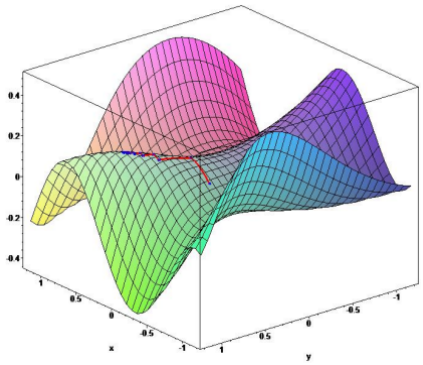
Lecture4 Optimization & Regularization

1. 优化的需求

我们已经知道梯度下降是如何完成的了



$$w^{(t+1)} = w^{(t)} - \eta_t \nabla_{w^{(t)}} L$$

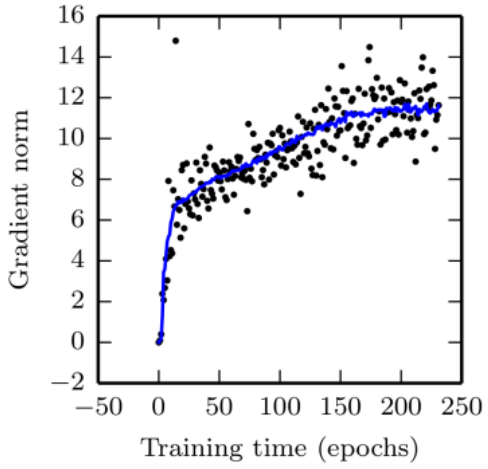
- w：某一层的参数
- $w^{(t)}$：在 t 次更新的时候，w 的值
- η_t：在 t 次更新的时候，学习率的值
- $\nabla_{w^{(t)}} L$：在 t 次更新的时候，w 下降在损失函数计算为 L 时的梯度

