

深度学习的实践

1. 训练、验证、测试集

Data:

training	Validation	test
----------	------------	------

training Validation test

小数据时代: 70% training / 30% test , 60% training / 20% Validation / 20% test

大数据时代: training set 占更大的比重, 比如 98% / 1% / 1%

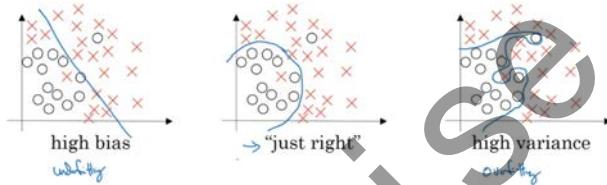
Guideline:

- 验证 Validation 与 test 是否符合同一分布, 以加快 ML 算法的速度

- 如果不需要使用无偏估计来评估 model, 则可以不需要 test

2. 偏差与方差

Bias and Variance

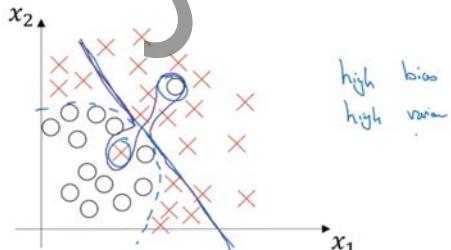


underfitting → high bias

overfitting → high variance

在 training 上训练, 最小化 training fit bias, 避免出现 underfitting, 但模型太复杂可能导致在 training 上表现良好而在 dev 上出现 high variance (BP overfitting).

High bias and high variance 的情况



在高维的情况下会出现

3. 机器学习基本方法

出现 high bias:

- 新的网络架构(选择更深的网络)
- 训练更长的时间
- 选择更加先进的优化算法

出现 high variance:

- 使用更多的数据
- 正则化
- 寻找合适的网络结构

4. 正则化

$$\text{Logistic regression: } J(w, b) = \frac{1}{m} \sum_{i=1}^m l(y^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \|w\|_2^2$$

$$L_2 \text{ 正则化 } \|w\|_2^2 = \sum_{j=1}^m w_j^2 = w^T w$$

$$L_1 \text{ 正则化 } \frac{\lambda}{m} \|w\|_1 = \frac{\lambda}{m} \sum_{j=1}^m |w_j|$$

$$\text{Neural network: } J(W^{(0)}, b^{(0)}, \dots, W^{(L)}, b^{(L)}) = \frac{1}{m} \sum_{i=1}^m l(y^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^L \|W^{(l)}\|_F^2 \quad \text{Frobenius norm}$$

$$\|W^{(l)}\|_F^2 = \sum_{i,j} W_{ij}^{(l)2} \quad W^{(l)} = (n^{(l)}, n^{(l-1)})$$

Weight decay:

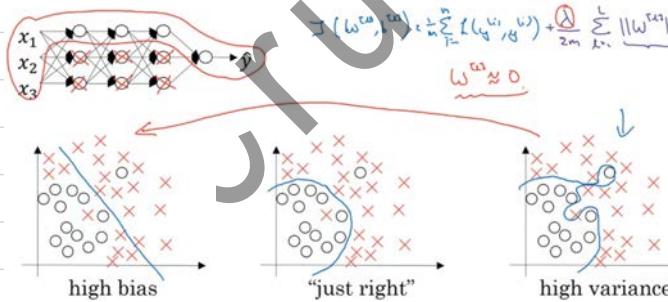
$$dW^{(l)} = (\text{form-backprop}) + \frac{\lambda}{m} W^{(l)}$$

$$W^{(l)} := W^{(l)} - \alpha dW^{(l)} = W^{(l)} - \alpha \left[(\text{form-backprop}) + \frac{\lambda}{m} W^{(l)} \right]$$

$$= \left(1 - \frac{\alpha \lambda}{m}\right) W^{(l)} - \alpha (\text{form-backprop})$$

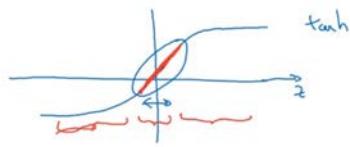
(小于 1, 使得 $W^{(l)}$ 不断衰减)

5. 为什么正则化有效?



