

人工神经网络的参数设置

胡浩基

浙江大学信息与电子工程学院

haoji_hu@zju.edu.cn



参数设置

1. 随机梯度下降
(Stochastic Gradient Descent, SGD)
2. 激活函数选择
3. 训练数据初始化
4. (W, b) 的初始化
5. Batch normalization
6. 目标函数选择
7. 参数更新策略
8. 训练建议

随机梯度下降

随机梯度下降

(1) 不用每输入一个样本就去变换参数，而是输入一批样本（叫做一个BATCH或MINI-BATCH），求出这些样本的梯度平均值后，根据这个平均值改变参数。

(2) 在神经网络训练中，BATCH的样本数大致设置为50-200不等。

随机梯度下降

```
batch_size = option.batch_size;
m = size(train_x,1);
num_batches = m / batch_size;
for k = 1 : iteration
    kk = randperm(m);
    for l = 1 : num_batches
        batch_x = train_x(kk((l - 1) * batch_size
+ 1 : l * batch_size), :);
        batch_y = train_y(kk((l - 1) * batch_size
+ 1 : l * batch_size), :);
        nn = nn_forward(nn,batch_x,batch_y);
        nn = nn_backpropagation(nn,batch_y);
        nn = nn_applygradient(nn);
    end
end
```

随机梯度下降

```
m = size(batch_x, 2);
```

前向计算

```
nn.cost(s) = 0.5 / m * sum(sum((nn.a{k} - batch_y).^2))  
+ 0.5 * nn.weight_decay * cost2;
```

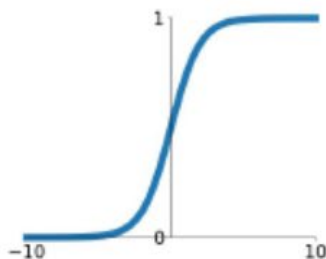
后向传播

```
nn.W_grad{nn.depth-1} =  
nn.theta{nn.depth}*nn.a{nn.depth-1}'/m +  
nn.weight_decay*nn.W{nn.depth-1};  
nn.b_grad{nn.depth-1} = sum(nn.theta{nn.depth}, 2)/m;
```

激活函数

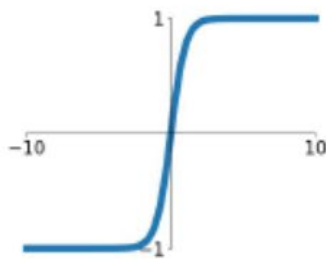
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



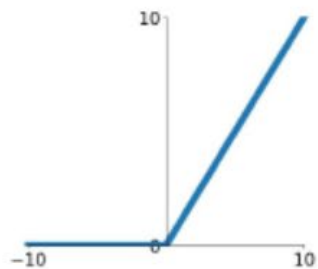
tanh

$$\tanh(x)$$



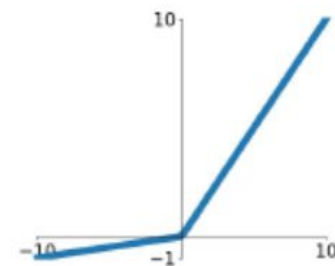
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