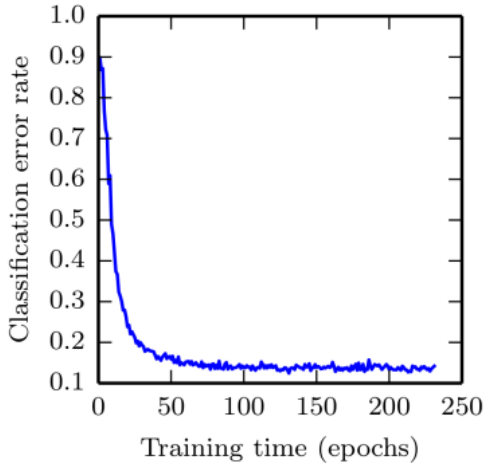
但是，针对梯度下降，其实有**很多个版本**

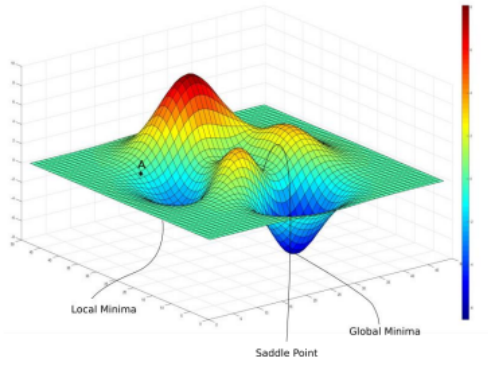
- w 如何初始化
- 学习率 η 怎么调整

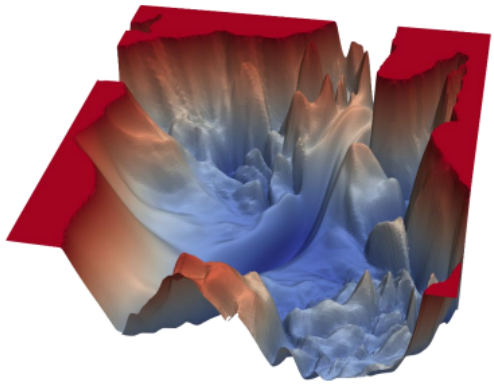
优化存在的挑战

- 步长（学习率）的选择存在问题
- 局部最小值
- 寻找的最优解所在的解空间非常复杂
    - 鞍点和其他平坦的范围

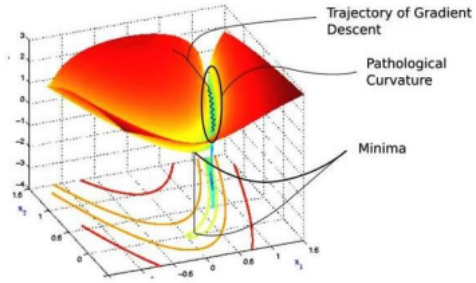


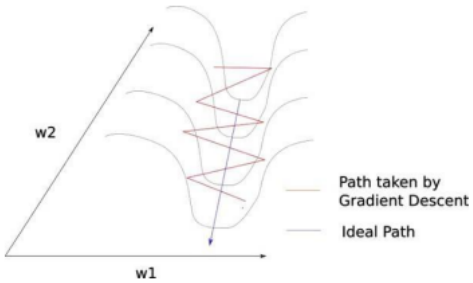


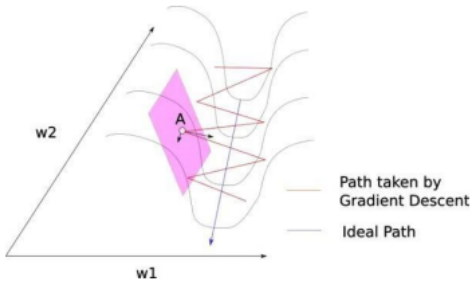


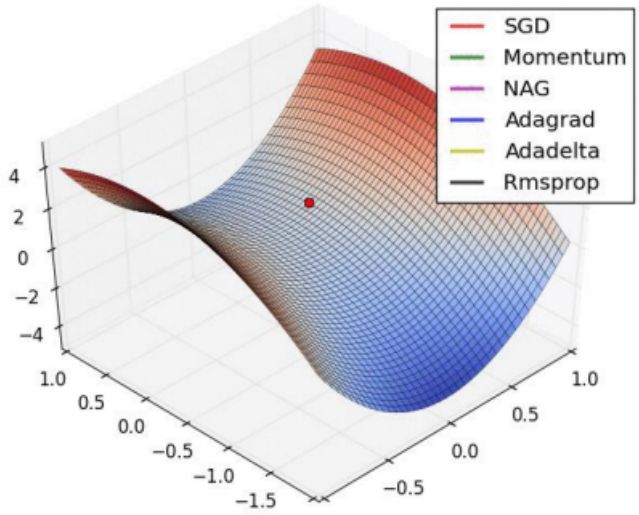


3-D representation for loss contour of a VGG-56 deep network's loss function on the CIFAR-10 dataset.









- 不同的优化方法下降的效果不一样

什么是最佳的优化

- 不幸的是，这个问题还没有一个普遍接受的答案
- 对于不同的优化方法，可以通过下面的网站查看一些可视化的优化效果
  https://github.com/Jaewan-Yun/optimizer-visualization
- 也可以看下面这个网站，总结了一些好的优化方法
  http://ruder.io/optimizing-gradient-descent/
  https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/

2. 梯度下降的优化方法

批处理梯度下降 Batch Gradient Descent

如果我们在在整个训练集训练一次后，进行一次梯度下降的话，那么叫做**批处理梯度下降**

$$\nabla_{w^{(t)}} L = \frac{1}{m} \sum_{i=1}^{m} \nabla_{w^{(t)}} L\left(w; x_i, y_i\right)$$

