

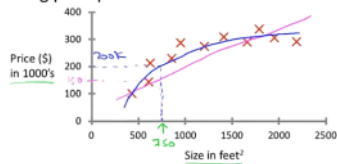
笔记一，线性分类器

2023年10月30日 20:34

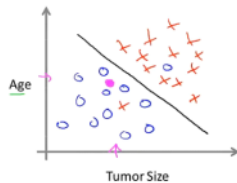
监督学习：使用标记数据训练模型，来预测出正确答案。
每一个样本都有正确的答案。

回归：输出是连续的

Housing price prediction.



分类：输出是离散的



无监督学习：使用的是未标记的数据集，让机器自己来分类。
事先并不知道有几类



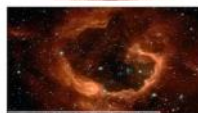
Organize computing clusters



Social network analysis



Market segmentation



Astronomical data analysis

Andrew Ng

Hypothesis:

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

$\theta^T x$

代价函数：均方误差 (MSE):

Parameters:

$$\theta_0, \theta_1$$

Cost Function:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Goal: minimize $J(\theta_0, \theta_1)$

Gradient descent algorithm

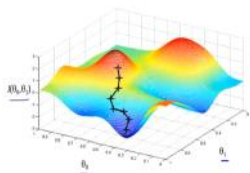
repeat until convergence {

$$\rightarrow \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad (\text{for } j = 0 \text{ and } j = 1)$$

}

梯度下降：使得代价函数更快的变小：

环顾四周，向哪个方向走下山会更快



Correct: Simultaneous update

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$\theta_0 := \text{temp0}$$

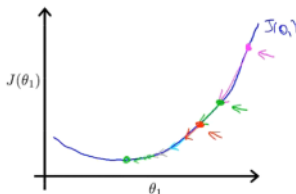
$$\theta_1 := \text{temp1}$$

Andrew Ng

Gradient descent can converge to a local minimum, even with the learning rate α fixed.

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease α over time.



Andrew Ng

梯度下降推导：

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \left(\frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \right)$$

$$= \frac{\partial}{\partial \theta_j} \left(\frac{1}{2m} \sum_{i=1}^m (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2 \right)$$

$$\theta_0, j = 0: \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})$$

$$\theta_1, j = 1: \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

Gradient Descent

New algorithm ($n \geq 1$):

n

$a_1=1$
 $a_2=2$
 $b_1=3$
 $b_2=4$

$$\theta, j=1: \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

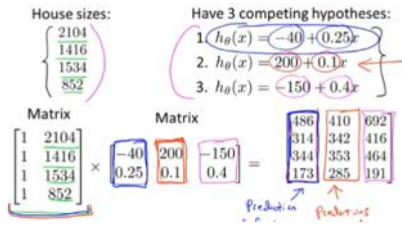
$$\sum_{i=1}^n a_i \cdot b_i$$

$$= 1 \times 3 + 2 \times 4 = 11$$

$$= \begin{bmatrix} 1 & 2 \end{bmatrix} \times \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

$$= [1 \times 3 + 2 \times 4] = 11$$

矩阵的运用:



Gradient Descent

Previously (n=1):

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

(simultaneously update θ_0, θ_1)

}

New algorithm (n ≥ 1):

Repeat {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update θ_j for $j = 0, \dots, n$)

}

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

(simultaneously update $\theta_0, \theta_1, \theta_2$)

...

特征缩放:

梯度下降得更快

Feature Scaling

Idea: Make sure features are on a similar scale.

E.g. x_1 = size (0-2000 feet²)

x_2 = number of bedrooms (1-5)

$x_1 = \frac{\text{size (feet}^2\text{)}}{2000}$

$x_2 = \frac{\text{number of bedrooms}}{5}$

\leq

θ_2

$J(\theta)$

θ_1

归一化:

$$x_i = \frac{x_i - \text{Avg}}{\text{Max} - \text{Min}}$$

梯度下降与正规方程:

Gradient Descent

- Need to choose α .
- Needs many iterations.
- Works well even when n is large.

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

m大, 大于 10^4
大多数模型

Normal Equation

- No need to choose α .
- Don't need to iterate.
- Need to compute $(X^T X)^{-1}$
- Slow if n is very large.

$$X = \begin{bmatrix} x_0^{(1)} & \dots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ x_0^{(m)} & \dots & x_n^{(m)} \end{bmatrix}$$

m小, 小于 10^4
只适用于线性回归

$$\theta = (X^T X)^{-1} X^T y$$

二分类问题:

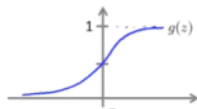
Logistic regression

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-z}}$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

Suppose predict "y = 1" if $h_\theta(x) \geq 0.5$ $\theta^T x \geq 0$

predict "y = 0" if $h_\theta(x) < 0.5$ $\theta^T x < 0$



Decision Boundary

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

Predict "y = 1" if $-3 + x_1 + x_2 \geq 0$

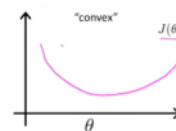
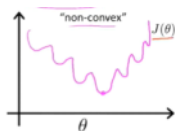
Non-linear decision boundaries

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

Predict "y = 1" if $-1 + x_1^2 + x_2^2 \geq 0$

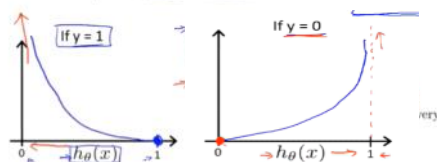
代价函数为什么不用上述的均方误差呢?

线性回归的代价函数是平滑的凸函数, 能找到最优的theta
而现在变得很复杂, 若适用梯度下降, 则找不到全局最优



Logistic regression cost function

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



y=1时, 说明h应该是大于0.5的,
所以当h趋于0时, 差距很大, 代价很大
当h趋于1时, 差距很小, 代价很小
我们的目标就是调整theta, 让h收敛于1.

Gradient Descent

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_{\theta}(x^{(i)})) \right]$$

Want $\min_{\theta} J(\theta)$:

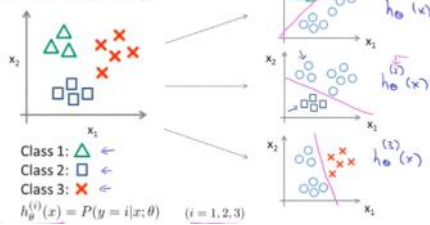
Repeat {

$$\theta_j := \theta_j - \alpha \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update all θ_j)

}

One-vs-all (one-vs-rest):



算出来的h, 表示的是属于第一个类别的概率

表示的是属于第二个类别的概率

表示的是属于第三个类别的概率

哪个数值大, 说明属于哪个

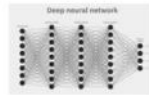
用二分类表示多分类:

多分类(softmax):

$$P(y = i) = \frac{e^{z_i}}{\sum_{j=0}^{K-1} e^{z_j}}, i \in \{0, \dots, K-1\}$$

为什么用指数函数: 求导简单, 爆炸式增长, 一点x就会有很大变化, 有利于区分

$$\text{loss}(x, \text{class}) = -\log \left(\frac{\exp(x[\text{class}])}{\sum_j \exp(x[j])} \right) = -x[\text{class}] + \log \left(\sum_j \exp(x[j]) \right)$$



Person, dog, cat
0, 1, 2

output
[0.1, 0.2, 0.3]

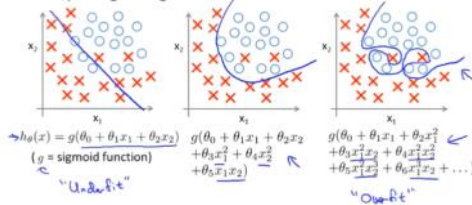
Target
1
class

多分类代价函数: 交叉熵:

$$\text{Loss}(x, \text{class}) = -0.2 + \log(\exp(0.1) + \exp(0.2) + \exp(0.3))$$

目标值的概率, 不能让所有预测的概率都很大, 概率越大, 比如是0.8, 0.9, 0.8, 区分度不好, loss越小

Example: Logistic regression



欠拟合与过拟合

欠拟合: 高偏差

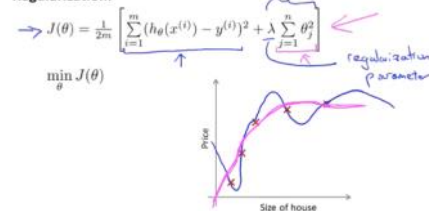
过拟合: 高方差

为了解决过拟合, 使用正则化

通过引入正则化参数, 减小theta

比如右侧如果用三次方来回归, 那么会过拟合, 减小三次方的系数就能够既保留样本, 又减小影响

Regularization.



怎么减小系数呢?

在代价函数中加入正则项

因为目标是让J(theta)小,

所以只有减小系数才能让J(theta)小

求导

本质就是比原来缩小一些

Gradient descent

Repeat {

$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$

$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right]$

}

$\theta_j := \theta_j (1 - \alpha \frac{\lambda}{m}) - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$

梯度下降引入正则化

Non-invertibility (optional/advanced).