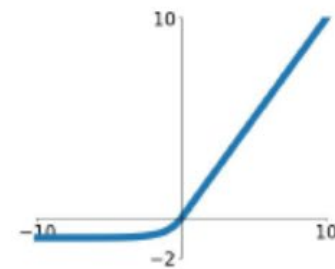


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

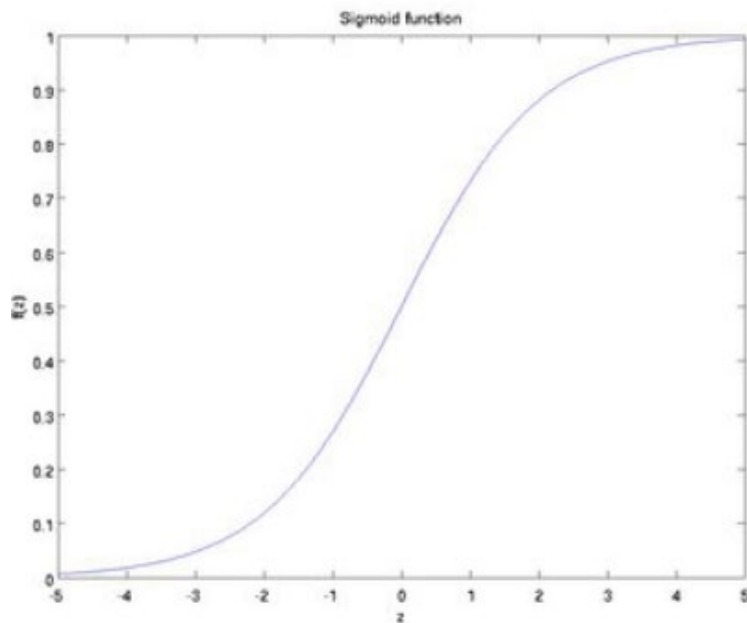
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