- m：训练集中训练样本 (x_i, y_i) 的总个数
- 对 batch 内的样本进行加权求平均
  - 减少噪音的影响
- 但注意，这只是一个整体样本梯度下降效果，这个估计可能与真实梯度不同
- 在实践中，我们将损失计算为所有训练样本的平均损失，然后我们计算这个数的梯度

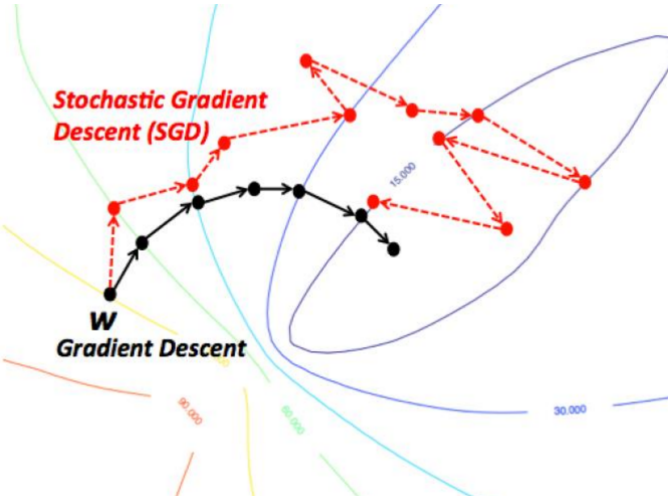
优点

- 可以使用加速度技术基于二阶导数（Hessian）
- 我们不仅可以测量梯度，也可以测量损失函数的**曲率**
- 可以对**收敛速度**做一个简单的理论分析

缺点

- **数据集可能太大了**，无法进行完整的梯度计算
- 在每次参数更新之前，多次为**数据很接近的样本**重新计算梯度（冗余）
- 损失表面是**非凸的和高维的**

随机梯度下降 Stochastic Gradient Descent



另一种方法是我们在逐个输入训练样本时便计算梯度，使用它来更新权值，这个方法叫做随机梯度下降

$$w^{(t+1)} = w^{(t)} - \eta_t \nabla_{w^{(t)}} L(w; x_i, y_i)$$

- 来一个样本，就计算一次梯度下降

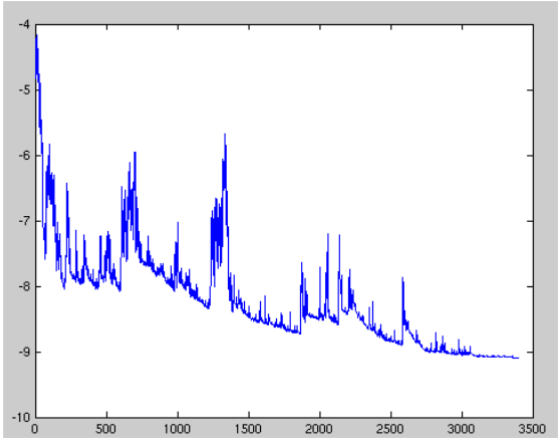
伪代码

- 随机选择各层权重 w 和学习率 η
- 重复此步骤，直到得到近似的最小值
    - 随机 Shuffle 训练集的样本
    - for i = 1,2,...,n:
        - 对每个样本 (x_i, y_i) 计算梯度并更新各层权重 w

优点

- 比梯度下降快
    - 从第一个样本开始更新梯度，而不是等待，此外，在考虑整个训练数据时，可能存在冗余
- 随机性有助于避免过拟合，从而提高准确性
- 适用于随时间变化的数据集

缺点



- 大多数情况下，**它是近似值的近似值**，所以它注定是不完美的
  - SGD 执行频繁的更新与高方差，导致目标函数波动很大
  - 但实际上这不是问题，事实上这是一个优势（噪声有助于防止过拟合）
- 主要问题是，对于大小为 1 的样本，**无法利用大规模并行性**