Suppose $m \leq n$
(Examples) (Features)

$$\theta = (X^T X)^{-1} X^T y$$

If $\lambda > 0$,

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix} \right)^{-1} X^T y$$

正规方程引入正则化, 而且引入后, 只要lambda>0, 就是一定可逆的了。

分类问题的正则化:

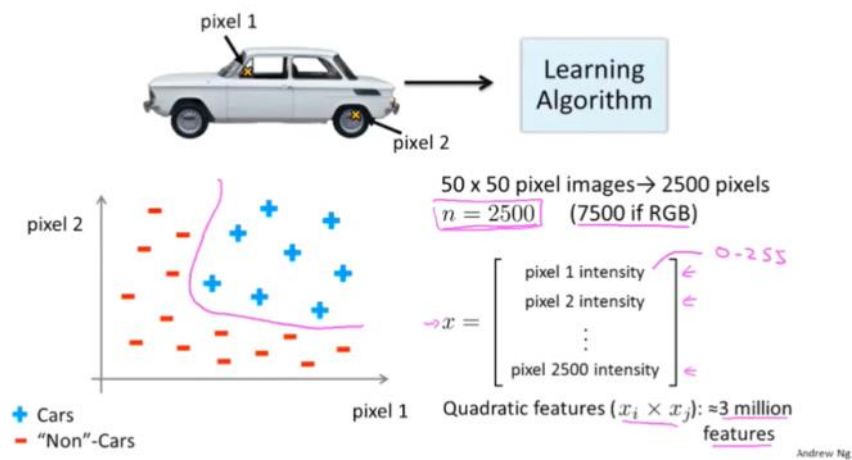
$$J_{\text{val}} = [\text{code to compute } J(\theta)];$$

$$J(\theta) = \left[-\frac{1}{m} \sum_{i=1}^m y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right] + \left[\frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2 \right]$$

笔记二，非线性分类器

2023年11月13日 16:10

由于实际问题中，特征量很大
采用之前的办法很费时间
因此采用神经网络



基本模型:

Neural Network

$\rightarrow a_i^{(j)}$ = "activation" of unit i in layer j

$\rightarrow \Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer $j + 1$

$\Theta^{(1)} \in \mathbb{R}^{3 \times 4}$

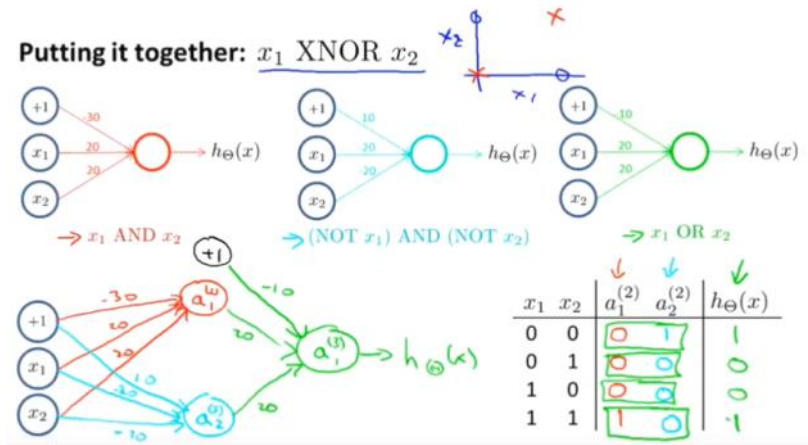
$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$

$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$

$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$

$h_{\Theta}(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$

If network has s_j units in layer j , s_{j+1} units in layer $j + 1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$.



Cost function

代价函数

Logistic regression:

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

Neural network:

$$\rightarrow h_{\Theta}(x) \in \mathbb{R}^K \quad (h_{\Theta}(x))_i = i^{th} \text{ output}$$

$$\rightarrow J(\Theta) = -\frac{1}{m} \left[\sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_{\Theta}(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_{\Theta}(x^{(i)}))_k) \right]$$

$$+ \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$



反向传播：代价函数是什么？代价函数就是所有预测值和目标值的偏差，所以我们定义：

$$L(\theta) = \sum_{n=1}^N C^n(\theta)$$

要让代价函数最小，那么需要求导数，到时候就用梯度下降来学习参数：

$$\theta^1 = \theta^0 - \eta \nabla L(\theta^0)$$

$$\theta^2 = \theta^1 - \eta \nabla L(\theta^1)$$

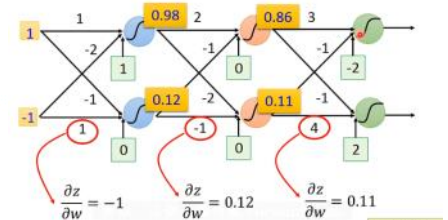
$$\frac{\partial L(\theta)}{\partial w} = \sum_{n=1}^N \frac{\partial C^n(\theta)}{\partial w}$$

Backpropagation – Forward pass

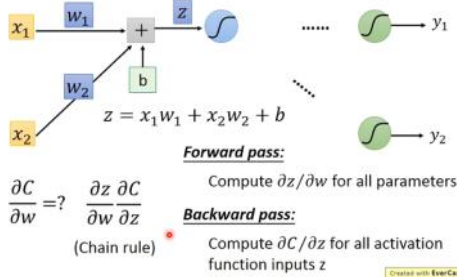
Compute $\partial z / \partial w$ for all parameters

$$\partial z / \partial w_1 = ? \quad x_1$$

$$\partial z / \partial w_2 = ? \quad x_2$$

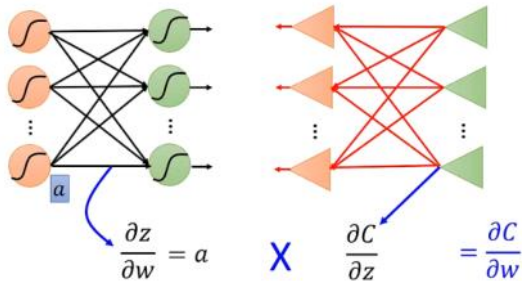


接下来，计算单个误差C的导数：

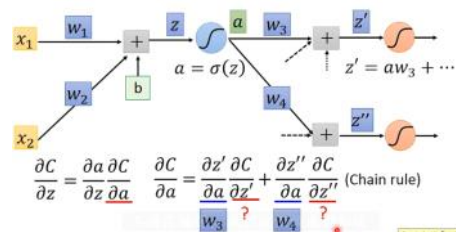


Forward Pass

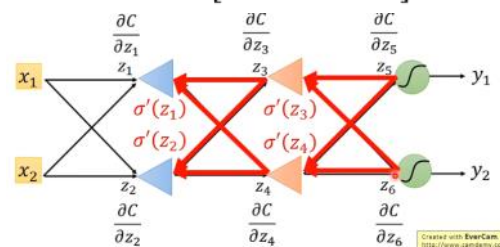
Backward Pass



Compute $\partial C / \partial z$ for all activation function inputs z



$$\frac{\partial C}{\partial z} = \sigma'(z) \left[w_3 \frac{\partial C}{\partial z'} + w_4 \frac{\partial C}{\partial z''} \right]$$



Parameter vector θ

→ $\theta \in \mathbb{R}^n$ (E.g. θ is "unrolled" version of $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$)

→ $\theta = [\theta_1, \theta_2, \theta_3, \dots, \theta_n]$

梯度检验：验证算法是否正确

$$\begin{aligned} \frac{\partial}{\partial \theta_1} J(\theta) &\approx \frac{J(\theta_1 + \epsilon, \theta_2, \theta_3, \dots, \theta_n) - J(\theta_1 - \epsilon, \theta_2, \theta_3, \dots, \theta_n)}{2\epsilon} \\ \frac{\partial}{\partial \theta_2} J(\theta) &\approx \frac{J(\theta_1, \theta_2 + \epsilon, \theta_3, \dots, \theta_n) - J(\theta_1, \theta_2 - \epsilon, \theta_3, \dots, \theta_n)}{2\epsilon} \\ &\vdots \\ \frac{\partial}{\partial \theta_n} J(\theta) &\approx \frac{J(\theta_1, \theta_2, \theta_3, \dots, \theta_n + \epsilon) - J(\theta_1, \theta_2, \theta_3, \dots, \theta_n - \epsilon)}{2\epsilon} \end{aligned}$$

```
for i = 1:n,
    thetaPlus = theta;
    thetaPlus(i) = thetaPlus(i) + EPSILON;
    thetaMinus = theta;
    thetaMinus(i) = thetaMinus(i) - EPSILON;
    gradApprox(i) = (J(thetaPlus) - J(thetaMinus)) / (2*EPSILON);
end;
```

Check that $\text{gradApprox} \approx \text{DVec}$

↑
From backprop.

