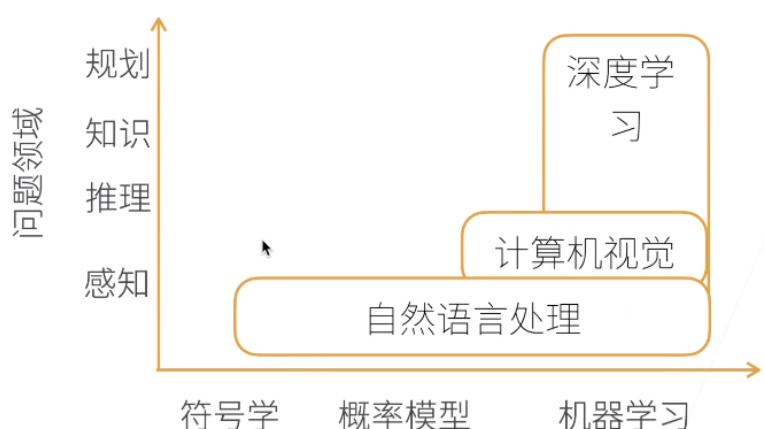


深度学习

什么是深度学习

AI 地图



预测与训练



N维数组样例

- N维数组是机器学习和神经网络的主要数据结构

0-d (标量)



1.0

一个类别

1-d (向量)



[1.0, 2.7, 3.4]

一个特征向量

2-d (矩阵)



[[1.0, 2.7, 3.4]
[5.0, 0.2, 4.6]
[4.3, 8.5, 0.2]]

一个样本—特征矩阵

3-d



[[[0.1, 2.7, 3.4]
[5.0, 0.2, 4.6]
[4.3, 8.5, 0.2]]]
[[3.2, 5.7, 3.4]
[5.4, 6.2, 3.2]
[4.1, 3.5, 6.2]]]

RGB图片 (宽x高x通道)

4-d



[[[[. . .
. . .]]]]

一个RGB图片

批量 (批量大小
x宽x高x通道)

5-d



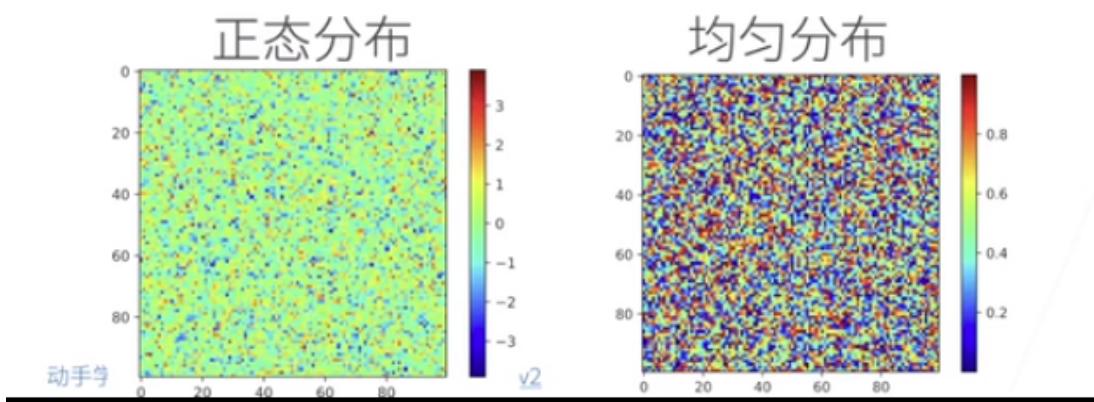
[[[[[. . .
. . .]]]]]

一个视频批量 (批量大小
x时间x宽x高x通道)



创建数组

- 创建数组需要
 - 形状：例如 3×4 矩阵
 - 每个元素的数据类型：例如32位浮点数
 - 每个元素的值，例如全是0，或者随机数



访问元素

一个元素: `[1, 2]`

0	1	2	3
1	5	6	7
2	9	10	11
3	13	14	15

一行: `[1, :]`

0	1	2	3
1	5	6	7
2	9	10	11
3	13	14	15

一列: `[1, :]`

0	1	2	3
1	5	6	7
2	9	10	11
3	13	14	15

子区域: `[1:3, 1:]`

0	1	2	3
1	5	6	7
2	9	10	11
3	13	14	15

子区域: `[:, ::2]`

0	1	2	3
1	5	6	7
2	9	10	11
3	13	14	15

子区域: `[1:3, 1:]` 拿到[1,3)行, 拿到[1,)列, 得到上面的第四个结果

第五个结果：跳着取行-->每三行取一行，得到第0行和第3行，跳着取列，得到0列和第2列。

向量

- 简单操作

$$c = a + b \quad \text{where } c_i = a_i + b_i$$

$$c = \alpha \cdot b \quad \text{where } c_i = \alpha b_i$$

$$c = \sin a \quad \text{where } c_i = \sin a_i$$

- 长度

$$\|a\|_2 = \left[\sum_{i=1}^m a_i^2 \right]^{\frac{1}{2}}$$

$$\|a\| \geq 0 \text{ for all } a$$

$$\|a + b\| \leq \|a\| + \|b\|$$

$$\|a \cdot b\| = |a| \cdot \|b\|$$

动手学深度学习 v2 • <https://courses.csail.mit.edu/6.S090/>

矩阵

- 范数

$$c = A \cdot b \text{ hence } \|c\| \leq \|A\| \cdot \|b\|$$

• 取决于如何衡量 b 和 c 的长度

- 常见范数

• 矩阵范数：最小的满足的上面公式的值

• Frobenius 范数

$$\|A\|_{\text{Frob}} = \left[\sum_{ij} A_{ij}^2 \right]^{\frac{1}{2}}$$

动手学深度学习 v2 • <https://courses.csail.mit.edu/6.S090/>

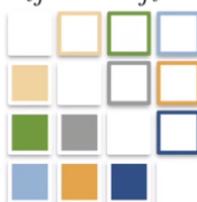
特殊矩阵



- 对称和反对称



$$A_{ij} = A_{ji} \text{ and } A_{ij} = -A_{ji}$$



- 正定

$$\|x\|^2 = x^\top x \geq 0 \text{ generalizes to } x^\top Ax \geq 0$$



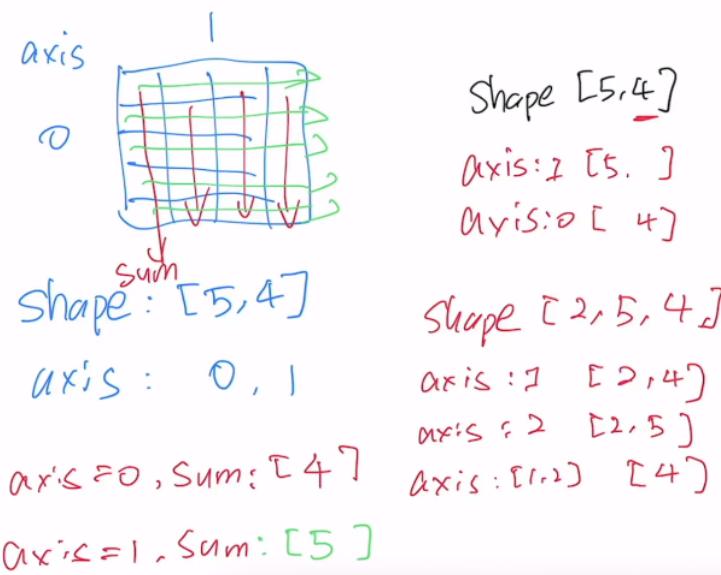
- 特征向量和特征值

- 不被矩阵改变方向的向量

$$Ax = \lambda x$$



- 对称矩阵总是可以找到特征向量



`keepdim=True`

`shape: [2, 5, 4]`

`axis: 1 [2, 1, 4]`

`axis: [1, 2] [2, 1, 1]`

torch不区分行向量和列向量吗？

torch中的行向量就是一维的张量，而列向量则是用二维的矩阵来表示。如果想要区分两者，则需要将两者都用矩阵表示，行向量用列为1，行为n的矩阵来表示。而列向量则使用行为1，列为n的矩阵来表示。

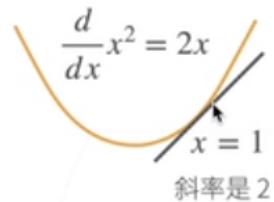
矩阵计算

标量导数

y	a	x^n	$\exp(x)$	$\log(x)$	$\sin(x)$
$\frac{dy}{dx}$	0	nx^{n-1}	$\exp(x)$	$\frac{1}{x}$	$\cos(x)$

a 不是 x 的函数

导数是切线的斜率



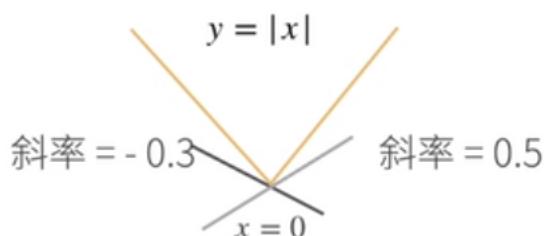
y	$u + v$	uv	$y = f(u), u = g(x)$
$\frac{dy}{dx}$	$\frac{du}{dx} + \frac{dv}{dx}$	$\frac{du}{dx}v + \frac{dv}{dx}u$	$\frac{dy}{du} \frac{du}{dx}$



亚导数

- 将导数拓展到不可微的函数

另一个例子



$$\frac{\partial}{\partial x} \max(x, 0) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \\ a & \text{if } x = 0, \quad a \in [0, 1] \end{cases}$$

$$\frac{\partial |x|}{\partial x} = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \\ a & \text{if } x = 0, \quad a \in [-1, 1] \end{cases}$$



梯度

- 将导数拓展到向量

	向量		
标量	x	$\frac{\partial y}{\partial x}$	$\frac{\partial y}{\partial \mathbf{x}}$
标量	y	$\frac{\partial y}{\partial x}$	$\frac{\partial y}{\partial \mathbf{x}}$
向量	\mathbf{y}	$\frac{\partial \mathbf{y}}{\partial x}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$

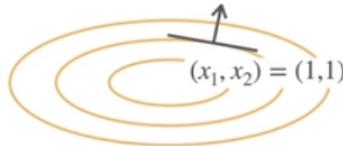
如果 \mathbf{x} 和 \mathbf{y} 都是向量，则求导得到一个矩阵。

$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \left[\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, \dots, \frac{\partial y}{\partial x_n} \right]$$

x	$\frac{\partial y}{\partial x}$	$\frac{\partial y}{\partial \mathbf{x}}$
\mathbf{y}	$\frac{\partial \mathbf{y}}{\partial x}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$

$$\frac{\partial}{\partial \mathbf{x}} x_1^2 + 2x_2^2 = [2x_1, 4x_2] \quad \text{方向 } (2, 4) \text{ 跟等高线正交}$$



如果 \mathbf{x} 是一个向量（列向量）， y 是一个标量，则 y 对于 \mathbf{x} 的导数也是一个向量（行向量）。将 $(x_1)^2 + 2(x_2)^2$ 画成等高线的形式，我们求出了其导数，如果 (x_1, x_2) 为 $(1, 1)$ ，则将其带入求出的导数，得到向量 $(2, 4)$ 就是 $(1, 1)$ 点的梯度，该向量与等高线正交，该向量指向值变化最大的方向。

样例

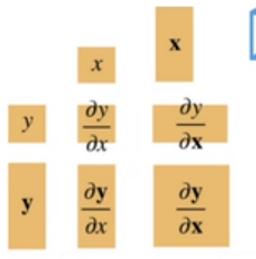
$$\begin{array}{c|cccc} y & a & au & \text{sum}(\mathbf{x}) & \|\mathbf{x}\|^2 \\ \hline \frac{\partial y}{\partial \mathbf{x}} & \mathbf{0}^T & a \frac{\partial u}{\partial \mathbf{x}} & \mathbf{1}^T & 2\mathbf{x}^T \end{array} \quad \begin{aligned} a &\text{ is not a function of } \mathbf{x} \\ \mathbf{0} \text{ and } \mathbf{1} &\text{ are vectors} \end{aligned}$$

$$\begin{array}{c|ccccc} y & u+v & uv & \langle \mathbf{u}, \mathbf{v} \rangle \\ \hline \frac{\partial y}{\partial \mathbf{x}} & \frac{\partial u}{\partial \mathbf{x}} + \frac{\partial v}{\partial \mathbf{x}} & \frac{\partial u}{\partial \mathbf{x}} v + \frac{\partial v}{\partial \mathbf{x}} u & \mathbf{u}^T \frac{\partial \mathbf{v}}{\partial \mathbf{x}} + \mathbf{v}^T \frac{\partial \mathbf{u}}{\partial \mathbf{x}} \end{array}$$



$\partial \mathbf{y} / \partial x$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \quad \frac{\partial \mathbf{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} \\ \frac{\partial y_2}{\partial x} \\ \vdots \\ \frac{\partial y_m}{\partial x} \end{bmatrix}$$



$\partial y / \partial \mathbf{x}$ 是行向量, $\partial \mathbf{y} / \partial x$ 是列向量

这个被称之为分子布局符号, 反过来的
版本叫分母布局符号



如果y为向量(列

向量), x为标量, 则求导结果仍然是列向量, 如上图所示。

总结: x列向量, y标量, 求导结果行向量; x标量, y列向量, 求导结果列向量。这种称为分子布局符号, 也可以使用分母布局符号, 但是必须采用一种布局。

$\partial \mathbf{y} / \partial \mathbf{x}$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial \mathbf{x}} \\ \frac{\partial y_2}{\partial \mathbf{x}} \\ \vdots \\ \frac{\partial y_m}{\partial \mathbf{x}} \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1}, \frac{\partial y_1}{\partial x_2}, \dots, \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1}, \frac{\partial y_2}{\partial x_2}, \dots, \frac{\partial y_2}{\partial x_n} \\ \vdots \\ \frac{\partial y_m}{\partial x_1}, \frac{\partial y_m}{\partial x_2}, \dots, \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

先将y看作向量, 得到一个列向量, 然
后再将x看作向量, 得到一个行向量, 最后得到一个矩阵

为什么深度学习中一般对标量求导而非对矩阵或向量? 这是因为loss一般都是标量

多个损失函数在神经网络中是需要累加梯度的.

• • •

\mathbf{y}	\mathbf{a}	\mathbf{x}	\mathbf{Ax}	$\mathbf{x}^T \mathbf{A}$	$\mathbf{x} \in \mathbb{R}^n, \quad \mathbf{y} \in \mathbb{R}^m, \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \in \mathbb{R}^{m \times n}$
$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$	$\mathbf{0}$	\mathbf{I}	\mathbf{A}	\mathbf{A}^T	a, \mathbf{a} and \mathbf{A} are not functions of \mathbf{x} $\mathbf{0}$ and \mathbf{I} are matrices

\mathbf{y}	$a\mathbf{u}$	\mathbf{Au}	$\mathbf{u} + \mathbf{v}$
$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$	$a \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$	$\mathbf{A} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{u}}{\partial \mathbf{x}} + \frac{\partial \mathbf{v}}{\partial \mathbf{x}}$



拓展到矩阵

	标量	向量	矩阵
标量	x (1,)	\mathbf{x} ($n, 1$)	\mathbf{X} (n, k)
向量	y (1,)	$\frac{\partial y}{\partial x}$ (1,)	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ ($1, n$)
矩阵	\mathbf{Y} (m, l)	$\frac{\partial \mathbf{Y}}{\partial \mathbf{x}}$ (m, l)	$\frac{\partial \mathbf{Y}}{\partial \mathbf{x}}$ (m, k, n)



自动求导

向量链式法则

- 标量链式法则

$$y = f(u), u = g(x) \quad \frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \frac{\partial u}{\partial x}$$

- 拓展到向量

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial u} \frac{\partial u}{\partial \mathbf{x}} \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

(1,n) (1,) (1,n) (1,n) (1,k) (k,n) (m,n) (m,k) (k,n)

例子 1

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

假设 $\mathbf{x}, \mathbf{w} \in \mathbb{R}^n, y \in \mathbb{R}$

$$z = (\langle \mathbf{x}, \mathbf{w} \rangle - y)^2$$

计算

$$\begin{aligned} \frac{\partial z}{\partial \mathbf{w}} &= \frac{\partial z}{\partial b} \frac{\partial b}{\partial a} \frac{\partial a}{\partial \mathbf{w}} \\ &= \frac{\partial b^2}{\partial b} \frac{\partial a - y}{\partial a} \frac{\partial \langle \mathbf{x}, \mathbf{w} \rangle}{\partial \mathbf{w}} \end{aligned}$$



分解

$$\begin{aligned} a &= \langle \mathbf{x}, \mathbf{w} \rangle \\ b &= a - y \\ z &= b^2 \end{aligned}$$

$$= 2b \cdot 1 \cdot \mathbf{x}^T$$

$$= 2(\langle \mathbf{x}, \mathbf{w} \rangle - y) \mathbf{x}^T$$



如果 \mathbf{x}, \mathbf{w} 为向量， y 为标量

例子 2

假设 $\mathbf{X} \in \mathbb{R}^{m \times n}$, $\mathbf{w} \in \mathbb{R}^n$, $\mathbf{y} \in \mathbb{R}^m$

$$z = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

计算

$$\begin{aligned}\frac{\partial z}{\partial \mathbf{w}} &= \frac{\partial z}{\partial \mathbf{b}} \frac{\partial \mathbf{b}}{\partial \mathbf{a}} \frac{\partial \mathbf{a}}{\partial \mathbf{w}} \\ &= \frac{\partial \|\mathbf{b}\|^2}{\partial \mathbf{b}} \frac{\partial \mathbf{a} - \mathbf{y}}{\partial \mathbf{a}} \frac{\partial \mathbf{X}\mathbf{w}}{\partial \mathbf{w}} \\ &= 2\mathbf{b}^T \times \mathbf{I} \times \mathbf{X} \\ &= 2(\mathbf{X}\mathbf{w} - \mathbf{y})^T \mathbf{X}\end{aligned}$$

分解

$$\begin{aligned}\mathbf{a} &= \mathbf{X}\mathbf{w} \\ \mathbf{b} &= \mathbf{a} - \mathbf{y} \\ z &= \|\mathbf{b}\|^2\end{aligned}$$

一个向量, y 也是一个向量。

x为m*n的矩阵，w为

自动求导

- 自动求导计算一个函数在指定值上的导数
- 它有别于
 - 符号求导

```
In[1]:= D[4 x3 + x2 + 3, x]
```

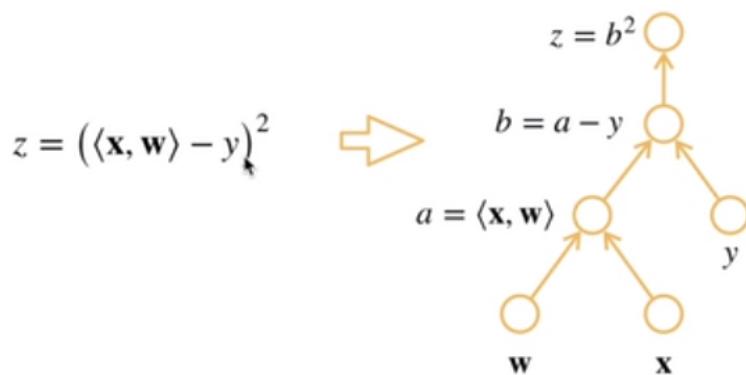
```
Out[1]= 2 x + 12 x2
```

- 数值求导

$$\frac{\partial f(x)}{\partial x} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

计算图

- 将代码分解成操作子
- 将计算表示成一个无环图



计算图

- 将代码分解成操作子
- 将计算表示成一个无环图
- 显示构造

```
from mxnet import sym

a = sym.var()
b = sym.var()
c = 2 * a + b
# bind data into a and b later
```

计算图

- 将代码分解成操作子
- 将计算表示成一个无环图
- 显式构造
 - Tensorflow/Theano/MXNet
 - 隐式构造
 - PyTorch/MXNet

```
from mxnet import autograd, nd

with autograd.record():
    a = nd.ones((2,1))
    b = nd.ones((2,1))
    c = 2 * a + b
```



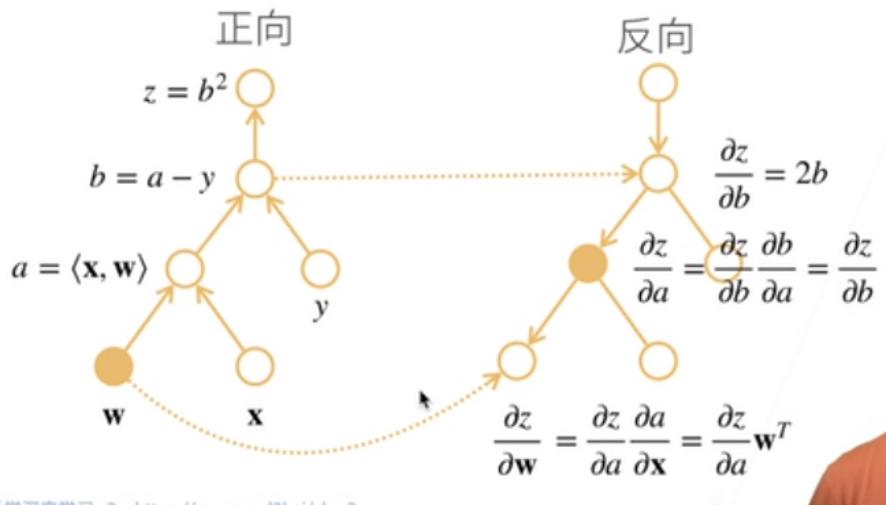
自动求导的两种模式

- 链式法则: $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u_n} \frac{\partial u_n}{\partial u_{n-1}} \dots \frac{\partial u_2}{\partial u_1} \frac{\partial u_1}{\partial x}$
- 正向累积 $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u_n} \left(\frac{\partial u_n}{\partial u_{n-1}} \left(\dots \left(\frac{\partial u_2}{\partial u_1} \frac{\partial u_1}{\partial x} \right) \right) \right)$
- 反向累积、又称反向传递

$$\frac{\partial y}{\partial x} = \left(\left(\left(\frac{\partial y}{\partial u_n} \frac{\partial u_n}{\partial u_{n-1}} \right) \dots \right) \frac{\partial u_2}{\partial u_1} \right) \frac{\partial u_1}{\partial x}$$