Mini-batch 梯度下降

那么为什么既不使用多于 1 个数据样本更新算梯度，也不是使用整个训练集呢？这种方法叫做 **mini-batch 梯度下降**

$$w^{(t+1)} = w^{(t)} - \frac{\eta_t}{|B|} \sum_{b \in B} \nabla_{w^{(t)}} L(w; b)$$

- 其中 B 是挑选出来的样本集大小 $1 \leq B \leq m$
  - m：训练集样本个数
- 它是更广义的随机梯度下降法（通常也称为 SGD）

伪代码

Algorithm 8.1 Stochastic gradient descent (SGD) update

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$
Require: Initial parameter $\boldsymbol{\theta}$
  $k \leftarrow 1$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
    $k \leftarrow k + 1$
  end while

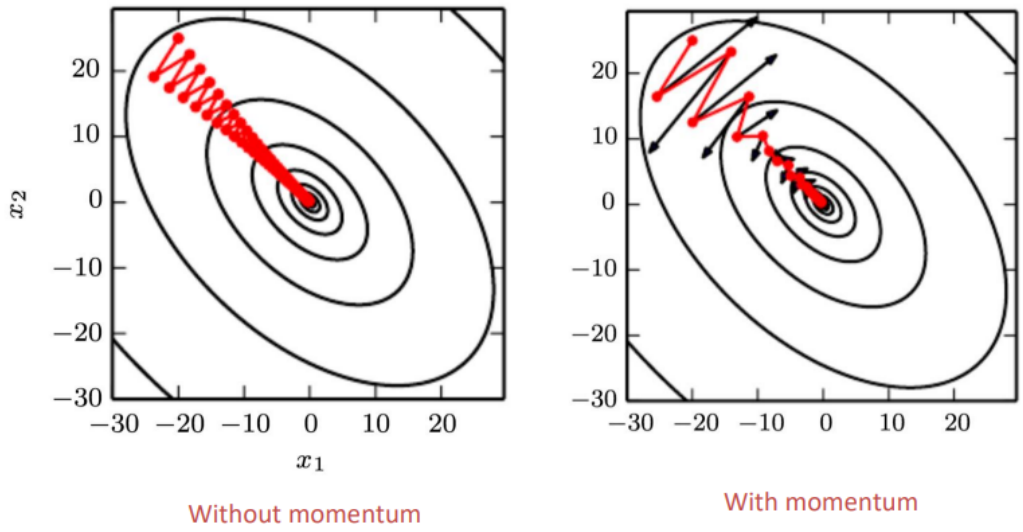
- $\epsilon_1, \epsilon_2, \ldots \epsilon_m$：学习率（范围为 mini-batch 的大小）
  - 当然可以设置 mini-batch 内所有的学习率都是一样的
- θ：权重参数（也就是之前的 w）
- $\mathbf{x}$：样本

- $\hat{\mathbf{g}}$：梯度

SGD 和传统 GD

- 当数据随时间变化时，SGD 工作得很好，而 GD 偏向于“过去”的样本
- SGD 可以选择信息量最大的样本
- SGD 还可以选择在前一个 epoch 中产生最大错误的样本，从而产生更大的梯度，从而更快地学习

带动量计算的 SGD



Without momentum

With momentum

- 类比于物理中的动量
- 继续考虑过去的梯度，但让它们的**贡献随时间呈指数衰减**
- 这是通过累积先前梯度值的**速度**参数来实现的
- 这抑制了振荡，产生了更稳健的梯度，进而导致更快的收敛

伪代码

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$.
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.
  end while

- α：描述动量的参数
  - 常见的取值为 0.5，0.9 或者 0.99
  - 通常它开始时的值很低，然后随时间增加
- $\mathbf{v}$：描述速度的参数

Nesterov 动量 SGD

- 就像标准的动量计算一样，但要使用未来梯度（这会产生更好的收敛）

Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

Require: Learning rate ϵ, momentum parameter α

Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding labels $\boldsymbol{y}^{(i)}$.

Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$.

Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$.

Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$.

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.

end while

二阶导优化

- 在求梯度（一阶导）的时候，上面三个图的梯度下降完全一样
- 到目前为止，我们看到的方法只依赖于一阶信息（梯度）
  - 梯度刻画的是方向
- 使用二阶导的信息（Hessian 矩阵）可以帮助更快的收敛
  - Hessian 刻画的是方向变化的快慢
- 主要基于牛顿法，最速下降

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

- $\boldsymbol{H}^{-1}$：Hessian 矩阵的逆
- $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$：梯度

Hessian 矩阵的计算

Hessian 是损失函数关于权值的所有组合的二重导数的矩阵

$$\frac{\partial^2}{\partial x_i \partial x_j} f(\boldsymbol{x})$$

$$\boldsymbol{H}(\boldsymbol{e}) = \begin{bmatrix} \frac{\partial^2 e}{\partial w_1^2} & \frac{\partial^2 e}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 e}{\partial w_1 \partial w_n} \\ \frac{\partial^2 e}{\partial w_2 \partial w_1} & \frac{\partial^2 e}{\partial w_2^2} & \cdots & \frac{\partial^2 e}{\partial w_2 \partial w_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 e}{\partial w_n \partial w_1} & \frac{\partial^2 e}{\partial w_n \partial w_2} & \cdots & \frac{\partial^2 e}{\partial w_n^2} \end{bmatrix}$$

- 如果 J 是局部二次的（ $\boldsymbol{H}$ 是正定的）那么我们可以直接跳到最小值
- 否则，重复二次逼近并更新参数
- 如果 $\boldsymbol{H}$ 有负特征值，我们可能会走向错误的方向

使用二阶导的问题

- 计算 Hessian 矩阵的逆是非常昂贵的
- $O(k^3)$：其中 k 是参数的个数

3. 学习率的优化方法

- 学习率对模型性能影响较大，难以确定
    - 损失函数可以在不同方向上以不同的方式变化
    - 动量有帮助，但它是另一个超参数
- 我们可以尝试学习每个参数的学习速率，并自动调整它们

Delta bar Delta

- 第一个启发式算法(1988)用于在训练过程中调整个体学习速率
- 如果损失相对于给定模型参数的偏导数的符号保持不变，那么学习率应该增加
- 如果它改变了，学习速度就会下降
- 只适用于批量梯度下降

AdaGrad

- 调整模型参数的学习速率，方法是将它们缩放成与梯度的所有过去的平方和的平方根成反比
- loss 的偏导数值越大的参数学习率越低
    - 在缓慢倾斜的方向上更快的进展
- 但长期的梯度历史会减慢速度
- Adagrad 的主要好处之一是，它消除了手动调整学习速率的需要
- Adagrad 的主要缺点是它在分母中积累了梯度的平方：由于每增加一项都是正的，所以在训练过程中积累的总和不断增加，这反过来又会导致学习速率下降，最终变得无穷小

伪代码

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, perhaps 10^{-7}, for numerical stability
Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$.
Compute update: $\Delta \boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$.
end while

RMSprop

- RMSprop 是一个(未发布的)修改使用指数移动平均线累积过去的梯度的 AdaGrad
- AdaGrad 设计用于凸函数，而 RMSprop 在非凸设置(典型的深度学习)中工作得更好

伪代码

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers
Initialize accumulation variables $\boldsymbol{r} = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$.
Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$.
end while

Adam

- 最好理解为将 RMSprop 与SGD + 动量结合起来的一种方式
- 使用平方梯度来缩放学习速率 RMSprop +动量梯度的移动平均
- 对超参数的选择是相当稳健的

伪代码

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)

Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)

Require: Small constant δ used for numerical stabilization (Suggested default: 10^{-8})

Require: Initial parameters $\boldsymbol{\theta}$

Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$

Initialize time step $t = 0$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

