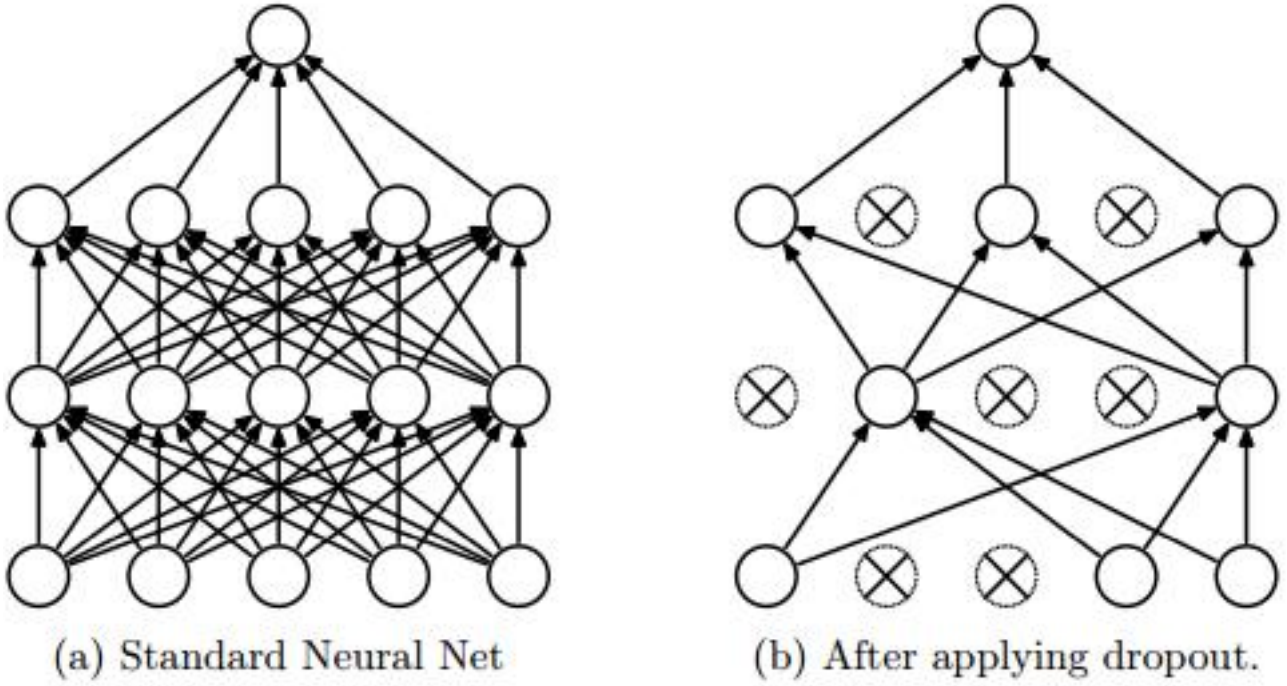
2.正则化

2.1 L2正则化

2.2 L1正则化

3. 最大模限制

4. Dropout



dropout策略思路

- Dropout 是一个超级有效、简单且是前阵子由 Srivastava 等人提出《*Dropout: A Simple Way to Prevent Neural Networks from Overfitting*》的，它是其它正则方法（如 L1 、 L2 、 最大模限制 ）的补充。在训练中， dropout 可理解为对整个神经网络进行抽样（出的网络），并基于输入数据仅仅更新抽样网络的参数。（因为这些抽样得到的网络是共享参数的，所以这些抽样出的网络的权重参数并非是独立的）。

Performance comparisons:

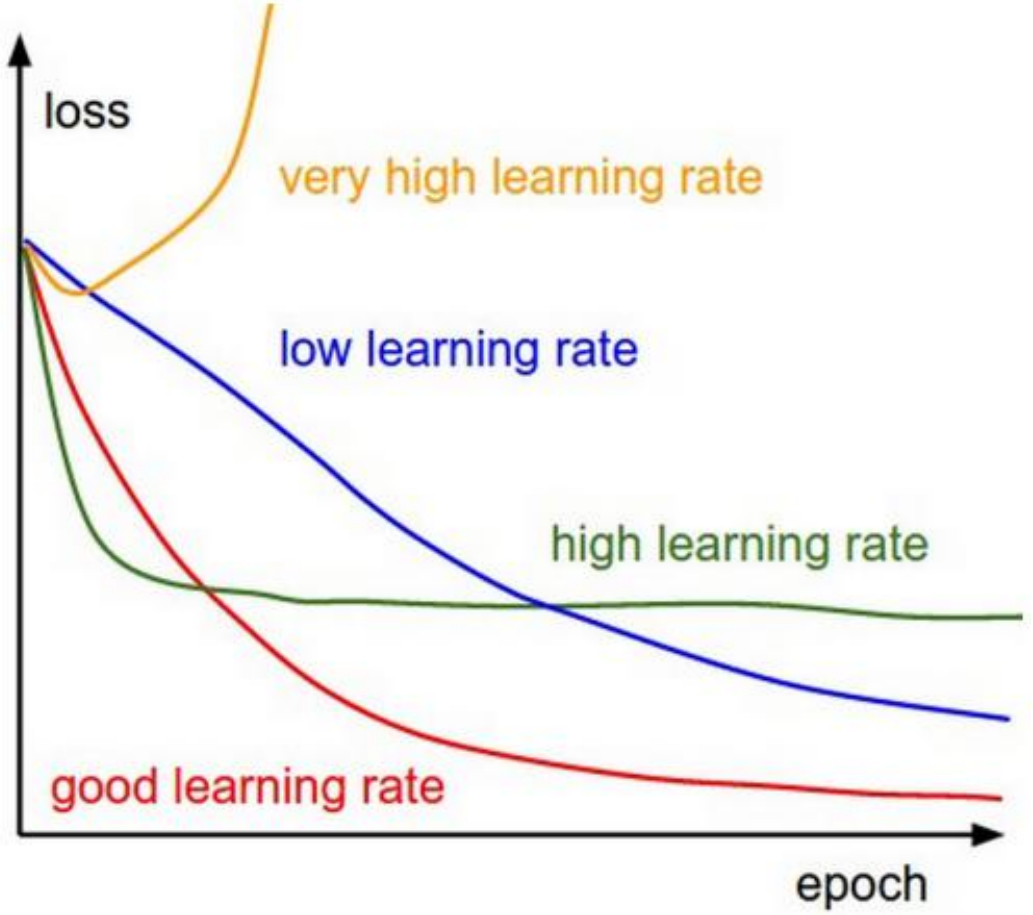
Method	Test Classification error %
L2	1.62
L2 + L1 applied towards the end of training	1.60
L2 + KL-sparsity	1.55
Max-norm	1.35
Dropout + L2	1.25
Dropout + Max-norm	**1.05**

Table 9: Comparison of different regularization methods on MNIST.