激活函数

常见非线性函数 $\varphi(x)$ 的选择。

(1) Sigmoid



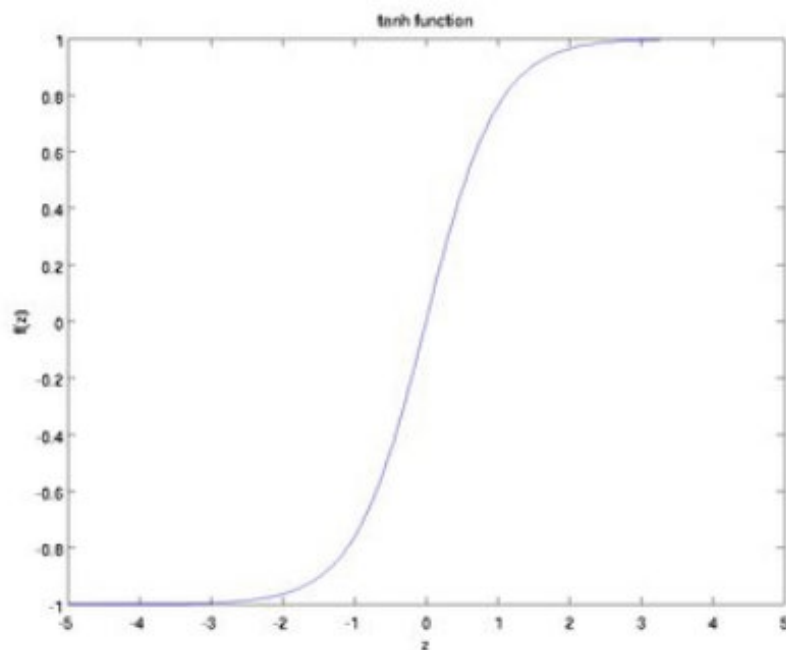
$$\varphi(x) = \frac{1}{1 + e^{-x}}$$

$$\varphi'(x) = \varphi(x)[1 - \varphi(x)]$$

激活函数

常见非线性函数 $\varphi(x)$ 的选择。

(2) tanh

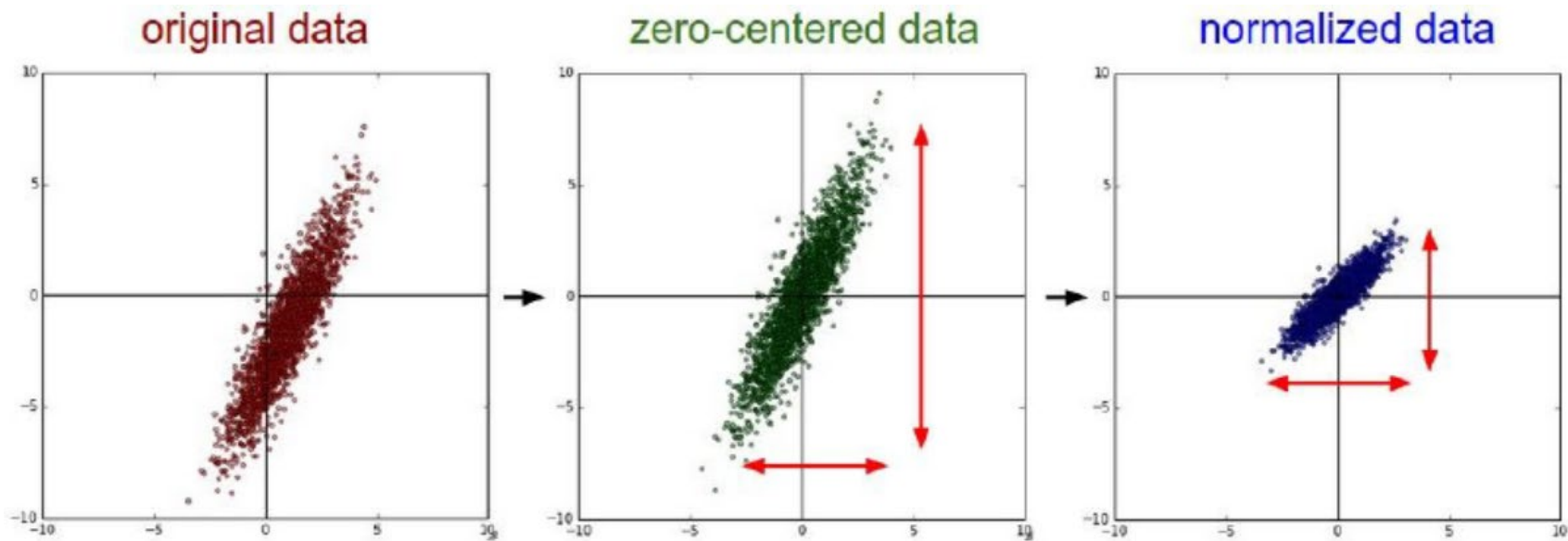


$$\varphi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\varphi'(x) = 1 - [\varphi(x)]^2$$

训练数据初始化

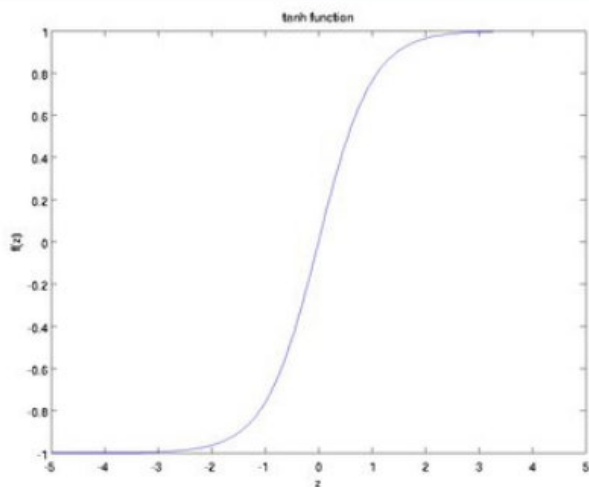
建议：做均值和方差归一化。。



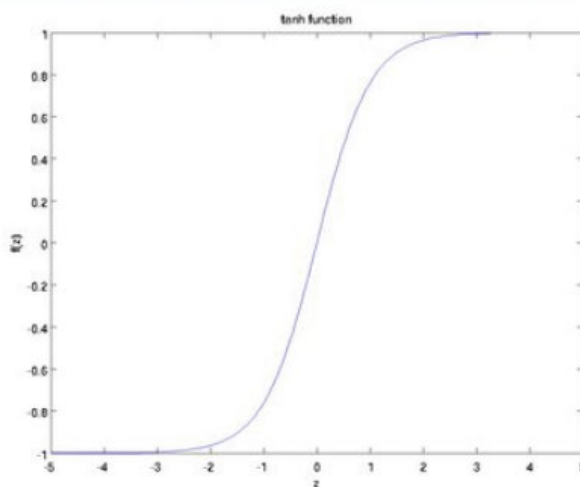
$$newX = \frac{X - mean(X)}{std(X)}$$

```
[U,V] = size(xTraining);  
avgX = mean(xTraining);  
sigma = std(xTraining);  
xTraining = (xTraining -  
repmat(avgX,U,1))./repmat(sigma,U,1);
```