Random initialization: Symmetry breaking

Initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\epsilon, \epsilon]$
(i.e. $-\epsilon \leq \Theta_{ij}^{(l)} \leq \epsilon$)

E.g.

```
Theta1 = rand(10,11) * (2*INIT_EPSILON) - INIT_EPSILON;
```

参数随机初始化：

如果相同的话，

那么后续步骤都一样，

算出来的权重也一样

神经网络步骤：1：随机初始化权重

2：前向传播算法计算预测值

3：计算代价函数

4：使用后向传播算法计算代价函数梯度

5：使用梯度检验来检验反向传播代码写得是否正确

6：使用梯度下降或者高级算法加上反向传播完成预测

Evaluating your hypothesis

Dataset:

Size	Price
2104	400
1600	330
2400	369
1416	232
3000	540
1985	300
1534	315
1427	199
1380	212
1494	243

60% Training set

20% Cross validation (CV)

20% Test set

分为训练集、交叉验证集和测试集：

Train/validation/test error

Training error:

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Cross Validation error:

$$J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_{\theta}(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

Test error:

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_{\theta}(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

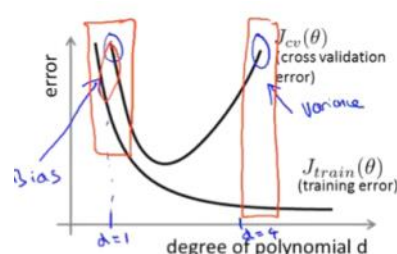
Model selection

- $h_{\theta}(x) = \theta_0 + \theta_1 x \rightarrow \min_{\theta} J(\theta) \rightarrow \theta^{(1)} \rightarrow J_{cv}(\theta^{(1)})$
- $h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \rightarrow \theta^{(2)} \rightarrow J_{cv}(\theta^{(2)})$
- $h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_3 x^3 \rightarrow \theta^{(3)} \rightarrow J_{cv}(\theta^{(3)})$
- \vdots
- $h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_{10} x^{10} \rightarrow \theta^{(10)} \rightarrow J_{cv}(\theta^{(10)})$

训练集：算出多个模型的theta

交叉验证集：选择出最好的模型

测试集：评价选出的模型



Bias (underfit):

→ $J_{train}(\theta)$ will be high
 $J_{cv}(\theta) \approx J_{train}(\theta)$

欠拟合

Variance (overfit):

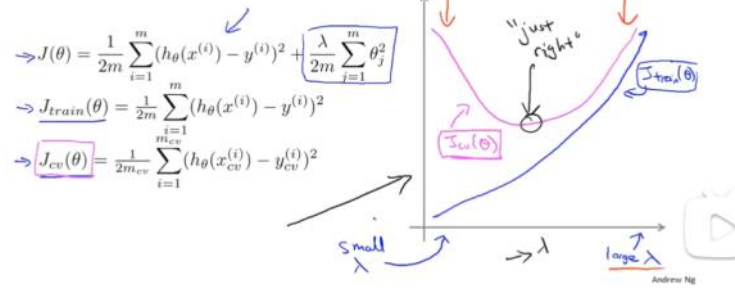
→ $J_{train}(\theta)$ will be low
 $J_{cv}(\theta) \gg J_{train}(\theta)$

过拟合

>>

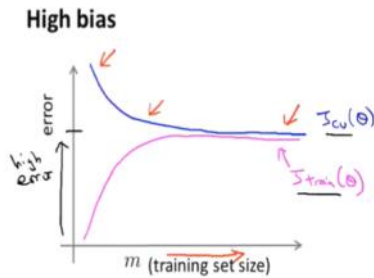
And

Bias/variance as a function of the regularization parameter λ

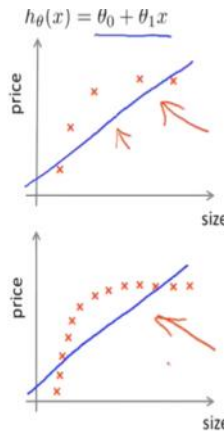


选择正则化参数

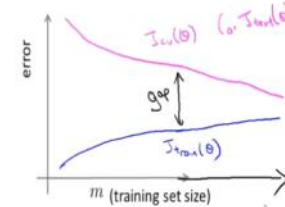
学习曲线:



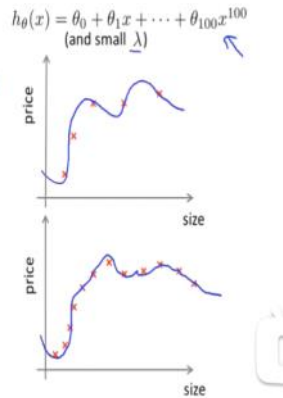
If a learning algorithm is suffering from high bias, getting more training data will not (by itself) help much.



High variance



If a learning algorithm is suffering from high variance, getting more training data is likely to help.



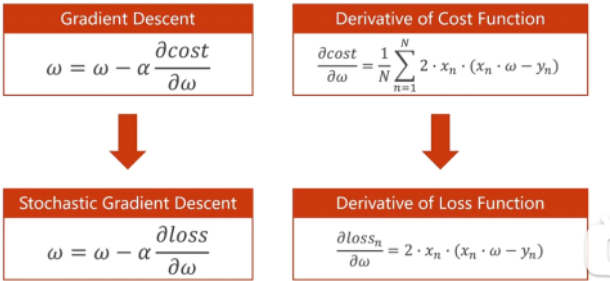
决定接下来做什么:

- Get more training examples → fixes high variance
- Try smaller sets of features → fixes high variance
- Try getting additional features → fixes high bias
- Try adding polynomial features (x_1^2, x_2^2, x_1x_2 , etc) → fixes high bias.
- Try decreasing λ → fixes high bias
- Try increasing λ → fixes high variance

笔记三,刘老师学习笔记（知识的补充）

2023年12月3日 16:11

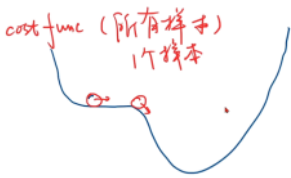
随机梯度下降:



Stochastic Gradient Descent

Derivative of Loss Function

损失不是所有样本的损失，而是随机一个样本的损失，这样可以防止停在鞍点局部最优



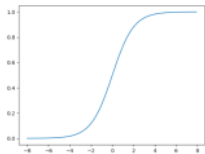
为什么要加激活函数:

A two layer neural network

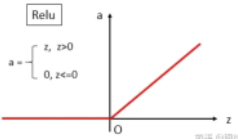
$$\begin{aligned} \hat{y} &= W_2(W_1 \cdot X + b_1) + b_2 \\ &= W_2 \cdot W_1 \cdot X + (W_2 b_1 + b_2) \\ &= W \cdot X + b \end{aligned}$$

如果不加激活函数，那么无论多少层都相当于一层，都可以化简而加入了非线性，使得多层有意义，表达式更复杂

二分类与多分类: 二分类:



激活函数: 之前是ReLU(), 最后是sigmoid, 因为在0到1之间, 就可以正好把他当做概率, 把他当做成是=1的概率, 大于0.5是一类, 小于0.5是另一类
损失函数: BCE, (Binary Cross Entropy), $LOSS = -(y \log(p(x)) + (1-y) \log(1-p(x)))$



多分类:

激活函数: 之前是ReLU(), 最后是softmax
损失函数: 交叉熵
分类器: 为了把输出结果变为概率, 在激活后加入一个softmax分类器 $Softmax(z_i) = \frac{e^{z_i}}{\sum_{c=1}^C e^{z_c}}$, 使用指数是为了更大的区分(爆炸式)

三、引入ReLU的原因

- 第一, 采用sigmoid等函数, 算激活函数时(指数运算), 计算量大, 反向传播求误差梯度时, 求导涉及除法, 计算量相对大, 而采用Relu激活函数, 整个过程的计算量节省很多。
- 第二, 对于深层网络, sigmoid函数反向传播时, 很容易就会出现 梯度消失 的情况 (在sigmoid接近饱和区时, 变换太缓慢, 导数趋于0, 这种情况会造成信息丢失) , 从而无法完成深层网络的训练。
- 第三, ReLu会使一部分神经元的输出为0, 这样就造成了 网络的稀疏性, 并且减少了参数的相互依存关系, 缓解了过拟合问题的发生。

训练误差	验证误差	情况
小	小	正常
小	大	过拟合
大	大	欠拟合

Resnet:

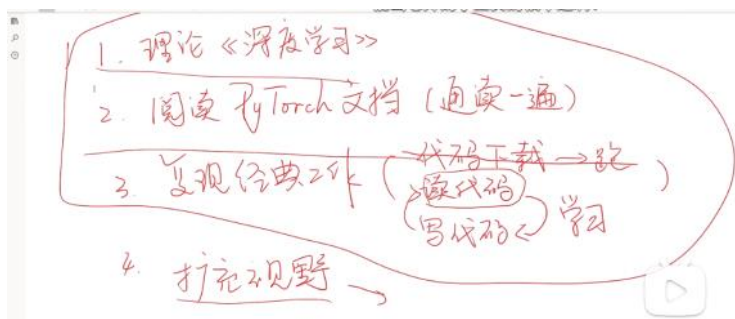
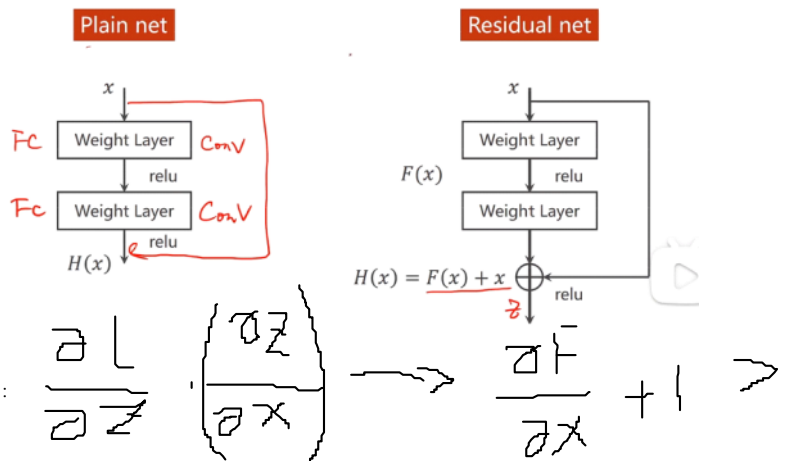
反向传播时，有可能有一部分导数会近似于0，
梯度下降：

$$\theta = \theta - \alpha \cdot \frac{\partial L}{\partial \theta}$$

所以，梯度近似于0时，参数就不变了

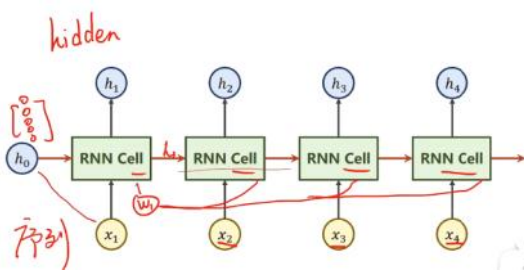
残差网络，梯度计算就等于（忽略激活函数）：

这样最少也是在1附近，而不是在0附近

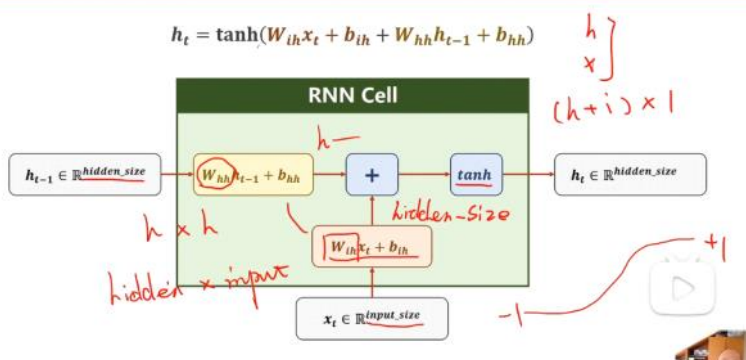


RNN:

(循环神经网络)



$h = 0$
for x in X ;
 $h = \text{linem}(x, h)$



• Suppose we have sequence with below properties:

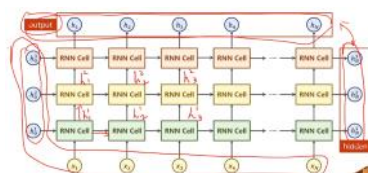
- **batchSize** batch的大小，每次处理几组
- **seqLen** 序列的长度，一组中有几个样本，比如天气预测中，每组有三天
- **inputSize, hiddenSize,**
- **numLayers** RNN的层数

• The shape of **input** and **h_0** of RNN:

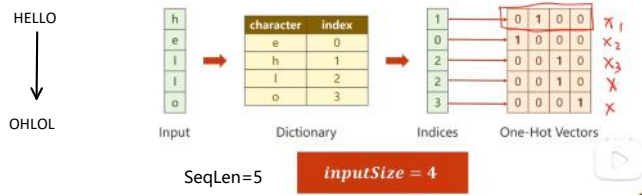
- **input.shape** = (seqLen, batchSize, inputSize)
- **h_0.shape** = (numLayers, batchSize, hiddenSize)

• The shape of **output** and **h_n** of RNN:

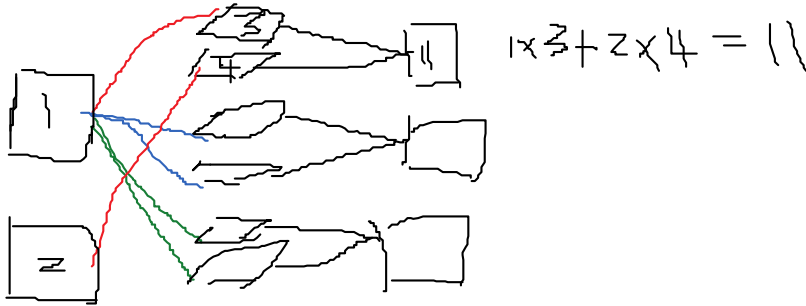
- **output.shape** = (seqLen, batchSize, hiddenSize)
- **h_n.shape** = (numLayers, batchSize, hiddenSize)



• The inputs of RNN Cell should be vectors of numbers.



卷积：2通道→3通道



笔记四：YOLO

2023年12月18日 16:51

假设有20个女生，80个男生，
目标是选出所有女生，结果选出了15个女生，15个男生
TP: (女变女) : 15, FP:(男变女) : 15
FN:(女变男): 5,

					FN
			FP		TP

标准：准确率与查全率：
准确率/精度 (Precision) : $\frac{TP}{TP+FP}$ =15/30 (结果中选对的有多少)
查全率: (Recall) : $\frac{TP}{TP+FN}$ =15/20 (结果中是否都选出来了)

笔记五、Faster R-CNN

2024年1月4日 20:44

一、FFN: 采用卷积网络提取图片特征：可以用vgg,resnet……

二、RPN:

2.1 对特征图(feature map)先采用像素分割，然后每个像素点设置N个anchor

在FasterRCNN的RPN网络部分，anchor为三个尺度{128, 256, 512}，三个比例{1:1, 1:2, 2:1}，所以一共9组anchor。

2.2 为IOU大的anchor分配正样本标签，说明有物体

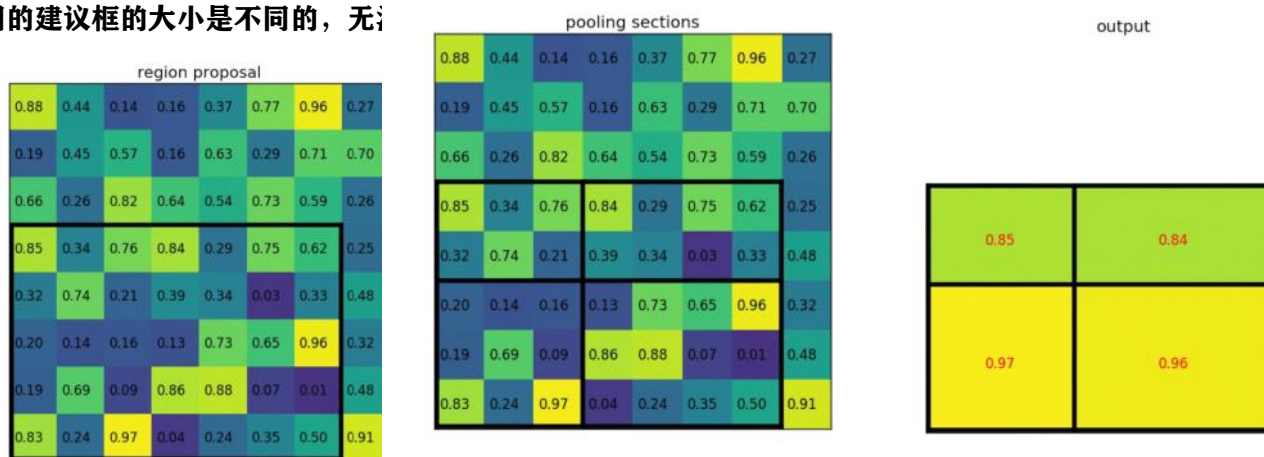
为IOU小的anchor分配负样本标签，说明没有物体

2.3 对这些正样本的anchor，进行回归，原来9个anchor，现在选出1个有物体的iou最大的了，告诉网

络，现在着重调整这个anchor的大小

三、Region of Interest Pooling (RoI) :

不同的建议框的大小是不同的，无



四、Region-based Convolutional Neural Network (R-CNN) :