直观看上, 当 λ 越大时, 为了最小化 cost function, W 会被设置地 $\rightarrow 0$, 相当于消除了部分神经元的影响。从而使得大的网络变为小的网络。但其实这些神经元还在, 只是影响变小了, 避免过拟合。

数学上, 以 $g(z) = \tanh(z)$ 为例



$g(z) = \tanh(z)$ 入越大, $W^{[l+1]}$ 越小。
在 z 较小的区域 $g(z)$ 几乎呈线性, 简化了网络结构。

$$X \uparrow \quad W^{[l+1]} \downarrow \quad z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l]}$$

Every layer \approx linear

6. Dropout 正则化

为每个神经元结点设置一个保留的概率, 这样保留下来的节点将构成一个较小的网络。

Implementing dropout — Inverted dropout:

对于 $layer=3$ 的 NN $keep_prob = 0.8$

$$d_3 = np.random.rand(a_3.shape[0], a_3.shape[1]) < keep_prob$$

$$a_3 = a_3 \cdot d_3$$

$a_3 /= keep_prob$ ← 有 20% 的神经元消失, 为了使下一层计算的期望不受影响。

notation: 在 test 阶段一般不使用 dropout, 否则预测结果会受到干扰。

dropout 的设置: 与 L2 范数一样, dropout 拥有收缩权重的效果。对于神经元较少的 layer, $keep_prob=1$; 对于神经元较多的 layer, $keep_prob$ 设置的较小。

dropout 缺点: 使 cost function 不再被明确地定义。因为每次都会删除一些节点, 不能定义 J 每次迭代后都会下降。

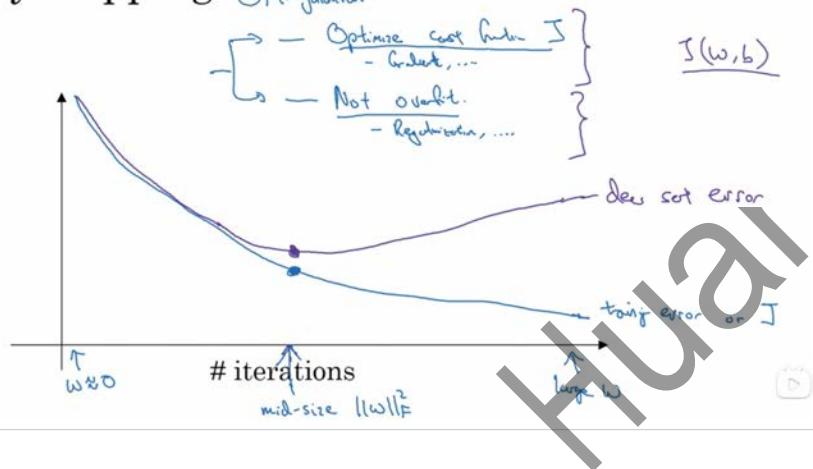
正确地使用 dropout:

- 关闭 dropout (设置 $keep_prob=1$)
- 运行代码, 确保 J 单调递减
- 打开 dropout

7. 其他避免 overfitting 的方法

- Data augmentation: 进行图像变换，获得更大的数据集。
- Early stopping: 在 dev 上的误差上升之前便停止迭代。缺点是不能在 bias 和 variance 上同时找到最优解。

Early stopping (Orthogonalization)

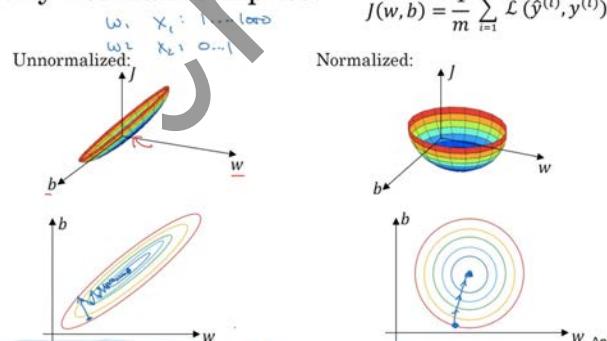


8. 归一化输入

- 计算每个特征在所有样本上的均值 $\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$
- $X := X - \mu$
- 归一化方差 $\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)^2$, $X = X / \sigma$

为什么 normalize?