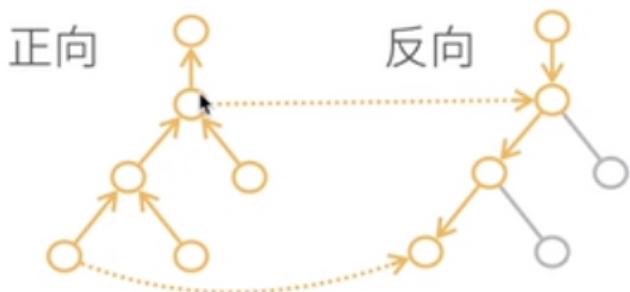
反向累积

$$z = (\langle \mathbf{x}, \mathbf{w} \rangle - y)^2$$



反向累积总结

- 构造计算图
- 前向：执行图，存储中间结果
- 反向：从相反方向执行图
 - 去除不需要的枝



复杂度

- 计算复杂度： $O(n)$, n 是操作子个数
 - 通常正向和反向的代价类似
- 内存复杂度： $O(n)$, 因为需要存储正向的所有中间结果
- 跟正向累积对比：
 - $O(n)$ 计算复杂度用来计算一个变量的梯度
 - $O(1)$ 内存复杂度



线性回归

一个简化模型



- 假设 1：影响房价的关键因素是卧室个数，卫生间个数，和居住面积，记为 x_1, x_2, x_3
- 假设 2：成交价是关键因素的加权和

$$y = w_1x_1 + w_2x_2 + w_3x_3 + b$$

权重和偏差的实际值在后面决定



线性模型

- 给定 n 维输入 $\mathbf{x} = [x_1, x_2, \dots, x_n]^T$
- 线性模型有一个 n 维权重和一个标量偏差

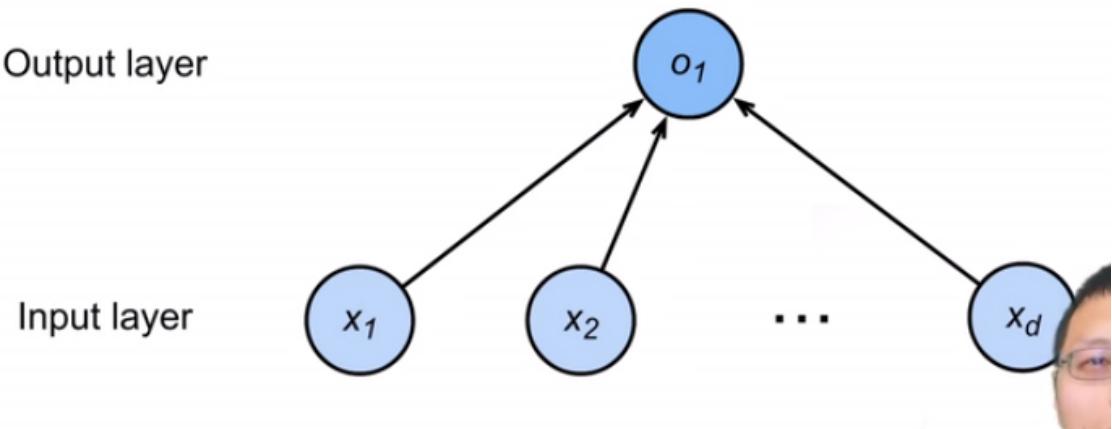
$$\mathbf{w} = [w_1, w_2, \dots, w_n]^T, \quad b$$

- 输出是输入的加权和

$$y = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$$

向量版本： $y = \langle \mathbf{w}, \mathbf{x} \rangle + b$

线性模型可以看做是单层神经网络



上图中箭头代表着权重，而带有权重的只有输入层这一层，因此被称为单层神经网络。

衡量预估质量

- 比较真实值和预估值，例如房屋售价和估价
- 假设 y 是真实值， \hat{y} 是估计值，我们可以比较

$$\ell(y, \hat{y}) = \frac{1}{2} (y - \hat{y})^2$$

这个叫做平方损失

训练数据



- 收集一些数据点来决定参数值（权重和偏差），例如过去6个月卖的房子
- 这被称之为训练数据
- 通常越多越好
- 假设我们有 n 个样本，记

$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]^T \quad \mathbf{y} = [y_1, y_2, \dots, y_n]^T$$



参数学习



- 训练损失

$$\ell(\mathbf{X}, \mathbf{y}, \mathbf{w}, b) = \frac{1}{2n} \sum_{i=1}^n (y_i - \langle \mathbf{x}_i, \mathbf{w} \rangle - b)^2 = \frac{1}{2n} \| \mathbf{y} - \mathbf{X}\mathbf{w} - b \| ^2$$

- 最小化损失来学习参数

$$\mathbf{w}^*, \mathbf{b}^* = \arg \min_{\mathbf{w}, b} \ell(\mathbf{X}, \mathbf{y}, \mathbf{w}, b)$$



损失函数--> 就是 $\{y_i(\text{实际价格}) - [(x_i, w) + b]\}^2$ 的和，也就是每个实际值与估计值的损失的和，也可以写成后面的向量的形式。我们的目标就是找到一个 w 和 b 使得损失最小，最后将这个 w 和 b 作为我们的解，记为 w^* 和 b^* 。

显示解



- 将偏差加入权重 $\mathbf{X} \leftarrow [\mathbf{X}, \mathbf{1}] \quad \mathbf{w} \leftarrow \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}$

$$\ell(\mathbf{X}, \mathbf{y}, \mathbf{w}) = \frac{1}{2n} \parallel \mathbf{y} - \mathbf{X}\mathbf{w} \parallel^2 \quad \frac{\partial}{\partial \mathbf{w}} \ell(\mathbf{X}, \mathbf{y}, \mathbf{w}) = \frac{1}{n} (\mathbf{y} - \mathbf{X}\mathbf{w})^T \mathbf{X}$$

- 损失是凸函数，所以最优解满足

$$\begin{aligned} \frac{\partial}{\partial \mathbf{w}} \ell(\mathbf{X}, \mathbf{y}, \mathbf{w}) &= 0 \\ \Leftrightarrow \frac{1}{n} (\mathbf{y} - \mathbf{X}\mathbf{w})^T \mathbf{X} &= 0 \\ \Leftrightarrow \mathbf{w}^* &= (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}\mathbf{y} \end{aligned}$$



我们可以将b放入w权重中，在x中加入一项1，就可以将损失函数的形式简化。然后我们对这一形式求偏导，我们认为凸函数的最优解一定是在其偏导数等于0的位置，最后带入损失函数得到最优解的形式。

总结

- 线性回归是对n维输入的加权，外加偏差
- 使用平方损失来衡量预测值和真实值的差异
- 线性回归有显示解
- 线性回归可以看做是单层神经网络

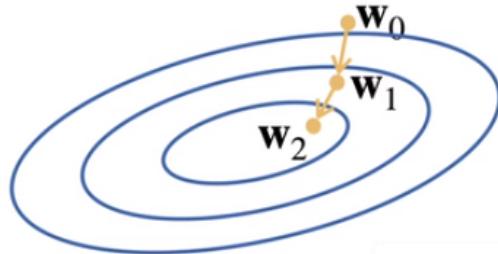
基础优化方法

梯度下降

- 挑选一个初始值 \mathbf{w}_0
- 重复迭代参数 $t=1,2,3$

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \frac{\partial \ell}{\partial \mathbf{w}_{t-1}}$$

- 沿梯度方向将增加损失函数值
- 学习率：步长的超参数

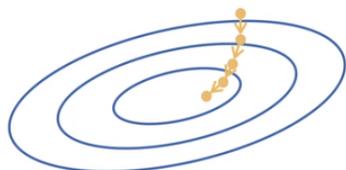


如果我们的模型没有最优解，则可以通过梯度下降的方法来逼近最优解。我们可以这么理解梯度下降，首先我们随机选取初始值 w_0 ，然后我们沿着 w_0 的负梯度方向（梯度方向是值上升最快的方向，则负梯度方向就是值上升最慢的方向，那选取 w_0 的负梯度方向就可以使得损失函数减小）来前进选取参数，我们将计算 w_t 的第二项作为向负梯度前进的步长，即每次从 w_{t-1} 向反方向前进多少距离。

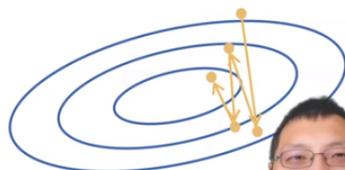
其中学习率是人为选择的超参数，如果我们前面的梯度损失函数没有求均值，那我们可以将学习率除以 n ，效果是一样的

选择学习率

不能太小



也不能太大



学习率不能选择的太小，否则接

近结果所花费的时间过长，计算梯度花费资源过多。但也不能选择的太小，否则步长过大，迈过了合适的 w ，使得整个训练过程一直在震荡，损失函数并没有发生实际减小。

小批量随机梯度下降

- 在整个训练集上算梯度太贵
 - 一个深度神经网络模型可能需要数分钟至数小时
- 我们可以随机采样 b 个样本 i_1, i_2, \dots, i_b 来近似损失

$$\frac{1}{b} \sum_{i \in I_b} \ell(\mathbf{x}_i, y_i, \mathbf{w})$$

- b 是批量大小，另一个重要的超参数

由于计算整个训练集的梯度过于昂贵，则我们可以选择批量 b ，来减小计算量，这个参数是另外一个重要的超参数。

选择批量大小

不能太小

不能太大

每次计算量太小，不适合并行来最大利用计算资源

内存消耗增加
浪费计算，例如如果所有样本都是相同的

总结

- 梯度下降通过不断沿着反梯度方向更新参数求解
- 小批量随机梯度下降是深度学习默认的求解算法
- 两个重要的超参数是批量大小和学习率

为什么采用平方损失而非绝对差值？因为绝对差值不可导，但其实区别不大

batch_size 是否会最终影响模型结果？在同样的计算次数的情况下（扫数据扫十遍），batch_size 越小实际上会对收敛越有利，因为从理论上说，随机梯度下降会带来噪音，如果样本很大，但是我们选择的 batch_size 很小，则会产生很大的噪音，有一定的噪音会使得模型鲁棒性更好

即使 batch_size 很大（没有很夸张），只要花费足够的时间，还是能够收敛的

问题10：针对 batchsize 大小的数据集进行网络训练的时候，网络中每个

参数更新时的减去的梯度是 batchsize 中每个样本对应参数梯度求和后取得
平均值吗？

是的，也就是说，我们通过一个批次的训练得到了新的参数，这个参数中的梯度是这个批次的数据的梯度的平均值，是一块数据的平均值，而不是一个数据。

问题11：随机梯度下降中的“随机”是指的批量大小是随机的吗？

每次随机采样batch_size大小的数据。

不是的，随机指的是

问题17：这样的data-iter写法，每次都把所有输入load进去，如果数据多

的话，最后内存会爆掉吧？有什么好的办法吗？

数据全都读入内存，只需批量大小的数据即可，利用GPU的情况下，一般是不会爆内存的

我们不需要将

问题20：这里使用生成器生成数据有什么优势呢 相比return？ 使用yield

的好处是，我们不需要先生成所有的batches，而是需要就去调用一遍即可。而且书写方便

问题21：如果样本大小不是批量数的整数倍，那需要随机剔除多余的样本

吗？

三种做法：将最后一个批次的样本缩小，如本课实例；第二种是直接丢弃最后一个不满足batch_size的batch；第三种是从下一次扫描中补足缺失的数据

问题24：老师请问这里面是没有进行收敛的判断吗？直接人为设置epoch

的大小吗？

判断收敛

的方法有很多，我们可以判断两次的损失差别不大就认为其收敛，我们也可以使用验证数据集，判断生成的参数应用到模型上对验证数据集的精度是否有所增加

问题25：本质上我们为什么要用SGD，是因为大部分的实际loss太复杂，

推导不出导数为0的解么？只能逐个batch去逼近？

是的

，除了线性回归，其他模型都没有显式解，即使是有显示解，我们也没有必要去花大量的时间去求取它。我们求解的问题都是non complete，而求解non complete就是机器学习的意义所在。

问题27：实际中有时候网络会输出nan，nan到底是怎么出现的？为什么

不是inf？数值稳定性不是会导致inf么？

除法

除0有可能导致这种问题

问题30：外层for循环中最后一行l=loss(net(),labels)就是为了print吗？这

里梯度要不要清零呢？会不会有影响？

是的，我们不

需要清零这个梯度，因为没有更新了，我们只有run forward,没有run backward(没有反向传播求导)

Softmax回归

回归 vs 分类



- 回归估计一个连续值
- 分类预测一个离散类别

MNIST: 手写数字识别 (10类) ImageNet: 自然物体分类 (1000类)

0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9

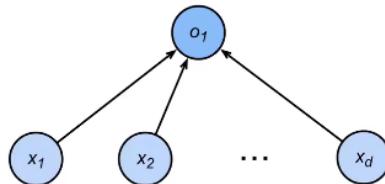


ImageNet

从回归到多类分类

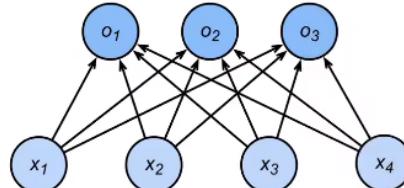
回归

- 单连续数值输出
- 自然区间 \mathbb{R}
- 跟真实值的区别作为损失



分类

- 通常多个输出
- 输出 i 是预测为第 i 类的置信度



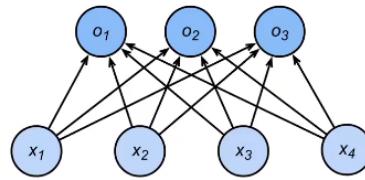
分类问题从回归问题的单输出变为了多输出，输出的个数就等于类别的个数

从回归到多类分类 – 均方损失

- 对类别进行一位有效编码

$$\mathbf{y} = [y_1, y_2, \dots, y_n]^\top$$

$$y_i = \begin{cases} 1 & \text{if } i = y \\ 0 & \text{otherwise} \end{cases}$$



- 使用均方损失训练

- 最大值最为预测

$$\hat{y} = \operatorname{argmax}_i o_i$$



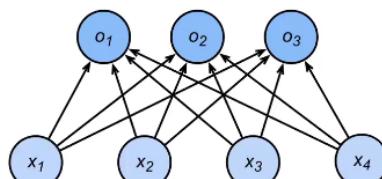
一位有效编码，所有类别中只有一个正确类别的值为1，其余为0

从回归到多类分类 – 无校验比例

- 对类别进行一位有效编码

- 最大值最为预测

$$\hat{y} = \operatorname{argmax}_i o_i$$



- 需要更置信的识别正确类（大余量）

$$o_y - o_i \geq \Delta(y, i)$$



对于分类问题，

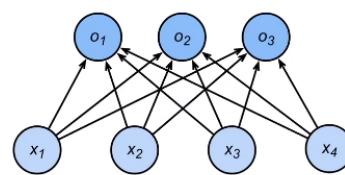
我们其实不在意得到的值有多大，但是我们需要正确类别的置信度 o_y 要远远大于其他非正确类别 o_i （大于一个阈值），这样就可以确保一个正确类与错误类拉开距离，也就是我们在意的是相对值。

从回归到多类分类 – 校验比例

- 输出匹配概率（非负，和为1）

$$\hat{\mathbf{y}} = \operatorname{softmax}(\mathbf{o})$$

$$\hat{y}_i = \frac{\exp(o_i)}{\sum_k \exp(o_k)}$$



- 概率 \mathbf{y} 和 $\hat{\mathbf{y}}$ 的区别作为损失



虽然我们在意相对值

，但是将其值置于一个合理的空间内有利于作出判断。我们作出一个新的操作，即softmax，将其作用再 \mathbf{o} 上，得到一个 $\hat{\mathbf{y}}$ ，是一个长为n的向量，但是它具有每个元素 $>=0$ 且和为1的性质。 y_i 是 \mathbf{o} 做指数（使得所有值均为正）/所有 \mathbf{o} 的指数的和（就可以保证所有值的和为1）这样，我们的 $\hat{\mathbf{y}}$ 就变成了一个概率，实际上实际的 \mathbf{y} 我们也可以作为概率（除正确类 $y=1$ 外其他

y_i 均为0，则所有 y 的和为1，且每个 y 值都不为负）。这样操作过后，我们得到了 y 和 y_{hat} 这两个概率，用他们的区别作为损失。

Softmax 和交叉熵损失

- 交叉熵常用来衡量两个概率的区别 $H(\mathbf{p}, \mathbf{q}) = \sum_i -p_i \log(q_i)$

- 将它作为损失

$$l(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_i y_i \log \hat{y}_i = - \log \hat{y}_y$$

- 其梯度是真实概率和预测概率的区别

$$\partial_{o_i} l(\mathbf{y}, \hat{\mathbf{y}}) = \text{softmax}(\mathbf{o})_i - y_i$$

一般我们使用交叉熵作为损失，上图中第二个公式其实求和只能得到一项，即 $y_i=1$ 的一项，此时 $i=y$ ，就可以得到上面的结果-->对真实类别的预测值求 \log 再取负。这样我们也可以解释-->对于分类问题，我们不关心非正确类的预测值，只关心正确类的预测值的置信度有多大。对损失求梯度，就是真实概率和预测概率的区别 $y_{\text{hat}}-y_i$ 。

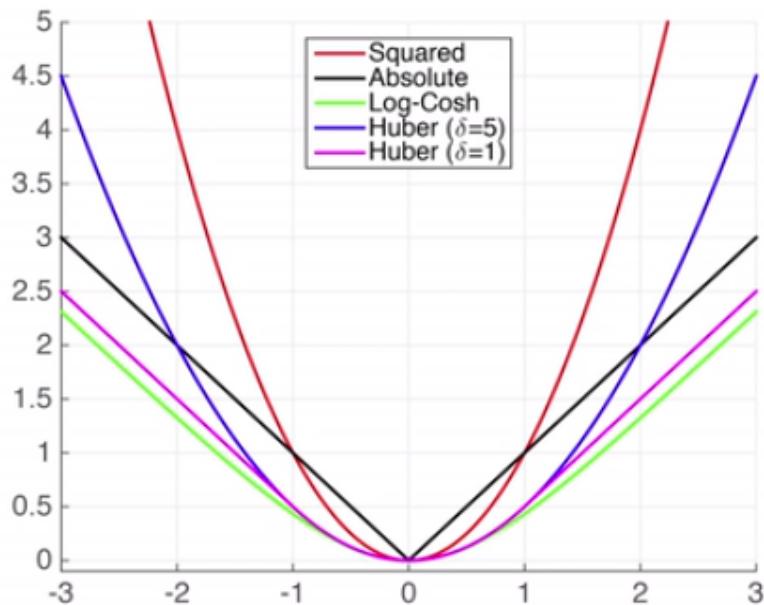


总结

- Softmax 回归是一个多类分类模型
- 使用 Softmax 操作子得到每个类的预测置信度
- 使用交叉熵来衡量预测和标号的区别

损失函数

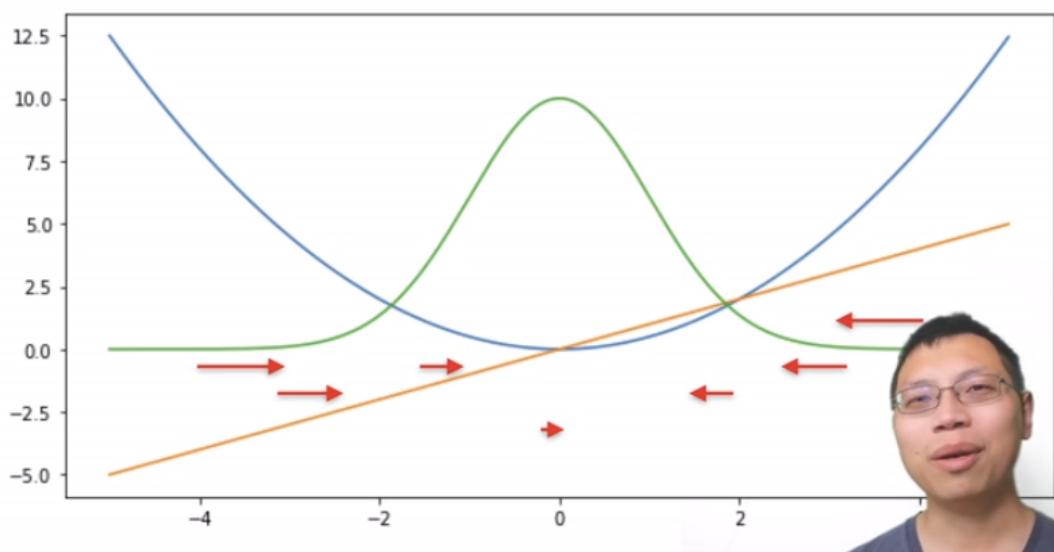
损失函数



平方损失函数L2 Loss

L2 Loss

$$l(y, y') = \frac{1}{2}(y - y')^2$$



蓝色-- $y=0$,变换 y' 时得到的曲线；绿色--似然函数曲线；橙色--L2 Loss的梯度。

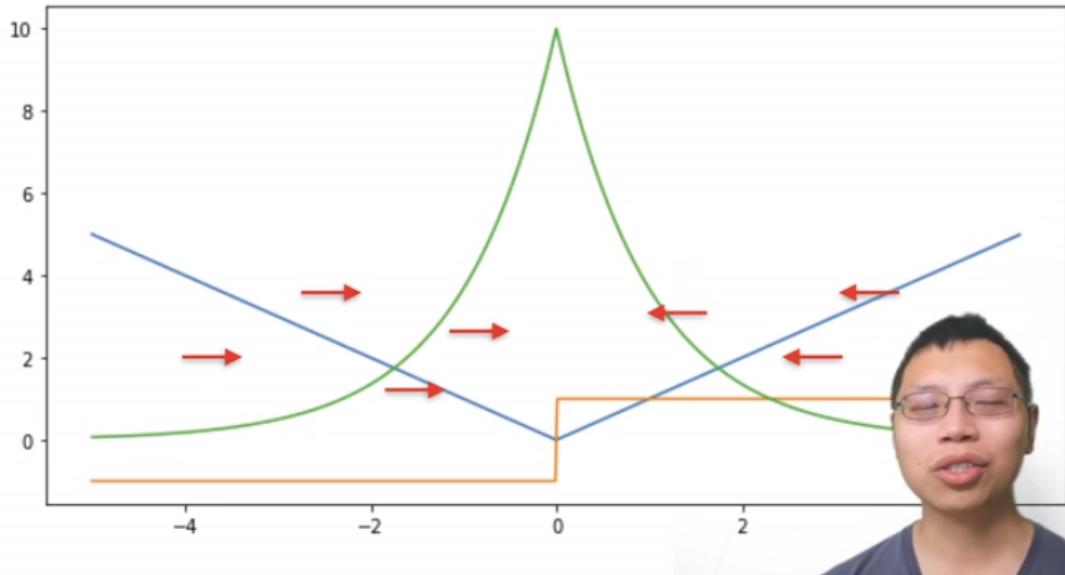
根据梯度下降法，如果 y' 与我们的期望值 y 相距较大，则梯度较大，则下降的梯度也较大，当靠近原点时(即梯度=0，此时损失较小)，梯度会变小，则下降的梯度也会变小(GSD 当前参数=上一参数-上一损失函数的梯度) (这里我们可以参照上图，逼近了我们想要的参数时，损失函数的梯度较小，则第二项即下降梯度就小，参数更新的幅度就变小了；远离了我们想要的参数，