(W,b) 的初始化



Sigmoid



tanh

梯度消失现象：如果 $W^T X + b$ 一开始很大或很小，那么梯度将趋近于0，反向传播后前面与之相关的梯度也趋近于0，导致训练缓慢。

因此，我们要使 $W^T X + b$ 一开始在零附近。

(W,b) 的初始化

一种比较简单有效的方法是：(W, b) 初始化从区间 $\left(-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}}\right)$

均匀随机取值。其中 d 为 (W, b) 所在层的神经元个数。

可以证明，如果 X 服从正态分布，均值0，方差1，且各个维度

无关，而 (W, b) 是 $\left(-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}}\right)$ 的均匀分布，则 $W^T X + b$ 是均值

为0，方差为1/3的正态分布。

```
nn.W{k} = 2*rand(height, width)/sqrt(width)-1/sqrt(width);
```

```
nn.b{k} = 2*rand(height, 1)/sqrt(width)-1/sqrt(width);
```

(W,b) 的初始化

参数初始化是一个热点领域，相关论文包括：

Understanding the difficulty of training deep feedforward neural networks
by Glorot and Bengio, 2010

Exact solutions to the nonlinear dynamics of learning in deep linear neural networks by
Saxe et al, 2013

Random walk initialization for training very deep feedforward networks by Sussillo and
Abbott, 2014

*Delving deep into rectifiers: Surpassing human-level performance on ImageNet
classification* by He et al., 2015

Data-dependent Initializations of Convolutional Neural Networks by Krähenbühl et al., 2015

All you need is a good init, Mishkin and Matas, 2015

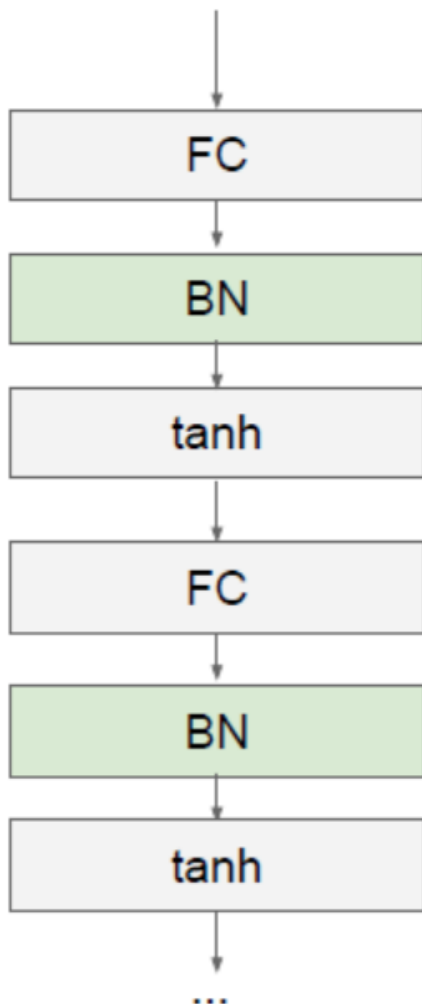
...

Batch Normalization

论文: Batch normalization accelerating deep network training by reducing internal covariate shift (2015)

基本思想: 既然我们希望每一层获得的值都在0附近, 从而避免梯度消失现象, 那么我们为什么不直接把每一层的值做基于均值和方差的归一化呢?