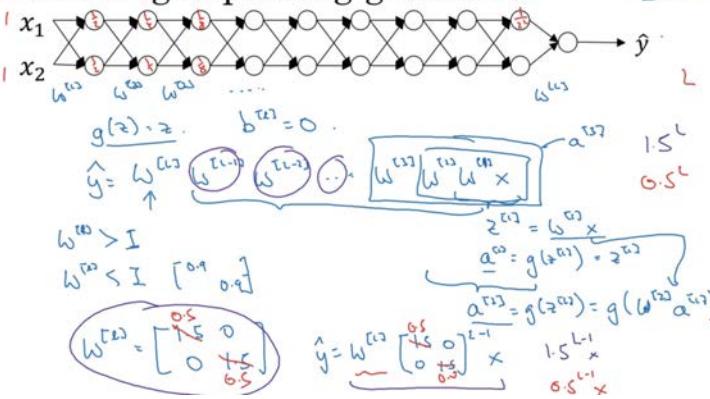
Why normalize inputs?



无论从什么地方开始迭代，都能更快地达到 optima

9. 梯度消失与梯度爆炸

Vanishing/exploding gradients



假设 $g(z) = z$, $b^{L2} = 0$,

$$\text{则 } \hat{y} = W^{L3} W^{L2} \dots W^{L1} W^{L0} x$$

· 如果 $W^{L0} > I$, 则 $\begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$

$$\text{则 } \hat{y} = W^{L3} \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{L-1} x, \text{ 指数级上升。}$$

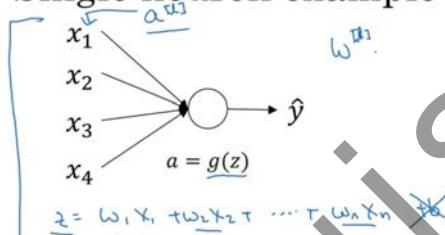
· 如果 $W^{L0} < I$, 则 $\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$,

$$\text{则 } \hat{y} = W^{L3} \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}^{L-1} x, \text{ 指数级下降。}$$

10. 初始化 W 解决梯度消失和梯度爆炸

(Xavier Initialization)

Single neuron example



当 n 较大时, 为了使得 z 较小, 则 W_i 应该也设置地小。

可以设置 $\text{Var}(W_i) = \frac{1}{n}$:

$$W^{L1} = np.random.randn(W^{L1}.shape) * np.sqrt(\frac{1}{n})$$

当使用 ReLU 时, $\text{Var}(W_i) = \frac{2}{n}$:

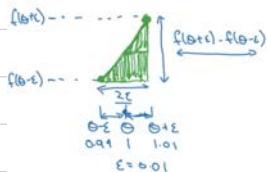
$$W^{L1} = np.random.randn(W^{L1}.shape) * np.sqrt(\frac{2}{n})$$

如果 x 的均值为 0, 标准方差为 1, 输出的 z 也会调整到相应范围。

11. 梯度的数值逼近

Checking your derivative computation

$$\underline{f'(\theta)} = \theta^2$$



$$\frac{f(\theta+\epsilon) - f(\theta-\epsilon)}{2\epsilon} \approx g(\theta)$$

$$\frac{(1.01)^2 - (0.99)^2}{2(0.01)} = 3.0001 \approx 3$$

$$g(\theta) = 3\theta^2 = 3$$

$$\text{Approx error: } 0.0001 \\ (\text{true slide: } 3.0301, \text{ error: } 0.03)$$

双边导数:

$$f'(0) = \lim_{\epsilon \rightarrow 0} \frac{f(0+\epsilon) - f(0-\epsilon)}{2\epsilon}$$