$t \leftarrow t + 1$

Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$

Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$

Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$

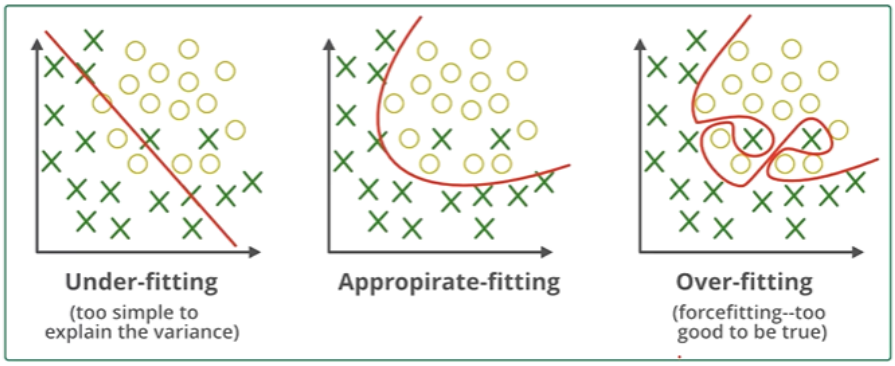
Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$

Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

4. 防止过拟合的操作



深度神经网络存在的问题

深度神经网络可能有数百万个参数，通常超过训练数据的大小，这可能导致过拟合

需要正则化损失函数

$$w^* = argmin \sum_{x,y} L(w_{1,\ldots,L}; x.y) + \lambda\Omega(\theta)$$

- $\lambda\Omega(\theta)$：只与 θ 有关，与样本 x 无关，直接对参数进行约束

可能的方法包括

- L1 正则化

- L2 正则化
- Dropout

目标是**减少模型容量 model capacity**

- 有的时候模型的搜索空间太大了，反而更容易过拟合

L1 正则化

- L1正则化强制稀疏权值
  - 很多权重变成 0
  - 删除一些连接
- 执行特征选择（我们应该保留输入数据的哪些特征？）

$$w^* = argmin \sum_{x,y} L(w_{1,...,L}; x.y) + \lambda \sum_l |w_l|$$

- 梯度更新公式：$w_l = w_l - \lambda\eta\frac{w_l}{|w_l|} - \eta\nabla_w L$
  - λ：正则化标量
  - η：学习率
  - $\frac{w_l}{|w_l|}$：它其实就只是一个 +1 或 -1 的符号

L2 正则化

- 最流行的回归类型，使权重更接近原点
- 很好的解析形式（可以推导并计算梯度）
- 通常 $\lambda = 10^{-1}/10^{-2}$

$$w^* = argmin \sum_{x,y} L(w_{1,...,L}; x.y) + \frac{\lambda}{2} \sum_l ||w_l||_2^2$$

- 梯度更新公式：$w_l = (1 - \lambda\eta)w_l - \eta_t\nabla_{w_l} L$
- 实际上，我们降低了与输出目标具有低协方差的特征（即不太重要的特征）的权重

数据增强

- 减少过拟合的另一种方法是拥有更多的训练数据，而不是减少模型的容量
- 这种方法特别容易应用于物体识别任务，我们可以获取输入数据并生成它的转换版本，模拟真实世界的场景（例如，遮挡，旋转等）