Batch Normalization



每一层FC (Fully Connected Layer)
接一个BN (Batch Normalization) 层

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

Batch Normalization

算法流程

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale and shift}$$

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

Batch Normalization

前向计算

```
y = nn.W{k-1} * nn.a{k-1} + repmat(nn.b{k-1},1,m);  
if nn.batch_normalization  
    nn.E{k-1} = nn.E{k-1}*nn.vecNum + sum(y,2);  
    nn.S{k-1} = nn.S{k-1}.^2*(nn.vecNum-1) + (m-  
1)*std(y,0,2).^2;  
    nn.vecNum = nn.vecNum + m;  
    nn.E{k-1} = nn.E{k-1}/nn.vecNum;  
    nn.S{k-1} = sqrt(nn.S{k-1}/(nn.vecNum-1));  
    y = (y - repmat(nn.E{k-1},1,m))./repmat(nn.S{k-  
1}+0.0001*ones(size(nn.S{k-1})),1,m);  
    y = nn.Gamma{k-1}*y+nn.Beta{k-1};  
end;
```


Batch Normalization

后向传播

```
nn.theta{k} = ((nn.W{k}'*nn.theta{k+1})) .* nn.a{k} .* (1
- nn.a{k});
if nn.batch_normalization
    x = nn.W{k-1} * nn.a{k-1} + repmat(nn.b{k-1},1,m);
    x = (x - repmat(nn.E{k-1},1,m))./repmat(nn.S{k-
1}+0.0001*ones(size(nn.S{k-1})),1,m);
    temp = nn.theta{k}.*x;
    nn.Gamma_grad{k-1} = sum(mean(temp,2));
    nn.Beta_grad{k-1} = sum(mean(nn.theta{k},2));
    nn.theta{k} = nn.Gamma{k-
1}*nn.theta{k}./repmat((nn.S{k-1}+0.0001),1,m);
end;
nn.W_grad{k-1} = nn.theta{k}*nn.a{k-1}'/m +
nn.weight_decay*nn.W{k-1};
nn.b_grad{k-1} = sum(nn.theta{k},2)/m;
```

目标函数选择

1. 可加正则项 (Regulation Term)

$$L(W) = F(W) + R(W)$$

$$= \frac{1}{2} \left(\sum_{i=1}^{batch_size} \|y_i - Y_i\|^2 + \beta \sum_k \sum_l W_{k,l}^2 \right)$$

前向计算

```
cost2 = cost2 + sum(sum(nn.W{k-1}.^2));  
nn.cost(s) = 0.5 / m * sum(sum((nn.a{k} - batch_y).^2))  
+ 0.5 * nn.weight_decay * cost2;
```

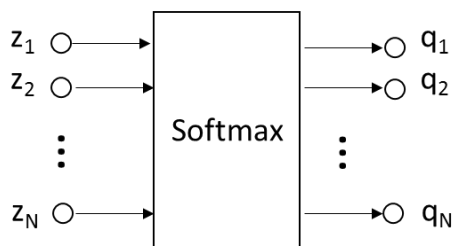
后向传播

```
nn.W_grad{k-1} = nn.theta{k}*nn.a{k-1}'/m +  
nn.weight_decay*nn.W{k-1};
```

目标函数选择

2. 如果是分类问题，F(W)可以采用SOFTMAX函数和交叉熵的组合。

(a) SOFTMAX函数：



$$q_i = \frac{\exp(z_i)}{\sum_{j=1}^N \exp(z_j)}$$

我们希望通过此网络学习由 $Z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_N \end{bmatrix}$ 到 $P = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_N \end{bmatrix}$ 的映射。其中 $\sum_{i=1}^N p_i = 1$

(b) 交叉熵 (Cross Entropy) :

$$\text{定义目标函数为 } E = - \sum_{i=1}^N p_i * \log(q_i)$$

目标函数选择

如果F(W)是SOFTMAX函数和交叉熵的组合，那么求导将会有非常简单形式：

$$\frac{\partial E}{\partial z_i} = q_i - p_i$$

前向计算

```
if strcmp(nn.objective_function, 'Cross Entropy')
    nn.cost(s) = -0.5*sum(sum(batch_y.*log(nn.a{k}))) / m
+ 0.5 * nn.weight_decay * cost2;
```

