深层神经网络的超参数调试、正则化以及优化

目 录

- 一.深度学习的实用层面
- 二. 优化算法
- 三. 超参数调试、Batch 正则化

一.深度学习的实用层面

训练集、验证集、测试集

（1）一般地，我们将所有的样本数据分成三个部分：Train/Dev/Test sets。
（2）Train sets用来训练算法模型；Dev sets用来验证不同算法的表现情况，从中选择最好的算法模型；Test sets用来测试最好算法的实际表现，作为该算法的无偏估计。
（3）之前通常设置Train sets和Test sets的数量比例为70%和30%。如果有Dev sets，则设置比例为60%、20%、20%，分别对应Train/Dev/Test sets。这种比例分配在样本数量不是很大时，例如100,1000,10000，是比较科学的。
（4）但是如果数据量很大时，例如100万，这种比例分配就不太合适了。科学的做法是要将Dev sets和Test sets的比例设置得很低，例如98%/1%/1%。因为Dev sets的目标是用来比较验证不同算法的优劣，Test sets目标是测试已选算法的实际表现，无偏估计。

一.深度学习的实用层面

参数的选择

（1）选择最佳的训练集（Training sets）、验证集（Development sets）、测试集（Test sets）对神经网络的性能影响非常重要。

（2）除此之外，在构建一个神经网络的时候，需要设置许多参数，例如神经网络的层数、每个隐藏层包含的神经元个数、学习因子（学习速率）、激活函数的选择等等。实际上很难在第一次设置的时候就选择到这些最佳的参数，而是需要通过不断地迭代更新来获得。

一.深度学习的实用层面

参数的循环选择

循环迭代的过程是这样的：
（1）先有个想法Idea，先选择初始的参数值，构建神经网络模型结构；
（2）然后通过代码Code的形式，实现这个神经网络；
（3）最后，通过实验Experiment验证这些参数对应的神经网络的表现性能。根据验证结果，对参数进行适当的调整优化，
（4）再进行下一次的Idea->Code->Experiment循环。通过很多次的循环，不断调整参数，选定最佳的参数值，从而让神经网络性能最优化。

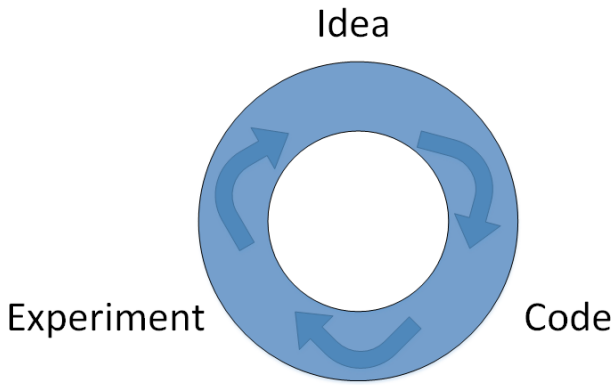
layers

hidden units

learning rates

activation functions

...



一.深度学习的实用层面

训练样本和测试样本分布上不匹配

训练样本和测试样本来自于不同的分布。

（1）举个例子，假设你开发一个手机app，可以让用户上传图片，然后app识别出猫的图片。在app识别算法中，你的训练样本可能来自网络下载，而你的验证和测试样本可能来自不同用户的上传。从网络下载的图片一般像素较高而且比较正规，而用户上传的图片往往像素不稳定，且图片质量不一。因此，训练样本和验证/测试样本可能来自不同的分布。

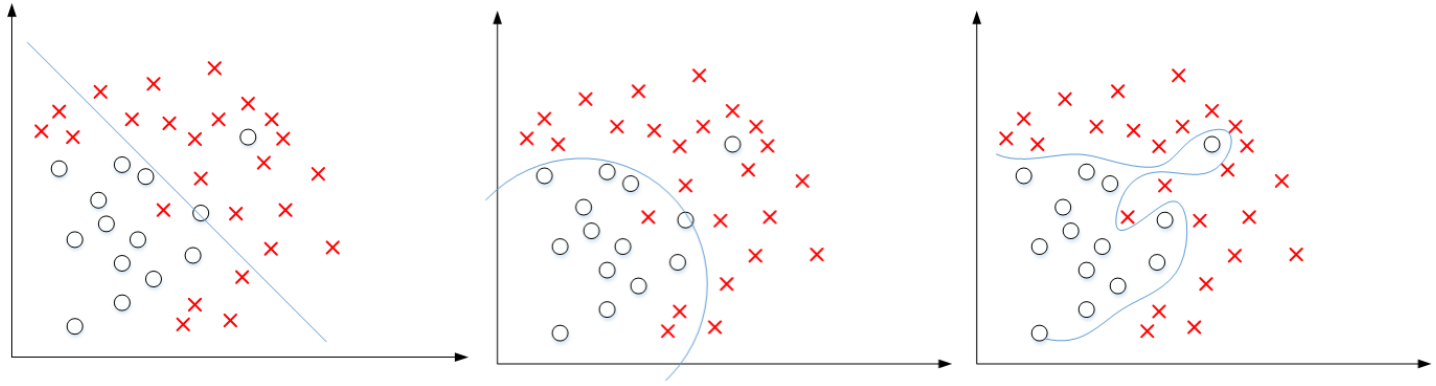
（2）解决这一问题的比较科学的办法是尽量保证Dev sets和Test sets来自于同一分布。

（3）值得一提的是，训练样本非常重要，通常我们可以将现有的训练样本做一些处理，例如图片的翻转、假如随机噪声等，来扩大训练样本的数量，从而让该模型更加强大。即使Train sets和Dev/Test sets不来自同一分布，使用这些技巧也能提高模型性能。