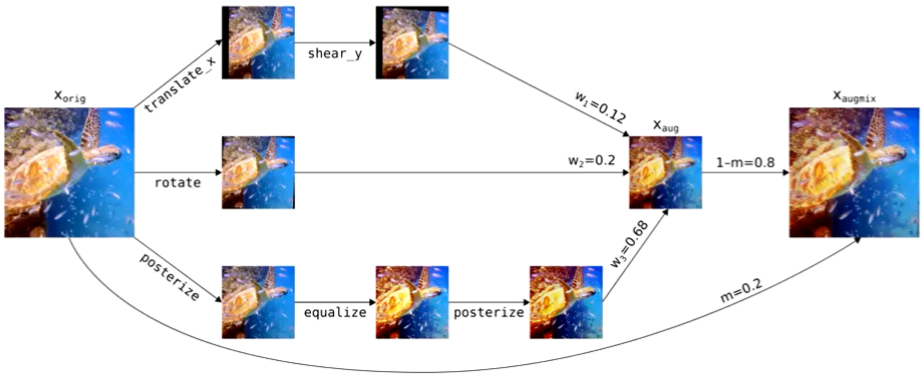
Mixup

$$\lambda \sim \text{Beta}(\alpha, \alpha)$$
$$\tilde{x} = \lambda x_i + (1 - \lambda) x_j$$
$$\tilde{y} = \lambda y_i + (1 - \lambda) y_j$$

```
# y1, y2 should be one-hot vectors
for (x1, y1), (x2, y2) in zip(loader1, loader2):
    lam = numpy.random.beta(alpha, alpha)
    x = Variable(lam * x1 + (1. - lam) * x2)
    y = Variable(lam * y1 + (1. - lam) * y2)
    optimizer.zero_grad()
    loss(net(x), y).backward()
    optimizer.step()
```

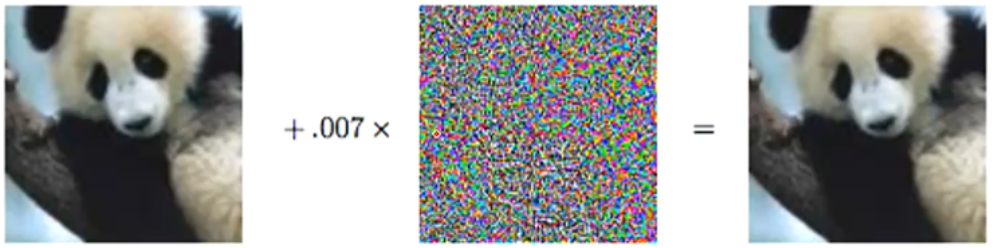
- Mixup 是一种简单的数据增强的方法
- 它通过线性插值构造**虚拟训练样本**
- 假设：增加 Mixup 插值的强度，生成的虚拟例子应该离训练的例子更远，使记忆更难实现。

AugMix



- AugMix 是一种数据增强方法，它随机采样各种增强操作后进行加权组合，允许我们探索原始图像周围语义等效的输入空间

对抗训练 Adversarial training



$\boldsymbol{x}$
"panda"
57.7% confidence

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

- 目前，提高对抗鲁棒性最有效的方法是对抗训练，即在标准训练中加入对抗例子。
- 白盒攻击：快速梯度符号法（FGSM）
  - $x^{adv} = x + \epsilon\, \mathbf{sign}(\nabla_x L(f(x), y))$

- x^{adv}：对抗训练生成的一个加入噪音的样本
- $\epsilon > 0$：控制扰动的大小

早些停止 Early Stopping

- 简单的思想：为了避免过拟合，当验证集的误差开始增加时停止训练，即使训练误差仍然在减少
- epoch 的数量可以看作是需要优化的超参数

Dropout

- 在每个 epoch，我们随机地“停用”一些单元，有效地训练基本网络的不同子网
- 在训练过程中，根据概率 p 随机“停用”一些单位
  - 有一定的概率不更新某一个参数
- 在测试时，使用所有的单元，它们的激活权重是 p
- 优点包括
  - 更快的训练
  - 更少的过拟合
  - 各个单元的鲁棒性更高