后向传播

```
case 'softmax'
    y = nn.W{nn.depth-1} * nn.a{nn.depth-1} +
repmat(nn.b{nn.depth-1}, 1, m);
    nn.theta{nn.depth} = nn.a{nn.depth} - batch_y;
```

参数更新策略

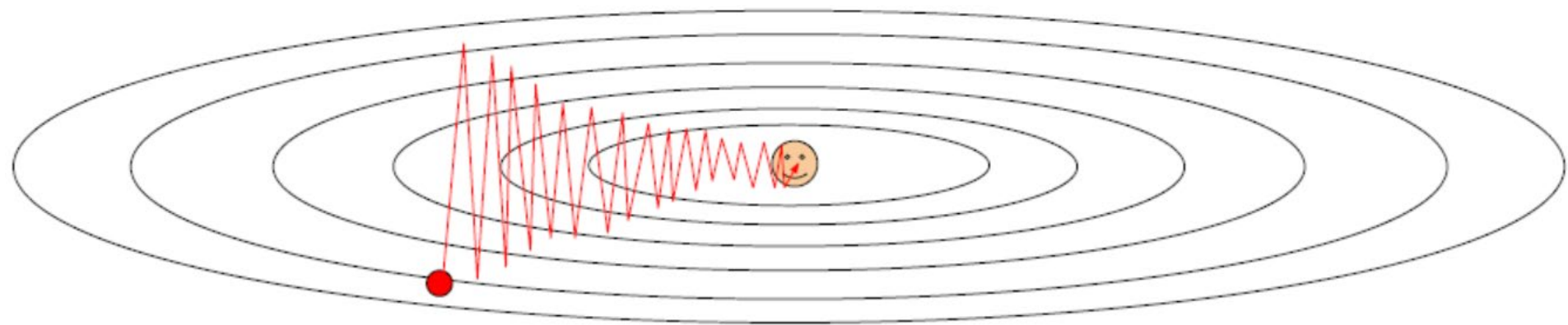
(1) 常规的更新 (Vanilla Stochastic Gradient Descent)

```
nn.W{k} = nn.W{k} - nn.learning_rate*nn.W_grad{k};  
nn.b{k} = nn.b{k} - nn.learning_rate*nn.b_grad{k};
```

参数更新策略

SGD的问题

(1) (W, b) 的每一个分量获得的梯度绝对值有大有小，一些情况下，将会迫使优化路径变成Z字形状。



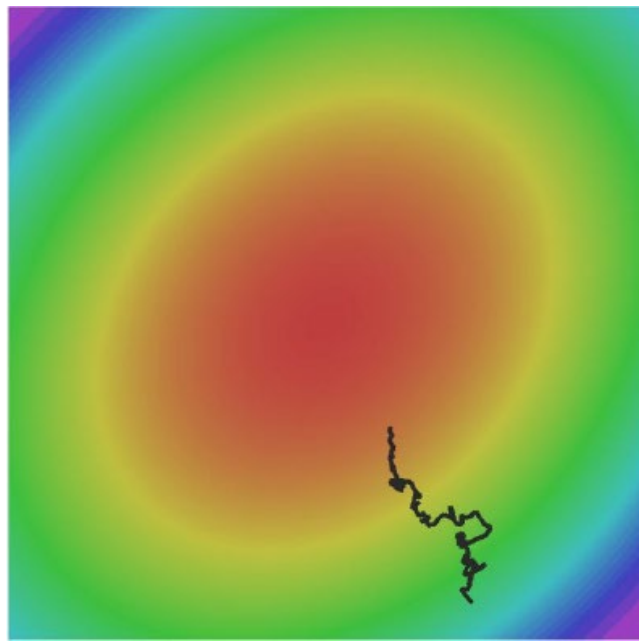
参数更新策略

SGD的问题

(2) SGD求梯度的策略过于随机，由于上一次和下一次用的是完全不同的BATCH数据，将会出现优化的方向随机的情况。

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^N \nabla_W L_i(x_i, y_i, W)$$