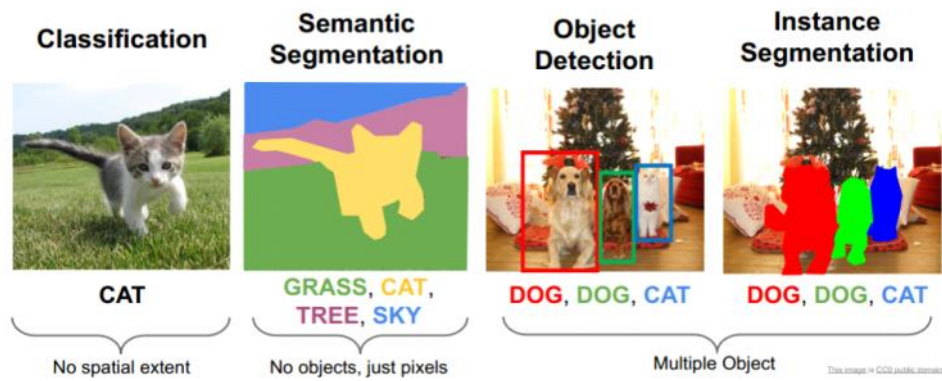
- R-CNN是Faster R-CNN里的最后一步。从图像获得卷积特征图后，我们通过RPN获取建议框并通过RoI池为每一个建议框提取特征，我们最终需要使用这些特征进行分类，最后通过全连接层来为每个可能的类别输出分数。
- R-CNN有两个不同目标：

1. 将建议框的物体进行具体分类（这里要加上一个“背景”类，用于删除错误的建议框）
2. 根据预测的类别更好地调整我们的边界框

- R-CNN把每个建议框提取的特征展平，并且使用两个具有ReLU激活层的4096大小的全连接层进行分类（对应两个不同的目标）：

1. 第一个全连接层有N+1个单元，其中N是物体的类别，加的1是背景类。
2. 另一个全连接层有4N个单元，对应我们边界框的4个偏移量
(x_center,y_center,width,height) ，其中N是类别数

CV领域：



	图像分类	语义分割	目标检测	实例分割
干什么的？	是什么东西（一张图片只有一个物体）	在像素级别进行的目标检测，每个像素点是属于那种物体？	是什么东西？在哪里（画个框圈起来）	对同一类别的不同个体也要分出来
普遍用的模型	ResNet		YOLO	
我做了什么？	天气识别，准确率已达到95.3%			

模型发展史：

1、图像分类： ➤ LeNet(1998):

LeNet又叫LeNet-5，这个5是指它的网络结构中有5个表示层，具体结构如下图所示：

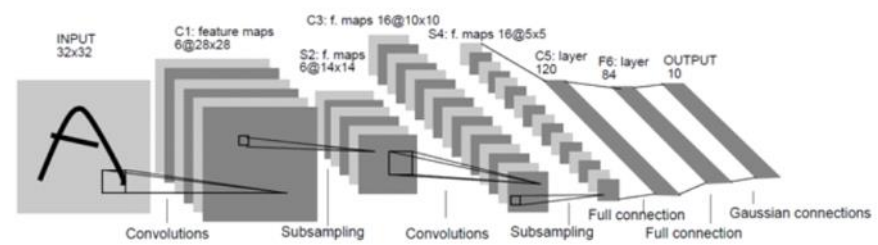


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeNet5包含Input、卷积层1、池化层1、卷积层2、池化层2、全连接层、输出层。

➤ AlexNet(2012):

我们言归正传，AlexNet为8层深度的CNN网络，其中包括5个卷积层和3个全连接层（不包括LRN层和池化层），如下图所示：

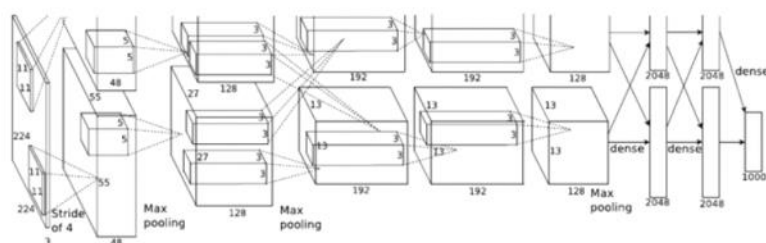


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440-186,624-64,896-64,896-43,264-4096-4096-1000.

AlexNet的特点和贡献：

- (1) **使用ReLU作为激活函数**，由于ReLU是非饱和函数，也就是说它的导数在大于0时，一直是1，因此解决了Sigmoid激活函数在网络比较深时的梯度消失问题，提高SGD（随机梯度下降）的收敛速度。
- (2) **使用Dropout方法避免模型过拟合**，该方法通过让全连接层的神经元（该模型在前两个全连接层引入Dropout）以一定的概率失去活性（比如0.5），失活的神经元不再参与前向和反向传播，相当于约有一半的神经元不再起作用。在预测的时候，让所有神经元的输出乘Dropout值（比如0.5）。这一机制有效缓解了模型的过拟合。
- (3) **重叠的最大池化**，之前的CNN中普遍使用平均池化，而AlexNet全部使用最大池化，避免平均池化的模糊化效果。并且，池化的步长小于核尺寸，这样使得池化层的输出之间会有重叠和覆盖，提升了特征的丰富性。

➤ VGG(2014):

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64	conv3-64	conv3-64	conv3-64
maxpool					
conv3-128	conv3-128	conv3-128	conv3-128	conv3-128	conv3-128
maxpool					
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

VGG的特点:

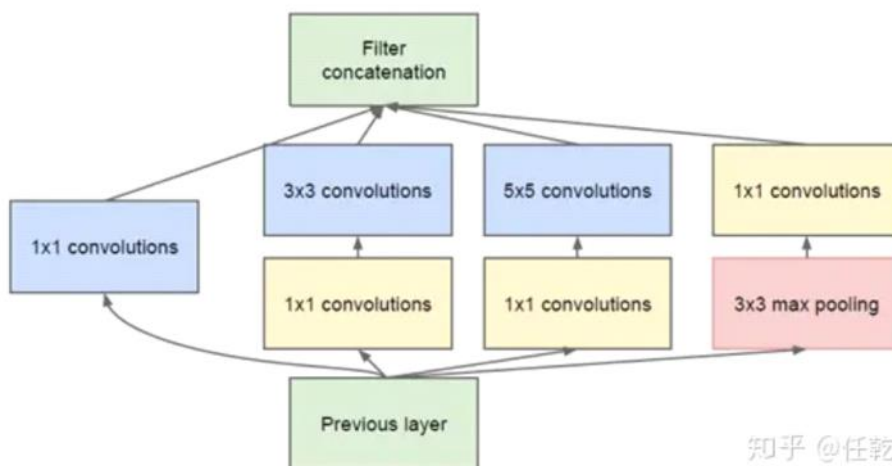
- (1) VGGNet拥有5段卷积，每段卷积内有2-3个卷积层，同时每段尾部都会连接一个最大池化层（用来缩小图片）。
- (2) 每段内的卷积核数量一样，越后边的段内卷积核数量越多，依次为：64-128-256-512-512。
- (3) 越深的网络效果越好。
- (4) LRN层作用不大（作者结论）。
- (5) 1x1的卷积也是很有效的，但是没有3*3的卷积好，大一些的卷积核可以学习更大的空间特征。不过1x1的卷积核可以用于增加模型的非线性变化，并可用于升维和降维。

VGG的突出贡献在于，证明了使用小的卷积核，增加网络深度可以有效的提高模型效果，而且VGGNet对其他数据集具有很好的泛化能力。到目前为止，VGGNet依然经常被用来提取图像特征。

为什么说小的卷积核堆叠能够提高模型效果呢？比如3x3的卷积核堆叠两层，则“感受野”就会变为5x5，堆叠三层，“感受野”就会变成7x7，而多层小的卷积核参数更少（ $3 \times 3 \times 3 = 27 < 1 \times 7 \times 7 = 49$ ），且增加了更多的非线性变化（三层卷积有三次ReLU），这样增加了模型的表达能力。

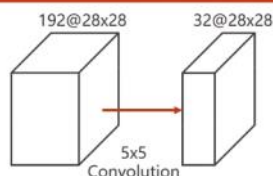
➤ GoogleNet(2014):

卷积核尺寸也是固定大小的。但是，在实际情况下，在不同尺度的图片里，需要不同大小的卷积核，这样才能使性能最好，或者或，对于同一张图片，不同尺寸的卷积核的表现效果是不一样的，因为他们的感受野不同。所以，我们希望让网络自己去选择，Inception便能够满足这样的需求，一个Inception模块中并列提供多种卷积核的操作，网络在训练的过程中通过调节参数自己去选择使用，同时，由于网络中都需要池化操作，所以此处也把池化层并列加入网络中。

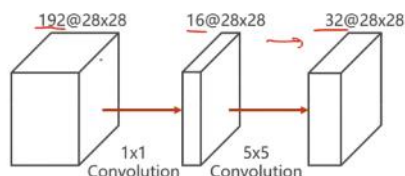


Why is 1x1 convolution?