误差为 $O(\epsilon^2)$

单边导数:

$$f'(0) = \lim_{\epsilon \rightarrow 0} \frac{f(0+\epsilon) - f(0)}{\epsilon}$$

误差为 $O(\epsilon)$

12. 梯度检验

将所有的参数 $W^{[0]}, b^{[0]}, W^{[1]}, \dots, W^{[L]}, b^{[L]}$ 连接成一个大的 vector:

$$J(W^{[0]}, b^{[0]}, \dots) = J(\theta)$$

对于 $dW^{[0]}, db^{[0]}, \dots, dW^{[L]}, db^{[L]}$, 同样操作。

之后便进行 gradient checking:

$$d\theta_{approx}[i] = \frac{J(\theta_1, \theta_2, \dots, \theta_i + \varepsilon, \dots) - J(\theta_1, \theta_2, \dots, \theta_i - \varepsilon, \dots)}{2\varepsilon}$$

$$d\theta[i] = \partial J / \partial (\theta_i)$$

利用公式 $\frac{\|d\theta_{approx} - d\theta\|_2}{\|d\theta_{approx}\|_2 + \|d\theta\|_2}$ 检验梯度

$$\approx 10^{-7}$$

↑

$$\approx 10^{-3}$$

↓

$\|\cdot\|_2$ 为欧几里得范数, 平方和再开根号

13. 梯度检验 notes

- 不要在 training 中使用 gradient checking, 只在 debug 中使用。
- 如果 gradient checking 有错误, 要逐项检查(找到哪个 $d\theta_{approx}[i]$ 与 $d\theta$ 的值相差较大)。
- 在 gradient checking 时, 不要忘了 regularization 项。
- 不能与 dropout 同时使用。

优化算法

1. Mini-batch gradient descent

$X: (n_x, m) \quad y: (1, m) \quad m = 5000000 \rightarrow 5000 \text{ mini batch (1000 样本)}$

$$X = \underbrace{[x^{(1)} \ x^{(2)} \ x^{(3)} \dots x^{(1000)}]}_{X^{(1)}: (n_x, 1000)} \ | \ \underbrace{x^{(1001)} \dots x^{(2000)}}_{x^{(2)}} \ | \ \dots \ | \ \dots x^{(m)}$$

mini batch $t: X^{(t)}, y^{(t)}$

$$y = \underbrace{[y^{(1)} \ y^{(2)} \dots y^{(1000)}]}_{y^{(1)}} \ | \ \underbrace{y^{(1001)} \dots y^{(2000)}}_{y^{(2)}} \ | \ \dots \ | \ \dots y^{(m)}$$

Mini-batch gradient descent

repeat $\frac{1}{2}$ for $t = 1, \dots, 5000 \ \{$

Forward prop on $X^{(t)}$.

$$\begin{aligned} z^{(t)} &= w^{(t)} X^{(t)} + b^{(t)} \\ A^{(t)} &= g^{(t)}(z^{(t)}) \end{aligned}$$

backward propagation (1000 examples)

$$\text{Compute cost } J^{(t)} = \frac{1}{1000} \sum_{i=1}^{\infty} l(y^{(i)}, \hat{y}^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum_{j} \|w^{(t)}\|_F^2.$$

Backprop to compute gradients w.r.t $J^{(t)}$ (using $(X^{(t)}, y^{(t)})$)

$$w^{(t+1)} = w^{(t)} - \alpha \nabla J^{(t)}, \quad b^{(t+1)} = b^{(t)} - \alpha \nabla b^{(t)}$$

3

"1 epoch"
→ pass through training set.

1 step of gradient descent
普通的 gradient descent 中,
用 $X^{(t)}, y^{(t)}$
(一个样本)

→ 一个 epoch 只能进行一次
gradient descent;

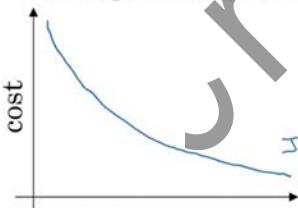
X, Y

mini-batch gradient descent

→ 一个 epoch 可以进行 5000 次。

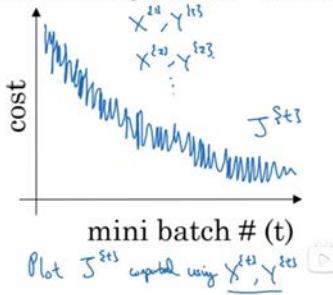
Training with mini batch gradient descent