参数更新策略

解决各个方向梯度不一致的方法：

(1) AdaGrad

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ

Require: Initial parameter θ

Require: Small constant δ , perhaps 10^{-7} , for numerical stability

Initialize gradient accumulation variable $\mathbf{r} = \mathbf{0}$

while stopping criterion not met **do**

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

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

 Accumulate squared gradient: $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g}$

 Compute update: $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \mathbf{g}$. (Division and square root applied element-wise)

 Apply update: $\theta \leftarrow \theta + \Delta\theta$

end while

参数更新策略

解决各个方向梯度不一致的方法：

(1) AdaGrad

```
if strcmp(nn.optimization_method, 'AdaGrad')

nn.rW{k} = nn.rW{k} + nn.W_grad{k}.^2;
nn.rb{k} = nn.rb{k} + nn.b_grad{k}.^2;

nn.W{k} = nn.W{k} -
nn.learning_rate*nn.W_grad{k} ./ (sqrt(nn.rW{k})+0.001);

nn.b{k} = nn.b{k} -
nn.learning_rate*nn.b_grad{k} ./ (sqrt(nn.rb{k})+0.001);
```

参数更新策略

解决各个方向梯度不一致的方法：

(2) RMSProp

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ , decay rate ρ .

Require: Initial parameter θ

Require: Small constant δ , usually 10^{-6} , used to stabilize division by small numbers.

Initialize accumulation variables $r = 0$

while stopping criterion not met **do**

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

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

 Accumulate squared gradient: $r \leftarrow \rho r + (1 - \rho) g \odot g$

 Compute parameter update: $\Delta\theta = -\frac{\epsilon}{\sqrt{\delta + r}} \odot g$. ($\frac{1}{\sqrt{\delta + r}}$ applied element-wise)

 Apply update: $\theta \leftarrow \theta + \Delta\theta$

end while

参数更新策略

解决各个方向梯度不一致的方法：

(2) RMSProp

```
elseif strcmp(nn.optimization_method, 'RMSProp')
    rho = 0.9; %rho=0.9
    nn.rW{k} = rho*nn.rW{k} + 0.1*nn.W_grad{k}.^2;
    nn.rb{k} = rho*nn.rb{k} + 0.1*nn.b_grad{k}.^2;

    nn.W{k} = nn.W{k} -
nn.learning_rate*nn.W_grad{k} ./ (sqrt(nn.rW{k})+0.001);

    nn.b{k} = nn.b{k} -
nn.learning_rate*nn.b_grad{k} ./ (sqrt(nn.rb{k})+0.001);
    %rho = 0.9
```