一.深度学习的实用层面

偏差（Bias）和方差（Variance）

偏差（Bias）和方差（Variance）是机器学习领域非常重要的两个概念和需要解决的问题。在传统的机器学习算法中，Bias和Variance是对立的，分别对应着欠拟合和过拟合，常常需要在Bias和Variance之间进行权衡。而在深度学习中，可以同时减小Bias和Variance，构建最佳神经网络模型。



一.深度学习的实用层面

高偏差和高方差的处理

机器学习中基本的一个诀窍就是避免出现high bias和high variance。
（1）首先，减少high bias的方法通常是增加神经网络的隐藏层个数、神经元个数，训练时间延长，选择其它更复杂的NN模型等。在base error不高的情况下，一般都能通过这些方式有效降低和避免high bias，至少在训练集上表现良好。
（2）其次，减少high variance的方法通常是增加训练样本数据，进行正则化Regularization，选择其他更复杂的NN模型等。

一.深度学习的实用层面

正则化

采用L2 regularization，其表达式为：

$$J(w^{[1]},b^{[1]},\cdots,w^{[L]},b^{[L]})=\frac{1}{m}\sum_{i=1}^{m}L(\hat{y}^{(i)},y^{(i)})+\frac{\lambda}{2m}\sum_{l=1}^{L}||w^{[l]}||^2$$

$$||w^{[l]}||^2=\sum_{i=1}^{n^{[l]}}\sum_{j=1}^{n^{[l-1]}}(w_{ij}^{[l]})^2$$

为什么只对w进行正则化而不对b进行正则化呢？
其实也可以对b进行正则化。但是一般w的维度很大，而b只是一个常数。相比较来说，参数很大程度上由w决定，改变b值对整体模型影响较小。所以，一般为了简便，就忽略对b的正则化了。

一.深度学习的实用层面

正则化

除了L2 regularization之外，还有另外一只正则化方法：L1 regularization。
其表达式为：

$$J(w^{[1]},b^{[1]},\cdots,w^{[L]},b^{[L]})=\frac{1}{m}\sum_{i=1}^{m}L(\hat{y}^{(i)},y^{(i)})+\frac{\lambda}{2m}\sum_{l=1}^{L}||w^{[l]}||$$

$$||w^{[l]}||=\sum_{i=1}^{n^{[l]}}\sum_{j=1}^{n^{[l-1]}}(w_{ij}^{[l]})^2$$

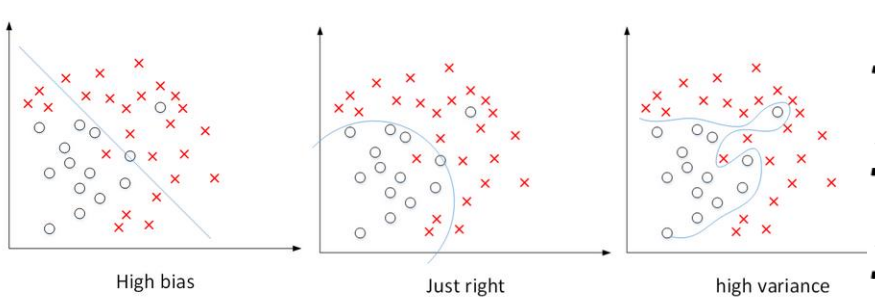
与L2 regularization相比，L1 regularization得到的w更加稀疏，即很多w为零值。其优点是节约存储空间，因为大部分w为0。
然而，实际上L1 regularization在解决high variance方面比L2 regularization并不更具优势。而且，L1的在微分求导方面比较复杂。所以，一般L2 regularization更加常用。

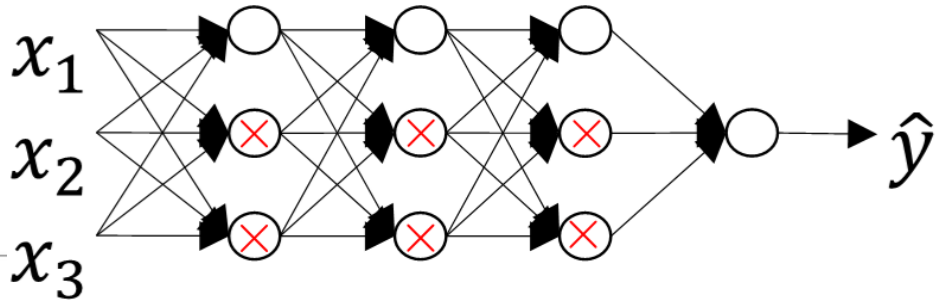
注意：L1、L2 regularization中的λ就是是正则化参数（超参数的一种）。

一.深度学习的实用层面

为什么正则化有效？

在未使用正则化的情况下，我们得到的分类超平面可能是类似上图右侧的过拟合。
但是，如果使用L2 regularization，当λ很大时，w[l]≈0。w[l]近似为零，意味着该神经网络模型中的某些神经元实际的作用很小，可以忽略。从效果上来看，其实是将某些神经元给忽略掉了。这样原本过于复杂的神经网络模型就变得不那么复杂了，而变得非常简单化了。如下图所示，整个简化的神经网络模型变成了一个逻辑回归模型。问题就从high variance变成了high bias了 。
因此，选择合适大小的λ值，就能够同时避免high bias和high variance，得到最佳模型。