损失函数梯度较大，第二项的值就大，则下降的梯度就大，参数更新的幅度就大) *学习率)。而有的时候，远离原点但我们不想要那么大的参数更新，就使用另外一种损失函数

绝对值损失函数 L1 Loss

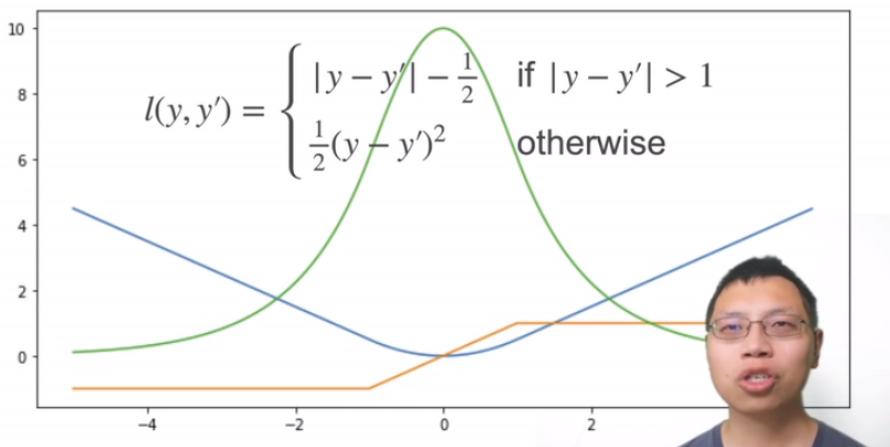
L1 Loss

$$l(y, y') = |y - y'|$$



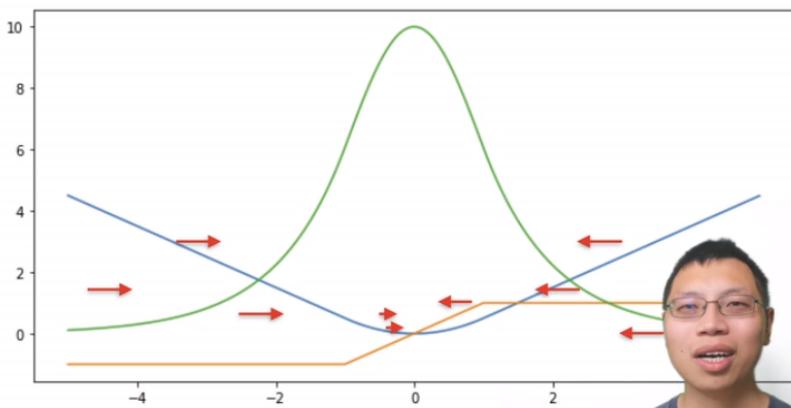
L1 loss的损失函数的梯度，在 $y' > 0$ 时，梯度为1， $y' < 0$ 时，梯度为-1，而在原点处不可导，我们就认为其梯度可以取-1~1。使用L1 loss的时候，参数更新的幅度与原点距离不再相关，而是恒定值，会带来稳定性的优点。但是缺点也很明显：在原点处不可导，当优化末期时，即预测值和真实值相差不多时，就会不那么稳定了

Huber's Robust Loss



结合了两者的特点，在 y' 和 y 的差 > 1 时，我们使用L1 Loss，而在 y' 和 y 的差 < 1 时，我们使用L2 loss。

Huber's Robust Loss



在优化前期，保证一个稳定的参数更新，在优化末期，保证参数更新幅度较小，防止出现数值上的问题

问题2：softmax回归和logistic回归分析是一样的吗，如果不一样的话，

哪些地方不同？

可以将logistic分析

看作二分类的问题，但是它只要一个输出+1或-1即可

问题13：Dataloader() 的num_workers 是并行了嘛？

是

问题17：老师好，pytorch训练好模型，测试的时候发现无论batchsize

设为1还是更多，测试的总时间都差不多，但正常理解如果设成4不应该

是设为1的4倍速度吗

batch_size不

会影响计算量，只会影响并行度

问题19：为什么不在accuracy函数中把除以len(y)做完呢？

cmp.type(y.dtype)是必要的吗？

因为有可能最后

一个batch_size是不完整的，如果直接除以y的长度，就会导致batch_size与y不等，导致结果出错

问题21：老师，在计算精度的时候，为什么需要使用new.eval()将模型设

置成评估模式？

表示不去计算梯度等信息

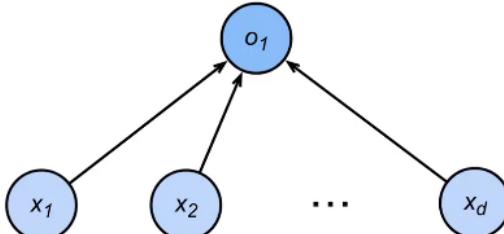
感知机

感知机

感知机

- 给定输入 \mathbf{x} , 权重 \mathbf{w} , 和偏移 b , 感知机输出:

$$o = \sigma(\langle \mathbf{w}, \mathbf{x} \rangle + b) \quad \sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$



其实是一个

二分类的问题

- 二分类: -1 或 1
 - Vs. 回归输出实数
 - Vs. Softmax 回归输出概率

训练感知机

```
initialize w = 0 and b = 0
repeat
    if  $y_i [\underbrace{\langle w, x_i \rangle + b}] \leq 0$  then
         $w \leftarrow w + y_i x_i$  and  $b \leftarrow b + y_i$ 
    end if
until all classified correctly
```

等价于使用批量大小为1的梯度下降，
并使用如下的损失函数

$$\ell(y, \mathbf{x}, \mathbf{w}) = \max(0, -y\langle \mathbf{w}, \mathbf{x} \rangle)$$

如果分类正确，则 \max 得到的结果一定 > 0 ，则 \max 得到 0，那就分类错误

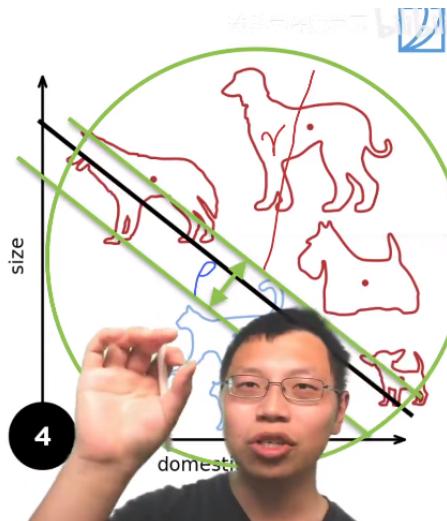
收敛定理

- 数据在半径 r 内
- 余量 ρ 分类两类

$$y(\mathbf{x}^\top \mathbf{w} + b) \geq \rho$$

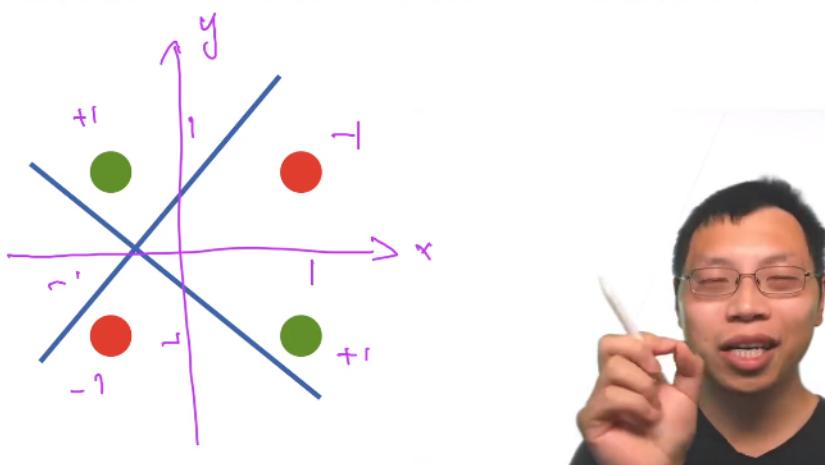
对于 $\|\mathbf{w}\|^2 + b^2 \leq 1$

- 感知机保证在 $\frac{r^2 + 1}{\rho^2}$ 步后收敛



XOR 问题 (Minsky & Papert, 1969)

感知机不能拟合 XOR 函数，它只能产生线性分割面



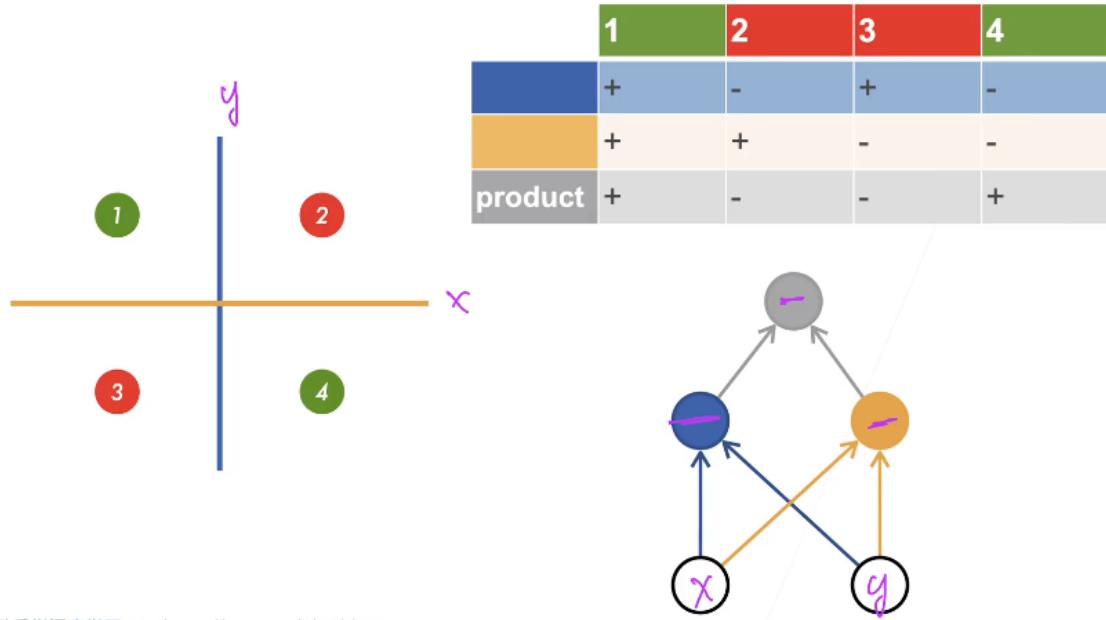
总结

- 感知机是一个二分类模型，是最早的AI模型之一
- 它的求解算法等价于使用批量大小为1的梯度下降
- 它不能拟合 XOR 函数，导致的第一次 AI 寒冬

多层感知机

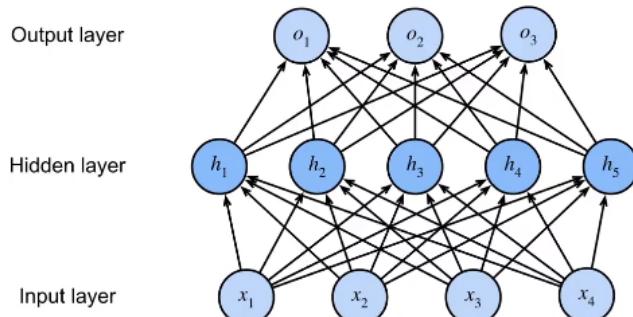
学习 XOR

神经元的输入和输出



先用蓝色的线做一下分类，将2、4记为负，将1、3记为正，然后用橙色的线做一下分类，将1、2记为正，将3、4记为负。然后对这两次分类做与操作，相同的记为一类，不同的记为一类。这样就从一开始的一层变为了多层，称为多层感知机。

单隐藏层



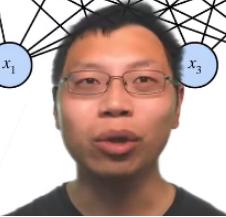
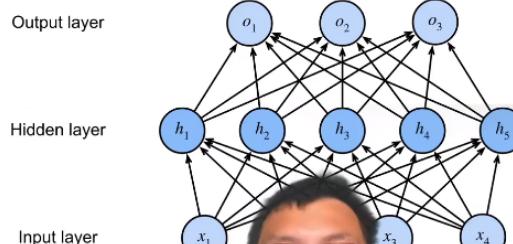
隐藏层大小是超参数

输入的大小是固定的，输出的大小是根据分类来决定的，因此我们能设置的只有隐藏层的大小。

单隐藏层 – 单分类

- 输入 $\mathbf{x} \in \mathbb{R}^n$
- 隐藏层 $\mathbf{W}_1 \in \mathbb{R}^{m \times n}, \mathbf{b}_1 \in \mathbb{R}^m$
- 输出层 $\mathbf{w}_2 \in \mathbb{R}^m, b_2 \in \mathbb{R}$

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$
$$o = \mathbf{w}_2^T \mathbf{h} + b_2$$



σ 是按元素的激活函数

输入为一个长度为 n 的向量；假设隐藏层的大小是 m , 则隐藏层权重 \mathbf{W}_1 就是一个大小为 $m \times n$ 的矩阵，偏执项 \mathbf{b}_1 就是一个长度为 m 的向量，经过隐藏层的输入会变成一个 m 大小的向量，由 \mathbf{w}_1 和 \mathbf{b}_1 两项组成。单分类的输出层只需要一个输出即长度为 m 的向量 \mathbf{w}_2 和一个标量 b_2 ，则经过输出层处理后，得到的是一个标量的输出，即我们所关心的值的输出。

单隐藏层

- 输入 $\mathbf{x} \in \mathbb{R}^n$
- 隐藏层 $\mathbf{W}_1 \in \mathbb{R}^{m \times n}, \mathbf{b}_1 \in \mathbb{R}^m$
- 输出层 $\mathbf{w}_2 \in \mathbb{R}^m, b_2 \in \mathbb{R}$

为什么需要非线性
激活函数?

$$\mathbf{h} = \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1$$

$$o = \mathbf{w}_2^T \mathbf{h} + b_2$$

$$\text{hence } o = \mathbf{w}_2^T \mathbf{W}_1 \mathbf{x} + b'$$

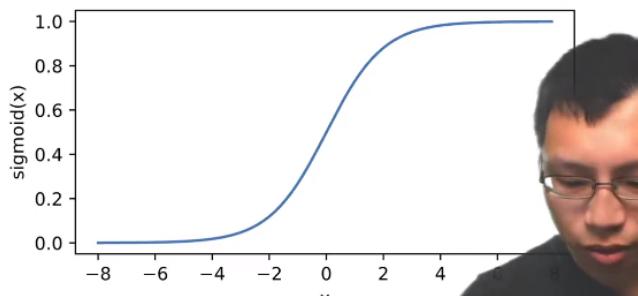
$\mathbf{w}_2^T \mathbf{W}_1$

仍然是线性

Sigmoid 激活函数

将输入投影到 $(0, 1)$, 是一个软的 $\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$

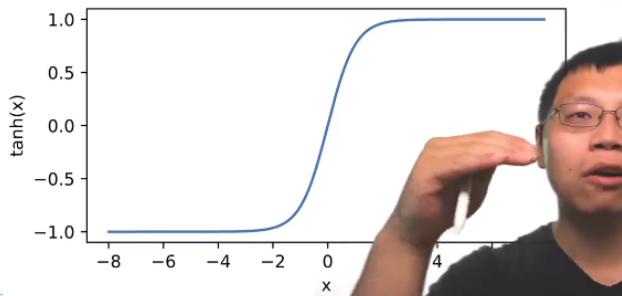
$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$



Tanh 激活函数

将输入投影到 $(-1, 1)$

$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}$$

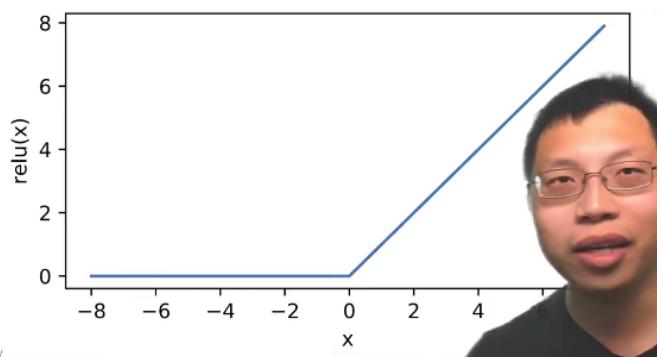


动手学深度学习 v2 · <https://>

ReLU 激活函数

ReLU: rectified linear unit

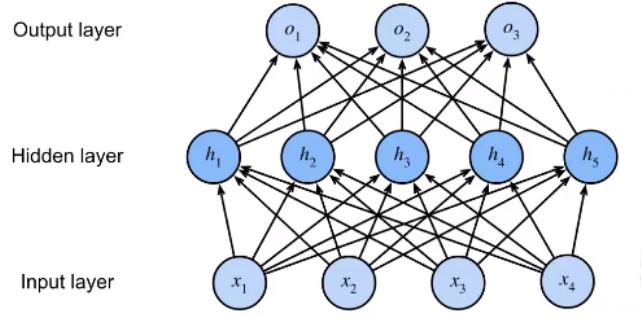
$$\text{ReLU}(x) = \max(x, 0)$$



动手学深度学习 v2 · <https://>

多类分类

$$y_1, y_2, \dots, y_k = \text{softmax}(o_1, o_2, \dots, o_k)$$



在softmax中加入了一

层，实现了多类分类

多类分类

- 输入 $\mathbf{x} \in \mathbb{R}^n$
- 隐藏层 $\mathbf{W}_1 \in \mathbb{R}^{m \times n}, \mathbf{b}_1 \in \mathbb{R}^m$
- 输出层 $\mathbf{W}_2 \in \mathbb{R}^{m \times k}, \mathbf{b}_2 \in \mathbb{R}^k$

Output layer

Hidden layer

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

Input layer

$$\mathbf{o} = \mathbf{W}_2^T \mathbf{h} + \mathbf{b}_2$$

$$\mathbf{y} = \text{softmax}(\mathbf{o})$$

输出的w2变成了m*k，因

为输出需要有k个单元，b也变成了k维的列向量，最后得到就是长度为k的向量，对应k个分类的成绩。和前面的二分类类似，但是输出有k个类别，因此权重w变成了一个矩阵，而偏置项也变了k维的向量。还是先将输入做激活，然后对结果做隐藏层的处理，最后得到最后的输出o，这里的o不需要进行激活，因为激活是为了防止层数的塌陷（即前面的变成单层感知机的情况），这里防止塌陷的层都是指隐藏层，而隐藏层没有必须是非线性的要求，主要就是将隐藏层的输出在做一次操作得到可以操作的输出的作用。

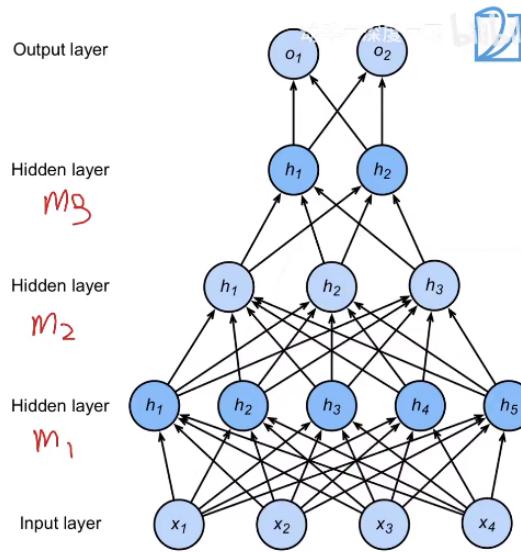
多隐藏层

$$\begin{aligned} h_1 &= \sigma(W_1x + b_1) \\ h_2 &= \sigma(W_2h_1 + b_2) \\ h_3 &= \sigma(W_3h_2 + b_3) \\ o &= W_4h_3 + b_4 \end{aligned}$$

3
=

超参数

- 隐藏层数
- 每层隐藏层的大小



动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

最后的输出层不需要激活，最后一层不需要激活函数。在多隐藏层的情况下，我们需要手动设置的超参数就增多了-->多少个隐藏层，每个隐藏层需要多大。

对于一个较大的输入，隐藏层我们将其慢慢的压缩维度，也就是说，我们可以设置多个隐藏层，并且每个隐藏层的大小慢慢变小，就可以实现这种效果。也可以在最下面一层进行一点点的扩充，如上图所示。但我们不会先压缩再扩充，因为压缩操作会损失较多的信息。

总结

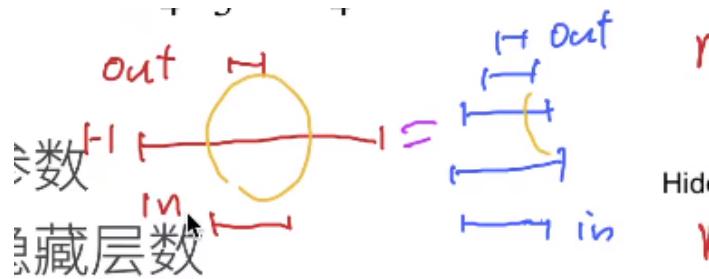
- 多层感知机使用隐藏层和激活函数来得到非线性模型
- 常用激活函数是Sigmoid, Tanh, ReLU
- 使用 Softmax 来处理多类分类
- 超参数为隐藏层数，和各个隐藏层大小