参数更新策略

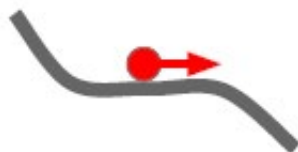
解决梯度随机性问题：

(3) Momentum

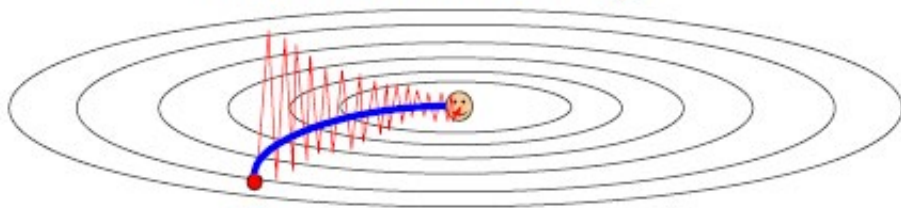
Local Minima



Saddle points



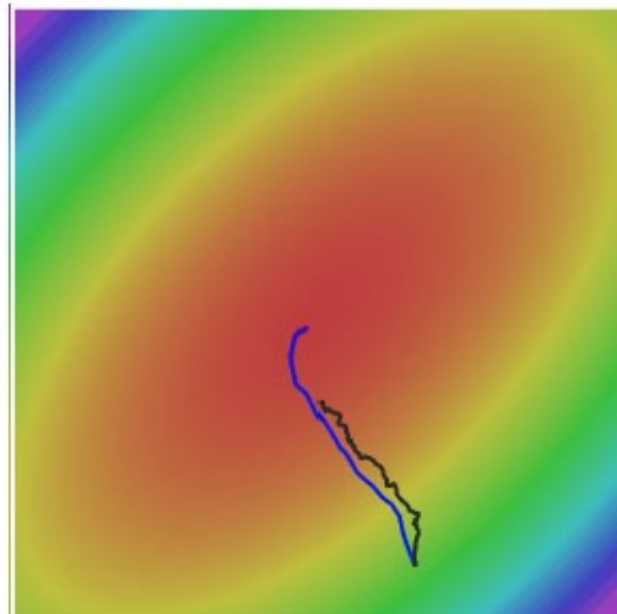
Poor Conditioning



SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$



参数更新策略

解决梯度随机性问题：

(3) Momentum

```
if strcmp(nn.optimization_method, 'Momentum')
nn.vW{k} = 0.5*nn.vW{k} + nn.learning_rate*nn.W_grad{k};

nn.vb{k} = 0.5*nn.vb{k} + nn.learning_rate*nn.b_grad{k};

nn.W{k} = nn.W{k} - nn.vW{k};
nn.b{k} = nn.b{k} - nn.vb{k}; %rho = 0.5;
```

参数更新策略

同时两个问题：

(4) Adam

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 θ

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

Initialize time step $t = 0$

while stopping criterion not met **do**

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

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

$t \leftarrow t + 1$

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

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

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

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

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

Apply update: $\theta \leftarrow \theta + \Delta \theta$

end while

参数更新策略

同时两个问题：

(4) Adam

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 θ

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

Initialize time step $t = 0$

while stopping criterion not met **do**

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

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

$t \leftarrow t + 1$

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

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

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

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

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

Apply update: $\theta \leftarrow \theta + \Delta \theta$

end while

参数更新策略

同时两个问题：

(4) Adam

```
if strcmp(nn.optimization_method, 'Adam')
nn.sW{k} = 0.9*nn.sW{k} + 0.1*nn.W_grad{k};
nn.sb{k} = 0.9*nn.sb{k} + 0.1*nn.b_grad{k};
nn.rW{k} = 0.999*nn.rW{k} + 0.001*nn.W_grad{k}.^2;
nn.rb{k} = 0.999*nn.rb{k} + 0.001*nn.b_grad{k}.^2;

nn.W{k} = nn.W{k} -
10*nn.learning_rate*nn.sW{k}./sqrt(1000*nn.rW{k}+0.00001);

nn.b{k} = nn.b{k} -
10*nn.learning_rate*nn.sb{k}./sqrt(1000*nn.rb{k}+0.00001);
%rho1 = 0.9, rho2 = 0.999, delta = 0.00001
```


训练建议

(1) 一般情况下，在训练集上的目标函数的平均值（cost）会随着训练的深入而不断减小，如果这个指标有增大情况，停下来。有两种情况：第一是采用的模型不够复杂，以致于不能在训练集上完全拟合；第二是已经训练很好了。

(2) 分出一些验证集（Validation Set），训练的本质目标是在验证集上获取最大的识别率。因此训练一段时间后，必须在验证集上测试识别率，保存使验证集上识别率最大的模型参数，作为最后结果。

(3) 注意调整学习率（Learning Rate），如果刚训练几步cost就增加，一般来说是学习率太高了；如果每次cost变化很小，说明学习率太低。

训练建议

(4) Batch Normalization 比较好用，用了这个后，对学习率、参数更新策略等不敏感。建议如果用Batch Normalization，更新策略用最简单的SGD即可，我的经验是加上其他反而不好。

(5) 如果不用Batch Normalization，我的经验是，合理变换其他参数组合，也可以达到目的。

(6) 由于梯度累积效应，AdaGrad, RMSProp, Adam三种更新策略到了训练的后期的会很慢，可以采用提高学习率的策略来补偿这一效应。



浙江大学
ZheJiang University



**Thank you and comments
are welcomed**