刘二大人 bilibili



Operations:
 $5^2 \times 28^2 \times 192 \times 32 = 120,422,400$



Operations:
 $1^2 \times 28^2 \times 192 \times 16 + 5^2 \times 28^2 \times 16 \times 32 = 12,433,648$

1x1卷积作用，
减少参数和运算
量，减少通道数

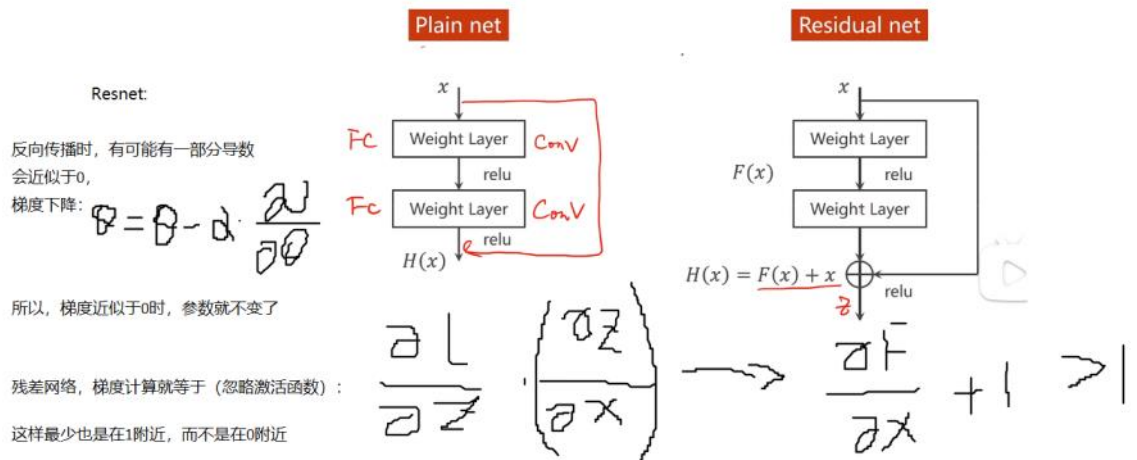
Lecturer: Hongpu Liu

Lecture 11-13

PyTorch Tutorial @ SLAM Research



➤ ResNet(2015):



➤ DenseNet(2017):

DenseNet的特点是：密集连接，来缓解梯度消失问题，加强特征传播，鼓励特征复用，极大的减少了参数量。DenseNet 是一种具有密集连接的卷积神经网络。在该网络中，任何两层之间都有直接连接，也就是说，网络每一层的输入都是前面所有层输出的并集，而该层所学习的特征图也会被直接传给其后面所有层作为输入。

下图是 DenseNet 的一个dense block示意图：

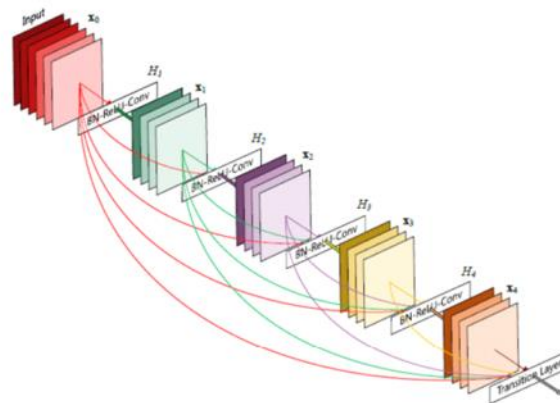
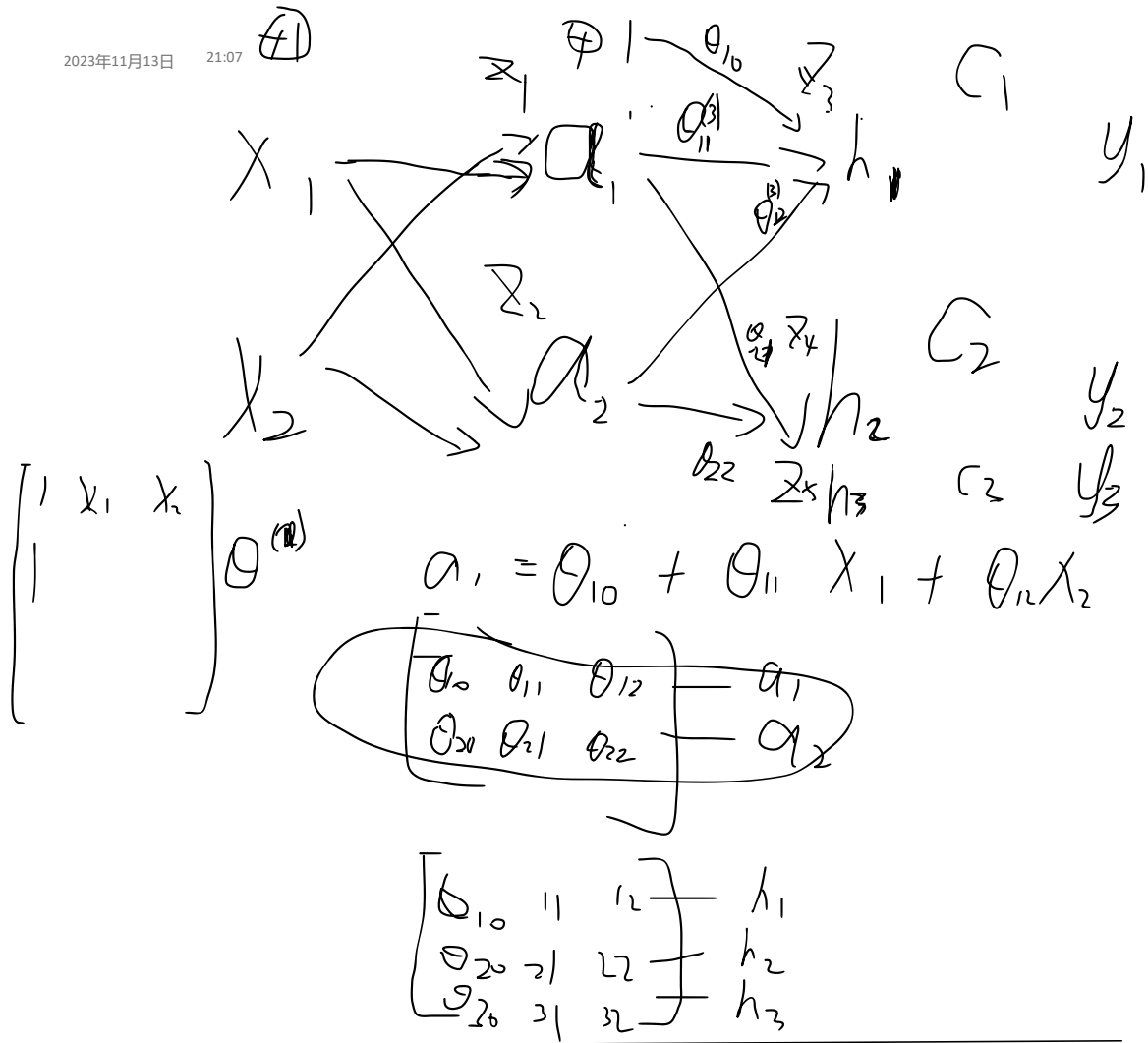


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.



①

①

①

 X_1 a_1 a_3 $z_3 | h$ X_2 a_2 a_4 2×3 2×3 1×3

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$$

$$\begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

$$h$$

$$\frac{dz_5}{da_3} = \theta_2^{(3)}$$

$$\begin{aligned}
h_1 &= w_1 a_1 + w_2 a_2 + b \\
&= w_1 \cdot (\theta_1^T x) + w_2 (\theta_2^T x) + b \\
&= w_1 \cdot (\theta_1 x_1 + \theta_2 x_2 + \theta_3) + \\
&\quad w_2 (\theta_4 x_1 + \theta_5 x_2 + \theta_6) + b \\
&= (w_1 \theta_1 + w_2 \theta_4) x_1 + \\
&\quad (w_1 \theta_2 + w_2 \theta_5) x_2 + \\
&\quad w_1 \theta_3 + w_2 \theta_6 + b \\
&= \theta^T x
\end{aligned}$$

$$y = Rx + b \quad \begin{array}{ll} x=50 & y=0.0018 \\ x=51 & y=0.0017 \end{array}$$