Batch gradient descent



iterations
每次迭代后 Cost function 变小
花费时间长

Mini-batch gradient descent



Cost function 的总体趋势是下降，呈波动式

mini-batch size = m : Batch gradient descent

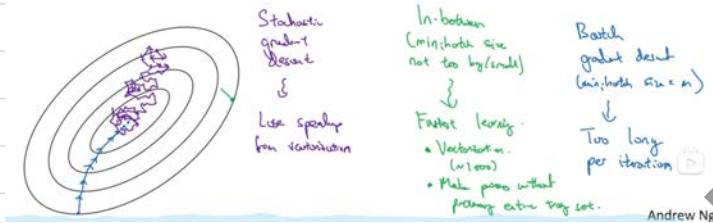
mini-batch size = 1 : Stochastic gradient descent

Choosing your mini-batch size

→ If mini-batch size = m : Batch gradient descent. $(X^{(1)}, Y^{(1)}) = (X, Y)$.

→ If mini-batch size = 1 : Stochastic gradient descent. Every example is it own
 $(X^{(1)}, Y^{(1)}) = (x^{(1)}, y^{(1)}) \dots (x^{(n)}, y^{(n)})$ mini-batch.

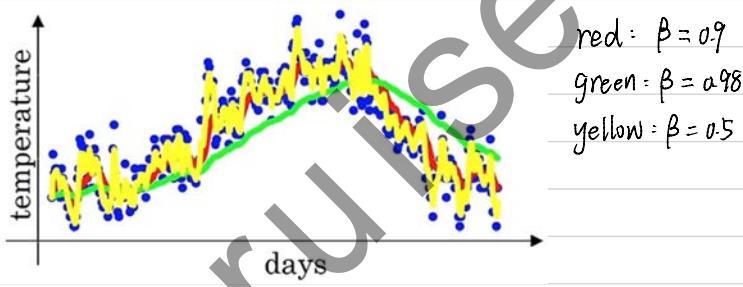
In practice: Some in-between 1 and m



2. 指数加权平均

$$V_t = \beta V_{t-1} + (1-\beta) \theta_t$$

β 昨天 \uparrow 去年的今天



当 $\beta = 0.9$ 时,

$$V_{100} = 0.9 V_{99} + 0.1 \theta_{100}, \quad V_{99} = 0.9 V_{98} + 0.1 \theta_{99}, \quad V_{98} = 0.9 V_{97} + 0.1 \theta_{98}, \dots$$

$$\downarrow \quad V_{100} = 0.9(0.9 V_{98} + 0.1 \theta_{99}) + 0.1 \theta_{100} = \dots = 0.1 \theta_{100} + 0.1 \times 0.9 \theta_{99} + 0.1 \times 0.9^2 \theta_{98} + \dots$$

目前的系数和 ≈ 1 . 称为偏差修正

总体上说, 存在 $\lim_{\epsilon \rightarrow 0} (1-\epsilon)^{\frac{1}{\epsilon}} = \frac{1}{e}$ 。在这里 ϵ 取 0.1, $0.9^{10} \approx 0.35 \approx \frac{1}{e}$

相当于只关注了过去 10 天的天气 (10 天下降到原来的 $\frac{1}{e}$)

指教加权平均的实现：

$$V_0 = 0$$

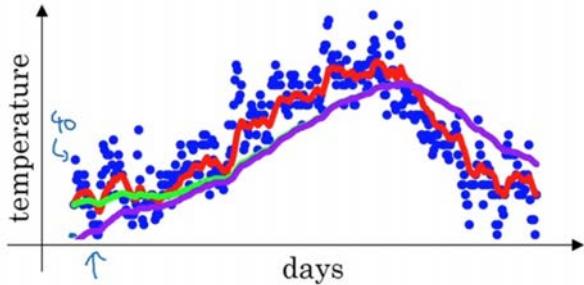
$$V_1 = \beta V_0 + (1 - \beta) \theta_1$$

$$V_2 = \beta V_1 + (1 - \beta) \theta_2$$

$$V_3 = \beta V_2 + (1 - \beta) \theta_3$$

...