问题2：请问老师神经网络中的一层网络到底是指什么？是一层神经元经过线性变换后称为一层网络，还是一层神经元经过线性变化加非线性变换后称为一层？

一般我们认为有权

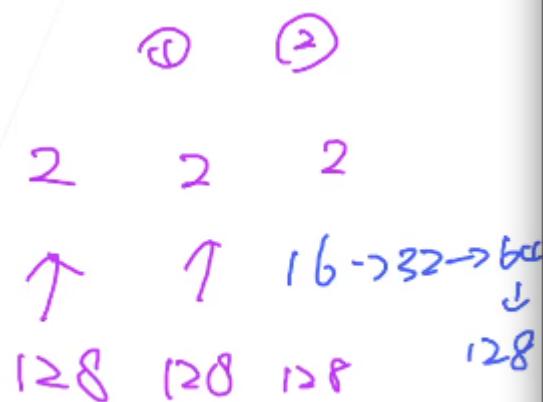
重w的才算作一层，因此单层隐藏层的MLP看起来有输入层，输出层和隐藏层三层，但是我们将输入层不看做一层，只认为有两个可以进行训练得到合适weight和bias的层。



我们可以用不同的方式来设计隐藏层，

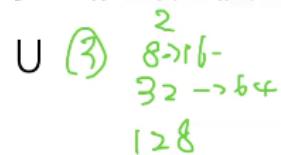
如图所示，虽然理论上两者的效果相同，但是左边的模型难以训练，右边的模型更好训练。我们可以叫左边的为“浅度学习”，一下子学所有特征非常容易过拟合；右边的为“深度学习”，一点点的学习不同的特征，最后拿到结果。

激活函数的作用就是引入非线性



对于一个任务输入为128，输出为2。我们先试线性无隐藏层，即单层感知机；然后再试加入大小为16的隐藏层，然后是大小32、64等等。如果16太简单，128太复杂，32和64的效果还可以。

到非线性模型



我们可以添加第二个隐藏层，用一些更合适的参数，如左图，尝试得到更好地结果。

问题18：老师，在设置隐藏层的时候，会先人为评估特征的数量，然后
再设置层数和单元数么

集做这样的调试，这就是调参，后面会讲

可以使用验证数据

模型选择

训练误差和泛化误差

- 训练误差：模型在训练数据上的误差
- 泛化误差：模型在新数据上的误差
- 例子：根据摸考成绩来预测未来考试分数
 - 在过去的考试中表现很好（训练误差）不代表未来考试一定会好（泛化误差）
 - 学生 A 通过背书在摸考中拿到很好成绩
 - 学生 B 知道答案后面的原因



验证数据集和测试数据集

- 验证数据集：一个用来评估模型好坏的数据集
 - 例如拿出 50% 的训练数据
 - 不要跟训练数据混在一起（常犯错误）
- 测试数据集：只用一次的数据集。例如
 - 未来的考试
 - 我出价的房子的实际成交价
 - 用在 Kaggle 私有排行榜中的数据集



由于验证数据集没有参与训练，就有可能反映出我们模型的好坏。但是验证数据集不能代表模型的泛化能力

K-则交叉验证

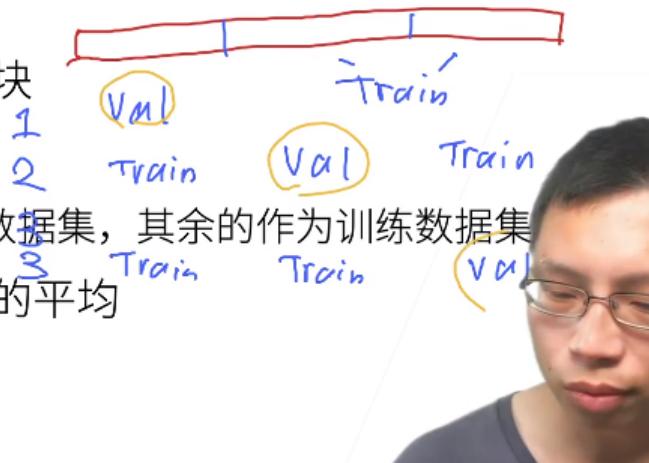
读书笔记

- 在没有足够多数据时使用 (这是常态)

- 算法：

- 将训练数据分割成 K 块
- For $i = 1, \dots, K$
 - 使用第 i 块作为验证数据集，其余的作为训练数据集
 - 报告 K 个验证集误差的平均

- 常用： $K = 5$ 或 10



如上图所示分配验证数据集和训练数据集，最后得到误差的平均

总结

- 训练数据集：训练模型参数
- 验证数据集：选择模型超参数
- 非大数据集上通常使用 k-折交叉验证

过拟合和欠拟合



数据

	简单	复杂
模型容量	正常	欠拟合
高	过拟合	正常

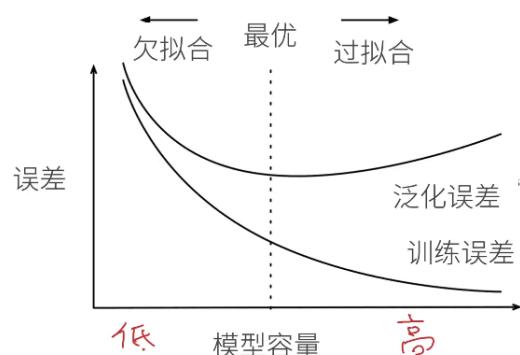
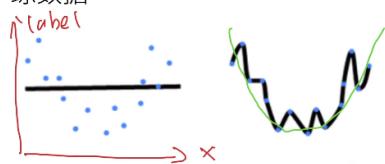
模型容量--即模型的复

杂度，以感知机为例，MLP就比单层的感知机复杂，则模型的性能就更好一点。如果在简单的数据集上使用深度很高的网络，则很有可能过拟合，也就是说模型将每一个样本都记住了，这样有可能导致泛化性不足。而如果使用简单地模型，比如线性模型，就无法拟合XOR函数，对于fashin-mnist也是，如果使用线性模型，有可能模型的性能就不会那么好。

模型容量

模型容量的影响

- 拟合各种函数的能力
- 低容量的模型难以拟合训练数据
- 高容量的模型可以记住所有的训练数据



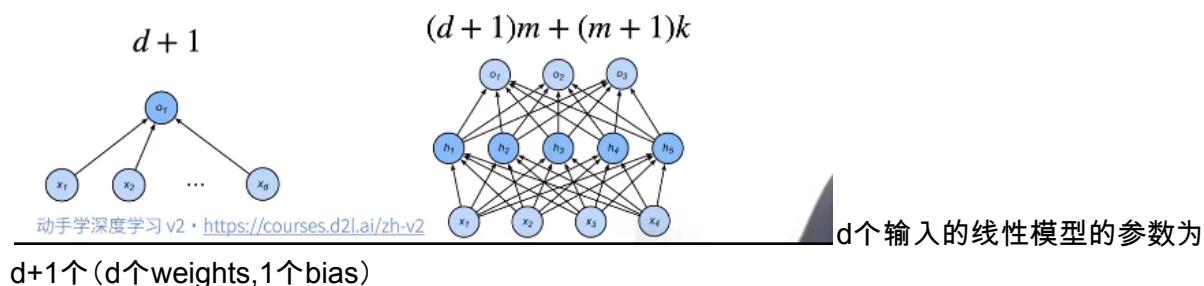
我们可以看到，在模型容量较低时，训练误差很大，但是如果我们提高模型容量，则训练误差就会不断缩小，从理论上来说，只要我们提高模型容量到一个值，可以使得模型拟合所有输入，使得训练误差为0，但是这样模型会拟合很多噪声进去，这其实是不好的。而泛化误差从一开始的提高模型容量使其下降，到达最优点的时候，到底最小值，此时如果继续增加模型容量，就会使得泛化误差上升，进入了过拟合阶段。

总结一下：欠拟合压根没有拟合数据，训练误差和泛化误差都很大；过拟合过于拟合数据，训练误差小但是泛化误差大，没有泛化能力。我们要找到这个平衡点，即泛化误差最小的一个点。而泛化误差和训练误差的gap一般被我们用来判断过拟合和欠拟合的程度。

我们的核心任务就是将泛化误差减小，同时gap也不能太大。为了将泛化误差降低，有时候我们不得不承受一部分的过拟合，过拟合不一定是一件坏事情。首先我们的模型要足够大，然后为了降低泛化误差，我们就可以调整模型容量。

估计模型容量

- 难以在不同的种类算法之间比较
~~树~~
 - 例如数模型和神经网络
- 给定一个模型种类，将有两个主要因素
 - 参数的个数
 - 参数值的选择范围



而对于有一个隐含层，输入为 d ，隐含层为 m ，则参数个数为 $(d+1)*m$ [d 个weights，1个bias，隐藏层容量为 m] + $(m+1)*k$ 个 [m 个weights，1个bias，输出层容量为 k]。

这样我们就可以简单地通过模型个数来判断哪个模型的容量更高。

而如果模型的参数值如果能在较大的范围内选择，则我们模型的复杂度一般就高，否则就较低。

线性分类器的 VC 维

- 2 维输入的感知机，VC 维 = 3
 - 能够分类任何三个点，但不是4个 (xor)



- 支持 N 维输入的感知机的 VC 维是 $N + 1$
- 一些多层次感知机的 VC 维 $O(N \log_2 N)$

我们可以看到，三个点的特征，我们都可以通过二维输入的一条线来分类它，但是对于四个点的XOR问题无法分类了。而且多层次的感知机的VC维是比线性的要高的。

VC 维的用处

- 提供为什么一个模型好的理论依据
 - 它可以衡量训练误差和泛化误差之间的间隔
- 但深度学习中很少使用
 - 衡量不是很准确
 - 计算深度学习模型的 VC 维很困难

数据复杂度

- 多个重要因素
 - 样本个数
 - 每个样本的元素个数
 - 时间、空间结构
 - 多样性



VS



总结

- 模型容量需要匹配数据复杂度，否则可能导致欠拟合和过拟合
- 统计机器学习提供数学工具来衡量模型复杂度
- 实际中一般靠观察训练误差和验证误差



问题1：我感觉svm从理论上讲应该对于分类分体效果不错，和神经网络想比，缺点在哪里？

数据集过大计算缓慢

，可以调整的部分不多

问题3：训练误差是training dataset? 泛化误差是testing dataset?
泛化误差严格来说就是未来的未知数据集上的误差

问题6：不是用trainingset和testingset来看overfitting和underfitting吗？

实际上overfitting和underfitting都是用验证数据集和训练数据集来看的，还是那句话，测试数据集只能用一次，只

能得到我们真实的结果，我们就是不停的训练模型，在验证数据集和训练数据集上跑模型来优化，尽量使得最后的测试结果尽如人意，但是结果也是不太一定的。

问题7：老师，如果是时间序列上的数据，训练集和验证集可能会有自相关，这时候应该怎么处理呢？

时间序列的数据集

我们将时间上靠后的取一部分作为验证数据集，其前的作为训练数据集

问题9：老师，深度学习一般训练集合比较大，所以K折交叉验证在深度学习中是不是没什么应用？训练成本太高了吧？

数据集较大时确实使

用不多，在传统的机器学习中采用较多

问题17：老师，如何有效设计超参数，是不是只能搜索？最好用的搜索是贝叶斯方法还是网格、随机？老师有推荐吗？

经验

问题19：k折交叉验证的目的是确定超参数吗？然后还要用这个超参数再训练一遍全数数据吗？

是的；也可以从k

折中选取一个精度最好的参数，缺陷是少使用了一部分数据集；也可以将测试数据在k个参数模型上都使用一次，然后取预测结果的均值，代价变成了k倍，但是模型稳定性更好一点。

问题34：如果训练是不平衡的，是否要先考虑测试集是否也是不平衡的

，再去决定是否使用一个平衡的验证集？

根据实际情况选择

问题35：我在训练的时候，x轴是迭代次数，在验证集上loss也会发生这种先下降后上升的趋势，是不是错的？是因为发生了过拟合？

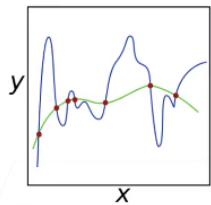
是的

权重衰减

使用均方范数作为硬性限制

- 通过限制参数值的选择范围来控制模型容量

$$\min \ell(\mathbf{w}, b) \text{ subject to } \|\mathbf{w}\|^2 \leq \theta$$



- 通常不限制偏移 b （限不限制都差不多）
- 小的 θ 意味着更强的正则项



限制w的L2范数，使其小于一个值

使用均方范数作为柔性限制

为什么是L2

- 对每个 θ ，都可以找到 λ 使得之前的目标函数等价于下面

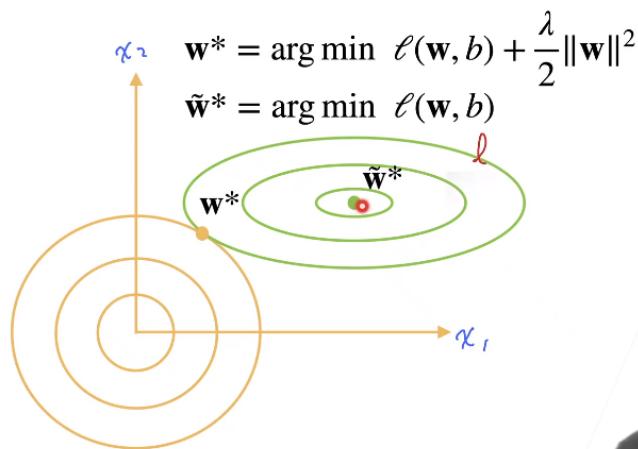
$$\min \ell(\mathbf{w}, b) + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

- 可以通过拉格朗日乘子来证明
- 超参数 λ 控制了正则项的重要程度
 - $\lambda = 0$: 无作用
 - $\lambda \rightarrow \infty, \mathbf{w}^* \rightarrow \mathbf{0}$



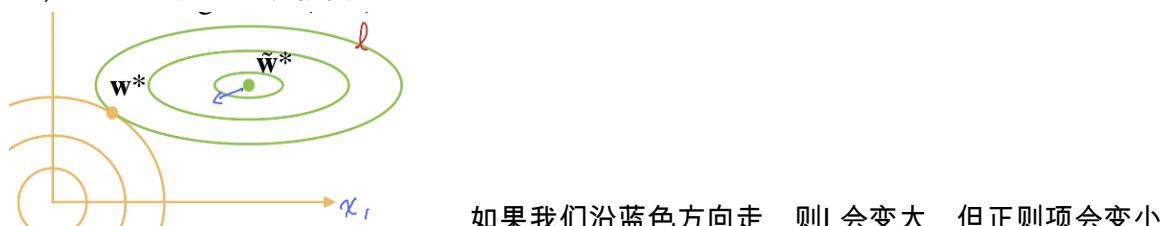
通过后面一项来使损失函数增大，从而通过 λ 控制权重的大小

演示对最优解的影响

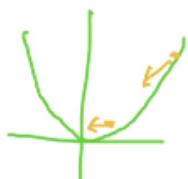


图中绿色的为损失函数的等高线，则它的最中心就是只优化损失函数时的最低点。

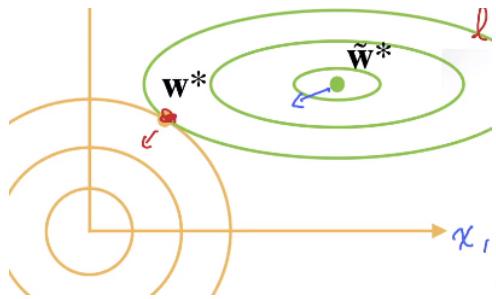
橙色为后面添加的正则项，假设正则项只有两个变量，则它的图像就如橙色。但是 $w \sim$ 最优值的 w_1, w_2 对于正则项来说就很大了。



2



由于L2的特点，在最低点梯度较小，较远点的梯度较大，则沿着蓝色方向移动，L变动较小，但是正则项的变化较大。



在这个点时，沿着这个方向前进，我们假设 w 的减小不足以弥补 L 的增加。而往红色箭头的反向前进则会正则的增加大于 L 的减少。所以在 w^* 形成一个平衡点。总的来说，正则项的引入使得总的损失函数往原点方向前进了，相对于原来的最优点的值会变得小一些，如果模型所有值都变小，则模型复杂度就变低了。

参数更新法则

- 计算梯度

$$\frac{\partial}{\partial \mathbf{w}} \left(\ell(\mathbf{w}, b) + \frac{\lambda}{2} \|\mathbf{w}\|^2 \right) = \frac{\partial \ell(\mathbf{w}, b)}{\partial \mathbf{w}} + \lambda \mathbf{w}$$

- 时间 t 更新参数

$$\mathbf{w}_{t+1} = (1 - \eta \lambda) \mathbf{w}_t - \eta \frac{\partial \ell(\mathbf{w}_t, b_t)}{\partial \mathbf{w}_t}$$

- 通常 $\eta \lambda < 1$ ，在深度学习中通常叫做权重衰退

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \frac{\partial \ell}{\partial \mathbf{w}_{t-1}}$$

(原梯度表达式)

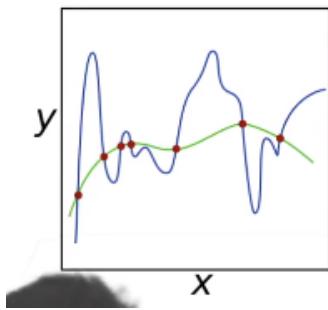
总结

- 权重衰退通过 L2 正则项使得模型参数不会过大，从而控制模型复杂度
- 正则项权重是控制模型复杂度的超参数

问题2：为什么参数不过大复杂度就低呢？

取参数

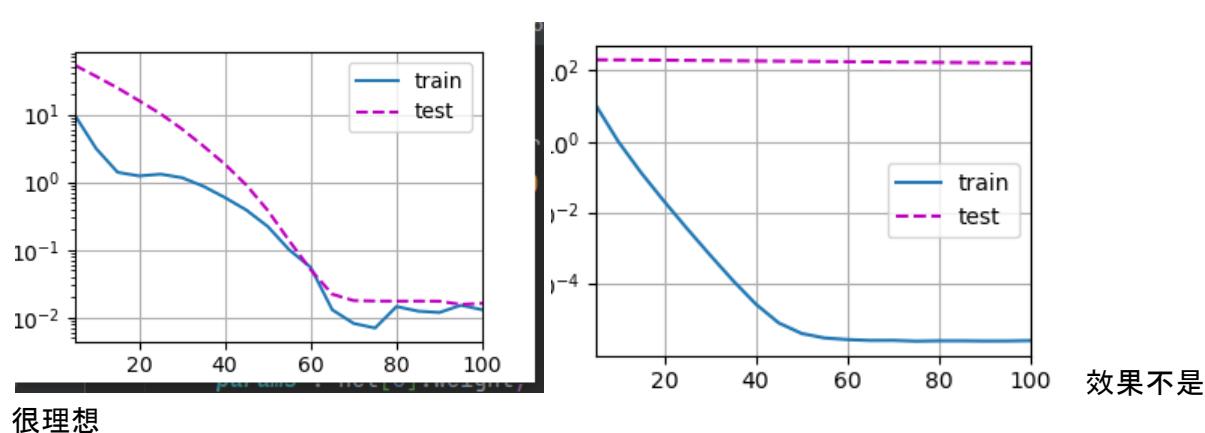
限制模型只在一个很小的范围内



比如我们要拟合这个输入，如果模型的参数可取值的范围较大，则拟合的曲线可能如上图中的蓝色一样非常不平滑，而限制参数的取值范围可以使曲线平滑许多。

定义 L_2 范数惩罚

```
In [4]: def l1_penalty(w):
    return torch.sum(torch.abs(w))
```

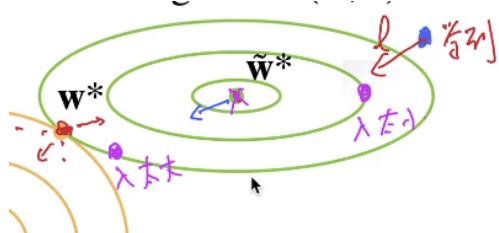


问题4：实践中权重衰减的值一般设置多少为好呢？之前在跑代码的时候总感觉权重衰减的效果并不那么好
一般取0.001和0.0001等等。权重衰退的效果有限，还有其他的方法。

$$-\frac{\lambda}{2} \|\mathbf{w}\|_2^2$$

这里用的是 L_2 范数的平方来做正则项

问题6：问老师，为什么要把 w 往小的拉？如果最优解的 W 就是比较大的数，那权重衰减是不是会有反作用？



一般我们都学习不到最优点的，都需要lambda来调，

因为实际上数据都有噪音。

问题9：实际应用中，lambda作为超参数是一次次在训练后调整优化了吗？
调整到什么时候达到满意的效果有什么方法论或最佳实践吗？

看验

证和训练数据集的gap过大，我们就可以增加lambda，但是lambda缺失作用有限。

问题10：老师刚才在解释数据噪音的时候，好像是说如果噪音越大，W就比
较大，这个是经验所得还是可以证明

是的

丢弃法

丢弃法



动机

- 一个好的模型需要对输入数据的扰动鲁棒
- 使用有噪音的数据等价于 Tikhonov 正则
- 丢弃法：在层之间加入噪音

数据内加上噪音则等于加入正则项

无偏差的加入噪音

- 对 \mathbf{x} 加入噪音得到 \mathbf{x}' , 我们希望 $E[\mathbf{x}'] = \mathbf{x}$

$$E[x'_i] = p \cdot 0 + (1-p) \frac{x_i}{1-p} = x_i$$

$$E[\mathbf{x}'] = \mathbf{x}$$

- 丢弃法对每个元素进行如下扰动

$$x'_i = \begin{cases} 0 & \text{with probability } p \\ \frac{x_i}{1-p} & \text{otherwise} \end{cases}$$



使用drop out后期望没有

发生变化

问题11: dropout随机置0对求梯度和反向传播的影响是什么?

被

dropout丢弃的不会更新梯度

问题18: 老师, 请问可以再解释一下为什么“推理中的dropout是直接返回输入”吗?

dropout只在

更新权重的时候使用, 我们将其视为正则项, 其作用就是限制权重大小, 使模型复杂度降低

问题19: 请问沐神, dropout函数返回值的表达式`return X*mask/(1.0-p)`没被丢弃的输入部分的值会因为表达式分母`(1-p)`的存在而改变, 而训练数据的标签还是原来的值, 这怎么理解?

dropout只会

改变使用了dropout了的隐藏层的输出

问题20: 请问沐神, 训练时使用dropout, 推理时不用。那会不会导致推理时输出结果翻倍了? 比如`dropout=0.5`, 推理时输出结果是训练时2个子神经网络的叠加而翻倍?