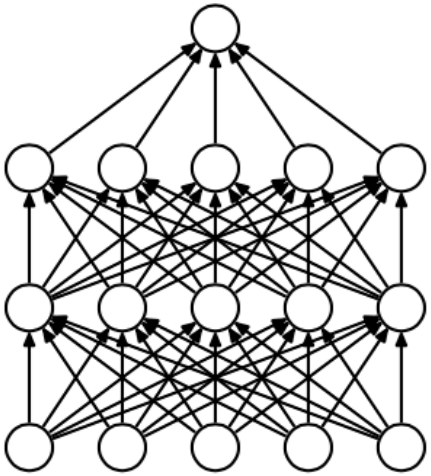


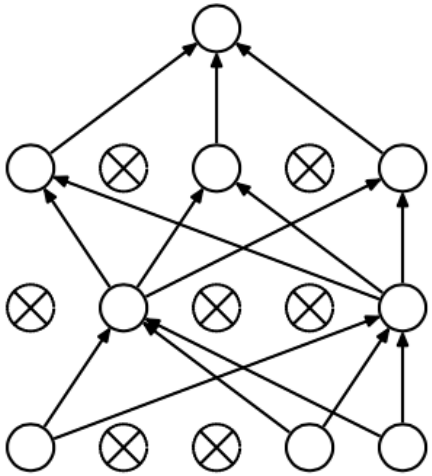


一.深度学习的实用层面

Dropout 正则化

Dropout是指在深度学习网络的训练过程中，对于每层的神经元，按照一定的概率将其暂时从网络中丢弃。
也就是说，每次训练时，每一层都有部分神经元不工作，起到简化复杂网络模型的效果，从而避免发生过拟合。
值得注意的是，使用dropout训练结束后，在测试和实际应用模型时，不需要进行dropout和随机删减神经元，所有的神经元都在工作。



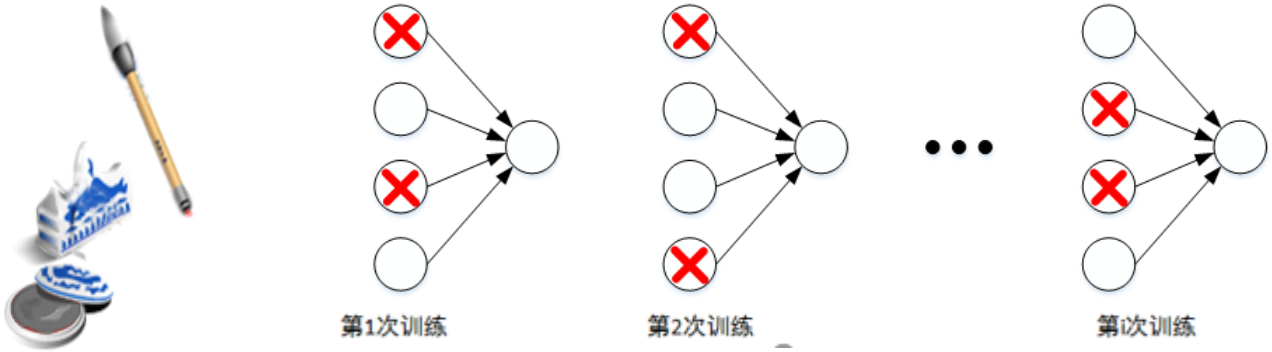


一.深度学习的实用层面

理解Dropout 正则化

（1）Dropout通过每次迭代训练时，随机选择不同的神经元，相当于每次都在不同的神经网络上进行训练，类似机器学习中Bagging的方法，能够防止过拟合。

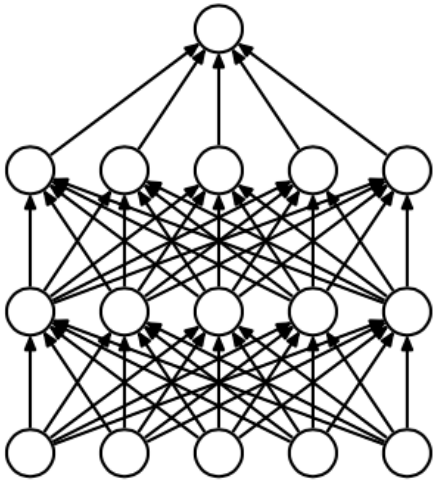
（2）除此之外，还可以从权重w的角度来解释为什么dropout能够有效防止过拟合。对于某个神经元来说，某次训练时，它的某些输入在dropout的作用被过滤了。而在下一次训练时，又有不同的某些输入被过滤。经过多次训练后，某些输入被过滤，某些输入被保留。这样，该神经元就不会受某个输入非常大的影响，影响被均匀化了。也就是说，对应的权重w不会很大。这从从效果上来说，与L2 regularization是类似的，都是对权重w进行“惩罚”，减小了w的值。

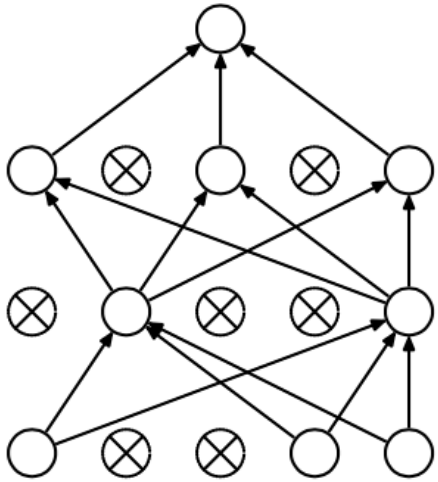


一.深度学习的实用层面

Dropout 正则化的总结

对于同一组训练数据，利用不同的神经网络训练之后，求其输出的平均值可以减少overfitting。Dropout就是利用这个原理，每次丢掉一定数量的隐藏层神经元，相当于在不同的神经网络上进行训练，这样就减少了神经元之间的依赖性，即每个神经元不能依赖于某几个其他的神经元（指层与层之间相连接的神经元），使神经网络更加能学习到与其他神经元之间的更加健壮（robust）的特征。





一.深度学习的实用层面

Dropout 正则化的使用

在使用dropout的时候，有几点需要注意。
（1）首先，不同隐藏层的dropout系数keep_prob可以不同。一般来说，神经元越多的隐藏层，keep_out可以设置得小一些.，例如0.5；神经元越少的隐藏层，keep_out可以设置的大一些，例如0.8，设置是1。
（2）另外，实际应用中，不建议对输入层进行dropout，如果输入层维度很大，例如图片，那么可以设置dropout，但keep_out应设置的大一些，例如0.8，0.9。总体来说，就是越容易出现overfitting的隐藏层，其keep_prob就设置的相对小一些。没有准确固定的做法，通常可以根据validation进行选择。