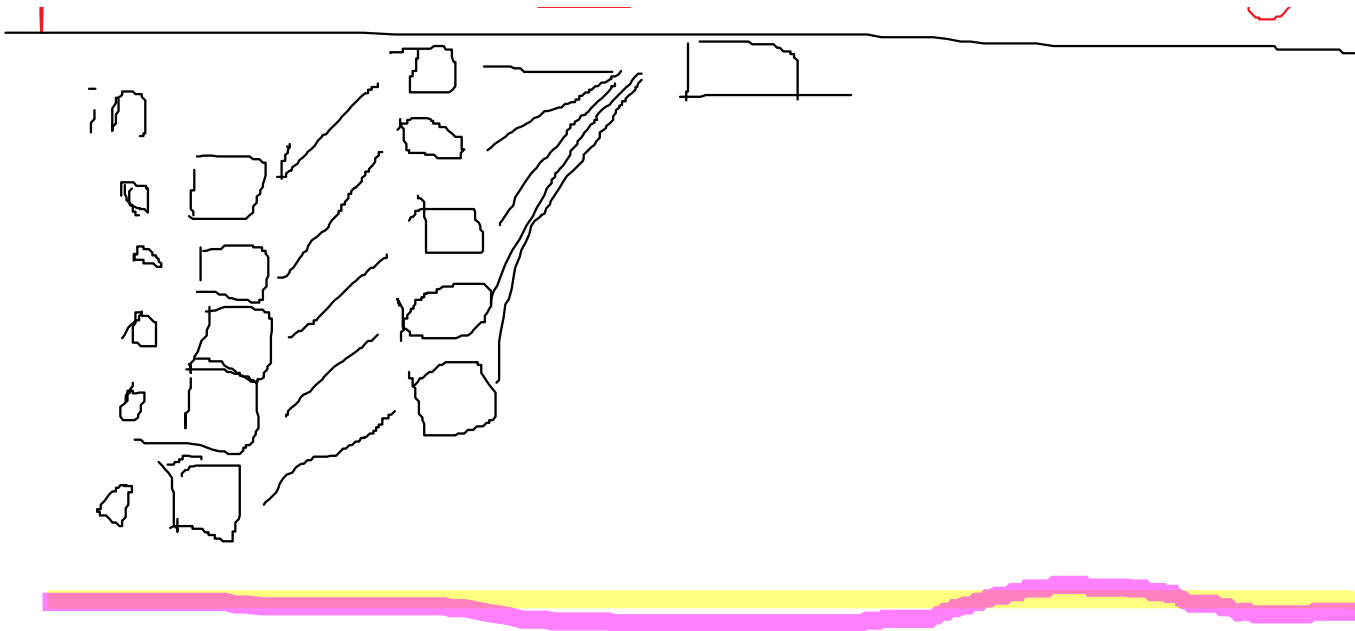
$$50R + b = 0.0018$$

$$51R + b =$$

$$R = -0.0001$$

$$y = 0.0018 - x$$

$$b = 0.0018 + 0.005 = 0.0068$$



512*7*7 convx1(512,128,size(1*1)) => 128*5*5


- 1解决方案（1）（1）
- 2感知体系白皮书（2022）
- 3平台信息化
- 4顶层设计
- 5智慧灯杆项目
- 6智慧产业城市体项目
- 7唐山柳林水库
- 8数字城市公共基础设施建设
- 9智慧城市创新应用
- 10华为中国智慧城市发展研究
- 11数字孪生城市框架
- 12数字孪生赋能智慧城市
- 13新型智慧城市整体规划建设方案
- 14城市大脑首部建设标准
- 15智慧环卫
- 16:174页智慧系统
- 17城市规划GIS
- 18城市大脑案例集
- 19网通管建设方案
- 20城市大脑趋于治理
- 21智慧城市可行性研究报告

Titanic - Machine Learning from Disaster

Submit Prediction

...

OverviewDataCodeModelsDiscussionLeaderboardRulesTeamSubmissions

3828	SereinSerein	<div></div>	0.77990	0	21h
3829	Radhakrishnan B	<div></div>	0.77990	1	2h
3830	Miguel Sta. Cruz	<div></div>	0.77990	22	34m
3831	HelloQiLin	<div></div>	0.77990	2	4m
<div><div></div><div><div>Your Best Entry!</div><div>Your most recent submission scored 0.77990, which is an improvement of your previous score of 0.52392. Great job!</div></div><div><div>Tweet this</div></div></div>					
3832	Thanh San	<div></div>	0.77751	4	2mo

天气

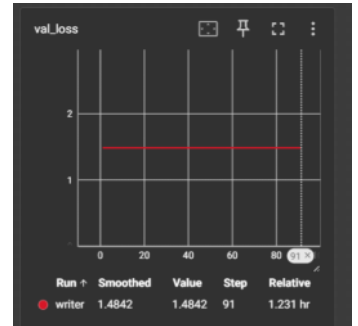
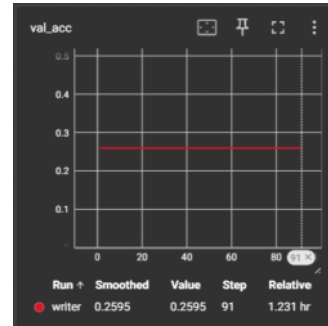
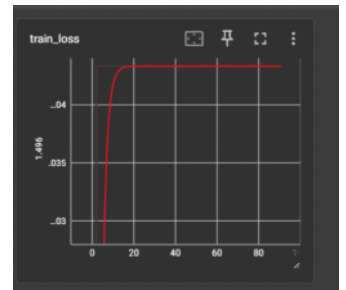
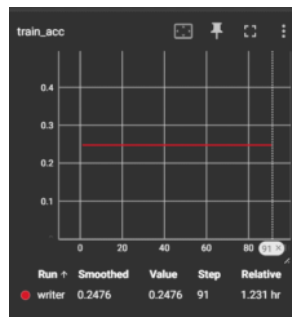
2023年12月8日 11:37

采用resnet18, 并且设置了dropout0.5, 学习率0.005,0.001,0.0005

判定: loss上升, 学习率过大

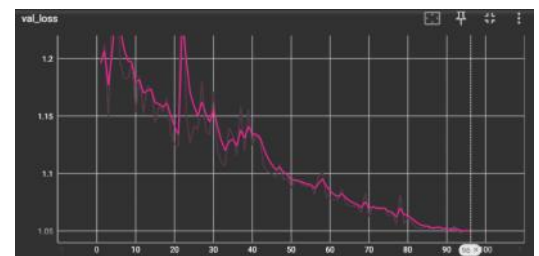
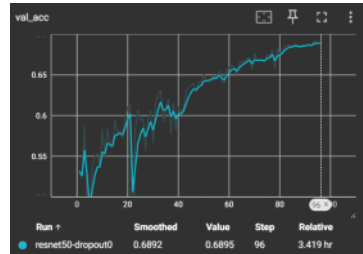
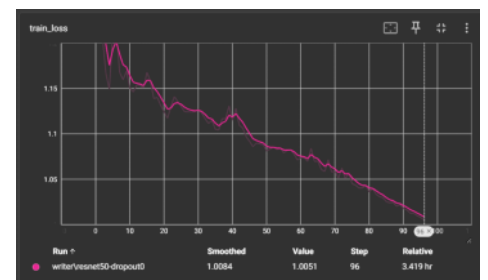
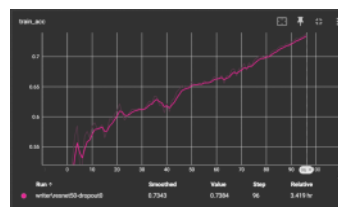
```
# import shutil
# import os

# if __name__ == '__main__':
#     path = '/kaggle/working'
#     if os.path.exists(path):
#         shutil.rmtree(path)
#     print('删除完成')
#     else:
#     print('原本为空')
```



采用resnet50, 设置dropout0.15, 学习率从0.001递减

判定, 训练集的损失高于测试集的损失, 并且测试集下降得逐渐变缓慢, 所以应该是过拟合
而且下降呈直线, 说明学习率过小



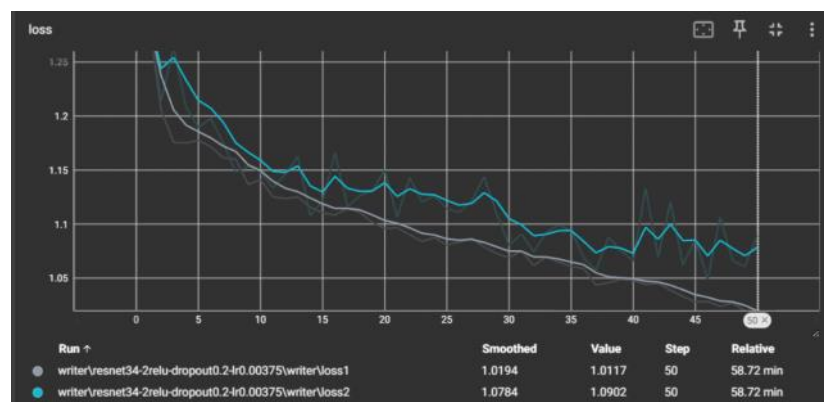
Resnet34-2relu-dropout0.2-lr0.00375

验证集准确率结果: 0.68,

```
def forward(self, x):
    x = self.net(x)
    x = self.relu(x)
    x = self.dropout(x)
    x = self.fc1(x)
    x = self.relu(x)
    x = self.fc2(x)
    x = self.output(x)
    return x
```