由于`1-p`的存在, 方差

没有变化

问题26: dropout和权重衰减都属于正则, 为何dropout效果更好现在更常用呢?

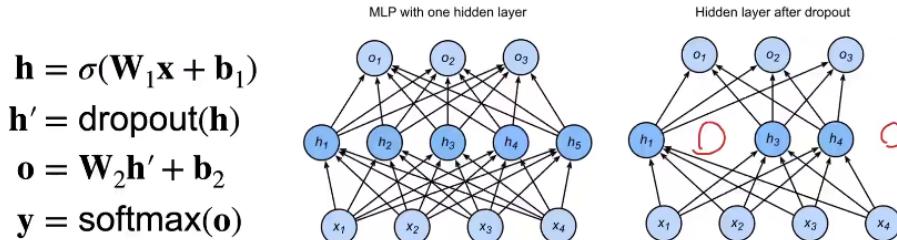
dropout一般

用于全连接层上

使用丢弃法

清华大学公开课

- 通常将丢弃法作用在隐藏全连接层的输出上



推理中的丢弃法

- 正则项只在训练中使用：他们影响模型参数的更新
- 在推理过程中，丢弃法直接返回输入

$$\mathbf{h} = \text{dropout}(\mathbf{h})$$

- 这样也能保证确定性的输出

总结

- 丢弃法将一些输出项随机置0来控制模型复杂度
- 常作用在多层感知机的隐藏层输出上
- 丢弃概率是控制模型复杂度的超参数

数值稳定性

- 考虑如下有 d 层的神经网络

$$\mathbf{h}^t = f_t(\mathbf{h}^{t-1}) \quad \text{and} \quad y = \ell \circ f_d \circ \dots \circ f_1(\mathbf{x})$$

- 计算损失 ℓ 关于参数 \mathbf{W}_t 的梯度

$$\frac{\partial \ell}{\partial \mathbf{W}^t} = \frac{\partial \ell}{\partial \mathbf{h}^d} \underbrace{\frac{\partial \mathbf{h}^d}{\partial \mathbf{h}^{d-1}} \dots \frac{\partial \mathbf{h}^{t+1}}{\partial \mathbf{h}^t}}_{d-t \text{ 次矩阵乘法}} \frac{\partial \mathbf{h}^t}{\partial \mathbf{W}^t}$$

动手学深度学习 v2 · <https://courses.d2l.ai/zh/v2>

假设有 d 层的神

经网络，记层数为 t 层。

由于向量对向量求导结果是一个矩阵，则损失对 \mathbf{w}^t 的导数就是 $d-t$ 次矩阵乘法。

数值稳定性的常见两个问题

梯度爆炸



$$1.5^{100} \approx 4 \times 10^{17}$$

梯度消失



$$0.8^{100} \approx 2 \times 10^{-10}$$

例子：MLP

- 加入如下 MLP (为了简单省略了偏移)

$$f_t(\mathbf{h}^{t-1}) = \sigma(\mathbf{W}^t \mathbf{h}^{t-1}) \quad \sigma \text{ 是激活函数}$$

$$\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} = \text{diag} (\sigma'(\mathbf{W}^t \mathbf{h}^{t-1})) (\mathbf{W}^t)^T \quad \sigma' \text{ 是 } \sigma \text{ 的导数函数}$$

$$\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} \text{diag} (\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})) (\mathbf{W}^i)^T$$



梯度爆炸

- 使用 ReLU 作为激活函数

$$\sigma(x) = \max(0, x) \quad \text{and} \quad \sigma'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

• $\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} \text{diag}(\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})) (\mathbf{W}^i)^T$ 的一些元素会来自于 $\prod_{i=t}^{d-1} (\mathbf{W}^i)^T$

- 如果 $d-t$ 很大，值将会很大

使用ReLU作为激活函数，则等式右边的(\mathbf{W}^i) T 的列有可能乘以0或者1，则(\mathbf{W}^i) T 中的某些列会完全保留，某些列会完全丢弃变为0.如果d-t很大，并且保留的列数较多，则矩阵的值会很大，造成梯度爆炸。

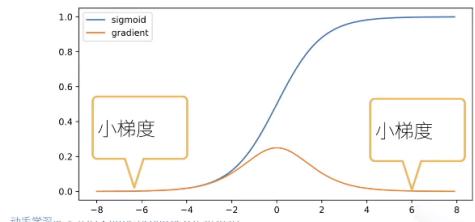
梯度爆炸的问题

- 值超出值域 (infinity)
 - 对于 16位浮点数尤为严重 (数值区间 6e-5 - 6e4)
- 对学习率敏感
 - 如果学习率太大 -> 大参数值 -> 更大的梯度
 - 如果学习率太小 -> 训练无进展
 - 我们可能需要在训练过程不断调整学习率

梯度消失

- 使用 sigmoid 作为激活函数

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



当输入大于6时，激活函数的到数据就会接近于0

梯度消失

- 使用 sigmoid 作为激活函数

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

- $\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} \text{diag}(\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})) (\mathbf{W}^i)^T$ 的元素值是 d-t 个大小数值的乘积

$$0.8^{100} \approx 2 \times 10^{-10}$$



梯度消失的问题

- 梯度值变成 0
 - 对 16 位浮点数尤为严重
 - 训练没有进展
 - 不管如何选择学习率
 - 对于底部层尤为严重
 - 仅仅顶部层训练的较好
 - 无法让神经网络更深
- 总结**
- 当数值过大或者过小时会导致数值问题
 - 常发生在深度模型中，因为其会对 n 个数累乘

让训练更加稳定

让训练更加稳定

- 目标：让梯度值在合理的范围内
 - 例如 [1e-6, 1e3]
- 将乘法变加法
 - ResNet, LSTM
- 归一化
 - 梯度归一化，梯度裁剪
- 合理的权重初始和激活函数

让每层的方差是一个常数

- 将每层的输出和梯度都看做随机变量
- 让它们的均值和方差都保持一致

正向	反向
$\mathbb{E}[h_i^t] = 0$	$\mathbb{E} \left[\frac{\partial \ell}{\partial h_i^t} \right] = 0$
$\text{Var}[h_i^t] = a$	$\text{Var} \left[\frac{\partial \ell}{\partial h_i^t} \right] = b \quad \forall i, t$

a 和 b 都是常数

我们都使其均值为同一个数，并且正向的输出的方差保持一致为a，反向的梯度的方差保持为b，这里a和b都是常数。这是我们期望深度神经网络满足的要求。

权重初始化

- 在合理值区间里随机初始参数
- 训练开始的时候更容易有数值不稳定
 - 远离最优解的地方损失函数表面可能很复杂
 - 最优解附近表面会比较平
- 使用 $\mathcal{N}(0, 0.01)$ 来初始可能对小网络没问题，但不能保证深度神经网络



如果随机初始参数在一个离最

优解较远的区域内，并且这个区域较陡，导致梯度较大，则有可能会使得梯度出现爆炸。

例子：MLP

- 假设

- $w_{i,j}^t$ 是 i.i.d，那么 $\mathbb{E}[w_{i,j}^t] = 0$, $\text{Var}[w_{i,j}^t] = \gamma_t$
- h_i^{t-1} 独立于 $w_{i,j}^t$

- 假设没有激活函数 $\mathbf{h}^t = \mathbf{W}^t \mathbf{h}^{t-1}$, 这里 $\mathbf{W}^t \in \mathbb{R}^{n_t \times n_{t-1}}$

$$\mathbb{E}[h_i^t] = \mathbb{E} \left[\sum_j w_{i,j}^t h_j^{t-1} \right] = \sum_j \mathbb{E}[w_{i,j}^t] \mathbb{E}[h_j^{t-1}] = 0$$



假设 $w_{i,j}$ 是独立同分布，则其均值

=0, 方差为 γ_t (t 为层数)。当前层的权重和当前层的输入其实是一个独立的事件。

假设在 t 层的输入时没有激活函数，则 t 层的输出就等于 $\mathbf{W}^t \mathbf{h}^{t-1}$ ，则其均值就可以分解成以下的式子，由于我们前面的假设，则最后得到的均值为0。

正向方差

$$\begin{aligned}
 \text{Var}[h_i^t] &= \mathbb{E}[(h_i^t)^2] - \mathbb{E}[h_i^t]^2 = \mathbb{E}\left[\left(\sum_j w_{i,j}^t h_j^{t-1}\right)^2\right] \\
 &= \mathbb{E}\left[\sum_j \left(w_{i,j}^t\right)^2 \left(h_j^{t-1}\right)^2 + \sum_{j \neq k} w_{i,j}^t w_{i,k}^t h_j^{t-1} h_k^{t-1}\right] \\
 &= \sum_j \mathbb{E}\left[\left(w_{i,j}^t\right)^2\right] \mathbb{E}\left[\left(h_j^{t-1}\right)^2\right] \quad \text{if } n_{t-1}\gamma_t = 1 \\
 &= \sum_j \text{Var}[w_{i,j}^t] \text{Var}[h_j^{t-1}] = n_{t-1}\gamma_t \text{Var}[h_j^{t-1}]
 \end{aligned}$$

来看正向方差的推导过程。

首先方差可以展开为平方项的均值减去均值的平方项，由于我们假设均值为0，则去掉第二项。然后将其表达式展开，再将E带入求和项，由于均值为0，则这两个均值项就等于其方差项，由于我们一开始假设wij的方差为gama t，则其和就是(n t-1)*gama t，则想要等式两端相等就需要(n t-1)* gama t=1.

反向均值和方差

- 跟正向情况类似

$$\frac{\partial \ell}{\partial \mathbf{h}^{t-1}} = \frac{\partial \ell}{\partial \mathbf{h}^t} \mathbf{W}^t \quad \Rightarrow \quad \left(\frac{\partial \ell}{\partial \mathbf{h}^{t-1}}\right)^T = (\mathbf{W}^t)^T \left(\frac{\partial \ell}{\partial \mathbf{h}^t}\right)^T$$

$$\mathbb{E}\left[\frac{\partial \ell}{\partial h_i^{t-1}}\right] = 0$$

$$\text{Var}\left[\frac{\partial \ell}{\partial h_i^{t-1}}\right] = n_t \gamma_t \text{Var}\left[\frac{\partial \ell}{\partial h_j^t}\right] \quad \Rightarrow \quad n_t \gamma_t = 1$$

则要满足我们前面对均值和方差的期望，必须满足(n t)*gama t=1和(n t-1)*gama t=1，很显然，这个条件是难以满足的。

Xavier 初始

- 难以需要满足 $\underline{n_{t-1}\gamma_t = 1}$ 和 $\underline{n_t\gamma_t = 1}$
- Xavier 使得 $\gamma_t(n_{t-1} + n_t)/2 = 1 \rightarrow \gamma_t = 2/(n_{t-1} + n_t)$
 - 正态分布 $\mathcal{N}(0, \sqrt{2/(n_{t-1} + n_t)})$
 - 均匀分布 $\mathcal{U}(-\sqrt{6/(n_{t-1} + n_t)}, \sqrt{6/(n_{t-1} + n_t)})$
 - 分布 $\mathcal{U}[-a, a]$ 和方差是 $a^2/3$
- 适配权重形状变换，特别是 n_t



假设线性的激活函数

- 假设 $\sigma(x) = \alpha x + \beta$

$$\mathbf{h}' = \mathbf{W}^t \mathbf{h}^{t-1} \quad \text{and} \quad \mathbf{h}^t = \sigma(\mathbf{h}')$$

$$\mathbb{E}[h_i^t] = \mathbb{E}[\alpha h_i' + \beta] = \beta \quad \Rightarrow \quad \beta = 0$$

$$\begin{aligned}\text{Var}[h_i^t] &= \mathbb{E}[(h_i^t)^2] - \mathbb{E}[h_i^t]^2 \\ &= \mathbb{E}[(\alpha h_i' + \beta)^2] - \beta^2 \\ &= \mathbb{E}[\alpha^2(h_i')^2 + 2\alpha\beta h_i' + \beta^2] - \beta^2 \\ &= \alpha^2 \text{Var}[h_i']\end{aligned}$$

也就是说，如果我们想要经过激活函数后仍然使得方差和均值不变，就需要满足 $\beta = 0, \alpha = 1$ ，也就是说，激活函数等于其自身。

反向

- 假设 $\sigma(x) = \alpha x + \beta$

$$\frac{\partial \ell}{\partial \mathbf{h}'} = \frac{\partial \ell}{\partial \mathbf{h}'} (\mathbf{W}^t)^T \quad \text{and} \quad \frac{\partial \ell}{\partial \mathbf{h}^{t-1}} = \alpha \frac{\partial \ell}{\partial \mathbf{h}'}$$

$$\mathbb{E} \left[\frac{\partial \ell}{\partial h_i^{t-1}} \right] = 0 \quad \Rightarrow \quad \beta = 0$$

$$\text{Var} \left[\frac{\partial \ell}{\partial h_i^{t-1}} \right] = \alpha^2 \text{Var} \left[\frac{\partial \ell}{\partial h_j'} \right] \quad \Rightarrow \quad \alpha = 1$$

检查常用激活函数

- 使用泰勒展开

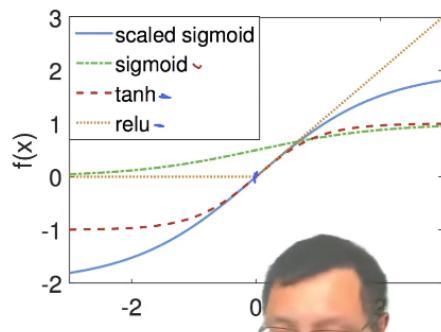
$$\text{sigmoid}(x) = \frac{1}{2} + \frac{x}{4} - \frac{x^3}{48} + O(x^5)$$

$$\tanh(x) = 0 + x - \frac{x^3}{3} + O(x^5)$$

$$\text{relu}(x) = 0 + x \quad \text{for } x \geq 0$$

- 调整 sigmoid:

$$4 \times \text{sigmoid}(x) - 2$$



tanh和relu函数在

原点基本能够满足 $f(x)=x$ 的要求。虽然sigmoid不满足我们的要求，但是对其进行一些调整之后，就能够基本满足我们的要求了。

总结

- 合理的权重初始值和激活函数的选取可以提升数值稳定性

问题3：老师，如果训练一开始，在验证集上准确率在提升，但是训练两个 epoch之后，突然验证集上准确率就变成50%左右了之后稳定在50%，这是为什么呢？

一般都是数值稳定性

出现了问题。

问题12：输出或者参数符合正态分布有利于学习，请教下是否有理论还是经验所得？

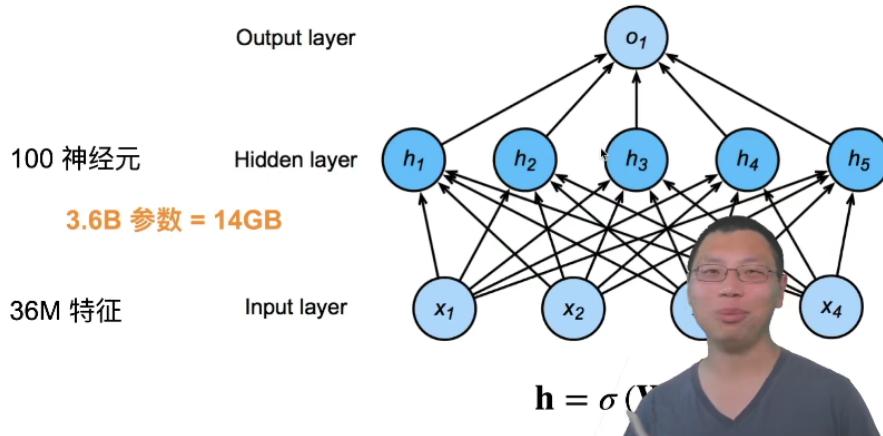
我们想要的是输出咋

一个合理的区间内，如果符合某种分布，则有利于计算方差和均值，并不一定非要是正态分布。

卷积神经网络

基础

卷积操作子



两个原则

- 平移不变性 (Translation Invariance)
- 局部性 (Locality)



寻找对象有两个原

则：(1) 分类器在图片中的任何一个地方都适用
(2) 不需要看全局的信息，局部的就可以了

重新考察全连接层

- 将输入和输出变形为矩阵（宽度，高度）
- 将权重变形为4-D张量 (h, w) 到 (h', w')

$$h_{i,j} = \sum_{k,l} w_{i,j,k,l} x_{k,l} = \sum_{a,b} v_{i,j,a,b} x_{i+a,j+b}$$

- v 是 w 的重新索引 $v_{i,j,a,b} = w_{i,j,i+a,j+b}$



我们在前面的操作

是将图片展开成一个向量，而在这里，我们需要图片的空间信息，则将输入输出都变为矩阵。

然后我们设定 v 来做 w 的重新索引。这样就将全连接从一维变成了二维，并且对下标做了更新。这里的 i 和 j 为 中心坐标， a 和 b 为与中心的两个轴上的偏移量。

原则 #1 - 平移不变性

- x 的平移导致 h 的平移 $h_{i,j} = \sum_{a,b} v_{i,j,a,b} x_{i+a,j+b}$
- v 不应该依赖于 (i, j)
- 解决方案： $v_{i,j,a,b} = v_{a,b}$

$$h_{i,j} = \sum_{a,b} v_{a,b} x_{i+a,j+b}$$



这就是 2 维卷积 交叉相关

可以看到，这里的 x 的 i 或 j 发生变化时，则相应的 v 也会发生变化，而我们想要 v 不会因为平移而发生变化，则将 v 的 i 和 j 设为常数，不再是与 i 和 j 相关的变量。

原则 #2 - 局部性

$$h_{i,j} = \sum_{a,b} v_{a,b} x_{i+a,j+b}$$

- 当评估 $h_{i,j}$ 时，我们不应该用远离 $x_{i,j}$ 的参数
- 解决方案：当 $|a|, |b| > \Delta$ 时，使得 $v_{a,b} = 0$

$$h_{i,j} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} v_{a,b} x_{i+a,j+b}$$



评估 $h_{i,j}$ 时，我们想要局部的参数而非全局的参数，则我们设定阈值，一旦偏移量超过了这个阈值，我们就将 $v_{a,b}$ 设为 0，这样就避免了其他不在范围内的对象的影响。

总结

- 对全连接层使用平移不变性和局部性得到卷积层

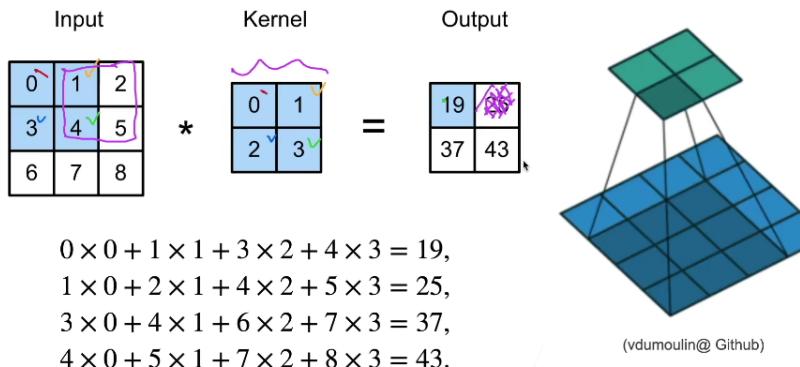
$$h_{i,j} = \sum_{a,b} v_{i,j,a,b} x_{i+a,j+b}$$

↓

$$h_{i,j} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} v_{a,b} x_{i+a,j+b}$$


卷积层

二维交叉相关



二维卷积层

交叉相关 vs 卷积

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 19 & 25 \\ 37 & 43 \end{bmatrix}$$

- 输入 $\mathbf{X} : n_h \times n_w$
- 核 $\mathbf{W} : k_h \times k_w$
- 偏差 $b \in \mathbb{R}$
- 输出 $\mathbf{Y} : (n_h - k_h + 1) \times (n_w - k_w + 1)$

$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} + b$$

- \mathbf{W} 和 b 是可学习的参数

二维交叉相关

$$y_{i,j} = \sum_{a=1}^h \sum_{b=1}^w w_{a,b} x_{i+a,j+b}$$

二维卷积

$$y_{i,j} = \sum_{a=1}^h \sum_{b=1}^w w_{-a,-b} x_{i+a,j+b}$$

- 由于对称性，在实际使用中没有区别

一维和三维交叉相关

一维

$$y_i = \sum_{a=1}^h w_a x_{i+a}$$

- 文本
- 语言
- 时序序列

三维

$$y_{i,j,k} = \sum_{a=1}^h \sum_{b=1}^w \sum_{c=1}^d w_{a,b,c} x_{i+a,j+b,k+c}$$

- 视频
- 医学图像
- 气象地图



总结



- 卷积层将输入和核矩阵进行交叉相关，加上偏移后得到输出
- 核矩阵和偏移是可学习的参数
- 核矩阵的大小是超参数

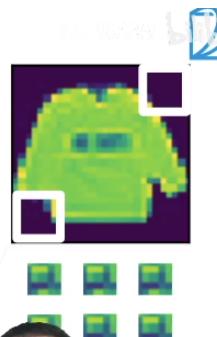
我们设定卷积核为 $n \times n$ ，则输出的卷积核矩阵就是 $n \times n$ 的，不会因为输入的变大或缩小就变化，因此卷积可以解决前面提到的线性隐藏层带来的权重变多的问题。

问题21：为什么不应该看那么远？感受野不是越大越好吗？ 类似于MLP的隐藏层，我们认为深一点的但是神经元小一点的神经网络比较好，因此我们将kernel_size设小一点，但是将网络设深一点，觉得这样会对训练有好处。

填充和步幅

填充

- 给定 (32×32) 输入图像
- 应用 5×5 大小的 卷积核
 - 第1层得到输出大小 28×28
 - 第7层得到输出大小 4×4
- 更大的卷积核可以更快地减小输出大小
 - 形状从 $n_h \times n_w$ 减少到 $(n_h - k_h + 1) \times (n_w - k_w + 1)$