六种正则方法基于 MNIST 数据集的性能比较

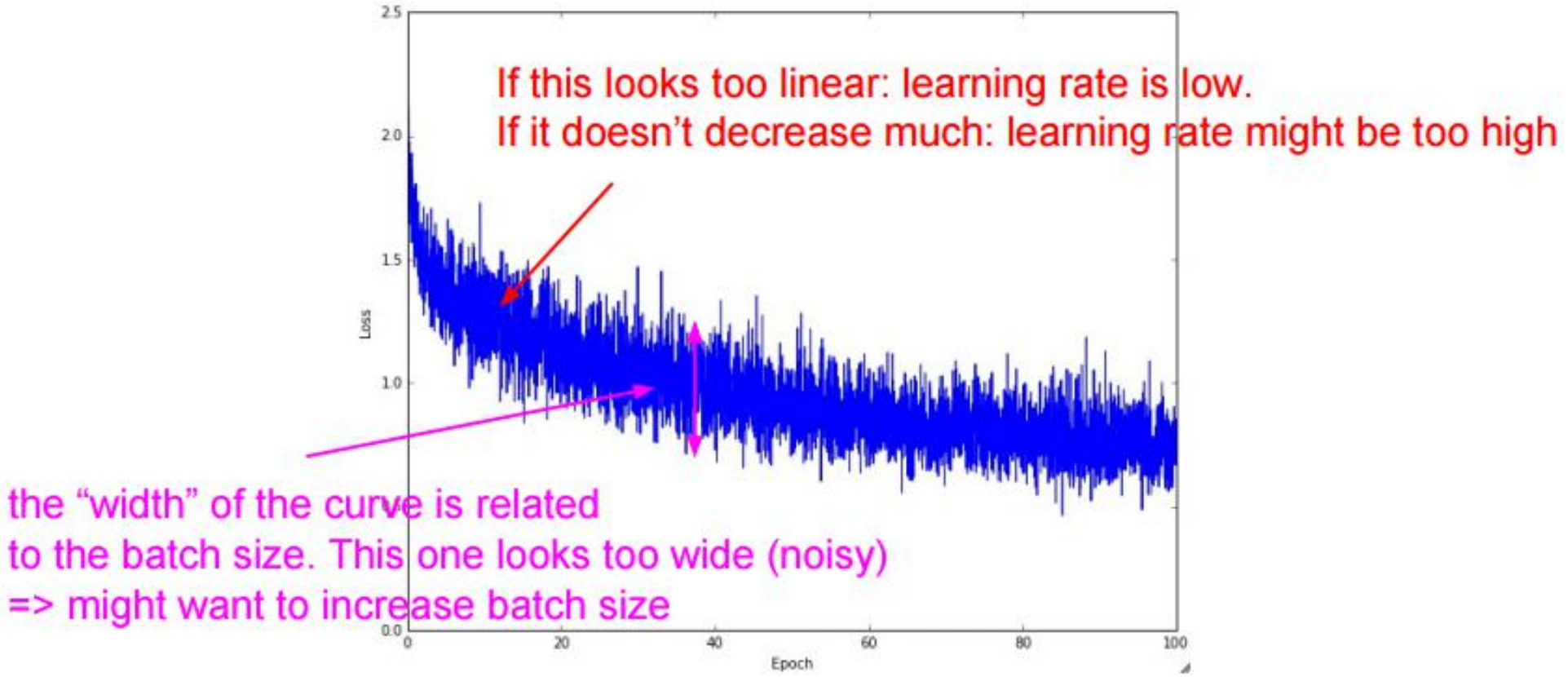
8.观察损失曲线：学习率

The learning rate:



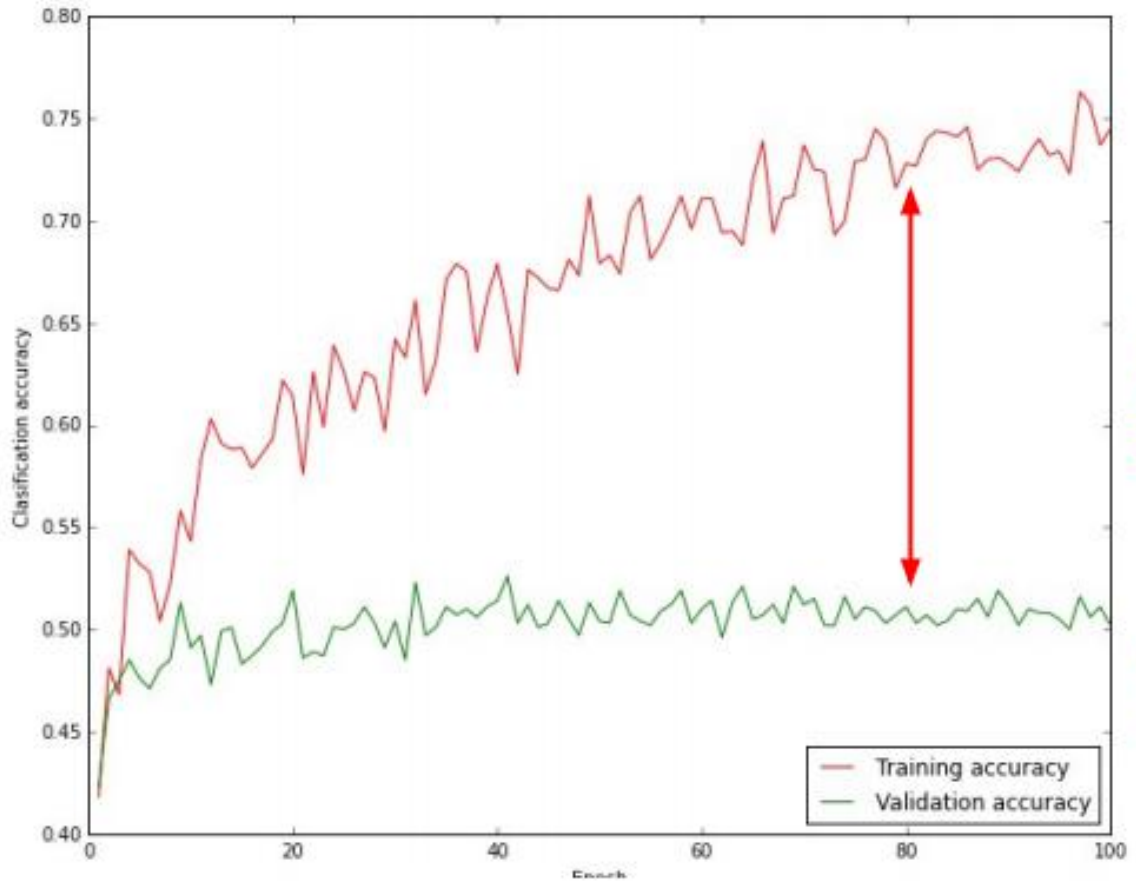
9.放大损失曲线：学习率、batch 大小

The loss curve:



10.准确率曲线

The accuracy:



big gap = overfitting

=> increase regularization strength

no gap

=> increase model capacity

11. Fine-tuning方法

- 如果你的数据量有限，那么，一般不建议自己完全从头训练起caffe模型。一般是找相关的项目或者模型，先finetuning一下，之后再慢慢的调整。一般fine tuning的方式，都是把learning rate（solver.prototxt）调低（为原来的十分之一），之后把训练模型的最后一层或者两层的学习速率调大一点————这就相当于，把模型的前面那些层的学习调低，使得参数更新的慢一点以达到微调的目的。

- 微调的时候，有时候训练数据特别少，而且希望模型的前面几层的参数保持不变。方法是使得这几个层的学习速率为0就可以了，比如设定lr_mult为0。

12.模型集成

- 在机器学习中，集成方法（ ensemble methods ）是指训练多个学习器并在之后将它们组合使用，最终得到一个强有力的分类器的方法。
- 几种集成方式的技巧：
- 1. 集成不同初始化的模型

  使用交叉验证集来确定最佳的超参数，再在基于最佳超参数的情况下，使用不同的随机初始化方法来初始化权重来训练多个模型。该方法的风险在于权重初始化方法的不同产生的差异。
- 2.集成 topN 表现的模型

  使用交叉验证集确定了最佳的超参数后，再选取表现最佳的前 topN 个模型进行集成。这可以提升集成模型的多样性，但风险就是这几个模型都是局部最优模型。实际实践中，这种做法可以达到不错的性能，因为不需要（在交叉验证后）对模型进行额外的重新训练。实际上，可以直接在 Caffe Model Zoo 中选择表现性能在 topN 的几个深度模型进行集成。

- 3.集成相同但不同阶段的模型

  当训练一个模型的代价很大时（比方时间），可以训练一个单一的网络模型但在不同的阶段记录网络权重参数，比方每个 epoch 后记录一次参数（相当于得到一个模型），用这些得到模型参数实现模型集成。显而易见的是，这种方法缺乏了网络的多样性（因为是基于一个网络的），但在实际中的表现还算不错，优点在于相比训练多个不同的网络模型做集成，该方法更易于实现。