对应上边学习率0.0011

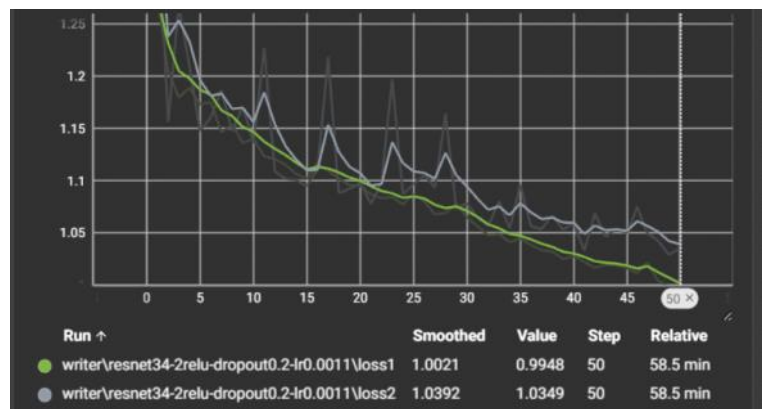
好像并没有什么区别, 看起来还是有些过拟合



对应上边学习率0.0011

好像并没有什么区别，看起来还是有些过拟合

0.712



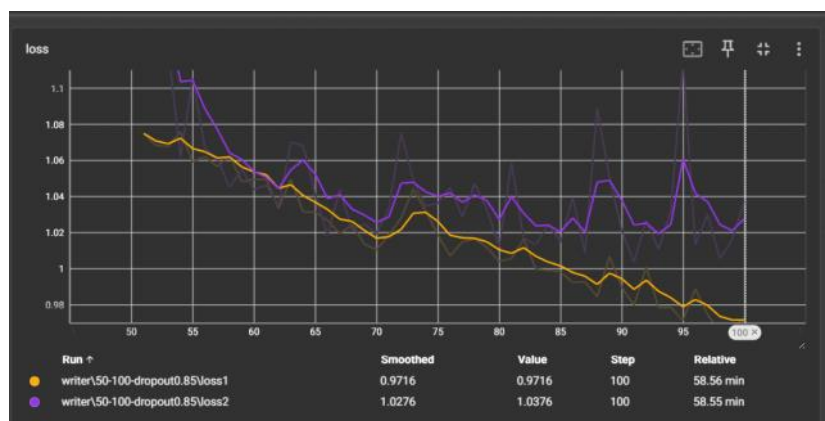
把上边这个再从50训练到100轮

发现loss差别越来越大了，这是什么
这是过拟合

还是有些过拟合，而且呈直线，
学习率小了



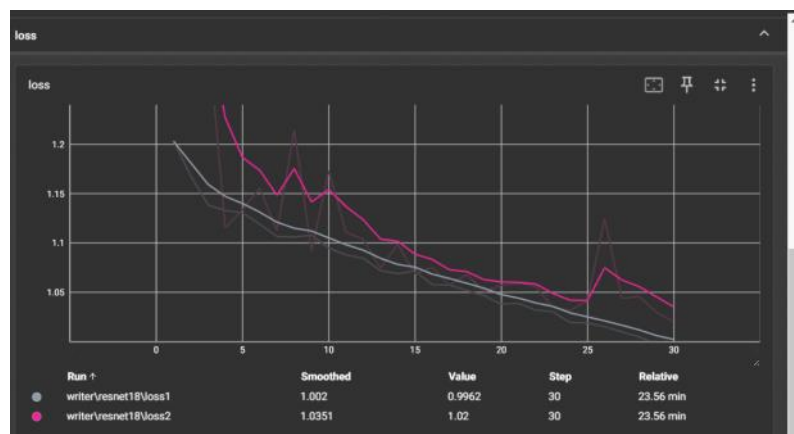
啊，dropout0.85了，还是过拟合



让我想想，可能是由于34的网络也不行？
再改成18吧。

Dropout0.15-lr0.0011

非常好，有些过拟合，

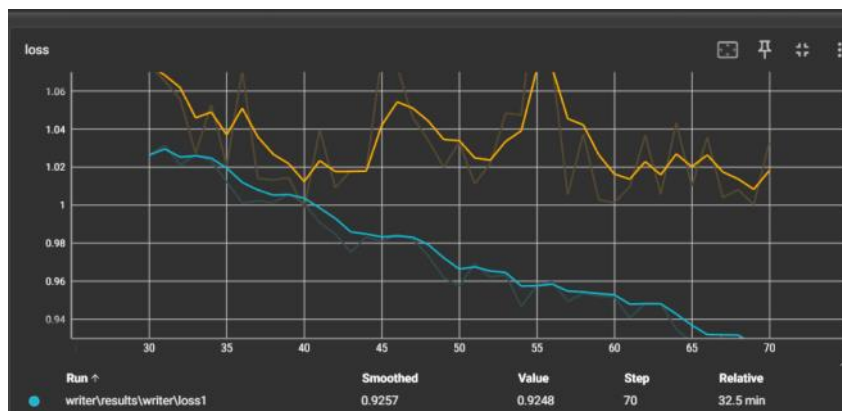


把dropout改成0.25，依然过拟合，改成0.45

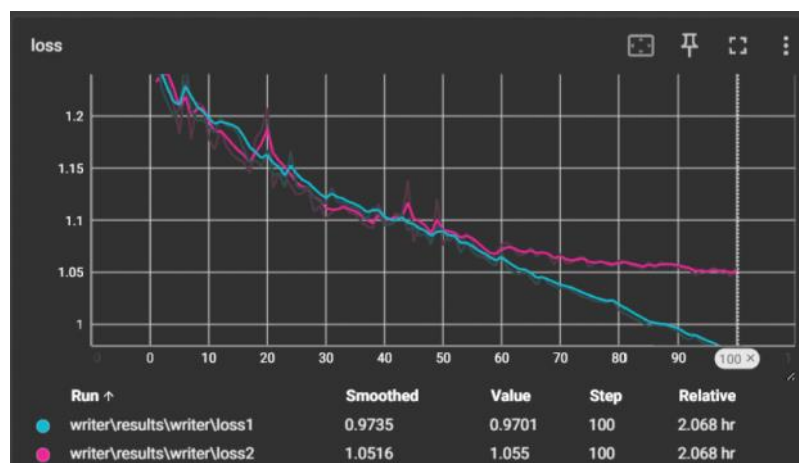
啊，过拟合好严重

难道说，18也太复杂了？

不知道啊，把dropout改成0.65，



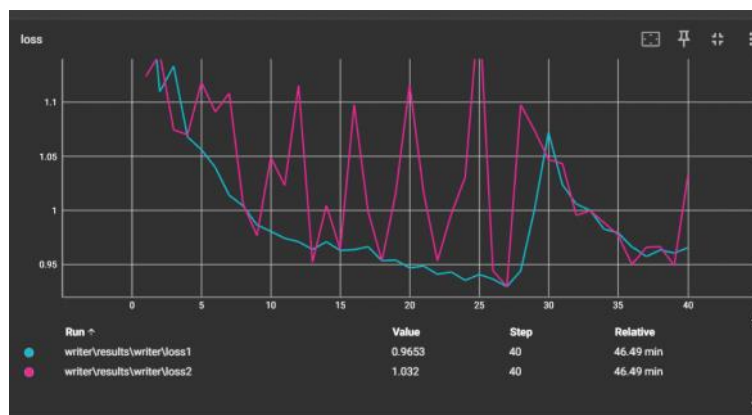
麻了，就这样吧，放弃了



应该是数据集不行，换数据集

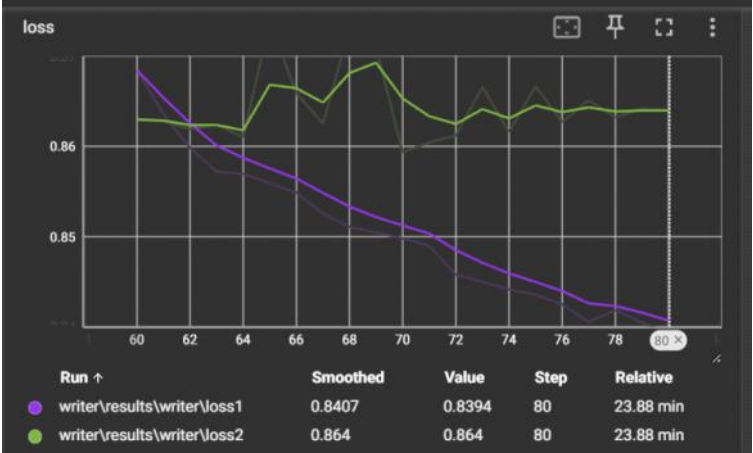
这个达到0.81了，但是震荡太大，

说明学习率过高了



很好，换了数据集，
现在经过多次调试，终于达到了这样的情况

0.883



新想法：

2023年12月16日 19:09

1、12.16：让机器学习新知识，不是单纯的会做题、会计算，而是让他知道为什么？



以女孩和男孩为例，女孩为0，男孩为1
传统的神经网络，自动提取特征，比如提取到是头发，
通过卷积计算后，长头发计算得到的结果更接近于0，而短头发计算的结果更接近于1。
之前之前的方法，不管什么地方，反正就是都做一遍卷积

网络是如何知道要提取头发这一特征的呢？

GooGlenet:在每个神经元那，先设置了多种可能，比如，1*1卷积，3*3卷积，然后再自适应挑选

但是人类在识别时，人类会有自己的思考，会有自己的辨别，
比如当人类在看到这幅图片时，会