一.深度学习的实用层面

Dropout 正则化的训练展示

在mnist数据集上的神经网络应用dropout：

```
def network_mnist(input, in_size, out_size, keep_prob):
    hidden_layer1 = add_layer(input, in_size, 1024, activation_function=tf.nn.relu)
    hidden_layer1 = tf.nn.dropout(hidden_layer1, keep_prob)
    hidden_layer2 = add_layer(hidden_layer1, 1024, 512, activation_function=tf.nn.relu)
    hidden_layer2 = tf.nn.dropout(hidden_layer2, keep_prob)
    hidden_layer3 = add_layer(hidden_layer2, 512, 256, activation_function=tf.nn.relu)
    hidden_layer3 = tf.nn.dropout(hidden_layer3, keep_prob)
    prediction = add_layer(hidden_layer3, 256, out_size, activation_function=tf.nn.softmax)
    return prediction
```

一.深度学习的实用层面

Dropout 正则化的训练展示

加入了Dropout后，在epoch大于20时，相对于原来的神经网络，还没出现明显的过拟合现象。

程序在“project/tf_build_net_dropout”文件夹

```
11:17:18 epoch:08 is_save:1 acc_max:97.6800% acc_val:97.6800% acc_train:96.5446%
11:17:20 epoch:09 is_save:0 acc_max:97.6800% acc_val:97.6600% acc_train:97.0174%
11:17:22 epoch:10 is_save:1 acc_max:97.7400% acc_val:97.7400% acc_train:97.2620%
11:17:24 epoch:11 is_save:1 acc_max:98.0400% acc_val:98.0400% acc_train:97.4153%
11:17:26 epoch:12 is_save:0 acc_max:98.0400% acc_val:98.0000% acc_train:97.6088%
11:17:28 epoch:13 is_save:0 acc_max:98.0400% acc_val:98.0400% acc_train:97.7074%
11:17:30 epoch:14 is_save:1 acc_max:98.1200% acc_val:98.1200% acc_train:97.9191%
11:17:33 epoch:15 is_save:1 acc_max:98.1400% acc_val:98.1400% acc_train:97.9556%
11:17:36 epoch:16 is_save:1 acc_max:98.2000% acc_val:98.2000% acc_train:98.1947%
11:17:38 epoch:17 is_save:1 acc_max:98.2200% acc_val:98.2200% acc_train:98.1144%
11:17:40 epoch:18 is_save:1 acc_max:98.2400% acc_val:98.2400% acc_train:98.3079%
11:17:42 epoch:19 is_save:1 acc_max:98.2600% acc_val:98.2600% acc_train:98.4320%
11:17:44 epoch:20 is_save:1 acc_max:98.3400% acc_val:98.3400% acc_train:98.3499%
11:17:46 epoch:21 is_save:1 acc_max:98.4800% acc_val:98.4800% acc_train:98.4850%
11:17:48 epoch:22 is_save:0 acc_max:98.4800% acc_val:98.3200% acc_train:98.5415%
11:17:50 epoch:23 is_save:0 acc_max:98.4800% acc_val:98.2000% acc_train:98.5872%
11:17:52 epoch:24 is_save:0 acc_max:98.4800% acc_val:98.3800% acc_train:98.7168%
11:17:54 epoch:25 is_save:0 acc_max:98.4800% acc_val:98.4600% acc_train:98.7350%
11:17:56 epoch:26 is_save:0 acc_max:98.4800% acc_val:98.3800% acc_train:98.8701%
11:17:58 epoch:27 is_save:1 acc_max:98.5200% acc_val:98.5200% acc_train:98.8792%
```

一.深度学习的实用层面

其他正则化的方法

除了L2 regularization和dropout regularization之外，还有其它减少过拟合的方法。

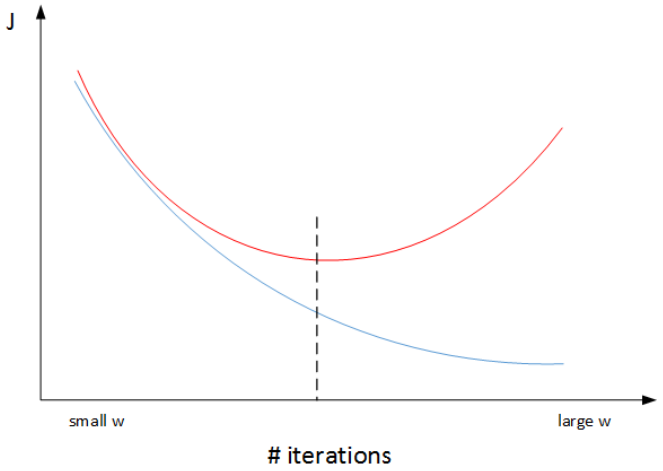
（1）增加训练样本数量。但是通常成本较高，难以获得额外的训练样本。但是，可以对已有的训练样本进行一些处理来“制造”出更多的样本，称为data augmentation。例如图片识别问题中，可以对已有的图片进行水平翻转、垂直翻转、任意角度旋转、缩放或扩大等等。如下图所示，这些处理都能“制造”出新的训练样本。虽然这些是基于原有样本的，但是对增大训练样本数量还是有很有帮助的，不需要增加额外成本，却能起到防止过拟合的效果。

一.深度学习的实用层面

其他正则化的方法

除了L2 regularization和dropout regularization之外，还有其它减少过拟合的方法。

（2）early stopping。一个神经网络模型随着迭代训练次数增加，train set error一般是单调减小的，而dev set error 先减小，之后又增大。也就是说训练次数过多时，模型会对训练样本拟合的越来越好，但是对验证集拟合效果逐渐变差，即发生了过拟合。因此，迭代训练次数不是越多越好，可以通过train set error和dev set error随着迭代次数的变化趋势，选择合适的迭代次数，即early stopping。