} 只需要当前时刻的值与前一天
的均值



当 $\beta = 0.98$ 时，修正曲线其实为紫色。

在初始时值会较低（因为 $V_0 = 0$ ）

$$\text{解决方法: } \frac{V_t}{1 - \beta^t} \quad (+ \text{为天数})$$

$$\text{当 } t=2 \text{ 时, } 1 - 0.98^2 = 0.0396$$

$$\frac{V_2}{0.0396} = \frac{0.0196\theta_1 + 0.02\theta_2}{0.0396}$$

这样便使得在初期时紫绿两线接近。当 t 增大时， $1 - \beta^t \rightarrow 1$ ，此时两条线趋于重合。

3. Momentum 梯度下降

基本思想：计算梯度的指教加权平均数。

效果：更快地达到 Cost function 的最小值。

On iteration t :

Compute dW, db on the current mini-batch

$$v_{dw} = \beta v_{dw} + (1 - \beta) dW$$

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

$$W = W - \alpha v_{dw}, \quad b = b - \alpha v_{db}$$

β 一般取 0.9

由于使得指教加权平均方法，纵轴方向的变化减小，横轴上的平均值较大。

4. RMSprop

On iteration t :

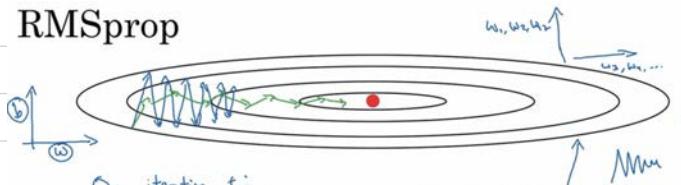
Compute dW, db on current mini-batch

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

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

$$\begin{aligned} W &:= W - \alpha \frac{dW}{\sqrt{S_{dw}}} \\ b &:= b - \alpha \frac{db}{\sqrt{S_{db}}} \end{aligned}$$

RMSprop



假设 b 的梯度为纵向, W 的

梯度为横向。

\leftrightarrow 使用 RMSprop, 可以减小梯度更新波动较大的情况。

On iteration t :

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) \underline{dW}^2 \quad \leftarrow \text{small}$$

$$\rightarrow S_{db} = \beta_2 S_{db} + (1-\beta_2) \underline{db}^2 \quad \leftarrow \text{large}$$

$$w := w - \frac{\alpha}{\sqrt{S_{dw}}} \underline{dW} \quad \leftarrow$$

$$b := b - \frac{\alpha}{\sqrt{S_{db}}} \underline{db} \quad \leftarrow$$

$\epsilon = 10^{-8}$, 避免分母为 0

5. Adam

基本思想: Momentum + RMSprop

Input: $V_{dw} = 0$, $S_{dw} = 0$, $V_{db} = 0$, $S_{db} = 0$

On iteration t :

Compute dW, db using current mini-batch

$$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dW, \quad V_{db} = \beta_1 V_{db} + (1-\beta_1) db \quad \leftarrow \text{Momentum}$$

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) (dW)^2, \quad S_{db} = \beta_2 S_{db} + (1-\beta_2) (db)^2 \quad \leftarrow \text{RMSprop}$$

偏差修正:

$$V_{dw}^{\text{corrected}} = V_{dw} / (1 - \beta_1^t), \quad V_{db}^{\text{corrected}} = V_{db} / (1 - \beta_1^t)$$

$$S_{dw}^{\text{corrected}} = S_{dw} / (1 - \beta_2^t), \quad S_{db}^{\text{corrected}} = S_{db} / (1 - \beta_2^t)$$

$$W := W - \alpha \frac{V_{dw}^{\text{corrected}}}{\sqrt{S_{dw}^{\text{corrected}}} + \epsilon}, \quad b := b - \alpha \frac{V_{db}^{\text{corrected}}}{\sqrt{S_{db}^{\text{corrected}}} + \epsilon}$$

超参数:

α : 需要调试

β_1 : default 0.9

β_2 : default 0.999

ϵ : recommended 10^{-8}

6. 学习率衰减

原因: 最初时, 较大的 α 能保证相对快速地收敛, 但随着学习的深入, 很难找到精确的最小值。因此逐渐减小 α , 使得结果会在极小值周围的更小的区域中波动。

常用的 decay function:

- 常用: $\alpha = \frac{1}{1 + \text{decay_rate} * \text{epoch_num}} \alpha_0$
- 指数衰减: $\alpha = \alpha_0 e^{-\frac{\text{decay_rate}}{\text{epoch_num}}} \alpha_0$
- $\alpha = \frac{k}{\sqrt{\text{epoch_num}}} \alpha_0 \quad \text{or} \quad \frac{k}{\sqrt{t}} \alpha_0$
- 高散值

7. Local optima in Neural Network

低维情况下容易发生 Local optima; 而在高维时, 只有当所有维度均为凸函数时才会发生 local optima, 概率极低。高维度下梯度为 0 的点有可能是鞍点。处于鞍点时可使用 optimization 算法进行改善。

超参数的调试

1. 调试处理

ML: 参数较少, 使用 grid DL: 超参数较多, 使用随机选择点调试。

调试时遵循“Coarse to fine”这一原则

2. 选择超参数的合适范围

一般选择超参数的值时, 都在一个区间内随机采样。但并不适用于所有超参数。

比如 learning rate α , 我们希望它的范围在 $0.001 \sim 1$ 之间, 但如果在 $[0, 1]$ 内采样, 只有 1% 的概率选中。

$r = -4 * np.random.rand()$ # r in $[-4, 0]$

$learning_rate = 10 ** r$

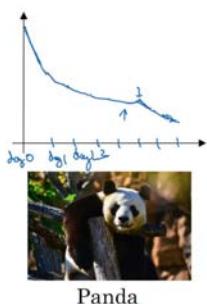
再比如 β , 一般取值在 $[0.9, 0.999]$

$r = -3 * np.random.rand()$

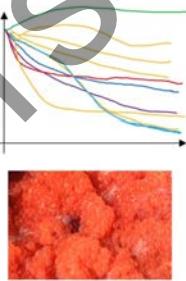
$beta = 1 - 10 * r$

3. 根据计算资源调参

Babysitting one
model



Training many
models in parallel



第一种方法: 当资源有限时,
仅调试一个模型。

第二种方法: 当资源充足时, 并
行调试多个模型, 选择最优
的一个。

4. Batch Norm

LR 中常使用归一化特征以加快 training。在深层的 NN 中, 也可以对各层的 $Z^{[l]}$ 进行归一化。

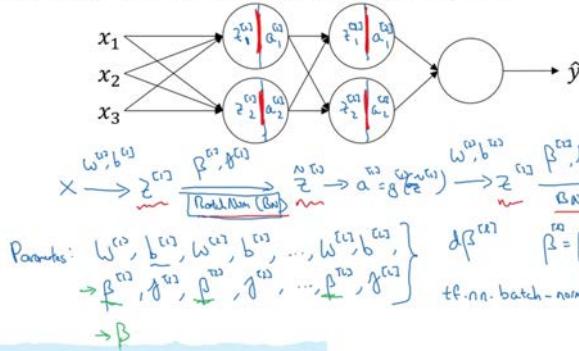
Input: $Z^{[l]}$: $Z^{[0]^{(0)}}, Z^{[1]^{(0)}}, \dots, Z^{[L]^{(0)}}$

$$\mu = \frac{1}{m} \sum z^{(i)} \quad \sigma^2 = \frac{1}{m} \sum (z^{(i)} - \mu)^2 \quad z_{\text{norm}}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

转化后的区的分量都是平均值为0且方差为1，但我们都希望各个隐藏层有独特的分布，所以我们令 $\tilde{z}^{(i)} = r z_{\text{norm}}^{(i)} + \beta$ 。若 $r = \sqrt{\sigma^2 + \epsilon}$, $\beta = \mu$, 则 $\tilde{z}_{\text{norm}}^{(i)} \approx \tilde{z}^{(i)}$

5. NN with batch norm

Adding Batch Norm to a network



for i in range(num):

Compute forward prop on X^t
Using $\tilde{z}^{(i)}$ replace $z^{(i)}$

Compute back prop: $dW^{(i)}, dr^{(i)}$
Update:

$$W^{(i)} := W^{(i)} - \alpha dW^{(i)} \\ r^{(i)} := r^{(i)} - \alpha dr^{(i)} \\ \beta^{(i)} := \beta^{(i)} - \alpha d\beta^{(i)}$$