填充



在输入周围添加额外的行 / 列

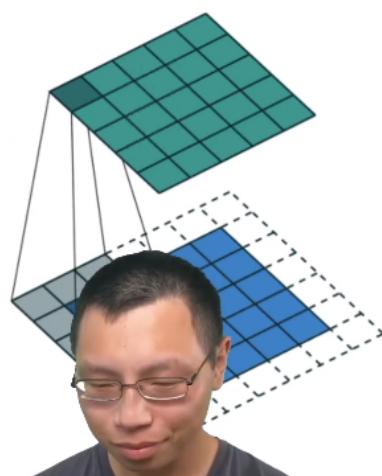
Input				
0	0	0	0	0
0	0	1	2	0
0	3	4	5	0
0	6	7	8	0
0	0	0	0	0

*

Kernel	
0	1
2	3

=

Output			
0	3	8	4
9	19	25	10
21	37	43	16
6	7	8	0



$$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$$

填充

- 填充 p_h 行和 p_w 列，输出形状为

$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$

- 通常取 $p_h = k_h - 1, p_w = k_w - 1$
 - 当 k_h 为奇数：在上下两侧填充 $p_h/2$
 - 当 k_h 为偶数：在上侧填充 $\lceil p_h/2 \rceil$ ，在下侧填充 $\lfloor p_h/2 \rfloor$

如果取左边的值，则输入和

输出形状不会发生变化。

步幅

- 填充减小的输出大小与层数线性相关
 - 给定输入大小 224×224 ，在使用 5×5 卷积核的情况下，需要 44 层将输出降低到 4×4
 - 需要大量计算才能得到较小输出



步幅

- 步幅是指行/列的滑动步长
 - 例：高度3 宽度2 的步幅

Input				
0	0	0	0	0
0	0	1	2	0
0	3	4	5	0
0	6	7	8	0
0	0	0	0	0

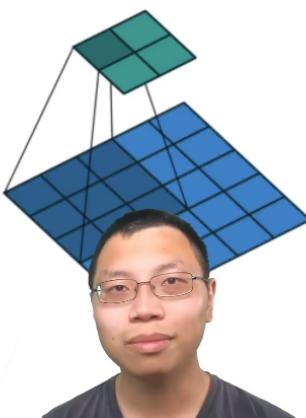
*

Kernel	
0	1
2	3

=

Output	
0	8
2	3
0	8

$$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$$
$$0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$$



步幅

- 给定高度 s_h 和宽度 s_w 的步幅，输出形状是

$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$$

- 如果 $p_h = k_h - 1, p_w = k_w - 1$

$$\lfloor (n_h + s_h - 1)/s_h \rfloor \times \lfloor (n_w + s_w - 1)/s_w \rfloor$$

- 如果输入高度和宽度可以被步幅整除

$$(n_h/s_h) \times (n_w/s_w)$$

在这里，原本的+1其实就是+步幅
 s_h ，因此，这里的变为了+s h，由于可能有走到最后输入不够卷积核步幅前进的情况，做一个
向下取整操作/s h。一般我们设定步幅为2，可以被整除。

总结



- 填充和步幅是卷积层的超参数
- 填充在输入周围添加额外的行 / 列，来控制输出形状的减
少量
- 步幅是每次滑动核窗口时的行/列的步长，可以成倍的减少
输出形状



问题2：这几个超参数得影响重要程度排序是怎样得，核大小，填充，步幅

一般取padding=kh-1，一般取stride=1(想要减小输出则一般设置步幅为2，然后计算输入大小
需要进行i次缩小才能得到结果，就可以将i个步幅为2的层均匀的插入到网络中)。其中核大小
是最重要的，一般选择3*3.

问题17：底层用大kernel，上层用小kernel，相比都用同样宽度的kernel，哪
种方案更适合目标有多尺度的情况？

简单的比较

通用，即尽量选择相同宽度kernel

多个输入和输出通道

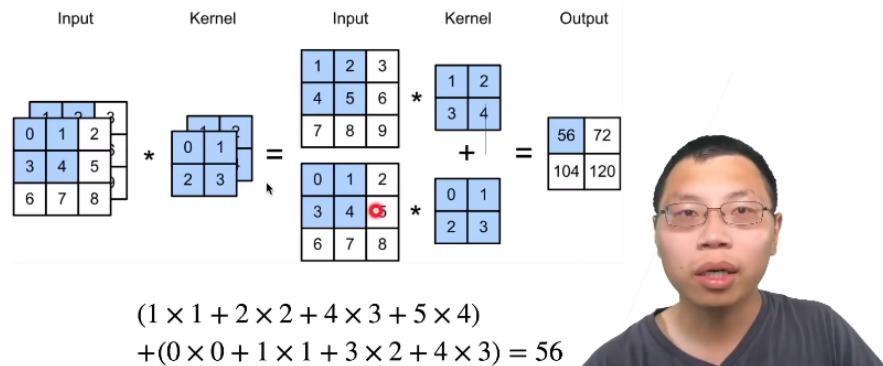
多个输入通道

- 彩色图像可能有 RGB 三个通道
- 转换为灰度会丢失信息



多个输入通道

- 每个通道都有一个卷积核，结果是所有通道卷积结果的和



多个输入通道

- 输入 $\mathbf{X} : c_i \times n_h \times n_w$
- 核 $\mathbf{W} : c_i \times k_h \times k_w$
- 输出 $\mathbf{Y} : m_h \times m_w$

$$\mathbf{Y} = \sum_{i=0}^{c_i} \mathbf{X}_{i,:,:} \star \mathbf{W}_{i,:,:}$$

其中 c_i 为通道数

多个输出通道

- 无论有多少输入通道，到目前为止我们只用到单输出通道
- 我们可以有多个三维卷积核，每个核生成一个输出通道
- 输入 $\mathbf{X} : c_i \times n_h \times n_w$
- 核 $\mathbf{W} : c_o \times c_i \times k_h \times k_w$
- 输出 $\mathbf{Y} : c_o \times m_h \times m_w$

$$\mathbf{Y}_{i,:,:} = \mathbf{X} \star \mathbf{W}_{i,:,:} \quad \text{for } i = 1, \dots,$$



多个输入和输出通道

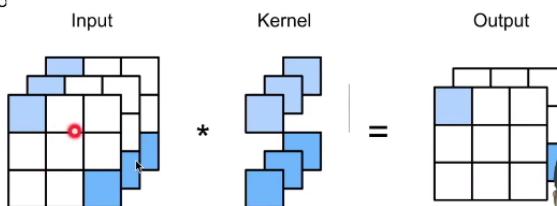
- 每个输出通道可以识别特定模式



- 输入通道核识别并组合输入中的模式

1 x 1 卷积层

$k_h = k_w = 1$ 是一个受欢迎的选择。它不识别空间模式，只是融合通道。



相当于输入形状为 $n_h n_w \times c_i$, 权重为 $c_o \times c_i$ 的全连接层



就是一个全连接层

二维卷积层

- 输入 $\mathbf{X} : c_i \times n_h \times n_w$
- 核 $\mathbf{W} : c_o \times c_i \times k_h \times k_w$
- 偏差 $\mathbf{B} : c_o \times c_i$
- 输出 $\mathbf{Y} : c_o \times m_h \times m_w$
- 计算复杂度 (浮点计算数 FLOP) $O(c_i c_o k_h k_w m_h m_w)$

$$\begin{array}{l} c_i = c_o = 100 \\ k_h = h_w = 5 \\ m_h = m_w = 64 \end{array} \quad \Rightarrow \quad 1\text{GFLOP}$$

- 10 层，1M 样本，10 PFlops
(CPU: 0.15 TF = 18h, GPU: 12 TF = 14min)

总结

- 输出通道数是卷积层的超参数
- 每个输入通道有独立的二维卷积核，所有通道结果相加得到一个输出通道结果
- 每个输出通道有独立的三维卷积核

问题21：网络越深，Padding 0 越多，这里是否会影响性能？

型精度

不会影响模

问题22：每个通道的卷积核都不一样吗？同一层不同通道的卷积核大小必须一样吗？

道的卷积核大小是一样的，因为不用写成两个操作，方便计算，当然也可以是不一样的

一般不同通

问题25：老师，如果是一个rgb图像，加上深度图，相当于输入是四个通道，做卷积是和rgb三通道同样做法吗？

深度变为三维卷积

不是，加入

问题31：多通道时，每个Kernel学习到了不同的参数，这样理解没错吧。那么卷积层中参数是如何共享的。

不共享参数

池化层

- 积对位置敏感
 - 检测垂直边缘

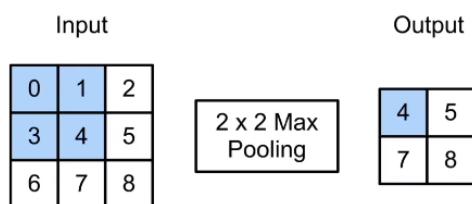
$$\begin{array}{c} \text{X} \\ \times \end{array} \quad \begin{bmatrix} 1. & 1. & 0. & 0. & 0. \\ 1. & 1. & 0. & 0. & 0. \\ 1. & 1. & 0. & 0. & 0. \\ 1. & 1. & 0. & 0. & 0. \end{bmatrix} \quad \begin{array}{c} \text{Y} \\ \end{array} \quad \begin{bmatrix} 0. & 1. & 0. & 0. \\ 0. & 1. & 0. & 0. \\ 0. & 1. & 0. & 0. \\ 0. & 1. & 0. & 0. \end{bmatrix}$$

1像素移位 导致 0 输出

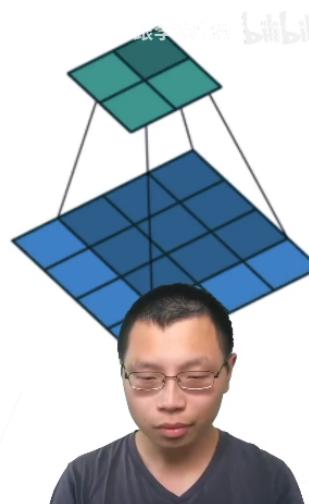
- 需要一定程度的平移不变性
 - 照明，物体位置，比例，外观等等因图像而异

二维最大池化

- 返回滑动窗口中的最大值



$$\max(0,1,3,4) = 4$$



二维最大池化

- 返回滑动窗口中的最大值

垂直边缘检测

卷积输出

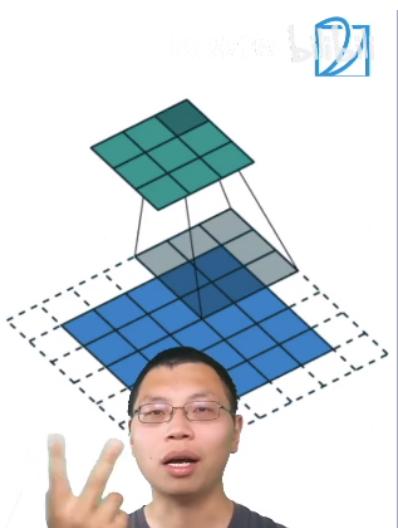
2 x 2 最大池化

$$\begin{array}{c} \text{[[1. 1. 0. 0. 0.} \\ \text{[1. 1. 0. 0. 0.} \\ \text{[1. 1. 0. 0. 0.} \\ \text{[1. 1. 0. 0. 0.} \end{array} \quad \begin{array}{c} \text{[[0. 1. 0. 0.} \\ \text{[0. 1. 0. 0.} \\ \text{[0. 1. 0. 0.} \\ \text{[0. 1. 0. 0.} \end{array} \quad \begin{array}{c} \text{[[1. 1. 1. 0.} \\ \text{[1. 1. 1. 0.} \\ \text{[1. 1. 1. 0.} \\ \text{[1. 1. 1. 0.} \end{array}$$

可容1像素移位

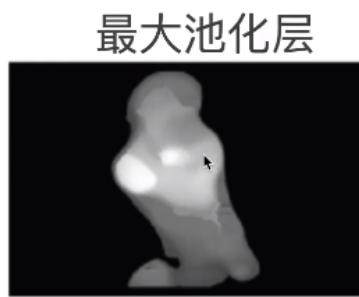
填充，步幅和多个通道

- 池化层与卷积层类似，都具有填充和步幅
- 没有可学习的参数
- 在每个输入通道应用池化层以获得相应的输出通道
- 输出通道数 = 输入通道数



平均池化层

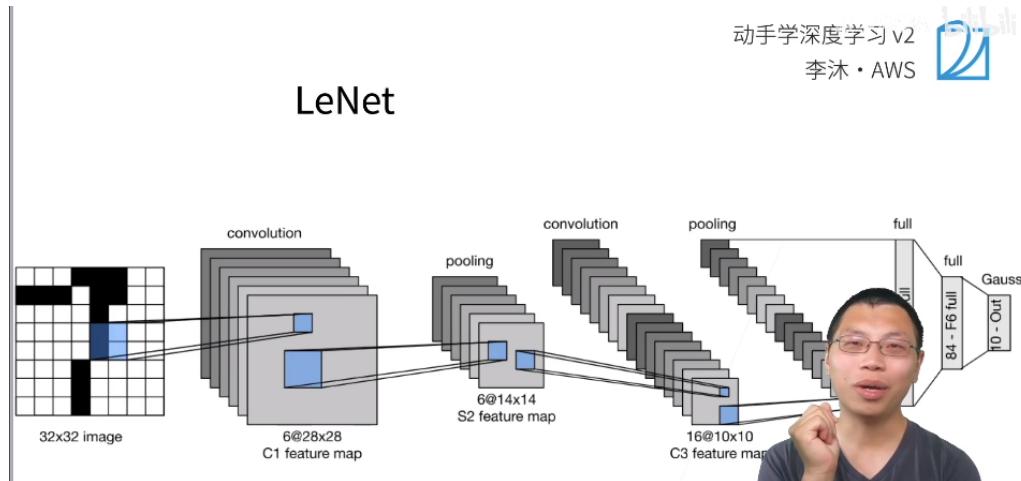
- 最大池化层：每个窗口中最强的模式信号
- 平均池化层：将最大池化层中的“最大”操作替换为“平均”



总结

- 池化层返回窗口中最大或平均值
- 缓解卷积层会位置的敏感性
- 同样有窗口大小、填充、和步幅作为超参数

LeNet

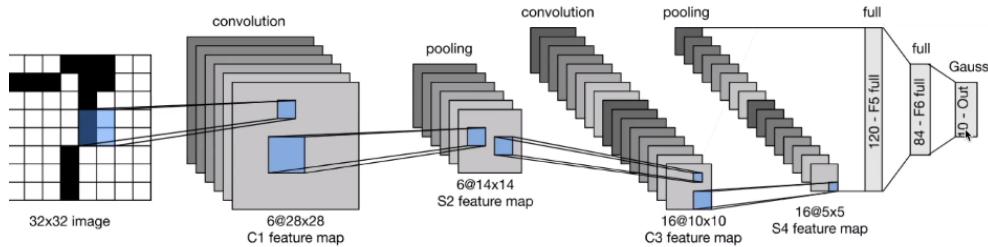


MNIST

- 50,000 个训练数据
- 10,000 个测试数据
- 图像大小 28×28
- 10类



动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>



总结

- LeNet是早期成功的神经网络
- 先使用卷积层来学习图片空间信息
- 然后使用全连接层来转换到类别空间

问题13: Conv2d 6->16, 16个通道是怎么取前面6个通道的信息的（每个都取，还是按一定组合取）？

多输入通道得到的结果是累加

，多输出通道是做多次卷积，但是不累加，每次运算的结果就是一个输出通道。

AlexNet

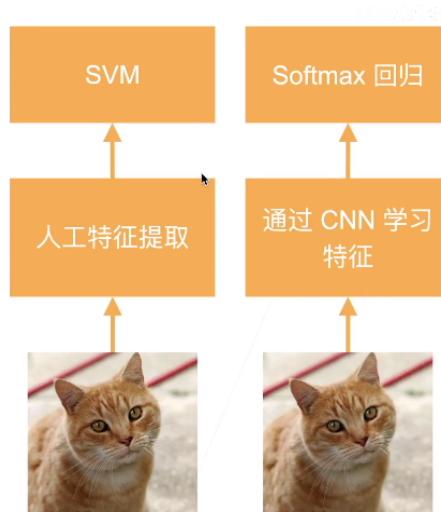
Hardware

	1970	1980	1990	2000	2010	2020	
Data (samples)	10^2 (e.g. iris)	10^3	$10x$ 10^4 OCR	$10x$ 神经网络	$100x$ 10^{7-8} web	$100x$ 10^{10} advertising	$1,000x$ 10^{12} social nets
RAM	1kB	100kB	10MB	100MB	1GB	100GB	
CPU	100kF (8080)	1MF (80186)	10MF (80486)	1GF (Intel Core)	100GF NVIDIA	>1PF (8xP3 Volta)	

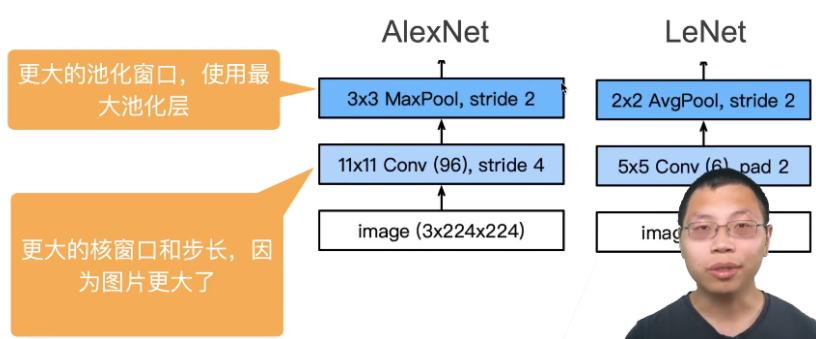
动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

AlexNet

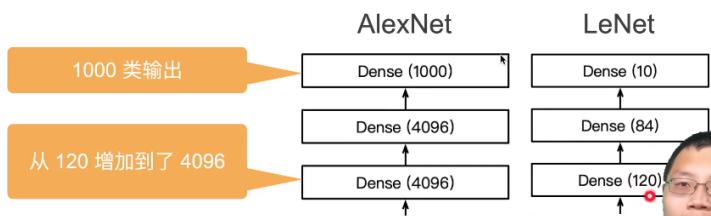
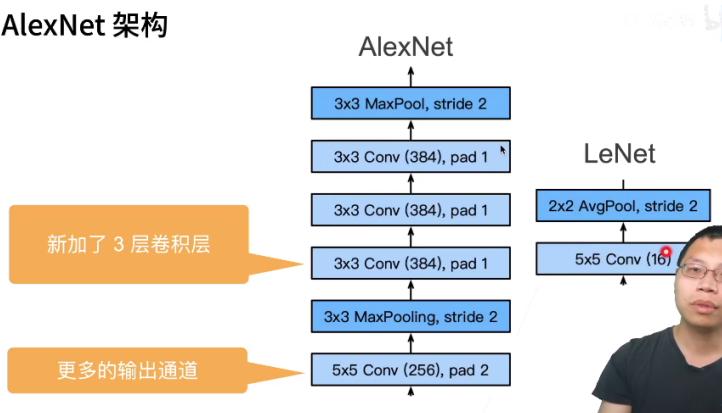
- AlexNet 赢了 2012 年 ImageNet 竞赛
- 更深更大的 LeNet
- 主要改进：
 - 丢弃法
 - ReLU
 - MaxPooling
- 计算机视觉方法论的改变



AlexNet 架构



AlexNet 架构



- 激活函数从 sigmoid 变到了 ReLu (减缓梯度消失)
- 隐藏全连接层后加入了丢弃层
- 数据增强



复杂度

	参数个数		FLOP	
	AlexNet	LeNet	AlexNet	LeNet
Conv1	35K	150	101M	1.2M
Conv2	614K	2.4K	415M	2.4M
Conv3-5	3M		445M	
Dense1	26M	0.48M	26M	0.48M
Dense2	16M	0.1M	16M	0.1M
Total	46M	0.6M	1G	4M
Increase	11x	1x	250x	1x

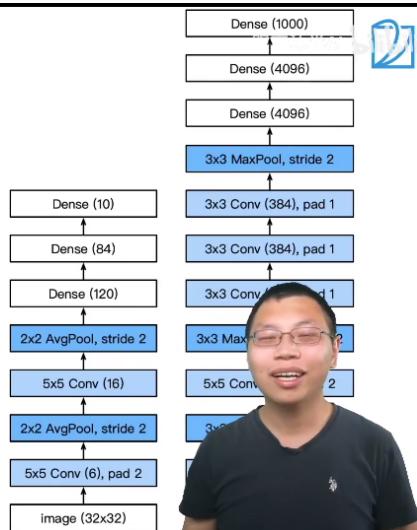
总结

- AlexNet 是更大更深的 LeNet, 10x 参数个数, 260x 计算复杂度
- 新进入了丢弃法, ReLU, 最大池化层, 和数据增强
- AlexNet 赢下了 2012 ImageNet 竞赛后, 标志着新一轮神经网络热潮的开始

VGG

VGG

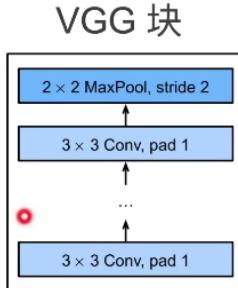
- AlexNet 比 LeNet 更深更大来得到更好的精度
 - 能不能更深和更大?
 - 选项
 - 更多的全连接层 (太贵)
 - 更多的卷积层
 - 将卷积层组合成块



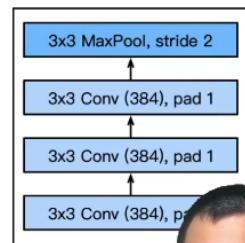
动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

VGG 块

- 深 vs. 宽?
 - 5×5 卷积
 - 3×3 卷积
 - 深但窄效果更好
 - VGG 块
 - 3×3 卷积 (填充 1)
(n 层, m 通道)
 - 2×2 最大池化层