一.深度学习的实用层面

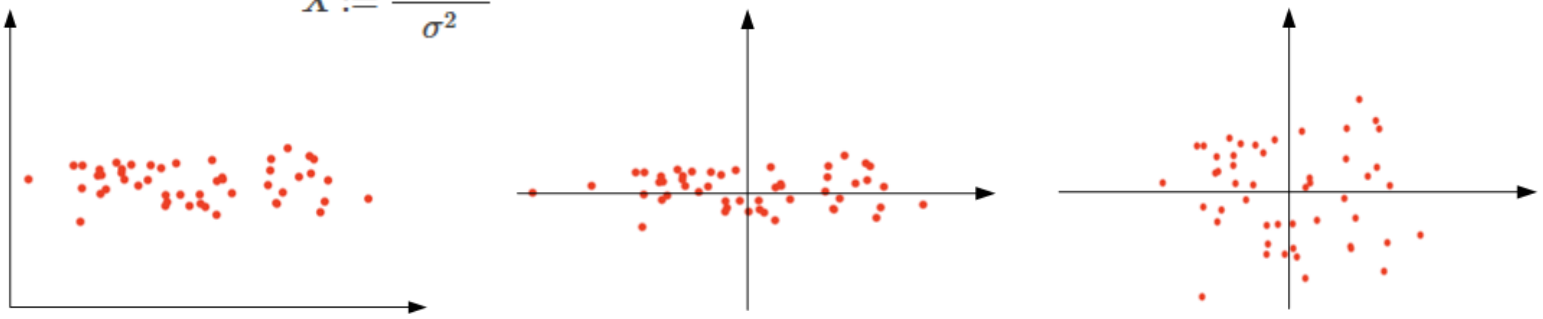
归一化输入

在训练神经网络时，归一化输入可以提高训练的速度。归一化输入就是对训练数据集进行归一化的操作，即将原始数据减去其均值μ后，再除以其方差σ^2：

$$\mu = \frac{1}{m}\sum_{i=1}^{m} X^{(i)}$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} (X^{(i)})^2$$

$$X := \frac{X - \mu}{\sigma^2}$$

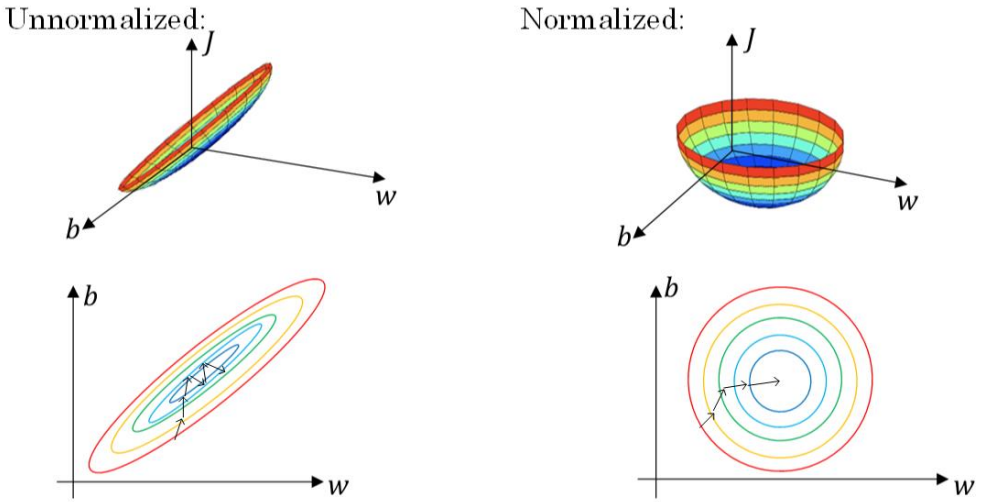


X $X - \mu$ $\frac{X - \mu}{\sigma^2}$

一.深度学习的实用层面

归一化输入

（1）值得注意的是，由于训练集进行了归一化处理，那么对于测试集或在实际应用时，应该使用同样的μ和σ2对其进行归一化处理。这样保证了训练集合测试集的归一化操作一致。
（2）如果进行了归一化操作，x1与x2分布均匀，w1和w2数值差别不大，得到的cost function与w和b的关系是类似圆形碗。对其进行梯度下降算法时，α可以选择相对大一些，且J一般不会发生振荡，保证了J是单调下降的。如下图所示。



二. 优化算法

Batch和Mini-batch 梯度下降

神经网络训练过程是对所有m个样本，称为batch，通过向量化计算方式，同时进行的。如果m很大，例如达到百万数量级，训练速度往往会很慢，因为每次迭代都要对所有样本进行进行求和运算和矩阵运算。将这种梯度下降算法称为Batch Gradient Descent。

为了解决这一问题，可以把m个训练样本分成若干个子集，称为mini-batches，这样每个子集包含的数据量就小了，例如只有1000，然后每次在单一子集上进行神经网络训练，速度就会大大提高。这种梯度下降算法叫做Mini-batch Gradient Descent。

二. 优化算法

Mini-batch 梯度下降

假设总的训练样本个数m=5000000，其维度为(n_x,m)。将其分成5000个子集，每个mini-batch含有1000个样本。我们将每个mini-batch记为X{t}，其维度为(n_x,1000)。相应的每个mini-batch的输出记为Y{t}，其维度为(1,1000)，且t=1,2, …,5000。

经过T次循环之后，所有m个训练样本都进行了梯度下降计算。这个过程，我们称之为经历了一个epoch。
对于Batch Gradient Descent而言，一个epoch只进行一次梯度下降算法；
而Mini-Batches Gradient Descent，一个epoch会进行T次梯度下降算法。