5. 预处理与参数初始化

除了选择优化方案和应用规则外，我们还可以使用其他标准做法来初始化网络参数和预处理输入数据

权重初始化

- 深度学习训练是迭代的，并且强烈依赖于初始化点（即，我们如何初始化网络参数）
- 重要的原则：**权重的非对称性**
  - 为什么？因为如果两个单元共享相同的激活、相同的权重和相同的输入，它们将以相同的方式更新（没有学习）
  - 所以，不要给所有权重都赋予相同的值（例如 0）
  - 权重由高斯或均匀抽样得出
- 但是，需要注意的是
  - **较大的权重**具有较强的对称破坏效应，会在正向和反向传播时传播较强的信号
  - 然而，它们也可能导致爆炸值，单位的饱和
  - 但是我们也不想要较小的权重
  - 此外，我们还希望保持输入和输出的方差相同（因为输出是下一层的输入）

均匀分布初始化 (tanh)

$$W_{ij} \sim U(-\sqrt{(\frac{6}{m+n})}, \sqrt{(\frac{6}{m+n})})$$

- 其中 m 是输入的个数，n 是输出的个数

Xavier 初始化 (tanh)

$$W_{ij} \sim N(0, \sqrt{\frac{1}{m}})$$

- 其中 m 是输入的个数，n 是输出的个数

均匀分布初始化 (sigmoid)

$$W_{ij} \sim U(-4\sqrt{(\frac{6}{m+n})}, 4\sqrt{(\frac{6}{m+n})})$$

- 其中 m 是输入的个数，n 是输出的个数

ReLU 初始化

$$W_{ij} \sim N(0, \sqrt{\frac{2}{m}})$$

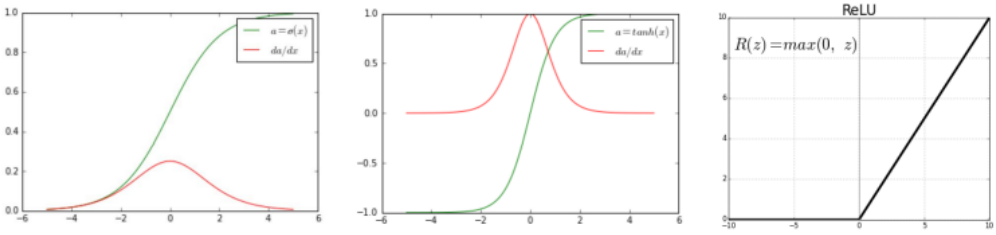
- 其中 m 是输入的个数，n 是输出的个数

预训练/ fine-tuning 工作

- 你有一个机器学习的模型 m
- 预训练：你有一个数据集 A，在这个数据集上你训练 m 来完成一些特定的任务
- 你有一个数据集 B，在你开始训练模型之前，你用 m 的一些参数来初始化你的模型

数据预处理

- 激活函数通常以零为中心



- 这是一件好事，因为它可以帮助我们避免饱和，而饱和度会导致**梯度消失**
- 同时，我们喜欢**单边饱和**，因为它有助于避免由于噪声而产生的方差
- 减去均值，训练数据也以零为中心
  - 否则可能导致梯度消失
- 对输入进行缩放，使其具有类似的对角线协方差
  - 否则，具有非常不同协方差的输入样本会产生非常不同的梯度，使梯度更新更加困难

单元标准化 Unit Normalization

当输入变量是正态分布时，减去均值除以标准差

批处理标准化 Batch Normalization

两个重要的原则，提供给网络各层的数据的分布应该是

- 以零为中心的
    - 我们已经解决了这个 问题
- 时间和数据不变（小批量）
    - 规范化每一层的激活！

$$
\begin{aligned}
\boldsymbol{Z} &= \boldsymbol{X}\boldsymbol{W} \\
\tilde{\boldsymbol{Z}} &= \boldsymbol{Z} - \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{Z}_{i,:} \\
\hat{\boldsymbol{Z}} &= \frac{\tilde{\boldsymbol{Z}}}{\sqrt{\epsilon + \frac{1}{m}\sum_{i=1}^{m} \tilde{\boldsymbol{Z}}_{i,:}^2}} \\
\boldsymbol{H} &= \max\{0, \gamma\hat{Z} + \boldsymbol{\beta}\}
\end{aligned}
$$