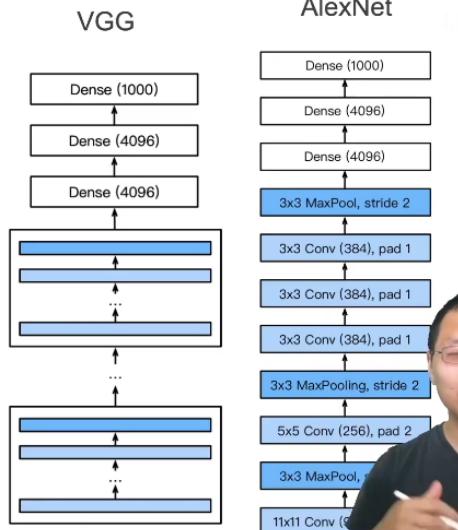
AlexNet的一部分



多的3x3卷积核比用更少的5x5卷积核要更高效

VGG 架构

- 多个VGG块后接全连接层
 - 不同次数的重复块得到不同的架构
VGG-16, VGG-19,



实验证明，使用更

进度

- LeNet (1995)
 - 2 卷积 + 池化层
 - 2 全连接层
- AlexNet
 - 更大更深
 - ReLu, Dropout, 数据增强
- VGG
 - 更大更深的 AlexNet (重复的 VGG 块)

总结

- VGG 使用可重复使用的卷积块来构建深度卷积神经网络
- 不同的卷积块个数和超参数可以得到不同复杂度的变种

NiN

全连接层的问题

- 卷积层需要较少的参数
 $c_i \times c_o \times k^2$
- 但卷积层后的第一个全连接层的参数
 - LeNet $16 \times 5 \times 5 \times 120 = 48k$
 - AlexNet $256 \times 5 \times 5 \times 4096 = 26M$
 - VGG $512 \times 7 \times 7 \times 4096 = 102M$

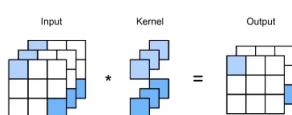
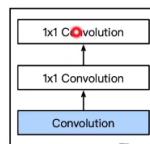
```
VGG:  
sequential1 output shape: (1, 64, 112, 112)  
sequential2 output shape: (1, 128, 56, 56)  
sequential3 output shape: (1, 256, 28, 28)  
sequential4 output shape: (1, 512, 14, 14)  
sequential5 output shape: (1, 512, 7, 7)  
dense0 output shape: (1, 4096)  
dropout0 output shape: (1, 4096)  
dense1 output shape: (1, 4096)  
dropout1 output shape: (1, 4096)  
dense2 output shape: (1, 10)
```



神经网络中卷积层后的第

一个全连接层的参数=输入所有的像素 (输入的高*宽*通道数) *输出所有的像素 (输出的高*宽*通道数)，占用大量的内存和带宽，并且容易过拟合。

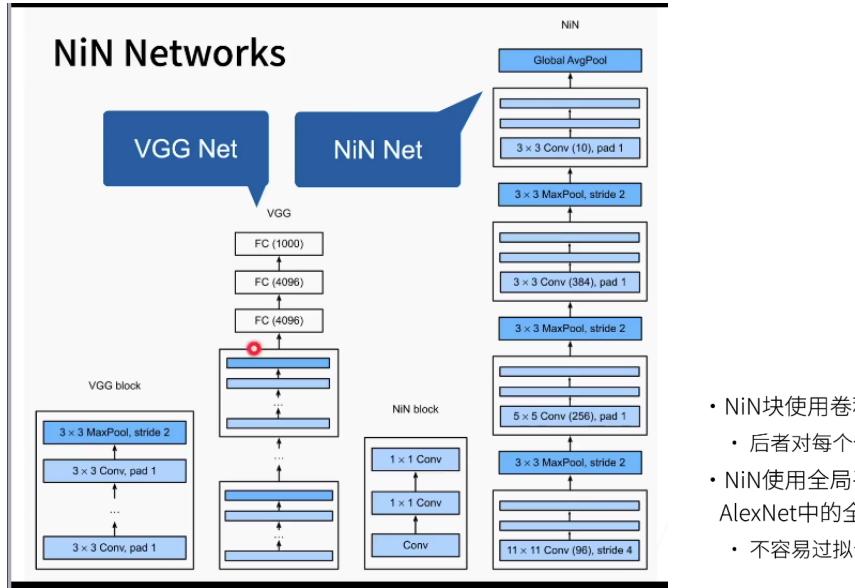
- 一个卷积层后跟两个全连接层
 - 步幅 1, 无填充, 输出形状跟卷积层输出一样
 - 起到全连接层的作用



用 1×1 的卷积核作为卷积层，从而得到全连接层的效果，我们可以认为是以每个像素来做全连接层，而不再是高宽了。

- 无全连接层
- 交替使用NiN块和步幅为2的最大池化层
 - 逐步减小高宽和增大通道数
- 最后使用全局平均池化层得到输出
 - 其输入通道数是类别数

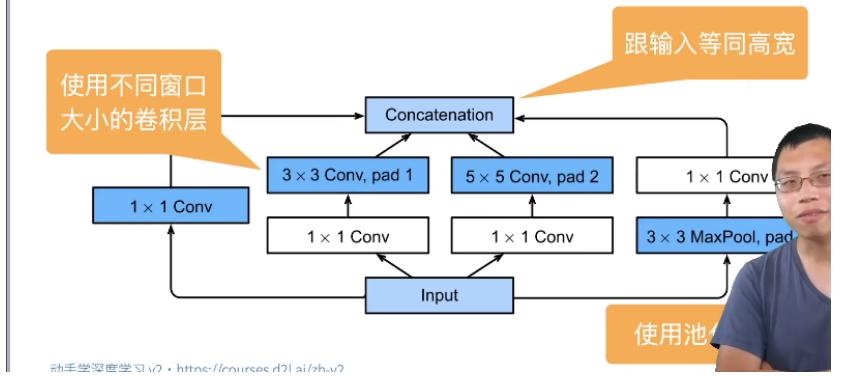
最后也不使用全连接层得到等于类别的输出。



GoogLeNet

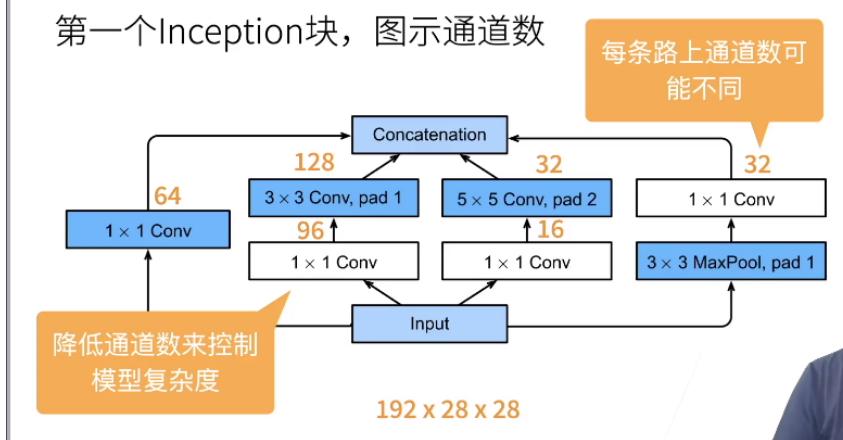
Inception块：小学生才做选择题，我全要了

4个路径从不同层面抽取信息，然后在输出通道维合并



Inception块

第一个Inception块，图示通道数



Inception块

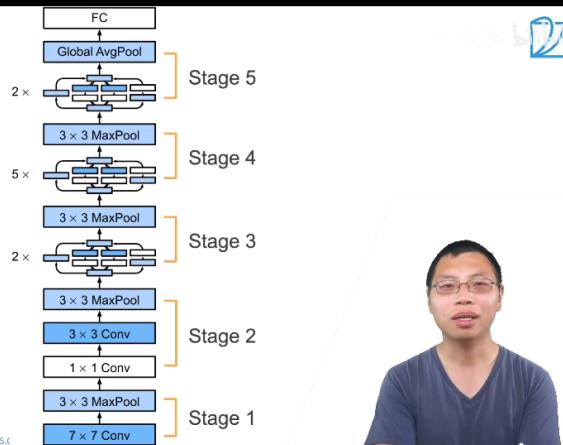
跟单3x3或5x5卷积层比，Inception块有更少的参数个数和计算复杂度

	#parameters	FLOPS
Inception	0.16 M	128 M
3x3 Conv	0.44 M	346 M
5x5 Conv	1.22 M	963 M



GoogLeNet

- 5段，9个 Inception块

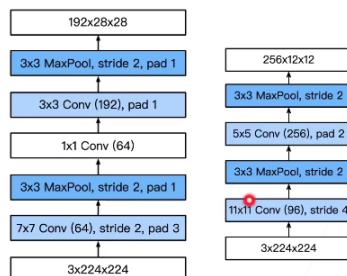


段 1 & 2

GoogLeNet

AlexNet

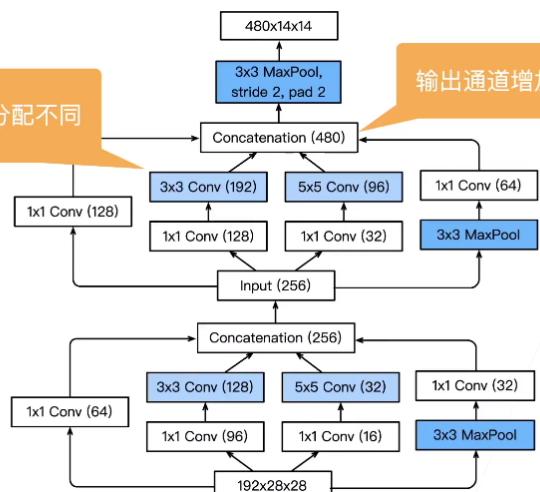
- 更小的宽口，更多的通道

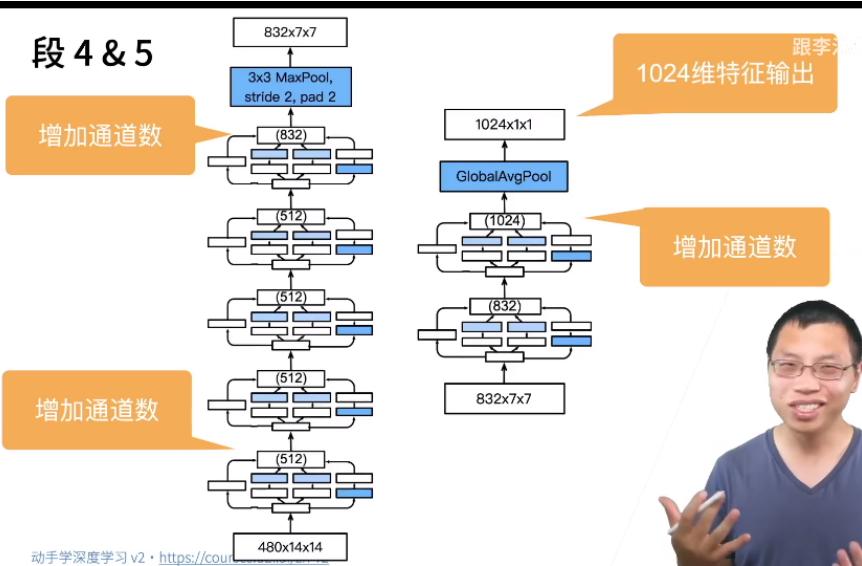


段 3

通道分配不同

输出通道增加





Inception 有各种后续变种

- Inception-BN (v2) - 使用 batch normalization (后面介绍)
- Inception-V3 - 修改了Inception块
 - 替换 5x5 为多个 3x3 卷积层
 - 替换 5x5 为 1x7 和 7x1 卷积层
 - 替换 3x3 为 1x3 和 3x1 卷积层
 - 更深
- Inception-V4 - 使用残差连接 (后面介绍)



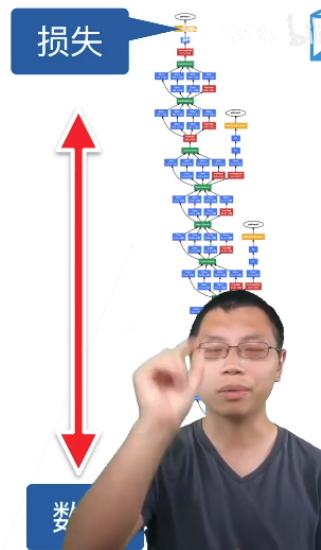
总结

- Inception块用4条有不同超参数的卷积层和池化层的路来抽取不同的信息
 - 它的一个主要优点是模型参数小，计算复杂度低
- GoogleNet使用了9个Inception块，是第一个达到上百层的网络
 - 后续有一系列改进

批量归一化

批量归一化

- 损失出现在最后，后面的层训练较快
- 数据在最底部
 - 底部的层训练较慢
 - 底部层一变化，所有都得跟着变
 - 最后的那些层需要重新学习多次
 - 导致收敛变慢
- 我们可以在学习底部层的时候避免变化顶部层吗？



动手学深度学习 v2 · <https://courses.csail.mit.edu/6.S095/>

批量归一化

- 固定小批量里面的均值和方差

$$\mu_B = \frac{1}{|B|} \sum_{i \in B} x_i \text{ and } \sigma_B^2 = \frac{1}{|B|} \sum_{i \in B} (x_i - \mu_B)^2 + \epsilon$$

然后再做额外的调整（可学习的参数）

$$x_{i+1} = \gamma \frac{x_i - \mu_B}{\sigma_B} + \beta$$

方差



我们使某个批量内输出和梯度

的满足一种分布，则我们就认为这些参数比较稳定了，使得参数不会发生剧烈的改变，使得学习细微的变动就更加容易了。

- 可学习的参数为 γ 和 β
- 作用在
 - 全连接层和卷积层输出上，激活函数前
 - 全连接层和卷积层输入上
- 对全连接层，作用在特征维
- 对于卷积层，作用在通道维

批量归一化在做什么？

- 最初论文是想用它来减少内部协变量转移
- 后续有论文指出它可能就是通过在每个小批量里加入噪音来控制模型复杂度

$$x_{i+1} = \gamma \frac{x_i - \hat{\mu}_B}{\hat{\sigma}_B} + \beta$$

随机偏移

随机缩放

- 因此没必要跟丢弃法混合使用

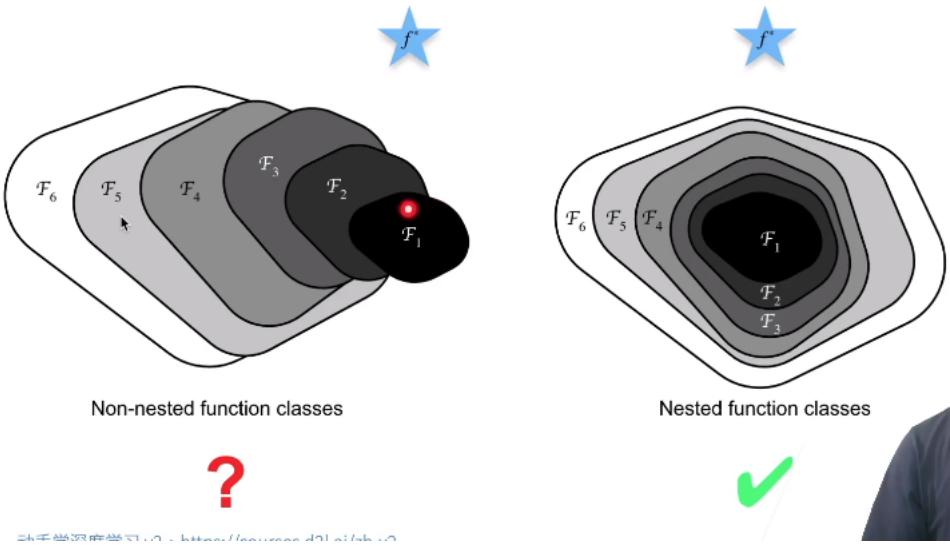


- 批量归一化固定小批量中的均值和方差，然后学习出适合的偏移和缩放
- 可以加速收敛速度，但一般不改变模型精度

问题9:不太理解，为啥加了batch norm 收敛时间变短

添加了bn后,学习率可以更大,则梯度更新的步长就变大了,到达我们想要的区域就更快了,即收敛变快了.

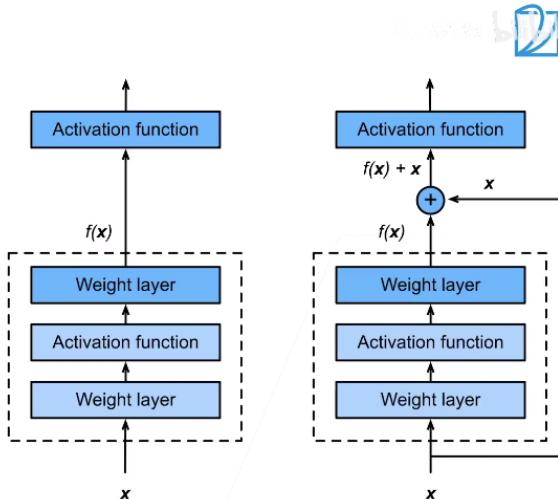
ResNet残差网络



格包含原来的小模型，则结果至少不会变的更差。

残差块

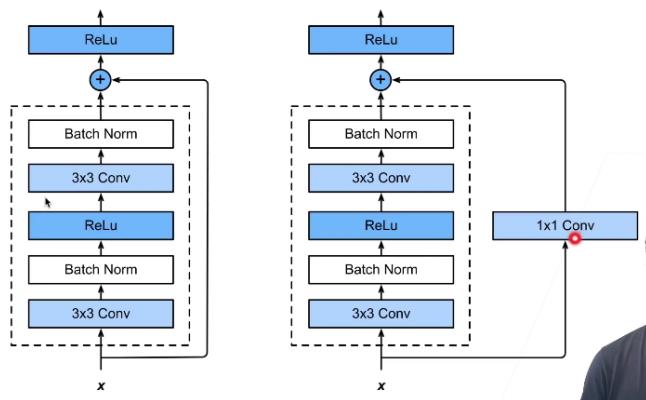
- 串联一个层改变函数类，我们希望能扩大函数类
- 残差块加入快速通道（右边）来得到 $f(x) = x + g(x)$ 的结构



可以理解为：能够

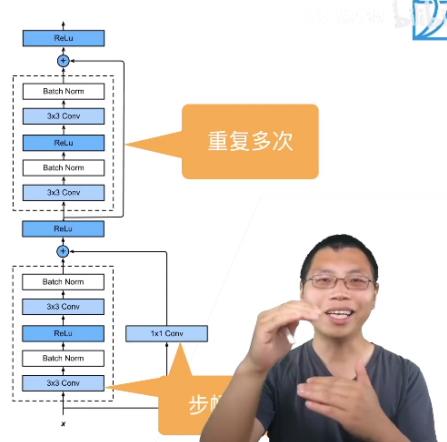
嵌入进去小的网络，先去拟合小的网络。

ResNet 块细节



ResNet 块

- 高宽减半 ResNet 块
(步幅 2)
- 后接多个高宽不变
ResNet 块

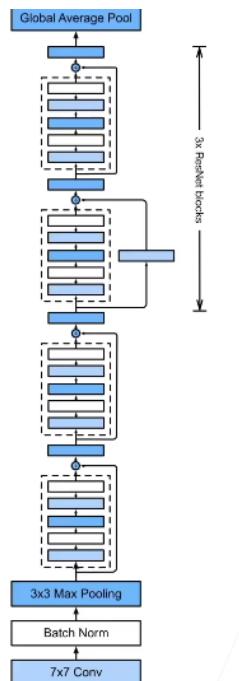


动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

这里的步幅为2

ResNet 架构

- 类似 VGG 和 GoogleNet 的总体架构
- 但替换成了 ResNet 块



动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

总结

- 残差块使得很深的网络更容易训练
 - 甚至可以训练一千层的网络
- 残差网络对随后的深层神经网络设计产生了深远影响，无论是否是卷积类网络还是全连接类网络。

问题16: $f(x) = x + g(x)$, 这样就能保证至少不会变坏吗？如果 $g(x)$ 不是变好，也不是什么都不干，而是变坏了呢？

如果 $g(x)$ 对权重的更新作用

很小，则其作用就不大，也不会产生负面影响

问题18:残差这个概念体现在什么地方? 就是因为 $f(x)=x+g(x)$, 所以 $g(x)$ 可以视为 $f(x)$ 的残差?

g(x)会是前面训练的比较简单地没有fitting太多特征的模型 , 而后面的 $f(x)$ 则是在 $g(x)$ 的基础上进一步的去fitting新的特征, 作出改进。

问题20:请问 在`__init__`里为什么定义两个bn, 这两个bn的参数一样? 为什么定义了`self.relu`在`forward`里面没有用? 麻烦老师可以解释解释
`nn.ReLU(inplace=True)` `inplace` 这个参数么? ? 谢谢

两个bn是两个不同的层 , 需要单独得到训练的参数, Relu不需要训练参数 , 只用一个就可以了

!

问题22:训练acc是不是正常训练时就是会稍微大于测试acc?这是不是意味着永远达不到100%识别?