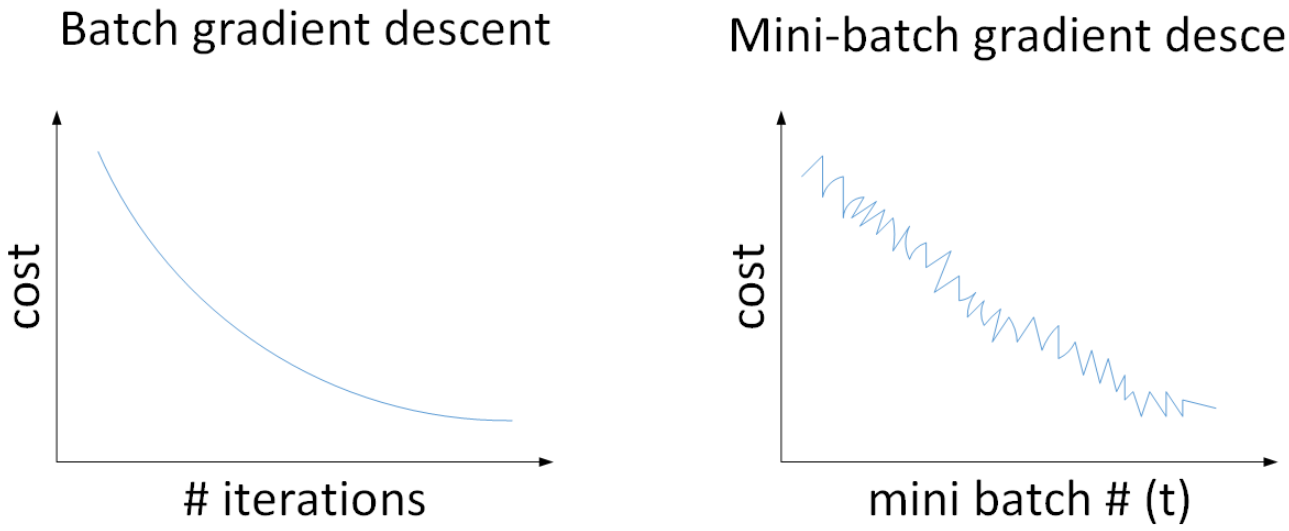
值得一提的是，对于Mini-Batches Gradient Descent，可以进行多次epoch训练。而且，每次epoch，最好是将总体训练数据重新打乱、重新分成T组mini-batches，这样有利于训练出最佳的神经网络模型。

二. 优化算法

Mini-batch 梯度下降为何有效？

对于一般的神经网络模型，使用Batch gradient descent，随着迭代次数增加，cost是不断减小的。然而，使用Mini-batch gradient descent，随着在不同的mini-batch上迭代训练，其cost不是单调下降，而是受类似noise的影响，出现振荡。但整体的趋势是下降的，最终也能得到较低的cost值。

Batch gradient descent和Mini-batch gradient descent的cost曲线如下图所示：



二. 优化算法

动量梯度下降算法

动量梯度下降算法，其速度要比传统的梯度下降算法快很多。
做法是在每次训练时，对梯度进行指数加权平均处理，然后用得到的梯度值更新权重W和常数项b。权重W和常数项b的指数加权平均表达式如下：

$$V_{dW} = \beta \cdot V_{dW} + (1-\beta) \cdot dW$$

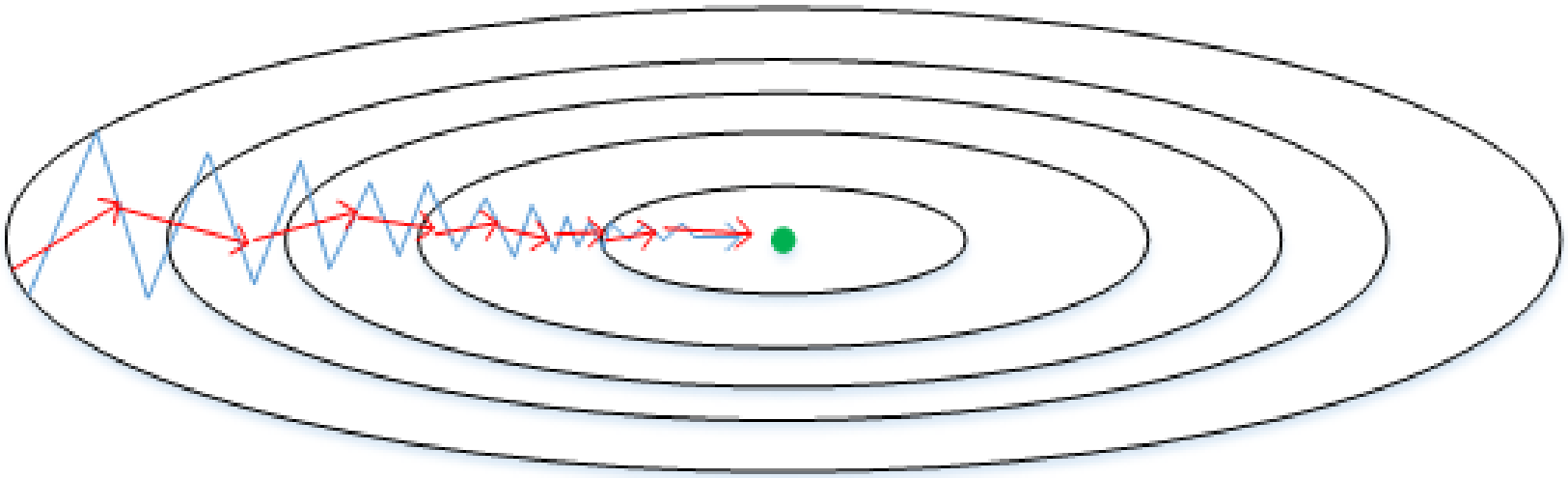
$$V_{db} = \beta \cdot V_{db} + (1-\beta) \cdot db$$