Andrew Ng

notation:

- Batch norm 同样适用于 Momentum, RMSprop, Adam 等优化算法。
- 不用更新 $b^{(i)}$ 是因为 $b^{(i)}$ 在 norm 中已经被抵消。

6. Batch norm 起作用的原因

减少样本变化时所引发的分布的改变。Batch norm 能够限制前层参数的更新对后续网络值分布的影响，使得每一层网络都有一定的独立性。

Batch norm 用于 regularization:

- 在使用 mini-batch gradient descend 时，每次计算都在一个 mini-batch 上进行 (64, 128...), 而不是在整个 dataset 上。这样计算的 μ 和 σ^2 都处在一定的噪声，这样计算出的区也会有噪声。
- 在每层的 activation value 上加入了一些 noise，这也与 dropout 类似。
- mini-batch 越大，noise 越小，会减小 regularization 的效果。

7. Batch norm at test time

Batch Norm at test time

$$\begin{aligned} \mu &= \frac{1}{m} \sum_i z^{(i)} \\ \sigma^2 &= \frac{1}{m} \sum (z^{(i)} - \mu)^2 \\ z_{\text{norm}}^{(i)} &= \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}} \\ \hat{z}^{(i)} &= \gamma z_{\text{norm}}^{(i)} + \beta \end{aligned}$$

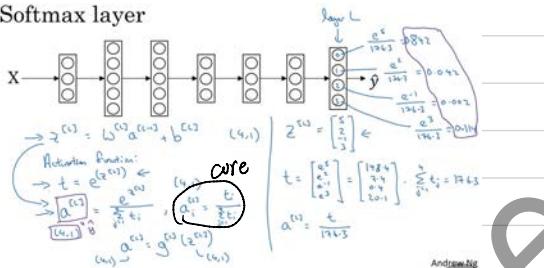
μ, σ^2 : estimate very approximately
 weighted average (across mini-batch).
 $x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, \dots$
 \downarrow
 $\theta_1, \theta_2, \theta_3, \dots$
 $\hat{z}_{\text{norm}}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$
 $\hat{z}^{(i)} = \gamma z_{\text{norm}}^{(i)} + \beta$

Andrew Ng

在 test 中需要对每一个 sample 进行预测。
 无法计算均值与方差。
 通常的作法是在 training 中使用指教
 加权平均，得到 μ 和 σ^2 。

8. Softmax

Softmax layer



Understanding softmax

$$\begin{aligned} \text{softmax}(x_i) &= \frac{e^{x_i}}{\sum_j e^{x_j}} \\ \alpha_i &= g^{(L)}(x_i^{(L)}) = \frac{e^{x_i}}{\sum_j e^{x_j}} \quad \text{"soft row"} \\ \alpha_i &= \frac{e^{x_i}}{e^{x_1} + e^{x_2} + e^{x_3} + e^{x_4}} = \frac{0.842}{0.042 + 0.002 + 0.114} \quad \text{"hard row"} \end{aligned}$$

Softmax regression generalizes logistic regression to C classes.

If $C=2$, softmax reduces to logistic regression. $\alpha_1 = \frac{0.842}{0.842 + 0.158}$

相比传统的 $[0, 1, 0], [0, 0, 1]$ 这些分类，softmax 的结果显得更加 soft。

$$\text{真实标签: } y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \text{实际输出: } \hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$$

$$\text{loss function: } l(\hat{y}, y) = - \sum_{j=1}^4 y_j \log \hat{y}_j \xrightarrow{\text{结合 } y} -\log \hat{y}_1$$

为了最小化 loss \hat{y}_1 的值则必须大。其实 loss function 的作用就是使真实标签对应的概率越大（极大似然估计）。

$$\text{对应的 cost function: } J(W^{(0)}, b^{(0)}, \dots) = \frac{1}{m} \sum_{i=1}^m l(\hat{y}^{(i)}, y)$$

$$\text{梯度计算公式: } dZ^{(L)} = \hat{y} - y$$