两个bn是两个不

在训练时加入大量噪音

, 有可能会使得最后的测试精度大于训练精度 , 因为最后的测试不会加入噪音 , 很有可能精度大于训练精度

计算机视觉

GPU和CPU

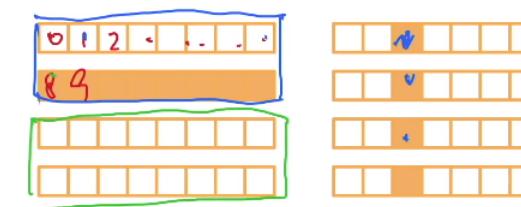
提升CPU利用率 I

- 在计算 $a + b$ 之前, 需要准备数据
 - 主内存 -> L3 -> L2 -> L1 -> 寄存器
 - L1 访问延时: 0.5 ns
 - L2 访问延时: 7 ns (14 x L1)
 - 主内存访问延时: 100ns (200 x L1)
 - 提升空间和时间的内存本地性
 - 时间: 重用数据使得保持它们在缓存里
 - 空间: 按序读写数据使得可以预读取

样例分析

深入浅出 Python

- 如果一个矩阵是按列存储，访问一行会比访问一列要快
 - CPU 一次读取 64 字节（缓存线）
 - CPU 会“聪明的”提前读取下一个（缓存线）



提升CPU利用率 II

- 高端 CPU 有几十个核
 - EC2 P3.16xlarge: 2 Intel Xeon CPUs, 32 物理核
- 并行来利用所有核
 - 超线程不一定提升性能，因为它们共享寄存器
- 左边比右边慢（python）

```
for i in range(len(a)):  
    c[i] = a[i] + b[i]           c = a + b
```

- 左边调用 n 次函数，每次调用有开销
- 右边很容易被并行（例如下面C++实现）

```
#pragma omp for  
for (i=0; i<a.size(); i++) {  
    c[i] = a[i] + b[i]  
}
```

CPU vs GPU

深入浅出 GPU



一般 / 高端

# 核	6 / 64	2K / 4K
TFLOPS	0.2 / 1	10 / 100
内存大小	32 GB / 1 TB	16 GB / 32 GB
内存带宽	30 GB/s / 100 GB/s	400 GB/s / 1 TB/s
控制流	强	弱



提升GPU利用率

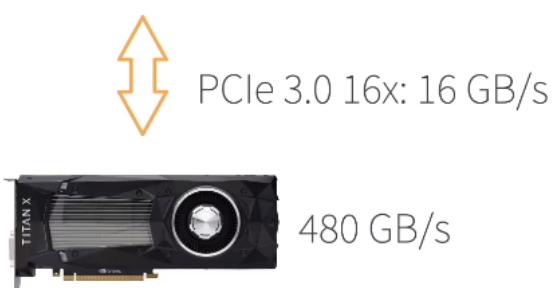
- 并行
 - 使用数千个线程
- 内存本地性
 - 缓存更小，架构更加简单
- 少用控制语句
 - 支持有限
 - 同步开销很大

CPU/GPU 带宽

深入浅出 GPU



- 不要频繁在CPU和GPU之前传数据：带宽限制，同步开销



480 GB/s

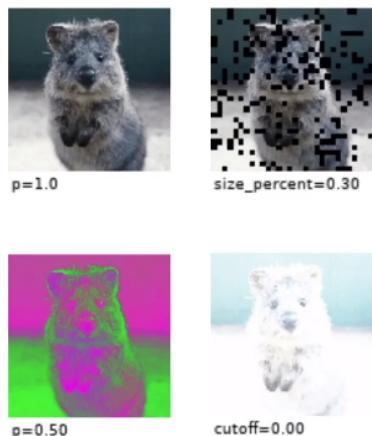


总结

- CPU：可以处理通用计算。性能优化考虑数据读写效率和多线程
- GPU：使用更多的小核和更好的内存带宽，适合能大规模并行的计算任务

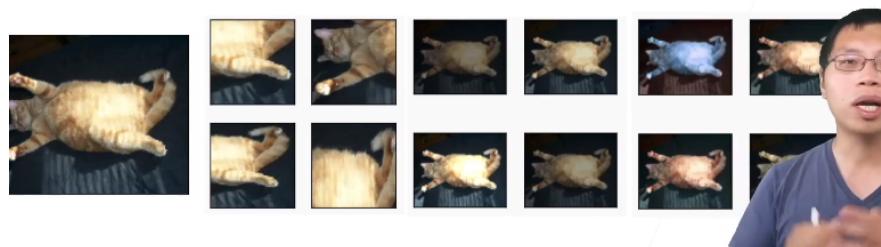
数据增广

数据增广



数据增强

- 增加一个已有数据集，使得有更多的多样性
 - 在语言里面加入各种不同的背景噪音
 - 改变图片的颜色和形状





翻转

- 左右翻转



- 上下翻转



- 不总是可行



切割

- 从图片中切割一块，然后变形到固定形状
 - 随机高宽比 (e.g. [3/4, 4/3])
 - 随机大小 (e.g. [8%, 100%])
 - 随机位置



颜色

- 改变色调，饱和度，亮度(e.g. [0.5, 1.5])



几十种其他的办法



总结

- 数据增广通过变形数据来获取多样性从而使得模型泛化性能更好
- 常见图片增广包括翻转、切割、变色

问题7：老师，做了图片增广后，训练数据的分布可能会和测试数据的分布不一样，那会对模型最终精度有影响吗？

训练数据分布没有改变

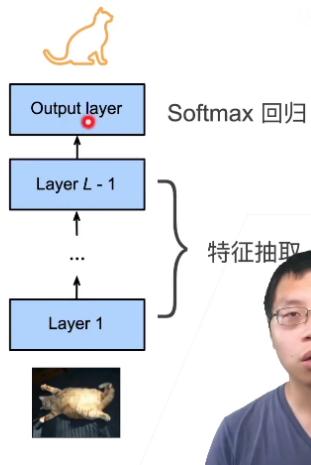
问题15：如何理解多样性增加，但是分布不变？

均值没有发生变化，但是方差变大了

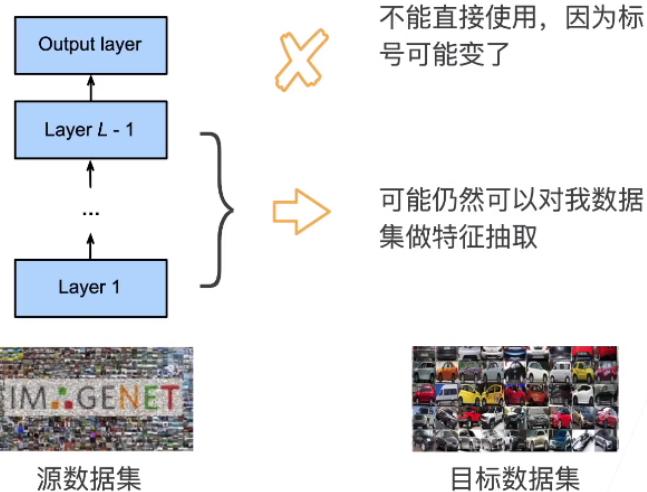
微调

网络架构

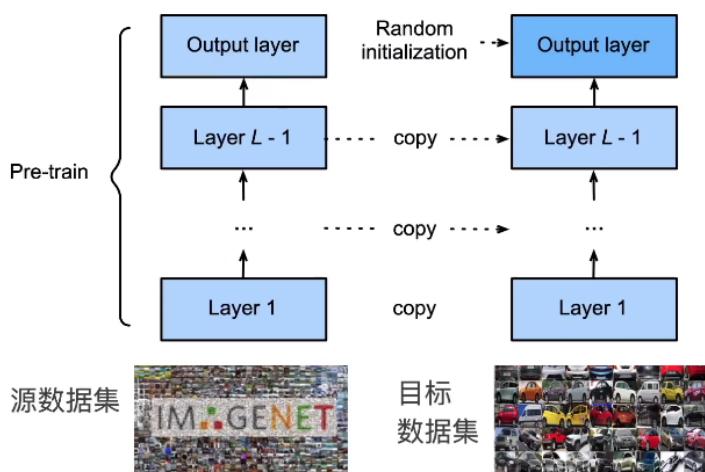
- 一个神经网络一般可以分成两块
 - 特征抽取将原始像素变成容易线性分割的特征
 - 线性分类器来做分类



微调



微调中的权重初始化

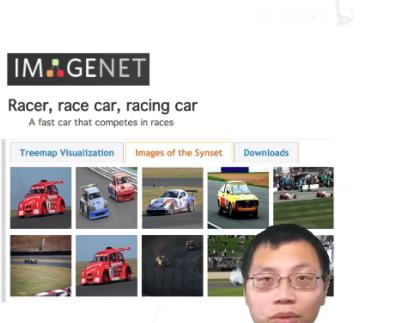


训练

- 是一个目标数据集上的正常训练任务，但使用更强的正则化
 - 使用更小的学习率
 - 使用更少的数据迭代
- 源数据集远复杂于目标数据，通常微调效果更好

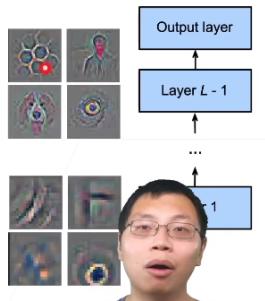
重用分类器权重

- 源数据集可能也有目标数据中的部分标号
- 可以使用预训练好模型分类器中对应标号对应的向量来做初始化



固定一些层

- 神经网络通常学习有层次的特征表示
 - 低层次的特征更加通用
 - 高层次的特征则更跟数据集相关
- 可以固定底部一些层的参数，不参与更新
 - 更强的正则



总结

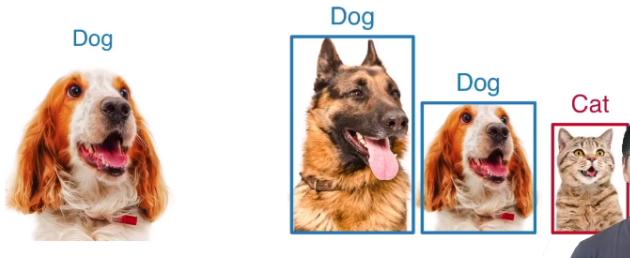
- 微调通过使用在大数据上得到的预训练好的模型来初始化模型权重来完成提升精度
- 预训练模型质量很重要
- 微调通常速度更快、精度更高

问题29：如果源数据集和目标数据集差异很大，微调的效果会下降吗，例如
Imagenet上的模型用到医疗影像分类

会，最好类似

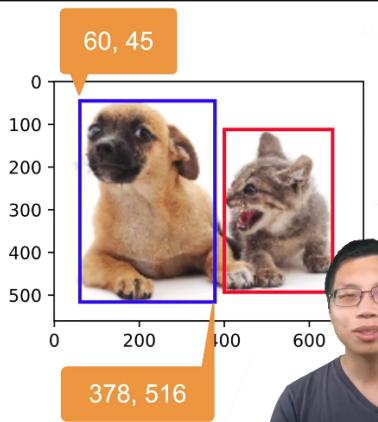
物体检测

图片分类



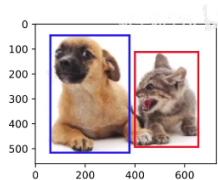
边缘框

- 一个边缘框可以通过4个数字定义,
- (左上x, 左上y, 右下x, 右下y)
- (左上x, 左上y, 宽, 高)



目标检测数据集

- 每行表示一个物体
 - 图片文件名, 物体类别, 边缘框
- COCO (cocodataset.org)
 - 80 物体, 330K 图片, 1.5M 物体



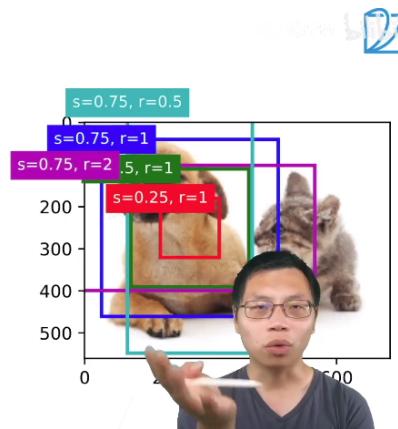
总结

- 物体检测识别图片里的多个物体的类别和位置
- 位置通常用边缘框表示

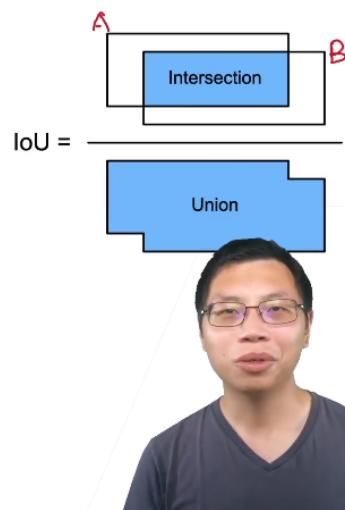
锚框

锚框

- 一类目标检测算法是基于锚框
- 提出多个被称为锚框的区域（边缘框）
- 预测每个锚框里是否含有关注的物体
- 如果是，预测从这个锚框到真实边缘框的偏移



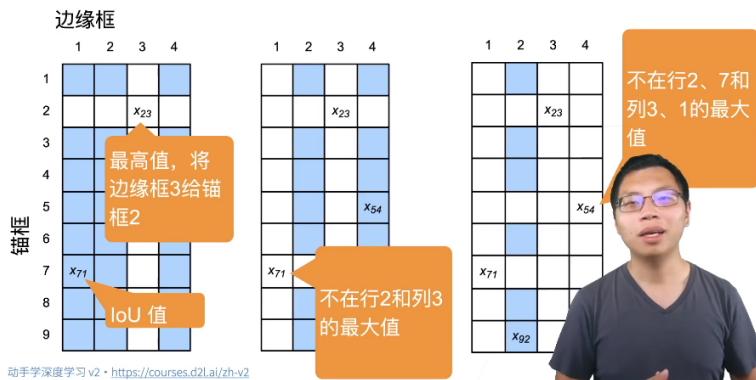
- IoU 用来计算两个框之间的相似度
 - 0 表示无重叠，1 表示重合
- 这是 Jaccard 指数的一个特殊情况
 - 给定两个集合 A 和 B



赋予锚框标号

- 每个锚框是一个训练样本
- 将每个锚框，要么标注成背景，要么关联上一个真实边缘框
 - 我们可能会生成大量的锚框
 - 这个导致大量的负类样本

赋予锚框标号

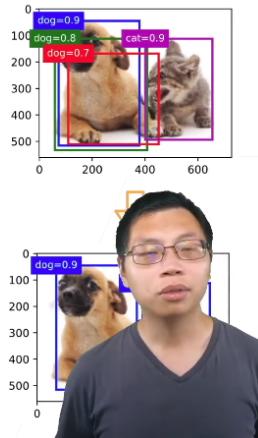


列为真实的边缘框，行为预测

的锚框，则每个锚框都对边缘框做IOU，求得其中的最大值，比如第一图的最大值为锚框2对分类3的IOU，则我们就认为锚框2是对分类3的有效预测，然后将分类3和锚框2从矩阵中删去，再迭代上面的过程，在分类都删除完毕后迭代结束。

使用非极大值抑制（NMS）输出

- 每个锚框预测一个边缘框
- NMS可以合并相似的预测
 - 选中是非背景类的最大预测值
 - 去掉所有其它和它IoU值大于 θ 的预测
- 重复上述过程直到所有预测要么被选中，要么被去掉



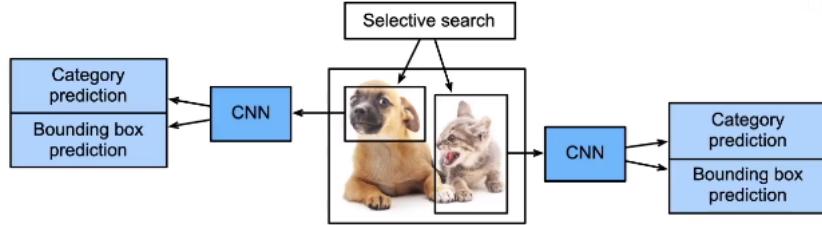
我们得到很多锚框，首先选中最大预测值的锚框，然后用其他锚框和这个最大值锚框做NMS，大于阈值的锚框就被删掉，使得所有锚框要么被删掉，要么被保留，使得相似度大于某些阈值的锚框就被删除了。

总结

- 一类目标检测算法基于锚框来预测
- 首先生成大量锚框，并赋予标号，每个锚框作为一个样本进行训练
- 在预测时，使用NMS来去掉冗余的预测

区域卷积神经网络R-CNNs

R-CNN

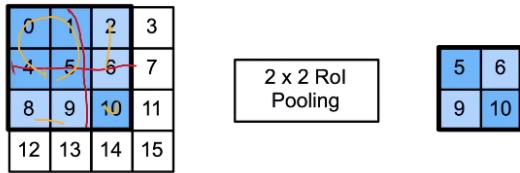


- 使用启发式搜索算法来选择锚框
- 使用预训练模型来对每个锚框抽取特征
- 训练一个SVM来对类别分类
- 训练一个线性回归模型来预测边缘框偏移



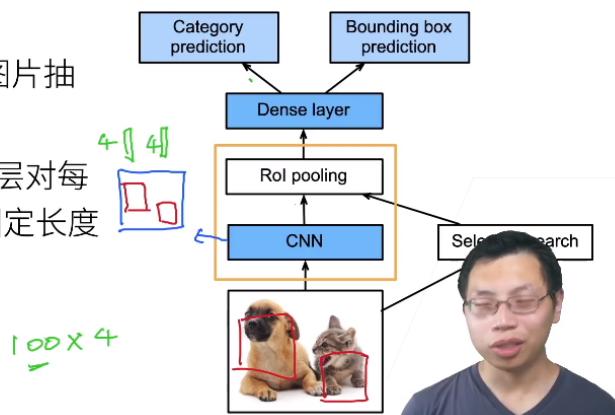
兴趣区域（RoI）池化层

- 给定一个锚框，均匀分割成 $n \times m$ 块，输出每块里的最大值
- 不管锚框多大，总是输出 nm 个值



Fast RCNN

- 使用CNN对图片抽取特征
- 使用RoI池化层对每个锚框生成固定长度特征

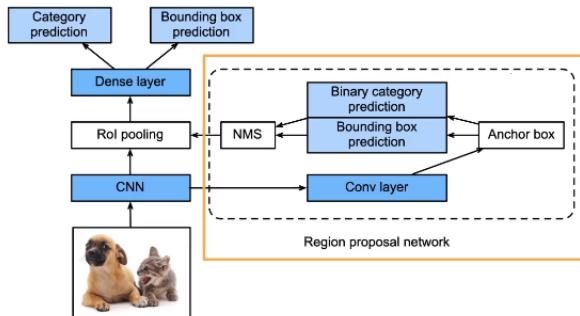


fast RCNN是使用CNN对整个图片抽取特征而不是对锚框了，这样我们再将selective search映射到CNN抽取的特征中，这样就大大加快了速度。

Faster R-CNN



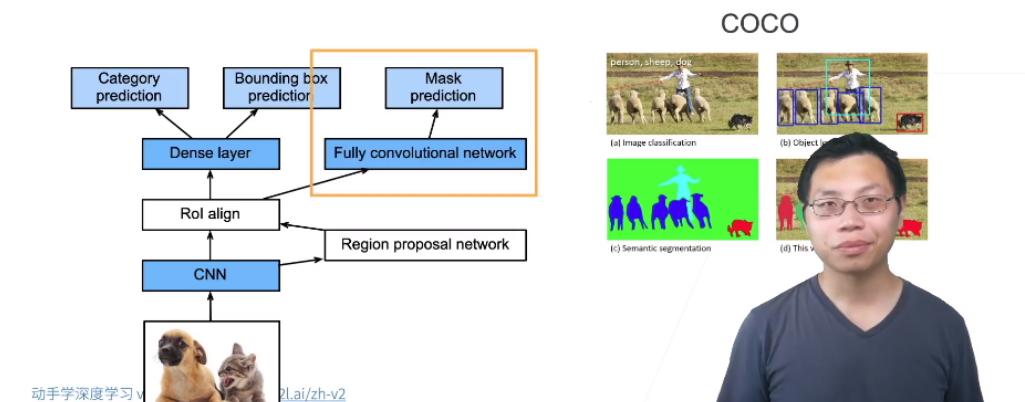
- 使用一个区域提议网络来替代启发式搜索来获得更好的锚框



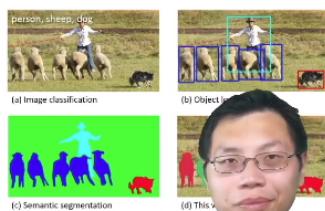
Mask R-CNN



- 如果有像素级别的~~标注~~，使用FCN来利用这些信息



COCO



在这里，使用Mask R-CNN会不影响像素级别的标号，如果我们使用前面的方法，则对区域切分的时候使用像素取整，则多次的取整回导致像素的偏移，导致边界无法确定编号。

而在使用Mask R-CNN时，切分像素不为整个像素的时候，会将一个像素切分，对每个分块给予权重。

总结

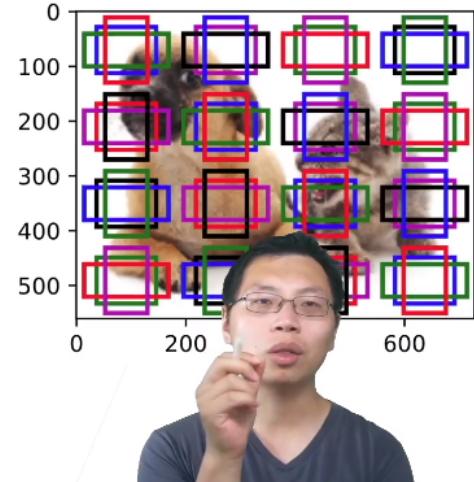
- R-CNN 是最早、也是最有名的一类基于锚框和CNN的目标检测算法
- Fast/Faster R-CNN持续提升性能
- Faster R-CNN 和 Mask R-CNN是在最求高精度场景下的常用算法

Single Shot Detection (SSD) 单发多框检测

生成锚框

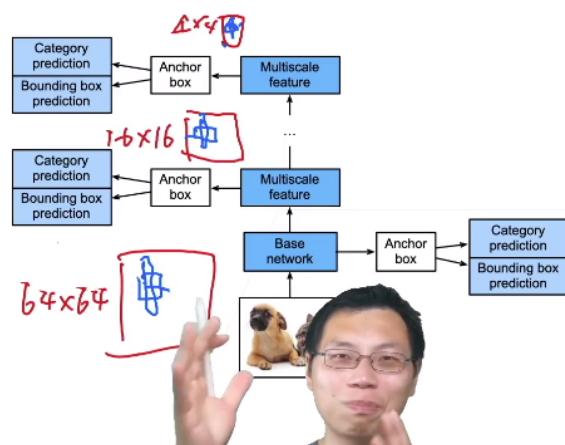
- 对每个像素，生成多个以它为中心的锚框
- 给定 n 个大小 s_1, \dots, s_n 和 m 个高宽比，那么生成 $n + m - 1$ 个锚框，其大小和高宽比分别为：

$$(s_1, r_1), (s_2, r_1), \dots, (s_n, r_1), (s_1, r_2), \dots, (s_1, r_m)$$



SSD 模型

- 一个基础网络来抽取特征，然后多个卷积层块来减半高宽
- 在每段都生成锚框
 - 底部段来拟合小物体，顶部段来拟合大物体
- 对每个锚框预测类别和边缘框



总结

- SSD 通过单神经网络来检测模型
- 以每个像素为中心的产生多个锚框
- 在多个段的输出上进行多尺度的检测

YOLO

YOLO（你只看一次）



- SSD中锚框大量重叠，因此浪费了很多计算
- YOLO 将图片均匀分成 $S \times S$ 个锚框
- 每个锚框预测 B 个边缘框
- 后续版本（V2,V3,V4...）有持续改进



问题2：请问一下，测试数据增强做平均，是结果做平均还是概率做平均？还是看实际情况选择？|

T

概率做平均