从动量的角度来看，以权重W为例，V_{dW}可以成速度V，dW可以看成是加速度a。指数加权平均实际上是计算当前的速度，当前速度由之前的速度和现在的加速度共同影响。而β<1，又能限制速度V_{dW}过大。也就是说，当前的速度是渐变的，而不是瞬变的，是动量的过程。这保证了梯度下降的平稳性和准确性，减少振荡，较快地达到最小值处。

二. 优化算法

动量梯度下降的过程

（1）原始的梯度下降算法如下图蓝色折线所示。在梯度下降过程中，梯度下降的振荡较大，尤其对于W、b之间数值范围差别较大的情况。此时每一点处的梯度只与当前方向有关，产生类似折线的效果，前进缓慢。
（2）而如果对梯度进行指数加权平均，如红色折线所示。这样使当前梯度不仅与当前方向有关，还与之前的方向有关，这样处理让梯度前进方向更加平滑，减少振荡，能够更快地到达最小值处。



二. 优化算法

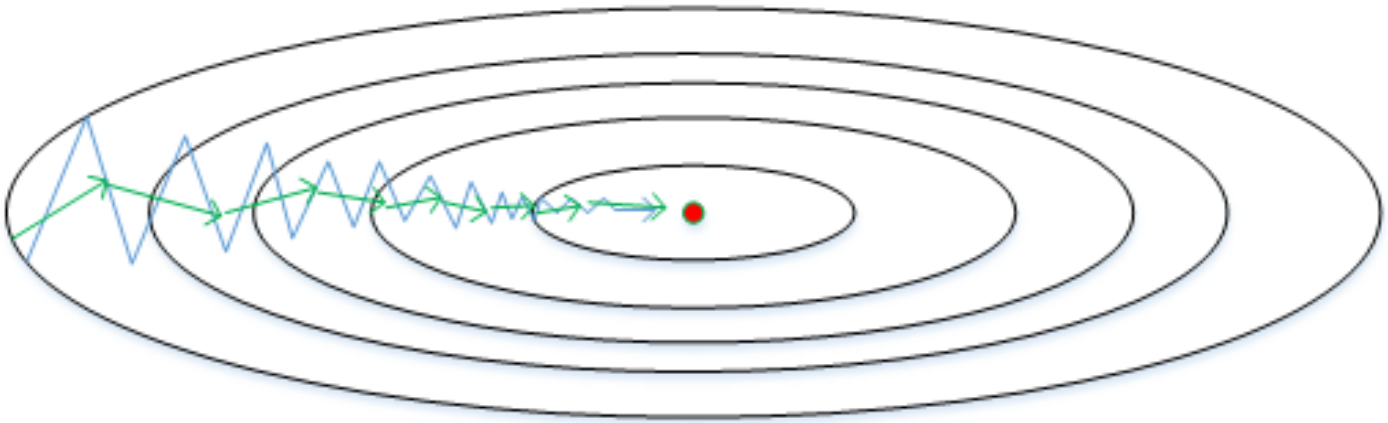
RMSprop梯度下降

RMSprop是另外一种优化梯度下降速度的算法。每次迭代训练过程中，其权重W和常数项b的更新表达式为：

$$S_W = \beta S_{dW} + (1-\beta) dW^2$$

$$S_b = \beta S_{db} + (1-\beta) db^2$$

$$W := W - \alpha \frac{dW}{\sqrt{S_W}},\ b := b - \alpha \frac{db}{\sqrt{S_b}}$$



二. 优化算法

Adam优化算法

Adam（Adaptive Moment Estimation）算法结合了动量梯度下降算法和RMSprop算法。其算法流程为：

$$V_{dW}=0,\ S_{dW},\ V_{db}=0,\ S_{db}=0$$

$$On\ iteration\ t:$$

$$Cimpute\ dW,\ db$$

$$V_{dW}=\beta_1 V_{dW}+(1-\beta_1)dW,\ V_{db}=\beta_1 V_{db}+(1-\beta_1)db$$

$$S_{dW}=\beta_2 S_{dW}+(1-\beta_2)dW^2,\ S_{db}=\beta_2 S_{db}+(1-\beta_2)db^2$$

$$V_{dW}^{corrected}=\frac{V_{dW}}{1-\beta_1^t},\ V_{db}^{corrected}=\frac{V_{db}}{1-\beta_1^t}$$

$$S_{dW}^{corrected}=\frac{S_{dW}}{1-\beta_2^t},\ S_{db}^{corrected}=\frac{S_{db}}{1-\beta_2^t}$$

$$W:=W-\alpha\frac{V_{dW}^{corrected}}{\sqrt{S_{dW}^{corrected}}+\varepsilon},\ b:=b-\alpha\frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}}+\varepsilon}$$

实际应用中，Adam算法结合了动量梯度下降和RMSprop各自的优点，使得神经网络训练速度大大提高。
但是Adam在训练后期的加速乏力。

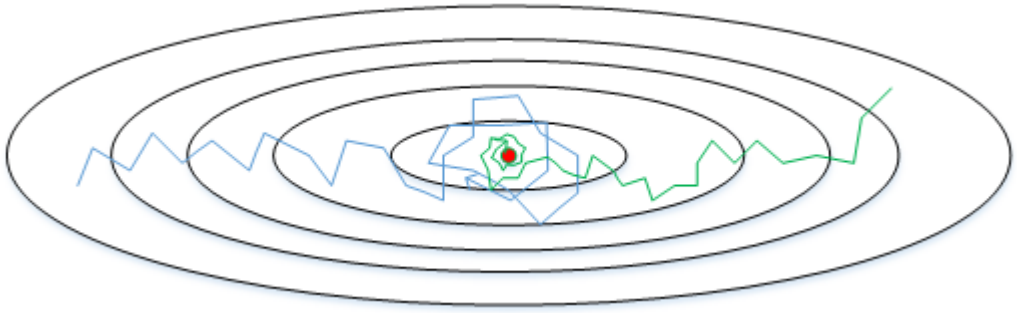
二. 优化算法

学习率衰减

Learning rate decay就是随着迭代次数增加，学习因子α逐渐减小。能有效提高神经网络训练速度。例如指数衰减如下：

$$\alpha = \frac{1}{1 + decay\_rate * epoch}\alpha_0$$

下面用图示的方式来解释这样做的好处。下图中，蓝色折线表示使用恒定的学习因子α，由于每次训练α相同，步进长度不变，在接近最优值处的振荡也大，在最优值附近较大范围内振荡，与最优值距离就比较远。绿色折线表示使用不断减小的α，随着训练次数增加，α逐渐减小，步进长度减小，使得能够在最优值处较小范围内微弱振荡，不断逼近最优值。相比较恒定的α来说，learning rate decay更接近最优值

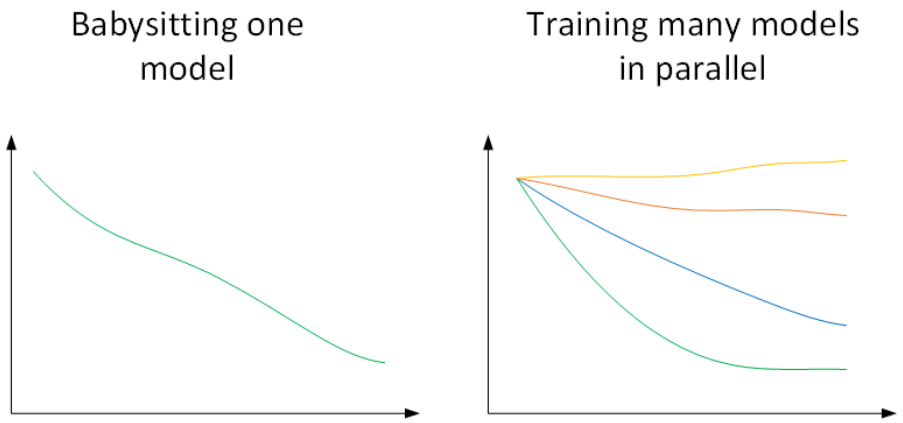


三. 超参数调试、Batch 正则化

深度神经网络需要调试的超参数

深度神经网络需要调试的超参数较多，包括：

- α：学习因子
- β：动量梯度下降因子
- $\beta_1, \beta_2, \varepsilon$：Adam算法参数
- #layers：神经网络层数
- #hidden units：各隐藏层神经元个数
- learning rate decay：学习因子下降参数
- mini-batch size：批量训练样本包含的样本个数



三. 超参数调试、Batch 正则化

Batch Normalization

Batch Normalization不仅可以让调试超参数更加简单，而且可以让神经网络模型更加“健壮”。也就是说较好模型可接受的超参数范围更大一些，包容性更强，使得更容易去训练一个深度神经网络。

Batch Normalization对第l层隐藏层的输入$Z^{[l-1]}$做如下标准化处理，忽略上标$[l-1]$：

$$\mu = \frac{1}{m}\sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m}\sum_i (z_i - \mu)^2$$

$$z_{norm}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

其中，m是单个mini-batch包含样本个数，ε是为了防止分母为零，可取值10^{-8}。这样，使得该隐藏层的所有输入$z^{(i)}$均值为0，方差为1。

三. 超参数调试、Batch 正则化

Batch Normalization

但是，大部分情况下并不希望所有的$z^{(i)}$均值都为0，方差都为1，也不太合理。通常需要对$z^{(i)}$进行进一步处理：

$$\tilde{z}^{(i)} = \gamma \cdot z_{norm}^{(i)} + \beta$$

上式中，γ和β是learnable parameters，类似于W和b一样，可以通过梯度下降等算法求得。这里，γ和β的作用是让$\tilde{z}^{(i)}$的均值和方差为任意值，只需调整其值就可以了。例如，令：

$$\gamma = \sqrt{\sigma^2 + \varepsilon}, \quad \beta = u$$

则$\tilde{z}^{(i)} = z^{(i)}$，即identity function。可见，设置γ和β为不同的值，可以得到任意的均值和方差。

这样，通过Batch Normalization，对隐藏层的各个$z^{[l](i)}$进行标准化处理，得到$\tilde{z}^{[l](i)}$，替代$z^{[l](i)}$。

三. 超参数调试、Batch 正则化

Batch Normalization的特点

值得注意的是，输入的标准化处理Normalizing inputs和隐藏层的标准化处理Batch Normalization是有区别的。

（1）Normalizing inputs使所有输入的均值为0，方差为1。

（2）Batch Normalization可使各隐藏层输入的均值和方差为任意值。

实际上，从激活函数的角度来说，如果各隐藏层的输入均值在靠近0的区域即处于激活函数的线性区域，这样不利于训练好的非线性神经网络，得到的模型效果也不会太好。这也解释了为什么需要用γ和β来对$z^{[l](i)}$作进一步处理。

总结

一、介绍深度学习如何应用与调参，进行正则化，以及Dropout的原理、归一化输入的好处；
二、介绍优化的方法，如mini-batch梯度下降、动量下降、Adam、学习率衰减等，通过梯度下降图展示它们为何有效；
三、介绍超参数的例子，以及batch norm。

谢谢聆听

参考网页：

DL的实用层面：https://blog.csdn.net/red_stone1/article/details/78208851
优化算法：https://blog.csdn.net/red_stone1/article/details/78348753
超参数调试：https://blog.csdn.net/red_stone1/article/details/78403416